Risø–R–1538(rev.ed.)(EN)

Component Manual for the X-ray-Tracing Package McXtrace, Version 1.0_rc1

Erik Knudsen, Andrea Prodi, Peter Kjær Willendrup and Kim Lefmann

Risø DTU, Roskilde, Denmark
March 2011

Abstract

The software package McXtrace is a tool for carrying out Monte Carlo ray-tracing simulations of xray scattering beamlines 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 for e.g. training, experimental planning or data analysis. McXtrace is based is based on a unique design, inhereted from its sister McStas, 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 McXtrace, and, together with the manual for the McXtrace system, it contains full documentation of all aspects of the program. It covers a description of all official components of the McXtracepackage with some theoretical background. Selected test instruments and representative McXtrace simulations performed with these instruments are described in the User Manual.

This report documents the components for McXtraceversion 1.0_rc1, released Mar, 2011.

The authors are:

Erik Knudsen
Materials Research Department, Risø DTU, Roskilde, Denmark
email: `erik.knudsen@risoe.dk`

Andrea Prodi
Niels Bohr Institute, University of Copenhagen, Denmark
email: `aprodi@nbi.ku.dk`

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

Kim Lefmann
Niels Bohr Institute, University of Copenhagen, Denmark
email: `lefmann@fys.ku.dk`

ISBN 978–87–550–3680–2
ISSN 0106–2840

Information Service Department · Risø DTU · 2011

Contents

Preface and acknowledgements

This document contains information on the x-ray scattering components which are the building blocks for defining instruments in the Monte Carlo Xray-tracing program McXtraceversion 1.0_rc1. The initial release in June 2011 of version 1.0 was presented in Ref. [1]. The reader of this document is not supposed to have specific knowledge of xray 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 McXtracesystem manual [?]. We assume familiarity with the use of the C programming language.

We would like to explicitly thank all the partners in this project:

- The European Synchrotron Radiation Facility (ESRF), Grenoble, France
- SAXSLAB Aps., Lundtofte, Denmark
- Ris DTU, Roskilde, Denmark
- Niels Bohr Institute, Univeristy of Copenhagen, Copenhagen, Denmark
- Faculty for Life Sciences, University of Copenhagen, Copenhagen, Denmark

As McXtracehas inherited much of its functionality from its sister McXtracewe take the oppurtunity to thank Dir. Kurt N. Clausen, PSI, for his continuous support to McXtrace and for having initiated the project. Continuous support to McXtrace 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 McXtrace system. We have further benefited from discussions with many other people in the 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 McXtrace.

In case of errors, questions, or suggestions, do not hesitate to contact the user/developer community by writing to the user mailiing list `users@mcxtrace.rg` or consult the McXtrace home page [2]. A special devlopement website (shared with the sister project McXtrace) complete with bug/request reporting service is available [?].

Some highlight of the feature in this the first release of McXtrace:

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

If you **appreciate** this software, please subscribe to the `users@mcxtrace.org` email list, send us a smiley message, and contribute to the package.

Chapter 1

About the component library

This McXtrace Component Manual consists of the following major parts:

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

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

1.1 Authorship

The component library is maintained by the McXtrace system group. A number of basic components "belongs" the McXtrace system, and are supported and tested by the McXtrace 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. McXtrace users are encouraged to send their contributions to us for inclusion in future releases.

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.

MCXTAS/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 McXtrace library.

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 not necessarily designed to overlap. This may lead to loss of rays. Warnings will be issued during simulation if sections of the instrument are not reached by any xrays, or if a significant number of xrays 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, absoprtion and reflectivity files, etc.
Such files distributed with McXtrace are located in the `data` sub-directory of the `MCXTRACE` library. Components that make use of the McXtrace file system, including the `read-table` library (see section A.2) may access all McXtrace data files without making local copies. Of course, you are welcome to define your own data files, and eventually contribute to McXtrace if you find them useful.
File extensions are not compulsory but help in identifying relevant files per application. We list powder and liquid data files from the McXtrace library in Tables 1.2 and 1.3. These files contain an extensive header describing physical properties with references, and are specially suited for the PowderN (see 7.1) and Isotropic_Sqw components (see **??**).
McXtrace itself generates both simulation and monitor data files, which structure is explained in the User Manual (see end of chapter 'Running McXtrace ').

MCXTAS/data File name	σ_{coh} [barns]	σ_{inc} [barns]	σ_{abs} [barns]	T_m [K]	c [m/s]	Note
Ag.laz	4.407	0.58	**63.3**	1234.9	2600	
Al2O3_sapphire.laz	15.683	0.0188	0.4625	2273		
Al.laz	1.495	0.0082	0.231	933.5	5100	.lau
Au.laz	7.32	0.43	**98.65**	1337.4	**1740**	
B4C.laz	19.71	6.801	**3068**	2718		
Ba.laz	3.23	0.15	29.0	1000	**1620**	
Be.laz	7.63	0.0018	0.0076	1560	13000	
BeO.laz	11.85	0.003	0.008	2650		.lau
Bi.laz	9.148	0.0084	0.0338	544.5	**1790**	
C60.lau	5.551	0.001	0.0035			
C_diamond.laz	5.551	0.001	0.0035	4400	18350	.lau
C_graphite.laz	5.551	0.001	0.0035	3800	18350	.lau
Cd.laz	3.04	3.46	**2520**	594.2	2310	
Cr.laz	1.660	1.83	3.05	2180	5940	
Cs.laz	3.69	0.21	29.0	301.6	**1090**	c in liquid
Cu.laz	7.485	0.55	3.78	1357.8	3570	
Fe.laz	11.22	0.4	2.56	1811	4910	
Ga.laz	6.675	0.16	2.75	302.91	2740	
Gd.laz	29.3	151	**49700**	1585	2680	
Ge.laz	8.42	0.18	2.2	1211.4	5400	
H2O_ice_1h.laz	7.75	160.52	0.6652	273		
Hg.laz	20.24	6.6	**372.3**	234.32	**1407**	
I2.laz	7.0	0.62	12.3	386.85		
In.laz	2.08	0.54	**193.8**	429.75	**1215**	
K.laz	.69	0.27	2.1	336.53	**2000**	
LiF.laz	4.46	0.921	**70.51**	1140		
Li.laz	0.454	0.92	**70.5**	453.69	6000	
Nb.laz	8.57	0.0024	1.15	2750	3480	
Ni.laz	13.3	5.2	4.49	1728	4970	
Pb.laz	11.115	0.003	0.171	600.61	**1260**	
Pd.laz	4.39	0.093	6.9	1828.05	3070	
Pt.laz	11.58	0.13	10.3	2041.4	2680	
Rb.laz	6.32	0.5	0.38	312.46	**1300**	
Se_alpha.laz	7.98	0.32	11.7	494	3350	
Se_beta.laz	7.98	0.32	11.7	494	3350	
Si.laz	2.163	0.004	0.171	1687	2200	
SiO2_quartza.laz	10.625	0.0056	0.1714	846		.lau
SiO2_quartzb.laz	10.625	0.0056	0.1714	1140		.lau
Sn_alpha.laz	4.871	0.022	0.626	505.08		
Sn_beta.laz	4.871	0.022	0.626	505.08	2500	
Ti.laz	1.485	2.87	6.09	1941	4140	
Tl.laz	9.678	0.21	3.43	577	**818**	
V.laz	.0184	4.935	5.08	2183	4560	
Zn.laz	4.054	0.077	1.11	692.68	3700	
Zr.laz	6.44	0.02	0.185	2128	3800	

Table 1.2: Powders of the McXtrace library [3, 4]. Low c and high σ_{abs} materials are highlighted. Files are given in LAZY format, but may exist as well in Crystallographica *.lau* format as well.

MCXTAS/data File name	σ_{coh} [barns]	σ_{inc} [barns]	σ_{abs} [barns]	T_m [K]	c [m/s]	Note
Cs_liq_tot.sqw	3.69	0.21	29.0	301.6	**1090**	Measured
Ge_liq_coh.sqw and Ge_liq_inc.sqw	8.42	0.18	2.2	1211.4	5400	Ab-initio MD
He4_liq_coh.sqw	1.34	0	0.00747	0	**240**	Measured
Ne_liq_tot.sqw	2.62	0.008	0.039	24.56	**591**	Measured
Rb_liq_coh.sqw and Rb_liq_inc.sqw	6.32	0.5	0.38	312.46	**1300**	Classical MD
Rb_liq_tot.sqw	6.32	0.5	0.38	312.46	**1300**	Measured

Table 1.3: Liquids of the McXtrace library [3, 4]. Low c and high σ_{abs} materials are highlighted.

1.5 Component source code

Source code for all components may be found in the `MCXTAS` 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 `MCXTAS` environment variable.
In case users only require to add new features, preserving the existing features of a component, using the `EXTEND` keyword in the instrument description file is recommended. 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 McXtrace 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 McXtrace 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 McXtrace home page [5] and in the User Manual [6].

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 McXtrace 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 McXtrace 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 [6] 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 [7].

Chapter 2

Monte Carlo Techniques and simulation strategy

This chapter explains the simulation strategy and the Monte Carlo techniques used in McXtrace. We first explain the concept of the x-ray weight factor, and discuss the statistical errors in dealing with sums of x-ray 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. More details are available in the Appendix concerning random numbers in the User manual.

2.1 X-ray simulations

X-ray scattering beamlines are built as a series of optical elements. Each of these elements modifies the beam characteristics (e.g. divergence, wavelength spread, spatial and temporal distributions) in a way which, for simple x-ray beam configurations, may be modelled with analytical methods.
However, real x-ray beamlines consist of a large number of optical elements, and this brings additional complexity by introducing strong correlations between x-ray beam parameters like divergence and position - which is the basis of the acceptance diagram method - but also wavelength and time. The usual analytical methods, such as phase-space theory, then reach their limit of validity in the description of the resulting effects.
In order to cope with this difficulty, Monte Carlo (MC) methods (for a general review, see Ref. [8]) may be applied to the simulation of x-ray instruments. The use of probability is commonplace in the description of microscopic physical processes. Integrating these events (absorption, scattering, reflection, ...) over the x-ray trajectories results in an estimation of measurable quantities characterizing the beamline. Moreover, using variance reduction (importance sampling) where possible, reduces the computation time and gives better accuracy.
Implementations of the MC method for X-ray beamlines already exist, most notable is probably *SHADOW*, originally developed by the late Franco Cerrina and coworkers, now developed further by M. Sanchez Del Rio at the ESRF[**?**]. Other implementation of the same concept are *Ray*[**?**] from BESSY and Xtrace([**?**]).

2.1.1 Monte Carlo ray tracing simulations

Mathematically, the Monte-Carlo method is an application of the law of large numbers [8, 9]. 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.

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

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 uniformly) 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 [8] for a detailed review on the Monte-Carlo method.

2.2 The x-ray weight

A totally realistic semi-classical simulation will require that each x-ray is at any time either present or lost. On many beamlines, only a very small fraction of the initial x-rays will ever be detected, and simulations of this kind will therefore waste much time in dealing with x-rays that never hit the detector.
An important way of speeding up calculations is to introduce a x-ray "weight factor" for each simulated 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 x-rays ray are considered later in the simulations, the x-ray weight will be multiplied by 0.10 when passing this component, but every x-ray is allowed to reflect in the component. In contrast, the totally realistic simulation of the component would require on average ten incoming x-rays for each reflected one.
Let the initial x-ray 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 x-ray ray after passage of the whole instrument becomes the product of all contributions

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

Each adjustement factor should be $0 < \pi_j < 1$, except in special circumstances, so that total flux can only decrease through the simulation. 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 and windows.
- Reflection from monochromator (and analyser) crystals with finite reflectivity and mosaicity.
- Reflections from mirrors.

- Passage of a continuous beam through a chopper.
- Scattering from all types of samples.

2.2.1 Statistical errors of non-integer counts

In a typical simulation, the result will consist of a count of x-ray histories (”rays”) with different weights. The sum of these weights is an estimate of the mean number of x-rays 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 rays hitting 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 x-ray weights are almost equal, $p_i \approx \overline{p}$, and that we observe a large number of x-rays, $N \geq 10$. 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 x-ray 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:

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

Assuming as before 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 some 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 McXtrace will keep track of $N = \sum_i p_i^0, I = \sum_i p_i^1$, and $M_2 = \sum_i p_i^2$.

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

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 x-ray ray is decided at the source, it is important to adjust the x-ray weight so that the combined effect of x-ray 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 x-ray is P, but we make the Monte Carlo choice of transmitting the x-ray every time: $f_{\text{MC}} = 1$. This must be reflected on the choice of weight multiplier π_j given by the master equation

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

This probability rule is general, and holds also if, e.g., it is decided to transmit only half of the rays ($f_{\text{MC}} = 0.5$). 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 x-ray 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 x-rays in many directions. In general, only x-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 x-rays 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 x-rays rays that should have reached the final detector.

2.4 Stratified sampling

One particular efficiency improvement technique is the so-called *stratified sampling.* It consists in partitioning the event distributions in representative sub-spaces, which are then all sampled individualy. The advantage is that we are then sure that each sub-space is well represented in the final integrals. This means that instead of shooting N events, we define D partitions and shoot $r = N/D$ events in each partition. We may define partitions so that they represent 'interesting' distributions, e.g. from events scattered on a monochromator or a sample. The sum of partitions should equal the total space integrated by the Monte Carlo method, and each partition must be sampled randomly.

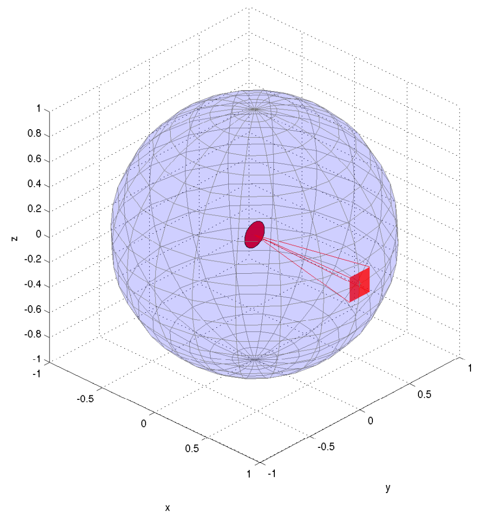

Figure 2.1: Illustration of the effect of direction focusing in McXtrace. Weights of x-rays emitted into a certain solid angle are scaled down by the full unit sphere area.

In the case of McXtrace, the stratified sampling is used when repeating events, i.e. when using the SPLIT keyword in the TRACE section on instrument descriptions. We emphasize here that the number of repetitions r should not exceed the dimensionality of the Monte Carlo integration space (which is $d = 10$ for x-ray events) and the dimensionality of the partition spaces, i.e. the number of random generators following the stratified sampling location in the instrument.

2.5 Accuracy of Monte Carlo simulations

When running a Monte Carlo, the meaningful quantities are obtained by integrating random events into a single value (e.g. flux), or onto an histogram grid. The theory [8] shows that the accuracy of these estimates is a function of the space dimension d and the number of events N. For large numbers N, the central limit theorem provides an estimate of the relative error as $1/\sqrt{N}$. However, the exact expression depends on the random distributions.
McXtrace uses a space with $d = 12$ parameters to describe x-rays (position, wavevector, weight, polarisation, phase, time). We show in Table 2.1 a rough estimate of the accuracy on integrals as a function of the number of records reaching the integration point. This stands both for integrated flux, as well as for histogram bins - for which the number of events per bin should be used for N.

Records	Accurarcy
10^3	10 %
10^4	2.5 %
10^5	1 %
10^6	0.25 %
10^7	0.05 %

Table 2.1: Accuracy estimate as a function of the number of statistical events used to estimate an integral with McXtrace.

Chapter 3

Source components

McXtrace contains a number of different source components, and any simulation will usually 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 Photon flux and Brilliance

The flux of the sources deserves special attention. The total intensity is defined as the sum of weights of all emitted xrays during one simulation (the unit of total photon weight is thus xrays 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 photon rays 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):

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

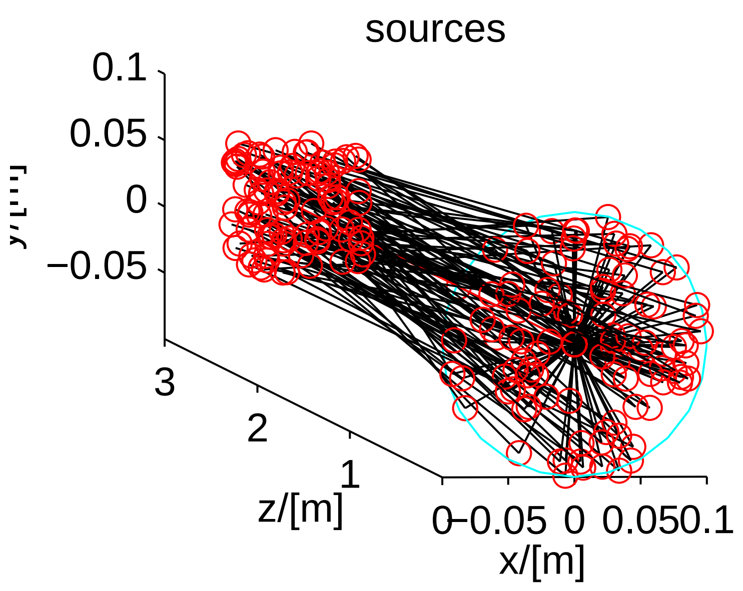

Figure 3.1: A circular source component (at z=0) emitting photon rays randomly, either from a model, or from a data file.

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 xray histories to simulate, the initial photon 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 recommend new McXtrace 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 3.6 for details.

3.1 Source_pt: A mathematical point emitting photons with a spectrum either uniform, gaussian or generated from a datafile

Name:	Source_pt
Author:	System
Input parameters	d,w,h
Optional parameters	λ_0,dλ,E, dE, *spectrum_file*, *incoherent*,*phase*
Notes	

The simplest source model, where a mathematical point source at $(0, 0, 0)$ emits photons. The wavevector of the emitted photons is picked randomly in a defining aperture w by h m at $(0, 0, d)$. Please note that this aperture is merely a virtual aperture used to reduce the sampling space. This has a few implications: Other components may be placed without reference to the aperture, but if the aperture does not fill the full acceptance window of subsequent components your simulations will be biased. The aperture is simply there to provide efficient sampling.
If a $spectrum\_file$ is not supplied, the xray is given a weight which is the total wavelength-integrated intensity downscaled by the solid subtended by the definning aperture.
If a $spectrum\_file$ *is* supplied, a slightly different strategy is adopted. In this case the wavelength/energy range implied by the datafile is sampled unformly and each ray is assigned a weight corresponding to the intensity indicated by linear interpolation between datapoints at that wavelength. This implies an oversampling of weak parts of the intensity spectrum.
Currently only completely coherent or fully incoherent beams are supported. If *phase* is specified emitted photons be assigne dthat phase, otherwise it is chosen randomly.

3.2 Source_flat: A flat surface emitting photons with a spectrum either uniform, gaussian or generated from a datafile

Name:	Source_flat
Author:	System
Input parameters	d,w,h
Optional parameters	r,x_{width},y_{height},d, λ_0,dλ, *spectrum_file*, *incoherent*,*phase*
Notes	

A simple source model, with a flat surface emitting photons. The surface in the xy-plane is specified as a rectangle with dimensions $x_{width} \times y_{height}$ m, or as a circle w radius,r. The initial xray position is chosen randomly in the source surface — its wavevector is chosen randomly (exactly as in the case of `Source\_pt` (section **??**) in the defining aperture with height h and width w placed at $(0, 0, dist)$.
Just as for `Source\_pt` the aperture is for efficiency purposes and, if misused, may cause biasing
A spectrum file may be supplied as for `Source\_pt`.
Currently only fully coherent or incoherent beams are supported. If *phase* is set (and not *randomphase* which takes precedence) a phase is set such that a photon emitted from $(x, y, 0)$ will be in phass with a photon at $(0, 0, 0)$, which has the phase *phase*..

Figure 3.2: Geometry of `Source_gaussian`

3.3 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 divergences 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.

3.4 Source_gaussian: Gaussian crossection source

Name:	Source_gaussian
Author:	System
Input parameters	r,σ_x,σ_y,$distance$
Optional parameters	γ,$focus_{xw}$,$focus_{yh}$,E_0,dE,λ_0,dλ,$phase$
Notes	

This component models a situation where a photons are emitted from a gaussian crossection in the XY-plane. Photons are emitted in the forward direction, in a $\frac{1}{\gamma}$-profile.

3.5 Source_lab: X-ray tube laboratory source

Name:	Source_lab
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_lab is a model of a laboratory X-ray tube. An electron ray hits a target of specified material. Currently, only single materiual targets are allowed[1].
An electron beam of transverse crossection (x_0, z_0) and energy E_0 impinges on the target of material. Wrt. the electron beam, the target is considereded infinitely thick. The beam

[1]To model multiple material targets one could construct a model with two or more sources simultaneously. This has consequences for intensity of the source which should be downscaled accordingly.

Figure 3.3: Geometry of the `Source_lab` component

Figure 3.4: Intensity vs. wavlenghth for a Cu-anode laboratory source.

is considered to have uniform intenisty. Thus, the spatial distribution of x-ray generation will be exponential in the depth of the material.
Further, an exit aperture is defined with dimensions (x_{width}, y_{height}). The centre of the aperture is situated at a distance wd m from where the electron beam hits the target slab at an elevation of $take\_off$ (see Figure 3.5). Note that the center of the exit aperture is the reference point of the `Source\_lab` coordinate system. In other words, the position specified in the instrument file `AT (x,y,z) RELATIVE somewhere` is the center of the exit aperture. Also note that the exit aperture is merely an opening. If the material absorption of the window, e.g. Be, is to be taken into account a `Filter.comp` (section **??**) could be inserted after the exit aperture.
For each photon to be generated, a monte carlo choice is made to either generate either a Bremstrahlung photon or one from one of the x-ray emission lines of the material. $frac$ of the photons are generated from characteristic emission, and $1-frac$ from Bremsstrahlung. In most cases Bremstrahlung is unwanted background, which is why the default is 0.9. Note that this *only* governs how much of the available statistics is diverted into simulating backgrouns. It does not have an impact on what intensity is detected in subsequent monitors — only on the errorbars of the detected numbers.
The spectral characteristics of the generated Bremsstrahlung is goverened by the model suggested by Kramer [?]. Although disputed in several subsequent papers, the model is simple, and sufficiently accurate for many background estimation purposes.
Characteristic emission on the other hand is sampled from a set of Lorentzian functions with central wavelengths found in the work by [?] with spectral widths taken from [?].
An example of beam spectral characteristics emitted from a Cu-anode targate detected 1 mm from an exit aperture of 1×1 cm 10 cm fround the target at a $take\_off$ angle of $6°$. is seen in figure 3.5.

3.6 Other sources components: virtual sources (event files)

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 xray. The purpose of this component is only to provide a standard means of defining a local coordinate system within the instrument definition. Other components may then be positioned relative to the **Arm** component using the McXtrace meta-language. The use of Arm components in the instrument definitions is not required but is recommended for clarity. **Arm** has no input parameters.
The first Arm instance in an instrument definition may be changed into a `Progress_bar`(sec. **??**) component in order to display simulation progress on the fly , and possibly save intermediate results.

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, $(x_{\rm min}, x_{\rm max}, y_{\rm min}, y_{\rm max})$ 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 photon 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. Photons that hit the plane within this area are ABSORB'ed, while all others are unaffected.

By using this component, some photons contributing to the background in a real experiment will be neglected. These are the ones that scatter off the side of the (real) beamstop, or penetrate 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 either height and width (x_{width}, y_{height}) or the four coordinates, $(x_{\rm min}, x_{\rm max}, y_{\rm min}, y_{\rm max})$ 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 photons 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;
%}
```

4.4 Filter: A general absoprtion filter model

Name:	Filter
Author:	System
Input parameters	*xwidth*, *yheight*, *zdepth* $material_datafile$
Optional parameters	*options*
Notes	not validated, absorption filter

This component is a filter in the shape of a ectangular block. Given an input file containing material parameters. Neccessary paramters are nominal density and a parametrization of mu as a function of wavelength (or energy).
The model is very simple: Firstly the X-ray is traced to find intersection points between ray and filter (0 or 2). If no intersection is found the xray is left untouched and nothing further happens. Assuming the ray intersects the filter: Secondly, the path length $\mathrm{d}l$ within the filter is computed. Thirdly a $\mu = f(\lambda, \text{material})$ is computed by interpolating in a datafile, and the xray weight is adjusted according to $p = p\exp(-\mathrm{d}l * \mu)$. The xray is left at the point where it exits the filter block (the $2n_{\downarrow}$ ntersection).
Some example data files are distributed with McXtrace in the `MCXTAS/data` directory as `*.txt` files. These tables have been extracted from the NIST x-ray database. To generate other datafiles see below- from the same source a simple shell script: `MCXTAS/data/get_xray_db_data` is also distributed with McXtraceRunning this script will connect to the NIST webiste and download a `.html` file. This output must now be modified such that `html`-tags are removed and all header lines begin with #

4.4.1 Example

This is an example of how to download and generate datafiles for the `Filter.comp` and others.
The distributed tables have been extracted from the NIST x-ray database. To ease generation of more dtafiles from the same source a simple shell script: `MCXTAS/data/get_xray_db_data` is also distributed with McXtrace
Running this script will connect to the NIST webiste and download a `.html` file. This output must now be modified wuch that `html`-tags are removed and all header lines begin with #.

```
 /usr/local/lib/mcxtrace/data/get_xray_db_data 3 ouput.html
```

where the second parameter (3) is the atom number of the material, for which we want to generate a datafile. Now open the generated datafile (output.dat) with your favourite text editor and make sure the file ends up looking like this

```
#Li (Z 3)
#Atomic weight: A[r]  6.941000
#Nominal density: rho 5.3300E-01
#    [a](barns/atom) = [/](cm^2 g^-1)    1.15258E+01
#    E(eV) [/](cm^2 g^-1) = f[2](e atom^-1)    6.06257E+06
#    2 edges. Edge energies (keV):
#
```

File name	Description
Be.txt	Beryllium filter block
Si.txt	Silica filter block
Al.txt	Aluminium.txt

Table 4.1: Some material data file to be used with the Filter component

```
#
#    K      5.47500E-02  L I    5.34000E-03
#
#Relativistic correction estimate f[rel] (H82,3/5CL) = -9.8613E-04,
#    -6.0000E-04 e atom^-1
#    Nuclear Thomson correction f[NT] = -7.1131E-04 e atom^-1
#
#
#Form Factors, Attenuation and Scattering Cross-sections
#Z=3, E = 0.001 - 433 keV
#
#      E            f[1]          f[2]        [mu/rho]      [sigma/rho]      [mu/rho]      [m
#                                          Photoelectric Coh+inc      Total
#     keV        e atom^-1      e atom^-1   cm^2 g^-1       cm^2 g^-1      cm^2 g^-1   cm^2 g
5.233200E-03  9.08733E-01  0.0000E+00  0.0000E+00  2.3914E-07  2.3914E-07  0.000E+00  2.369E+
5.313300E-03  8.59283E-01  0.0000E+00  0.0000E+00  2.5404E-07  2.5404E-07  0.000E+00  2.333E+
5.334660E-03  8.03599E-01  0.0000E+00  0.0000E+00  2.5813E-07  2.5813E-07  0.000E+00  2.324E+
5.366700E-03  8.56971E-01  1.0769E-01  1.2165E+05  2.6435E-07  1.2165E+05  0.000E+00  2.310E+
.
.
.
3.788588E+02  3.00000E+00  3.9121E-08  6.2602E-07  8.4389E-02  8.4390E-02  6.123E-07  3.273E-
4.050001E+02  3.00000E+00  3.3438E-08  5.0054E-07  8.2127E-02  8.2128E-02  4.895E-07  3.061E-
4.329451E+02  3.00000E+00  2.8581E-08  4.0022E-07  7.9892E-02  7.9892E-02  3.913E-07  2.864E-
```

Please make sure you don't forget to remove the html-tags in the bottom of the file as well. In the future we will set up a more streamlined way of doing this.

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 5.4.

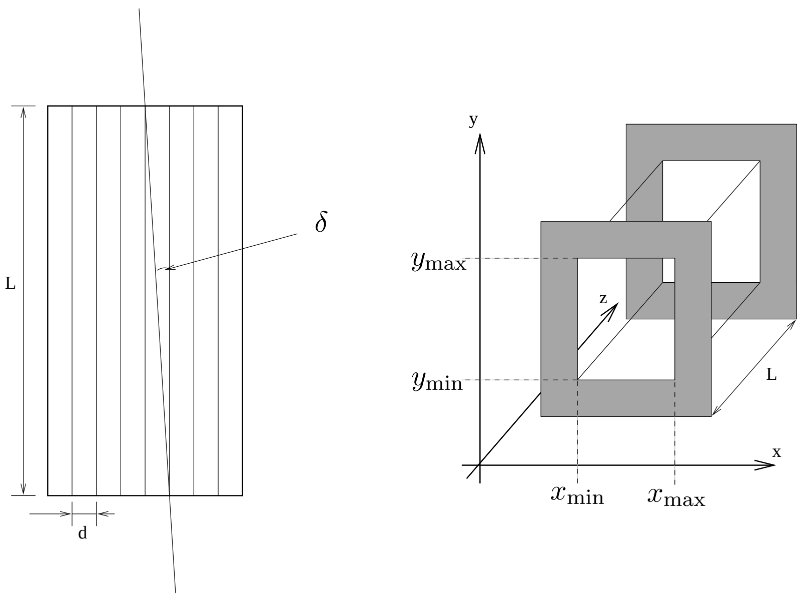

Figure 4.1: The geometry of a simple Soller blade collimators: The real Soller collimator, seen from the top (left), and a sketch of the component **Soller** (right). The symbols are defined in the text.

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

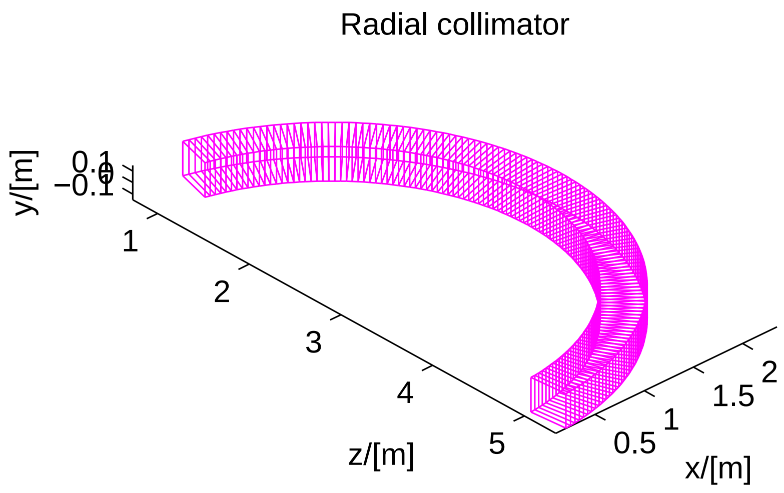

Figure 4.2: A radial collimator

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 them 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 $\pm w_1$, which supresses the channels shadow.
As an alternative, you may use the **Exact_radial_coll** contributed component. For a rectangular shaped collimator, instead of cylindrical/radial, you may use the Guide_channeled and the Guide_gravity components.

Chapter 5

Reflecting optical components: mirrors, and guides

This section describes advanced X-ray optics components such as mirrors and analyzer crystals. A description of the reflectivity of a mirror is found in section 5.2.2.

5.1 Mirror: The multilayer elliptic mirror

Name:	Multilayer Elliptic Mirror
Author:	System
Input parameters	θ, $s1$, $s2$, *length*, *width*, R
Optional parameters	
Notes	validated

The component **Multilayer Elliptic Mirror** models a single rectangular reflecting multilayer mirror plate with elliptical curvature. It can be used as a sample component, to *e.g.* assemble a Kirkpatrick-Baez focusing system or in combination with a double-crystal monochromator.
Figure 5.1*Left* shows a side view of a mirror (the blue section of the ellipse) in the McXtrace coordinate system. At the mirror center, the mirror tangent is parallel to the z axis and the mirror normal is parallel to the y axis. The width of the mirror is w and in $y - z$ plane the mirror has the curvature of an ellipse with major axis a and minor axis b,

$$\frac{z^2}{a^2} + \frac{y^2}{b^2} = 1\,,\ |x| < \frac{w}{2}\,. \tag{5.1}$$

The length of the mirror is L. The coordinates of the mirror center $(0, Y_0, Z_0)$ and the ellipse parameters a, b are determined uniquely by the central glancing angle, the source-mirror distance and the mirror-image distance. The position of the mirror is chosen to be at the positive side of the y axis.
The input parameters of this component are: θ [°], the incident angle; $s1$ [m], the distance from the source to the multilayer; $s2$ [m], the focusing distance of the multilayer; *length* [m], the length of the mirrors; *width* [m], the width of the mirror along the x-axis; R, the reflectivity.

5.1.1 Definition of the reference frames

The direction and position of the incoming photon is defined relative to the coordinate system illustrated in Fig. 5.1*Left* (in the code referred to as *McXtrace coordinate system*):

- the y-axis is parallel to the central mirror normal
- the z-axis is parallel to the central mirror tangent
- the origin is at the mirror center

However, all the calculations are conducted in another reference frame which is illustrated in Fig. 5.1 *Right*(in the following referred to as the *Ellipse coordinate system*):

- the z-axis is parallel to major axis of ellipse
- the y-axis is parallel to minor axis of ellipse
- the origin is at the center of the ellipse
- the mirror center at $(0, Y_0, Z_0)$, uniquely determined by the glancing angle at the mirror center, the source-mirror distance and mirror-image distance.

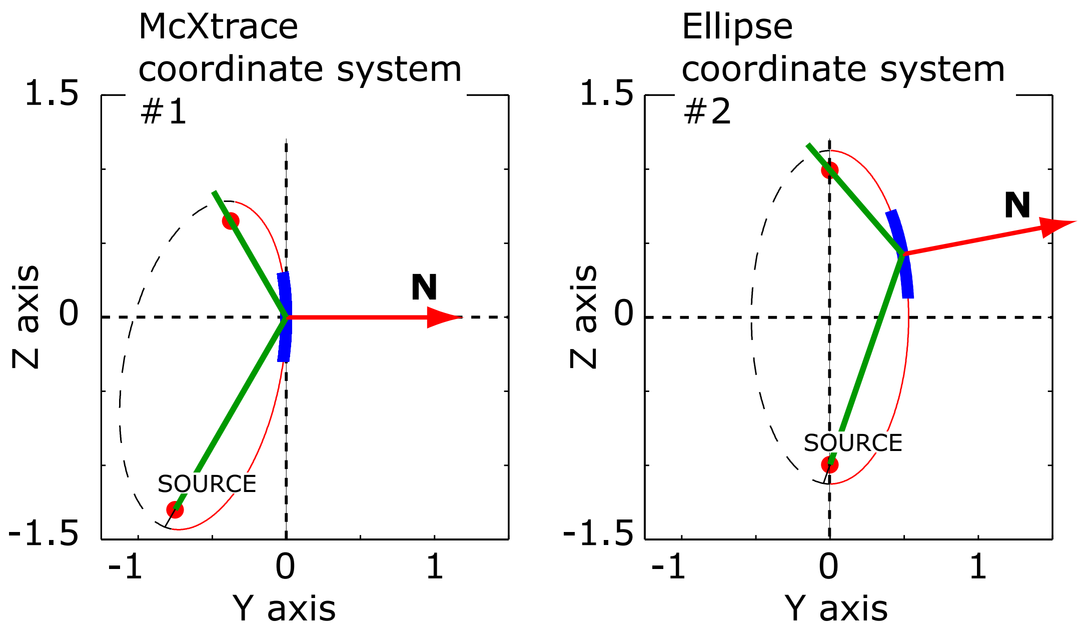

Figure 5.1: The same image in different coordinate systems.
Left: *McXtrace System* with the y-axis is parallel to the central mirror normal, the z-axis is parallel to the central mirror tangent and the origin is at the mirror center.
Right: *Ellipse System* with the z-axis parallel to major axis of ellipse, the y-axis is parallel to minor axis of ellipse and the origin is at the center of the ellipse.

5.1.2 Algorithm

1. The photon is generated with a starting point $\mathbf{S}$ and a direction $\mathbf{V}_{\text{in}}$ defined in the *McXtrace* coordinate system.
2. All calculations are performed in the *Ellipse* coordinate system, so to proceed the basis is changed to that reference frame.
3. The 2 intersections of the ray with the ellipse are determined.
4. It is checked if any of the intersections are within the area defined by the mirror.
5. If one of the solutions is valid, the reflection of that ray is determined.
6. The coordinates of the starting point and direction of the reflected ray are calculated using the basis of the *McXtrace* coordinate system.

5.2 Reflection of the ray in the mirror

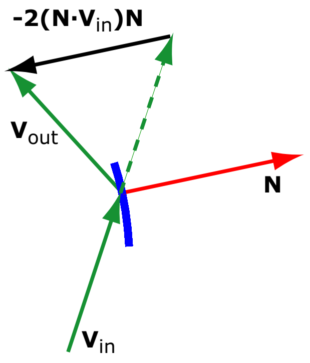

Figure 5.2: The reflection of the unit vector $\mathbf{V}_{\text{in}}$ in the mirror with the normal unit vector $\mathbf{N}$ is $\mathbf{V}_{\text{out}} = \mathbf{V}_{\text{in}} - 2(\mathbf{N} \cdot \mathbf{V}_{\text{in}})\mathbf{N}$

The tangent and normal to the ellipse $z^2/a^2 + y^2/b^2 = 1$ at the point (Y, Z) are found by implicit differentiation:

$$\frac{2z}{a^2} + \frac{2y}{b^2}\frac{dy}{dz} = 0\,, \tag{5.2}$$

so at the point (Y, Z) the slope of the tangent is $\frac{dy}{dz} = -\frac{Z\,b^2}{Y\,a^2}$. The slope of the normal is minus the inverse of the tangent slope, so the coordinates of the mirror normal are

$$N_x = 0 \quad N_y = \frac{a^2\,Y}{b^2\,Z} \quad N_z = 1\,. \tag{5.3}$$

With $\mathbf{V}_{\text{in}}$ and $\mathbf{N}$ denoting unit vectors (direction and normal respectively), the direction of the reflected ray is calculated as

$$\mathbf{V}_{\text{out}} = \mathbf{V}_{\text{in}} - 2(\mathbf{N} \cdot \mathbf{V}_{\text{in}})\mathbf{N} = \begin{pmatrix} V_{\text{in}x} - 2(\mathbf{N} \cdot \mathbf{V}_{\text{in}})N_x \\ V_{\text{in}y} - 2(\mathbf{N} \cdot \mathbf{V}_{\text{in}})N_y \\ V_{\text{in}z} - 2(\mathbf{N} \cdot \mathbf{V}_{\text{in}})N_z \end{pmatrix} \tag{5.4}$$

5.2.1 Mirror reflectivity

At present, the Multilayer Elliptic Mirror component uses a reflectivity table *reflect*, which 1st column is q [$\AA^{-1}$] and from the 2nd column on as the reflectivity R in [0-1] as function of tabulated energy [KeV]. An example file, calculated for a particular Si/W multilayer, is provided (`reflectivity.txt`). User provided reflectivity data files can be parsed by the component.

5.2.2 Mirror reflectivity calculation

To compute the reflectivity of the neutron supermirrors instead, we use an empirical formula derived from experimental data [10], see Fig. 5.3. The reflectivity is given by

$$R = \begin{cases} R_0 & \text{if } Q \leq Q_\mathrm{c} \\ \frac{1}{2}R_0(1 - \tanh[(Q - mQ_\mathrm{c})/W])(1 - \alpha(Q - Q_\mathrm{c})) & \text{if } Q > Q_\mathrm{c} \end{cases} \tag{5.5}$$

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

$$Q = |\mathbf{k_i} - \mathbf{k_f}| = \frac{m_\mathrm{n}}{\hbar}|\mathbf{v_i} - \mathbf{v_f}|, \tag{5.6}$$

m_n being the neutron mass. The number m in (5.5) 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 5.3.
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.
Alternatively, the Mirror, Guide and Guide_gravity components may use a reflectivity table *reflect*, which 1st column is q [$\AA^{-1}$] and 2nd column as the reflectivity R in [0-1]. For this purpose, we provide $m = 2$ and $m = 3$ reflectivity files from SwissNeutronics (`supermirror_m2.rfl` and `supermirror_m3.rfl` in `MCXTAS/lib/data/`).

5.2.3 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.

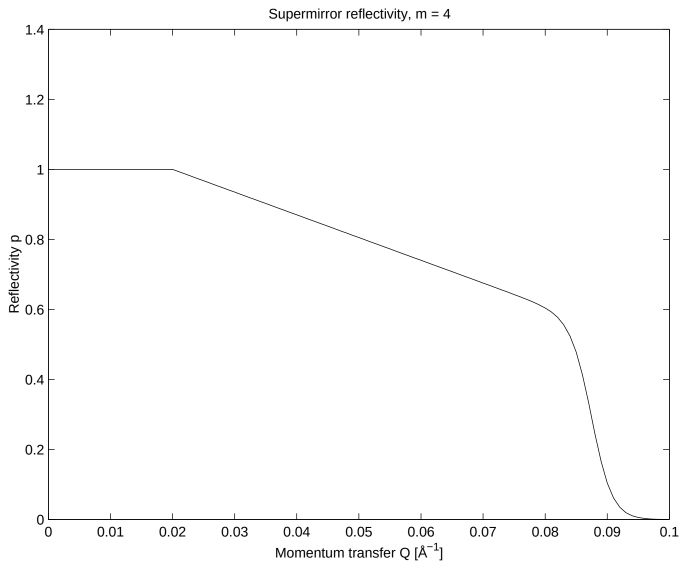

Figure 5.3: A typical reflectivity curve for a supermirror, Eq. (5.6). The used values are $m = 4$, $R_0 = 1$, $Q_\mathrm{c} = 0.02$ Å^{-1}, $\alpha = 6.49$ Å, $W = 1/300$ Å^{-1}.

5.3 Guide: The guide section

Name:	Guide
Author:	System
Input parameters	w_1, h_1, w_2, h_2, l, m, $reflect$
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 5.4. The reflecting properties are given by the values of R_0, m, Q_c, W, and α, as for **Mirror**, or alternatively from the reflectivity file $reflect$.
Guide may produce wrong results with gravitation support. Use **Guide_gravity** (section 5.5) in this case, or the **Guide_channeled** in section 5.4.

5.3.1 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:

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

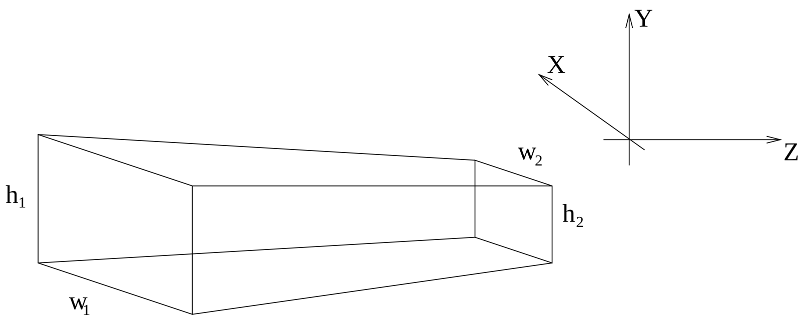

Figure 5.4: The geometry used for the guide component.

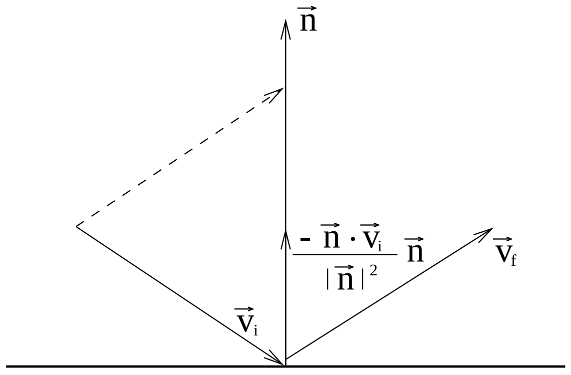

Figure 5.5: Neutron reflecting from mirror. $\mathbf{v}_\mathrm{i}$ and $\mathbf{v}_\mathrm{f}$ are the initial and final velocities, respectively, and $\mathbf{n}$ is a vector normal to the mirror surface.

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:

$$t^\alpha_\beta = \frac{(\mathbf{O}^\alpha_\beta - \mathbf{r}_0)\cdot\mathbf{n}^\alpha_\beta}{\mathbf{v}\cdot\mathbf{n}^\alpha_\beta}, \tag{5.7}$$

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

$$t_\mathrm{exit} = \frac{l - z_0}{v_z} \tag{5.8}$$

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

$$\mathbf{v}_\mathrm{f} = \mathbf{v}_\mathrm{i} - 2\mathbf{n}\cdot\frac{\mathbf{v}_\mathrm{i}}{|\mathbf{n}|^2}\mathbf{n} \tag{5.9}$$

This expression is arrived at by again considering the projection onto the mirror normal (see figure 5.5). The reflectivity of the mirror is taken into account as explained in section 5.1.

5.3.2 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.

5.4 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).

5.4.1 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.

5.4.2 Known problems

- This component may produce wrong results with gravitation support. Use Guide_gravity (section 5.5) 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).

5.5 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, rotating mode

This component is a variation of **Guide_channeled** (section 5.4) 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) of smaller rectangular guide channels.
The coating is specified as for the Guide and Mirror components by mean of the parameters R_0, m, Q_c, W, and α, or alternatively from the reflectivity file $reflect$.
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.
The component has also the capability to rotate at a given frequenccy in order to approximate a Fermi Chopper, including phase shift. The approximation resides in the fact that the component is considered fixed during neutron propagation inside slits. Beware that this component is then located at its entry window (not centered as the other Fermi choppers).
To activate gravitation support, either select the McXtrace gravitation support (`mcrun --gravitation ...` or from the Run dialog of `mcgui`), or set the gravitation field strength G (e.g. -9.81 on Earth).
This component is about 50 % slower than the `Guide` component, but has much more capabilities.
A contributed version Guide_honeycomb of this component exists with a honeycomb geometry.

5.6 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.

5.7 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. This may be automated with the `COPY` or the `JUMP ITERATE` mechanisms (see User manual).
An implementation of a continuous curved guide has been contributed as component Guide_curved.

Chapter 6

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 7.

6.1 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	In reflecting geometry, non polarized

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 McXtrace 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.

6.1.1 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

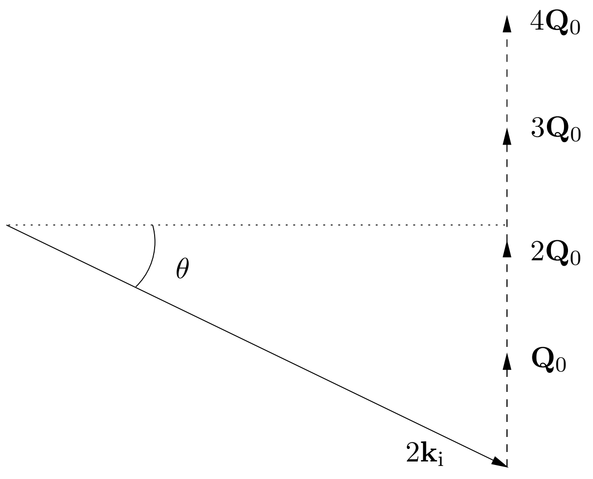

Figure 6.1: Selection of the Bragg order ("2" in this case).

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 7.2 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:

$$n\mathbf{Q}_0 = 2\mathbf{k}_i \sin\theta \tag{6.1}$$

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 6.1).
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 use α to compute the probability of reflection from the mosaic crystal

$$p_\mathrm{reflect} = R_0 e^{-\alpha^2/2\eta^2}, \tag{6.2}$$

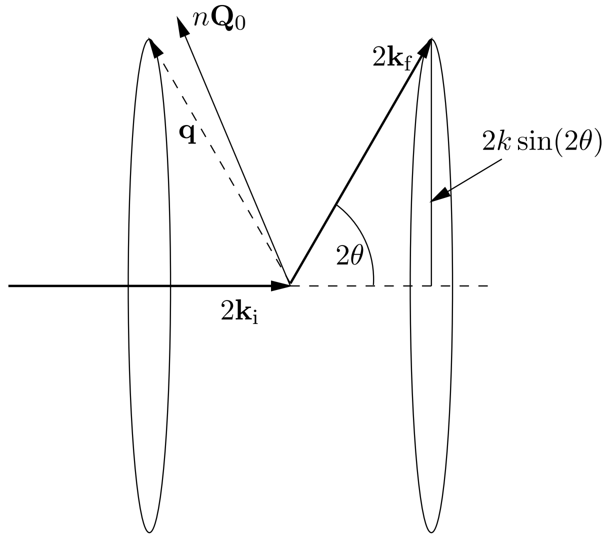

Figure 6.2: Scattering into the part of the Debye-Scherrer cone covered by the mosaic.

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

$$\mathbf{k}_\mathrm{f} = \mathbf{k}_\mathrm{i}\cos 2\theta + \sin 2\theta(\mathbf{c}\cos\varphi + \mathbf{b}\sin\varphi), \qquad \varphi \in [-\pi;\pi], \tag{6.3}$$

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 6.2). 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

$$\Delta q \approx \alpha 2k\sin\theta. \tag{6.4}$$

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

$$\Delta q \approx \varphi k\sin(2\theta) \tag{6.5}$$

(see figure 6.2). Hence we may sample φ from a Gaussian with standard deviation

$$\alpha\frac{2k\sin\theta}{k\sin(2\theta)} = \alpha\frac{2k\sin\theta}{2k\sin\theta\cos\theta} = \frac{\alpha}{\cos\theta} \tag{6.6}$$

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 $\frac{\alpha}{\cos\theta}$,

$$f_\mathrm{MC}(\varphi) = \frac{1}{\sqrt{2\pi}(\sigma/\cos\theta)}e^{-\varphi^2/2(\sigma/\cos\theta)^2} \tag{6.7}$$

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

$$\Pi(\varphi) = e^{\frac{-d(\varphi)^2}{2\sigma^2}} / \int_{-\pi}^{\pi} e^{\frac{-d(\varphi)^2}{2\sigma^2}} d\varphi \tag{6.8}$$

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

$$\pi_j = \Pi(\varphi)/f_{\mathrm{MC}}(\varphi). \tag{6.9}$$

In the implementation, the integral in (6.8) 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.

6.2 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	In reflecting geometry, non polarized

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 `MCXTAS/data`). In the same sense, the transmission can be modeled by a tabulated file *transmit* (for non-reflected neutrons, `*.trm` in `MCXTAS/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 variation 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**.

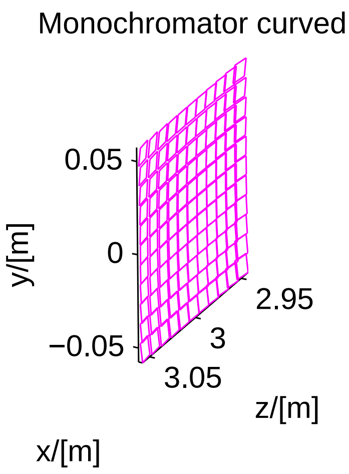

Figure 6.3: A curved monochromator

6.3 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 7.2.

6.4 Phase space transformer - moving monochromator

Eventhough there exist a few attempts to write dedicated phase space transformer components, there is an elegant way to put a monochromator into move, by mean of the EXTEND keyword. If you define a SPEED parameter for the instrument, the idea is to change the coordinate system before the monochromator, and restore it afterwards, as follow in the TRACE section:

```
DEFINE INSTRUMENT PST(SPEED=200, ...)
(...)
TRACE
(...)
COMPONENT Mono_PST_on=Arm()
AT ...
EXTEND %{
  vx = vx + SPEED; // monochromator moves transversaly by SPPED m/s
%}

COMPONENT  Mono=Monochromator(...)
AT (0,0,0) RELATIVE PREVIOUS

COMPONENT Mono_PST_off=Arm
AT (0,0,0) RELATIVE PREVIOUS
EXTEND  %{
  vz = vz - SPEED; // puts back neutron in static coordinate frame
%}
```

This solution does not contain acceleration, but is far enough for most studies, and it is very simple. In the latter example, the instance `Mono_PST_on` should itself be rotated to reflect according to a Bragg law.

Chapter 7

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. McXtrace contains both simple and detailed samples.

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, incoherent scattering, direct beam transmission and can assume *concentric* shape, i.e. can be used for modelling sample enviroments.

Next type is Bragg scattering from single crystals. The simplest single crystals are in fact the monochromator components like **Monochromator_flat**, presented in section 6.1. 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 **Saxs_Spheres**, which models scattering from a collection of hard spheres (dilute colloids).

7.0.1 Scattering notation

In sample components, we use a notation common for scattering experiments, where the wave vector transfer is denoted the *scattering vector*

$$\mathbf{q} \equiv \mathbf{k}_\mathrm{i} - \mathbf{k}_\mathrm{f}. \tag{7.1}$$

In analygo, the *energy transfer* is given by

$$\hbar\omega \equiv E_\mathrm{i} - E_\mathrm{f} = \frac{\hbar^2}{2m_\mathrm{n}}\left(k_\mathrm{i}^2 - k_\mathrm{f}^2\right). \tag{7.2}$$

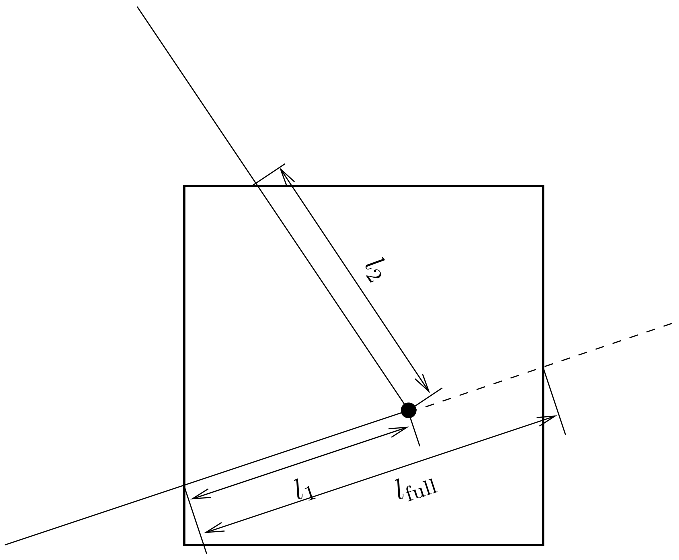

Figure 7.1: The geometry of a scattering event within a powder sample.

7.0.2 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* [11], 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 x-ray 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 7.1. 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

$$f_{\text{att}}(l) = \exp(-l(\mu^s + \mu^a)), \tag{7.3}$$

where the normalization $f_{\text{att}}(0) = 1$.
The probability for a given x-ray to be scattered from within the interval $[l_1; l_1 + dl]$ will be

$$P(l_1)dl = \mu^s f_{\text{att}}(l_1)dl, \tag{7.4}$$

while the probability for a x-ray 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

$$P(l_1, \Omega)dld\Omega = \mu^s f_{\text{att}}(l_1) f_{\text{att}}(l_2)\gamma(\Omega)d\Omega dl, \tag{7.5}$$

where $\gamma(\Omega)$ is the directional distribution of the scattered x-rays, and l_2 is determined by Monte Carlo chocies of l_1, Ω, and from the sample geometry, see e.g. figure 7.1.

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

$$f(l_1, \Omega) = g(\Omega)/l_{\text{full}}, \tag{7.6}$$

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

$$\pi_i(l_1, \Omega) = \mu^s l_{\text{full}} \exp\left[-(l_1 + l_2)(\mu^a + \mu^s)\right] \frac{\gamma(\Omega)}{g(\Omega)}. \tag{7.7}$$

In analogy with the source components, it is possible to define "interesting" directions for the scattering. One will then try to focus the scattered x-rays, choosing a $g(\Omega)$, which peaks around these directions. To do this, one uses (7.7), 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.

7.0.3 Future development of sample components

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

7.1 PowderN: A general powder sample

Name:	Powder_N
Author:	System
Input parameters	*radius*, *thickness*, *h*, *xwidth*, *yheight*, *zdepth*, σ_{abs}, σ_{inc}, Vc, f_{pack}, reflections, format, DW, concentic, and more
Optional parameters	
Notes	

The powder diffraction component **PowderN** models a powder sample with background coming only from incoherent scattering and no multiple scattering. At the users choice, a given percentage of the incoming events may be transmitted (attenuated) to model the direct beam. The component can also assume *concentric* shape, i.e. be used for describing sample environment (cryostat, sample container etc.).
The description of the powder comes from a file in one of the standard output formats LAZY, FULLPROF, or CRYSTALLOGRAPHICA.

7.1.1 Files formats: powder structures

Data files of type `lau` and `laz` in the McXtrace distribution data directory are self-documented in their header. A list of common powder definition files is available in Table 1.2 (page 9). 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.
The column signification may as well explicitely be set in the data file header using any of the lines:

```
#column_j     <index of the multiplicity 'j' column>
#column_d     <index of the d-spacing 'd' column>
#column_F2    <index of the squared str. factor '|F|^2' column [b]>
#column_F     <index of the structure factor norm '|F|' column>
#column_DW    <index of the Debye-Waller factor 'DW' column>
#column_Dd    <index of the relative line width Delta_d/d 'Dd' column>
#column_inv2d <index of the 1/2d=sin(theta)/lambda 'inv2d' column>
#column_q     <index of the scattering wavevector 'q' column>
```

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

```
#V_rho        <value of atom number density [at/Angs^3]>
#Vc           <value of unit cell volume Vc [Angs^3]>
#sigma_abs    <value of Absorption cross section [barns]>
#sigma_inc    <value of Incoherent cross section [barns]>
#Debye_Waller <value of Debye-Waller factor DW>
#Delta_d/d    <value of Detla_d/d width for all lines>
#density      <value of material density [g/cm^3]>
#weight       <value of material molar weight [g/mol]>
#nb_atoms     <value of number of atoms per unit cell>
```

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

7.1.2 Geometry, physical properties, concentricity

The sample has the shape of a solid cylinder, radius r and height h or a box-shaped sample of size *xwidth* x *yheight* x *zdepth*. At the users choice, an inner 'hollow' can be specified using the parameter *thickness*.
As the Isotropic_Sqw component **??**, PowderN assumes *concentric* shape, i.e. can contain other components inside the inner hollow. To allow this, two almost identical copies of the PowderN components must be set up *around* the internal component(s), for example:

```
COMPONENT Cryo = PowderN(reflections="Al.laz", radius = 0.01, thickness = 0.001,
                          concentric = 1)
AT (0,0,0) RELATIVE Somewhere
```

```
COMPONENT Sample = some_other_component(with geometry FULLY enclosed in the hollow)
AT (0,0,0) RELATIVE Somewhere

COMPONENT Cryo2 = COPY(Cryo)(concentric = 0)
AT (0,0,0) RELATIVE Somewhere
```

As outlined, the first instance of PowderN *must* have `concentric = 1` and the instance *must* have `concentric = 0`. Furthermore, the component(s) inside the hollow *must* have a geometry which can be fully contained inside the hollow.
In addition to the coherent scattering specified in the `reflections` file, absorption- and incoherent cross sections can be given using the input parameters σ_c^a and σ_i^s.
The Bragg scattering from the powder, σ_c^s 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 Vc, while the sample packing factor is f_{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, $\mathbf{v}_\mathrm{i}$, and a user-specified vector, $\mathbf{f}$.

7.1.3 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

$$n\lambda = 2d\sin\theta, \tag{7.8}$$

where n is the order of the scattering (an integer), λ is the x-ray wavelength, d is the lattice spacing of the sample, and 2θ is the scattering angle, see figure 7.2. As all crystal orientations are realised in a powder sample, the x-rays are scattered within a *Debye-Scherrer cone* of opening angle 4θ [11].
Equation (7.8) may be cast into the form

$$|\mathbf{Q}| = 2|\mathbf{k}|\sin\theta, \tag{7.9}$$

where $\mathbf{Q}$ is a vector of the reciprocal lattice, and $\mathbf{k}$ is the wave vector of the x-ray. 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

$$\left|\sum_j b_j \exp(\mathbf{R}_j \cdot \mathbf{Q})\right|^2, \tag{7.10}$$

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.

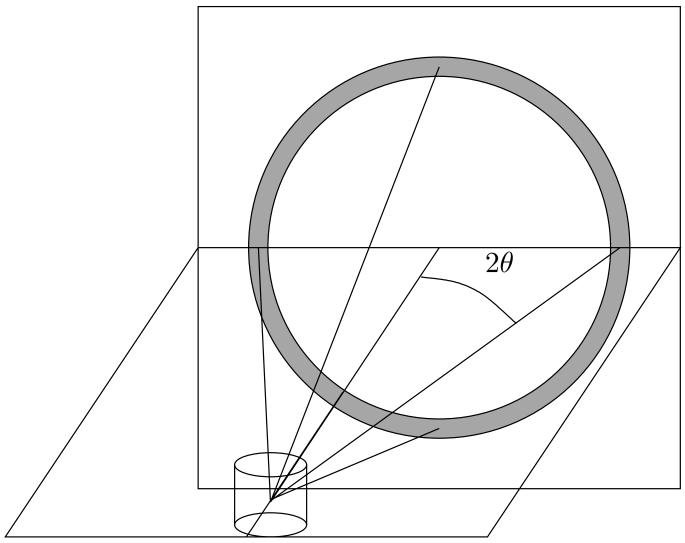

Figure 7.2: The scattering geometry of a powder sample showing part of the Debye-Scherrer cone (solid lines) and the Debye-Scherrer circle (grey).

The textbook expression for the scattering cross section corresponding to one Debye-Scherrer cone reads [12, ch.3.6], with $V = NV_0$ being the total sample volume:

$$\sigma_{\text{cone}} = \frac{V}{V_0^2} \frac{\lambda^3}{4 \sin\theta} \sum_Q |F(Q)|^2. \tag{7.11}$$

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

$$\sigma_{\text{cone,Q}} = j_Q f \exp(-2W) \frac{V}{V_0^2} \frac{\lambda^3}{4 \sin\theta} |F(Q)|^2 \tag{7.12}$$

$$= f \exp(-2W) \frac{N}{V_0} \frac{4\pi^3}{k^2} \frac{j_Q |F(Q)|^2}{Q} \tag{7.13}$$

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

$$\mu_{\text{Q}} = \sigma_{\text{cone,Q}}/V. \tag{7.14}$$

For calibration it may be useful to consider the total intensity scattered into a detector of effective height h, covering only one reflection [12, 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 x-rays, giving a resulting count intensity:

$$I = \Psi \sigma_{\mathrm{cone,Q}} \frac{h}{2\pi r \sin(2\theta)}, \tag{7.15}$$

where Ψ is the flux at the sample position.
For clarity we repeat the meaning and unit of the symbols:

Ψ	$\mathrm{s}^{-1}\mathrm{m}^{-2}$	Incoming intensity of x-rays
I	s^{-1}	Detected intensity of x-rays
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	m^3	Volume of unit cell
$\lvert F(\mathbf{Q})\rvert^2$	m^2	Structure factor
$\exp(-2W)$	1	Debye-Waller factor
μ_{Q}	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 $\mu^s = \mu^s_{\mathrm{inc}} + \sum_j \mu^s_j$, where μ^s_{inc} represents the incoherent scattering.

7.1.4 Algorithm

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

- Check if the x-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.

7.2 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	Pending documentation

7.3 Saxs_spheres: A sample of hard spheres for small-angle scattering

Name:	Saxs_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_{det}, d
Optional parameters	
Notes	Pedning documentation

7.4 Absorpion_phantom: A sample used in tomography type experiments

Name:	Absorption_phantom
Author:	System
Input parameters	
Optional parameters	
Notes	

Pending documentation

Chapter 8

Monitors and detectors

In real scattering experiments, detectors and monitors play quite different roles. One wants the detectors to be as efficient as possible, counting all photons (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 photons passing by. In computer simulations, it is of course possible to detect every xray 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 xray 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.

When a monitor detects a xray, a number counting variable is incremented: $n_i = n_{i-1} + 1$. In addition, the photon 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 photonts/sec.) is I_N, while the estimated errorbar is $\sqrt{M_{2,N}^2}$.

Several different monitor components have been developed for McXtrace, 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, a few lines of code in existing monitors can easily be modified.

Another solution is the "Swiss army knife" of monitors, **Monitor_nD**, that can handle almost any simulation requirement, but may prove challenging for inexperienced users or users who like to make their own modifications.

8.1 TOF_monitor: The time-of-flight monitor

Name:	TOF_monitor
Author:	System
Input parameters	x_{width}, y_{height}, y_{max}, n_{t}, t_0, E_1, filename
Optional parameters	x_{min},x_{max}, y_{min},y_{max},restore_xray
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 xray is propagated to the plane of the monitor by the kernel call PROP_Z0. An xray 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 xray. 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_{t} time intervals begin at t_0 and end at t_1 (alternatively, the interval length is specified by Δt). 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 sets of data divided in TOF bins. This file can be read and plotted by the mxplot tool; for details see the System Manual.

8.2 TOF2E_monitor: A time-of-flight monitor with simple energy analysis

Name:	E_monitor
Author:	System
Input parameters	x_{min}, x_{max}, y_{min}, y_{max}, n_{chan}, E_{min}, E_{max}, t_0, L_{flight}, filename
Optional parameters	
Notes	Not validated

The component **TOF2E_monitor** resembles **TOF_monitor** to a very large extent. Only this monitor converts the neutron flight time to energy - as would be done in an experiment. The *apparent* neutron energy, E_{app} is calculated from the apparent velocity, given by

$$v_{\mathrm{app}} = \frac{L_{\mathrm{flight}}}{t - t_0}, \qquad (8.1)$$

where the time offset, t_0 defaults to zero. E_{app} is binned in *nchan* bins between E_{min} and E_{max} (in meV).

The output parameters from **TOF2E_monitor** are the total counts, and a file with 1-dimensional data vs. E_{app}, similar to **TOF_monitor**.

8.3 E_monitor: The energy-sensitive monitor

Name:	E_monitor
Author:	System
Input parameters	x_{width}, y_{height}, $y_{\max}$, n_{E}, $E_{\min}$, $E_{\max}$, filename
Optional parameters	$x_{\min}$,$x_{\max}$, $y_{\min}$,$y_{\max}$,restore_xray
Notes	,

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

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

8.4 L_monitor: The wavelength sensitive monitor

Name:	L_monitor
Author:	System
Input parameters	x_{width}, y_{height}, $y_{\max}$, n_{λ}, $\lambda_{\min}$, $\lambda_{\max}$, filename
Optional parameters	$x_{\min}$,$x_{\max}$, $y_{\min}$,$y_{\max}$,restore_xray
Notes	,

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

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

8.5 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 **mcplot**, see the system manual.

8.6 Divergence_monitor: A divergence sensitive monitor

Name:	Divergence_monitor
Author:	System
Input parameters	$x_{\rm min}$, $x_{\rm max}$, $y_{\rm min}$, $y_{\rm max}$, $n_{\rm v}$, $n_{\rm h}$, $\eta_{\rm v,max}$, $\eta_{\rm h,max}$, 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 $\eta_{\rm h} = \tan^{-1}(v_x/v_z)$ and $\eta_{\rm v} = \tan^{-1}(v_y/v_z)$. The neutron counts are being histogrammed into $n_{\rm v} \times n_{\rm h}$ pixels. The divergence range accepted is in the vertical direction $[-\eta_{\rm v,max}; \eta_{\rm v,max}]$, 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: $n(\eta_{\rm v}, \eta_{\rm h}), I(\eta_{\rm v}, \eta_{\rm h}), M_2(\eta_{\rm v}, \eta_{\rm h})$. The arrays are written to a file, `filename`, and can be read e.g. by the tool **MC_plot**, see the system manual.

8.7 DivPos_monitor: A divergence and position sensitive monitor

Name:	DivPos_monitor
Author:	System
Input parameters	$x_{\rm min}$, $x_{\rm max}$, $y_{\rm min}$, $y_{\rm max}$, n_x, $n_{\rm h}$, $\eta_{\rm h,max}$, 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 $\eta_{\rm h} = \tan^{-1}(v_x/v_z)$. The detector window is set by the x and y input coordinates.
The neutron counts are being histogrammed into $n_x \times n_{\rm h}$ pixels. The horizontal divergence range accepted is $[-\eta_{\rm h,max}; \eta_{\rm h,max}]$, 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: $n(x, \eta_{\rm h}), I(x, \eta_{\rm h}), M_2(x, \eta_{\rm h})$. The arrays are written to a file and can be read e.g. by the tool **mcplot**, see the system manual.
This component can be used for measuring acceptance diagrams [13]. **PSD_monitor** can easily be changed into being sensitive to y and vertical divergence by a 90 degree rotation around the z-axis.

8.8 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	$file$, $x_{width}, y_{height}, z_{depth}$, $bins$, 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 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 $x_{width}, y_{height}, z_{depth}$ 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`.

8.8.1 The Monitor_nD geometry

The monitor shape can be selected among seven geometries:

1. (*square*) The default geometry is flat rectangular in (xy) plane with dimensions $x_{\min}, x_{\max}, y_{\min}$, $y_{\max}$, or x_{width}, y_{height}.
2. (*box*) A rectangular box with dimensions $x_{width}, y_{height}, z_{depth}$.
3. (*disk*) When choosing this geometry, the detector is a flat disk in (xy) plane. The radius is then
$$\text{radius} = \max(\text{abs }[x_{\min}, x_{\max}, y_{\min}, y_{\max}, x_{width}/2, y_{height}/2]). \tag{8.2}$$
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
$$\text{radius} = \max(\text{abs }[x_{\min}, x_{\max}, x_{width}/2]), \tag{8.3}$$
and the height along y is
$$\text{height} = |y_{\max} - y_{\max}| \text{or} y_{height}. \tag{8.4}$$

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`.
7. (*previous*) The detector has the shape of the previous component. This may be a surface or a volume. In this case, the neutron is detected on previous component, and there is not neutron propagation.

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).
The *parallel* option keyword is of common use in the case where the **Monitor_nD** is superposed with other components. It ensures that neutrons are detected independently of other geometrical constrains. This is generally the case when you need e.g. to place more than one monitor at the same place.

8.8.2 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 user3           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.

8.8.3 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 (up to the cadmium cut-off):

$$\Phi_c = \int_0^{0.5eV} \frac{d\Phi}{d\lambda} \frac{\lambda}{\lambda_{2200m/s}} d\lambda \tag{8.5}$$

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.

8.8.4 The output files

By default, the file names will be the component name, followed by a time stamp and 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.

8.8.5 Monitor equivalences

In the following table 8.1, we show how the Monitor_nD may substitute any other McStas monitor.

8.8.6 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.

8.8.7 Monitoring user variables

There are two ways 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 transfer at a sample. The only requirement is to define the `user1` (and optionally `user2,user3`) variables of a given Monitor_nD instance.

McStas monitor	Monitor_nD equivalent
Divergence_monitor	*options*=”dx bins=*ndiv* limits=$[-\alpha/2\alpha/2]$, lambda bins=*nlam* limits=$[\lambda_0\ \lambda_1]$ file=$file$”
DivLambda_monitor	*options*=”dx bins=*nh* limits=$[-h_{max}/2h_{max}/2]$, dy bins=*nv* limits=$[-v_{max}/2v_{max}/2]$” *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=$[-h_{max}/2h_{max}/2]$” *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*=”theta 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=$[\lambda_0\ \lambda_1]$, 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 8.1: 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).

Setting directly the user variables (simple)

The first method uses directly the `user1` and `username1` component parameters to transfert directly the value and label, such as in the following example:

```
TRACE
(...)
COMPONENT UserMonitor = Monitor_nD(
  user1    = log(t), username1="Log(time)",
  options  ="auto user1")
```

The values to assign to `user2` and `user3` must be global instrument variables, or a component output variables as in `user1=MC_GETPAR(some_comp, outpar)`. Similarly, the `user2,user3` and `username2,username3` parameters may be used to control the second and third user variable, to produce eventually 2D/3D user variable correlation data and custom event lists.

Setting indirectly the user variables (only for professionals)

It is possible to control the user variables of a given Monitor_nD instance anywhere in the instrument description. This method requires more coding, but has the advantage that a variable may be defined to store the result of a computation locally, and then transfert it into the UserMonitor, all fitting in an EXTEND block.
This is performed in a 4 steps process:

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

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

2. Initialize the label of the user variable (optional):

   ```
   INITIALIZE
   %{
     (...)
     MONND_USER_TITLE(UserMonitor, 1, "Log(time)");
   %}
   ```

   The value '1' could be '2' or '3' for the `user2,user3` variable.

3. Set the user variable value in a TRACE component EXTEND block:

   ```
   TRACE
   (...)
   COMPONENT blah = blah_comp(...)
   ```

```
EXTEND
%{  // attach a value to user1 in UserMonitor, could be much more comlex here.
  MONND_USER_VALUE(UserMonitor, 1, log(t));
%}
(...)
```

4. 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), access any component OUTPUT parameter using the `MC_GETPAR` C macro(see chapter A), or simply use a global instrument variable. Instrument parameters can not be used directly.

Example: Number of neutron bounces in a guide

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;
%}
TRACE
(...)
COMPONENT Guide_in = Arm() AT (...)
EXTEND
%{
  nbounces = 0;
%}

COMPONENT Guide1 = Guide_gravity(...) AT (...) RELATIVE PREVIOUS
EXTEND
%{
  if (SCATTERED) nbounces += GVars.N_reflection[0];
%}
(... many guide instances, copy/paste and change names automatically ...)
COMPONENT COPY(Guide1) = COPY(Guide1) AT (...) RELATIVE PREVIOUS
EXTEND
```

```
%{
  if (SCATTERED) nbounces += GVars.N_reflection[0];
%}

// monitor nbounces
COMPONENT UserMonitor = Monitor_nD(
  user1=nbounces, username1="Number of bounces",
  options="auto user1") AT (...)
(...)
```

8.8.8 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 8.8.7 in an EXTEND block.

Chapter 9

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 McXtrace 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.

9.1 Virtual_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 continues 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. Exept when using MPI, 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.

9.2 Virtual_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 continues 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 (see Appendix 2).
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* McXtrace simulation events. We recommand to control the simulation on `repeat-count` by using a very larger ncount value.

9.3 Res_sample: A sample-like component for resolution calculation

Name:	Res_sample
Author:	(System); Alan Tennant, HMI
Input parameters	r, r, h, r_{focus}, x_{target}, y_{target}, z_{target}, E_0, ΔE
Optional parameters	x_w, y_h, z_d, x_{focus}, y_{focus}, $a_{\text{v,focus}}$, $a_{\text{h,focus}}$, 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 9.5.

The shape of Res_sample is either a hollow cylinder or a rectangular box. The hollow cylinder shape is specified with the outer radius, r and thickness, respectively, and the height, h. If these parameters are unspecified, the shape is instead a box of dimensions x_w, y_h, and z_d. 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_{focus}, within a rectangular target with measures $(x_{\text{focus}}, y_{\text{focus}})$ 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 which 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 9.5.

9.4 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 9.5.
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**.

9.5 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 McXtrace 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 9.3, 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.

9.6 Progress_bar: Simulation progress and automatic saving

Name:	Progress_bar
Author:	System
Input parameters	percent, flag_save, profile
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.
The optional 'profile' parameter, when set to a file name, will produce the number of statistical events reaching each component in the simulation. This may be used to identify positions where events are lost.

9.7 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 McXtrace 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 `MCXTRACE` 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` McXtrace 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 (`mcxtrace-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 x-rays from one point to an other. Except when using the special `ALLOW_BACKPROP;` call prior to exectuting any `PROP_*` propagation, the x-rays which have negative propagation times are removed automatically.

- **ABSORB**. This macro issues an order to the overall McXtrace simulator to interrupt the simulation of the current x-ray history and to start a new one.

- **PROP_Z0**. Propagates the x-ray to the $z = 0$ plane, by adjusting (x, y, z) and t accordingly from knowledge of the x-ray wavevector (k_x, k_y, k_z). If the propagation time is negative, the x-ray is absorbed, except if a `ALLOW_BACKPROP;` preceeds it.

  For components that are centered along the z-axis, use the `_intersect` functions to determine intersection time(s), and then a `PROP_DL` call.

- **PROP_X0,PROP_Y0**. These macros are analogous to `PROP_Z0` except they propaget to the $x = 0$ and $y = 0$ planes respectively.

- **PROP_DL**(*dl*). Propagates the x-ray by the length dl, adjusting (x, y, z) and ϕ accordingly, from knowledge of the x-ray wavevector.

- **ALLOW_BACKPROP**. Indicates that the next propagation routine will not remove the x-ray, even if negative propagation lengths 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 x-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**(*comp*, *outpar*). This may be used in e.g. 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_XRAY**($index, x, y, z, kx, ky, kz, phi,\ Ex, Ey, Ez, p$) stores the current x-ray 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_XRAY**($index, x, y, z, kx, ky, kz, phi, Ex, Ey, Ez, p$) restores the x-ray state to the one at the input of the component *index*. To ignore a component effect, use RESTORE_XRAY (INDEX_CURRENT_COMP, $x, y, z, kx, ky, kz, phi, Ex, Ey, Ez, p$) at the end of its TRACE section, or in its EXTEND section. These x-ray 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 x-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 x-ray histories to simulate to n.
- **mcget_ncount**(). Returns the number of x-ray histories to simulate (usually set by option `-n`).
- **mcget_run_num**(). Returns the number of x-ray 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,φ,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 McXtrace 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 McXtrace front-end programs.
- **DETECTOR_OUT_1D**(...). Used to output the results from a one-dimensional detector. Integrated intensities error etc. is also reported as for DETECTOR_OUT_0D.
- **DETECTOR_OUT_2D**(...). Used to output the results from a two-dimentional detector. Integrated intensities error etc. is also reported as for DETECTOR_OUT_0D.
- **DETECTOR_OUT_3D**(...). Used to output the results from a three-dimentional detector. Arguments are the same as in DETECTOR_OUT_2D, but with an additional z axis. Resulting data files are treated as 2D data, but the 3rd dimension is specified in the *type* field. Integrated intensities error etc. is also reported as for DETECTOR_OUT_0D.

- **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

- **inside_rectangle**(&x, &y, x, xw, yh). Return 1 if $-xw/2 \leq x \leq xw/2$ AND $-yh/2 \leq y \leq yh/2$. Else return 0.

- **box_intersect**(&l_1, &l_2, x, y, z, k_x, k_y, k_z, d_x, d_y, d_z). Calculates the (0, 1, or 2) intersections between the x-ray path and a box of dimensions d_x, d_y, and d_z, centered at the origin for a x-ray with the parameters (x, y, z, k_x, k_y, k_z). The intersection lengths are returned in the variables l_1 and l_2, with $l_1 < l_2$. In the case of less than two intersections, t_1 (and possibly t_2) are set to zero. The function returns true if the x-ray intersects the box, false otherwise.

- **cylinder_intersect**(&l_1, &l_2, x, y, z, k_x, k_y, k_z, r, h). Similar to **box_intersect**, but using a cylinder of height h and radius r, centered at the origin.

- **sphere_intersect**(&l_1, &l_2, x, y, z, k_x, k_y, k_z, r). Similar to **box_intersect**, but using a sphere of radius r.

- **ellipsoid_intersect**(&l_1, &l_2, x, y, z, k_x, k_y, k_z, a,b,c,Q,). Similar to **box_intersect**, but using an ellipsoid with half-axis a,b,c oriented by the rotation matrix Q. If $Q = I$, a is along the x-axis, b along y and c along z

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 sample the normal distribution is explained in Ref. [14, ch.7].

- **randpm1**(). Returns a random number distributed uniformly between -1 and 1.

- **randtriangle**(). Returns a random number from a triangular distribution 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 x-ray 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 and return an empty table. 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.

- **Table_Free**(&*Table*) free allocated memory blocks.

- **Table_Value2d**(*Table*, X, Y) Uses 2D linear interpolation on a Table, from (X,Y) coordinates and returns the corresponding value.

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 is 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 'MCXTAS/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 McXtrace ' chapter in the User Manual), and if not found, in the data sub-directory of the MCXTAS library location. This way, you do not need to have local copies of the McXtrace 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
n = Table_Info_Array(Table);     // display informations for all blocks (also retu

y = Table_Index(Table[0], 2,5);  // read in 1st block the 3rd row, 6th column elem
                                 // 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 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
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, $m_{\rm n}$	
HBAR	$1.05459 \cdot 10^{-34}$ Js	Planck constant, $\hbar$	
PI	3.14159265...	π	
FLT_MAX	3.40282347E+38F	a big float value	
CELE	1.602176487e-19	Elementary charge (C)	
M_C	299792458	Speed of light in vacuum (m/s)	
E2K	0.506773091264796	Wavenumber (1/AA)	Energy (keV)
K2E	1.97326972808327	Energy (keV)	Wavenumber (1/AA)

Appendix B

The McXtrace terminology

This is a short explanation of phrases and terms which have a specific meaning within McXtrace. 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 McXtrace core team.

- **Arm** A generic McXtrace component which defines a frame of reference for other components.
- **Component** One unit (*e.g.* optical element) in a neutron spectrometer. These are considered as Types of elements to be instantiated in an Instrument description.
- **Component Instance** A named Component (of a given Type) inserted in an Instrument description.
- **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 McXtrace components defining a neutron spectrometer.
- **Kernel** The McXtrace language definition and the associated compiler
- **McXtrace** 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 McXtrace 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] See http://www.mcxtrace.org.

[3] ICSD, Inorganic Crystal Structure Database. See http://icsd.ill.fr.

[4] A.-J. Dianoux and G. Lander. *ILL Neutron Data Booklet.* OCP Science, 2003.

[5] See http://www.mcstas.org.

[6] 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.

[7] See http://neutron.risoe.dk/McZilla/.

[8] F. James. *Rep. Prog. Phys.*, **43**, 1145, 1980.

[9] Grimmett, G. R., and Stirzakerand D. R. *Probability and Random Processes, 2nd Edition.* Clarendon Press, Oxford, 1992.

[10] 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.

[11] G.E. Bacon. *Neutron Diffraction.* Oxford University Press, 1975.

[12] G.L. Squires. *Thermal Neutron Scattering.* Cambridge University Press, 1978.

[13] L. D. Cussen. *J. Appl. Cryst.*, **36**, 1204, 2003.

[14] 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(rev.ed.)(EN)**

Title and author(s)

Component Manual to the Neutron Ray-Tracing Package McXtrace, Version 1.0_rc1

Peter Kjær Willendrup, Erik Knudsen, Kim Lefmann and Emmanuel Farhi

ISBN	ISSN
ISBN 978–87–550–3680–2	0106–2840

Dept. or group	Date
Materials Research Department	Mar, 2011

Groups own reg. number(s)	Project/contract No.
—	—

Pages	Tables	Illustrations	References
90	2	15	10

Abstract (Max. 2000 char.)

The software package McXtrace is a tool for carrying out Monte Carlo ray-tracing simulations of xray scattering beamlines 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 for e.g. training, experimental planning or data analysis. McXtrace is based is based on a unique design, inhereted from its sister McStas, 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.

Descriptors

X-Ray Instrumentation; Monte Carlo Simulation; Software

Available on request from:
Information Service Department, Risø DTU
(Afdelingen for Informationsservice, Risø DTU)
P.O. Box 49, DK–4000 Roskilde, Denmark
Phone +45 4677 4004, Telefax +45 4677 4013