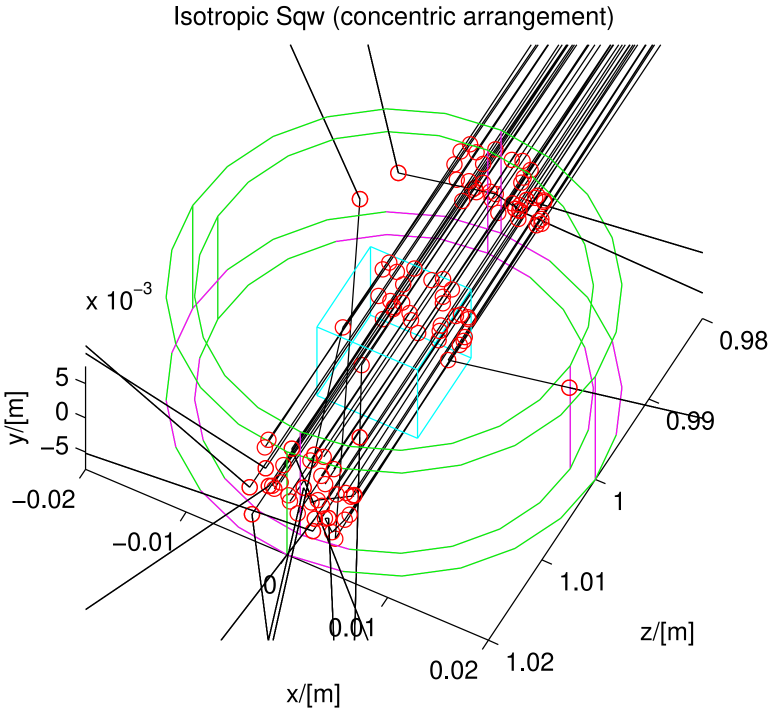
Risø–R–1538(EN)

Component Manual for the Neutron Ray-Tracing Package McStas, Version 1.9

Kim Lefmann, Peter Kjær Willendrup, Kristian Nielsen, Emmanuel Farhi, and Klaus Lieutenant



Risø National Laboratory, Roskilde, Denmark
November 2005

Abstract

The software package McStas is a tool for carrying out Monte Carlo ray-tracing simulations of neutron scattering instruments with high complexity and precision. The simulations can compute most aspects of the performance of instruments and samples and can thus be used to optimize the use of existing equipment, design new instrumentation, and carry out full virtual experiments. McStas is based on a unique design where an automatic compilation process translates high-level textual instrument descriptions into efficient ANSI-C code. This design makes it simple to set up typical simulations and also gives essentially unlimited freedom to handle more unusual cases.

This report constitutes the component manual for McStas, and, together with the manual for the McStas system, it contains full documentation of all aspects of the program. It covers a description of all official components of the McStas package with some theoretical background. Selected test instruments and representative McStas simulations performed with these instruments are described in the User Manual.

This report documents the components for McStas version 1.9, released November 15, 2005.

The authors are:

Kim Lefmann
Materials Research Department, Risø National Laboratory, Roskilde, Denmark
email: `kim.lefmann@risoe.dk`

Peter Kjær Willendrup
Materials Research Department, Risø National Laboratory, Roskilde, Denmark
email: `peter.willendrup@risoe.dk`

Kristian Nielsen
email: `kristian.nielsen@mail.tele.dk`

Emmanuel Farhi
Institut Laue-Langevin, Grenoble, France
email: `farhi@ill.fr`

Klaus Lieutenant
Institut Laue-Langevin, Grenoble, France
email: `lieutenant@ill.fr`

ISBN 87–550–3482–9
ISSN 0106–2840
Pitney Bowes Management Services Denmark A/S · Risø National Laboratory · 2005

Contents

Preface and acknowledgements

This document contains information on the neutron scattering components which are the building blocks for defining instruments in the Monte Carlo neutron ray-tracing program McStas version 1.9. The initial release in October 1998 of version 1.0 was presented in Ref. [1]. The reader of this document is not supposed to have specific knowledge of neutron scattering, but some basic understanding of physics is helpful in understanding the theoretical background for the component functionality. For details about setting up and running simulations, we refer to the McStas system manual [2]. We assume familiarity with the use of the C programming language.

It is a pleasure to thank Dir. Kurt N. Clausen, PSI, for his continuous support to McStas and for having initiated the project. Continuous support to McStas has also come from Prof. Robert McGreevy, ISIS. Apart from the authors of this manual, also Per-Olof Åstrand, NTNU Trondheim, has contributed to the development of the McStas system. We have further benefited from discussions with many other people in the neutron scattering community, too numerous to mention here.

The users who contributed components to this manual are acknowledged as authors of the individual components. We encourage other users to contribute components with manual entries for inclusion in future versions of McStas.

In case of errors, questions, or suggestions, do not hesitate to contact the authors at `mcstas@risoe.dk` or consult the McStas home page [3]. A special bug/request reporting service is available [4].

Important developments on the component side in McStas version 1.9 as compared to version 1.4 (the last version of the component manual; then a section of the system manual) include

- Validation of most components against analytical formula, and benchmarking in simple cases
- Newly added, realistic source components
  - `ISIS_moderator` ISIS source model based on MCNPX (D. Champion and S. Ansell, ISIS)
  - `Virtual_tripoli4_input/output` Trioli4 (similar to MCNP) files reading/writing (G. Campioni, LLB)
  - `SNS_source` SNS source model based on MCNPX (G. Granroth, SNS)
  - `Source_gen` ILL sources Maxwellian parameters (E. Farhi/N. Kernavanois/H. Bordallo, ILL)

- ESS_moderator_short Calculated source model for the short pulse target station of the ESS project (K. Lefmann, Risø)
- ESS_moderator_long Calculated source model for the long pulse target station of the ESS project (K. Lefmann, Risø)

- Newly added, optical components
  - Radial_collimator Radial collimator with both approximated and exact options (E. Farhi, ILL)
  - FermiChopper and Vitess_ChopperFermi Two Fermi Chopper components (M. Poehlmann, G. Zsigmond, ILL and PSI)
  - Guide_tapering A rectangular tapered guide (U. Filges, PSI)
  - Guide_curved Non-focusing curved neutron guide (R. Stewart, ILL)

- A suite of sample components
  - Phonon_simple An isotropic acoustic phonon (K. Lefmann, Risø)
  - PowderN. N lines powder diffraction (P.K. Willendrup, Risø)
  - Sans_spheres hard spheres in thin solution, mono disperse (L. Arleth, the Royal Veterinary and Agricultural University (DK), K. Lefmann, Risø)
  - Isotropic_Sqw isotropic inelastic sample (powder, liquid, glass) elastic/inelastic scattering from $S(q, \omega)$ data (E. Farhi, V. Hugouvieux, ILL)
  - SANS_* A collection of samples for SANS (H. Frielinghaus, FZ-Jülich)

We would like to kindly thank all McStas component contributors. This is the way we improve the software alltogether.

The McStas project has been supported by the European Union, initially through the XENNI program and the RTD “Cool Neutrons” program in FP4, In FP5, McStas was supported strongly through the “SCANS” program. Currently, in FP6, McStas is supported through the Joint Research Activity “MCNSI” under the Integrated Infrastructure Initiative “NMI3”, see the WWW home pages [5, 6].

If you **appreciate** this software, please subscribe to the neutron-mc@risoe.dk email list, send us a smiley message, and contribute to the package. We also encourage you to refer to this software when publishing results, with the following citations:

- K. Lefmann and K. Nielsen, Neutron News **10/3**, 20, (1999).
- P. Willendrup, E. Farhi and K. Lefmann, Physica B, **350**, 735 (2004).

Chapter 1

About the component library

This McStas Component Manual consists of the following major parts:

- An introduction to the use of Monte Carlo methods in McStas.
- A thorough description of system components, with one chapter per major category: Sources, optics (two chapters), monochromators, samples, monitors, and other components.
- The McStas library functions and definitions that aid in the writing of simulations and components in Appendix A.
- A detailed explanation of the use of random numbers in Appendix B.
- An explanation of the McStas terminology in Appendix C.

Additionally, you may refer to the list of example instruments from the library in the McStas User Manual.

1.1 Authorship

The component library is maintained by the McStas system group. A number of basic components "belongs" the McStas system, and are supported and tested by the McStas team.

Other components are contributed by specific authors, who are listed in the code for each component they contribute as well as in this manual. McStas users are encouraged to send their contributions to us for inclusion in future releases.

Some contributed components have later been taken over for further development by the McStas system group, with permission from the original authors. The original authors will still appear both in the component code and in the McStas manual.

1.2 Symbols for neutron scattering and simulation

In the description of the theory behind the component functionality we will use the usual symbols $\mathbf{r}$ for the position (x, y, z) of the particle (unit m), and $\mathbf{v}$ for the particle velocity

(v_x, v_y, v_z) (unit m/s). Another essential quantity is the neutron wave vector $\mathbf{k} = m_\mathrm{n}\mathbf{v}/\hbar$, where m_n is the neutron mass. $\mathbf{k}$ is usually given in Å^{-1}, while neutron energies are given in meV. The neutron wavelength is the reciprocal wave vector, $\lambda = 2\pi/k$. In general, vectors are denoted by boldface symbols.

Subscripts "i" and "f" denotes "initial" and "final", respectively, and are used in connection with the neutron state before and after an interaction with the component in question.

The spin of the neutron is given a special treatment. Despite the fact that each physical neutron has a well defined spin value, the McStas spin vector $\mathbf{s}$ can have any length between zero (unpolarized beam) and unity (totally polarized beam). Further, all three cartesian components of the spin vector are present simultaneously, although this is physically not permitted by quantum mechanics. Presently, no official McStas component utilize the neutron spin.

1.3 Component coordinate system

All mentioning of component geometry refer to the local coordinate system of the individual component. The axis convention is so that the z axis is along the neutron propagation axis, the y axis is vertical up, and the x axis points left when looking along the z-axis, completing a right-handed coordinate system. Most components 'position' (as specified in the instrument description with the `AT` keyword) corresponds to their input side at the nominal beam position. However, a few components are radial and thus positioned in their centre.

Components are usually not designed to overlap. This may lead to loss of neutron rays. Warnings will be issued during simulation if sections of the instrument are not reached by any neutron rays, or if neutrons are removed. This is usually the sign of either overlapping components or a very low intensity.

1.4 About data files

Some components require external data files, e.g. lattice crystallographic definitions for Laue and powder pattern diffraction, $S(q, \omega)$ tables for inelastic scattering, neutron events files for virtual sources, transmission and reflectivity files, etc.

Such files distributed with McStas are located in the `data` sub-directory of the `MCSTAS` library. Components that make use of the McStas file system, including the `read-table` library (see section A.2) may access all McStas data files without making local copies. Of course, you are welcome to define your own data files, and eventually contribute to McStas if you find them useful.

File extensions are not compulsory but help in identifying relevant files per application.

McStas itself generates both simulation and monitor data files, which structure is explained in the User Manual (see end of chapter 'Running McStas ').

MCSTAS/data	Description
*.lau	Laue pattern file, as issued from Crystallographica. For use with Single_crystal, PowderN, and Isotropic_Sqw. Data: [h k l Mult. d-space 2Theta F-squared]
*.laz	Powder pattern file, as obtained from Lazy/ICSD. For use with PowderN, Isotropic_Sqw and possibly Single_crystal.
*.trm	transmission file, typically for monochromator crystals and filters. Data: [k (Angs-1) , Transmission (0-1)]
*.rfl	reflectivity file, typically for mirrors and monochromator crystals. Data: [k (Angs-1) , Reflectivity (0-1)]
*.sqw	$S(q, \omega)$ files for Isotropic_Sqw component. Data: [q] [ω] [$S(q, \omega)$]

Table 1.1: Data files of the McStas library.

1.5 Component source code

Source code for all components may be found in the `MCSTAS` library subdirectory of the McStas installation; the default is `/usr/local/lib/mcstas/` on Unix-like systems and `C:\mcstas\lib` on Windows systems, but it may be changed using the `MCSTAS` environment variable.

In case users only require to add new features, preserving the existing features of a component, it is recommanded to make use of the `EXTEND` keyword in the instrument description itself, as documented in the User Manual. For larger modification of a component, it is advised to make a copy of the component file into the working directory. A component file in the local directory will in McStas take precedence over a library component of the same name.

1.6 Documentation

As a complement to this Component Manual, we encourage users to use the `mcdoc` front-end which enables to display both the catalog of the McStas library, e.g using:

`mcdoc`

as well as the documentation of specific components, e.g with:

`mcdoc --text` *name*
`mcdoc` *file.comp*

The first line will search for all components matching the *name*, and display their help section as text. For instance, `mcdoc .laz` will list all available Lazy data files, whereas `mcdoc --text Monitor` will list most Monitors. The second example will display the help corresponding to the *file.comp* component, using your BROWSER setting, or as text if unset. The `--help` option will display the command help, as usual.

An overview of the component library is also given at the McStas home page [3] and in the User Manual [2].

1.7 Component validation

Some components were checked for release 1.9: the Fermi choppers, the velocity selectors, 2 of the guide components and Source_gen. The results are sumarized in a talk available online (`http://www.ill.fr/tas/mcstas/doc/ValMcStas.pdf`).

Velocity selector and Fermi chopper were treated as black boxes and the resulting line shapes cross-checked against analytical functions for some cases. The component 'Selector' showed no dependence on the distance between guide and selector axe. This is corrected at the moment. Apart from that the component yielded correct results. That was different with the Fermi chopper components. The component 'Chopper_Fermi', which has been part of the McStas distribution for a long time, gave wrong results and was removed from the package. The new 'Vitess_ChopperFermi' (transferred from the VITESS package) showed mainly correct behaviour. Little bugs were corrected after the first tests. At the moment, there is only the problem left that it underestimates the influence of a shadowing cylinder. With the contributed 'FermiChopper' component, there were also minor problems, which are all corrected in the meantime.

For the guides, several trajectories through different kinds of guides (straight, convergent, divergent) were calculated analytically and positions, directions and losses of reflections compared to the values calculated in the components. This was done for 'Guide' and 'Guide_gravity'; in the latter case calculations were performed with and without gravity. Additionally a cross-check against the VITESS guide module was performed. Waviness, chamfers and channels were not checked. After correction of a bug in 'Guide_gravity', both components worked perfectly (within the conditions tested.)

'Source_gen' was cross-checked against the VITESS source module for the case of 3 Maxwellians describing the moderator characteristic and typical sizes the guide and its distance to the moderator. It showed the same line shape as a functions of wavelength and divergence and the same absolute values.

1.8 Disclaimer, bugs

We would like to emphasize that the usage of both the McStas software, as well as its components are the responsability of the users. Indeed, obtaining accurate and reliable results requires a substantial work when writing instrument descriptions. This also means that users should read carefully both the documentation from the manuals [2] and from the component itself (using `mcdoc` *comp*) before reporting errors. Most anomalous results often originate from a wrong usage of some part of the package.

Anyway, if you find that either the documentation is not clear, or the behavior of the simulation is undoubtedly anomalous, you should report this to us at `mcstas@risoe.dk` and refer to our special bug/request reporting service [4].

Chapter 2

Monte Carlo Techniques and simulation strategy

This chapter explains the simulation strategy and the Monte Carlo techniques used in McStas. We first explain the concept of the neutron weight factor, and discuss the statistical errors in dealing with sums of neutron weights. Secondly, we give an expression for how the weight factor transforms under a Monte Carlo choice and specialize this to the concept of direction focusing. Finally, we present a way of generating random numbers with arbitrary distributions.

2.1 Neutron spectrometer simulations

Neutron scattering instruments are built as a series of neutron optics elements. Each of these elements modifies the beam characteristics (e.g. divergence, wavelength spread, spatial and time distributions) in a way which may be modeled through analytical methods, for simplified neutron beam configurations. This stands for individual elements such as guides [7, 8], choppers [9, 10], Fermi choppers [11, 12], velocity selectors [13], monochromators [14–17], and detectors [18–20]. In the case of selected neutron instrument parts, one may use efficiently the so-called acceptance diagram theory [8, 21, 22] within which the neutron beam distributions are considered to be homogeneous or gaussian. However, the concatenation of a high number of neutron optical elements, which indeed constitute real instruments, brings additional complexity by introducing strong correlations between neutron beam parameters: divergence and position - which is the basis of the acceptance diagram method - but also wavelength and time. The usual analytical methods (phase-space theory...) then reach their limit of validity in the description of the resulting fine effects.

In principle, computing individual neutron event propagation at each instrument part, using analytical and numerical models, is not such a hard task. The use of probablilities is common to describe microscopic physical processes. Integrating all these events over the propagation path will result in an estimation of measurable quantities characterizing the neutron instrument. Moreover, using variance reduction (*e.g.* importance sampling), whenever possible, will both speed-up the computation and achieve a better accuracy. What we just sketched is nothing else than the basis of the Monte-Carlo (MC) method

[23], applied to neutron ray-tracing instrumentation.

2.1.1 Monte Carlo ray tracing simulations

Mathematically, the Monte-Carlo method is an application of the law of large numbers [23, 24]. Let $f(u)$ be a finite continuous integrable function of parameter u for which an integral estimate is desirable. The discrete statistical mean value of f (computed as a series) in the uniformly sampled interval $a < u < b$ converges to the mathematical mean value of f over the same interval.

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1, a\leq u_i \leq b}^{n} f(u_i) = \frac{1}{b-a} \int_a^b f(u)du \tag{2.1}$$

In the case were the u_i values are regularly sampled, we come to the well known midpoint integration rule. In the case were the u_i values are randomly (but regularly) sampled, this is the Monte-Carlo integration technique. As random generators are not perfect, we rather talk about *quasi*-Monte-Carlo technique. We encourage the reader to refer to James [23] for a detailed review on the Monte-Carlo method.

Although early implementations of the method for neutron instruments used *home-made* computer programs (see e.g. papers by J.R.D. Copley, D.F.R. Mildner, J.M. Carpenter, J. Cook), more general packages have been designed, providing models for most parts of the simulations. These present existing packages are: NISP [25], ResTrax [26], McStas [1, 3, 27], Vitess [28, 29], and IDEAS [30]. Their usage usually covers all types of neutron spectrometers, most of the time through a user-friendly graphical interface, without requiring programming skills.

The neutron ray-tracing Monte-Carlo method has been used widely for *e.g.* guide studies [21, 31, 32], instrument optimization and design [33, 34]. Most of the time, the conclusions and general behaviour of such studies may be obtained using the classical analytical approaches, but accurate estimates for the flux, the resolutions, and generally the optimum parameter set, benefit advantageously from MC methods.

Recently, the concept of virtual experiments, *i.e.* full simulations of a complete neutron experiment, has been suggested as the main goal for neutron ray-tracing simulations. The goal is that simulations should be of benefit to not only instrument builders, but also to users for training, experiment planning, diagnostics, and data analysis.

2.2 The neutron weight

A totally realistic semi-classical simulation will require that each neutron is at any time either present or lost. In many instruments, only a very small fraction of the initial neutrons will ever be detected, and simulations of this kind will therefore waste much time in dealing with neutrons that never hit the detector.

An important way of speeding up calculations is to introduce a neutron "weight factor" for each simulated neutron ray and to adjust this weight according to the path of the ray. If *e.g.* the reflectivity of a certain optical component is 10%, and only reflected neutrons ray are considered in the simulations, the neutron weight will be multiplied by 0.10 when passing this component, but every neutron is allowed to reflect in the component. In contrast, the totally realistic simulation of the component would require in average ten incoming neutrons for each reflected one.

Let the initial neutron weight be p_0 and let us denote the weight multiplication factor in the j'th component by π_j. The resulting weight factor for the neutron ray after passage of the whole instrument becomes the product of all contributions

$$p = p_n = p_0 \prod_{j=1}^{n} \pi_j. \tag{2.2}$$

For convenience, the value of p is updated (within each component) during the simulation.

Simulation by weight adjustment is performed whenever possible. This includes

- Transmission through filters.
- Transmission through Soller blade collimator (in the approximation which does not take each blade into account).
- Reflection from monochromator (and analyser) crystals with finite reflectivity and mosaicity.
- Passage of a continuous beam through a chopper.
- Scattering from samples.

2.2.1 Statistical errors of non-integer counts

In a typical simulation, the result will consist of a count of neutrons histories ("rays") with different weights. The sum of these weights is an estimate of the mean number of neutrons hitting the monitor (or detector) per second in a "real" experiment. One may write the counting result as

$$I = \sum_i p_i = N\overline{p}, \tag{2.3}$$

where N is the number of neutrons in the detector and the vertical bar denote averaging. By performing the weight transformations, the (statistical) mean value of I is unchanged. However, N will in general be enhanced, and this will improve the accuracy of the simulation.

To give an estimate of the statistical error, we proceed as follows: Let us first for simplicity assume that all the counted neutron weights are almost equal, $p_i \approx \overline{p}$, and that we observe a large number of neutrons, $N \geq 10$. Then N almost follows a normal distribution with the uncertainty $\sigma(N) = \sqrt{N}$ [1]. Hence, the statistical uncertainty of the observed intensity becomes

$$\sigma(I) = \sqrt{N}\overline{p} = I/\sqrt{N}, \tag{2.4}$$

as is used in real neutron experiments (where $\overline{p} \equiv 1$). For a better approximation we return to Eq. (2.3). Allowing variations in both N and $\overline{p}$, we calculate the variance of the resulting intensity, assuming that the two variables are independent:

$$\sigma^2(I) = \sigma^2(N)\overline{p}^2 + N^2\sigma^2(\overline{p}). \tag{2.5}$$

[1] This is not correct in a situation where the detector counts a large fraction of the neutrons in the simulation, but we will neglect that for now.

Assuming that N follows a normal distribution, we reach $\sigma^2(N)\overline{p}^2 = N\overline{p}^2$. Further, assuming that the individual weights, p_i, follow a Gaussian distribution (which in many cases is far from the truth) we have $N^2\sigma^2(\overline{p}) = \sigma^2(\sum_i p_i) = N\sigma^2(p_i)$ and reach

$$\sigma^2(I) = N\left(\overline{p}^2 + \sigma^2(p_i)\right). \tag{2.6}$$

The statistical variance of the p_i's is estimated by $\sigma^2(p_i) \approx (\sum_i p_i^2 - N\overline{p}^2)/(N-1)$. The resulting variance then reads

$$\sigma^2(I) = \frac{N}{N-1}\left(\sum_i p_i^2 - \overline{p}^2\right). \tag{2.7}$$

For almost any positive value of N, this is very well approximated by the simple expression

$$\sigma^2(I) \approx \sum_i p_i^2. \tag{2.8}$$

As a consistency check, we note that for all p_i equal, this reduces to eq. (2.4)

In order to compute the intensities and uncertainties, the detector components in McStas thus must keep track of $N = \sum_i p_i^0, I = \sum_i p_i^1$, and $M_2 = \sum_i p_i^2$.

2.3 Weight factor transformations during a Monte Carlo choice

When a Monte Carlo choice must be performed, *e.g.* when the initial energy and direction of the neutron ray is decided at the source, it is important to adjust the neutron weight so that the combined effect of neutron weight change and Monte Carlo probability of making this particular choice equals the actual physical properties we like to model.

Let us follow up on the simple example of transmission. The probability of transmitting the real neutron is T, but we make the Monte Carlo choice of transmitting the neutron ray each time: $f_{\mathrm{MC}} = 1$. This must be reflected on the choice of weight multiplier π_j gven by the master equation

$$f_{\mathrm{MC}}\pi_j = P. \tag{2.9}$$

This probability rule is general, and holds also if, e.g., it is decided to transmit only half of the rays ($f_{\mathrm{MC}} = 0.5$). An important different example is elastic scattering from a powder sample, where the Monte-Carlo choices are the particular powder line to scatter from, the scattering position within the sample and the final neutron direction within the Debye-Scherrer cone.

2.3.1 Direction focusing

An important application of weight transformation is direction focusing. Assume that the sample scatters the neutron rays in many directions. In general, only neutron rays in some of these directions will stand any chance of being detected. These directions we call the *interesting directions*. The idea in focusing is to avoid wasting computation time on neutrons scattered in the other directions. This trick is an instance of what in Monte Carlo terminology is known as *importance sampling.*

If *e.g.* a sample scatters isotropically over the whole 4π solid angle, and all interesting directions are known to be contained within a certain solid angle interval $\mathbf{\Delta\Omega}$, only these solid angles are used for the Monte Carlo choice of scattering direction. According to Eq. (2.9), the weight factor will then have to be changed by the amount $\pi_j = |\mathbf{\Delta\Omega}|/(4\pi)$. One thus ensures that the mean simulated intensity is unchanged during a "correct" direction focusing, while a too narrow focusing will result in a lower (*i.e.* wrong) intensity, since we cut neutrons rays that should have counted.

2.4 Adaptive sampling

Another strategy to improve sampling in simulations is adaptive importance sampling, where McStas during the simulations will determine the most interesting directions and gradually change the focusing according to that. Implementation of this idea is found in the **Source_adapt** and **Source_Optimizer** components.

Chapter 3

Source components

McStas contains a number of different source components, and any simulation will contain exactly one of these sources. The main function of a source is to determine a set of initial parameters $(\mathbf{r}, \mathbf{v}, t)$ for each neutron ray. This is done by Monte Carlo choices from suitable distributions. For example, in most present sources the initial position is found from a uniform distribution over the source surface, which can be chosen to be either circular or rectangular. The initial neutron velocity is selected within an interval of either the corresponding energy or the corresponding wavelength. Polarization is not relevant for sources, and we initialize the neutron average spin to zero: $\mathbf{s} = (0, 0, 0)$.

For time-of-flight sources, the choice of the emission time, t, is being made on basis of detailed analytical expressions. For other sources, t is set to zero. In the case one would like to use a steady state source with time-of-flight settings, the emission time of each neutron ray should be determined using a Monte Carlo choice. This may be achieved by the `EXTEND` keyword in the instrument description source as in the example below:

```
TRACE

COMPONENT MySource=Source_gen(...) AT (...)
EXTEND
%{
  t = 1e-3*randpm1(); /* set time to +/- 1 ms */
%}
```

3.0.1 Neutron flux

The flux of the sources deserves special attention. The total neutron intensity is defined as the sum of weights of all emitted neutron rays during one simulation (the unit of total neutron weight is thus neutrons per second). The flux, ψ, at an instrument is defined as intensity per area perpendicular to the beam direction.

The source flux, Φ, is defined in different units: the number of neutrons emitted per second from a 1 cm^2 area on the source surface, with direction within a 1 ster. solid angle, and with wavelength within a 1 Å interval. The total intensity of real neutrons emitted towards a given diaphragm (units: n/sec) is therefore (for constant Φ):

$$I_{\rm total} = \Phi A \Delta\Omega \Delta\lambda, \tag{3.1}$$

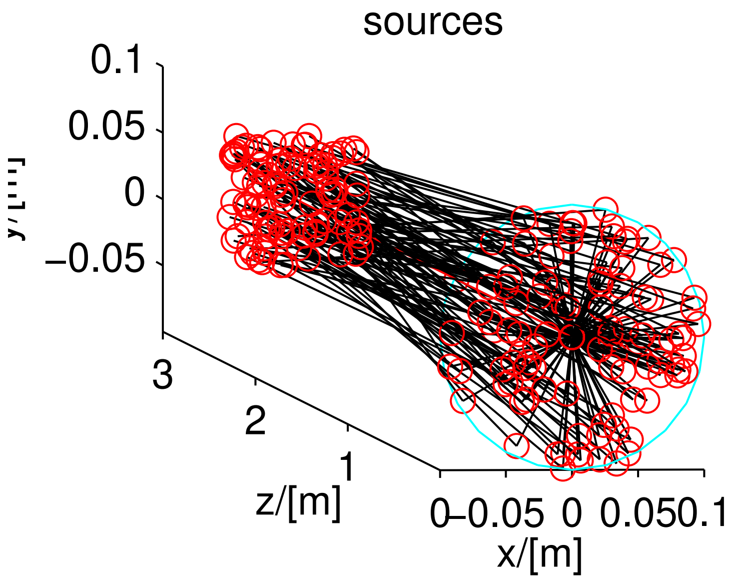


Figure 3.1: A circular source component (at z=0) emitting neutron events randomly, either from a model, or from a data file.

where A is the source area, $\Delta\Omega$ is the solid angle of the diaphragm as seen from the source surface, and $\Delta\lambda$ is the width of the wavelength interval in which neutrons are emitted (assuming a uniform wavelength spectrum).

The simulations are performed so that detector intensities are independent of the number of neutron histories simulated (although more neutron histories will give better statistics). If $N_{\rm sim}$ denotes the number of neutron histories to simulate, the initial neutron weight p_0 must be set to

$$p_0 = \frac{N_{\rm total}}{N_{\rm sim}} = \frac{\Phi(\lambda)}{N_{\rm sim}} A\Omega\Delta\lambda, \tag{3.2}$$

where the source flux is now given a λ-dependence.

As a start, we recommand new McStas users to use the **Source_simple** component. Slightly more realistic sources are **Source_Maxwell_3** for continuous sources or **Moderator** for time-of-flight sources.

Optimizers can dramatically improve the statistics, but may occasionally give wrong results, due to misleaded optimization. You should always check such simulations with (shorter) non-optimized ones.

Other ways to speed-up simulations are to read events from a file. See section 3.11 for details.

3.1 Source_simple: A simple continuous source with a flat energy/wavelength spectrum

Name:	Source_simple
Author:	System
Input parameters	r_s, z_{foc}, w, h, E_0, ΔE, Ψ
Optional parameters	λ_0, $d\lambda$
Notes	Validated. t=0

This component is a simple source with an energy distribution which is uniform in the range $E_0 \pm dE$ (alternatively: a wavelength distribution in the range $\lambda_0 \pm d\lambda$). This component is not used for detailed time-of-flight simulations, so we put $t = 0$ for all neutron rays.

The initial neutron ray position is chosen randomly from within a circle of radius r_s in the $z = 0$ plane. This geometry is a fair approximation of a cylindrical cold/thermal source with the beam going out along the cylinder axis.

The initial neutron ray direction is focused onto a rectangular target of width w, height h, parallel to the xy plane placed at $(0, 0, z_{foc})$.

The initial weight of the created neutron ray, p_0, is set to the energy-integrated flux, Ψ, times the source area, πr_s^2 times a solid-angle factor, which is basically the solid angle of the focusing rectangle. See also the section 3.0.1 on source flux.

This component replaces the obsolete components Source_flux_lambda, Source_flat, Source_flat_lambda, and Source_flux.

3.2 Source_div: A continuous source with specified divergence

Name:	Source_div
Author:	System
Input parameters	w, h, δ_h, δ_v, E_0, ΔE
Optional parameters	λ_0, $\Delta\lambda$, gauss
Notes	Validated. t=0

Source_div is a rectangular source, $w \times h$, which emits a beam of a specified divergence around the direction of the z axis. The beam intensity is uniform over the whole of the source, and the energy (or wavelength) distribution of the beam is uniform over the specified energy range $E_0 \pm \Delta E$ (in meV), or alternatively the wavelength range $\lambda_0 \pm \delta\lambda$ (in Å).

The source divergencies are δ_h and δ_v (FWHM in degrees). If the `gauss` flag is set to 0 (default), the divergence distribution is uniform, otherwise it is Gaussian.

This component may be used as a simple model of the beam profile at the end of a guide or at the sample position.

3.3 Source_Maxwell_3: A continuous source with a Maxwellian spectrum

Name:	Source_Maxwell_3
Author:	System
Input parameters	h, w, d_{foc}, xw, yh, λ_{low}, λ_{high}, I_1, T_1
Optional parameters	I_2, T_2, I_3, T_3
Notes	Validated. t=0

This component is a source with a Maxwellian energy/wavelength distribution sampled in the range λ_{low} to λ_{high}. The initial neutron ray position is chosen randomly from within a rectangle of area $h \times w$ in the $z = 0$ plane. The initial neutron ray direction is focused within a solid angle, defined by a rectangular target of width xw, height yh, parallel to the xy plane placed at $(0, 0, d_{\mathrm{foc}})$. The energy distribution used is a sum of 1, 2, or 3 Maxwellians with temperatures T_1 to T_3 and integrated intensities I_1 to I_3.

For one single Maxwellian, the intensity in a small wavelength interval $[\lambda, \lambda + d\lambda]$ is $I_1 M(\lambda, T_1)d\lambda$ where $M(\lambda, T_1) = 2\alpha^2 \exp(-\alpha/\lambda^2)/\lambda^5$ is the normalized Maxwell distribution ($\alpha = 949.0$ K Å$^2/T_1$). The initial weight of the created neutron ray, p_0, is calculated according to Eq. (3.2), with $\Psi(\lambda)$ replaced by $\sum_{j=1}^{3} I_j M(\lambda, T_j)$.

The component **Source_gen** (see section 3.4) works on the same principle, but provides more options concerning wavelength/energy range specifications, shape, etc.

Maxwellian parameters for some continuous sources are given in Table 4.1. As nobody knows exactly the characteristics of the sources (it is not easy to measure spectrum there), these figures should be used with caution.

3.4 Source_gen: A general continuous source

Name:	Source_gen
Author:	(System) E. Farhi, ILL
Input parameters	w, h, xw, yh, E_0, ΔE, T_1, T_2, T_3, I_1, I_2, I_3
Optional parameters	r, λ_0, $d\lambda$, E_{min}, E_{max}, λ_{min}, λ_{max}
Notes	Validated for Maxwellian expressions. t=0

This component is a continuous neutron source (rectangular or circular), which aims at a rectangular target centered at the beam. The angular divergence is given by the dimensions of the target. The shape may be rectangular (dimension h and w), or a disk of radius r. The wavelength/energy range to emit is specified either using center and half width, or using minimum and maximum boundaries, alternatively for energy and wavelength. The flux spectrum is specified with the same Maxwellian parameters as in component Source_Maxwell_3 (refer to section 3.3).

Maxwellian parameters for some continuous sources are given in Table 4.1. As nobody knows exactly the characteristics of the sources (it is not easy to measure spectrum there), these figures should be used with caution.

Source Name	T_1	I_1	T_2	I_2	T_3	I_3	factor
PSI cold source	150.4	3.67e11	38.74	3.64e11	14.84	0.95e11	* I_{target} (mA)
ILL VCS (H1)	216.8	1.24e13	33.9	1.02e13	16.7	3.042e12	
ILL HCS (H5)	413.5	10.22e12	145.8	3.44e13	40.1	2.78e13	*2
ILL Thermal(H2)	683.7	5.874e12	257.7	2.51e13	16.7	1.034e12	/2.25
ILL Hot source	1695	1.74e13	708	3.9e12			

Table 3.1: Flux parameters for present sources used in components Source_gen and Source_Maxwell_3. For some cases, a correction factor to the intensity should be used to reach measured data; for the PSI cold source, this correction factor is the beam current, I_{target}, which is currently of the order 1.2 mA.

3.5 Moderator: A time-of-flight source (pulsed)

Name:	Moderator
Author:	(System) Mark Hagen, SNS
Input parameters	r_s, E_0, E_1, z_f, w, h, τ_0, E_c, γ
Optional parameters	
Notes	

The simple time-of-flight source component **Moderator** resembles the source component **Source_simple** described in 3.1. **Moderator** is circular with radius r_s and focuses on a rectangular target of area $w \times h$ in a distance z_f. The initial velocity is chosen with a linear distribution within an interval, defined by the minimum and maximum energies, E_0 and E_1, respectively.

The initial time of the neutron is determined on basis of a simple heuristical model for the time dependence of the neutron intensity from a time-of-flight source. For all neutron energies, the flux decay is assumed to be exponential,

$$\Psi(E,t) = \exp(-t/\tau(E)), \tag{3.3}$$

where the decay constant is given by

$$\tau(E) = \begin{cases} \tau_0 & ; E < E_c \\ \tau_0/[1+(E-E_c)^2/\gamma^2] & ; E \geq E_c \end{cases} \tag{3.4}$$

The decay parameters are τ_0 (in μs), E_c, and γ (both in meV).

Other pulsed source models are available from contributed components. See section 3.11.

3.6 ISIS_moderator: ISIS pulsed moderators

Name:	ISIS_moderator
Author:	S. Ansell and D. Champion, ISIS
Input parameters	Face,$E0, E1, dist, xw, yh, CAngle, SAC$
Optional parameters	modXsize,modYsize
Notes	Validated. Low statistics above 20 Å. Kink aroung 9 Å.

3.6.1 Introduction

The following document describes the functions obtained for models of TS2 as described in Table 3.2:

target	3.4cm diameter tantalum clad tungsten
reflector	Be + D_2O (80:20) at 300K
Composite Moderator Coupled	H_2 + CH_4 Groove: 3x8.$\dot{3}$ cm 26K solid-CH_4 Hydrogen: 12x11cm 22K liquid H_2
Poisoned Moderator Decoupled	solid-CH_4 26K Narrow: Gd poison at 2.4 cm - 8 vanes Broad: 3.3 cm – not fully decoupled
PreModerators	0.85 cm and 0.75 cm H_2O

Table 3.2: Description of Models

TS1 model is from the tungsten target as currently installed and positioned. The model also includes the MERLIN moderator, this makes no significant difference to the other moderator faces.

3.6.2 Using the McStas Module

You MUST first set the environment variable 'MCTABLES' to be the full path of the directory containing the table files:

```
BASH: export MCTABLES=/usr/local/lib/mcstas/contrib/ISIS\_tables/\\
TCSH: setenv MCTABLES /usr/local/lib/mcstas/contrib/ISIS\_tables/
```

In Windows this can be done using the 'My Computer' properties and selecting the 'Advanced' tab and the Environment variables button. This can of course be overridden by placing the appropriate moderator (h.*face*) files in the working directory.

The module requires a set of variables listed in Table 3.3 and described below.

The *Face* variable determines the moderator surface that will be viewed. There are two types of *Face* variable: i) Views from the centre of each moderator face defined by the name of the moderator, for TS1: Water, H2, CH4, Merlin and TS2: Hydrogen, Groove, Narrow, Broad. ii) Views seen by each beamline, for TS1: Prisma, Maps, crisp etc. and for TS2: E1-E9 (East) and W1-W9 (West).

The McStas distribution includes some example moderator files for TS1 (water,h2,ch4) and TS2 (broad, narrow, hydrogen, groove), but others are available at `http://www.isis.rl.ac.uk/Computing/Sof` including instrument specific models.

Variables *E0* and *E1* define an energy window for sampled neutrons. This can be used to increase the statistical accuracy of chopper and mirrored instruments. However, *E0* and *E1* cannot be equal (although they can be close). By default these arguments select energy in meV, if negative values are given, selection will be in terms of Angstroms.

Variables *dist*, *xw* and *yh* are the three component which will determine the directional acceptance window. They define a rectangle with centre at (0,0,dist) from the moderator position and with width *xw* meters and height *yh* meters. The initial direction of all the neutrons are chosen (randomly) to originate from a point on the moderator surface and along a vector, such that without obstruction (and gravitational effects), they would pass through the rectangle. This should be used as a directional guide. All the neutrons start from the surface of the moderator and will be diverted/absorbed if they encountered other components. The guide system can be turned off by setting *dist* to zero.

The *CAngle* variable is used to rotate the viewed direction of the moderator and reduces the effective solid angle of the moderator face. Currently it is only for the horizontal plane. This is redundant since there are beamline specific h.*face* files.

The two variables *modYsize* and *modXsize* allow the moderators to be effectively reduced/increased. If these variables are given negative or zero values then they default to the actual visible surface size of the moderators.

The last variable *SAC* will correct for the different solid angle seen by two focussing windows which are at different distances from the moderator surface. The normal measurement of flux is in neutrons/second/Å/cm^2/str, but in a detector it is measured in neutrons/second. Therefore if all other denominators in the flux are multiplied out then the flux at a point-sized focus window should follow an inverse square law. This solid angle correction is made if the *SAC* variable is set equal to 1, it will not be calculated if *SAC* is set to zero. It is advisable to select this variable at all times as it will give the most realistic results

3.6.3 Comparing TS1 and TS2

The Flux data provided in both sets of h.*face* files is for 60 μAmp sources. To compare TS1 and TS2, the TS1 data must be multiplied by three (current average strength of TS1 source 180 μAmps). When the 300 μAmp upgrade happens this factor should be revised accordingly.

3.6.4 Bugs

Sometimes if a particularly long wavelength ($>$ 20 Å) is requested there may be problems with sampling the data. In general the data used for long wavelengths should only be taken as a guide and not used for accurate simulations. At 9 Åthere is a kink in the distribution which is also to do with the MCNPX model changing. If this energy is sampled over then the results should be considered carefully.

Variable	Type	Options	Units	Description
Face (TS2)	char*	i) Hydrogen Groove Narrow Broad, ii) E1-E9 W1-W9	–	String which designates the name of the face
Face (TS1)	char*	i) H2 CH4 Merlin Water, ii) Maps Crisp Gem EVS HET HRPD Iris Mari Polaris Prisma Sandals Surf SXD Tosca	–	String which designates the name of the face
E0	float	0<E0<E1	meV (Å)	Only neutrons above this energy are sampled
E1	float	E0<E1<1e10	meV (Å)	Only neutrons below this energy are sampled
dist	float	$0 < dist < \infty$	m	Distance of focus window from face of moderator
xw	float	$0 < xw < \infty$	m	x width of the focus window
yh	float	$0 < yh < \infty$	m	y height of the focus window
CAngle	float	-360 $< CAngle <$ 360	^{o}	Horizontal angle from the normal to the moderator surface
modXsize	float	$0 < modXsize < \infty$	m	Horizontal size of the moderator (defaults to actual size)
modYsize	float	$0 < modYsize < \infty$	m	Vertical size of the moderator (defaults to actual size)
SAC	int	0,1	n/a	Solid Angle Correction

Table 3.3: Brief Description of Variables

3.7 Source_adapt: A neutron source with adaptive importance sampling

Name:	Source_adapt
Author:	K. Nielsen
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, $E0$, dE, dist, xw, yh, Φ
Optional parameters	α, β (plenty, default values are ok)
Notes	partially validated

Source_adapt is a neutron source that uses adaptive importance sampling to improve the efficiency of the simulations. It works by changing on-the-fly the probability distributions from which the initial neutron state is sampled so that samples in regions that contribute much to the accuracy of the overall result are preferred over samples that contribute little. The method can achieve improvements of a factor of ten or sometimes several hundred in simulations where only a small part of the initial phase space contains useful neutrons. This component uses the correlation between neutron energy, initial direction and initial position.

The physical characteristics of the source are similar to those of **Source_simple** (see section 3.1). The source is a thin rectangle in the x-y plane with a flat energy spectrum in a user-specified range. The flux, Φ, per area per steradian per Ångstrøm per second is specified by the user.

The initial neutron weight is given by Eq. (3.2) using $\Delta\lambda$ as the total wavelength range of the source. A later version of this component will probably include a λ-dependence of the flux.

We use the input parameters *dist*, *xw*, and *yh* to set the focusing as for Source_simple (section 3.1). The energy range will be from $E_0 - dE$ to $E_0 + dE$. *filename* is used to give the name of a file in which to output the final sampling destribution, see below. $N_{\rm eng}$, $N_{\rm pos}$, and $N_{\rm div}$ are used to set the number of bins in each dimensions. Good general-purpose values for the optimization parameters are $\alpha = \beta = 0.25$. The number of bins to choose will depend on the application. More bins will allow better adaption of the sampling, but will require more neutron histories to be simulated before a good adaption is obtained. The output of the sampling distribution is only meant for debugging, and the units on the axis are not necessarily meaningful. Setting the filename to `NULL` disables the output of the sampling distribution.

3.7.1 Optimization disclaimer

A warning is in place here regarding potentially wrong results using optimization techniques. It is highly recommanded in any case to benchmark 'optimized' simulations against non-optimized ones, checking that obtained results are the same, but hopefully with a much improved statistics.

3.7.2 The adaption algorithm

The adaptive importance sampling works by subdividing the initial neutron phase space into a number of equal-sized bins. The division is done on the three dimensions of energy, horizontal position, and horizontal divergence, using $N_{\rm eng}$, $N_{\rm pos}$, and $N_{\rm div}$ number of bins in each dimension, respectively. The total number of bins is therefore

$$N_{\rm bin} = N_{\rm eng} N_{\rm pos} N_{\rm div} \tag{3.5}$$

Each bin i is assigned a sampling weight w_i; the probability of emitting a neutron within bin i is

$$P(i) = \frac{w_i}{\sum_{j=1}^{N_{\rm bin}} w_j} \tag{3.6}$$

In order to avoid false learning, the sampling weight of a bin is kept larger than $w_{\rm min}$, defined as

$$w_{\rm min} = \frac{\beta}{N_{\rm bin}} \sum_{j=1}^{N_{\rm bin}} w_j, \qquad 0 \leq \beta \leq 1 \tag{3.7}$$

This way a (small) fraction β of the neutrons are sampled uniformly from all bins, while the fraction $(1-\beta)$ are sampled in an adaptive way.

Compared to a uniform sampling of the phase space (where the probability of each bin is $1/N_{\rm bin}$), the neutron weight must be adjusted as given by (2.9)

$$\pi_1 = \frac{P_1}{f_{\rm MC,1}} = \frac{1/N_{\rm bin}}{P(i)} = \frac{\sum_{j=1}^{N_{\rm bin}} w_j}{N_{\rm bin} w_i}, \tag{3.8}$$

where P_1 is understood by the "natural" uniform sampling.

In order to set the criteria for adaption, the **Adapt_check** component is used (see section 3.8). The source attemps to sample only from bins from which neutrons are not absorbed prior to the position in the instrument at which **Adapt_check** is placed. Among those bins, the algorithm attemps to minimize the variance of the neutron weights at the **Adapt_check** position. Thus bins that would give high weights at the **Adapt_check** position are sampled more often (lowering the weights), while those with low weights are sampled less often.

Let $\pi = p_{\rm ac}/p_0$ denote the ratio between the neutron weight p_1 at the **Adapt_check** position and the initial weight p_0 just after the source. For each bin, the component keeps track of the sum Σ of π's as well as of the total number of neutrons n_i from that bin. The average weight at the **Adapt_source** position of bin i is thus Σ_i/n_i.

We now distribute a total sampling weight of β uniformly among all the bins, and a total weight of $(1-\beta)$ among bins in proportion to their average weight Σ_i/n_i at the **Adapt_source** position:

$$w_i = \frac{\beta}{N_{\rm bin}} + (1-\beta) \frac{\Sigma_i/n_i}{\sum_{j=1}^{N_{\rm bins}} \Sigma_j/n_j} \tag{3.9}$$

After each neutron event originating from bin i, the sampling weight w_i is updated.

This basic idea can be improved with a small modification. The problem is that until the source has had the time to learn the right sampling weights, neutrons may be emitted with high neutron weights (but low probability). These low probability neutrons may account for a large fraction of the total intensity in detectors, causing large variances in the result. To avoid this, the component emits early neutrons with a lower weight, and later neutrons with a higher weight to compensate. This way the neutrons that are emitted with the best adaption contribute the most to the result.

The factor with which the neutron weights are adjusted is given by a logistic curve

$$F(j) = C \frac{y_0}{y_0 + (1 - y_0)e^{-r_0 j}} \tag{3.10}$$

where j is the index of the particular neutron history, $1 \leq j \leq N_{\text{hist}}$. The constants y_0, r_0, and C are given by

$$y_0 = \frac{2}{N_{\text{bin}}} \tag{3.11}$$

$$r_0 = \frac{1}{\alpha} \frac{1}{N_{\text{hist}}} \log \left(\frac{1 - y_0}{y_0} \right) \tag{3.12}$$

$$C = 1 + \log \left(y_0 + \frac{1 - y_0}{N_{\text{hist}}} e^{-r_0 N_{\text{hist}}} \right) \tag{3.13}$$

The number α is given by the user and specifies (as a fraction between zero and one) the point at which the adaption is considered good. The initial fraction α of neutron histories are emitted with low weight; the rest are emitted with high weight:

$$p_0(j) = \frac{\Phi}{N_{\text{sim}}} A \Omega \Delta\lambda \frac{\sum_{j=1}^{N_{\text{bin}}} w_j}{N_{\text{bin}} w_i} F(j) \tag{3.14}$$

The choice of the constants y_0, r_0, and C ensure that

$$\int_{t=0}^{N_{\text{hist}}} F(j) = 1 \tag{3.15}$$

so that the total intensity over the whole simulation will be correct

Similarly, the adjustment of sampling weights is modified so that the actual formula used is

$$w_i(j) = \frac{\beta}{N_{\text{bin}}} + (1 - \beta) \frac{y_0}{y_0 + (1 - y_0)e^{-r_0 j}} \frac{\psi_i / n_i}{\sum_{j=1}^{N_{\text{bins}}} \psi_j / n_j} \tag{3.16}$$

3.7.3 The implementation

The heart of the algorithm is a discrete distribution p. The distribution has N *bins*, $1 \ldots N$. Each bin has a value v_i; the probability of bin i is then $v_i / (\sum_{j=1}^{N} v_j)$.

Two basic operations are possible on the distribution. An *update* adds a number a to a bin, setting $v_i^{\text{new}} = v_i^{\text{old}} + a$. A *search* finds, for given input b, the minimum i such that

$$b \leq \sum_{j=1}^{i} v_j. \tag{3.17}$$

The search operation is used to sample from the distribution p. If r is a uniformly distributed random number on the interval $[0; \sum_{j=1}^{N} v_j]$ then $i = \text{search}(r)$ is a random number distributed according to p. This is seen from the inequality

$$\sum_{j=1}^{i-1} v_j < r \leq \sum_{j=1}^{i} v_j, \tag{3.18}$$

from which $r \in [\sum_{j=1}^{i-1} v_j; v_i + \sum_{j=1}^{i-1} v_j]$ which is an interval of length v_i. Hence the probability of i is $v_i/(\sum_{j=1}^{N} v_j)$. The update operation is used to adapt the distribution to the problem at hand during a simulation. Both the update and the add operation can be performed very efficiently.

As an alternative, you may use the **Source_Optimizer** component (see section 3.9).

3.8 Adapt_check: The adaptative importance sampling monitor

Name:	Adapt_check
Author:	K. Nielsen
Input parameters	source_comp
Optional parameters	
Notes	validated

The component **Adapt_check** is used together with the Source_adapt component - see section 3.7 for details. When placed somewhere in an instrument using Source_adapt as a source, the source will optimize for neutrons that reach that point without being absorbed (regardless of neutron position, divergence, wavelength, *etc*).

The Adapt_check component takes as single input parameter *source_comp* the name of the Source_adapt component instance, for example:

```
...
COMPONENT mysource = Source_adapt(...)
...
COMPONENT mycheck = Adapt_check(source_comp = mysource)
...
```

Only one instance of Adapt_check is allowed in an instrument.

3.9 Source_Optimizer: A general Optimizer for McStas

Name:	Source_Optimizer
Author:	E. Farhi, ILL
Input parameters	options
Optional parameters	bins, step, keep
Notes	partially validated

The component **Source_Optimizer** is not exactly a source, but rather a neutron beam modifier. It should be positioned after the source, anywhere in the instrument description. The component optimizes the whole neutron flux in order to achieve better statistics at each **Monitor_Optimizer** location(s) (see section 3.10 for this latter component). It acts on any incoming neutron beam (from any source type), and more than one optimization criteria location can be placed along the instrument.

The usage of the optimizer is very simple, and usually does not require any configuration parameter. Anyway the user can still customize the optimization through various *options*.

In contrast to **Source_adapt**, this optimizer does not record correlations between neutron parameters. Nevertheless it is rather efficient, enabling the user to increase the number of events at optimization criteria locations by typically a factor of 20. Hence, the signal error bars will decrease by a factor 4.5, since the overall flux remains unchanged.

3.9.1 The optimization algorithm

When a neutron reaches the **Monitor_Optimizer** location(s), the component records its previous position (x, y) and speed (v_x, v_y, v_z) when it passed in the **Source_Optimizer**. Some distribution tables of *good* neutrons characteristics are then built.

When a *bad* neutron comes to the **Source_Optimizer** (it would then have few chances to reach **Monitor_Optimizer**), it is changed into a better one. That means that its position and velocity coordinates are translated to better values according to the *good* neutrons distribution tables. The neutron energy ($\sqrt{v_x^2 + v_y^2 + v_z^2}$) is kept (as far as possible).

The **Source_Optimizer** works as follow:

1. First of all, the **Source_Optimizer** determines some limits (*min* and *max*) for variables x, y, v_x, v_y, v_z.

2. Then the component records the non-optimized flux distributions in arrays with *bins* cells (default is 10 cells). This constitutes the *Reference* source.

3. The **Monitor_Optimizer** records the *good* neutrons (that reach it) and communicate an *Optimized* beam requirement to the **Source_Optimizer**. However, retains '*keep*' percent of the original *Reference* source is sent unmodified (default is 10 %). The *Optimized* source is thus:

$$\begin{array}{rcl} \textit{Optimized} & = & \textit{keep} * \textit{Reference} \\ & + & (1 - \textit{keep}) \text{ [Neutrons that will reach monitor].} \end{array}$$

4. The **Source_Optimizer** transforms the *bad* neutrons into *good* ones from the *Optimized* source. The resulting optimised flux is normalised to the non-optimized one:
$$p_{optimized} = p_{initial} \frac{\text{Reference}}{\text{Optimized}}, \tag{3.19}$$
and thus the overall flux at **Monitor_Optimizer** location is the same as without the optimizer. Usually, the process sends more *good* neutrons from the *Optimized* source than that in the *Reference* one. The energy (and velocity) spectra of neutron beam is also kept, as far as possible. For instance, an optimization of v_z will induce a modification of v_x or v_y to try to keep $|\mathbf{v}|$ constant.

5. When the *continuous* optimization option is activated (by default), the process loops to Step (3) every '*step*' percent of the simulation. This parameter is computed automatically (usually around 10 %) in *auto* mode, but can also be set by user.

During steps (1) and (2), some non-optimized neutrons with original weight $p_{initial}$ may lead to spikes on detector signals. This is greatly improved by lowering the weight p during these steps, with the *smooth* option. The component optimizes the neutron parameters on the basis of independant variables (1D phase-space optimization). However, it usually does work fine when these variables are correlated (which is often the case in the course of the instrument simulation). The memory requirements of the component are very low, as no big n-dimensional array is needed.

3.9.2 Using the Source_Optimizer

To use this component, just install the **Source_Optimizer** after a source (but any location is possible afterwards in principle), and use the **Monitor_Optimizer** at a location where you want to reach better statistics.

```
/* where to act on neutron beam */
COMPONENT optim_s = Source_Optimizer(options="")
...
/* where to have better statistics */
COMPONENT optim_m = Monitor_Optimizer(
xmin = -0.05, xmax = 0.05,
ymin = -0.05, ymax = 0.05,
optim_comp = optim_s)
...
/* using more than one Monitor_Optimizer is possible */
```

The input parameter for **Source_Optimizer** is a single *options* string that can contain some specific optimizer configuration settings in clear language. The formatting of the *options* parameter is free, as long as it contains some specific keywords, that can be sometimes followed by values.

The default configuration (equivalent to *options* = "") is

$$\begin{aligned} options \;=\;& \text{"}\textit{continuous}\text{ optimization, }\textit{auto}\text{ setting, }\textit{keep} = 0.1,\ \textit{bins} = 0.1, \\ & \textit{smooth}\text{ spikes, SetXY+SetDivV+SetDivS".} \end{aligned}$$

Parameters keep and step should be between 0 and 1. Additionally, you may restrict the optimization to only some of the neutron parameters, using the *SetXY, SetV, SetS, SetDivV, SetDivS* keywords. The keyword modifiers *no* or *not* revert the next option. Other options not shown here are:

```
verbose         displays optimization process (debug purpose).
unactivate      to unactivate the Optimizer.
file=[name]     Filename where to save optimized source distributions
```

The *file* option will save the source distributions at the end of the optimization. If no name is given the component name will be used, and a '.src' extension will be added. By default, no file is generated. The file format is in a McStas 2D record style.

As an alternative, you may use the Source_adapt component (see section 3.7) which performs a 3D phase-space optimization.

3.10 Monitor_Optimizer: Optimization locations for the Source_Optimizer

Name:	Source_Optimizer
Author:	E. Farhi, ILL
Input parameters	optim_comp
Optional parameters	x_{min}, x_{max}, y_{min},y_{max}
Notes	partially validated

The **Monitor_Optimizer** component works with the **Source_Optimizer** component. See section 3.9 for usage.

The input parameters for **Monitor_Optimizer** are the rectangular shaped opening coordinates x_{min}, x_{max}, y_{min}, y_{max}, and the name of the associated instance of **Source_Optimizer** used in the instrument description file (one word, without quotes).

As many Monitor_Optimizer instances as required may be used in an instrument, for possibly more than one optimization location. Multiple instances may all have an effect on the total intensity.

3.11 Other sources components: contributed pulsed sources, virtual sources (event files)

There are many other source definitions in McStas.

Detailed pulsed source components are available for new facilities in a number of contributed components:

- SNS (**contrib/SNS_source**),
- ISIS (**contrib/ISIS_moderator**) see section 3.6,
- ESS-project (**ESS_moderator_long** and **ESS_moderator_short**).

When no analytical model (e.g. a Maxwellian distribution) exits, one may have access to measurements, estimated flux distributions, event files, and - better - to MCNP/Triploli4 neutron event records. The following components are then useful:

- **misc/Virtual_input** can read a McStas event file (in text or binary format), often bringing an order-of-magnitude speed-up. See section 10.2.
- **contrib/Virtual_tripoli4_input** does the same, but from event files (text format) obtained from the *Tripoli4* [35] reactor simulation program. Such files are usually huge.
- **misc/Vitess_input** can read *Vitess* [29] neutron event binary files.
- **optics/Filter_gen** reads a 1D distribution from a file, and may either modify or set the flux according to it. See section 4.4.
- A component for reading MCNP "PTRAC" records is planed for a future release. Please contact us if you like to participate.

Chapter 4

Beam optical components: Arms, slits, collimators, and filters

This chapter contains a number of optical components that is used to modify the neutron beam in various ways, as well as the "generic" component **Arm**.

4.1 Arm: The generic component

Name:	Arm
Author:	System
Input parameters	(none)
Optional parameters	(none)
Notes	

The component **Arm** is empty; is resembles an optical bench and has no effect on the neutron ray. The purpose of this component is only to provide a standard means of defining a local co-ordinate system within the instrument definition. Other components may then be positioned relative to the **Arm** component using the McStas meta-language. The use of Arm components in the instrument definitions is not required but is recommended for clarity. **Arm** has no input parameters.

4.2 Slit: A beam defining diaphragm

Name:	Slit
Author:	System
Input parameters	$x_{\rm min}$, $x_{\rm max}$, $y_{\rm min}$, $y_{\rm max}$
Optional parameters	r, $p_{\rm cut}$
Notes	

The component **Slit** is a very simple construction. It sets up an opening at $z = 0$, and propagates the neutrons onto this plane (by the kernel call PROP_Z0). Neutrons within the slit opening are unaffected, while all other neutrons are discarded by the kernel call ABSORB.

By using Slit, some neutrons contributing to the background in a real experiment will be neglected. These are the ones that scatter off the inner side of the slit, penetrates the slit material, or clear the outer edges of the slit.

The input parameters of **Slit** are the four coordinates, $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$ defining the opening of the rectangle, or the radius r of a circular opening, depending on which parameters are specified.

The slit component can also be used to discard insignificant (*i.e.* very low weight) neutron rays, that in some simulations may be very abundant and therefore time consuming. If the optional parameter $p_{\rm cut}$ is set, all neutron rays with $p < p_{\rm cut}$ are ABSORB'ed. This use is recommended in connection with **Virtual_output**.

4.3 Beamstop: A neutron absorbing area

Name:	Beamstop
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}
Optional parameters	r
Notes	

The component **Beamstop** can be seen as the reverse of the **Slit** component. It sets up an area at the $z = 0$ plane, and propagates the neutrons onto this plane (by the kernel call PROP_Z0). Neutrons within this area are ABSORB'ed, while all other neutrons are unaffected.

By using this component, some neutrons contributing to the background in a real experiment will be neglected. These are the ones that scatter off the side of the beamstop, or penetrates the absorbing material. Further, the holder of the beamstop is not simulated.

Beamstop can be either circular or rectangular. The input parameters of **Beamstop** are the four coordinates, $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$ defining the opening of a rectangle, or the radius r of a circle, depending on which parameters are specified.

If the "direct beam" (e.g. after a monochromator or sample) should not be simulated, it is possible to emulate an ideal beamstop so that only the scattered beam is left; without the use of **Beamstop**: This method is useful for instance in the case where only neutrons scattered from a sample are of interest. The example below removes the direct beam and any background signal from other parts of the instrument

```
COMPONENT MySample=V_sample(...) AT (...)
EXTEND
%{
  if (!SCATTERED) ABSORB;
%}
```

File name	Description
Be.trm	Berylium filter for cold neutron spectrometers (e.g. $k < 1.55$ Å^{-1})
HOPG.trm	Highly oriented pyrolithic graphite for $\lambda/2$ filtering (e.g. thermal beam at $k = 1.64$, 2.662, and 4.1 Å^{-1})
Sapphire.trm	Sapphire (Al_2O_3) filter for fast neutrons ($k > 6$ Å^{-1})

Table 4.1: Some transmission data files to be used with e.g. the Filter_gen component

4.4 Filter_gen: A general filter using a transmission table

Name:	Filter_gen
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, $file$
Optional parameters	*options*
Notes	validated, flat filter

This component is an ideal flat filter that changes the neutron flux according to a 1D input table (text file).

Filter_gen may act as a source (*options*="set") or a filter (*options*="multiply", default mode). The table itself is a 2 column free format file which accept comment lines. The first table column represent wavevector, energy, or wavelength, as specified in the *options* parameter, whereas the second column is the transmission/weight modifier.

A usage example as a source would use `options="wavelength, set"`, if the first column in the data is suppposted to be λ (in Å). Another example using the component as a filter would be `options="energy, multiply"` if the first column is E (in meV).

The input parameters are the filter window size x_{min}, x_{max}, y_{min}, y_{max}, the behaviour specification string *options* and the file to use $file$. Additionally, rescaling can be made automatic with the *scaling* and relative *thickness* parameters. If for instance the transmission data file corresponds to a 5 cm thick filter, and one would like to simulate a 10 cm thick filter, then use *thickness* = 2.

Some example data files are given with McStas in the `MCSTAS/data` directory as `*.trm` files for transmission.

The filter geometry is a flat plane. A geometry with finite thickness can be simulated by surrounding this component with two slits.

4.5 Collimator_linear: The simple Soller blade collimator

Name:	Collimator_linear
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, L, δ
Optional parameters	
Notes	

Collimator_linear models a standard linear Soller blade collimator. The collimator has two identical rectangular openings, defined by the x and y values. Neutrons not clearing both openings are ABSORB'ed. The length of the collimator blades is denoted L, while the distance between blades is called d.

The collimating effect is taken care of by employing an approximately triangular transmission through the collimator of width (FWHM) δ, which is given in arc minutes, *i.e.* $\delta = 60$ is one degree. If $\delta = 0$, the collimating effect is disabled, so that the component only consists of two rectangular apertures.

For a more detailed Soller collimator simulation, taking every blade into account, it is possible to use **Channeled_guide** with absorbing walls, see section 5.3.

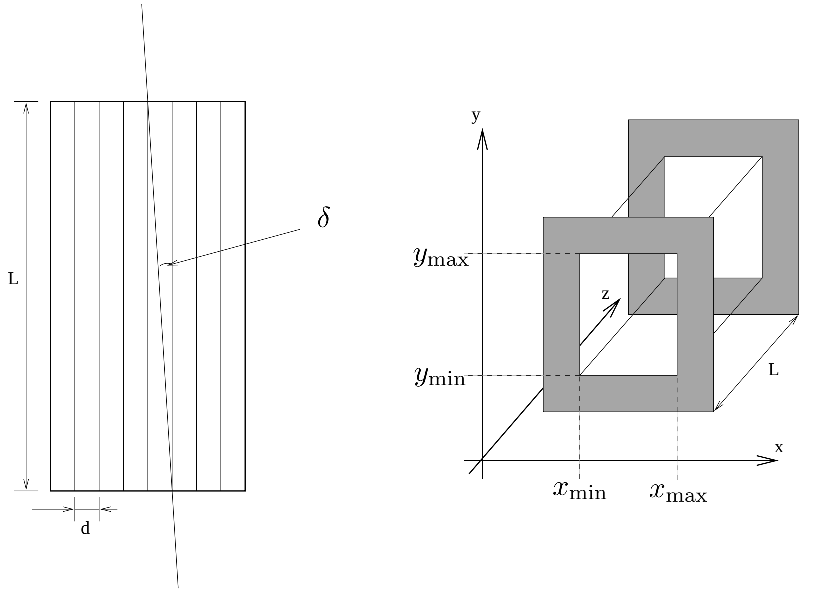


Figure 4.1: The geometry of a simple Soller blade collimators: The real Soller collimator, seen from the top (left), and a sketch of the component **Soller** (right). The symbols are defined in the text.

4.5.1 Collimator transmission

The horizontal divergence, η_h, is defined as the angle between the neutron path and the vertical $y - z$ plane along the collimator axis. We then define the collimation angle as the maximal allowed horizontal divergence: $\delta = \tan^{-1}(d/L)$, see Fig. 4.1. Neutrons

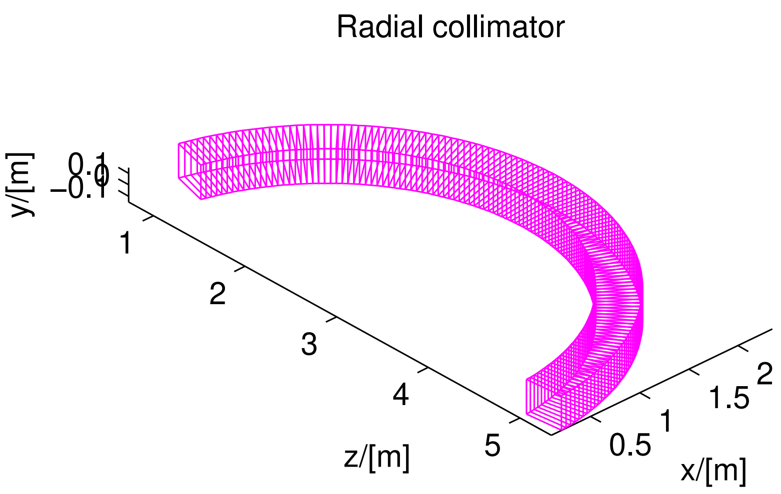


Figure 4.2: A radial collimator

with a horizontal divergence angle $|\eta_h| \geq \delta$ will always hit at least one collimator blade and will thus be ABSORB'ed. For smaller divergence angles, $|\eta_h| < \delta$, the fate of the neutron depends on its exact entry point. Assuming that a typical collimator has many blades, the absolute position of each blade perpendicular to the collimator axis is thus mostly unimportant. A simple statistical consideration now shows that the transmission probability is $T = 1 - \tan|\eta_h|/\tan\delta$. Often, the approximation $T \approx 1 - |\eta_h|/\delta$ is used, giving a triangular transmission profile.

4.5.2 Algorithm

The algorithm of Collimator_linear is roughly as follows:

1. Check by propagation if the neutron ray clear the entry and exit slits, otherwise ABSORB.
2. Check if $|\eta_h| < \delta$, otherwise ABSORB.
3. Simulate the collimator transmission by a weight transformation:

$$\pi_i = T = 1 - \tan|\eta_h|/\tan\delta, \tag{4.1}$$

4.6 Collimator_radial: A radial Soller blade collimator

Name:	Collimator_radial
Author:	(System) E.Farhi, ILL
Input parameters	w_1, h_1, w_2, h_2, len, θ_{min}, θ_{max}, $nchan$, $radius$
Optional parameters	$divergence$, $nblades$, roc and others
Notes	Validated

This radial collimator works either using an analytical approximation like **Collimator_linear** (see section 4.5), or with an exact model.

The input parameters are the inner radius *radius*, the radial length *len*, the input and output window dimensions w_1, h_1, w_2, h_2, the number of Soller channels *nchan* (each of then being a single linear collimator) covering the angular interval $[\theta_{min}, \theta_{max}]$ angle with respect to the z-axis.

If the *divergence* parameter is defined, the approximation level is used as in Collimator_linear (see section 4.5). On the other hand, if you perfer to describe exactly the number of blades *nblades* assembled to build a single collimator channel, then the model is exact, and traces the neutron trajectory inside each Soller. The computing efficiency is then lowered by a factor 2.

The component can be made oscillating with an amplitude of *roc* times $\pm w_1$.

Chapter 5

Reflecting optical components: mirrors, and guides

This section describes advanced neutron optical components such as supermirrors and guides as well as various rotating choppers. A description of the reflectivity of a supermirror is found in section 5.1.

This section describes advanced neutron optical components such as supermirrors and guides. A description of the reflectivity of a supermirror is found in section 5.1.

5.1 Mirror: The single mirror

Name:	Mirror
Author:	System
Input parameters	l, h, m
Optional parameters	R_0, Q_c, W, α
Notes	validated, no gravitation support

The component **Mirror** models a single rectangular neutron mirror plate. It can be used as a sample component or to *e.g.* assemble a complete neutron guide by putting multiple mirror components at appropriate locations and orientations in the instrument definition, much like a real guide is build from individual mirrors.

In the local coordinate system, the mirror lies in the first quadrant of the x-y plane, with one corner at $(0, 0, 0)$.

The input parameters of this component are the rectangular mirror dimensions (l, h) and the values of R_0, m, Q_c, W, and α for the mirror reflectivity. As a special case, if $m = 0$ then the reflectivity is zero for all Q, *i.e.* the surface is completely absorbing.

This component may produce wrong results with gravitation.

5.1.1 Mirror reflectivity

To compute the reflectivity of the supermirrors, we use an empirical formula derived from experimental data [36], see Fig. 5.1. The reflectivity is given by

$$R = \begin{cases} R_0 & \text{if } Q \leq Q_\mathrm{c} \\ \frac{1}{2}R_0(1 - \tanh[(Q - mQ_\mathrm{c})/W])(1 - \alpha(Q - Q_\mathrm{c})) & \text{if } Q > Q_\mathrm{c} \end{cases} \quad (5.1)$$

Here Q is the length of the scattering vector (in Å^{-1}) defined by

$$Q = |\mathbf{k_i} - \mathbf{k_f}| = \frac{m_\mathrm{n}}{\hbar}|\mathbf{v_i} - \mathbf{v_f}|, \tag{5.2}$$

m_n being the neutron mass. The number m in (5.1) is a parameter determined by the mirror materials, the bilayer sequence, and the number of bilayers. As can be seen, $R = R_0$ for $Q < Q_\mathrm{c}$, where Q_c is the critical scattering wave vector for a single layer of the mirror material. At higher values of Q, the reflectivity starts falling linearly with a slope α until a "soft cut-off" at $Q = mQ_\mathrm{c}$. The width of this cut-off is denoted W. See the example reflection curve in figure 5.1.

It is **important** to notice that when $m < 1$, the reflectivity remains constant at $R = R_0$ up to $q = Qc$, and *not* $m.Q_c$. This means that $m < 1$ parameters behave like $m = 1$ materials.

5.1.2 Algorithm

The function of the component can be described as

1. Propagate the neutron ray to the plane of the mirror.
2. If the neutron trajectory intersects the mirror plate, it is reflected, otherwise it is left untouched.
3. Reflection of the incident velocity $\mathbf{v}_\mathrm{i} = (v_x, v_y, v_z)$ gives the final velocity $\mathbf{v}_\mathrm{f} = (v_x, v_y, -v_z)$.
4. Calculate $Q = 2m_\mathrm{n}v_z/\hbar$.
5. The neutron weight is adjusted with the amount $\pi_i = R(Q)$.
6. To avoid spending large amounts of computation time on very low-weight neutrons, neutrons for which the reflectivity is lower than about 10^{-10} are ABSORB'ed.

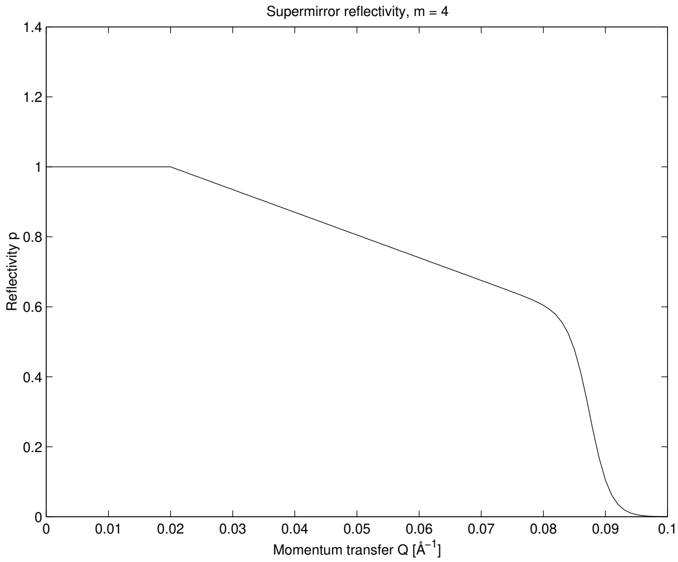


Figure 5.1: A typical reflectivity curve for a supermirror, Eq. (5.2). The used values are $m = 4$, $R_0 = 1$, $Q_c = 0.02$ Å^{-1}, $\alpha = 6.49$ Å, $W = 1/300$ Å^{-1}.

5.2 Guide: The guide section

Name:	Guide
Author:	System
Input parameters	$w_1, h_1,\ w_2, h_2,\ l,\ m$
Optional parameters	R_0, Q_c, W, α
Notes	validated, no gravitation support

The component **Guide** models a guide tube consisting of four flat mirrors. The guide is centered on the z axis with rectangular entrance and exit openings parallel to the x-y plane. The entrance has the dimensions (w_1, h_1) and placed at $z = 0$. The exit is of dimensions (w_2, h_2) and is placed at $z = l$ where l is the guide length. See figure 5.2. The reflecting properties are given by the values of R_0, m, Q_c, W, and α, as for **Mirror**.

Guide may produce wrong results with gravitation support. Use **Guide_gravity** (section 5.4) in this case. For a more general guide simulation, see **Guide_channeled** in section 5.3.

5.2.1 Guide geometry and reflection

For computations on the guide geometry, we define the planes of the four guide sides by giving their normal vectors (pointing into the guide) and a point lying in the plane:

$$\begin{array}{rclrcl}
\mathbf{n}_1^v & = & (l, 0, (w_2 - w_1)/2) & \mathbf{O}_1^v & = & (-w_1/2, 0, 0) \\
\mathbf{n}_2^v & = & (-l, 0, (w_2 - w_1)/2) & \mathbf{O}_2^v & = & (w_1/2, 0, 0) \\
\mathbf{n}_1^h & = & (0, l, (h_2 - h_1)/2) & \mathbf{O}_1^h & = & (0, -h_1/2, 0) \\
\mathbf{n}_2^h & = & (0, -l, (h_2 - h_1)/2) & \mathbf{O}_2^h & = & (0, h_1/2, 0)
\end{array}$$

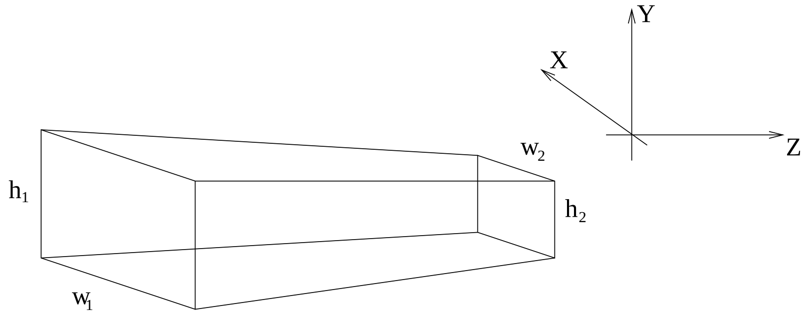


Figure 5.2: The geometry used for the guide component.

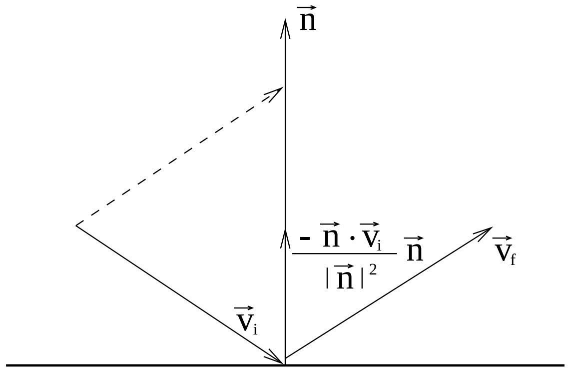


Figure 5.3: Neutron reflecting from mirror. $\mathbf{v}_\mathrm{i}$ and $\mathbf{v}_\mathrm{f}$ are the initial and final velocities, respectively, and $\mathbf{n}$ is a vector normal to the mirror surface.

In the following, we refer to an arbitrary guide side by its origin $\mathbf{O}$ and normal $\mathbf{n}$.

With these definitions, the time of intersection of the neutron with a guide side can be computed by considering the projection onto the normal:

$$t^\alpha_\beta = \frac{(\mathbf{O}^\alpha_\beta - \mathbf{r}_0) \cdot \mathbf{n}^\alpha_\beta}{\mathbf{v} \cdot \mathbf{n}^\alpha_\beta}, \tag{5.3}$$

where α and β are indices for the different guide walls, assuming the values (h,v) and (1,2), respectively. For a neutron that leaves the guide directly through the guide exit we have

$$t_\mathrm{exit} = \frac{l - z_0}{v_z} \tag{5.4}$$

The reflected velocity $\mathbf{v}_\mathrm{f}$ of the neutron with incoming velocity $\mathbf{v}_\mathrm{i}$ is computed by the formula

$$\mathbf{v}_\mathrm{f} = \mathbf{v}_\mathrm{i} - 2\frac{\mathbf{n} \cdot \mathbf{v}_\mathrm{i}}{|\mathbf{n}|^2}\mathbf{n} \tag{5.5}$$

This expression is arrived at by again considering the projection onto the mirror normal (see figure 5.3). The reflectivity of the mirror is taken into account as explained in section 5.1.

5.2.2 Algorithm

1. The neutron is initially propagated to the $z = 0$ plane of the guide entrance.
2. If it misses the entrance, it is ABSORB'ed.
3. Otherwise, repeatedly compute the time of intersection with the four mirror sides and the guide exit.
4. The smallest positive t thus found gives the time of the next intersection with the guide (or in the case of the guide exit, the time when the neutron leaves the guide).
5. Propagated the neutron ray to this point.
6. Compute the reflection from the side.
7. Update the neutron weight factor by the amount $\pi_i = R(Q)$.
8. Repeat this process until the neutron leaves the guide.

There are a few optimizations possible here to avoid redundant computations. Since the neutron is always inside the guide during the computations, we always have $(\mathbf{O}-\mathbf{r}_0)\cdot\mathbf{n} \leq 0$. Thus $t \leq 0$ if $\mathbf{v} \cdot \mathbf{n} \geq 0$, so in this case there is no need to actually compute t. Some redundant computations are also avoided by utilizing symmetry and the fact that many components of $\mathbf{n}$ and $\mathbf{O}$ are zero.

5.3 Guide_channeled: A guide section component with multiple channels

Name:	Guide_channeled
Author:	System
Input parameters	$w_1, h_1, w_2, h_2, l, k, m_x, m_y$
Optional parameters	$d, R_0, Q_{cx}, Q_{cy}, W, \alpha_x, \alpha_y$
Notes	validated, no gravitation support

The component **Guide_channeled** is a more complex variation of **Guide** described in the previous section. It allows the specification of different supermirror parameters for the horizontal and vertical mirrors, and also implements guides with multiple channels as used in neutron bender devices. By setting the m value of the supermirror coatings to zero, nonreflecting walls are simulated; this may be used for a very detailed simulation of a Soller collimator, see section 4.5.

The input parameters are w_1, h_1, w_2, h_2, and l to set the guide dimensions as for **Guide** (entry window, exit window, and length); k to set the number of channels; d to set the thickness of the channel walls; and R_0, W, Q_{cx}, Q_{cy}, α_x, α_y, m_x, and m_y to set the supermirror parameters as described under **Guide** (the names with x denote the vertical mirrors, and those with y denote the horizontal ones).

5.3.1 Algorithm

The implementation is based on that of **Guide**.

1. Calculate the channel which the neutron will enter.
2. Shift the x coordinate so that the channel can be simulated as a single instance of the **Guide** component.
3. (do the same as in **Guide**.)
4. Restore the coordinates when the neutron exits the guide or is absorbed.

5.3.2 Known problems

- This component may produce wrong results with gravitation support. Use Guide_gravity (section 5.4) in this case.
- The focusing channeled geometry (for $k > 1$ and different values of w_1 and w_2) is buggy (wall slopes are not computed correctly, and the component 'leaks' neutrons).

5.4 Guide_gravity: A guide with multiple channels and gravitation handling

Name:	Guide_gravity
Author:	System
Input parameters	w_1, h_1, w_2, h_2, l, k, m
Optional parameters	d, R_0, Q_c, W, α, wavy, chamfers, k_h, n, G
Notes	validated, **with** gravitation support

This component is a variation of **Guide_channeled** (section 5.3) with the ability to handle gravitation effects and functional channeled focusing geometry. Channels can be specified in two dimensions, producing a 2D array (k, k_h) on smaller guide channels.

Waviness effects, supposed to be randomly distributed (*i.e.* non-periodic waviness) can be specified globally, or for each part of the guide section. Additionally, chamfers may be defined the same way. Chamfers originate from the substrate manufacturing, so that operators do not harm themselves with cutting edges. Usual dimensions are about tens of millimeters. They are treated as absorbing edges around guide plates, both on the input and output surfaces, but also aside each mirror.

The straight section of length l may be divided into n bits of same length within which chamfers are taken into account.

To activate gravitation support, either select the McStas gravitation support, or set the gravitation field strength G (e.g. -9.81 on Earth).

5.5 Bender: a bender model (non polarizing)

Name:	Bender
Author:	Philipp Bernhardt
Input parameters	r, W_{in}, l, w, h
Optional parameters	$k, d, R_{0[a,i,s]}, \alpha_{[a,i,s]}, m_{[a,i,s]}, Q_{c[a,i,s]}, W_{[a,i,s]}$
Notes	partly validated, no gravitation support

The Bender component is simulating an ideal curved neutron guide (bender). It is bent to the negative X-axis and behaves like a parallel guide in the Y axis. Opposite curvature may be achieved by a $(0, 0, 180)$ rotation (along Z-axis).

Bender radius r, entrance width w and height h are required parameters. To define the length, you may either enter the deviation angle W_{in} or the length l. Three different reflectivity profiles R_0, Q_c, W, m, α can be given (see section 5.1): for outer walls (index a), for inner walls (index i) and for the top and bottom walls (index s).

To get a better transmission coefficient, it is possible to split the bender into k channels which are separated by partitions with the thickness of d. The partitioning walls have the same coating as the exterior walls.

Because the angle of reflection doesn't change, the routine calculates the reflection coefficent for the concave and, if necessary, for the convex wall only onces, together with the number of reflections. Nevertheless the exact position, the time, and the divergence is calculated at the end of the bender, so there aren't any approximations.

The component is shown *straight* on geometrical views (mcdisplay/Trace), and the next component may be placed directly at distance $r.W_{in} = l$ *without* rotation.

Results have been compared succesfully with analytical formula in the case of an ideal reflection and cross-checked with the program `haupt`.

An other implementation of the Bender is available as the contributed component Guide_curved.

5.6 Curved guides

Real curved guides are usually made of many straight elements (about 1 m long) separated with small gaps (e.g. 1 mm). Sections of about 10 m long are separated with bigger gaps for accessibility and pumping purposes.

We give here an example description of such a section. Let us have a curved guide of total length L, made of n elements with a curvature radius R. Gaps of size d separate elements from each other. The rotation angle of individual straight guide elements is $\alpha_z = (L + d)/R * 180/\pi$ in degrees.

In order to build an independent curved guide section, we define `Arm` components at the begining and end of it.

```
COMPONENT CG_In = Arm() AT (...)

COMPONENT CG_1  = Guide_gravity(l=L/n, ...)
AT (0,0,0) RELATIVE PREVIOUS

COMPONENT CG_2  = Guide_gravity(l=L/n, ...)
AT (0,0,L/n+d) RELATIVE PREVIOUS
ROTATED (0, (L/n+d)/R*180/PI, 0) RELATIVE PREVIOUS
...
COMPONENT CG_Out = Arm() AT (0,0,L/n) RELATIVE PREVIOUS
```

The `Guide` component should be duplicated n times by copy-paste, but changing the instance name, e.g. CG_1, CG_2, ..., CG_n.

An implementation of a continuous curved guide has been contributed as component Guide_curved.

Chapter 6

Moving optical components: Choppers and velocity selectors

We list in this chapter some moving optical components, like chopper, that may be used for TOF class instrument simulations, and velocity selector used for partially monochromatize continuous beams.

6.1 Chopper: The disc chopper

Name:	Chopper
Author:	Phillipp Bernhardt, Univ. Erlangen (System)
Input parameters	w, R, f, n, ϕ
Optional parameters	IsFirst, n_{pulse}
Notes	

To cut a continuous neutron beam into short pulses, or to control the pulse shape (in time) from a pulsed source, one can use a disc chopper (see figure 6.1). This is a fast rotating disc with the rotating axis parallel to the neutron beam. The disk consists of neutron absorbing materials. To form the pulses the disk has openings through which the neutrons can pass.

Component **Chopper** has n openings, which are symmetrically positioned on the disc. The direction of rotation can be controlled, which allows to simulate e.g. counter-rotating choppers. The phase (in seconds) is defined by the time where one slit is positioned at the top. The sides of the slits are pointing towards the center of the disc. The disc is considered infinitely thin. There is no parameter for the size of the slits in the radial direction; use e.g. a **Slit** component in front of the chopper.

When simulating the chopping of a continuous beam, most of the neutrons could easily be lost. To improve efficiency, one can set the flag `IsFirst`, which will allow every neutron ray to pass the **Chopper**, but modify the time, t, to a (random) time at which it is possible to pass. Of course, there should be only one “first chopper” in any simulation. To simulate frame overlap from a “first chopper”, one can specify the number of frames to study by the parameter n_{pulse}.

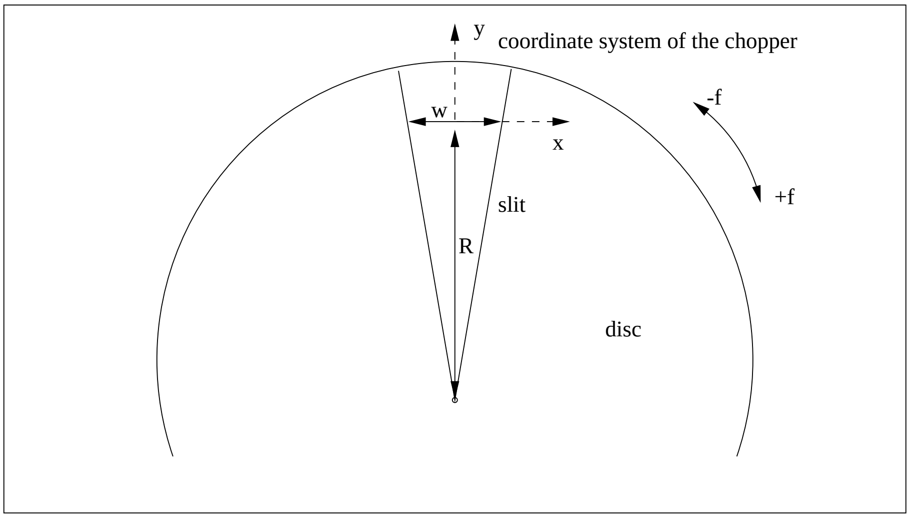


Figure 6.1: Sketch of a disc chopper with geometry parameters

6.2 FermiChopper: The Fermi-chopper

Name:	FermiChopper
Author:	M. Poehlmann, C. Carbogno, H. Schober, E. Farhi
Input parameters	$R, y_{min} y_{max}, \nu, w, length$,Nslit,phase
Optional parameters	$m, Q_c, R_0, \alpha, W, R_{slit}$,zero_time
Notes	validated

6.2.1 The chopper geometry and parameters

The Fermi chopper is a rotating vertical cylinder containing a set of collimating slits (*slit package*). Main geometry parameters are the radius R, minimum and maximum height y_{min} and y_{max} (see Fig. 6.2). In this implementation, the slits are straight, but may be coated with super-mirror. Main parameters for the slits are the number of slits $Nslit$, the length *length* and width w of each slit, the width of the separating Cd-blades is neglected. The slit walls reflectivity is modelled just like in guide components by the m-value ($m > 1$ for super mirrors), the critical scattering vector Q_c, the slope of reflectivity α, the low-angle reflectivity R_0 and the width of supermirror cut-off W. For $m = 0$ the blades are completly absorbing. The AT position of the component is its center.

The angular speed of the chopper is $\omega = 2\pi\nu$, where ν is the rotation frequency. The angle *phase* for which the chopper is in the 'open' state for most of the neutrons coming in (z' axis of the rotating frame parallel to the z axis of the static frame) is also an input parameter. The time window may optionally be shifted to zero when setting the `zero_time=1` option.

The curvature of the slit channels is specified with the R_{slit} parameter. Positive sign indicates that the deviation 'bump' due to curvature is in the x' positive side, and the center of curvature is in the x' negative side.

The component was validated extensivelly by K. Lieutenant. As an alternative, one

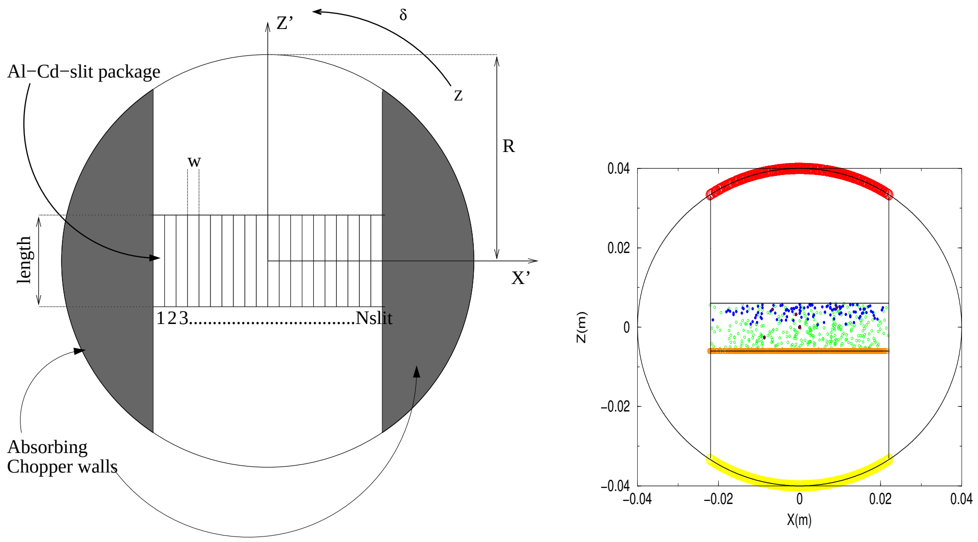


Figure 6.2: Geometry of the Fermi-chopper (left) and Neutrons in the chopper (right).

may use the **Vitess_ChopperFermi** component (eventhough slower and without supermirror support).

6.2.2 Propagation in the Fermi-chopper

As can be seen in figure 6.2, neutrons first propagate onto the cylinder surface of the chopper (yellow curve). Then the program checks the interaction with the entrance of the slit package (orange line) and calculates which slit is hit. If the slit coating is reflecting ($m > 0$), multiple reflections are calculated (green, blue and maroon circles), otherwise the neutrons are absorbed as soon as they interact with the blades. Finally the remaining neutrons propagate to the exit of the chopper (red curve).

The rotation of the chopper is characterized by the angle δ between the rotating z' and the static z-axis. $\delta(t)$ is defined by:

$$\delta(t) = \widehat{z, z'} = \omega.(t - t_0)$$

where t is the absolute time. The chopper should better be *time focussing*: slow neutrons should pass before the fast ones, so that they finally hit the detectors at the same time. Therefore the signs of ω and δ are very important: For $t > t_0$, δ is positive and points anti-clockwise.

Since the rotation is applied along the y - axis, we can simplify the problem to two dimensions. The orthogonal transformation matrix T from the static (xz) to the rotating frame $(x'z')$ is:

$$T_{xz \to x'z'} = \begin{pmatrix} \cos(\delta) & \sin(\delta) \\ -\sin(\delta) & \cos(\delta) \end{pmatrix} \qquad (6.1)$$

Together with the equation for a non-accelerated, linear propagation $\vec{r} = \vec{r_0} + \vec{v}t$ the orthogonal transformation produces a curve in the X'-Z'-plane known as *archidemic spiral*,

Parameter	unit	meaning
radius	[m]	chopper cylinder radius
ymin	[m]	lower y bound of cylinder
ymax	[m]	upper y bound of cylinder
Nslit	[1]	number of chopper slits
length	[m]	channel length of the Fermi chopper
w	[m]	width of one chopper slit. May also be specified as *width*=w*Nslit for total width of slit package.
nu	[Hz]	chopper frequency
phase	[deg]	chopper phase at t=0
zero_time	[1]	shit time window around 0 if true
curvature	[m^{-1}]	Curvature of slits (1/radius of curvature)
m	[1]	
alpha	[Å]	
Qc	[$Å^{-1}$]	slit coating parameters. See section 5.1.1
W	[$Å^{-1}$]	
R0	[1]	

Table 6.1: FermiChopper component parameters

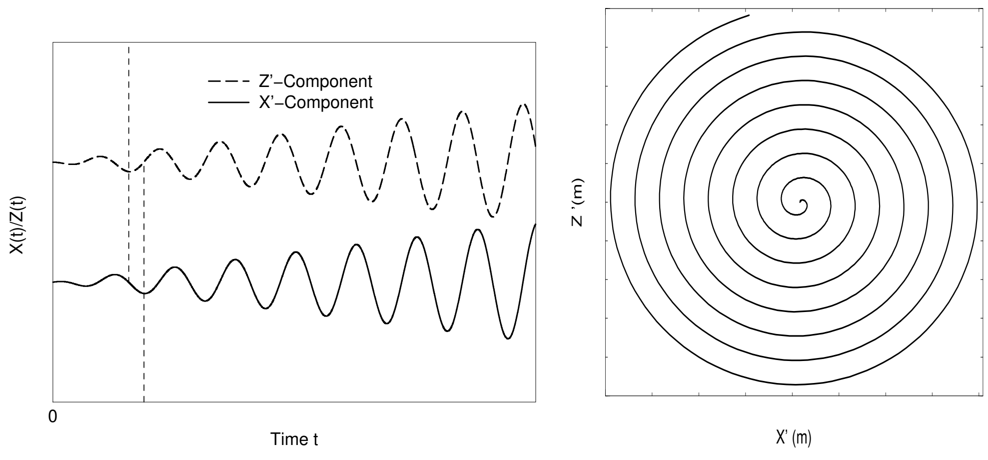


Figure 6.3: The x' and z' component as a function of time in the rotating frame (left). A typical neutron trajectory in the rotating frame (right).

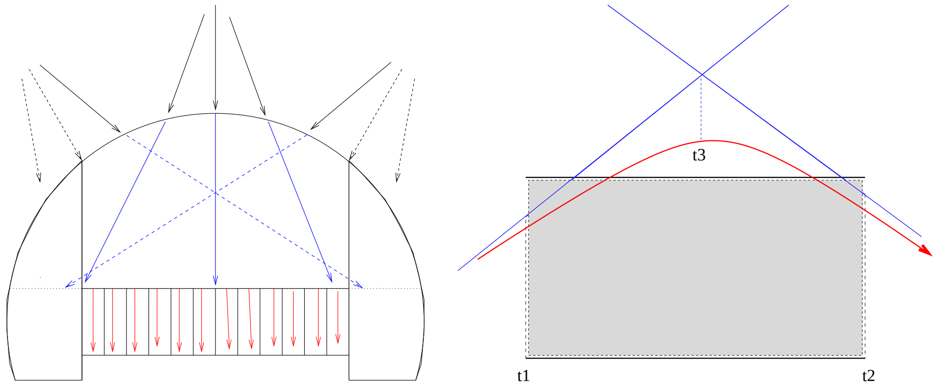


Figure 6.4: The different steps in the algorithm (left). A neutron trajectory in a slit (right)

as can be seen in figure 6.3. The two vector components $s(t) = (x', z')$ follow the equation:

$$s(t) = \begin{pmatrix} x' \\ z' \end{pmatrix} = T.\begin{pmatrix} x(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} (x_0 + v_x.t)cos(\delta(t)) + (z_0 + v_z.t)sin(\delta(t)) \\ -(x_0 + v_x.t)sin(\delta(t)) + (z_0 + v_z.t)cos(\delta(t)) \end{pmatrix}. \quad (6.2)$$

For a fixed chopper rotation speed, the neutron trajectory tends to strech from a spiral curve for slow neutrons to a straight line for fast neutrons. For real Fermi chopper settings ν (about 100 Hz on IN6 at the ILL), neutron trajectories are found to be nearly straight for 1000 m/s neutron velocities [37].

The basis of the algorithm is to find the intersections of these spiral trajectories with the chopper outer cylinder and then the slit package, in the rotating frame.

For this purpose, the *Ridders's* root finding method was implemented [38] in order to solve

$$x'(t) = d \text{ or } z'(t) = d \quad (6.3)$$

This method provides faster and more accurate intersection determination than other common algorithms. E.g. the secant method fails more often and may give wrong results (outside chopper) whereas the bisection method (a.k.a Picard dichotomy) is slightly slower.

Standard slit packages (non super-mirror)

The neutrons are first propagated to the outer chopper cylinder and their coordinates are transformed into the rotating frame using T. Neutrons outside the slit channel (chopper opening), or hitting the top and bottom caps are absorbed (yellow dots in Fig. 6.2). The side from which the neutron approaches the chopper is known (positive or negative z'-axis of the rotating frame) so that the calculation of the time of interaction with the slit package entrance t_1 is performed solving $z' = \pm\frac{\text{length}}{2}$ in Eq. (6.2). Using the result of the numerical algorithms the neutron propagates to the entrance of the slit package (orange circles in Fig. 6.2). Neutrons getting aside the slit package entrance are absorbed. Additionally, the slit package exit time t_2 is estimated the same way with $z' = \mp\frac{\text{length}}{2}$, in order to evaluate the whole time-of-flight in the chopper. The index of the slit which was hit is also computed, as we know the x' coordinate in the rotating frame at the slit entrance.

Differentiating Eq. (6.2) for x coordinate

$$\dot{x}'(t) = v_x'(t) = [v_x + \omega.(z + v_z(t))] \cos(\omega(t - t_0)) + [v_z - \omega.(x + v_x(t))] \sin(\omega(t - t_0)) \quad (6.4)$$

we may estimate the tangents to the spiral neutron trajectory in the rotating frame at times t_1 and t_2. The intersection of these two lines gives an intermediate time t_3.

If the neutron remains in the same slit at this point, then there is no intersection with the slit walls (direct flight), and the neutron may be propagated to the slit output, and then to the cylinder output. A last check is made for the neutron to pass the chopper aperture in the cylinder.

If the neutron changes of slit channel at this point, we may determine the intersection time of the neutron trajectory within $[t_1, t_3]$ or $[t_3, t_2]$, as seen in Fig. 6.4. If walls are not reflecting, we just absorb neutrons here.

The reflections (super-mirror slits)

If slit walls are reflecting, neutron is first propagated to the slit separating surface. Then the velocity in the rotating frame is computed using Eq. (6.2). Perpendicular velocity v_x' is reverted for reflection, and inverse T transformation is performed. Reflected intensity is computed the same way as for the guide component (see section 5.1). The remaining time t_2 to the slit output is estimated and the tangent intersection process is iterated, until neutron exits.

The propagation is finalized when determining the intersection of the neutron trajectory with the outer surface of the chopper cylinder. The neutron must then pass its aperture, else it is absorbed.

Curved slit packages

The effect of curvature can significantly improve the flux and energy resolution shape.

As all (xz) cordinates are transformed into $(x'z')$, the most efficient way to take into account the curvature is to include it in the transformation Eq. (6.2) by 'morphing' the curved rotating real space to a straight still frame. Then instead of solving

$$x'(t) = d - \Delta_{x'}(z') \text{ where } \Delta_{x'}(z') = R_{slit}.(1 - \sqrt{1 - (z'/R_{slit})^2}) \quad (6.5)$$

with Δ being the gap between the straight tangent line at the slit center and the real slit shape, we perform the additional transformation

$$x' \rightarrow x' + \Delta_{x'}(z') \quad (6.6)$$

The additional transformation counter-balance the real curvature so that the rest of the algorithm is written as if slits were straight. This applies to all computations in the rotating frame, and thus as well to reflections on super mirror coatings.

6.3 Vitess_ChopperFermi: The Fermi Chopper from Vitess

Name:	Vitess_ChopperFermi
Author:	Geza Zsigmond
Input parameters	GeomOption, N_{chan}, f, h, w_{tot}, l, r_{curv}, d, ϕ, w_{wall}, GeomFile
Optional parameters	zerotime, N_{gates}
Notes	validated

The component **Vitess_ChopperFermi** simulates a Fermi chopper with absorbing walls. The shape of the channels can be straight, curved with circular, or curved with ideal (i.e. close to a parabolic) shape. This is determined by the parameter 'GeomOption'. In the option 'straight Fermi chopper', the very fast neutrons are transmitted with only a time modulation and lower speed neutrons are modulated both in time of flight and wavelength. If the channels are curved, the highest transmission occurs for a wavelength

$$\lambda_{\text{opt}} = \frac{3956[\text{mÅ/s}]}{2\omega r_{\text{curv}}} \tag{6.7}$$

with

$$\omega = 2\pi f \tag{6.8}$$

The optimal shape is calculated in an exact way and is close to parabolic; in this case, transmission is as high for the optimal wavelength as in the case of a straight Fermi chopper for the limit $\lambda \to 0$. In the more realistic case of circular shapes channels, the transmission is slightly lower. In general, neutrons are transmitted through a curved Fermi chopper with a time AND wavelength modulation .

The rotation axis is vertical (y-axis), i.e. the path length through the channels is given by the length l along the z-axis. The inital orientation is given by the phase ϕ of the chopper - $\phi = 0$ means transmission orientation.

Geometry for **straight** and **circular** channels: The geometry of the chopper consists of a rectangular shaped object with a channel system. In transmission position, there are N_{gates} slits of width w_{slit} each along the x-axis, separated by absorbing walls of thickness w_{wall} (see figure 6.5). The total width w_{tot} is given by

$$w_{\text{tot}} = N_{\text{gates}} w_{\text{slit}} + (N_{\text{gates}} + 1) w_{\text{wall}} \tag{6.9}$$

The rectangular channel system is surrounded by a so-called shadowing cylinder; it is a part of a cylinder with vertical symmetry axis and diameter

$$d \geq \sqrt{l^2 + w_{\text{tot}}^2} \tag{6.10}$$

It serves to prevent transmission of neutrons which do not fly through the channels; but it also reduces the transmission, because the cylinder removes neutrons in front of the channel entrance or behind the channel exit (see figure 6.5).

Geometry for **parabolic** channels: In this case, the Fermi chopper is supposed to be a full cylinder, i.e. the central channels are longer than those on the edges. The other features are the same as for the other options. (see figure 6.6).

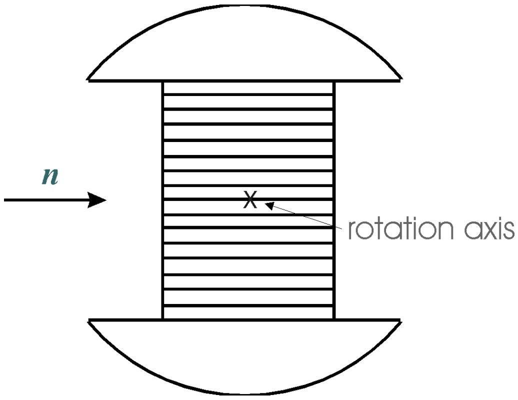


Figure 6.5: geometry of a staight Fermi chopper

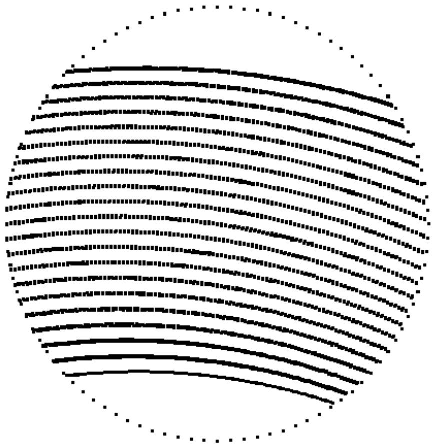


Figure 6.6: geometry of a curved Fermi chopper

The algorithm works with a rotating chopper framework. Neutrons hitting the channel walls are absorbed. The channels are approximated by N_{gates} gates. If the trajectory takes a course through all the gates, the neutron passes the Fermi chopper. There are gates at the entrance and the exit of the channel. The other gates are situated close to the centre of the Fermic chopper. Precision of the simulation increases with the number of gates, but also the computing time needed. The use of four channels already gives exact transmission shapes for lower wavelengths ($\lambda < 6$ Å) and good approximation for higher ones. It is recommended to use larger number of channels only for a check.

The option 'zerotime' may be used to reset the time at the chopper position. The time is set to a value between -$T_{\mathrm{p}}/2$ and +$T_{\mathrm{p}}/2$ (with T_{p} being the maximal pulse length), depending on the phase of the chopper at the moment of passing the chopper centre. The result is the generation of only 1 pulse instead of several; this is useful for TOF instruments on continuous sources.

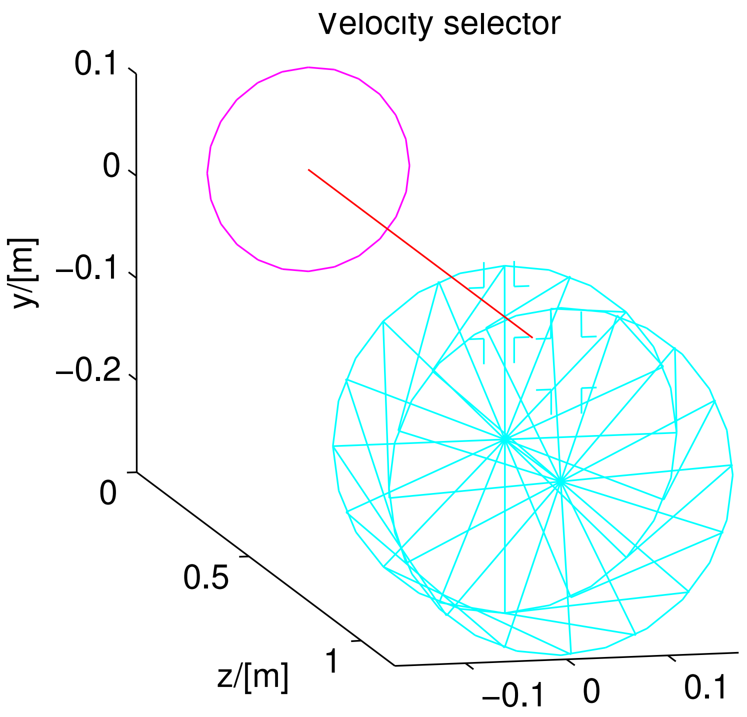


Figure 6.7: A velocity selector

6.4 V_selector: A rotating velocity selector

Name:	V_selector
Author:	System
Input parameters	L_0, L_1, ω, r_0, ϕ, N, height, width
Optional parameters	
Notes	validated, position is center of input aperture

The component **V_selector** models a rotating velocity selector constructed from N collimator blades arranged radially on an axis. Two identical slits (*height* $\times$ *width*) at a 12 o'clock position allow neutron passage at the position of the blades. The blades are "twisted" on the axis so that a stationary velocity selector does not transmit neutrons; the total twist angle is denoted ϕ (in degrees).

Further input parameters for **V_selector** the distance between apertures, L_0, the length of the collimator blades, L_1, the height from rotation axix to the slit centre, r_0, the rotation speed ω (in rpm), and the blade thickness t.

The local coordinate system has its Origo at the slit centre.

The component Selector produces equivalent results.

6.4.1 Velocity selector transmission

By rotating the selector you allow transmittance of neutrons rays with velocities around a nominal value, given by

$$V_0 = \omega L/\phi, \tag{6.11}$$

which means that the selector has turned the twist angle ϕ during the typical neutron flight time L/V_0. The actual twist angle is $\phi' = \omega t = \omega L/V$.

Neutrons having a velocity slightly different from V_0 will either be transmitted or absorbed depending on the exact position of the rotator blades when the neutron enters the selector. Assuming this position to be unknown and integrating over all possible positions (assuming zero thickness of blades), we arrive at a transmission factor

$$T = \begin{cases} 1 - (N/2\pi)|\phi - \omega L/V| & \text{if } (N/2\pi)|\phi - \omega L/V| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

where N is the number of collimator blades.

A horisontal divergence changes the above formula because of the angular difference between the entry and exit points of the neutron. The resulting transmittance resembles the one above, only with V replaced by V_z and ϕ replaced by $(\phi+\psi)$, where ψ is the angular difference due to the divergence. An additional vertical divergence does not change this formula, but it may contribute to ψ. (We have here ignored the very small non-linearity of ψ along the neutron path in case of both vertical and horisontal divergence).

Adding the effect of a finite blade thickness, t, reduces the transmission by the overall factor

$$\left(1 - \frac{Nt}{2\pi r}\right), \tag{6.13}$$

where r is the distance from the rotation axis. We ignore the variation of r along the neutron path and use just the average value.

Chapter 7

Monochromators

In this class of components, we are concerned with elastic Bragg scattering from monochromators. **Monochromator_flat** models a flat thin mosaic crystal with a single scattering vector perpendicular to the surface. The component **Monochromator_curved** is physically similar, but models a singly or doubly bend monochromator crystal arrangement.

A much more general model of scattering from a single crystal is found in the component **Single_crystal**, which is presented under Samples, chapter 8.

7.1 Monochromator_flat: An infinitely thin, flat mosaic crystal with a single scattering vector

Name:	Monochromator_flat
Author:	System
Input parameters	z_{min}, z_{max}, y_{min}, y_{max}, η_{h}, η_{v}, R_0, Q_0
Optional parameters	d_{m}
Notes	

This component simulates an infinitely thin single crystal with a single scattering vector, $Q_0 = 2\pi/d_m$, perpendicular to the surface. A typical use for this component is to simulate a simple monochromator or analyzer.

The monochromator dimensions are given by the length, z_{w}, and the height, y_{h}. As the parameter names indicate, the monochromator is placed in the $z-y$ plane of the local coordinate system. This definition is made to ensure that the physical monochromator angle (often denoted `A1`) will equal the McStas rotation angle of the Monochromator component around the y-axis. R_0 is the maximal reflectivity and η_{h} and η_{v} are the horizontal and vertical mosaicities, respectively, see explanation below.

7.1.1 Monochromator physics and algorithm

The physical model used in **Monochromator_flat** is a rectangular piece of material composed of a large number of small micro-crystals. The orientation of the micro-crystals deviates from the nominal crystal orientation so that the probability of a given micro-crystal orientation is proportional to a Gaussian in the angle between the given and the nominal orientation. The width of the Gaussian is given by the mosaic spread, η, of the

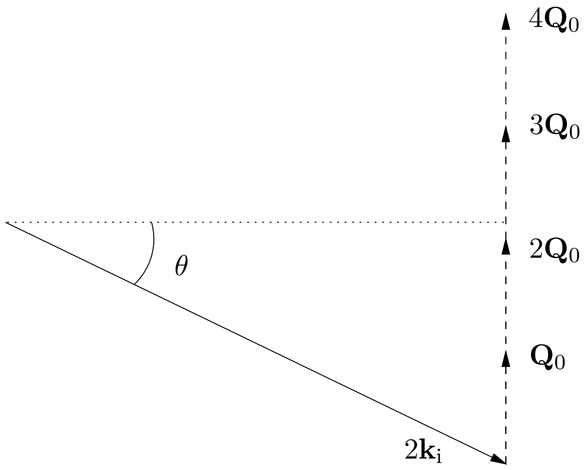


Figure 7.1: Selection of the Bragg order ("2" in this case).

crystal (given in units of arc minutes). η is assumed to be large compared to the inherent Bragg width of the scattering vector (often a few arc seconds). (The mosaicity gives rise to a Gaussian reflectivity profile of width similar to - but not equal - the intrinsic mosaicity. In this component, and in real life, the mosaicity given is that of the reflectivity signal.)

As a further simplification, the crystal is assumed to be infinitely thin. This means that multiple scattering effects are not simulated. It also means that the total reflectivity, r_0 is used as a parameter for the model rather than the atomic scattering cross section, implying that the scattering efficiency does not vary with neutron wavelength. The variance of the lattice spacing ($\Delta d/d$) is assumed to be zero, so this component is not suitable for simulating backscattering instruments (use the component Single_crystal in section 8.3 for that).

When a neutron trajectory intersects the crystal, the first step in the computation is to determine the probability of scattering. This probability is then used in a Monte Carlo choice deciding whether to scatter or transmit the neutron. The physical scattering probability is the sum of the probabilities of first- second-, and higher-order scattering - up to the highest order possible for the given neutron wavelength. However, in most cases at most one order will have a significant scattering probability, and the computation thus considers only the order that best matches the neutron wavelength.

The scattering of neutrons from a crystal is governed by Bragg's law:

$$n\mathbf{Q}_0 = 2\mathbf{k}_i \sin\theta \tag{7.1}$$

The scattering order is specified by the integer n. We seek only one value of n, namely the one which makes $n\mathbf{Q}_0$ closest to the projection of $2\mathbf{k}_i$ onto $\mathbf{Q}_0$ (see figure 7.1).

Once n has been determined, the Bragg angle θ can be computed. The angle α is the amount one would need to turn the nominal scattering vector $\mathbf{Q}_0$ for the monochromator to be in Bragg scattering condition. We now used α to compute the probability of reflection from the mosaic crystal

$$p_\mathrm{reflect} = R_0 e^{-\alpha^2/2\eta^2}, \tag{7.2}$$

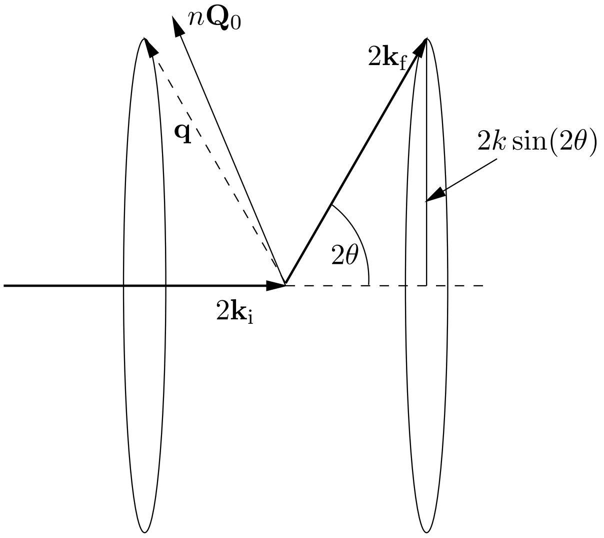


Figure 7.2: Scattering into the part of the Debye-Scherrer cone covered by the mosaic.

The probability p_reflect is used in a Monte Carlo choice to decide whether the neutron is transmitted or reflected.

In the case of reflection, the neutron will be scattered into the Debye-Scherrer cone, with the probability of each point on the cone being determined by the mosaic. The Debye-Scherrer cone can be described by the equation

$$\mathbf{k}_\mathrm{f} = \mathbf{k}_\mathrm{i}\cos 2\theta + \sin 2\theta(\mathbf{c}\cos\varphi + \mathbf{b}\sin\varphi), \qquad \varphi \in [-\pi;\pi], \tag{7.3}$$

where $\mathbf{b}$ is a vector perpendicular to $\mathbf{k}_\mathrm{i}$ and $\mathbf{Q}_0$, $\mathbf{c}$ is perpendicular to $\mathbf{k}_\mathrm{i}$ and $\mathbf{b}$, and both $\mathbf{b}$ and $\mathbf{c}$ have the same length as $\mathbf{k}_\mathrm{i}$ (see figure 7.2). When choosing φ (and thereby $\mathbf{k}_\mathrm{f}$), only a small part of the full $[-\pi;\pi]$ range will have appreciable scattering probability in non-backscattering configurations. The best statistics is thus obtained by sampling φ only from a suitably narrow range.

The (small) deviation angle α of the nominal scattering vector $n\mathbf{Q}_0$ corresponds to a Δq of

$$\Delta q \approx \alpha 2k\sin\theta. \tag{7.4}$$

The angle φ corresponds to a Δk_f (and hence Δq) of

$$\Delta q \approx \varphi k\sin(2\theta) \tag{7.5}$$

(see figure 7.2). Hence we may sample φ from a Gaussian with standard deviation

$$\alpha\frac{2k\sin\theta}{k\sin(2\theta)} = \alpha\frac{2k\sin\theta}{2k\sin\theta\cos\theta} = \frac{\alpha}{\cos\theta} \tag{7.6}$$

to get good statistics.

What remains is to determine the neutron weight. The distribution from which the scattering event is sampled is a Gaussian in φ of width $\frac{\alpha}{\cos\theta}$,

$$f_\mathrm{MC}(\varphi) = \frac{1}{\sqrt{2\pi}(\sigma/\cos\theta)}e^{-\varphi^2/2(\sigma/\cos\theta)^2} \tag{7.7}$$

In the physical model, the probability of the scattering event is proportional to a Gaussian in the angle between the nominal scattering vector $\mathbf{Q}_0$ and the actual scattering vector $\mathbf{q}$. The normalization condition is that the integral over all φ should be 1. Thus the probability of the scattering event in the physical model is

$$\Pi(\varphi) = e^{\frac{-d(\varphi)^2}{2\sigma^2}} / \int_{-\pi}^{\pi} e^{\frac{-d(\varphi)^2}{2\sigma^2}} d\varphi \tag{7.8}$$

where $d(\varphi)$ denotes the angle between the nominal scattering vector and the actual scattering vector corresponding to φ. According to equation (2.9), the weight adjustment π_j is then given by

$$\pi_j = \Pi(\varphi)/f_{\mathrm{MC}}(\varphi). \tag{7.9}$$

In the implementation, the integral in (7.8) is computed using a 15-order Gaussian quadrature formula, with the integral restricted to an interval of width $5\sigma/\cos\theta$ for the same reasons discussed above on the sampling of φ.

7.2 Monochromator_curved: A curved mosaic crystal with a single scattering vector

Name:	Monochromator_curved
Author:	(System) Peter Link, FRM-2
Input parameters	z_{w}, y_{h}, gap, η_{h}, η_{v}, n_{h}, n_{v}, R_0, Q, r_{h}, r_{v}
Optional parameters	d_{m}, η, h, w, verbose, transmit, reflect
Notes	

This component simulates an array of infinitely thin single crystals with a single scattering vector perpendicular to the surface and a mosaic spread. This component is used to simulate a singly or doubly curved monochromator or analyzer in reflecting geometry.

The component uses rectangular pieces of monochromator material as described in **Monochromator_curved**. The scattering vector is named Q, and as described in **Monochromator_flat**, multiples of Q will be applied. Other important parameters are the piece height and width, y_{h} and z_{w}, respectively, the horizontal and vertical mosaicities, η_{h} and η_{v}, respectively. If just one mosaicity, η, is specified, this the same for both directions.

The number of pieces vertically and horizontally are called n_{v} and n_{h}, respectively, and the vertical and horizontal radii of curvature are named r_{v} and r_{h}, respectively. All single crystals are positioned in the same vertical plane, but tilted accordingly to the curvature radius.

The constant monochromator reflectivity, R_0 can be replaced by a file of tabulated reflectivities $reflect$ (`*.rfl` in `MCSTAS/data`). In the same sense, the transmission can be modeled by a tabulated file *transmit* (for non-reflected neutrons, `*.trm` in `MCSTAS/data`). The most useful of these files for Monochromator_curved are `HOPG.rlf` and `HOPG.trm`.

As for **Monochromator_flat**, the crystal is assumed to be infinitely thin, and the varition in lattice spacing, $(\Delta d/d)$, is assumed to be zero. Hence, this component is not suitable for simulating backscattering instruments or to investigate multiple scattering effects.

The theory and algorithm for scattering from the individual blades is described under **Monochromator_flat**.

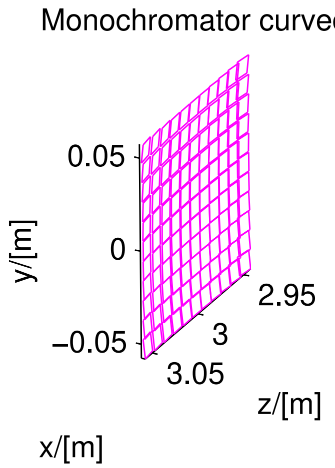


Figure 7.3: A curved monochromator

7.3 Single_crystal: Thick single crystal monochromator plate with multiple scattering

The **Single_crystal** component may be used to study more complex monochromators, including incoherent scattering, thickness and multiple scattering. Please refer to section 8.3.

Chapter 8

Samples

This class of components models the sample of the experiment. This is by far the most challenging part of a neutron scattering instrument to model. However, for purpose of simulating instrument performance, details of the samples are rather unimportant, allowing for simple approximations. On the contrary, for full virtual experiments it is of importance to have realistic and detailed sample descriptions. McStas contains both simple and detailed samples.

We first consider incoherent scattering. The simple component **V-sample** performs both incoherent scattering and absorption.

An important component class is elastic Bragg scattering from an ideal powder. The component **PowderN** models a powder scatterer with reflections given in an input file. The component includes absorption and incoherent scattering.

Next type is Bragg scattering from single crystals. The simplest single crystals are in fact the monochromator components like **Monochromator_flat**, presented in section 7.1. The monochromators are models of a thin mosaic crystal with a single scattering vector perpendicular to the surface. Much more advanced, the component **Single_crystal** is a general single crystal sample (with multiple scattering) that allows the input of an arbitrary unit cell and a list of structure factors, read from a LAZY / Crystallographica file. This component also allows anisotropic mosaicity and $\Delta d/d$ lattice space variation.

Isotropic small-angle scattering is simulated in **Sans_Spheres**, which models scattering from a collection of hard spheres.

Inelastic scattering from a dispersion is exemplified by the component **Phonon_simple**, which models scattering from a single acoustic phonon branch.

For a more general sample model, the **Isotropic_Sqw** component is able to simulate all kinds of isotropic materials: Liquids, glasses, polymers, powders, etc, with $S(q,\omega)$ table specified by an input file. Physical processes include coherent/incoherent scattering, both elastic and inelastic, with absorption and multiple scattering. Moreover, this component may be used concentrically, to model a sample environment. Thus it may handle most samples except single crystals.

Sample Process	Coherent Elastic	Coherent Inelastic	Incoherent Elastic	Incoherent Inelastic	Absorption	Multi. Scatt.
Phonon_simple		X			X	
Isotropic_Sqw	X	X	X	X	X	X
PowderN	N lines		X		X	
Sans_spheres	colloid				X	
Single_crystal	X		X		X	X
V_sample			X		X	

Table 8.1: Processes implemented in sample components

8.0.1 Neutron scattering notation

In sample component, we use the notation common for neutron scattering, where the wave vector transfer is denoted the *scattering vector*

$$\mathbf{q} \equiv \mathbf{k}_\mathrm{i} - \mathbf{k}_\mathrm{f}. \tag{8.1}$$

In analygo, the *energy transfer* is given by

$$\hbar\omega \equiv E_\mathrm{i} - E_\mathrm{f} = \frac{\hbar^2}{2m_\mathrm{n}}\left(k_\mathrm{i}^2 - k_\mathrm{f}^2\right). \tag{8.2}$$

8.0.2 Weight transformation in samples; focusing

Within many samples, the incident beam is attenuated by scattering and absorption, so that the illumination varies considerably throughout the sample. For single crystals, this phenomenon is known as *secondary extinction* [39], but the effect is important for all samples. In analytical treatments, attenuation is difficult to deal with, and is thus often ignored, making a *thin sample approximation.* In Monte Carlo simulations, the beam attenuation is easily taken care of, as will be shown below. In the description, we ignore multiple scattering, which is however implemented in some sample components.

The sample has an absorption cross section per unit cell of σ_c^a and a scattering cross section per unit cell of σ_c^s. The neutron path length in the sample before the scattering event is denoted by l_1, and the path length within the sample after the scattering is denoted by l_2, see figure 8.1. We then define the inverse penetration lengths as $\mu^s = \sigma_c^s/V_c$ and $\mu^a = \sigma_c^a/V_c$, where V_c is the volume of a unit cell. Physically, the attenuation along this path follows

$$f_\mathrm{att}(l) = \exp(-l(\mu^s + \mu^a)), \tag{8.3}$$

where the normalization $f_\mathrm{att}(0) = 1$.

The probability for a given neutron ray to be scattered from within the interval $[l_1; l_1 + dl]$ will be

$$P(l_1)dl = \mu^s f_\mathrm{att}(l_1)dl, \tag{8.4}$$

while the probability for a neutron to be scattered from within this interval into the solid angle Ω *and* not being scattered further or absorbed on the way out of the sample is

$$P(l_1, \Omega)dld\Omega = \mu^s f_\mathrm{att}(l_1) f_\mathrm{att}(l_2)\gamma(\Omega)d\Omega dl, \tag{8.5}$$

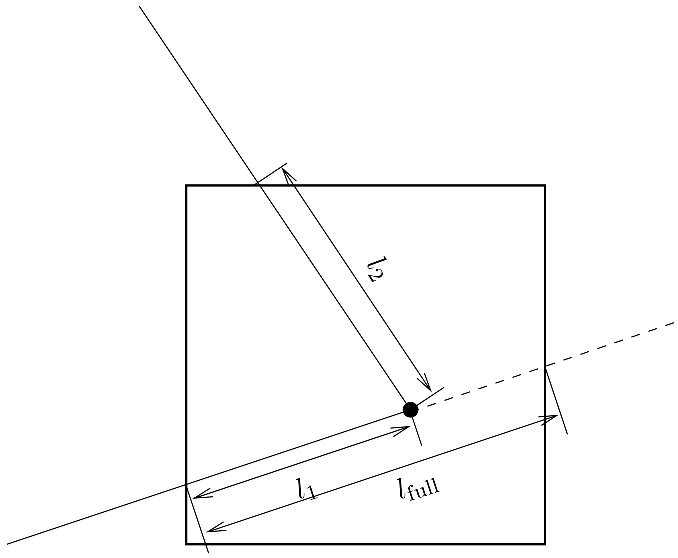


Figure 8.1: The geometry of a scattering event within a powder sample.

where $\gamma(\Omega)$ is the directional distribution of the scattered neutrons, and l_2 is determined by Monte Carlo chocies of l_1, Ω, and from the sample geometry, see e.g. figure 8.1.

In our Monte-Carlo simulations, we may choose the scattering parameters by making a Monte-Carlo choice of l_1 and Ω from a distribution different from $P(l_1, \Omega)$. By doing this, we must adjust π_i according to the probability transformation rule (2.9). If we *e.g.* choose the scattering depth, l_1, from a flat distribution in $[0; l_{\text{full}}]$, and choose the directional dependence from $g(\Omega)$, we have a Monte Carlo probability

$$f(l_1, \Omega) = g(\Omega)/l_{\text{full}}, \tag{8.6}$$

l_{full} is here the path length through the sample as taken by a non-scattered neutron (although we here assume that all simulated neutrons are being scattered). According to (2.9), the neutron weight factor is now adjusted by the amount

$$\pi_i(l_1, \Omega) = \mu^s l_{\text{full}} \exp\left[-(l_1 + l_2)(\mu^a + \mu^s)\right] \frac{\gamma(\Omega)}{g(\Omega)}. \tag{8.7}$$

In analogy with the source components, it is possible to define "interesting" directions for the scattering. One will then try to focus the scattered neutrons, choosing a $g(\Omega)$, which peaks around these directions. To do this, one uses (8.7), where the fraction $\gamma(\Omega)/g(\Omega)$ corrects for the focusing. One must choose a proper distribution so that $g(\Omega) > 0$ in every interesting direction. If this is not the case, the Monte Carlo simulation gives incorrect results. All samples have been constructed with a focusing and a non-focusing option.

8.0.3 Future development of sample components

There is still room for much more development of functionality in McStas samples.

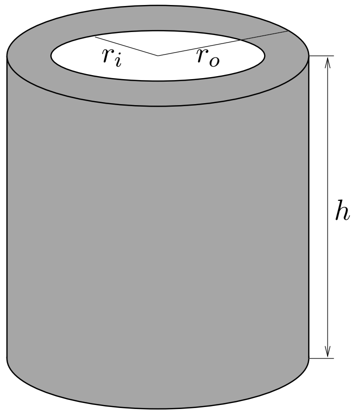


Figure 8.2: The geometry of the hollow-cylinder vanadium sample.

A more general SANS sample is under development. In addition, a reflectometry sample will soon be developed. In the mean time, you may use the `SiC` contributed component.

In general, all samples are assumed to be homogeneous. There would also be potential in developing an inhomogeneous sample, e.g. with spatially varying lattice constant, relevant for stress/strain scanners. Inhomogeneously absorbing sample for tomography could also be possible. Further, no polarization effects are yet taken into account in any of the samples.

8.1 V_sample: An incoherent scatterer, the V-sample

Name:	V_sample
Author:	System
Input parameters	$r_{\rm i}$, $r_{\rm o}$, h, $r_{\rm foc}$, $x_{\rm target}$, $y_{\rm target}$, $z_{\rm target}$
Optional parameters $\sigma_{\rm abs}$, $\sigma_{\rm inc}$, V_0, $f_{\rm pack}$, target_index	w_x, h_y, t_z, $w_{\rm focus}$, $h_{\rm focus}$, $w_{\rm foc,angle}$, $h_{\rm foc,angle}$,
Notes	validated

A sample with incoherent scattering, e.g. vanadium, is frequently used for calibration purposes, as this gives an isotropic, elastically scattered beam.

The component **V_sample** has *only* absorption and incoherent scattering. For the sample geometry, we default use a hollow cylinder (which has the solid cylinder as a limiting case). The sample dimensions are: Inner radius $r_{\rm i}$, outer radius $r_{\rm o}$, and height h, see figure 8.2.

Alternatively, the sample geometry can be made rectangular by specifying the width, w_x, the height, h_y, and the thickness, t_z.

The incoherent and absorption cross sections for V are default for the component. For other choices, the parameters $\sigma_{\rm inc}$, $\sigma_{\rm abs}$, and the unit cell volume V_0 should be specified. For a loosely packed sample, also the packing factor, $f_{\rm pack}$ can be specified (default value of 1).

8.1.1 Physics and algorithm

The incoherent scattering gives a uniform angular distribution of the scattered neutrons from each nucleus: $\gamma(\Omega) = 1/4\pi$. For the focusing we choose to have a uniform distribution on a target sphere of radius $r_{\rm foc}$, at the position $(x_{\rm target}, y_{\rm target}, z_{\rm target})$ in the local coordinate system. This gives an angular distribution (in a small angle approximation) of

$$g(\Omega) = \frac{1}{4\pi}\frac{x_{\rm t}^2 + y_{\rm t}^2 + z_{\rm t}^2}{(\pi r_{\rm t}^2)}. \tag{8.8}$$

The focusing can alternatively be performed on a rectangle with dimensions $w_{\rm focus}$, $h_{\rm focus}$, or uniformly in angular space (in a small-angle approximation), using $w_{\rm foc,angle}$, $h_{\rm foc,angle}$. The focusing location can be picked to be a downstream component by specifying `target_index`.

When calculating the neutron path length within the cylinder, the kernel function `cylinder_intersect` is used twice, once for the outer radius and once for the inner radius.

Multiple scattering is not inlcuded in this component. To obtain intensities similar to real measured ones, we therefore do not take attenuation from scattering into account for the outgoing neutron ray.

8.1.2 Remark on functionality

When simulating a realistic incoherent hollow cylinder sample one finds that the resulting direction dependence of the scattered intensity is *not* isotropic. This is explained by the variation of attenuation with scattering angle. One test result is shown in the instrument example chapter of the McStas User Manual.

8.2 PowderN: A general powder sample

Name:	Powder_N
Author:	System
Input parameters	r, h, $\sigma_{\rm abs}$, $\sigma_{\rm inc}$, V_0, $f_{\rm pack}$, filename, format, DW
Optional parameters	
Notes	validation in progress

The powder diffraction component **PowderN** models a powder sample with background coming only from incoherent scattering and no multiple scattering. The description of the powder comes from a file in one of the standard output formats LAZY, FULLPROF, or CRYSTALLOGRAPHICA.

8.2.1 Files formats: powder structures

Data files of type `lau` and `laz` in the McStas distribution data directory are self-documented in their header. They do not need any additional parameters to be used, as in the example:

```
PowderN(<geometry parameters>, filename="Al.laz")
```

Other column-based file formats may also be imported e.g. with parameters such as:

```
format=Crystallographica
format=Fullprof
format={1,2,3,4,0,0,0,0}
```

In the latter case, the indices define order of columns parameters multiplicity, lattice spacing, F^2, Debye-Waller factor and intrinsic line width.

Other component parameters may as well be specified in the data file header with lines e.g.:

```
#Vc
#sigma_abs
#sigma_inc
#Debye_Waller
#Delta_d/d
```

Further details on file formats are available in the `mcdoc` page of the component.

8.2.2 Geometry and physical properties

The sample has the shape of a solid cylinder, radius r and height h. The absorption and incoherent cross sections are given by σ_c^a σ_i, c^s.

The Bragg scattering from the powder, σ_c, c^a is calculated from the input file, with the parameters Q, $|F(Q)|^2$, and j for the scattering vector, structure factor, and multiplicity, respectively. The volume of the unit cell is denoted V_0, while the sample packing factor is $f_{\rm pack}$.

Focusing is performed by only scattering into one angular interval, $d\phi$ of the Debye-Scherrer circle. The center of this interval is located at the point where the Debye-Scherrer circle intersects the half-plane defined by the initial velocity, $\mathbf{v}_{\rm i}$, and a user-specified vector, $\mathbf{f}$.

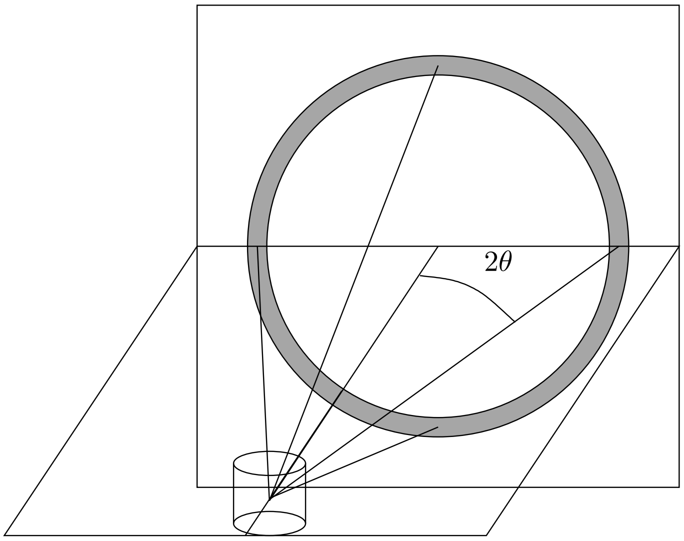


Figure 8.3: The scattering geometry of a powder sample showing part of the Debye-Scherrer cone (solid lines) and the Debye-Scherrer circle (grey).

8.2.3 Powder scattering

An ideal powder sample consists of many small crystallites, although each crystallite is sufficiently large not to cause measurable size broadening. The orientation of the crystallites is evenly distributed, and there is thus always a large number of crystallites oriented to fulfill the Bragg condition

$$n\lambda = 2d \sin\theta, \tag{8.9}$$

where n is the order of the scattering (an integer), λ is the neutron wavelength, d is the lattice spacing of the sample, and 2θ is the scattering angle, see figure 8.3. As all crystal orientations are realised in a powder sample, the neutrons are scattered within a *Debye-Scherrer cone* of opening angle 4θ [39].

Equation (8.9) may be cast into the form

$$|\mathbf{Q}| = 2|\mathbf{k}| \sin\theta, \tag{8.10}$$

where $\mathbf{Q}$ is a vector of the reciprocal lattice, and $\mathbf{k}$ is the wave vector of the neutron. It is seen that only reciprocal vectors fulfilling $|\mathbf{Q}| < 2|\mathbf{k}|$ contribute to the scattering. For a complete treatment of the powder sample, one needs to take into account all these $\mathbf{Q}$-values, since each of them contribute to the attenuation.

The strength of the Bragg reflections is given by their structure factors

$$\left| \sum_j b_j \exp(\mathbf{R}_j \cdot \mathbf{Q}) \right|^2, \tag{8.11}$$

where the sum runs over all atoms in one unit cell. This structure factor is non-zero only when Q equals a reciprocal lattice vector.

The textbook expression for the scattering cross section corresponding to one Debye-Scherrer cone reads [40, ch.3.6], with V being the total sample volume:

$$\sigma_{\text{cone}} = \frac{V}{V_0^2}\frac{\lambda^3}{4\sin\theta}\sum_Q |F(Q)|^2. \tag{8.12}$$

For our purpose, this expression should be changed slightly. Firstly, the sum over structure factors for a particular Q is replaced by the sum over essentially different reflections multiplied by their multiplicity, j. Then, a finite packing factor, f, is defined for the powder, and finally, the Debye-Waller factor is multiplied on the elastic cross section to take lattice vibrations into account (no inelastic background is simulated, however). We then reach

$$\sigma_{\text{cone,Q}} = j_Q f \exp(-2W)\frac{V}{V_0^2}\frac{\lambda^3}{4\sin\theta}|F(Q)|^2, \tag{8.13}$$

in the thin sample approximation. For samples of finite thickness, the beam is being attenuated by the attenuation coefficient

$$\mu_{\text{Q}} = \sigma_{\text{cone,Q}}/V. \tag{8.14}$$

For calibration it may be useful to consider the total intensity scattered into a detector of effective height h, covering only one reflection [40, ch.3.6]. A cut though the Debye-Scherrer cone perpendicular to its axis is a circle. At the distance r from the sample, the radius of this circle is $r\sin(2\theta)$. Thus, the detector (in a small angle approximation) counts a fraction $h/(2\pi r\sin(2\theta))$ of the scattered neutrons, giving a resulting count intensity:

$$I = \Psi\sigma_{\text{cone,Q}}\frac{h}{2\pi r\sin(2\theta)}, \tag{8.15}$$

where Ψ is the flux at the sample position.

For clarity we repeat the meaning and unit of the symbols:

Ψ	$\mathrm{s^{-1}m^{-2}}$	Incoming intensity of neutrons
I	$\mathrm{s^{-1}}$	Detected intensity of neutrons
h	m	Height of detector
r	m	Distance from sample to detector
f	1	Packing factor of the powder
j	1	Multiplicity of the reflection
V_0	$\mathrm{m^3}$	Volume of unit cell
$\lvert F(\mathbf{Q})\rvert^2$	$\mathrm{m^2}$	Structure factor
$\exp(-2W)$	1	Debye-Waller factor
μ_{Q}	$\mathrm{m^{-1}}$	Linear attenuation factor due to scattering from one powder line.

A powder sample will in general have several allowed reflections $\mathbf{Q}_j$, which will all contribute to the attenuation. These reflections will have different values of $|F(\mathbf{Q}_j)|^2$ (and hence of Q_j), j_j, $\exp(-2W_j)$, and θ_j. The total attenuation through the sample due to scattering is given by $\mu^s = \mu^s_{\text{inc}} + \sum_j \mu^s_j$, where μ^s_{inc} represents the incoherent scattering.

8.2.4 Algorithm

The algorithm of **PowderN** can be summarized as

- Check if the neutron ray intersects the sample (otherwise ignore the following).
- Calculate the attenuation coefficients for scattering and absorption.
- Perform Monte Carlo choices to determine the scattering position, scattering type (coherent/incoherent), and the outgoing direction.
- Perform the necessary weight factor transformation.

8.3 Single_crystal: The single crystal component

Name:	Single_crystal
Author:	Kristian Nielsen
Input parameters	$x_{width}, y_{height}, z_{thick}, \vec{a}, \vec{b}, \vec{c}, \Delta d/d$, mosaic, reflections
Optional parameters	$\sigma_{abs}, \sigma_{inc}$, ...
Notes	partially validated, centered

The **Single_crystal** component models a thick, flat single crystal with multiple scattering and absorption with elastic coherent scattering. An elastic incoherent background may also be simulated. It may be used to describe samples for diffraction, but also for accurate monochromator descriptions.

The input parameters for the component are *xwidth*, *yheight*, and *zthick* to define the dimensions of the crystal in meters (area is centered); *delta_d_d* to give the value of $\Delta d/d$ (no unit); (ax, ay, az), (bx, by, bz), and (cx, cy, cz) to define the axes of the direct lattice of the crystal (the sides of the unit cell) in units of Ångstrøm; and *reflections*, a string giving the name of the file with the list of structure factors to consider. The mosaic is specified *either* isotropically as *mosaic*, *or* anisotropically as *mosaic_h* (rotation around the Y axis), *mosaic_v* (rotation around the Z axis), and *mosaic_n* (rotation around the X axis); in all cases in units of full-width-half-maximum minutes of arc.

Optionally, the absorption cross-section at 2200 m/s and the incoherent cross-section may be given as *absorbtion* and *incoherent* (in barns), with default of zero; and *p_transmit* may be assigned a fixed Monte Carlo probability for transmission through the crystal without any interaction.

The user must specify a list of reciprocal lattice vectors $\boldsymbol{\tau}$ to consider along with their structure factors $|F_{\boldsymbol{\tau}}|^2$. The user must also specify the coordinates (in direct space) of the unit cell axes $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, from which the reciprocal lattice will be computed.

In addition to coherent scattering, **Single_crystal** also handles incoherent scattering amd absorption. The incoherent scattering cross-section is supplied by the user as a constant σ_{inc}. The absorption cross-section is supplied by the user at 2200 m/s, so the actual cross-section for a neutron of velocity v is $\sigma_{\mathrm{abs}} = \sigma_{2200} \frac{2200\ \mathrm{m/s}}{v}$.

8.3.1 The physical model

The textbook expression for the scattering cross-section of a crystal is [40, ch.3]:

$$\left(\frac{d\sigma}{d\Omega}\right)_{\mathrm{coh.el.}} = N\frac{(2\pi)^3}{V_0}\sum_{\boldsymbol{\tau}} \delta(\boldsymbol{\tau} - \boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 \qquad (8.16)$$

Here $|F_{\boldsymbol{\tau}}|^2$ is the structure factor (defined in section 8.2), N is the number of unit cells, V_0 is the volume of an individual unit cell, and $\boldsymbol{\kappa}(= \mathbf{k}_i - \mathbf{k}_f)$ is the scattering vector. $\delta(\boldsymbol{x})$ is a 3-dimensional delta function in reciprocal space, so for given incoming wave vector $\mathbf{k}_i$ and lattice vector $\boldsymbol{\tau}$, only a single final wave vector $\mathbf{k}_f$ is allowed. In general, this wavevector will not fulfill the conditions for elastic scattering ($k_f = k_i$). In a real crystal, however, reflections are not perfectly sharp. Because of imperfection and finite-size effects, there will be a small region around $\boldsymbol{\tau}$ in reciprocal space of possible scattering vectors.

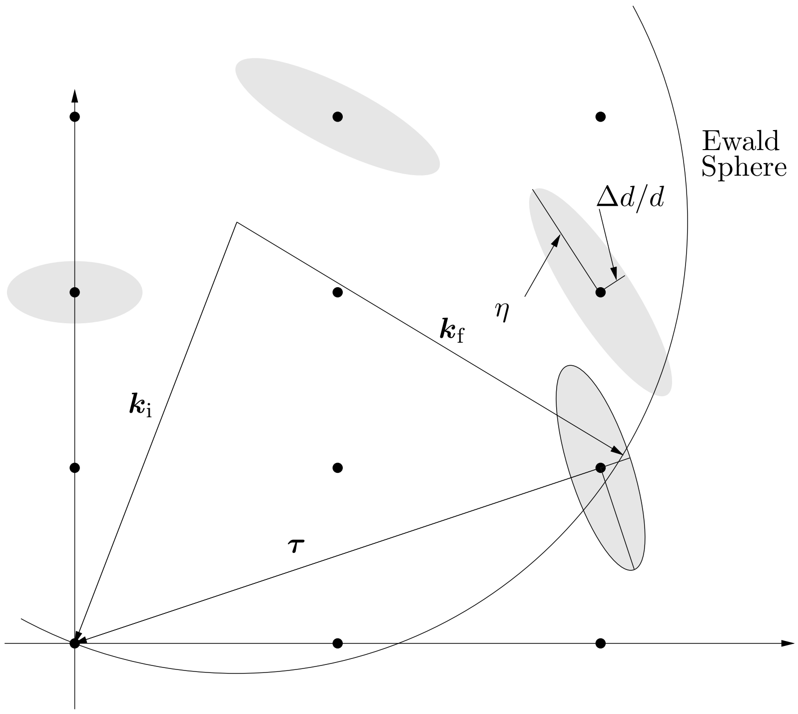


Figure 8.4: Ewald sphere construction for a single neutron showing the Gaussian broadening of reciprocal lattice points in their local coordinate system.

Single_crystal simulates a crystal with a mosaic spread η and a lattice plane spacing uncertainty $\Delta d/d$. In such crystals the reflections will not be completely sharp; there will be a small region around each reciprocal lattice point of the crystal that contains valid scattering vectors.

We model the mosaicity and $\Delta d/d$ of the crystal with 3-dimensional Gaussian functions in reciprocal space (see figure 8.4). Two of the axes of the Gaussian are perpendicular to the reciprocal lattice vector $\boldsymbol{\tau}$ and model the mosaicity. The third one is parallel to $\boldsymbol{\tau}$ and models $\Delta d/d$. We assume that the mosaicity is small so that the possible directions of the scattering vector may be approximated with a Gaussian in rectangular coordinates.

If the mosaic is isotropic (the same in all directions), the two Gaussian axes perpendicular to $\boldsymbol{\tau}$ are simply arbitrary normal vectors of equal length given by the mosaic. But if the mosaic is anisotropic, the two perpendicular axes will in general be different for each scattering vector. In the absence of anything better, **Single_crystal** uses a model which is at least mathematically plausible and which works as expected in the two common cases: (1) isotropic mosaic, and (2) two mosaic directions ("horizontal and vertical mosaic") perpendicular to a scattering vector.

The basis for the model is a three-dimensional Gaussian distribution in Euler angles giving the orientation probability distribution for the micro-crystals; that is, the misorientation is given by small rotations around the X, Y, and Z axes, with the rotation angles having (in general different) Gaussian probability distributions. For given scattering vector $\boldsymbol{\tau}$, a rotation of the micro-crystals around an axis parallel to $\boldsymbol{\tau}$ has no effect on the direction of the scattering vector. Suppose we form the intersection between the three-dimensional Gaussian in Euler angles and a plane through the origin perpendicular to $\boldsymbol{\tau}$. This gives a two-dimensional Gaussian, say with axes defined by unit vectors $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ and mosaic widths η_1 and η_2.

We now let the mosaic for $\boldsymbol{\tau}$ be defined by rotations around $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ with angles having Gaussian distributions of widths η_1 and η_2. Since $\boldsymbol{g}_1$, $\boldsymbol{g}_2$, and $\boldsymbol{\tau}$ are perpendicular, a small rotation of $\boldsymbol{\tau}$ around $\boldsymbol{g}_1$ will change $\boldsymbol{\tau}$ in the direction of $\boldsymbol{g}_2$. The two axes of the Gaussian mosaic in reciprocal space that are perpendicular to $\boldsymbol{\tau}$ will thus be given by $\tau\eta_2\boldsymbol{g}_1$ and $\tau\eta_1\boldsymbol{g}_2$.

We now derive a quantitative expression for the scattering cross-section of the crystal in the model. For this, we introduce a *local coordinate system* for each reciprocal lattice point $\boldsymbol{\tau}$ and use $\boldsymbol{x}$ for vectors written in local coordinates. The origin is $\boldsymbol{\tau}$, the first axis is parallel to $\boldsymbol{\tau}$ and the other two axes are perpendicular to $\boldsymbol{\tau}$. In the local coordinate system, the 3-dimensional Gaussian is given by

$$G(x_1, x_2, x_3) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\frac{1}{2}\left(\frac{x_1^2}{\sigma_1^2}+\frac{x_2^2}{\sigma_2^2}+\frac{x_3^2}{\sigma_3^2}\right)} \tag{8.17}$$

The axes of the Gaussian are $\sigma_1 = \tau\Delta d/d$ and $\sigma_2 = \sigma_3 = \eta\tau$. Here we used the assumption that η is small, so that $\tan\eta \approx \eta$ (with η given in radians). By introducing the diagonal matrix

$$D = \begin{pmatrix} \frac{1}{2}\sigma_1^2 & 0 & 0 \\ 0 & \frac{1}{2}\sigma_2^2 & 0 \\ 0 & 0 & \frac{1}{2}\sigma_3^2 \end{pmatrix}$$

equation (8.17) can be written as

$$G(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\boldsymbol{x}^{\mathrm{T}} D \boldsymbol{x}} \tag{8.18}$$

again with $\boldsymbol{x} = (x_1, x_2, x_3)$ written in local coordinates.

To get an expression in the coordinates of the reciprocal lattice of the crystal, we introduce a matrix U such that if $\boldsymbol{y} = (y_1, y_2, y_3)$ are the global coordinates of a point in the crystal reciprocal lattice, then $U(\boldsymbol{y}+\boldsymbol{\tau})$ are the coordinates in the local coordinate system for $\boldsymbol{\tau}$. The matrix U is given by

$$U^{\mathrm{T}} = (\hat{u}_1, \hat{u}_2, \hat{u}_3),$$

where $\hat{u}_1$, $\hat{u}_2$, and $\hat{u}_3$ are the axes of the local coordinate system, written in the global coordinates of the reciprocal lattice. Thus $\hat{u}_1 = \boldsymbol{\tau}/\tau$, and $\hat{u}_2$ and $\hat{u}_3$ are unit vectors perpendicular to $\hat{u}_1$ and to each other. The matrix U is unitarian, that is $U^{-1} = U^{\mathrm{T}}$. The translation between global and local coordinates is

$$\boldsymbol{x} = U(\boldsymbol{y}+\boldsymbol{\tau}) \qquad \boldsymbol{y} = U^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{\tau}$$

The expression for the 3-dimensional Gaussian in global coordinates is

$$G(\boldsymbol{y}) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-(U(\boldsymbol{y}+\boldsymbol{\tau}))^{\mathrm{T}} D (U(\boldsymbol{y}+\boldsymbol{\tau}))} \tag{8.19}$$

The elastic coherent cross-section is then given by

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{coh.el.}} = N\frac{(2\pi)^3}{V_0}\sum_{\boldsymbol{\tau}} G(\boldsymbol{\tau}-\boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 \tag{8.20}$$

8.3.2 The algorithm

The overview of the algorithm used in the Single_crystal component is as follows:

1. Check if the neutron intersects the crystal. If not, no action is taken.
2. Search through a list of reciprocal lattice points of interest, selecting those that are close enough to the Ewald sphere to have a non-vanishing scattering probability. From these, compute the total coherent cross-section σ_{coh} (see below), the absorption cross-section $\sigma_{\text{abs}} = \sigma_{2200}\frac{2200\text{ m/s}}{v}$, and the total cross-section $\sigma_{\text{tot}} = \sigma_{\text{coh}}+\sigma_{\text{inc}}+\sigma_{\text{abs}}$.
3. The transmission probability is $\exp(-\frac{\sigma_{\text{tot}}}{V_0}\ell)$ where ℓ is the length of the flight path through the crystal. A Monte Carlo choice is performed to determine whether the neutron is transmitted. Optionally, the user may set a fixed Monte Carlo probability for the first scattering event, for example to boost the statistics for a weak reflection.
4. For non-transmission, the position at which the neutron will interact is selected from an exponential distribution. A Monte Carlo choice is made of whether to scatter coherently or incoherently. Absorption is treated by weight adjustment (see below).
5. For incoherent scattering, the outgoing wave vector $\boldsymbol{k}_{\text{f}}$ is selected with a random direction.
6. For coherent scattering, a reciprocal lattice vector is selected by a Monte Carlo choice, and $\boldsymbol{k}_{\text{f}}$ is found (see below).
7. Adjust the neutron weight as dictated by the Monte Carlo choices made.
8. Repeat from (2) until the neutron is transmitted (to simulate multiple scattering).

For point 2, the distance *dist* between a reciprocal lattice point and the Ewald sphere is considered small enough to allow scattering if it is less than five times the maximum axis of the Gaussian, $dist \leq 5\max(\sigma_1,\sigma_2,\sigma_3)$.

8.3.3 Choosing the outgoing wave vector

The final wave vector $\boldsymbol{k}_{\text{f}}$ must lie on the intersection between the Ewald sphere and the Gaussian ellipsoid. Since η and $\Delta d/d$ are assumed small, the intersection can be approximated with a plane tangential to the sphere, see figure 8.5. The tangential point is taken to lie on the line between the center of the Ewald sphere $-\boldsymbol{k}_{\text{i}}$ and the reciprocal lattice point $\boldsymbol{\tau}$. Since the radius of the Ewald sphere is k_{i}, this point is

$$\boldsymbol{o} = (1-k_{\text{i}}/\rho)\boldsymbol{\rho} - \boldsymbol{\tau}$$

where $\rho = \boldsymbol{k}_{\text{i}} - \boldsymbol{\tau}$.

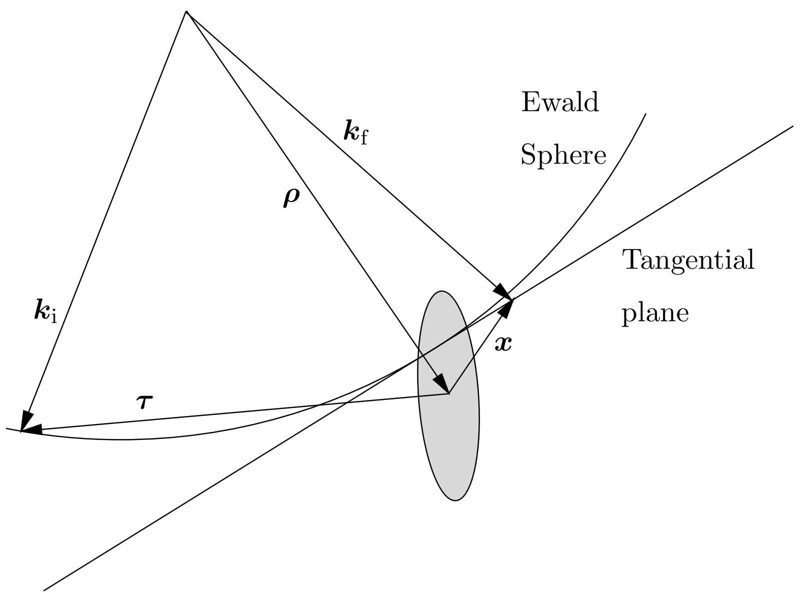


Figure 8.5: The scattering triangle in the single crystal.

The equation for the plane is

$$\boldsymbol{P}(\boldsymbol{t}) = \boldsymbol{o} + B\boldsymbol{t}, \qquad \boldsymbol{t} \in \mathbb{R}^2 \tag{8.21}$$

Here $B = (\boldsymbol{b}_1, \boldsymbol{b}_2)$ is a 3×2 matrix with the two generators for the plane $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$. These are (arbitrary) unit vectors in the plane, being perpendicular to each other and to the plane normal $\boldsymbol{n} = \boldsymbol{\rho}/\rho$.

Each $\boldsymbol{t}$ defines a potential final wave vector $\boldsymbol{k}_{\mathrm{f}}(\boldsymbol{t}) = \boldsymbol{k}_{\mathrm{i}} + \boldsymbol{P}(\boldsymbol{t})$. The value of the 3-dimensional Gaussian for this $\boldsymbol{k}_{\mathrm{f}}$ is

$$G(\boldsymbol{x}(\boldsymbol{t})) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\boldsymbol{x}(\boldsymbol{t})^{\mathrm{T}} D \boldsymbol{x}(\boldsymbol{t})} \tag{8.22}$$

where $\boldsymbol{x}(\boldsymbol{t}) = \boldsymbol{\tau} - (\boldsymbol{k}_{\mathrm{i}} - \boldsymbol{k}_{\mathrm{f}}(\boldsymbol{t}))$ is given in local coordinates for $\boldsymbol{\tau}$. It can be shown that equation (8.22) can be re-written as

$$G(\boldsymbol{x}(\boldsymbol{t})) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\alpha} e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}} M (\boldsymbol{t}-\boldsymbol{t}_0)} \tag{8.23}$$

where $M = B^{\mathrm{T}} D B$ is a 2×2 symmetric and positive definite matrix, $\boldsymbol{t}_0 = -M^{-1} B^{\mathrm{T}} D \boldsymbol{o}$ is a 2-vector, and $\alpha = -\boldsymbol{t}_0^{\mathrm{T}} M \boldsymbol{t}_0 + \boldsymbol{o}^{\mathrm{T}} D \boldsymbol{o}$ is a real number. Note that this is a two-dimensional Gaussian (not necessarily normalized) in $\boldsymbol{t}$ with center $\boldsymbol{t}_0$ and axis defined by M.

To choose $\boldsymbol{k}_{\mathrm{f}}$ we sample $\boldsymbol{t}$ from the 2-dimensional Gaussian distribution (8.23). To do this, we first construct the Cholesky decomposition of the matrix $(\frac{1}{2}M^{-1})$. This gives a 2×2 matrix L such that $LL^{\mathrm{T}} = \frac{1}{2}M^{-1}$ and is possible since M is symmetric and positive definite. It is given by

$$L = \begin{pmatrix} \sqrt{\nu_{11}} & 0 \\ \frac{\nu_{12}}{\sqrt{\nu_{11}}} & \sqrt{\nu_{22} - \frac{\nu_{12}^2}{\nu_{11}}} \end{pmatrix} \qquad \text{where } \frac{1}{2}M^{-1} = \begin{pmatrix} \nu_{11} & \nu_{12} \\ \nu_{12} & \nu_{22} \end{pmatrix}$$

Now let $\boldsymbol{g} = (g_1, g_2)$ be two random numbers drawn form a Gaussian distribution with mean 0 and standard deviation 1, and let $\boldsymbol{t} = L\boldsymbol{g} + \boldsymbol{t}_0$. The probability of a particular $\boldsymbol{t}$ is then

$$\begin{aligned}
P(\boldsymbol{t})d\boldsymbol{t} &= \frac{1}{2\pi}e^{-\frac{1}{2}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}}d\boldsymbol{g} && (8.24)\\
&= \frac{1}{2\pi}\frac{1}{\det L}e^{-\frac{1}{2}(L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0))^{\mathrm{T}}(L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0))}d\boldsymbol{t} && (8.25)\\
&= \frac{1}{2\pi}\frac{1}{\det L}e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}}M(\boldsymbol{t}-\boldsymbol{t}_0)}d\boldsymbol{t} && (8.26)
\end{aligned}$$

where we used that $\boldsymbol{g} = L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0)$ so that $d\boldsymbol{g} = \frac{1}{\det L}d\boldsymbol{t}$. This is just the normalized form of (8.23). Finally we set $\boldsymbol{k}'_{\mathrm{f}} = \boldsymbol{k}_{\mathrm{i}} + \boldsymbol{P}(\boldsymbol{t})$ and $\boldsymbol{k}_{\mathrm{f}} = (k_{\mathrm{i}}/k'_f)\boldsymbol{k}'_{\mathrm{f}}$ to normalize the length of $\boldsymbol{k}_{\mathrm{f}}$ to correct for the (small) error introduced by approximating the Ewald sphere with a plane.

8.3.4 Computing the total coherent cross-section

To determine the total coherent scattering cross-section, the differential cross-section must be integrated over the Ewald sphere:

$$\sigma_{\mathrm{coh}} = \int_{\mathrm{Ewald}} \left(\frac{d\sigma}{d\Omega}\right)_{\mathrm{coh.el.}} d\Omega$$

For small mosaic we may approximate the sphere with the tangential plane, and we thus get from (8.20) and (8.23):

$$\begin{aligned}
\sigma_{\mathrm{coh},\boldsymbol{\tau}} &= \int N\frac{(2\pi)^3}{V_0}G(\boldsymbol{\tau}-\boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 d\Omega && (8.27)\\
&= \frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^3}{V_0}\frac{1}{(\sqrt{2\pi})^3}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2\int e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}}M(\boldsymbol{t}-\boldsymbol{t}_0)}d\boldsymbol{t} && (8.28)\\
&= \det(L)\frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{3/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2\int e^{-\frac{1}{2}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}}d\boldsymbol{g} && (8.29)\\
&= 2\pi\det(L)\frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{3/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2 && (8.30)\\
&= \frac{\det(L)}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{5/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2 && (8.31)\\
\sigma_{\mathrm{coh}} &= \sum_{\boldsymbol{\tau}}\sigma_{\mathrm{coh},\boldsymbol{\tau}} && (8.32)
\end{aligned}$$

As before, we let $\boldsymbol{g} = L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0)$ so that $d\boldsymbol{t} = \det(L)d\boldsymbol{g}$.

Neutron weight factor adjustment We now calculate the correct neutron weight adjustment for the Monte Carlo choices made. In three cases is a Monte Carlo choice made with a probability different from the probability of the corresponding physical event: When deciding whether to transmit the neutron or not, when simulating absorption, and when selecting the reciprocal lattice vector $\boldsymbol{\tau}$ to scatter from.

If the user has choosen a fixed transmission probability $f(\text{transmit}) = p_{\text{transmit}}$, the neutron weight must be adjusted by

$$\pi(\text{transmit}) = \frac{P(\text{transmit})}{f(\text{transmit})}$$

where $\P(\text{transmit}) = \exp(-\frac{\sigma_{\text{tot}}}{V_0}\ell)$ is the physical transmission probability. Likewise, for non-transmission the adjustment is

$$\pi(\text{no transmission}) = \frac{1 - P(\text{transmit})}{1 - f(\text{transmit})}.$$

Absorption is never explicitly simulated, so the Monte Carlo probability of coherent or incoherent scattering is $f(\text{coh}) + f(\text{inc}) = 1$. The physical probability of coherent or incoherent scattering is

$$P(\text{coh}) + P(\text{inc}) = \frac{\sigma_{\text{coh}} + \sigma_{\text{inc}}}{\sigma_{\text{tot}}},$$

so again a weight adjustment $\pi(\text{coh}|\text{inc}) = \Pi(\text{coh}|\text{inc})/f(\text{coh}|\text{inc})$ is needed.

When choosing the reciprocal lattice vector $\boldsymbol{\tau}$ to scatter from, the relative probability for $\boldsymbol{\tau}$ is $r_{\boldsymbol{\tau}} = \sigma_{\text{coh},\boldsymbol{\tau}}/|F_{\boldsymbol{\tau}}|^2$. This is done to get better statistics for weak reflections. The Monte Carlo probability for the reciprocal lattice vector $\boldsymbol{\tau}$ is thus

$$f(\boldsymbol{\tau}) = \frac{r_{\boldsymbol{\tau}}}{\sum_{\boldsymbol{\tau}} r_{\boldsymbol{\tau}}}$$

whereas the physical probability is $P(\boldsymbol{\tau}) = \sigma_{\text{coh},\boldsymbol{\tau}}/\sigma_{\text{coh}}$. A weight adjustment is thus needed of

$$\pi(\boldsymbol{\tau}) = \frac{P(\boldsymbol{\tau})}{f(\boldsymbol{\tau})} = \frac{\sigma_{\text{coh},\boldsymbol{\tau}} \sum_{\boldsymbol{\tau}} r_{\boldsymbol{\tau}}}{\sigma_{\text{coh}}\ r_{\boldsymbol{\tau}}}.$$

In most cases, however, only one reflection is possible, whence $\pi = 1$.

8.3.5 Implementation details

The equations describing **Single_crystal** are quite complex, and consequently the code is fairly sizeable. Most of it is just the expansion of the vector and matrix equations in individual coordinates, and should thus be straightforward to follow.

The implementation pre-computes a lot of the necessary values in the `INITIALIZE` section. It is thus actually very efficient despite the complexity. If the list of reciprocal lattice points is big, however, the search through the list will be slow. The precomputed data is stored in the structures `hkl_info` and in an array of `hkl_data` structures (one for each reciprocal lattice point in the list). In addition, for every neutron event an array of `tau_data` is computed with one element for each reciprocal lattice point close to the Ewald sphere. Except for the search for possible $\boldsymbol{\tau}$ vectors, all computations are done in local coordinates using the matrix U to do the necessary transformations.

The list of reciprocal lattice points is specified in an ASCII data file. Each line contains seven numbers, separated by white space. The first three numbers are the (h, k, l) indices of the reciprocal lattice point, and the last number is the value of the structure factor $|F_{\boldsymbol{\tau}}|^2$, in barns. The middle three numbers are not used and may be omitted; they are nevertheless recommanded since this makes the file format compatible with the output from the Crystallographica program [41]. Any line beginning with any character of `#;/%` is considered to be a comment, and lines which can not be read as vectors/matrices are ignored. Example data `*.lau` files are given in directory `MCSTAS/data`. Powder files from ICSD/LAZY [42] and Fullprof [43] may also be used, but as the equivalent $\vec{q}$ vectors are superposed, not all Bragg spots will be simulated, and the intensity will be scaled by the multiplicity for each the spot.

8.4 Sans_spheres: A sample of hard spheres for small-angle scattering

Name:	Sans_spheres
Author:	(System); Lise Arleth, Veterinary University of Denmark
Input parameters	R, x_w, y_h, z_t, r, σ_a, ϕ, $\Delta\rho$, $R_{\rm det}$, d
Optional parameters	
Notes	

The component **Sans_spheres** models a sample of small independent spheres of radius R, which are uniformly distributed in a rectangular volume $x_w \times y_h \times z_t$ with a volume fraction ϕ. The absorption cross section density for the spheres (or is it from the solution?) is σ_a (in units of m^{-1}), specified for neutrons at 2200 m/s. Absorption and incoherent scattering from the medium is neglected. The difference in scattering length density (the contrast) between the hard spheres and the medium is called $\Delta\rho$. d denotes the distance to the (presumed circular) SANS detector of radius R.

8.4.1 Small-angle scattering cross section

The neutron intensity scattered into a solid angle $\Delta\Omega$ for a flat isotropic SANS sample in transmission geometry is given by [44]:

$$I_s(q) = \Psi \Delta\Omega T A z_{\rm max} \frac{d\sigma_v}{d\Omega}(q), \tag{8.33}$$

where Ψ is the neutron flux, T is the sample transmission, A is the illuminated sample area, and $z_{\rm max}$ the length of the neutron path through the sample.

In this component, we consider only scattering from a thin solution of monodisperse hard spheres of radius R, where the volume-specific scattering cross section is given by [44]

$$\frac{d\sigma_v}{d\Omega}(q) = n(\Delta\rho)^2 V^2 f(q), \tag{8.34}$$

where $f(q) = \left(3\frac{\sin(qR)-qR\cos(qR)}{(qR)^3}\right)^2$, n is the number density of spheres, and $V = 4/3\pi R^3$ is the sphere volume. (The density is thus $n = \phi/V$.)

Multiple scattering is ignored.

8.4.2 Algorithm

All neutrons, which hit the sample volume, are scattered. (Hence, no direct beam is simulated.) For scattered neutrons, the following steps are taken:

1. Choose a value of q uniformly in the interval $[0; q_{\rm max}]$.
2. Choose a polar angle, α, for the **q**-vector uniformly in $[0; \pi]$.
3. Scatter the neutron according to (q, α).
4. Calculate and apply the correct weight factor correction.

8.4.3 Calculating the weight factor

The scattering position is found by a Monte Carlo choice uniformly along the whole (unscattered) beam path with the sample, length l_{full}, giving $f_l = 1/l_{\mathrm{full}}$. The direction focusing on the detector gives (in an small angle approximation) $f_\Omega = d^2/(\pi R_{\mathrm{det}}^2)$.

Hence, the total weight tranformation factor becomes (more explanation to come)

$$\pi_j = l_{\mathrm{full}}(\pi R_{\mathrm{det}}^2/d^2)/(4\pi)n(\Delta\rho)^2V^2f(q)\exp(-\mu_a l), \tag{8.35}$$

where μ_a is the linear attenuation factor due to absorption and l is the total neutron path length within the sample.

8.5 Phonon_simple: A simple phonon sample

Name:	Phonon_simple
Author:	Kim Lefmann, Risø National Laboratory
Input parameters	r_o, h, r_{foc}, x_{target}, y_{target}, z_{target}, σ_{abs}, σ_{inc}, a, b, c, M, DW, T
Optional parameters	w_x, h_y, t_z, w_{focus}, h_{focus}, $w_{foc,angle}$, $h_{foc,angle}$, target_index
Notes	only validated qualitatively

This component models a simple phonon signal from a single crystal of a pure element in an *fcc* crystal structure. Only one isotropic acoustic phonon branch is modelled, and the longitudinal and transverse dispersions are identical with the velocity of sound being c. Other physical parameters are the atomic mass, M, the lattice parameter, a, the scattering length, b, the Debye-Waller factor, `DW`, and the temperature, T. Incoherent scattering and absorption are taken into account by the cross sections $\sigma_{\rm abs}$ and $\sigma_{\rm inc}$.

The sample can have the form of a cylinder with height h and radius r_0, or a box with dimensions w_x, h_y, t_z.

Phonons are emitted into a specific range of solid angles, specified by the location (x_t, y_t, z_t) and the focusing radius, r_0. Alternatively, the focusing is given by a rectangle, $w_{\rm focus}$ and $h_{\rm focus}$, and the focus point is given by the index of a down-stream component, `target_index`.

Multiple scattering is not included in this component.

8.5.1 The phonon cross section

The inelastic phonon cross section for a Bravais crystal of a pure element is given by Ref. [40, ch.3]

$$\begin{aligned} \frac{d^2\sigma'}{d\Omega dE_{\rm f}} &= b^2 \frac{k_{\rm f}}{k_{\rm i}} \frac{(2\pi)^3}{V_0} \frac{1}{2M} \exp(-2W) \\ &\times \sum_{\tau,q,p} \frac{(\boldsymbol{\kappa}\cdot\mathbf{e}_{q,p})^2}{\omega_{q,p}} \left\langle n_{q,p} + \frac{1}{2} \mp \frac{1}{2} \right\rangle \delta(\omega \pm \omega_{q,p}) \delta(\kappa \pm \mathbf{q} - \tau), \end{aligned} \tag{8.36}$$

where both annihilation and creation of one phonon is considered (represented by the plus and minus sign in the dispersion delta functions, respectively). In the equation, $\exp(-2W)$ is the Debye-Waller factor, `DW` and V_0 is the volume of the unit cell. The sum runs over the reciprocal lattice vectors, τ, over the polarisation index, p, and the N allowed wave vectors $\mathbf{q}$ within the Brillouin zone (where N is the number of unit cells in the crystal). Further, $\mathbf{e}_{q,p}$ is the polarization unit vectors, $\omega_{q,p}$ the phonon dispersion, and the Bose factor is $\langle n_{q,p} \rangle = (\hbar \exp(|\omega_{q,p}|/k_{\rm B}T) - 1)^{-1}$.

We have simplified this expression by assuming no polarization dependence of the dispersion, giving $\sum_p (\boldsymbol{\kappa}\cdot\mathbf{e}_{q,p})^2 = \kappa^2$. We assume that the inter-atomic interaction is nearest-neighbour-only so that the phonon dispersion becomes:

$$d_1(\mathbf{q}) = c_1/a\sqrt{z - s_q}, \tag{8.37}$$

where $z = 12$ is the number of nearest neighbours and $s_q = \sum_{\rm nn} \cos(\mathbf{q}\cdot\mathbf{r}_{\rm nn})$, where in turn $\mathbf{r}_{\rm nn}$ is the lattice positions of the nearest neighbours.

This dispersion relation may be modified with a small effort, since it is given as a separate c-function attatched to the component.

To calculate $d\sigma/d\Omega$ we need to transform the $\mathbf{q}$ sum into an integral over the Brillouin zone by $\sum_q \to NV_{\mathrm{c}}(2\pi)^{-3}\int_{\mathrm{BZ}} d^3\mathbf{q}$. The $\boldsymbol{\kappa}$ sum can now be removed by expanding the $\mathbf{q}$ integral to infinity. All in all, the partial differential cross section reads

$$\begin{aligned}\frac{d^2\sigma'}{d\Omega dE_{\mathrm{f}}}(\boldsymbol{\kappa},\omega) &= Nb^2\frac{k_{\mathrm{f}}}{k_{\mathrm{i}}}\frac{1}{2M}\int\frac{\hbar\kappa^2}{\hbar\omega_q}\left\langle n_q+\frac{1}{2}\mp\frac{1}{2}\right\rangle\delta(\omega\pm\omega_q)\delta(\boldsymbol{\kappa}\pm\mathbf{q})d^3\mathbf{q} \\ &= Nb^2\frac{k_{\mathrm{f}}}{k_{\mathrm{i}}}\frac{\hbar^2\kappa^2}{2M\hbar\omega_q}\left\langle n_\kappa+\frac{1}{2}\pm\frac{1}{2}\right\rangle\delta(\hbar\omega\pm d_1(\kappa)). \qquad (8.38)\end{aligned}$$

8.5.2 The algorithm

All neutrons, which hit the sample volume, are scattered into a particular range of solid angle, $\Delta\Omega$, like many other components. One of the difficult things in scattering from a dispersion is to take care to fulfill the dispersion criteria and to find the correct weight transformation.

In **Phonon_simple**, the following steps are taken:

1. If the sample is hit, calculate the total path length inside the sample, otherwise leave the neutron ray unchanged.
2. Choose a scattering point inside the sample
3. Choose a direction for the final wave vector, $\hat{\mathbf{k}}_{\mathrm{f}}$ within $\Delta\Omega$.
4. Calculate possible values of k_{f} so that the dispersion relation is fulfilled for the corresponding value of $\mathbf{k}_{\mathrm{f}}$. (There is always at least one possible k_{f} value [39].)
5. Choose one of the calculated k_{f} values.
6. Propagate the neutron to the scattering point and adjust the neutron velocity according to k_{f}.
7. Calculate and apply the correct weight factor correction, see below.

8.5.3 The weight transformation

Before making the weight transformation, we need to calculate the probability for scattering along one certain direction Ω from one phonon mode. To do this, we must integrate out the delta functions in the cross section (8.38). We here use that $\hbar\omega_q = \hbar^2(k_i^2 - k_f^2)/(2m_{\mathrm{N}})$, $\kappa = \mathbf{k}_{\mathrm{i}} - k_{\mathrm{f}}\hat{\mathbf{k}}_f$, and the integration rule $\int\delta(f(x)) = (df/dx)(0)^{-1}$. Now, we reach

$$\left(\frac{d\sigma'}{d\Omega}\right)_j = \int\frac{d^2\sigma'}{d\Omega dE_{\mathrm{f}}}dE_{\mathrm{f}} = Nb^2\frac{k_{\mathrm{f}}}{k_{\mathrm{i}}}\frac{\hbar^2\kappa^2}{2Md_1(\kappa_j)J(k_{\mathrm{f},j})}\left\langle n_\kappa+\frac{1}{2}\pm\frac{1}{2}\right\rangle. \qquad (8.39)$$

where the Jacobian reads

$$J = 1 - \frac{m_{\mathrm{N}}}{k_{\mathrm{f}}\hbar^2}\frac{\partial}{\partial k_{\mathrm{f}}}\left(d_1(\kappa)\right). \qquad (8.40)$$

A rough order-of-magnitude consideration gives $\frac{k_{\mathrm{f},j}}{k_{\mathrm{i}}} \approx 1$, $J \approx 1$, $\langle n_\kappa + \frac{1}{2} \pm \frac{1}{2} \rangle \approx 1$, $\frac{\hbar^2 \kappa^2}{2M d_1(\kappa)} \approx \frac{m}{M}$. Hence, $\left(\frac{d\sigma}{d\Omega}\right)_j \approx N b^2 \frac{m}{M}$, and the phonon cross section becomes a fraction of the total scattering cross section $4\pi N b^2$, as it must be. The differential cross section per unit volume is found from (8.39) by replacing N with $1/V_0$.

The total weight transformation now becomes

$$\pi_i = a_{\mathrm{lin}} l_{\mathrm{max}} n_{\mathrm{s}} \Delta\Omega b^2 \frac{k_{\mathrm{f},j}}{k_{\mathrm{i}}} \frac{\hbar^2 \kappa}{2 V_0 M d_1(\kappa) J(k_{\mathrm{f},j})} \left\langle n_\kappa + \frac{1}{2} \pm \frac{1}{2} \right\rangle , \tag{8.41}$$

where n_s is the number of possible dispersion values in the chosen direction.

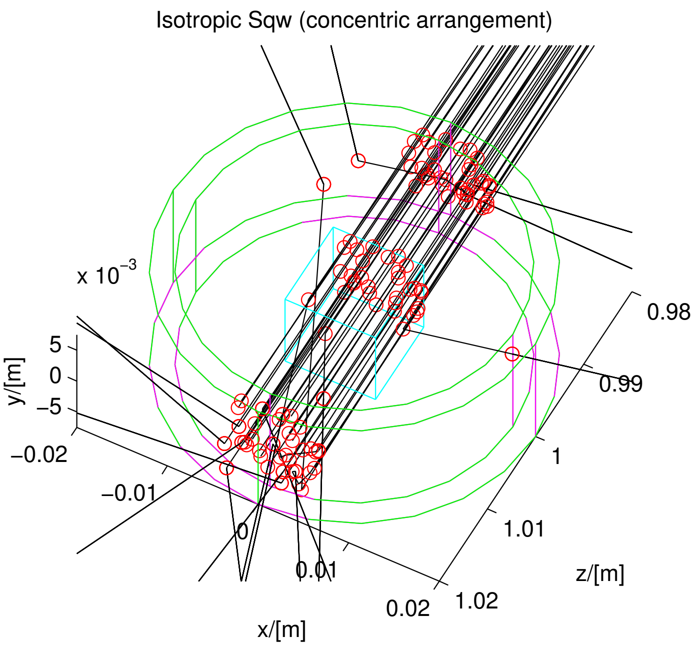


Figure 8.6: An $l-^4$He sample in a cryostat, simulated with the Isotropic_Sqw component in concentric geometry.

8.6 Isotropic_Sqw: A general $S(q, \omega)$ coherent and incoherent scatterer

Name:	Isotropic_Sqw
Author:	V. Hugouvieux, E. Farhi
Input parameters	Sqw_*coh*, σ_{coh}, Sqw_*inc*, σ_{inc}, V_ρ, σ_{abs}, T, x_{width}, y_{height}, z_{thich}, r_o, r_i, thickness
Optional parameters	$q_{min}, q_{max}, \omega_{min}, \omega_{max}, d\phi$, order
Notes	partly validated

The component assumes that the sample has the structure of an isotropic material. This stands for liquids, glasses (amorphous systems), polymers, gaz, and may be extended to powders. It simulates coherent and incoherent neutron scattering, and may be used to model isotropic samples, but also sample environments as concentric geometries are possible. The may input for the component is $S(q, \omega)$ tables, or powder structure files.

8.6.1 Neutron interaction with matter

When a neutron enters a material, according to usual models and letting the absorption aside to begin with, it 'sees' atoms as disks with a surface equal to the total scattering cross section of material σ. Each coherent and incoherent process is associated with a given probability to hit these cross-sections, according to σ_{coh} or σ_{inc}. We may choose randomly a scattering position along the path, using e.g. an expeonential decay probability. If the scattering condition is not satisfied, the neutron is transmitted, and leaves the sample. In any case, the absorption lowers the intensity according to an $e^{-\rho\sigma_{abs}d}$ absorption law along the propagation path d. In this process, the neutron is considered to be a particule.

Once the neutron 'knows' that something (terrible) is going to occur, it looks for a possible excitation to interact with. Then we turn to the wave description of the neutron, which interacts with the whole volume. The distribution of excitations, from which derives their relative intensity in the scattered beam, is simply the dynamic structre factor - or scattering law - $S(q,\omega)$. According to the definition of the density of states, we may use $g(\omega)$ as the probability law to scatter at a given energy transfer.

The neutron leaves the scattering point when a suitable (q,ω) choice has been found to satisfy the conservation laws. The process is iterated until the neutron leaves the volume of the material, eventually producing multiple scattering contributions.

8.6.2 Theoretical side

Pair correlation function $g(r)$ and Dynamic structure factor $S(q,\omega)$

Following Squires ([40], p63), the neutron differential scattering cross section for both coherent and incoherent processes is

$$\frac{d^2\sigma}{d\Omega dE_f} = \frac{\sigma}{4\pi}\frac{k_f}{k_i} N S(q,\omega) \tag{8.42}$$

with usual notations: $N = \rho V$ is the number of atoms in the scattering volume V with atomic number density ρ, E_f, E_i, k_f, k_i are the energy and wavevectors of final and initial states respectively, and σ is the total scattering cross-section. The unit of the dynamical structure factor $S(q,\omega)$ is an inverse energy. We define its norm

$$|S| = \iint S(q,\omega) dq d\omega. \tag{8.43}$$

Some easely measureable quantities in a liquid are the *static pair correlation function* $g(r)$ and the *structure factor* $S(q)$, defined as:

$$\begin{aligned}
\rho g(\vec{r}) &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq i}\langle\delta(\vec{r}+\vec{r}_i-\vec{r}_j)\rangle && (8.44)\\
S(\vec{q}) &= \int S(\vec{q},\omega) d\omega && (8.45)\\
&= 1+\rho\int_V [g(\vec{r})-1] e^{i\vec{q}.\vec{r}} d\vec{r} && (8.46)\\
&= 1+\rho\int_0^\infty [g(r)-1]\frac{\sin(qr)}{qr} 4\pi r^2 dr \text{ in isotropic materials.} && (8.47)
\end{aligned}$$

The latter expression, in isotropic materials, may be Fourier transformed as:

$$g(r) - 1 = \frac{1}{2\pi^2\rho}\int_0^\infty q^2 [S(q)-1]\frac{sin(qr)}{qr} dq \tag{8.48}$$

Both $g(r)$ and $S(q)$ converge to unity for large r and q values respectively, and they are representative of the atoms spatial distribution. In a liquid $\lim_{q\to 0} S(q) = \rho k_B T \chi_T$ where $\chi_T = (\frac{\partial \rho}{\partial P})_{V,T}$ is the compressibility [45]. In perfect gases, $S(q) = 1$ for all q. These quantities are obtained experimentaly from diffractometers.

The static pair correlation function $g(r)$ is the probability to find a neighbouring atom at a given distance (unitless). Since $g(0) = 0$, Eq. (8.48) provides a useful normalization sum-rule for $S(q)$:

$$\int_0^\infty q^2[S(q) - 1]dq = -2\pi^2\rho \tag{8.49}$$

This means that the integrated oscilations (around 1) of $S(q)$ are directly related to the density of the material ρ. In practice, the function $S(q)$ is often known on a restricted range $q \in [0, q_{max}]$, due to either limitations in the sample molecular dynamics simulation, or the measurement itself. In first approximation we consider that Eq. (8.49) can be applied in this range, i.e. we neglect the large q contributions. Then, for any non-normnalized data set S_{raw}, we may obtain the normalized $S = \alpha S_{raw}$ with

$$\alpha \simeq \frac{q_{max}^3/3 - 2\pi^2\rho}{\int_0^{q_{max}} q^2 S_{raw}(q)dq} \tag{8.50}$$

This normalization rule is only applied when parameter `norm_sqw=-1`.

We may measure, e.g. with time-of-flight instruments, the *density of states* $g_\omega(\omega)$ which is the fraction of modes whose energy lie between ω and $\omega + d\omega$ [46]

$$g_\omega(\omega) = \frac{\int S(q,\omega)dq}{|S|}. \tag{8.51}$$

This function is normalized to unity, $\int g_\omega(\omega)d\omega = 1$ and is a probability distribution of mode energies in the material.

Drawing probabilities from $S(q,\omega)$

The main idea to implement the scattering from $S(q,\omega)$ is to basically make two consecutive Monte Carlo choices, applying the well known *joint probability* theorem:

$$P(q \cap \omega) = P(\omega).P(q \mid \omega). \tag{8.52}$$

Thus we define $P(\omega)$ as the *cumulated distribution* (primitive) of the density of states $g_\omega(\omega)$:

$$P(\omega) = \int_0^\omega g_\omega(\omega')d\omega' \tag{8.53}$$

The function $P(\omega)$ is the probability for an excitation to have an energy lower than ω.

Similarly, we define the conditional probability $P(q \mid \omega)$ to be, for each energy lying between ω and $\omega + d\omega$:

$$\hat{g}_q(q \mid \omega) = \frac{S(q,\omega)}{\int S(q,\omega)dq} \tag{8.54}$$

$$P(q \mid \omega) = \int_0^q \hat{g}_q(q' \mid \omega)dq' \tag{8.55}$$

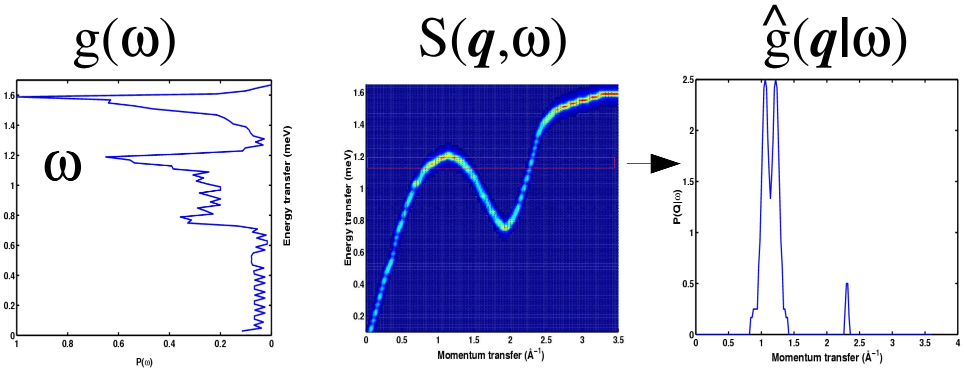


Figure 8.7: The probability functions extracted from $S(q, \omega)$. The energy transfer is first selected from the density of states, then the wavevector is obtained from $\hat{g}_q$.

The former $\hat{g}_q$ is the density of wavevector transfers for a selected energy transfer, and the latter $P(q \mid \omega)$ is the probability for an excitation to have a wavevector lower than q, for a given energy transfer ω.

8.6.3 The method

Choosing the interaction type

The method used is similar to the one adopted in the `Single_crystal` component (section 8.3).

We first compute the absorption and total cross-section

$$\sigma_{abs} = \sigma_{abs}^{2200} \frac{2200 m/s}{v} \tag{8.56}$$

$$\sigma_{tot} = \sigma_{abs} + \sigma_{coh} + \sigma_{inc} \tag{8.57}$$

as well as the neutron trajectory intersection with the geometry. This provides the total path length in the sample d_{out} to the exit. Defining the linear attenuation $\mu = \rho \sigma_{tot}$, the probability that the neutron scatters (or be absorbed) along path d_{out} is $e^{-\mu d_{out}}$. If this condition is not satisfied, the neutron leaves the sample unchanged. In the other case, we adjust the neutron weight by a factor $\frac{\sigma_{coh}+\sigma_{inc}}{\sigma_{tot}}$ to account for the portion of absorbed neutrons along the path. Additionally, we choose randomly the type of interaction with fractions σ_{coh} and σ_{inc}.

Choosing the interaction position

If the straight path to the sample volume exit is d_{out}, the probability that the neutron scatters before exiting the sample at a distance d_{scatt} is:

$$P(d_{scatt} < d_{out}) = \int_0^{d_{out}} \mu e^{-\mu x} dx = 1 - e^{-\mu d_{out}}. \tag{8.58}$$

From that law, we may compute the cumulated distribution, which gives the probability for scattering to occur at a distance lower than d_{scatt}, knowing that the neutron interacts before d_{out}. This law may be analytically inverted so that the path length d_{scatt} may be obtained directly from a uniform distribution random number ξ

$$d_{scatt} = -\frac{1}{\mu}\ln(1-\xi[1-e^{-\mu d_{out}}]). \tag{8.59}$$

Choosing the q and ω transfer

If no $S(q,\omega)$ data is available and the scattering process has been chosen as incoherent, we set $\omega = 0$ and select randomly an outgoing wave vector $\boldsymbol{k}_f$.

In case the $S(q,\omega)$ data is available for the scattering process (coherent or incoherent), a random choice is made to select the energy transfert using $P(\omega)$ with the $g(\omega)$ probability distribution (Eq. 8.53). Similarly, we use $P(q \mid \omega)$ to select a wavevector transfer (Eq. 8.54).

Chosing a (q,ω) set and applying Eq. (8.52), we have obtained a probabilistic nomalized evaluation of the dynamical structure factor, which we multiply by the norm $|S|$ to obtain $S(q,\omega)$:

$$S(q,\omega) = |S|.P(\omega).P(q \mid \omega). \tag{8.60}$$

Then a selection between energy gain and loss is done with the detailed balance ratio $e^{-\hbar\omega/k_BT}$.

Choosing the scattered wave vector

The next step is to check that conservation laws

$$\hbar\omega = E_i - E_f = \frac{\hbar^2}{2m}(k_i^2 - k_f^2) \tag{8.61}$$

$$\vec{q} = \vec{k}_i - \vec{k}_f \tag{8.62}$$

can be satisfied. These conditions are closely related to the method for selecting the outgoing wave vector direction.

When the final wave vector has to be computed, the quantities $\vec{k}_i$, $\hbar\omega$ and $q = |\vec{q}|$ are known. From the energy conservation law Eq. (8.62), we select ramdomly one of the two roots, k_f^+ and k_f^-. The scattered wave vector is noted : $\vec{k}_f = k_f\vec{\hat{k}}_s$ where $\vec{\hat{k}}_s$ is a unit vector.
Since we only know the norm of the scattering vector $\vec{q}$, the momentum conservation law Eq. (8.61) may be expressed as

$$q^2 = |\vec{k}_i - \vec{k}_f|^2 = |k_i^2 + k_f^2 - 2k_f\vec{k}_i \cdot \vec{\hat{k}}_s \tag{8.63}$$

where $\vec{k}_i \cdot \vec{\hat{k}}_s$ stands for the dot product of the vectors.
Now, we should solve :

$$\vec{k}_i \cdot \vec{\hat{k}}_s = \frac{1}{2k_f}(k_i^2 + k_f^2 - q^2) = C \tag{8.64}$$

$$|\vec{\hat{k}}_s| = 1 \tag{8.65}$$

where C is a constant. $\vec{\hat{k}}_s$ can be decomposed as : $\vec{\hat{k}}_s = B\vec{k}_i + \vec{u}_0$ where B is a constant and $\vec{u}_0$ is a vector of $Vect(\vec{k}_i)^{\perp}$ (that is the orthogonal of the space generated by $\vec{k}_i$), which is a plane P. Since we have : $\vec{k}_i \cdot \vec{\hat{k}}_s = C$, we may write : $B = \frac{C}{k_i^2}$. The vectors $\vec{u}_0$ such that

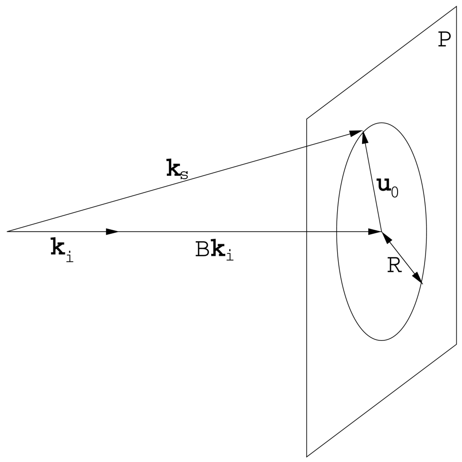


Figure 8.8: How to compute the outgoing wavevector direction $\vec{\hat{k}}_s$

$|\vec{\hat{k}}_s| = 1$ define a circle of radius R : $|\vec{u}_0| \;=\; R$. Since $\vec{u}_0$ and $B\vec{k}_i$ are orthogonal, we find :

$$\frac{C^2}{k_i^2} + R^2 = |\vec{\hat{k}}_s|^2 = 1 \tag{8.66}$$

from which we deduce the radius of the circle :

$$R = \sqrt{1 - \frac{C^2}{k_i^2}}. \tag{8.67}$$

Let us now define an orthonormal basis $(\vec{u}_1, \vec{u}_2)$ of the plane containing $\vec{u}_0$. $\vec{u}_0$ can be decomposed as : $\vec{u}_0 \;=\; R(\cos\theta\vec{u}_1 + \sin\theta\vec{u}_2)$, where θ can be randomly drawn for a uniform distribution. Finally, we obtain :

$$\vec{\hat{k}}_s = \frac{C^2}{k_i^2}\vec{k}_i + R(\cos\theta\vec{u}_1 + \sin\theta\vec{u}_2) \tag{8.68}$$

Extension to powder elastic scattering

In principle, the component can work in purtely elastic mode if only the $\omega = 0$ column is available in S. Anyway, in the diffractionists world, people do not usually define scattering with $S(q)$ (Eq. 8.45), but through the scattering vector $\boldsymbol{\tau}$, multiplicity $z(\tau)$ (for powders), and $|F^2|$ structure factors including Debye-Waller factors, as in Eq. 8.16.

When doing diffraction, and neglecting inelastic contribution as first approximation, we may integrate Eq. 8.42, keeping $k_i = k_f$.

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{coh.el.}}(|q|) = \int_0^\infty \frac{d^2\sigma_{coh}}{d\Omega dE_f} dE_f = \frac{N\sigma_{coh}}{4\pi} S_{coh}(q) \tag{8.69}$$

$$= N\frac{(2\pi)^3}{V_0}\sum_{\boldsymbol{\tau}} \delta(\boldsymbol{\tau} - \boldsymbol{q})|F_{\boldsymbol{\tau}}|^2 \text{ from Eq. (8.16)} \tag{8.70}$$

with $V_0 = 1/\rho$ being the volume of a lattice unit cell. Then we may write the formal equivalence:

$$S_{coh}(q) = \frac{2(2\pi)^4\rho}{\sigma_{coh}} \sum_{|\tau|} z(q)\delta(\tau - q)|F_\tau)|^2 \text{ in a powder.} \tag{8.71}$$

The normalization rule Eq. (8.50) is then applied.

Of course, the component PowderN (see section 8.2) can handle powder samples more efficiently (faster, better accuracy), but can not be used currently in concentric geometry, nor handle multiple scattering.

Important remarks and limitations

Since the choice of the interaction type, we know that the neutron *must* scatter, with an appropriate $\vec{k}_f$ outgoing wave vector. If any of the choices in the method fails:

1. the roots k_f^+ and k_f^- are imaginary, which means that conservation laws can not be satisfied and for instance the selected energy transfert is higher than the incoming neutron energy
2. the radius of the target circle R is imaginary

then a new (q,ω) set is drawn, and the process is iterated until sucess or - at last - removal of the neutron event. This latter absorption is then reported at the end of the simulation, as it never occurs in reality - neutrons that scatter do find a suitable (q,ω) set.

A way to avoid this is to focus the q,ω range from the $S(q,\omega)$ data file to the actual beam characteristics (matching instrument dynamical range). This is done e.g. by setting the component parameter *wmax* to the maximum energy of the incoming beam. In a few words, the $S(q,\omega)$ data set extension in Q and ω should preferably be smaller than the extension of the incoming beam and the scattered beam of interest for the experiment. Otherwise some Monte carlo choices lead to scattering in non useful regions of S, which reduces dramatically the algorithm efficiency, possibly removing neutrons by underestimating intensity. An automatic probability table optimization may be activated when setting parameter `auto_qw=1`. This is higly recommanded, but may sometimes miss a few neutrons if the q,ω beam range has been guessed too small.

Focusing the q and ω range, to the one being able to scatter the incoming beam, when using the component does improve significantly the speed of the computation. Additionally, if you restrict the scattering the first order only (parameter order=1), then you may specify the angular vertical extension $d\phi$ of the scattering area to gain optimized focusing. This option does not apply when handling multiple scattering (which emits in 4π many times before exiting the sample).

Parameter	type	meaning
Sqw_coh	string	Coherent scattering data file name. Use 0 or "" to disable
Sqw_inc	string	Incoherent scattering data file name. Use 0 or "" to scatter isotropically (Vanadium like)
sigma_coh	[barns]	Coherent scattering cross-section. 0 to disable
sigma_inc	[barns]	Incoherent scattering cross-section. 0 to disable
sigma_abs	[barns]	Absorption cross-section. 0 to disable
V_rho	[Å^{-3}]	atomic number density. May also be specified with molar weight *weight* in [g/mol] and material *density* in [g/cm^3]
T	[K]	Temperature. 0 disables detailed balance
xwidth	[m]	
yheight	[m]	dimensions of a box shaped geometry
zthick	[m]	
radius_o	[m]	dimensions of a cylinder shaped geometry
radius_i	[m]	sphere geometry if radius_i=0
thickness	[m]	thickness of hollow shape
norm_sqw	number	When positive, the value is used as the integral of S(q,w). Null value uses computed integral. Any negative value will normalize on the density (recommanded)
auto_qw	boolean	Automatically optimize probability tables during simulation (recommanded)
order	integer	Limit multiple scattering up to given order. 0 means no limitations
concentric	boolean	Enables to 'enter' inside concentric hollow geometries

Table 8.2: Main Isotropic_Sqw component parameters

8.6.4 The implementation

Geometry

The geometry for the component may be box, cylinder and sphere shaped, either filled or hollow. Relevant parameters for this purpose are as follow:

- **box**: dimensions are $x_{width} \times y_{height} \times z_{thick}$.
- **box, hollow**: *idem*, and the side wall thickness is set with *thickness*.
- **cylinder**: dimensions are r_o for the radius and y_{height} for the heigh.
- **cylinder, hollow**: *idem*, and hollow part is set with either r_i internal radius, or *thickness*.
- **sphere**: dimension is r_o for the radius.

- **sphere, hollow**: *idem*, and hollow part is set with either r_i internal radius, or *thickness*.

The AT position corresponds to the center of the sample.

Hollow shapes are particularly useful to model complex sample environments. Refer to section below for more details on this topic.

Dynamical structure factor

The material behaviour is specified through the total scattering cross-sections σ_{coh}, σ_{inc}, σ_{abs}, and the $S(q,\omega)$ data files.

If you are lucky enough to have access to separated coherent and incoherent contributions, simply set Sqw_coh and Sqw_inc paremeter to the files names. If on the other hand you have access to a global data set, use Sqw_coh, set the σ_{coh} parameter to the sum of both contributions $\sigma_{coh} + \sigma_{inc}$, and set $\sigma_{inc} = 0$. This way we only use one of the two implemented scattering channels. Such global data sets may originate from previous experiments, as far as you have applied all known corrections (multiple scattering, geometry, ...).

In any case, the accuracy of the $S(q,\omega)$ data limits the Q and ω resolution of the simulation. The sampling of data files should then be as thin as possible.

If the Sqw_inc parameter is left unset but the σ_{inc} is *not* zero, an isotropic incoherent elastic scattering is used, just like the V_sample component (see section 8.1).

Anyway, as explained below, it is also possible to only simulate the elastic scattering from a powder file (see below). Moreover, as there are two process channels (coherent and incoherent), it is possible to simulate a mixture of two powders, with fractions proportional to σ_{coh} and σ_{inc}.

Files formats: $S(q,\omega)$

The format of the data files is free text, consisting of three numerical block, separated by empty lines or comments, in the following order

1. A vector of length m containing wavevector q values, in Å^{-1}.
2. A vector of length n containing energy ω values, in meV.
3. A matrix of size m rows by n columns, of $S(q,\omega)$ values, in meV^{-1}.

Any line beginning with any character of `#;/%` is considered to be a comment, and lines which can not be read as vectors/matrices are ignored. Example files are listed as `*.sqw` files in directory `MCSTAS/data`.

The file header may optionally contain parameter settings for the material, as comments, with keywords as is the following example:

```
#V_0         0.07
#sigma_abs   5
#sigma_inc   4.8
#sigma_coh   1
#Temperature 10
```

Some `sqw` data files are included in the McStas distribution data directory, and they contain material parameter settings in their header, so that you may use:

```
Isotropic_Sqw(<geometry parameters>, Sqw_coh="He4.sqw", T=10)
```

Files formats: $S(q)$

This file format provides a mean to import directly an $S(q)$ data set, when setting parameters:

```
powder_format=qSq
```

The 'Sqw_coh' (or 'Sqw_inc') file should contains a single numerical block, which column assignment is defaulted as q and $S(q)$ being the first and second columns respectively. This may be overridden from the file header with '#column' keywords, as in the example:

```
#column_q  2
#column_Sq 1
```

Such files can only handle elastic scattering.

Files formats: powder structures (LAZY, Fullprof, Crystallographica)

Data files as used by the component PowderN may also be read. Data files of type `lau` and `laz` in the McStas distribution data directory are self-documented in their header. They do not need any additional parameters to be used, as in the example:

```
Isotropic_Sqw(<geometry parameters>, Sqw_coh="Al.laz")
```

Other column-based file formats may also be imported e.g. with parameters such as:

```
powder_format=Crystallographica
powder_format=Fullprof
powder_Dd    =0
powder_DW    =1
```

The last two parameters may as well be specified in the data file header with lines:

```
#Debye_Waller 1
#Delta_d/d    1e-3
```

The powder description is then translated into $S(q)$ by using Eq. (8.71). In this case, the density $\rho = n/V_0$ is the number of atoms in the inverse volume of the unit cell.

As the component builds an $S(q)$ from the powder structure description, the accuracy of the Isotropic_sqw component is limited by the binning during that conversion. This is usually enough to describe sample environments including powders (aluminium, copper, ...), but it is recommanded to rather use PowderN for faster and accurate powder diffraction, eventthough this latter does not implement multiple scattering.

Such files can only handle elastic scattering.

Concentric geometries, sample environment

The component has been designed in a way which enables to describe complex imbricated set-ups, i.e. what you need to simulate sample environments. To do so, one has first to use hollow shapes, they keep in mind that each surrounding geometry should be first declared before the central position (usially the sample) with the `concentric=1` parameter, but also duplicated (with an other instance name) at a symetric position with regards to the centre as in the example (shown in Fig. 8.7):

```
COMPONENT s_in=Isotropic_Sqw(
  thickness=0.001, radius_o=0.02, yheight=0.015,
  Sqw_coh="Al.laz", concentric=1)
AT (0,0,1) RELATIVE a

COMPONENT sample=Isotropic_Sqw(
  xwidth=0.01, yheight=0.01, zthick=0.01,
  Sqw_coh="He4_coh.sqw")
AT (0,0,1) RELATIVE a

COMPONENT s_out=Isotropic_Sqw(
  thickness=0.001, radius_o=0.02, yheight=0.015,
  Sqw_coh="Al.laz")
AT (0,0,1) RELATIVE a
```

Central component may be of any type, not specifically an Isotropic_sqw instance. It could be for instance a Single_crystal or a PowderN.. In principle, the number of surrounding shells is not restricted. The only restriction is that neutrons that scatter (in 4π) can not come back in the instrument description, so that some of the multiple scattering events are lost. namely, in the previous example, neutrons scattered by the outer wall of the cryostat `s_out` can not come back to the sample or to the other cryostat wall `s_in`. As these neutrons have usually few chances to reach the rest of the simulation, we expect that the approximation is fair.

Chapter 9

Monitors and detectors

In real neutron experiments, detectors and monitors play quite different roles. One wants the detectors to be as efficient as possible, counting all neutrons (absorbing them in the process), while the monitors measure the intensity of the incoming beam, and must as such be almost transparent, interacting only with (roughly) 0.1-1% of the neutrons passing by. In computer simulations, it is of course possible to detect every neutron ray without absorbing it or disturbing any of its parameters. Hence, the two components have very similar functions in the simulations, and we do not distinguish between them. For simplicity, they are from here on just called **monitors**.

Another important difference between computer simulations and real experiments is that one may allow the monitor to be sensitive to any neutron property, as *e.g.* direction, energy, and divergence, in addition to what is found in real-world detectors (space and time). One may, in fact, let the monitor record correlations between these properties, as seen for example in the divergence/position sensitive monitor in section 9.6.

When a monitor detects a neutron ray, a number counting variable is incremented: $n_i = n_{i-1} + 1$. In addition, the neutron weight p_i is added to the weight counting variable: $I_i = I_{i-1} + p_i$, and the second moment of the weight is updated: $M_{2,i} = M_{2,i-1} + p_i^2$. As also discussed chapter 2, after a simulation of N rays the detected intensity (in units of neutrons/sec.) is I_N, while the estimated errorbar is $\sqrt{M_{2,N}^2}$.

Many different monitor components have been developed for McStas, but we have decided to support only the most important ones. One example of the monitors we have omitted is the single monitor, **Monitor**, that measures just one number (with errorbars) per simulation. This effect is mirrored by any of the 1- or 2-dimensional components we support, e.g. the PSD_monitor. In case additional functionality of monitors is required, the few code lines of existing monitors can easily be modified.

However, the ultimate solution is the use of the "Swiss army knife" of monitors, **Monitor_nD**, that can face almost any simulation requirement, but will prove challenging for users who like to perform own modifications.

9.1 TOF_monitor: The time-of-flight monitor

Name:	TOF_monitor
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, n_{chan}, t_0, t_1, filename
Optional parameters	Δt
Notes	

The component **TOF_monitor** has a rectangular opening in the (x, y) plane, given by the x and y parameters, like for **Slit**. The neutron ray is propagated to the plane of the monitor by the kernel call PROP_Z0. A neutron ray is counted if it passes within the rectangular opening given by the x and y limits.

Special about **TOF_monitor** is that it is sensitive to the arrival time, t, of the neutron ray. Like in a real time-of-flight detector, the time dimension is binned into small time intervals. Hence this monitor maintains a one-dimensional histogram of counts. The n_{chan} time intervals begin at t_0 and end at t_1 (alternatively, the interval length is specified by Δt). As usual in time-of-flight analysis, all times are given in units of μs.

The output parameters from **TOF_monitor** are the three count numbers, N, I, and M_2 for the total counts in the monitor. In addition, a file, `filename`, is produced with a list of the same three data divided in different TOF bins. This file can be read and plotted by the MCplot tool; see the System Manual.

9.2 E_monitor: The energy-sensitive monitor

Name:	E_monitor
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, n_{chan}, E_{min}, E_{max}, filename
Optional parameters	
Notes	

The component **E_monitor** resembles **TOF_monitor** to a very large extent. Only this monitor is sensitive to the neutron energy, which in binned in *nchan* bins between E_{min} and E_{max} (in meV).

The output parameters from **E_monitor** are the total counts, and a file with 1-dimensional data vs. E, similar to **TOF_monitor**.

9.3 L_monitor: The wavelength sensitive monitor

Name:	L_monitor
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, n_{chan}, λ_{min}, λ_{max}, filename
Optional parameters	
Notes	

The component **L_monitor** is very similar to **TOF_monitor** and **E_monitor**. This component is just sensitive to the neutron wavelength. The wavelength spectrum is output in a one-dimensional histogram. between λ_{min} and λ_{max} (measured in Å).

As for the two other 1-dimensional monitors, this component outputs the total counts and a file with the histogram.

9.4 PSD_monitor: The PSD monitor

Name:	PSD_monitor
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, n_x, n_y, filename
Optional parameters	
Notes	

The component **PSD_monitor** resembles other monitors, e.g. **TOF_Monitor**, and also propagates the neutron ray to the detector surface in the (x, y)-plane, where the detector window is set by the x and y input coordinates. The PSD monitor, though, is not sensitive to the arrival time of the neutron ray, but rather to its position. The rectangular monitor window, given by the x and y limits is divided into $n_x \times n_y$ pixels.

The output from **PSD_monitor** is the integrated counts, n, I, M_2, as well as three two-dimensional arrays of counts: $n(x, y), I(x, y), M_2(x, y)$. The arrays are written to a file, `filename`, and can be read e.g. by the tool **MC_plot**, see the system manual.

9.5 Divergence_monitor: A divergence sensitive monitor

Name:	Divergence_monitor
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, n_{v}, n_{h}, $\eta_{\mathrm{v,max}}$, $\eta_{\mathrm{h,max}}$, filename
Optional parameters	
Notes	

The component **Divergence_monitor** is a two-dimensional monitor, which resembles **PSD_Monitor**. As for this component, the detector window is set by the x and y input coordinates. **Divergence_ monitor** is sensitive to the neutron divergence, defined by $\eta_{\mathrm{h}} = \tan^{-1}(v_x/v_z)$ and $\eta_{\mathrm{v}} = \tan^{-1}(v_y/v_z)$. The neutron counts are being histogrammed into $n_{\mathrm{v}} \times n_{\mathrm{h}}$ pixels. The divergence range accepted is in the vertical direction $[-\eta_{\mathrm{v,max}}; \eta_{\mathrm{v,max}}]$, and similar for the horizontal direction.

The output from **PSD_monitor** is the integrated counts, n, I, M_2, as well as three two-dimensional arrays of counts: $n(\eta_{\mathrm{v}}, \eta_{\mathrm{h}}), I(\eta_{\mathrm{v}}, \eta_{\mathrm{h}}), M_2(\eta_{\mathrm{v}}, \eta_{\mathrm{h}})$. The arrays are written to a file, `filename`, and can be read e.g. by the tool **MC_plot**, see the system manual.

9.6 DivPos_monitor: A divergence and position sensitive monitor

Name:	DivPos_monitor
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, n_x, n_{h}, $\eta_{\mathrm{h,max}}$, filename
Optional parameters	
Notes	

DivPos_monitor is a two-dimensional monitor component, which is sensitive to both horizontal position (x) and horizontal divergence defined by $\eta_{\mathrm{h}} = \tan^{-1}(v_x/v_z)$. The detector window is set by the x and y input coordinates.

The neutron counts are being histogrammed into $n_x \times n_{\mathrm{h}}$ pixels. The horizontal divergence range accepted is $[-\eta_{\mathrm{h,max}}; \eta_{\mathrm{h,max}}]$, and the horizontal position range is the size of the detector.

The output from **PSD_monitor** is the integrated counts, n, I, M_2, as well as three two-dimensional arrays of counts: $n(x, \eta_{\mathrm{h}}), I(x, \eta_{\mathrm{h}}), M_2(x, \eta_{\mathrm{h}})$. The arrays are written to a file and can be read e.g. by the tool **MC_plot**, see the system manual.

This component can be used for measuring acceptance diagrams [22]. **PSD_monitor** can easily be changed into being sensitive to y and vertical divergence by a 90 degree rotation around the z-axis.

9.7 Monitor_nD: A general Monitor for 0D/1D/2D records

Name:	Monitor_nD
Author:	System, E. Farhi
Input parameters	$x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, options
Optional parameters	$file$, $x_{width}, y_{height}, z_{thick}$, $bins$, min, max
Notes	

The component **Monitor_nD** is a general Monitor that may output any set of physical parameters regarding the passing neutrons. The generated files are either a set of 1D signals ([Intensity] *vs.* [Variable]), or a single 2D signal ([Intensity] *vs.* [Variable 1] *vs.* [Variable 1]), and possibly a simple long list of the selected physical parameters for each neutron.

The input parameters for **Monitor_nD** are its dimensions $x_{\min}, x_{\max}, y_{\min}$, $y_{\max}$ (in meters) and an *options* string describing what to detect, and what to do with the signals, in clear language. The $x_{width}, y_{height}, z_{thick}$ may also be used to enter dimensions.

Eventhough the possibilities of Monitor_nD are numerous, its usage remains as simple as possible, specially in the `options` parameter, which 'understands' normal language. The formatting of the *options* parameter is free, as long as it contains some specific keywords, that can be sometimes followed by values. The *no* or *not* option modifier will revert next option. The *all* option can also affect a set of monitor configuration parameters (see below).

As the usage of this component enables to monitor virtually anything, and thus the combinations of options and parameters is infinite, we shall only present the most basic configuration. The reader should refer to the on-line component help, using e.g. `mcdoc Monitor_nD.comp`.

9.7.1 The Monitor_nD geometry

The monitor shape can be selected among six geometries:

1. (*square*) The default geometry is flat rectangular in (xy) plane with dimensions $x_{\min}, x_{\max}, y_{\min}$, $y_{\max}$.

2. (*box*)A rectangular box with dimensions $x_{width}, y_{height}, z_{thick}$.

3. (*disk*) When choosing this geometry, the detector is a flat disk in (xy) plane. The radius is then
$$\text{radius} = \max(\text{abs }[x_{\min}, x_{\max}, y_{\min}, y_{\max}]). \tag{9.1}$$

4. (*sphere*) The detector is a sphere with the same radius as for the *disk* geometry.

5. (*cylinder*) The detector is a cylinder with revolution axis along y (vertical). The radius in (xz) plane is
$$\text{radius} = \max(\text{abs }[x_{\min}, x_{\max}]), \tag{9.2}$$
and the height along y is
$$\text{height} = |y_{\max} - y_{\max}|. \tag{9.3}$$

6. (*banana*) The same as the cylinder, but without the top/bottom caps, and on a restricted angular range. The angular range is specified using a `theta` variable limit specification in the `options`.

By default, the monitor is flat, rectangular. Of course, you can choose the orientation of the **Monitor_nD** in the instrument description file with the usual `ROTATED` modifier.

For the *box*, *sphere* and *cylinder*, the outgoing neutrons are monitored by default, but you can choose to monitor incoming neutron with the *incoming* option.

At last, the *slit* or *absorb* option will ask the component to absorb the neutrons that do not intersect the monitor. The *exclusive* option word removes neutrons which are similarly outside the monitor limits (that may be other than geometrical).

9.7.2 The neutron parameters that can be monitored

There are many different variables that can be monitored at the same time and position. Some can have more than one name (e.g. `energy` or `omega`).

```
kx ky kz k wavevector [Angs-1] (    usually axis are
vx vy vz v            [m/s]         x=horz., y=vert., z=on axis)
x y z                 [m]      Distance, Position
kxy vxy xy radius     [m]      Radial wavevector, velocity and position
t time                [s]      Time of Flight
energy omega          [meV]
lambda wavelength     [Angs]
p intensity flux      [n/s] or [n/cm^2/s]
ncounts               [1]
sx sy sz              [1]      Spin
vdiv ydiv dy          [deg]    vertical divergence (y)
hdiv divergence xdiv  [deg]    horizontal divergence (x)
angle                 [deg]    divergence from  direction
theta longitude       [deg]    longitude (x/z) [for sphere and cylinder]
phi   lattitude       [deg]    lattitude (y/z) [for sphere and cylinder]
```

as well as two other special variables

```
user user1            will monitor the [Mon_Name]_Vars.UserVariable{1|2}
user2                 to be assigned in an other component (see below)
```

To tell the component what you want to monitor, just add the variable names in the *options* parameter. The data will be sorted into *bins* cells (default is 20), between some default *limits*, that can also be set by user. The *auto* option will automatically determine what limits should be used to have a good sampling of signals.

9.7.3 Important options

Each monitoring records the flux (sum of weights p) versus the given variables, except if the `signal=<variable>` word is used in the `options`. The *cm2* option will ask to normalize the flux to the monitor section surface, and the `capture` option uses the gold foil integrated 'capture' flux weightening:

$$\Phi_c = \int \frac{d\Phi}{d\lambda} \frac{\lambda}{\lambda_{2200m/s}} d\lambda \tag{9.4}$$

The `auto` option is probably the most useful one: it asks the monitor to determine automatically the best limits for each variable, in order to obtain the most significant monitored histogram. This option should preceed each variable, or be located after all variables in which case they are all affected. On the other hand, one may manually set the limits with the `limits=[min max]` option.

The `log` and `abs` options should be positioned before each variable to specify logarithmic binning and absolute value respectively.

The *borders* option will monitor variables that are outside the limits. These values are then accumulated on the 'borders' of the signal.

9.7.4 The output files

By default, the file names will be the component name, followed by automatic extensions showing what was monitored (such as `MyMonitor.x`). You can also set the filename in *options* with the *file* keyword followed by the file name that you want. The extension will then be added if the name does not contain a dot (.). Finally, the $filename$ parameter may also be used.

The output files format are standard 1D or 2D McStas detector files. The *no file* option will *unactivate* monitor, and make it a single 0D monitor detecting integrated flux and counts. The *verbose* option will display the nature of the monitor, and the names of the generated files.

The 2D output

When you ask the **Monitor_nD** to monitor only two variables (e.g. *options* = "x y"), a single 2D file of intensity versus these two correlated variables will be created.

The 1D output

The **Monitor_nD** can produce a set of 1D files, one for each monitored variable, when using 1 or more than 2 variables, or when specifying the *multiple* keyword option.

The List output

The **Monitor_nD** can additionally produce a *list* of variable values for neutrons that pass into the monitor. This feature is additive to the 1D or 2D output. By default only 1000 events will be recorded in the file, but you can specify for instance "*list* 3000 neutrons" or "*list all* neutrons". This last option might require a lot of memory and generate huge files.

9.7.5 Monitor equivalences

In the following table, we show how the Monitor_nD may substitute any other McStas monitor.

9.7.6 Usage examples

- ```
  COMPONENT MyMonitor = Monitor_nD(
      xmin = -0.1, xmax = 0.1,
      ymin = -0.1, ymax = 0.1,
      options = "energy auto limits")
  ```
  will monitor the neutron energy in a single 1D file (a kind of E_monitor)

- `options = "banana, theta limits=[10,130], bins=120, y bins=30"`
  is a theta/height banana detector.

- `options = "banana, theta limits=[10,130], auto time"`
  is a theta/time-of-flight banana detector.

- `options="x bins=30 limits=[-0.05 0.05] ; y"`
  will set the monitor to look at x and y. For y, default bins (20) and limits values (monitor dimensions) are used.

- `options="x y, auto, all bins=30"`
  will determine itself the required limits for x and y.

- `options="multiple x bins=30, y limits=[-0.05 0.05], all auto"`
  will monitor the neutron x and y in two 1D files.

- `options="x y z kx ky kz, all auto"`
  will monitor each of theses variables in six 1D files.

- `options="x y z kx ky kz, list all, all auto"`
  will monitor all theses neutron variables in one long list, one row per neutron event.

- `options="multiple x y z kx ky kz, and list 2000, all auto"`
  will monitor all theses neutron variables in one list of 2000 events and in six 1D files.

- `options="signal=energy, x y"`
  is a PSD monitor recording the mean energy of the beam as a function of x and y.

9.7.7 Monitoring user variables

There is a simple way to monitor any quantity with Monitor_nD. This may be e.g. the number of neutron bounces in a guide, or the wavevector and energy transfert at a sample. The only requirement is to define the `user1` (and optionally `user2`) variables of a given Monitor_nD instance. This is performed in a 4 steps process:

- Declare that you intend to monitor user variables in a Monitor_nD instance (defined in TRACE):
  ```

McStas monitor	Monitor_nD equivalent
Divergence_monitor	*options*=”dx bins=*ndiv* limits=[$-\alpha/2 \alpha/2$], lambda bins=*nlam* limits=[λ_0 λ_1] file=$file$”
DivLambda_monitor	*options*=”dx bins=*nh* limits=[$-h_{max}/2 h_{max}/2$], dy bins=*nv* limits=[$-v_{max}/2 v_{max}/2$]” *filename*=$file$
DivPos_monitor	*options*=”dx bins=*ndiv* limits=[$-\alpha/2 \alpha/2$], x bins=*npos*” *xmin*=x_{min} *xmax*=x_{max}
E_monitor	*options*=”energy bins=*nchan* limits=[$E_{min} E_{max}$]”
EPSD_monitor	*options*=”energy bins=n_E limits=[$E_{min} E_{max}$], x bins=*nx*” *xmin*=x_{min} *xmax*=x_{max}
Hdiv_monitor	*options*=”dx bins=*nh* limits=[$-h_{max}/2 h_{max}/2$]” *filename*=$file$
L_monitor	*options*=”lambda bins=*nh* limits=[$-\lambda_{max}/2 \lambda_{max}/2$]” *filename*=$file$
Monitor_4PI	*options*=”sphere”
Monitor	*options*=”unactivate”
PSDcyl_monitor	*options*=”theta bins=*nr*,y bins=*ny*, cylinder” *filename*=$file$ *yheight*=*height* *xwidth*=2*radius
PSDlin_monitor	*options*=”x bins=*nx*” *xmin*=x_{min} *xmax*=x_{max} *ymin*=y_{min} *ymax*=y_{max} *filename*=$file$
PSD_monitor_4PI	*options*=”x y, sphere”
PSD_monitor	*options*=”x bins=*nx*, y bins=*ny*” *xmin*=x_{min} *xmax*=x_{max} *ymin*=y_{min} *ymax*=y_{max} *filename*=$file$
TOF_cylPSD_monitor	*options*=”theta bins=n_ϕ, time bins=*nt* limits=[t_0, t_1], cylinder” *filename*=$file$ *yheight*=*height* *xwidth*=2*radius
TOFLambda_monitor	*options*=”lambda bins=n_λ limits=[λ_0 λ_1], time bins=*nt* limits=[t_0, t_1]” *filename*=$file$
TOFlog_mon	*options*=”log time bins=*nt* limits=[t_0, t_1]”
TOF_monitor	*options*=”time bins=*nt* limits=[t_0, t_1]”

Table 9.1: Using Monitor_nD in place of other components. All limits specifications may be advantageously replaced by an *auto* word preceeding each monitored variable. Not all file and dimension specifications are indicated (e.g. filename, xmin, xmax, ymin, ymax).

```
DECLARE
%{ (...)
  %include "monitor_nd-lib"
  MONND_DECLARE(UserMonitor); // will monitor custom things in UserMonitor
%}
```

- Initialize the label of the user variable (optional):

```
INITIALIZE
%{
  (...)
  MONND_USER_TITLE(UserMonitor, 1, "Log(time)");
%}
```

  The value '1' could be '2' for the `user2` variable.

- Set the user variable value in a TRACE component EXTEND block:

```
TRACE
(...)
COMPONENT blah = blah_comp(...)
EXTEND
%{  // attach a value to user1 in UserMonitor
  MONND_USER_VALUE(UserMonitor, 1, log(t));
%}
(...)
```

- Tell the Monitor_nD instance to record user variables:

```
TRACE
(...)
COMPONENT UserMonitor = Monitor_nD(options="auto user1")
(...)
```

Setting the user variable values may either make use of the neutron parameters (x,y,z, vx,vy,vz, t, sx,sy,sz, p), access the internal variables of the component that sets the user variables (in this example, those from the `blah` instance), and access any component OUTPUT parameter using the `MC_GETPAR` C macro(see chapter A).

In the following example, we show how the number of bounces in a polygonal guide may be monitored. Let us have a guide made of many Guide_gravity instances. We declare a global simulation variable `nbounces`, set it to 0 for each neutron entering the guide, and sum-up all bounces from each section, accessing the `Gvars` OUTPUT variable of component Guide_gravity. Then we ask Monitor_nD to look at that value.

```
DECLARE
%{
  double nbounces;
```

```
%}
INITIALIZE
%{
  MONND_USER_TITLE(UserMonitor, 1, "Number of bounces");
%}
TRACE
(...)
COMPONENT Guide_in = Arm() AT (...)
EXTEND
%{
  nbouces = 0;
%}

COMPONENT Guide1 = Guide_gravity(...) AT (...) RELATIVE PREVIOUS
EXTEND
%{
  if (SCATTERED) nbounces += GVars.N_reflection[0];
%}
(... many guide instances, copy/paste and change names ...)
COMPONENT GuideN = Guide_gravity(...) AT (...) RELATIVE PREVIOUS
EXTEND
%{
  if (SCATTERED) nbounces += GVars.N_reflection[0];
%}

COMPONENT Guide_out = Arm() AT (...) RELATIVE PREVIOUS
EXTEND
%{ // set value to monitor
  MONND_USER_VALUE(UserMonitor, 1, nbounces);
%}

// monitor nbounces
COMPONENT UserMonitor = Monitor_nD(options="auto user1") AT (...)
(...)
```

9.7.8 Monitoring neutron parameter correlations, PreMonitor_nD

The first imediate usage of the Monitor_nD component is when one requires to identify cross-correlations between some neutron parameters, e.g. position and divergence (*aka* phase-space diagram). This latter monitor would be merely obtained with:

```
options="x dx, auto", bins=30
```

This example records the correlation between position and divergence of neutrons at a given instrument location.

Name:	PreMonitor_nD
Author:	System, E. Farhi
Input parameters	comp
Optional parameters	
Notes	

But it is also possible to search for cross-correlation between two part of the instrument simulation. One example is the acceptance phase-diagram, which shows the neutron caracteristics at the input required to reach the end of the simulation. This *spatial* correlation may be revealed using the **PreMonitor_nD** component. This latter stores the neutron parameters at a given instrument location, to be used at an other Monitor_nD location for monitoring.

The only parameter of **PreMonitor_nD** is the name of the associated Monitor_nD instance, which should use the `premonitor` option, as in the following example:

```
COMPONENT CorrelationLocation = PreMonitor_nD(comp = CorrelationMonitor)
AT (...)

  (... e.g. a guide system )

COMPONENT CorrelationMonitor  = Monitor_nD(
   options="x dx, auto, all bins=30, premonitor")
AT (...)
```

which performs the same monitoring as the previous example, but with a spatial correlation constrain. Indeed, it records the position *vs* the divergence of neutrons at the correlation location, but only if they reach the monitoring position. All usual Monitor_nD variables may be used, except the user variables. These latter may be defined as described in section 9.7.7 in an EXTEND block.

Chapter 10

Special-purpose components

The chapter deals with components that are not easily included in any of the other chapters because of their special nature, but which are still part of the McStas system.

One part of these components deals with splitting simulations into two (or more) stages. For example, a guide system is often not changed much, and a long simulation of neutron rays "surviving" through the guide system could be reused for several simulations of the instrument back-end, speeding up the simulations by (typically) one or two orders of magnitude. The components for doing this trick is **Virtual_input** and **Virtual_output**, which stores and reads neutron rays, respectively.

Other components perform the simulation of the instrument resolution functions. These are **Res_sample** and **TOF_Res_sample**, which are to be placed at the sample position, and **Res_monitor**, that should be localized at the position of the instrument detector.

Progress_bar is a simulation utility that displays the simulation status, but assumes the form of a component.

10.1 Vitual_output: Saving the first part of a split simulation

Name:	Virtual_output
Author:	System
Input parameters	filename
Optional parameters	buffer-size, type
Notes	

The component **Virtual_output** stores the neutron ray parameters at the end of the first part of a split simulation. The idea is to let the next part of the split simulation be performed by another instrument file, which reads the stored neutron ray parameters by the component **Virtual_input**.

All neutron ray parameters are saved to the output file, which is by default of “text” type, but can also assume the binary formats “float” or “double”. The storing of neutron rays continue until the specified number of simulations have been performed.

`buffer-size` may be used to limit the size of the output file, but absolute intentities are then likely to be wrong. We recommend to use the default value of zero, saving all neutron rays. The size of the file is then controlled indirectly with the general *ncounts* parameter.

10.2 Vitual_input: Starting the second part of a split simulation

Name:	Virtual_input
Author:	System
Input parameters	filename
Optional parameters	repeat-count, type
Notes	

The component **Virtual_input** resumes a split simulation where the first part has been performed by another instrument and the neutron ray parameters have been stored by the component **Virtual_output**.

All neutron ray parameters are read from the input file, which is by default of “text” type, but can also assume the binary formats “float” and “double”. The reading of neutron rays continue until the specified number of rays have been simulated or till the file has been exhausted. If desirable, the input file can be reused a number of times, determined by the optional parameter “repeat-count”. This is only useful if the present simulation makes use of MC choices, otherwise the same outcome will result for each repetition of the simulation.

Care should be taken when dealing with absolute intensities, which will be correct only when the input file has been exhausted at least once.

The simulation ends with either the end of the repeated file counts, or with the normal end with *ncount* McStas simulation events. We recommand to control the simulation on `repeat-count` by using a very large ncount value, e.g. $ncount = 1e10$.

10.3 Res_sample: A sample-like component for resolution calculation

Name:	Res_sample
Author:	(System); Alan Tennant, HMI
Input parameters	$r_{\rm i}$, $r_{\rm o}$, h, $r_{\rm focus}$, $x_{\rm target}$, $y_{\rm target}$, $z_{\rm target}$, E_0, ΔE
Optional parameters	x_w, y_h, z_t, $x_{\rm focus}$, $y_{\rm focus}$, $a_{\rm v,focus}$, $a_{\rm h,focus}$, target index
Notes	

The component **Res_sample** scatters neutron rays isotropically in direction and uniformly in energy. Regardless of the state of the incoming neutron ray, all directions and energies for the scattered ray have the same probability, within specified intervals.

The component is meant for computation of the resolution function, but may also be used for test and debugging purposes. For actual calculations of the resolution function, **Res_sample** should be used together with **Res_monitor**, described in section 10.5.

The shape of Res_sample is either a hollow cylinder or a rectangular box. The hollow cylinder shape is specified with the inner and outer radius, $r_{\rm i}$ and $r_{\rm o}$, respectively, and the height, h. If these parameters are unspecified, the shape is instead a box of dimensions x_w, y_h, and z_t. The component only propagates neutron rays that are scattered; other rays are absorbed. The scattering probability is proportional to the neutron flight path length inside the sample, to make a true volume weighting of the sample. The reason for this is that the resolution function of an instrument is independent of any sample properties such as scattering and absorbtion cross sections but will in general depend on sample size and shape.

The point of scattering inside the sample is chosen uniformly along the neutron flight path inside the sample, and the scattered neutron ray is given a random energy and direction. This energy is selected in the interval $[E_0 - \Delta E; E_0 + \Delta E]$ which hence must be chosen large enough to cover all interesting neutron energies. Similarly, the scattered direction is chosen in a user-specified range, either within a sphere of radius $r_{\rm foc}$, within a rectangular target with measures $(x_{\rm focus}, y_{\rm focus})$ or in the specified angular range. This target is positioned at the x_{target}, y_{target}, z_{target} point in space, or using the target_index for wich e.g. 1 is the further component, -1 is the previous, etc...

A special feature, used when computing resolution functions, is that the component stores complete information about the scattering event in the output parameter *res_struct*. The information includes initial and final wave vectors, the coordinates of the scattering point, and the neutron weight after the scattering event. From this information the scattering parameters $(\mathbf{Q}, \omega)$ can be recorded for every scattering event and used to compute the resolution function. For an example of using the information in the output parameter, see the description of the **Res_monitor** component in section 10.5.

10.4 TOF_Res_sample: A sample-like component for TOF resolution calculation

Name:	TOF_Res_sample
Author:	System
Input parameters	r_i, r_o, h, r_focus, x_target, y_target, z_target, t_0, Δt
Optional parameters	x_w, y_h, z_t, x_focus, y_focus, $a_\mathrm{v,focus}$, $a_\mathrm{h,focus}$, target index
Notes	

The component **TOF_Res_sample** scatters neutron rays isotropically in position within a specified angular range. As for **Res_sample**, this component is meant for computation of the resolution function, but in this case for one time bin in a time-of-flight (TOF) instrument. The component selects uniformly the neutron energy so that neutron arrival time at the TOF detector lies within one time bin, specified by t_0 and Δt. For actual calculations of the resolution function, **TOF_Res_sample** should be used together with **Res_monitor**, described in section 10.5.

The shape of **TOF_Res_sample** is either a hollow cylinder or a rectangular box. The hollow cylinder shape is specified with the inner and outer radius, r_i and r_o, respectively, and the height, h. If these parameters are unspecified, the shape is instead a box of dimensions x_w, y_h, and z_t.

The component only propagates neutron rays that are scattered; other rays are absorbed. As for **Res_sample**, the scattering probability is proportional to the neutron flight path length inside the sample. The point of scattering in the sample is chosen uniformly along the neutron flight path inside the sample, and the scattered direction is chosen in a user-specified range, either within a sphere of radius r_foc, within a rectangular target with measures ($x_\mathrm{focus}, y_\mathrm{focus}$) or in the specified angular range. This target is positioned at the x_{target}, y_{target}, z_{target} point in space, or using target_index.

This component stores complete information about the scattering event in the output parameter *res_struct*, see **Res_Sample**.

10.5 Res_monitor: The monitor for resolution calculation

Name:	Res_monitor
Author:	(System); Alan Tennant, HMI
Input parameters	x_min, x_max, y_min, y_max, filename, res_sample, buffer size
Optional parameters	x_w, y_h, z_t, options
Notes	

The component **Res_monitor** is used for calculating the resolution function of a particular instrument with detector of the given shape, size, and position. The shape of **Res_monitor** is by default rectangular, but can be a box, a sphere, a disk, or a cylinder, depending on the parameter "options". The component works like a normal monitor, but also records all scattering events and stores them to a file that can later be read by the McStas frontend tool `mcresplot`.

For time-of-flight (TOF) instruments, Res_monitor should be understood as giving the resolution of one time bin of the TOF-detector only; the bin properties being specified in the preceding **TOF_Res_sample**.

As described in section 10.3, the **Res_monitor** should be used in connection with one of the components **Res_sample** or **TOF_Res_sample**, the name of which should be passed as an input parameter to **Res_monitor**. For example

```
COMPONENT mysample = Res_sample( ... )
...
COMPONENT det = Res_monitor(res_sample_comp = mysample, ...)
...
```

The output file is in ASCII format, one line per scattering event, with the following columns:

- $\mathbf{k}_\mathrm{i}$, the three components of the initial wave vector.
- $\mathbf{k}_\mathrm{f}$, the three components of the final wave vector.
- $\mathbf{r}$, the three components of the position of the scattering event in the sample.
- p_i, the neutron weight just after the scattering event.
- p_f, the relative neutron weight adjustment from sample to detector (so the total weight in the detector is $p_\mathrm{i}p_\mathrm{f}$).

From $\mathbf{k}_\mathrm{i}$ and $\mathbf{k}_\mathrm{f}$, we may compute the scattering parameters $\kappa = \mathbf{k}_\mathrm{i} - \mathbf{k}_\mathrm{f}$ and $\hbar\omega = \hbar^2/(2m_\mathrm{n})(\mathbf{k}_\mathrm{i}^2 - \mathbf{k}_\mathrm{f}^2)$. The vectors are given in the local coordinate system of the resolution sample component. The wave vectors are in units of Å^{-1}, the energy transfer in meV.

The output parameters from **Res_monitor** are the three count numbers, *Nsum*, *psum*, and *p2sum*, and the handle *file* of the output file.

10.6 Progress_bar: Simulation progress and automatic saving

Name:	Progress_bar
Author:	System
Input parameters	percent, flag_save
Optional parameters	
Notes	

This component displays the simulation progress and status but does not affect the neutron parameters. The display is updated in regular intervals of the full simulation; the default step size is 10 %, but it may be changed using the `percent` parameter (from 0 to 100). The estimated computation time is displayed at the begining and actual simulation time is shown at the end.

Additionally, setting the `flag_save` to 1 results in a regular save of the data files during the simulation. This means that is is possible to view the data before the end of the computation, and have also a trace of it in case of computer crash. The achieved percentage of the simulation is stored in these temporary data files. Technically, this save is equivalent to sending regularly a USR2 signal to the running simulation.

10.7 Beam_spy: A beam analyzer

Name:	Beam_spy
Author:	System
Input parameters	
Optional parameters	
Notes	should overlap previous component

This component is at the same time an Arm and a simple Monitor. It analyzes all neutrons reaching it, and computes statistics for the beam, as well as the intensity.

This component does not affect the neutron beam, and does not contain any propagation call. Thus it gets neutrons from the previous component in the instrument description, and should better be placed at the same position, with `AT (0,0,0) RELATIVE PREVIOUS`.

Appendix A

Libraries and conversion constants

The McStas Library contains a number of built-in functions and conversion constants which are useful when constructing components. These are stored in the `share` directory of the `MCSTAS` library.

Within these functions, the 'Run-time' part is available for all component/instrument descriptions. The other parts are dynamic, that is they are not pre-loaded, but only imported once when a component requests it using the `%include` McStas keyword. For instance, within a component C code block, (usually SHARE or DECLARE):

```
%include "read_table-lib"
```

will include the 'read_table-lib.h' file, and the 'read_table-lib.c' (unless the `--no-runtime` option is used with `mcstas`). Similarly,

```
%include "read_table-lib.h"
```

will *only* include the 'read_table-lib.h'. The library embedding is done only once for all components (like the SHARE section). For an example of implementation, see **Res_monitor**.

In this Appendix, we present a short list of both each of the library contents and the run-time features.

A.1 Run-time calls and functions (`mcstas-r`)

Here we list a number of preprogrammed macros which may ease the task of writing component and instrument definitions.

A.1.1 Neutron propagation

Propagation routines perform all necessary operations to transport neutron rays from one point to an other. Except when using the special `ALLOW_BACKPROP;` call prior to exectuting any `PROP_*` propagation, the neutron rays which have negative propagation times are removed automatically.

- **ABSORB**. This macro issues an order to the overall McStas simulator to interrupt the simulation of the current neutron history and to start a new one.

- **PROP_Z0**. Propagates the neutron to the $z = 0$ plane, by adjusting (x, y, z) and t accordingly from knowledge of the neutron velocity (vx, vy, vz). If the propagation time is negative, the neutron ray is absorbed.

  For components that are centered along the z-axis, use the `_intersect` functions to determine intersection time(s), and then a `PROP_DT` call.

- **PROP_DT**(dt). Propagates the neutron through the time interval dt, adjusting (x, y, z) and t accordingly from knowledge of the neutron velocity.

- **PROP_GRAV_DT**(dt, Ax, Ay, Az). Like **PROP_DT**, but it also includes gravity using the acceleration (Ax, Ay, Az). In addition to adjusting (x, y, z) and t, also (vx, vy, vz) is modified.

- **ALLOW_BACKPROP**. Indicates that the next propagation routine will not remove the neutron ray, even if negative propagation times are found. Further propagations are not affected.

- **SCATTER**. This macro is used to denote a scattering event inside a component. It should be used e.g to indicate that a component has interacted with the neutron ray (e.g. scattered or detected). This does not affect the simulation (see, however, **Beamstop**), and it is mainly used by the `MCDISPLAY` section and the `GROUP` modifier See also the SCATTERED variable (below).

A.1.2 Coordinate and component variable retrieval

- **MC_GETPAR**(). This may be used in the FINALLY section of an instrument definition to reference the output parameters of a component.

- **NAME_CURRENT_COMP** gives the name of the current component as a string.

- **POS_A_CURRENT_COMP** gives the absolute position of the current component. A component of the vector is referred to as POS_A_CURRENT_COMP.i where i is x, y or z.

- **ROT_A_CURRENT_COMP** and **ROT_R_CURRENT_COMP** give the orientation of the current component as rotation matrices (absolute orientation and the orientation relative to the previous component, respectively). A component of a rotation matrix is referred to as ROT_A_CURRENT_COMP$[m][n]$, where m and n are 0, 1, or 2 standing for x, y and z coordinates respectively.

- **POS_A_COMP**(*comp*) gives the absolute position of the component with the name *comp*. Note that *comp* is not given as a string. A component of the vector is referred to as POS_A_COMP(*comp*).i where i is x, y or z.

- **ROT_A_COMP**(*comp*) and **ROT_R_COMP**(*comp*) give the orientation of the component *comp* as rotation matrices (absolute orientation and the orientation relative to its previous component, respectively). Note that *comp* is not given as a string. A component of a rotation matrice is referred to as ROT_A_COMP(*comp*)$[m][n]$, where m and n are 0, 1, or 2.

- **INDEX_CURRENT_COMP** is the number (index) of the current component (starting from 1).

- **POS_A_COMP_INDEX**(*index*) is the absolute position of component *index*. POS_A_COMP_INDEX (INDEX_CURRENT_COMP) is the same as POS_A_CURRENT_COMP. You may use POS_A_COMP_INDEX (INDEX_CURRENT_COMP+1) to make, for instance, your component access the position of the next component (this is usefull for automatic targeting). A component of the vector is referred to as POS_A_COMP_INDEX(*index*).*i* where i is x, y or z.

- **POS_R_COMP_INDEX** works the same as above, but with relative coordinates.

- **STORE_NEUTRON**($index, x, y, z, vx, vy, vz, t, sx, sy, sz, p$) stores the current neutron state in the trace-history table, in local coordinate system. *index* is usually INDEX_CURRENT_COMP. This is automatically done when entering each component of an instrument.

- **RESTORE_NEUTRON**($index, x, y, z, vx, vy, vz, t, sx, sy, sz, p$) restores the neutron state to the one at the input of the component *index*. To ignore a component effect, use RESTORE_NEUTRON (INDEX_CURRENT_COMP, $x, y, z, vx, vy, vz, t, sx, sy, sz, p$) at the end of its TRACE section, or in its EXTEND section. These neutron states are in the local component coordinate systems.

- **SCATTERED** is a variable set to 0 when entering a component, which is incremented each time a SCATTER event occurs. This may be used in the `EXTEND` sections to determine whether the component interacted with the current neutron ray.

- **extend_list**(n, &*arr*, &*len*, *elemsize*). Given an array *arr* with *len* elements each of size *elemsize*, make sure that the array is big enough to hold at least n elements, by extending *arr* and *len* if necessary. Typically used when reading a list of numbers from a data file when the length of the file is not known in advance.

- **mcset_ncount**(n). Sets the number of neutron histories to simulate to n.

- **mcget_ncount**(). Returns the number of neutron histories to simulate (usually set by option `-n`).

- **mcget_run_num**(). Returns the number of neutron histories that have been simulated until now.

A.1.3 Coordinate transformations

- **coords_set**(x, y, z) returns a Coord structure (like POS_A_CURRENT_COMP) with x, y and z members.

- **coords_get**(P, &x, &y, &z) copies the x, y and z members of the Coord structure P into x, y, z variables.

- **coords_add**(a, b), **coords_sub**(a, b), **coords_neg**(a) enable to operate on coordinates, and return the resulting Coord structure.
- **rot_set_rotation**(*Rotation t*, ϕ_x, ϕ_y, ϕ_z) Get transformation matrix for rotation first ϕ_x around x axis, then ϕ_y around y, and last ϕ_z around z. t should be a 'Rotation' ([3][3] 'double' matrix).
- **rot_mul***(Rotation t1, Rotation t2, Rotation t3)* performs $t3 = t1.t2$.
- **rot_copy***(Rotation dest, Rotation src)* performs $dest = src$ for Rotation arrays.
- **rot_transpose***(Rotation src, Rotation dest)* performs $dest = src^t$.
- **rot_apply***(Rotation t, Coords a)* returns a Coord structure which is $t.a$

A.1.4 Mathematical routines

- **NORM**(x, y, z). Normalizes the vector (x, y, z) to have length 1.
- **scalar_prod**$(a_x, a_y, a_z, b_x, b_y, b_z)$. Returns the scalar product of the two vectors (a_x, a_y, a_z) and (b_x, b_y, b_z).
- **vec_prod**$(a_x, a_y, a_z, b_x, b_y, b_z, c_x, c_y, c_z)$. Sets (a_x, a_y, a_z) equal to the vector product $(b_x, b_y, b_z) \times (c_x, c_y, c_z)$.
- **rotate**$(x, y, z, v_x, v_y, v_z, \varphi, a_x, a_y, a_z)$. Set (x, y, z) to the result of rotating the vector (v_x, v_y, v_z) the angle φ (in radians) around the vector (a_x, a_y, a_z).
- **normal_vec**($\&n_x$, $\&n_y$, $\&n_z$, x, y, z). Computes a unit vector (n_x, n_y, n_z) normal to the vector (x, y, z).
- **solve_2nd_order**($*t$, A, B, C). Solves the 2^{nd} order equation $At^2 + Bt + C = 0$ and returns the smallest positive solution into pointer $*t$.

A.1.5 Output from detectors

Details about using these functions are given in the McStas User Manual.

- **DETECTOR_OUT_0D**(...). Used to output the results from a single detector. The name of the detector is output together with the simulated intensity and estimated statistical error. The output is produced in a format that can be read by McStas front-end programs.
- **DETECTOR_OUT_1D**(...). Used to output the results from a one-dimentional detector.
- **DETECTOR_OUT_2D**(...). Used to output the results from a two-dimentional detector.
- **DETECTOR_OUT_3D**(...). Used to output the results from a three-dimentional detector. Arguments are the same as in DETECTOR_OUT_2D, but with the additional z axis (the signal). Resulting data files are treated as 2D data, but the 3rd dimension is specified in the *type* field.

- **mcinfo_simulation***(FILE *f, mcformat, char *pre, char *name)* is used to append the simulation parameters into file f (see for instance **Res_monitor**). Internal variable $mcformat$ should be used as specified. Please contact the authors for further information.

A.1.6 Ray-geometry intersections

- **box_intersect**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, d_x, d_y, d_z). Calculates the (0, 1, or 2) intersections between the neutron path and a box of dimensions d_x, d_y, and d_z, centered at the origin for a neutron with the parameters (x, y, z, v_x, v_y, v_z). The times of intersection are returned in the variables t_1 and t_2, with $t_1 < t_2$. In the case of less than two intersections, t_1 (and possibly t_2) are set to zero. The function returns true if the neutron intersects the box, false otherwise.

- **cylinder_intersect**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, r, h). Similar to **box_intersect**, but using a cylinder of height h and radius r, centered at the origin.

- **sphere_intersect**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, r). Similar to **box_intersect**, but using a sphere of radius r.

A.1.7 Random numbers

- **rand01**(). Returns a random number distributed uniformly between 0 and 1.

- **randnorm**(). Returns a random number from a normal distribution centered around 0 and with $\sigma = 1$. The algorithm used to get the normal distribution is explained in Ref. [47, ch.7].

- **randpm1**(). Returns a random number distributed uniformly between -1 and 1.

- **randvec_target_circle**(&v_x, &v_y, &v_z, &$d\Omega$, aim$_x$, aim$_y$, aim$_z$, r_f). Generates a random vector (v_x, v_y, v_z), of the same length as (aim$_x$, aim$_y$, aim$_z$), which is targeted at a *disk* centered at (aim$_x$, aim$_y$, aim$_z$) with radius r_f (in meters), and perpendicular to the *aim* vector.. All directions that intersect the circle are chosen with equal probability. The solid angle of the circle as seen from the position of the neutron is returned in $d\Omega$. This routine was previously called **randvec_target_sphere** (which still works).

- **randvec_target_rect_angular**(&v_x, &v_y, &v_z, &$d\Omega$, aim$_x$, aim$_y$, aim$_z$,h, w, Rot) does the same as randvec_target_circle but targetting at a rectangle with angular dimensions h and w (in **radians**, not in degrees as other angles). The rotation matrix *Rot* is the coordinate system orientation in the absolute frame, usually ROT_A_CURRENT_COMP.

- **randvec_target_rect**(&v_x, &v_y, &v_z, &$d\Omega$, aim$_x$, aim$_y$, aim$_z$,$height, width, Rot$) is the same as randvec_target_rect_angular but *height* and *width* dimensions are given in meters. This function is useful to e.g. target at a guide entry window or analyzer blade.

A.2 Reading a data file into a vector/matrix (Table input, read_table-lib)

The `read_table-lib` provides functionalities for reading text (and binary) data files. To use this library, add a `%include "read_table-lib"` in your component definition DECLARE or SHARE section. Tables are structures of type t_{Table} (see `read_table-lib.h` file for details):

```
/* t_Table structure (most important members) */
double *data;     /* Use Table_Index(Table, i j) to extract [i,j] element */
long    rows;     /* number of rows */
long    columns;  /* number of columns */
char   *header;   /* the header with comments */
char   *filename; /* file name or title */
double  min_x;    /* minimum value of 1st column/vector */
double  max_x;    /* maximum value of 1st column/vector */
```

Available functions to read *a single* vector/matrix are:

- **Table_Init**(&$Table$, *rows*, *columns*) returns an allocated Table structure. Use *rows* = *columns* = 0 not to allocate memory. Calls to Table_Init are optional, since initialization is being performed by other functions already.
- **Table_Read**(&$Table$, $filename$, *block*) reads numerical block number *block* (0 to catenate all) data from *text* file $filename$ into $Table$, which is as well initialized in the process. The block number changes when the numerical data changes its size, or a comment is encoutered (lines starting by '`# ; % /`'). If the data could not be read, then $Table$.*data* is NULL and $Table$.*rows* = 0. You may then try to read it using Table_Read_Offset_Binary. Return value is the number of elements read.
- **Table_Read_Offset**(&$Table$, $filename$, *block*, &*offset*, n_{rows}) does the same as Table_Read except that it starts at offset *offset* (0 means begining of file) and reads n_{rows} lines (0 for all). The *offset* is returned as the final offset reached after reading the n_{rows} lines.
- **Table_Read_Offset_Binary**(&$Table$, $filename$, *type*, *block*, &*offset*, n_{rows}, $n_{columns}$) does the same as Table_Read_Offset, but also specifies the *type* of the file (may be "float" or "double"), the number n_{rows} of rows to read, each of them having $n_{columns}$ elements. No text header should be present in the file.
- **Table_Rebin**(&$Table$) rebins all $Table$ rows with increasing, evenly spaced first column (index 0), e.g. before using Table_Value. Linear interpolation is performed for all other columns. The number of bins for the rebinned table is determined from the smallest first column step.
- **Table_Info**($Table$) print information about the table $Table$.
- **Table_Index**($Table, m, n$) reads the $Table[m][n]$ element.

- **Table_Value**($Table, x, n$) looks for the closest x value in the first column (index 0), and extracts in this row the n-th element (starting from 0). The first column is thus the 'x' axis for the data. It does not contain any interpolation method.
- **Table_Free**(&$Table$) free allocated memory blocks.

Available functions to read *an array* of vectors/matrices in a *text* file are:

- **Table_Read_Array**($File$, &n) read and split $file$ into as many blocks as necessary and return a t_{Table} array. Each block contains a single vector/matrix. This only works for text files. The number of blocks in put into n.
- **Table_Free_Array**(&$Table$) free the $Table$ array.
- **Table_Info_Array**(&$Table$) display information about all data blocks.

The format of text files is free. Lines starting by '`# ; % /`' characters are considered to be comments, and stored in $Table.header$. Data blocks are vectors and matrices. Block numbers are counted starting from 1, and changing when a comment is found, or the column number changes. For instance, the file 'MCSTAS/data/BeO.trm' (Transmission of a Berylium filter) looks like:

```
# BeO transmission, as measured on IN12
# Thickness: 0.05 [m]
# [ k(Angs-1) Transmission (0-1) ]
# wavevector multiply
1.0500  0.74441
1.0750  0.76727
1.1000  0.80680
...
```

Binary files should be of type "float" (i.e. REAL*32) and "double" (i.e. REAL*64), and should *not* contain text header lines. These files are platform dependent (little or big endian).

The $filename$ is first searched into the current directory (and all user additional locations specified using the `-I` option, see the 'Running McStas ' chapter in the User Manual), and if not found, in the `data` sub-directory of the `MCSTAS` library location. This way, you do not need to have local copies of the McStas Library Data files (see table 1.1).

A usage example for this library part may be:

```
t_Table Table;       // declare a t_Table structure
char file[]="BeO.trm";  // a file name
double x,y;

Table_Read(&Table, file, 1);  // initialize and read the first numerical block
Table_Info(Table);            // display table informations
...
x = Table_Index(Table, 2,5);  // read the 3rd row, 6th column element
                              // of the table. Indexes start at zero in C.
```

```
y = Table_Value(Table, 1.45,1);  // look for value 1.45 in 1st column (x axis)
                               // and extract 2nd column value of that row
Table_Free(&Table);            // free allocated memory for table
```

Additionally, if the block number (3rd) argument of **Table_Read** is 0, all blocks will be catenated. The **Table_Value** function assumes that the 'x' axis is the first column (index 0). Other functions are used the same way with a few additional parameters, e.g. specifying an offset for reading files, or reading binary data.

This other example for text files shows how to read many data blocks:

```
t_Table *Table;       // declare a t_Table structure array
long     n;
double y;

Table = Table_Read_Array("file.dat", &n); // initialize and read the all numerical block
n = Table_Info_Array(Table);     // display informations for all blocks (also returns n)

y = Table_Index(Table[0], 2,5);  // read in 1st block the 3rd row, 6th column element
                                 // ONLY use Table[i] with i < n !
Table_Free_Array(Table);         // free allocated memory for Table
```

You may look into, for instance, the source files for **Monochromator_curved** or **Virtual_input** for other implementation examples.

A.3 Monitor_nD Library

This library gathers a few functions used by a set of monitors e.g. Monitor_nD, Res_monitor, Virtual_output, etc. It may monitor any kind of data, create the data files, and may display many geometries (for `mcdisplay`). Refer to these components for implementation examples, and ask the authors for more details.

A.4 Adaptative importance sampling Library

This library is currently only used by the components **Source_adapt** and **Adapt_check**. It performs adaptative importance sampling of neutrons for simulation efficiency optimization. Refer to these components for implementation examples, and ask the authors for more details.

A.5 Vitess import/export Library

This library is used by the components **Vitess_input** and **Vitess_output**, as well as the `mcstas2vitess` utility. Refer to these components for implementation examples, and ask the authors for more details.

A.6 Constants for unit conversion etc.

The following predefined constants are useful for conversion between units

Name	Value	Conversion from	Conversion to
DEG2RAD	$2\pi/360$	Degrees	Radians
RAD2DEG	$360/(2\pi)$	Radians	Degrees
MIN2RAD	$2\pi/(360 \cdot 60)$	Minutes of arc	Radians
RAD2MIN	$(360 \cdot 60)/(2\pi)$	Radians	Minutes of arc
V2K	$10^{10} \cdot m_{\mathrm{N}}/\hbar$	Velocity (m/s)	**k**-vector (Å^{-1})
K2V	$10^{-10} \cdot \hbar/m_{\mathrm{N}}$	**k**-vector (Å^{-1})	Velocity (m/s)
VS2E	$m_{\mathrm{N}}/(2e)$	Velocity squared (m^2 s^{-2})	Neutron energy (meV)
SE2V	$\sqrt{2e/m_{\mathrm{N}}}$	Square root of neutron energy (meV$^{1/2}$)	Velocity (m/s)
FWHM2RMS	$1/\sqrt{8\log(2)}$	Full width half maximum	Root mean square (standard deviation)
RMS2FWHM	$\sqrt{8\log(2)}$	Root mean square (standard deviation)	Full width half maximum
MNEUTRON	$1.67492 \cdot 10^{-27}$ kg	Neutron mass, m_{n}	
HBAR	$1.05459 \cdot 10^{-34}$ Js	Planck constant, $\hbar$	
PI	3.14159265...	π	
FLT_MAX	3.40282347E+38F	a big float value	

Appendix B

Random numbers in McStas

B.1 Transformation of random numbers

In order to perform the Monte Carlo choices, one needs to be able to pick a random number from a given distribution. However, most random number generators only give uniform distributions over a certain interval. We thus need to be able to transform between probability distributions, and we here give a short explanation on how to do this.

Assume that we pick a random number, x, from a distribution $\phi(x)$. We are now interested in the shape of the distribution, $\Psi(y)$, of the transformed $y = f(x)$, assuming $f(x)$ is monotonous. All random numbers lying in the interval $[x; x + dx]$ are transformed to lie within the interval $[y; y + f'(x)dx]$, so the resulting distribution must be $\Psi(y) = \phi(x)/f'(x)$.

If the random number generator selects numbers uniformly in the interval $[0; 1]$, we have $\phi(x) = 1$ (inside the interval; zero outside), and we reach

$$\Psi(y) = \frac{1}{f'(x)} = \frac{d}{dy} f^{-1}(y). \tag{B.1}$$

By indefinite integration we reach

$$\int \Psi(y) dy = f^{-1}(y) = x, \tag{B.2}$$

which is the essential formula for random number transformation, since we in general know $\Psi(y)$ and like to determine the relation $y = f(x)$. Let us illustrate with a few examples of transformations relevant for the McStas components.

The circle For finding a random point within the circle of radius R, one would like to choose the polar angle, ϕ, from a uniform distribution in $[0; 2\pi]$, giving $\Psi_\phi = 1/(2\pi)$. and the radius from the (normalised) distribution $\Psi_r = 2r/R^2$.

For the radial part, eq. (B.2) becomes $y/(2\pi) = x$, whence ϕ is found simply by multiplying a random number (x) with 2π.

For the radial part, the left side of eq. (B.2), gives $\int \Psi(r)dr = \int 2r/R^2 dr = r^2/R^2$, which from (B.2) should equal x. Hence we reach the wanted transformation $r = R\sqrt{x}$.

The sphere For finding a random point on the surface of the unit sphere, we need to determine the two angles, (θ, ϕ).

Ψ_ϕ is chosen from a uniform distribution in $[0; 2\pi]$, giving $\phi = 2\pi x$ as for the circle.

The probability distribution of θ should be $\Psi_\theta = \sin(\theta)$ (for $\theta \in [0; \pi]$), whence by eq. (B.2) $\theta = \cos^{-1}(x)$.

Exponential decay In a simple time-of-flight source, the neutron flux decays exponentially after the initial activation at $t = 0$. We thus want to pick an initial neutron emission time from the normalised distribution $\Psi(t) = \exp(-t/\tau)/\tau$. Use of Eq. (B.2) gives $x = 1 - \exp(-t/\tau)$. For convenience we now use the random variable $x_1 = 1 - x$ (with the same distributions as x), giving the simple expression $t = -\tau \ln(x_1)$.

Normal distributions The important normal distribution can not be reached as a simple transformation of a uniform distribution. In stead, we rely on a specific algorithm for selecting random numbers with this distribution.

B.2 Random generators

Eventhough there is the possibility to use the system random generator, as well as ther initial McStas version 1.1 random generator, the default algorithm is the so-called "Mersenne Twister", by Makoto Matsumoto and Takuji Nishimura. See `http://www.math.keio.ac.jp/~matumoto/e` for original source.

It is considered today to be by far the best random generator, which means that both its period is extremely large, and cross-correlations are negligible.

Appendix C

The McStas terminology

This is a short explanation of phrases and terms which have a specific meaning within McStas. We have tried to keep the list as short as possible with the risk that the reader may occasionally miss an explanation. In this case, you are more than welcome to contact the McStas core team.

- **Arm** A generic McStas component which defines a frame of reference for other components.
- **Component** One unit (*e.g.* optical element) in a neutron spectrometer.
- **Definition parameter** An input parameter for a component. For example the radius of a sample component or the divergence of a collimator.
- **Input parameter** For a component, either a definition parameter or a setting parameter. These parameters are supplied by the user to define the characteristics of the particular instance of the component definition. For an instrument, a parameter that can be changed at simulation run-time.
- **Instrument** An assembly of McStas components defining a neutron spectrometer.
- **Kernel** The McStas language definition and the associated compiler
- **McStas** Monte Carlo Simulation of Triple Axis Spectrometers (the name of this package). Pronounciation ranges from *mex-tas*, to *mac-stas* and *m-c-stas*.
- **Output parameter** An output parameter for a component. For example the counts in a monitor. An output parameter may be accessed from the instrument in which the component is used using `MC_GETPAR`.
- **Run-time** C code, contained in the files `mcstas-r.c` and `mcstas-r.h` included in the McStas distribution, that declare functions and variables used by the generated simulations.
- **Setting parameter** Similar to a definition parameter, but with the restriction that the value of the parameter must be a number.

Bibliography

[1] K. Lefmann and K. Nielsen. *Neutron News*, **10**, 20–23, 1999.

[2] Peter Willendrup, Emmanuel Farhi, Kim Lefmann, and Klaus Lieutenant. *User and Programmers Guide to the Neutron Ray-Tracing Package McStas, Version 1.9*. Risoe Report, 2005.

[3] See http://www.mcstas.org.

[4] See http://neutron.risoe.dk/McZilla/.

[5] See http://mcnsi.risoe.dk/.

[6] See http://neutron-eu.net/en/.

[7] H. Maier-Leibnitz and T. Springer. *Reactor Sci. Technol.*, **17**, 217, 1963.

[8] D.F.R. Mildner. *Nucl. Instr. Meth.*, **A290**, 189, 1990.

[9] R.A. Lowde. *J. Nucl. Energy, Part A: Reactor Sci.*, **11**, 69, 1960.

[10] J.R.D. Copley. *Nucl. Instr. Meth.*, **A510**, 318, 2003.

[11] E. Fermi, J. Marshall, and L. Marshall. *Phys. Rev.*, **72**, 193, 1947.

[12] J. Peters. *Nucl. Instr. Meth.*, **A540**, 419, 2005.

[13] C.D. Clark, E.W.J. Mitchell, D.W. Palmer, and I.H. Wilson. *J. Sci. Instrum.*, **43**, 1, 1966.

[14] A.K. Freund. *Nucl. Instr. Meth.*, **213**, 495, 1983.

[15] V.F. Sears. *Acta Cryst.*, **A53**, 35, 1997.

[16] G. Shirane, S.M. Shapiro, and J.M. Tranquada. *Neutron Scattering with a Triple-Axis Spectrometer*. Cambridge University Press, 2002.

[17] L. Alianelli. *J. Appl. Cryst.*, **37**, 732, 2004.

[18] V. Radeka. *IEEE Trans. Nucl. Sci.*, **NS-21**, 51, 1974.

[19] V. Peskov, G. Charpak, W. Dominik, and F. Sauli. *Nucl. Instr. and Meth.*, **A277**, 547, 1989.

[20] G. Manzin, B. Guerard, F.A.F. Fraga, and L.M.S. Margato. *Nucl. Instr. Meth.*, **A535**, 102, 2004.

[21] J.R.D. Copley. *J. Neut. Research*, **1**, 21, 1993.

[22] L. D. Cussen. *J. Appl. Cryst.*, **36**, 1204, 2003.

[23] F. James. *Rep. Prog. Phys.*, **43**, 1145, 1980.

[24] Grimmett, G. R., and Stirzakerand D. R. *Probability and Random Processes, 2nd Edition.* Clarendon Press, Oxford, 1992.

[25] P. A. Seeger, L. L. Daemen, T. G. Thelliez, and R. P. Hjelm. *Physica B*, **283**, 433, 2000.

[26] J. Saroun and J. Kulda. *Physica B*, **234**, 1102, 1997. See website http://omega.ujf.cas.cz/restrax/.

[27] P. Willendrup, E. Farhi, and K. Lefmann. *Physica B*, **350**, 735, 2004.

[28] D. Wechsler, G. Zsigmond, F. Streffer, and F. Mezei. *Neutron News*, **25**, 11, 2000.

[29] See http://www.hmi.de/projects/ess/vitess/.

[30] W.-T. Lee and X.-L. Wang. *Neutron News*, **13**, 30, 2002. See website http://www.sns.gov/ideas/.

[31] E. Farhi, T. Hansen, A. Wildes, R. Ghosh, and K. Lefmann. *Appl. Phys.*, **A 74**, S1471, 2002.

[32] C. Schanzer, P. Boni, U. Filges, and T. Hils. *Nucl. Instr. Meth.*, **A 529**, 63, 2004.

[33] G. Zsigmond, K. Lieutenant, and S. Manoshin et al. *Nucl. Instr. Meth.*, **A 529**, 218, 2004.

[34] K. Lieutenant. *J. Phys.: Condens. Matter*, **17**, S167, 2005.

[35] See http://www.nea.fr/html/dbprog/tripoli-abs.html.

[36] K. N. Clausen, D. F. McMorrow, K. Lefmann, G. Aeppli, T. E. Mason, A. Schröder, M. Issikii, M. Nohara, and H. Takagi. *Physica B*, **241-243**, 50–55, 1998.

[37] Y. Blanc. *ILL Report*, **83BL21G**, 1983.

[38] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes (2nd Edition).* Cambridge University Press, 2002.

[39] G.E. Bacon. *Neutron Diffraction.* Oxford University Press, 1975.

[40] G.L. Squires. *Thermal Neutron Scattering.* Cambridge University Press, 1978.

[41] Crystallographica, Oxford Cryosystem, 1998. See http://www.crystallographica.com.

[42] ICSD, Inorganic Crystal Structure Database. See http://icsd.ill.fr.

[43] Fullprof powder refinement. See http://www-llb.cea.fr/fullweb/fp2k/fp2k.htm.

[44] A.-J. Dianoux and G. Lander. *ILL Neutron Data Booklet.* OCP Science, 2003.

[45] P.A. Egelstaff. *An introduction to the liquid state.* Academic Press, London, 1967.

[46] S.W. Lovesey. *Theory of neutron scattering from condensed matter.* Oxford Clarendon Press, 1984.

[47] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1986.

Index

Bibliographic Data Sheet			**Risø–R–1538(EN)**

Title and author(s)

Component Manual to the Neutron Ray-Tracing Package McStas, Version 1.9

Kim Lefmann, Peter Kjær Willendrup, Kristian Nielsen, Emmanuel Farhi, Klaus Lieutenant

ISBN			ISSN
ISBN 87–550–3482–92 (Internet)			0106–2840
Dept. or group			Date
Materials Research Department			November 15, 2005
Groups own reg. number(s)		Project/contract No.	
—		—	
Pages	Tables	Illustrations	References
132	2	15	10

Abstract (Max. 2000 char.)

The software package McStas is a tool for carrying out Monte Carlo ray-tracing simulations of neutron scattering instruments with high complexity and precision. The simulations can compute all aspects of the performance of instruments and can thus be used to optimize the use of existing equipment, design new instrumentation, and carry out virtual experiments. McStas is based on a unique design where an automatic compilation process translates high-level textual instrument descriptions into efficient ANSI-C code. This design makes it simple to set up typical simulations and also gives essentially unlimited freedom to handle more unusual cases.
This report constitutes the reference manual for McStas, and, together with the manual for the McStas components, it contains full documentation of all aspects of the program. It covers the various ways to compile and run simulations, a description of the meta-language used to define simulations, and some example simulations performed with the program.

Descriptors

Neutron Instrumentation; Monte Carlo Simulation; Software

Available on request from:
Information Service Department, Risø National Laboratory
(Afdelingen for Informationsservice, Forskningscenter Risø)
P.O. Box 49, DK–4000 Roskilde, Denmark
Phone +45 4677 4004, Telefax +45 4677 4013