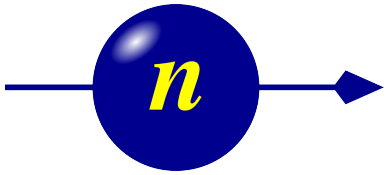
Physics Physics Department,
Technical University of Denmark
2800 Kongens Lyngby, Denmark

Component Manual for the Neutron Ray-Tracing Package McStas, version 3.0



P. Willendrup, E. Farhi, E. Knudsen, U. Filges, K. Lefmann

December 15th, 2020

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

This report constitutes the reference manual for McStas, and, together with the manual for the McStas components, it contains documentation of most aspects of the program. It covers the various ways to compile and run simulations, a description of the meta-language used to define simulations, and some example simulations performed with the program.

This report documents McStas version 3.0, released December 15th, 2020

The authors are:

Peter Kjær Willendrup <pkwi@fysik.dtu.dk>
Physics Department, Technical University of Denmark, Kongens Lyngby, Denmark

Emmanuel Farhi <farhi@ill.fr>
Institut Laue-Langevin, Grenoble, France

Erik Knudsen <erkn@fysik.dtu.dk>
Physics Department, Technical University of Denmark, Kongens Lyngby, Denmark

Jakob Garde <jaga@fysik.dtu.dk>
Physics Department, Technical University of Denmark, Kongens Lyngby, Denmark

Kim Lefmann <lefmann@nbi.dk>
Niels Bohr Institute, University of Copenhagen, Denmark

as well as authors who left the project:

Peter Christiansen <pchristi@hep.lu.se>
Materials Research Department, Risø National Laboratory, Roskilde, Denmark
Present address: University of Lund, Lund, Sweden
Klaus Lieutenant <klaus.lieutenant@helmholtz-berlin.de>
Institut Laue-Langevin, Grenoble, France
Present address: Helmotlz Zentrum Berlin, Germany
Kristian Nielsen <kristian-nielsen@mail.tele.dk>
Materials Research Department, Risø National Laboratory, Roskilde, Denmark
Presently associated with: MySQL AB, Sweden

ISBN 978–87–550–3680–2
ISSN 0106–2840

Physics Department · DTU · 2020

Contents

Preface and acknowledgements

This document contains information on the neutron scattering components which are the building blocks for defining instruments in the Monte Carlo neutron ray-tracing program McStas version 3.0. The initial release in October 1998 of version 1.0 was presented in Ref. [LN99] and further developed though version 2.0 as presented in Ref. [Wil+14]. The reader of this document is not supposed to have specific knowledge of neutron scattering, but some basic understanding of physics is helpful in understanding the theoretical background for the component functionality. For details about setting up and running simulations, we refer to the McStas system manual [Wil+05]. We assume familiarity with the use of the C programming language.

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

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

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

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

The McStas project has been supported by the European Union through"XENNI / Cool Neutrons" (FP4), "SCANS" (FP5), "nmi3/MCNSI" (FP6), "'nmi3-ii/E-learning" and "nmi3-ii/MCNSI7" (FP7) [Nmi; Mcna]. McStas was supported directly from the construction project for the ISIS second target station (TS2/EU), see [Ts2]. Currently McStas is supported through the Danish involvement in the *Data Management and Software Center*, a subdivision of the European Spallation Source (ESS), see [Ess] and the European Union through "SINE2020/WP3 e-elarning" and and "SINE2020/WP8 e-Tools" (Horizon2020). the home pages [Sin].

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

- P. Willendrup, E. Farhi, E. Knudsen, U. Filges and K. Lefmann, Journal of Neutron Research, **17** (2014) 35.

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

1. About the component library

This McStas Component Manual consists of the following major parts:

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

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

1.1. Authorship

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

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

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

1.2. Symbols for neutron scattering and simulation

In the description of the theory behind the component functionality we will use the usual symbols $\mathbf{r}$ for the position (x, y, z) of the particle (unit m), and $\mathbf{v}$ for the particle velocity (v_x, v_y, v_z) (unit m/s). Another essential quantity is the neutron wave vector $\mathbf{k} = m_\mathrm{n}\mathbf{v}/\hbar$, where m_n is the neutron mass. $\mathbf{k}$ is usually given in $\mathrm{Å}^{-1}$, while neutron energies are given in meV. The neutron wavelength is the reciprocal wave vector, $\lambda = 2\pi/k$. In general, vectors are denoted by boldface symbols.

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

The spin of the neutron is given a special treatment. Despite the fact that each physical neutron has a well defined spin value, the McStas spin vector **s** can have any length between zero (unpolarized beam) and unity (totally polarized beam). Further, all three cartesian components of the spin vector are present simultaneously, although this is physically not permitted by quantum mechanics. For further details about polarization handling, you may refer to the Appendix A.

1.3. Component coordinate system

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

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

1.4. About data files

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

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

File extensions are not compulsory but help in identifying relevant files per application. We list powder and liquid data files from the McStas 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 8.3) and Isotropic_Sqw components (see 8.7). Whenever using any $S(q,\omega)$ data for the Isotropic_Sqw component, we kindly request to cite the corresponding reference.

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

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

Table 1.1.: Data files of the McStas library.

1.5. Component source code

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

In case users only require to add new features, preserving the existing features of a component, 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 McStas take precedence over a library component of the same name.

1.6. Documentation

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

```
1 mcdoc
```

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

```
1 mcdoc --text <name>
2 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.

MCSTAS/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 McStas library [Ics; DL03]. 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.

MCSTAS/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**	liquid cesium. Bodensteiner et al, Phys. Rev. A, 45 (1992) 5709 and 46 (1992) 3574.
D2_liq_21_tot.sqw	7.64	0	0.00052	21		liquid deuterium. Measured. Frei et al, PHYSICAL REVIEW B 80 (2009) 064301
D2_pow_21_tot.sqw	7.64	0	0.00052	12		powder deuterium. Measured. Frei et al, PHYSICAL REVIEW B 80 (2009) 064301
D2O_liq_290_coh and D_liq_290_inc	15.4	4.1	0.0012	290		liquid heavy water. Classical MD. E. Farhi et al, J. Nucl. Sci. Tech. (2015)
D2O_liq_290_tot	15.4	4.1	0.0012	290		liquid heavy water. Measured. E. Farhi et al, J. Nucl. Sci. Tech. (2015)
Ge_liq_coh.sqw and Ge_liq_inc.sqw	8.42	0.18	2.2	1211.4	5400	liquid germanium. Ab-initio MD. Hugouvieux V, Farhi E, Johnson MR, et al., PRB 75 (2007) 104208
H2O_liq_290_coh and H2O_liq_290_inc	7.75	161	0.665	290	1530	liquid water. Classical MD. E. Farhi et al, J. Nucl. Sci. Tech. (2015)
H2O_liq_290_tot	7.75	161	0.665	290	1530	liquid water. Measured. E. Farhi et al, J. Nucl. Sci. Tech. (2015)
He4_liq_coh.sqw	1.34	0	0.00747	2	**240**	liquid helium. Measured (constant width). R.J. Donnelly et al., J. Low Temp. Phys., 44 (1981) 471
He4_liq_1_coh.sqw	1.34	0	0.00747	1	**240**	liquid helium. Measured. K. Andersen et al, J Phys Cond Mat, 6 (1994) 821
Ne_liq_tot.sqw	2.62	0.008	0.039	24.56	**591**	liquid neon. Measured. Buyers, Sears et al, Phys Rev A 11 (1975) 697
Rb_liq_coh.sqw and Rb_liq_inc.sqw	6.32	0.5	0.38	312.46	**1300**	liquid rubidium. Classical MD. E. Farhi, V. Hugouvieux, M.R. Johnson, W. Kob, Journal of Computational Physics 228 (2009)

An overview of the component library is also given at the McStas home page [Mcs] and in the User Manual [Wil+05].

1.7. Component validation

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

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

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

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

1.8. Disclaimer, bugs

We would like to emphasize that the usage of both the McStas software, as well as its components are the responsability of the users. Indeed, obtaining accurate and reliable results requires a substantial work when writing instrument descriptions. This also means that users should read carefully both the documentation from the manuals [Wil+05] 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 [Git].

2. Monte Carlo Techniques and simulation strategy

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

2.1. Neutron spectrometer simulations

2.1.1. Monte Carlo ray tracing simulations

The behavior of a neutron scattering instrument can in principle be described by a complex integral over all relevant parameters, like initial neutron energy and divergence, scattering vector and position in the sample, etc. However, in most relevant cases, these integrals are not solvable analytically, and we hence turn to Monte Carlo methods. The neutron ray-tracing Monte Carlo method has been used widely for guide studies [Cop93; Far+02; Sch+04], instrument optimization and design [ZLa04; Lie05]. Most of the time, the conclusions and general behavior of such studies may be obtained using the classical analytic approaches, but accurate estimates for the flux, resolution and generally the optimum parameter set, benefit considerably from MC methods.

Mathematically, the Monte-Carlo method is an application of the law of large numbers [Jam80; GRR92]. Let $f(u)$ be a finite continuous integrable function of parameter u for which an integral estimate is desirable. The discrete statistical mean value of f (computed as a series) in the uniformly sampled interval $a < u < b$ converges to the mathematical mean value of f over the same interval.

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

In the case were the u_i values are regularly sampled, we come to the well known midpoint integration rule. In the case were the u_i values are randomly (but 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 consult James [Jam80] for a detailed review on the Monte-Carlo method.

2.2. The neutron weight

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

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

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

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

Each adjustment factor should be $0 < \pi_j < 1$, except in special circumstances, so that total flux can only decrease through the simulation, see section 2.3. 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.
- Transmission through Soller blade collimators and velocity selectors (in the approximation which does not take each blade into account).
- Reflection from monochromator (and analyzer) crystals with finite reflectivity and mosaicity.
- Reflection from guide walls.
- 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 neutrons histories ("rays") with different weights. The sum of these weights is an estimate of the mean number of neutrons hitting the monitor (or detector) per second in a "real" experiment. One may write the counting result as

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

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

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

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

as is used in real neutron experiments (where $\overline{p} \equiv 1$). For a better approximation we return to Eq. (2.3). Allowing variations in both N and $\overline{p}$, we calculate the variance of the resulting intensity, assuming that the two variables are statistically 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 monitor/detector components in McStas will keep track of $N = \sum_i p_i^0, I = \sum_i p_i^1$, and $M_2 = \sum_i p_i^2$.

2.3. Weight factor transformations during a Monte Carlo choice

When a Monte Carlo choice must be performed, *e.g.* when the initial energy and direction of the neutron ray is decided at the source, it is important to adjust the neutron weight

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

so that the combined effect of neutron weight change and Monte Carlo probability of making this particular choice equals the actual physical properties we like to model.

Let us follow up on the simple example of transmission. The probability of transmitting the real neutron is P, but we make the Monte Carlo choice of transmitting the neutron ray each time: $f_{\mathrm{MC}} = 1$. This must be reflected on the choice of weight multiplier $\pi_j = P$. Of course, one could simulate without weight factor transformation, in our notation written as $f_{\mathrm{MC}} = P, \pi_j = 1$. To generalize, weight factor transformations are given by the master equation

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

This probability rule is general, and holds also if, e.g., it is decided to transmit only half of the rays ($f_{\mathrm{MC}} = 0.5$). An important different example is elastic scattering from a powder sample, where the Monte-Carlo choices are the particular powder line to scatter from, the scattering position within the sample and the final neutron direction within the Debye-Scherrer cone. This weight transformation is much more complex than described above, but still boils down to obeying the master transformation rule 2.9.

2.3.1. Direction focusing

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

If *e.g.* a sample scatters isotropically over the whole 4π solid angle, and all interesting directions are known to be contained within a certain solid angle interval $\mathbf{\Delta\Omega}$, only these solid angles are used for the Monte Carlo choice of scattering direction. This implies $f_{\mathrm{MC}}(\Delta\Omega) = 1$. However, if the physical events are distributed uniformly over the unit sphere, we would have $P(\Delta\Omega) = \Delta\Omega/(4\pi)$, according to Eq. (2.9). One thus ensures that the mean simulated intensity is unchanged during a "correct" direction focusing, while a too narrow focusing will result in a lower (*i.e.* wrong) intensity, since we cut neutrons rays that should have reached the final detector.

2.4. Adaptive and Stratified sampling

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

An other class of 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 individually. The advantage is that we are then sure that each sub-space

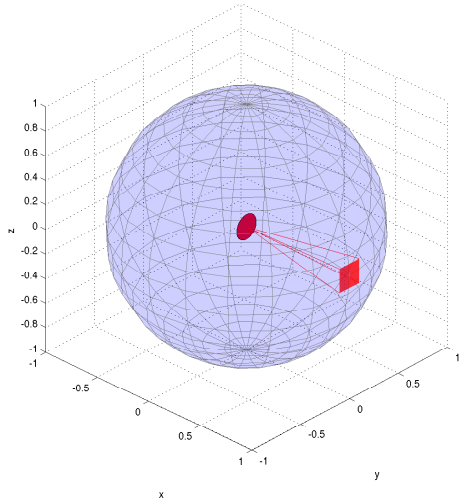


Figure 2.1.: Illustration of the effect of direction focusing in McStas. Weights of neutrons emitted into a certain solid angle are scaled down by the full unit sphere area.

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. In conjunction with adaptive sampling, 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.

In the case of McStas, an ad-hoc implementation of adaptive stratified is used when repeating events, such as in the Virtual sources (Virtual_input, Vitess_input, Virtual_mcnp_input, Virtual_tripoli4_input) and 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 neutron 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 [Jam80] 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.

Records	Accuracy
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 McStas.

McStas uses a space with $d = 10$ parameters to describe neutrons (position, velocity, spin, 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.

3. Source components

McStas 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:

```
1  TRACE
2
3  COMPONENT MySource=Source_gen(...) AT (...)
4  EXTEND
5  %{
6    t = 1e-3*randpm1(); /* set time to +/- 1 ms */
7  %}
```

3.0.1. Neutron flux

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

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

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

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).

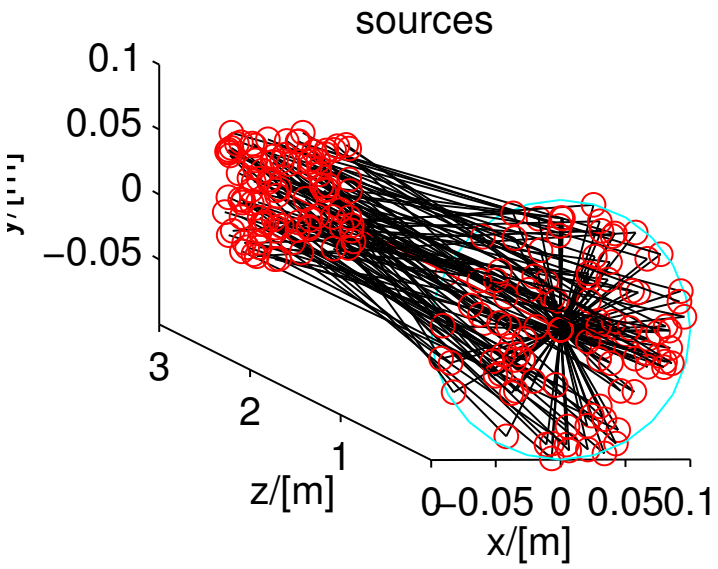


Figure 3.1.: A circular source component (at z=0) emitting neutron events randomly, either from a model, or from a data file.

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_{sim} denotes the number of neutron histories to simulate, the initial neutron weight p_0 must be set to

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

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

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

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

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

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

Input Parameters for component `Source_simple` from sources

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

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

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

3.2. Source_div: A continuous source with specified divergence

Input Parameters for component `Source_div` from sources

```
1 <Parameter = value>, [Unit], Description
```

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.

3.3. Source_Maxwell_3: A continuous source with a Maxwellian spectrum

Input Parameters for component `Source_Maxwell_3` from sources

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

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

3.4. Source_gen: A general continuous source

Input Parameters for component `Source_gen` from sources

```
1 <Parameter = value>, [Unit], Description
```

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

Maxwellian parameters for some continuous sources are given in Table 3.1. As nobody knows exactly the characteristics of the sources (it is not easy to measure spectrum there), these figures should be used with caution.

Source Name	T_1	I_1	T_2	I_2	T_3	I_3	factor
PSI cold source	150.4	3.67e11	38.74	3.64e11	14.84	0.95e11	* I_{target} (mA)
ILL VCS (H1)	216.8	1.24e13	33.9	1.02e13	16.7	3.042e12	58MW
ILL HCS (H5)	413.5	10.22e12	145.8	3.44e13	40.1	2.78e13	58MW
ILL Thermal(H2)	683.7	5.874e12	257.7	2.51e13	16.7	1.034e12	58MW, /2.25
ILL Hot source	1695	1.74e13	708	3.9e12			58MW
PIK cold source	204.6	5.38e12	73.9	7 2.50e12	23.9	9.51e12	
HZB cold source	43.7	1.4e12	137.2	2.08e12			10MW, radius=.155
HZB bi-spectral	43.7	1.4e12	137.2	2.08e12	293	1.77e12	10MW
HZB thermal tube	293.0	2.64e12					10MW
FRM2 cold	35.0	9.38e12	547.5	2.23e12	195.4	1.26e13	20MW
FRM2 thermal	285.6	3.06e13	300.0	1.68e12	429.9	6.77e12	20MW
LLB cold,14MW	220	2.09e12	60	3.83e12	20	1.04e12	14MW
TRIGA thermal	300	3.5e11					1MW

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

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

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

Input Parameters for component `Moderator` from sources

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

where the decay constant is given by

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

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

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

3.6. ISIS_moderator: ISIS pulsed moderators

Input Parameters for component `ISIS_moderator` from sources

```
1 <Parameter = value>, [Unit], Description
```

3.6.1. Introduction

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

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

Table 3.2.: Description of Models

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

3.6.2. Using the McStas Module

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

```
1 BASH: export MCTABLES=/usr/local/lib/mcstas/contrib/ISIS_tables/
2 TCSH: setenv MCTABLES /usr/local/lib/mcstas/contrib/ISIS_tables/
```

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

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

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

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

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

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

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

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

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

3.6.3. Comparing TS1 and TS2

The Flux data provided in both sets of h.*face* files is for 60 μAmp sources. To compare TS1 and TS2, the TS1 data must be multiplied by three (current average strength of

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

Table 3.3.: Brief Description of Variables

TS1 source 180 μAmps). When the 300 μAmp upgrade happens this factor should be revised accordingly.

3.6.4. Bugs

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

3.7. Source_adapt: A neutron source with adaptive importance sampling

Input Parameters for component `Source_adapt` from sources

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

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

3.7.1. Optimization disclaimer

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

3.7.2. The adaption algorithm

The adaptive importance sampling works by subdividing the initial neutron phase space into a number of equal-sized bins. The division is done on the three dimensions of energy,

horizontal position, and horizontal divergence, using N_{eng}, N_{pos}, and N_{div} number of bins in each dimension, respectively. The total number of bins is therefore

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

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

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

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

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

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

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

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

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

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

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

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

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

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

This basic idea can be improved with a small modification. The problem is that until the source has had the time to learn the right sampling weights, neutrons may be emitted with high neutron weights (but low probability). These low probability neutrons may

account for a large fraction of the total intensity in detectors, causing large variances in the result. To avoid this, the component emits early neutrons with a lower weight, and later neutrons with a higher weight to compensate. This way the neutrons that are emitted with the best adaption contribute the most to the result.

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

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

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

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

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

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

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

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

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

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

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

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

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

3.7.3. The implementation

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

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

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

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

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

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

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

3.8. Adapt_check: The adaptive importance sampling monitor

Input Parameters for component `Adapt_check` from sources

```
1 <Parameter = value>, [Unit], Description
```

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

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

```
1 ...
2 COMPONENT mysource = Source_adapt(...)
3 ...
4 COMPONENT mycheck = Adapt_check(source_comp = mysource)
5 ...
```

Only one instance of Adapt_check is allowed in an instrument.

We suggest, as alternative method, to make use of the `SPLIT` keyword, as described in the McStas User Manual.

3.9. Source_Optimizer: A general Optimizer for McStas

Input Parameters for component `Source_Optimizer` from sources

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

3.9.1. The optimization algorithm

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

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

The **Source_Optimizer** works as follow:

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

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

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

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

4. The **Source_Optimizer** transforms the *bad* neutrons into *good* ones from the *Optimized* source. The resulting optimised flux is normalised to the non-optimized one:

$$p_{optimized} = p_{initial} \frac{\text{Reference}}{\text{Optimized}}, \tag{3.19}$$

and thus the overall flux at **Monitor_Optimizer** location is the same as without the optimizer. Usually, the process sends more *good* neutrons from the *Optimized* source than that in the *Reference* one. The energy (and velocity) spectra of neutron beam is also kept, as far as possible. For instance, an optimization of v_z will induce a modification of v_x or v_y to try to keep $|\mathbf{v}|$ constant.

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

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

3.9.2. Using the Source_Optimizer

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

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

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

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

options = "*continuous* optimization, *auto* setting, *keep* = 0.1, *bins* = 0.1, *smooth* spikes, SetXY+SetDivV+SetDivS".

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

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

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

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

3.10. Monitor_Optimizer: Optimization locations for the Source_Optimizer

Input Parameters for component `Monitor_Optimizer` from sources

```
<Parameter = value>, [Unit], Description
```

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

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

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

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

There are many other source definitions in McStas.

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

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

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

- **misc/Virtual_input** can read a McStas event file (in text or binary format), often bringing an order-of-magnitude speed-up. See section 10.2.
- **contrib/Virtual_tripoli4_input** does the same, but from event files (text format) obtained from the *Tripoli4* [Tri] reactor simulation program. Such files are usually huge.
- **contrib/Virtual_mcnp_input** can read MCNP "PTRAC" event files (text format) obtained from the *MCNP* [Mcnb] reactor simulation program. Such files are usually huge.
- **misc/Vitess_input** can read *Vitess* [Vit] neutron event binary files.
- **optics/Filter_gen** reads a 1D distribution from a file, and may either modify or set the flux according to it. See section 4.4.

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

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

4.1. Arm: The generic component

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

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

The first Arm instance in an instrument definition may be changed into a `Progress_bar` component in order to display on the fly the simulation progress, and possibly save intermediate results.

4.2. Slit: A beam defining diaphragm

Input Parameters for component `Slit` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

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

4.3. Beamstop: A neutron absorbing area

Input Parameters for component `Beamstop` from optics

```
<Parameter = value>, [Unit], Description
```

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

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

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

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

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

4.4. Filter_gen: A general filter using a transmission table

Input Parameters for component `Filter` from optics

```
<Parameter = value>, [Unit], Description
```

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

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

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

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

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

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

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

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

4.5. Collimator_linear: The simple Soller blade collimator

Input Parameters for component `Collimator_linear` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

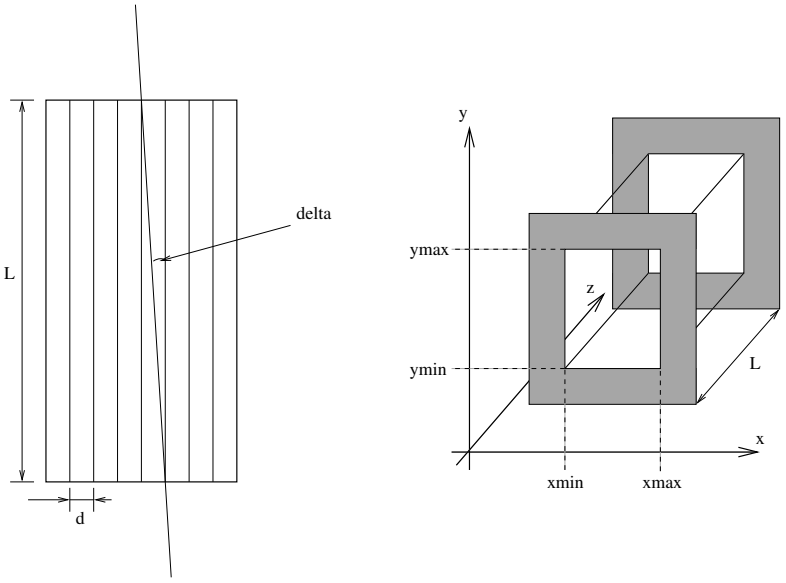


Figure 4.1.: The geometry of a simple Soller blade collimators: The real Soller collimator, seen from the top (left), and a sketch of the component **Soller** (right). The symbols are defined in the text.

4.5.1. Collimator transmission

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

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.

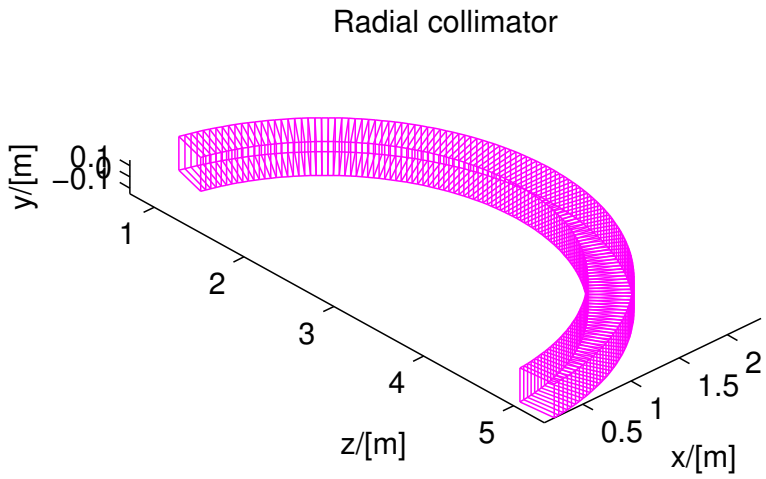


Figure 4.2.: A radial collimator

5. Reflecting optical components: mirrors, and guides

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

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

5.1. Mirror: The single mirror

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

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

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

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

This component may produce wrong results with gravitation.

5.1.1. Mirror reflectivity

To compute the reflectivity of the supermirrors, we use an empirical formula derived from experimental data [Cla+98], see Fig. 5.1. The reflectivity is given by

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

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

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

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

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

Alternatively, the Mirror, Guide and Guide_gravity components may use a reflectivity table $reflect$, which 1st column is q [$\mathring{A}^{-1}$] 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 `MCSTAS/lib/data/`).

5.1.2. Algorithm

The function of the component can be described as

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

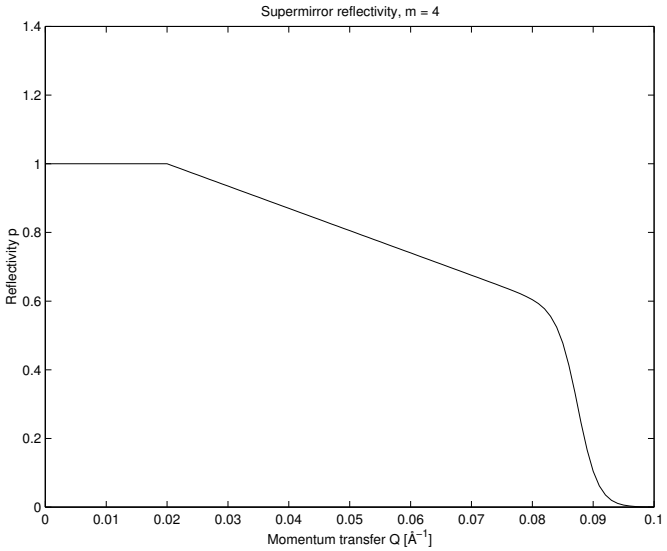


Figure 5.1.: A typical reflectivity curve for a supermirror, Eq. (5.2). The used values are $m = 4$, $R_0 = 1$, $Q_c = 0.02$ Å^{-1}, $\alpha = 6.49$ Å, $W = 1/300$ Å^{-1}.

5.2. Guide: The guide section

Input Parameters for component `Guide` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

Guide may produce wrong results with gravitation support. Use **Guide_gravity** (section 5.4) in this case, or the **Guide_channeled** in section 5.3.

5.2.1. Guide geometry and reflection

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

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

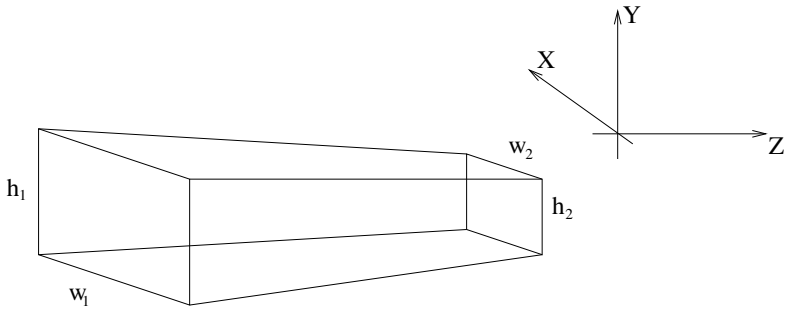


Figure 5.2.: The geometry used for the guide component.

In the following, we refer to an arbitrary guide side by its origin **O** and normal **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:

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

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

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

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

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

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

5.2.2. Algorithm

1. The neutron is initially propagated to the $z = 0$ plane of the guide entrance.
2. If it misses the entrance, it is ABSORB'ed.
3. Otherwise, repeatedly compute the time of intersection with the four mirror sides and the guide exit.
4. The smallest positive t thus found gives the time of the next intersection with the guide (or in the case of the guide exit, the time when the neutron leaves the guide).
5. Propagated the neutron ray to this point.

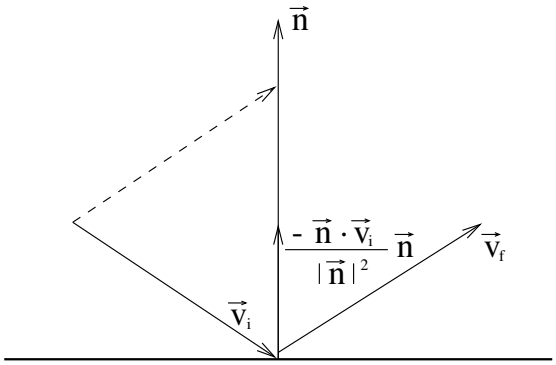


Figure 5.3.: Neutron reflecting from mirror. $\mathbf{v}_\mathrm{i}$ and $\mathbf{v}_\mathrm{f}$ are the initial and final velocities, respectively, and $\mathbf{n}$ is a vector normal to the mirror surface.

6. Compute the reflection from the side.

7. Update the neutron weight factor by the amount $\pi_i = R(Q)$.

8. Repeat this process until the neutron leaves the guide.

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

5.3. Guide_channeled: A guide section component with multiple channels

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

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

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

5.3.1. Algorithm

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

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

5.3.2. Known problems

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

5.4. Guide_gravity: A guide with multiple channels and gravitation handling

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

This component is a variation of **Guide_channeled** (section 5.3) with the ability to handle gravitation effects and functional channeled focusing geometry. Channels can be specified in two dimensions, producing a 2D array (k, k_h) 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 McStas gravitation support (`mcrun --gravitatio` 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.

5.5. Bender: a bender model (non polarizing)

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

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

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

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

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

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

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

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

5.6. Curved guides

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

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

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

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

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

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

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

6. Moving optical components: Choppers and velocity selectors

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

6.1. DiskChopper: The disc chopper

Input Parameters for component `DiskChopper` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

Component **DiskChopper** is an infinately thin, absorbing disc of radius R with n slit openings of height h and angular width θ_0. The slits are symmetrically disposed on the disc. If unset, the slit height h will extend to the centre of the disc ($h = R$).

The **DiskChopper** is self-centering, meaning that the centre of the slit openings will automatically be positioned at the centre of the beam axis (see figure 6.1). To override this behaviour, set the paramter $compat = 1$, positioning the chopper centre at height $-R$ - as implemented in the original **Chopper** component.

Optionally, each slit can have a central, absorbing insert - a *beamstop* of angular width θ_1. For more exotic chopper definitions, use the `GROUP` keyword, see below for an example.

The direction of rotation can be controlled, which allows to simulate e.g. counter-rotating choppers. The phase or time-delay t_0 (in seconds) is defined by the time where the first of the n slits is positioned at the top. As an alternative, an angular phase can be given using the ϕ_0 parameter.

By default, neutrons hitting outside the physical extent of the disc are absorbed. This behaviour can be overruled by setting parameter $abs\_out = 0$.

When simulating the chopping of a continuous beam, most of the neutrons could easily be lost. To improve efficiency, one can set the flag `IsFirst`, which will allow every neutron ray to pass the **DiskChopper**, but modify the time, t, to a (random) time at which it is possible to pass. Of course, there should be only one "first chopper" in

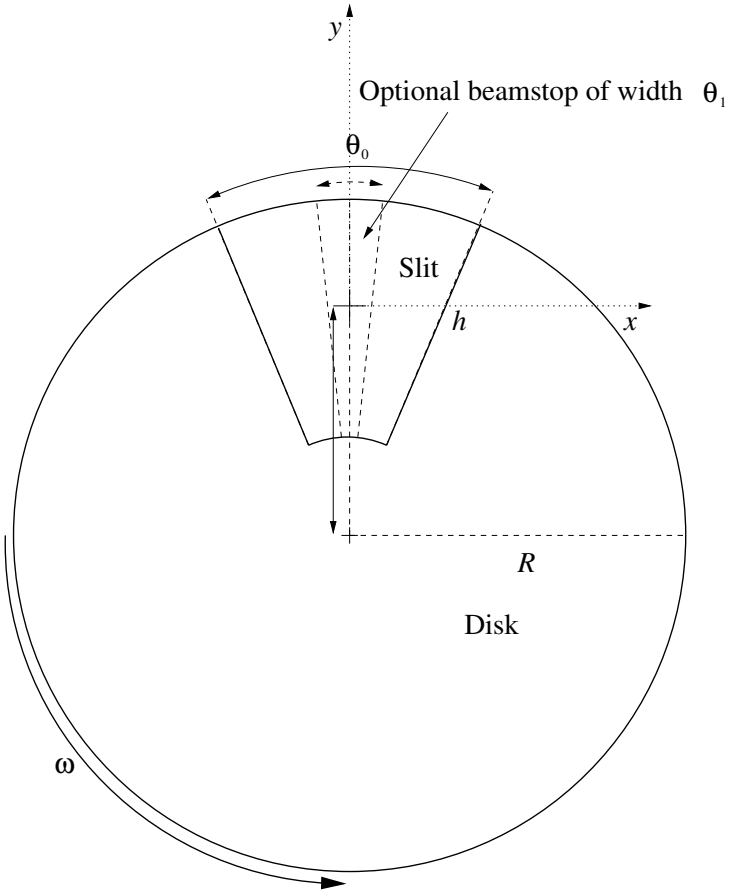


Figure 6.1.: Sketch of a disc chopper with geometry parameters

any simulation. To simulate frame overlap from a "first chopper", one can specify the number of frames to study by the parameter n_{pulse}.

For more advanced chopper geometries than those mentioned above, it is possible to set up a `GROUP` of choppers:

```
1 COMPONENT Chop1 = DiskChopper(omega=2500, R=0.3, h=0.2, theta_0=20, n=1)
2 AT (0, 0, 1.1) RELATIVE Source
3 GROUP Choppers
4
5 COMPONENT Chop2 = DiskChopper(omega=2500, R=0.3, h=0.2, theta_0=20, n=1,
6                               phi_0=40)
7 AT (0, 0, 1.1) RELATIVE Source
8 GROUP Choppers
```

The result of such a **DiskChopper** `GROUP`ing can be seen in figure 6.2

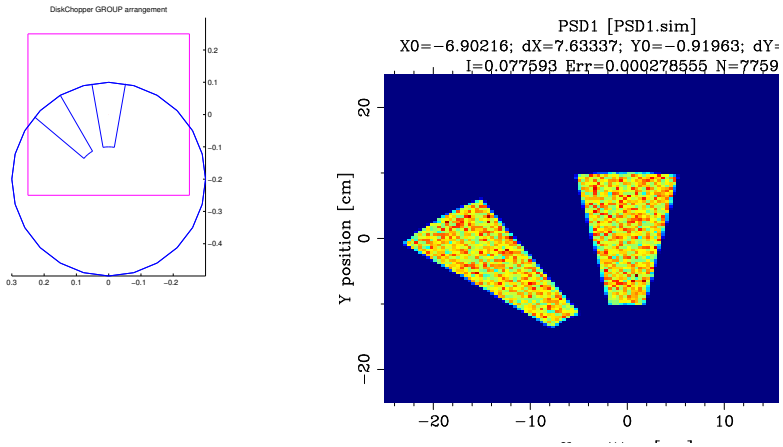


Figure 6.2.: `mcdisplay` rendering and monitor output from a DiskChopper `GROUP`

6.2. FermiChopper: The Fermi-chopper

Input Parameters for component `FermiChopper` from optics

```
1 <Parameter = value>, [Unit], Description
```

6.2.1. The chopper geometry and parameters

The Fermi chopper is a rotating vertical cylinder containing a set of collimating slits (*slit package*). Main geometry parameters are the radius R, minimum and maximum height y_{min} and y_{max} (see Fig. 6.3). In this implementation, the slits are by default straight, but may be coated with super-mirror, and curved. Main parameters for the slits are the number of slits $Nslit$, the length $length$ and width w of each slit, the width of the separating Cd-blades is neglected. The slit walls reflectivity is modelled just like in guide components by the m-value ($m > 1$ for super mirrors), the critical scattering vector Q_c, the slope of reflectivity α, the low-angle reflectivity R_0 and the width of supermirror cut-off W. For $m = 0$ the blades are completly absorbing. The AT position of the component is its center.

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

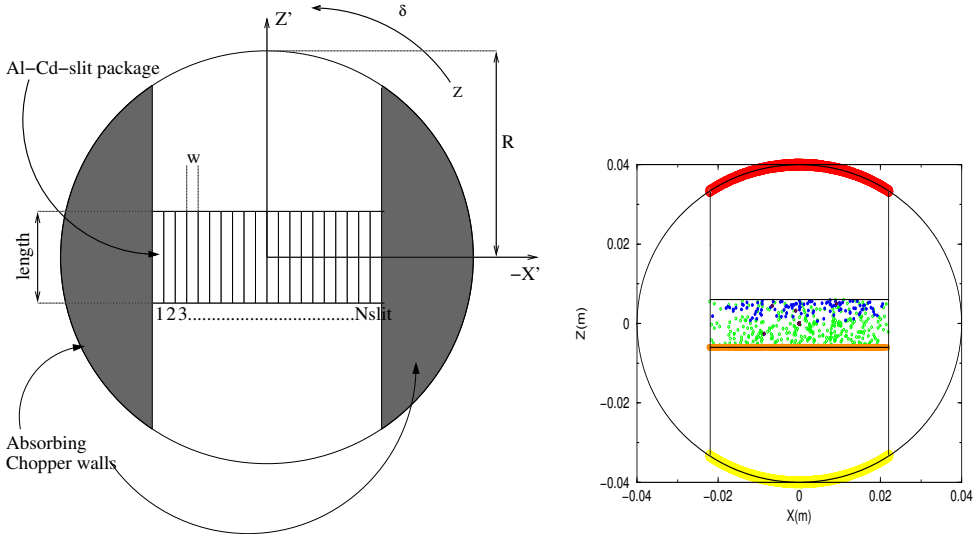


Figure 6.3.: Geometry of the Fermi-chopper (left) and Neutrons in the chopper (right).

setting the `zero_time=1` option. A phase guess value may be set automatically using the `zero_time=2` option.

The curvature of the slit channels is specified with the *curvature* parameter. Positive sign indicates that the deviation 'bump' due to curvature is in the x' positive side, and the center of curvature is in the x' negative side. The optimal radius of curvature R is related to frequency ν and neutron velocity v with: $v = 4\pi R\nu$.

The component was validated extensively by K. Lieutenant. As an alternative, one may use the **Vitess_ChopperFermi** component (eventhough slower and without super-mirror support) or the **FermiChopper_ILL** contributed component. The Guide_gravity component has also a rotating mode, using an approximation of a Fermi Chopper.

6.2.2. Propagation in the Fermi-chopper

As can be seen in figure 6.3, neutrons first propagate onto the cylinder surface of the chopper (yellow curve). Then the program checks the interaction with the entrance of the slit package (orange line) and calculates which slit is hit. If the slit coating is reflecting ($m > 0$), multiple reflections are calculated (green, blue and maroon circles), otherwise the neutrons are absorbed as soon as they interact with the blades. Finally the remaining neutrons propagate to the exit of the chopper (red curve).

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

$$\delta(t) = \widehat{z, z'} = \omega.(t - t_0) = \omega.t + \phi_0$$

where t is the absolute time, t_0 is the chopper delay, and ϕ_0 is the chopper phase. The chopper should better be *time focussing*: slow neutrons should pass before the fast ones,

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

Table 6.1.: FermiChopper component parameters

so that they finally hit the detectors at the same time. Therefore the signs of ω and δ are very important: For $t > t_0$, δ is positive and points anti-clockwise.

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

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

Together with the equation for a non-accelerated, linear propagation $\vec{r} = \vec{r_0} + \vec{v}t$ the orthogonal transformation produces a curve in the Z'-X'-plane known as *archidemic spiral*, as can be seen in figure 6.4. The two vector components $s(t) = (z', x')$ follow the equation:

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

For a fixed chopper rotation speed, the neutron trajectory tends to strech from a spiral curve for slow neutrons to a straight line for fast neutrons. For real Fermi chopper settings ν (about 100 Hz on IN6 at the ILL), neutron trajectories are found to be nearly straight for 1000 m/s neutron velocities [Bla83].

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

For this purpose, the *Ridders's* root finding method was implemented [Pre+02] in

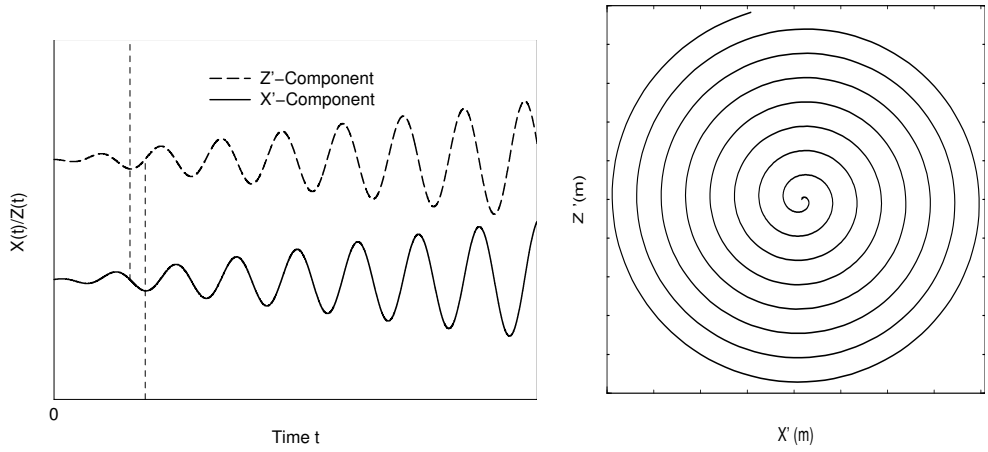


Figure 6.4.: The x' and z' component as a function of time in the rotating frame (left). A typical neutron trajectory in the rotating frame (right).

order to solve

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

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

Standard slit packages (non super-mirror)

The neutrons are first propagated to the outer chopper cylinder and their coordinates are transformed into the rotating frame using T. Neutrons outside the slit channel (chopper opening), or hitting the top and bottom caps are absorbed (yellow dots in Fig. 6.3). The side from which the neutron approaches the chopper is known (positive or negative z'-axis of the rotating frame) so that the calculation of the time of interaction with the slit package entrance t_1 is performed solving $z' = \pm\frac{\text{length}}{2}$ in Eq. (6.2). Using the result of the numerical algorithms the neutron propagates to the entrance of the slit package (orange circles in Fig. 6.3). Neutrons getting aside the slit package entrance are absorbed. Additionally, the slit package exit time t_2 is estimated the same way with $z' = \mp\frac{\text{length}}{2}$, in order to evaluate the whole time-of-flight in the chopper. The index of the slit which was hit is also computed, as we know the x' coordinate in the rotating frame at the slit entrance.

Differentiating Eq. (6.2) for x coordinate

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

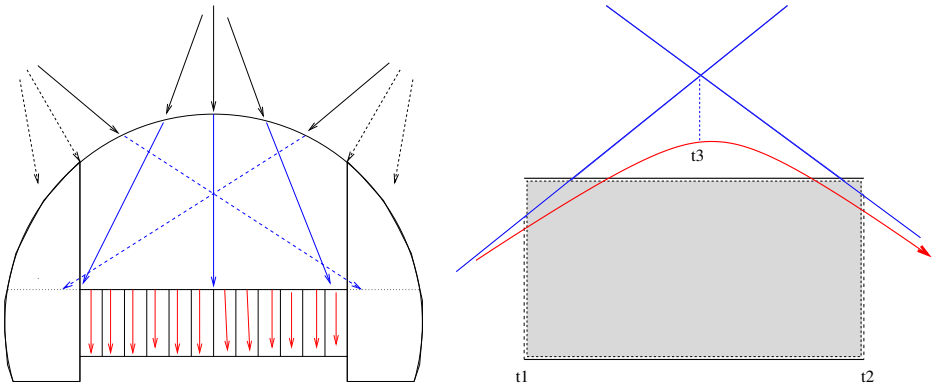


Figure 6.5.: The different steps in the algorithm (left). A neutron trajectory in a slit (right)

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

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

If the neutron changes of slit channel at this point, we may determine the intersection time of the neutron trajectory within $[t_1, t_3]$ or $[t_3, t_2]$, as seen in Fig. 6.5. If walls are not reflecting, we just absorb neutrons here.

The reflections (super-mirror slits)

If slit walls are reflecting, neutron is first propagated to the slit separating surface. Then the velocity in the rotating frame is computed using Eq. (6.2). Perpendicular velocity v'_x is reverted for reflection, and inverse T transformation is performed. Reflected intensity is computed the same way as for the guide component (see section 5.1). The remaining time t_2 to the slit output is estimated and the tangent intersection process is iterated, until neutron exits. Remember that super mirror $m < 1$ parameters behave like $m = 1$ materials (see section 5.1.1). Selecting $m = 0$ sets the blabes absorbing.

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

WARNING: Issues have been reported for the supermirror slit option in this component. The component works correctly when using the standard, absorbing slits. We will be back with more information during the course of 2017. Meanwhile we suggest to instead use Guide_channeled with the rotate/derotate option shown in the test instrument Test_Fermi.instr.

Curved slit packages

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

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

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

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

$$x' \to x' + \Delta_{x'}(z') \tag{6.6}$$

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

6.3. Vitess_ChopperFermi: The Fermi Chopper from Vitess

Input Parameters for component `Vitess_ChopperFermi` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

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

with

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

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

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

Geometry for **straight** and **circular** channels: The geometry of the chopper consists of a rectangular shaped object with a channel system. In transmission position, there are N_{gates} slits of width w_{slit} each along the x-axis, separated by absorbing walls of thickness w_{wall} (see figure 6.6). The total width w_{tot} is given by

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

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

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

It serves to prevent transmission of neutrons which do not fly through the channels; but it also reduces the transmission, because the cylinder removes neutrons in front of the channel entrance or behind the channel exit (see figure 6.6).

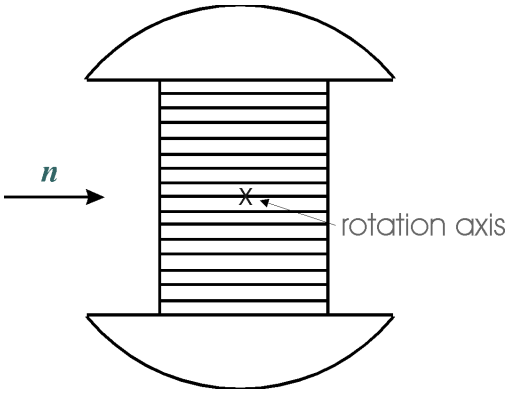


Figure 6.6.: geometry of a staight Fermi chopper

Geometry for **parabolic** channels: In this case, the Fermi chopper is supposed to be a full cylinder, i.e. the central channels are longer than those on the edges. The other features are the same as for the other options. (see figure 6.7).

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

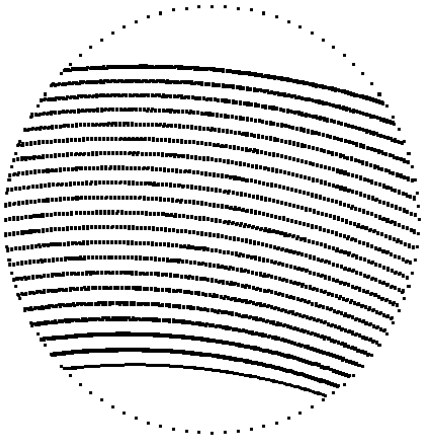


Figure 6.7.: geometry of a curved Fermi chopper

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

This component is about twice slower than the `FermiChopper` component.

The component must be placed after a component which sets a non zero flight path to the Fermi Chopper (e.g. not an Arm).

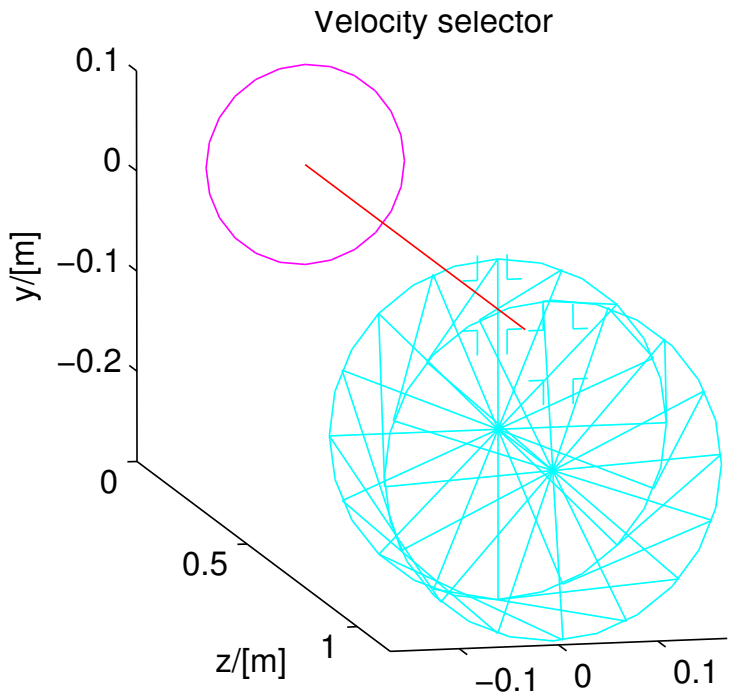


Figure 6.8.: A velocity selector

6.4. V_selector: A rotating velocity selector

Input Parameters for component `V_selector` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

The component Selector produces equivalent results.

6.4.1. Velocity selector transmission

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

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

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

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

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

where N is the number of collimator blades.

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

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

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

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

6.5. Selector: another approach to describe a rotating velocity selector

Input Parameters for component `Selector` from optics

```
1 <Parameter = value>, [Unit], Description
```

The component **Selector** describes the same kind of rotating velocity selector as **V_selector** - compare description there - but it uses different parameters and a different algorithm:

The position of the apertures relative to the z-axis (usually the beam centre) is defined by the four parameters $xmin, xmax, ymin, ymax$. Entry and exit apertures are always identical and situated directly before and behind the rotor. There are num blades of thickness $width$ twisted by the angle α (in degrees) on a length len. The selector rotates

with a speed feq (in rotation per second); its axle is in a distance *radius* below the z-axis.

First the neutron is propagated to the entrance window. The loss of neutrons hitting the thin side of the blades is taken into account by multiplying the neutron weight by a factor

$$p(r) = \theta_i(r)/\theta_o \tag{6.14}$$

$$\theta_o = 360^o/num \tag{6.15}$$

θ_i is the opening between two blades for the distance r between the neutron position (at the entrance) and the selector axle. The difference between θ_o and θ_i is determined by the blade thickness. The neutron is now propagated to the exit window. If it is outside the regarded channel (between the two actual blades), it is lost; otherwise it remains in the exit plane.

WARNING - Differences between **Selector** and **V_selector**:

- **Selector** has a different coordinate system than **V_selector**; in **Selector** the origin lies in the entrance plane of the selector.
- The blades are twisted to the other side, i.e. to the left above the axle in **Selector**.
- Speed of rotation is given in rotation per second, not in rotations per minute as in **V_selector**.

7. Monochromators

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

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

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

Input Parameters for component `Monochromator_flat` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

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

7.1.1. Monochromator physics and algorithm

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

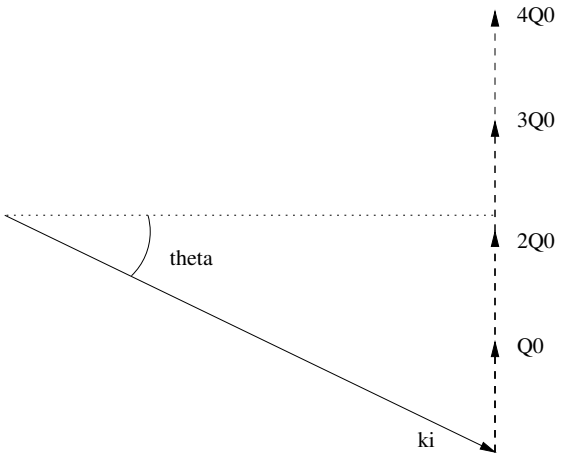


Figure 7.1.: Selection of the Bragg order ("2" in this case).

As a further simplification, the crystal is assumed to be infinitely thin. This means that multiple scattering effects are not simulated. It also means that the total reflectivity, r_0 is used as a parameter for the model rather than the atomic scattering cross section, implying that the scattering efficiency does not vary with neutron wavelength. The variance of the lattice spacing ($\Delta d/d$) is assumed to be zero, so this component is not suitable for simulating backscattering instruments (use the component Single_crystal in section 8.4 for that).

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

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

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

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

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

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

The probability p_{reflect} is used in a Monte Carlo choice to decide whether the neutron is transmitted or reflected.

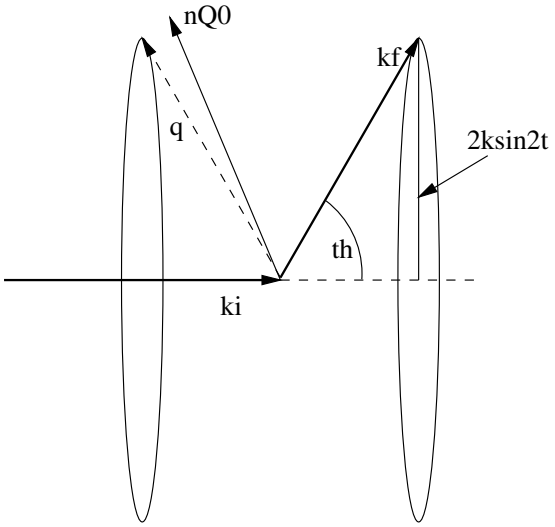


Figure 7.2.: Scattering into the part of the Debye-Scherrer cone covered by the mosaic.

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

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

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

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

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

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

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

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

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

to get good statistics.

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

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

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

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

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

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

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

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

Input Parameters for component `Monochromator_curved` from optics

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

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

As for **Monochromator_flat**, the crystal is assumed to be infinitely thin, and the variation in lattice spacing, $(\Delta d/d)$, is assumed to be zero. Hence, this component is not

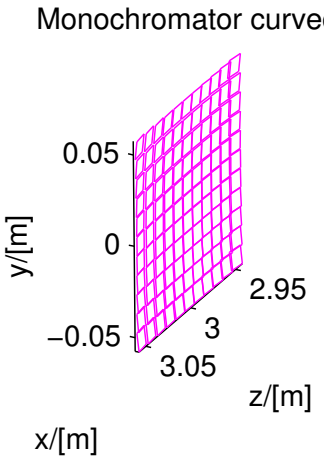


Figure 7.3.: A curved monochromator

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**.

7.3. Single_crystal: Thick single crystal monochromator plate with multiple scattering

The **Single_crystal** component may be used to study more complex monochromators, including incoherent scattering, thickness and multiple scattering. Please refer to section 8.4.

7.4. 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.

8. Samples

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

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

An important component class is elastic Bragg scattering from an ideal powder. The component **PowderN** models a powder scatterer with reflections given in an input file. To scatter on a single Bragg peak, the **Powder1** component may be used. 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 7.1. The monochromators are models of a thin mosaic crystal with a single scattering vector perpendicular to the surface. Much more advanced, the component **Single_crystal** is a general single crystal sample (with multiple scattering) that allows the input of an arbitrary unit cell and a list of structure factors, read from a LAZY / Crystallographica file. This component also allows anisotropic mosaicity and $\Delta d/d$ lattice space variation.

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

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

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

8.0.1. Neutron scattering notation

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

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

Sample	Coherent		Incoherent			
Process	Elastic	Inelastic	Elastic	Inelastic	Absorption	Multi. Scatt.
Phonon_simple		X			1	
Isotropic_Sqw	X	X	X	X	2	X
Powder1	1 line		X		1	
PowderN	N lines		X		1	
Sans_spheres	colloid				1	
Single_crystal	X		X		2	X
V_sample			X	QE broad.	1	
Tunneling_sample		X	X	QE broad.	1	

Table 8.1.: Processes implemented in sample components. Absorption: 1=single only, 2=with secondary

In analygo, the *energy transfer* is given by

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

8.0.2. Weight transformation in samples; focusing

Within many samples, the incident beam is attenuated by scattering and absorption, so that the illumination varies considerably throughout the sample. For single crystals, this phenomenon is known as *secondary extinction* [Bac75], but the effect is important for all samples. In analytical treatments, attenuation is difficult to deal with, and is thus often ignored, making a *thin sample approximation*. In Monte Carlo simulations, the beam attenuation is easily taken care of, as will be shown below. In the description, we ignore multiple scattering, which is however implemented in some sample components.

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

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

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

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

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

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

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

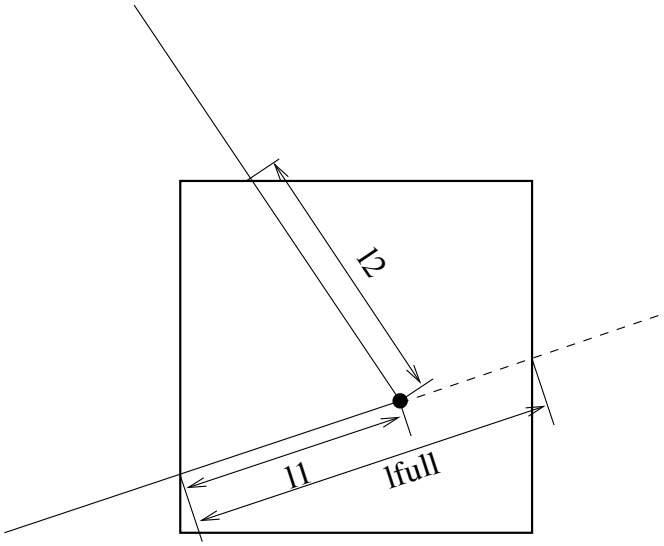


Figure 8.1.: The geometry of a scattering event within a powder sample.

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

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

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

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

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

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

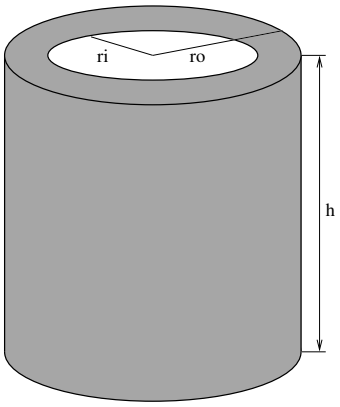


Figure 8.2.: The geometry of the hollow-cylinder vanadium sample.

8.0.3. Future development of sample components

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

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

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

8.1. Incoherent: An incoherent scatterer, the V-sample

Input Parameters for component `Incoherent` from samples

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

The incoherent and absorption cross sections for V are default for the component. For other choices, the parameters σ_{inc}, σ_{abs}, and the unit cell volume V_0 should be specified.

For a loosely packed sample, also the packing factor, f_{pack} can be specified (default value of 1).

8.1.1. Physics and algorithm

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

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

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

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

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

8.1.2. Remark on functionality

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

The `Samples_vanadium` and `Samples_incoherent` test/example instruments exist in the distribution for this component.

8.2. Tunneling_sample: An incoherent inelastic scatterer

Input Parameters for component `Tunneling_sample` from samples

```
1 <Parameter = value>, [Unit], Description
```

The component **Tunneling_sample** displays incoherent inelastic scattering as found in a number of systems, *e.g.* containing mobile hydrogen.

For the sample geometry, we default use a hollow cylinder (which has the solid cylinder as a limiting case). The sample dimensions are: Inner radius r_{i}, outer radius r_{o}, and height h. This geometry is the same as the default for **V_sample**, see figure 8.2.

As for **V_sample**, the sample geometry can be made rectangular by specifying the width, w_x, the height, h_y, and the thickness, t_z.

Also the focusing properties are the same as for **V_sample**. For the focusing is performed as a uniform distribution on a target sphere of radius r_{foc}, at the position $(x_{\mathrm{target}}, y_{\mathrm{target}}, z_{\mathrm{target}})$ in the local coordinate system. The focusing can alternatively be performed on a rectangle with dimensions w_{focus}, h_{focus}, or uniformly in angular space (in a small-angle approximation), using $w_{\mathrm{foc,\ angle}}$, $h_{\mathrm{foc,\ angle}}$. The focusing location can be picked to be a downstream component by specifying `target_index`.

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

The inelastic scattering takes place as a quasielastic (Lorentzian) component, which is chosen with probability f_{QE}. The broadening of the signal is given by Γ (HWHM). In addition, a tunneling signal is present with a probability of f_{tun} and a tunneling energy of $\pm E_{\mathrm{tun}}$. The tunneling peaks are weighted by the usual factor $k_{\mathrm{f}}/k_{\mathrm{i}}$.

The total scattering cross section is given by

$$\begin{aligned}\frac{d^2\sigma}{d\Omega dE_{\mathrm{f}}}(q,\omega) = \frac{\sigma_{\mathrm{inc}}}{4\pi} \times \{&(1 - f_{\mathrm{QE}} - f_{\mathrm{inel}})\delta(\hbar\omega) \\ + \quad & f_{\mathrm{QE}}\frac{\Gamma}{(\hbar\omega)^2 + \Gamma^2} + \frac{f_{\mathrm{inel}}}{2}\frac{k_{\mathrm{f}}}{f_{\mathrm{i}}}\left[\delta(\hbar\omega - E_{\mathrm{tun}}) + \delta(\hbar\omega + E_{\mathrm{tun}})\right]\Big\}\end{aligned} \tag{8.9}$$

The component takes care that $f_{\mathrm{QE}} + f_{\mathrm{tun}} \leq 1$, otherwise an error is returned.

The component accounts for absorption, but not multiple scattering. To obtain intensities similar to real measured ones, we therefore do not take attenuation from scattering into account for the outgoing neutron ray.

8.3. PowderN: A general powder sample

Input Parameters for component `PowderN` from samples

```
<Parameter = value>, [Unit], Description
```

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.

A usage example of this component can be found in the `Neutron site/Tutorial/templateDIFF` instrument from the `mcgui`.

8.3.1. Files formats: powder structures

Data files of type `lau` and `laz` in the McStas distribution data directory are self-documented in their header. A list of common powder definition files is available in Table 1.2 (page 15). 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]>
```

```
3   #sigma_abs     <value of Absorption cross section [barns]>
4   #sigma_inc     <value of Incoherent cross section [barns]>
5   #Debye_Waller <value of Debye-Waller factor DW>
6   #Delta_d/d     <value of Detla_d/d width for all lines>
7   #density       <value of material density [g/cm^3]>
8   #weight        <value of material molar weight [g/mol]>
9   #nb_atoms      <value of number of atoms per unit cell>
```

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

8.3.2. 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 8.7, 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:

```
1 COMPONENT Cryo = PowderN(reflections="Al.laz", radius = 0.01, thickness =
      0.001,
2                          concentric = 1)
3 AT (0,0,0) RELATIVE Somewhere
4
5 COMPONENT Sample = some_other_component(with geometry FULLY enclosed in the
       hollow)
6 AT (0,0,0) RELATIVE Somewhere
7
8 COMPONENT Cryo2 = COPY(Cryo)(concentric = 0)
9 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, $\mathbf{v}_\text{i}$, and a user-specified vector, $\mathbf{f}$.

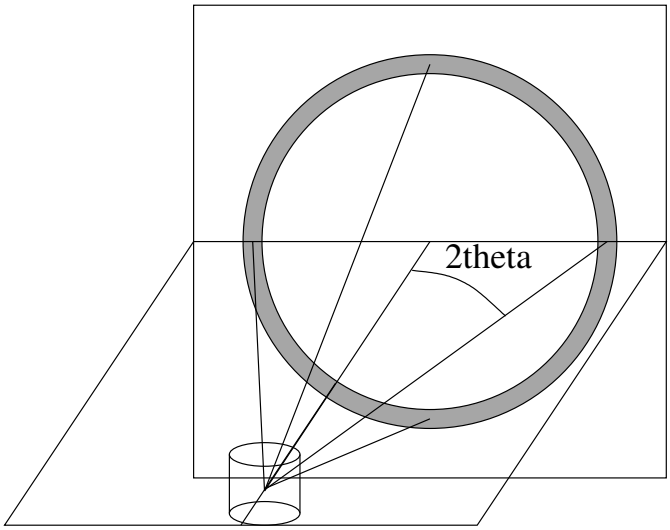


Figure 8.3.: The scattering geometry of a powder sample showing part of the Debye-Scherrer cone (solid lines) and the Debye-Scherrer circle (grey).

8.3.3. Powder scattering

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

$$n\lambda = 2d\sin\theta, \tag{8.10}$$

where n is the order of the scattering (an integer), λ is the neutron wavelength, d is the lattice spacing of the sample, and 2θ is the scattering angle, see figure 8.3. As all crystal orientations are realised in a powder sample, the neutrons are scattered within a *Debye-Scherrer cone* of opening angle 4θ [Bac75].

Equation (8.10) may be cast into the form

$$|\mathbf{Q}| = 2|\mathbf{k}|\sin\theta, \tag{8.11}$$

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

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

$$\left|\sum_j b_j \exp(\mathbf{R}_j \cdot \mathbf{Q})\right|^2, \tag{8.12}$$

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

The textbook expression for the scattering cross section corresponding to one Debye-Scherrer cone reads [Squ78, ch.3.6], with $V = NV_0$ being the total sample volume:

$$\sigma_{\text{cone}} = \frac{V}{V_0^2}\frac{\lambda^3}{4\sin\theta}\sum_Q |F(Q)|^2. \tag{8.13}$$

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

$$\begin{aligned}\sigma_{\text{cone, Q}} &= j_Q f \exp(-2W)\frac{V}{V_0^2}\frac{\lambda^3}{4\sin\theta}|F(Q)|^2 &(8.14)\\ &= f\exp(-2W)\frac{N}{V_0}\frac{4\pi^3}{k^2}\frac{j_Q|F(Q)|^2}{Q} &(8.15)\end{aligned}$$

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

$$\mu_{\text{Q}} = \sigma_{\text{cone,Q}}/V. \tag{8.16}$$

For calibration it may be useful to consider the total intensity scattered into a detector of effective height h, covering only one reflection [Squ78, ch.3.6]. A cut though the Debye-Scherrer cone perpendicular to its axis is a circle. At the distance r from the sample, the radius of this circle is $r\sin(2\theta)$. Thus, the detector (in a small angle approximation) counts a fraction $h/(2\pi r\sin(2\theta))$ of the scattered neutrons, giving a resulting count intensity:

$$I = \Psi\sigma_{\text{cone,Q}}\frac{h}{2\pi r\sin(2\theta)}, \tag{8.17}$$

where Ψ is the flux at the sample position.

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

```
\begin{tabular}{ccl}
$\Psi$ & s$^{-1}$m$^{-2}$ & Incoming intensity of neutrons \\
$I$    & s$^{-1}$ & Detected intensity of neutrons \\
$h$    & m        & Height of detector \\
$r$    & m        & Distance from sample to detector \\
$f$    & 1        & Packing factor of the powder \\
$j$    & 1        & Multiplicity of the reflection \\
$V_0$  & m$^{3}$  & Volume of unit cell\\
$|F(\textbf{Q})|^2$ & m$^2$  & Structure factor \\
$\exp(-2W)$ & 1  & Debye-Waller factor \\
$\mu_\textrm{Q}$ & m$^{-1}$ & Linear attenuation factor due to scattering
    from
```

```
12 one powder line. \\
13 \end{tabular}
```

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

8.3.4. Algorithm

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

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

8.4. Single_crystal: The single crystal component

Input Parameters for component `Single_crystal` from samples

```
1 <Parameter = value>, [Unit], Description
```

The **Single_crystal** component models a thick, flat single crystal with multiple scattering and absorption with elastic coherent scattering. An elastic incoherent background may also be simulated. It may be used to describe samples for diffraction, but also for accurate monochromator descriptions. The component is currently under further review. The current documentation is outdated, especially with respect to the model of crystal mosaicity.

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

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

The user must specify a list of reciprocal lattice vectors $\boldsymbol{\tau}$ to consider along with their structure factors $|F_{\boldsymbol{\tau}}|^2$. The user must also specify the coordinates (in direct space) of the unit cell axes $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, from which the reciprocal lattice will be computed. See section 8.4.5 for file format specifications.

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

8.4.1. The physical model

The textbook expression for the scattering cross-section of a crystal is [Squ78, ch.3]:

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{coh.el.}} = N\frac{(2\pi)^3}{V_0}\sum_{\boldsymbol{\tau}}\delta(\boldsymbol{\tau}-\boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 \tag{8.18}$$

Here $|F_{\boldsymbol{\tau}}|^2$ is the structure factor (defined in section 8.3), N is the number of unit cells, V_0 is the volume of an individual unit cell, and $\boldsymbol{\kappa}(=\mathbf{k}_i-\mathbf{k}_f)$ is the scattering vector. $\delta(\boldsymbol{x})$ is a 3-dimensional delta function in reciprocal space, so for given incoming wave vector $\mathbf{k}_i$ and lattice vector $\boldsymbol{\tau}$, only a single final wave vector $\mathbf{k}_f$ is allowed. In general, this wavevector will not fulfill the conditions for elastic scattering ($k_f = k_i$). In a real

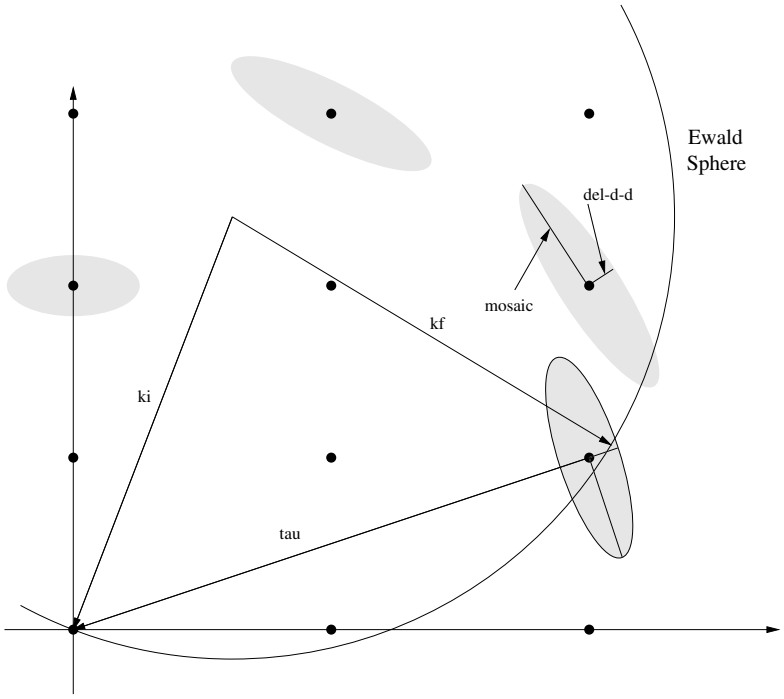


Figure 8.4.: Ewald sphere construction for a single neutron showing the Gaussian broadening of reciprocal lattice points in their local coordinate system.

crystal, however, reflections are not perfectly sharp. Because of imperfection and finite-size effects, there will be a small region around $\boldsymbol{\tau}$ in reciprocal space of possible scattering vectors.

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

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

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

vertical mosaic") perpendicular to a scattering vector.

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

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

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

$$G(x_1, x_2, x_3) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\frac{1}{2}(\frac{x_1^2}{\sigma_1^2}+\frac{x_2^2}{\sigma_2^2}+\frac{x_3^2}{\sigma_3^2})} \tag{8.19}$$

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

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

equation (8.19) can be written as

$$G(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\boldsymbol{x}^{\mathrm{T}} D \boldsymbol{x}} \tag{8.20}$$

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

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

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

where $\hat{u}_1$, $\hat{u}_2$, and $\hat{u}_3$ are the axes of the local coordinate system, written in the global coordinates of the reciprocal lattice. Thus $\hat{u}_1 = \boldsymbol{\tau}/\tau$, and $\hat{u}_2$ and $\hat{u}_3$ are unit vectors

perpendicular to $\hat{u}_1$ and to each other. The matrix U is unitarian, that is $U^{-1} = U^{\mathrm{T}}$. The translation between global and local coordinates is

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

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

$$G(\boldsymbol{y}) = \frac{1}{(\sqrt{2\pi})^3}\frac{1}{\sigma_1\sigma_2\sigma_3}e^{-(U(\boldsymbol{y}+\boldsymbol{\tau}))^{\mathrm{T}}D(U(\boldsymbol{y}+\boldsymbol{\tau}))} \tag{8.21}$$

The elastic coherent cross-section is then given by

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{coh.el.}} = N\frac{(2\pi)^3}{V_0}\sum_{\boldsymbol{\tau}} G(\boldsymbol{\tau} - \boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 \tag{8.22}$$

8.4.2. The algorithm

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

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

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

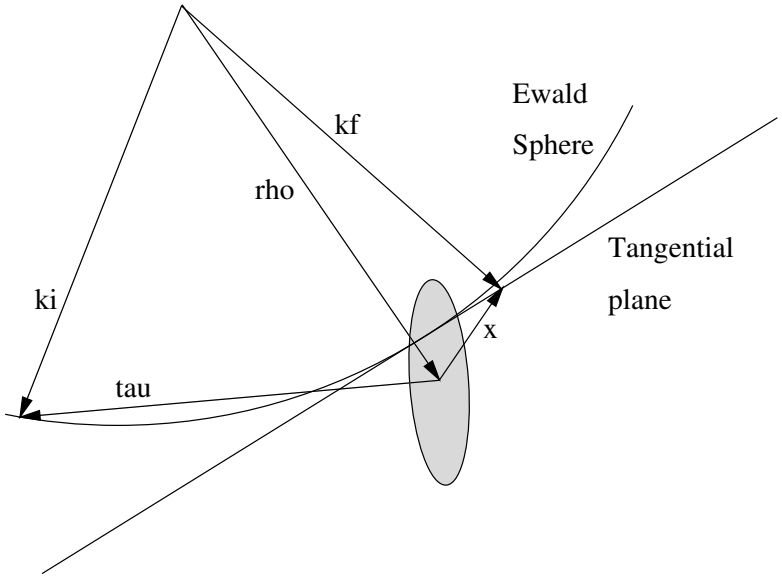


Figure 8.5.: The scattering triangle in the single crystal.

8.4.3. Choosing the outgoing wave vector

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

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

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

The equation for the plane is

$$\boldsymbol{P}(\boldsymbol{t}) = \boldsymbol{o} + B\boldsymbol{t}, \qquad \boldsymbol{t} \in \mathbb{R}^2 \tag{8.23}$$

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

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

$$G(\boldsymbol{x}(\boldsymbol{t})) = \frac{1}{(\sqrt{2\pi})^3}\frac{1}{\sigma_1\sigma_2\sigma_3}e^{-\boldsymbol{x}(\boldsymbol{t})^\mathrm{T} D \boldsymbol{x}(\boldsymbol{t})} \tag{8.24}$$

where $\boldsymbol{x}(\boldsymbol{t}) = \boldsymbol{\tau} - (\boldsymbol{k}_\mathrm{i} - \boldsymbol{k}_\mathrm{f}(\boldsymbol{t}))$ is given in local coordinates for $\boldsymbol{\tau}$. It can be shown that equation (8.24) can be re-written as

$$G(\boldsymbol{x}(\boldsymbol{t})) = \frac{1}{(\sqrt{2\pi})^3}\frac{1}{\sigma_1\sigma_2\sigma_3}e^{-\alpha}e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^\mathrm{T} M (\boldsymbol{t}-\boldsymbol{t}_0)} \tag{8.25}$$

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

To choose $\boldsymbol{k}_{\mathrm{f}}$ we sample $\boldsymbol{t}$ from the 2-dimensional Gaussian distribution (8.25). To do this, we first construct the Cholesky decomposition of the matrix $(\frac{1}{2}M^{-1})$. This gives a 2×2 matrix L such that $LL^{\mathrm{T}} = \frac{1}{2}M^{-1}$ and is possible since M is symmetric and positive definite. It is given by

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

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

$$\begin{aligned} P(\boldsymbol{t})d\boldsymbol{t} &= \frac{1}{2\pi}e^{-\frac{1}{2}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}}d\boldsymbol{g} && (8.26) \\ &= \frac{1}{2\pi}\frac{1}{\det L}e^{-\frac{1}{2}(L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0))^{\mathrm{T}}(L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0))}d\boldsymbol{t} && (8.27) \\ &= \frac{1}{2\pi}\frac{1}{\det L}e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}}M(\boldsymbol{t}-\boldsymbol{t}_0)}d\boldsymbol{t} && (8.28) \end{aligned}$$

where we used that $\boldsymbol{g} = L^{-1}(\boldsymbol{t} - \boldsymbol{t}_0)$ so that $d\boldsymbol{g} = \frac{1}{\det L}d\boldsymbol{t}$. This is just the normalized form of (8.25). Finally we set $\boldsymbol{k}'_{\mathrm{f}} = \boldsymbol{k}_{\mathrm{i}} + \boldsymbol{P}(\boldsymbol{t})$ and $\boldsymbol{k}_{\mathrm{f}} = (k_{\mathrm{i}}/k'_f)\boldsymbol{k}'_{\mathrm{f}}$ to normalize the length of $\boldsymbol{k}_{\mathrm{f}}$ to correct for the (small) error introduced by approximating the Ewald sphere with a plane.

8.4.4. Computing the total coherent cross-section

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

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

For small mosaic we may approximate the sphere with the tangential plane, and we thus get from (8.22) and (8.25):

$$\begin{aligned}
\sigma_{\mathrm{coh},\boldsymbol{\tau}} &= \int N\frac{(2\pi)^3}{V_0}G(\boldsymbol{\tau}-\boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 d\Omega && (8.29)\\
&= \frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^3}{V_0}\frac{1}{(\sqrt{2\pi})^3}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2\int e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}}M(\boldsymbol{t}-\boldsymbol{t}_0)}d\boldsymbol{t} && (8.30)\\
&= \det(L)\frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{3/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2\int e^{-\frac{1}{2}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}}d\boldsymbol{g} && (8.31)\\
&= 2\pi\det(L)\frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{3/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2 && (8.32)\\
&= \frac{\det(L)}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{5/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2 && (8.33)\\
\sigma_{\mathrm{coh}} &= \sum_{\boldsymbol{\tau}}\sigma_{\mathrm{coh},\boldsymbol{\tau}} && (8.34)
\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8.4.5. Implementation details

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

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

The list of reciprocal lattice points is specified in an ASCII data file. Each line contains seven numbers, separated by white space. The first three numbers are the (h, k, l) indices of the reciprocal lattice point, and the last number is the value of the structure factor $|F_{\boldsymbol{\tau}}|^2$, in barns. The middle three numbers are not used and may be omitted; they are nevertheless recommended since this makes the file format compatible with the output from the Crystallographica program [Cry]. Any line beginning with any character of `#;/%` is considered to be a comment, and lines which can not be read as vectors/matrices are ignored.

The column signification may also explicitely be set in the data file header using any of the lines:

```
#column_h <index of the Bragg Qh column>
#column_k <index of the Bragg Qk column>
#column_l <index of the Bragg Ql column>
#column_F2 <index of the squared str. factor '|F|^2' column [b]>
#column_F  <index of the structure factor norm '|F|' column>
```

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

```
#sigma_abs <value of Absorption cross section [barns]>
#sigma_inc <value of Incoherent cross section [barns]>
#Delta_d/d <value of Detla_d/d width for all lines>
#lattice_a <value of the a lattice parameter [Angs]>
#lattice_a <value of the b lattice parameter [Angs]>
#lattice_a <value of the c lattice parameter [Angs]>
#lattice_aa <value of the alpha lattice angle [deg]>
#lattice_bb <value of the beta  lattice angle [deg]>
#lattice_cc <value of the gamma lattice angle [deg]>
```

Example data `*.lau` files are given in directory `MCSTAS/data`.

These files contain an extensive self-documented header defining most the sample parameters, so that only the file name and mosaicity should be given to the component:

```
1  Single_crystal(xwidth=0.01, yheight=0.01, zdepth=0.01,
2    mosaic = 5, reflections="YBaCuO.lau")
```

Powder files from ICSD/LAZY [Ics] and Fullprof [Ful] may also be used (see Table 1.2, page 15). We do not recommend to use these as the equivalent $\vec{q}$ vectors are superposed, not all Bragg spots will be simulated, and the intensity will not be scaled by the multiplicity for each spot.

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

Input Parameters for component `Sans_spheres` from samples

```
1 <Parameter = value>, [Unit], Description
```

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

A usage example of this component can be found in the `Neutron site/tests/SANS` instrument from the `mcgui`.

8.5.1. Small-angle scattering cross section

The neutron intensity scattered into a solid angle $\Delta\Omega$ for a flat isotropic SANS sample in transmission geometry is given by [DL03]:

$$I_s(q) = \Psi \Delta\Omega T A z_{\max} \frac{d\sigma_v}{d\Omega}(q), \tag{8.35}$$

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

In this component, we consider only scattering from a thin solution of monodisperse hard spheres of radius R, where the volume-specific scattering cross section is given by [DL03]

$$\frac{d\sigma_v}{d\Omega}(q) = n(\Delta\rho)^2 V^2 f(q), \tag{8.36}$$

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

Multiple scattering is ignored.

8.5.2. Algorithm

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

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

8.5.3. Calculating the weight factor

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

Hence, the total weight tranformation factor becomes

$$\pi_j = l_{\text{full}}(\pi R_{\text{det}}^2/d^2)/(4\pi)n(\Delta\rho)^2V^2f(q)\exp(-\mu_a l), \tag{8.37}$$

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

This component does NOT simulate absolute intensities. This latter depends on the detector parameters.

Some alternative implementations exist as contributed components.

The `SANS` test/example instrument exists in the distribution for this component.

8.6. Phonon_simple: A simple phonon sample

Input Parameters for component `Phonon_simple` from samples

```
1 <Parameter = value>, [Unit], Description
```

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

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

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

Multiple scattering is not included in this component.

A usage example of this component can be found in the `Neutron site/tests/Test_Phonon` instrument from the `mcgui`.

8.6.1. The phonon cross section

The inelastic phonon cross section for a Bravais crystal of a pure element is given by Ref. [Squ78, ch.3]

$$\begin{aligned}\frac{d^2\sigma'}{d\Omega dE_{\text{f}}} &= b^2\frac{k_{\text{f}}}{k_{\text{i}}}\frac{(2\pi)^3}{V_0}\frac{1}{2M}\exp(-2W)\\ &\times \sum_{\tau,q,p}\frac{(\boldsymbol{\kappa}\cdot\mathbf{e}_{q,p})^2}{\omega_{q,p}}\left\langle n_{q,p}+\frac{1}{2}\mp\frac{1}{2}\right\rangle\delta(\omega\pm\omega_{q,p})\delta(\kappa\pm\mathbf{q}-\tau),\end{aligned}\tag{8.38}$$

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

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

$$d_1(\mathbf{q}) = c_1/a\sqrt{z-s_q},\tag{8.39}$$

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

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

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

$$\begin{aligned}\frac{d^2\sigma'}{d\Omega dE_{\rm f}}(\boldsymbol{\kappa},\omega) &= Nb^2\frac{k_{\rm f}}{k_{\rm i}}\frac{1}{2M}\int\frac{\hbar\kappa^2}{\hbar\omega_q}\left\langle n_q+\frac{1}{2}\mp\frac{1}{2}\right\rangle\delta(\omega\pm\omega_q)\delta(\boldsymbol{\kappa}\pm\mathbf{q})d^3\mathbf{q}\\ &= Nb^2\frac{k_{\rm f}}{k_{\rm i}}\frac{\hbar^2\kappa^2}{2M\hbar\omega_q}\left\langle n_\kappa+\frac{1}{2}\pm\frac{1}{2}\right\rangle\delta(\hbar\omega\pm d_1(\kappa)).\end{aligned} \tag{8.40}$$

8.6.2. The algorithm

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

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

1. If the sample is hit, calculate the total path length inside the sample, otherwise leave the neutron ray unchanged.
2. Choose a scattering point inside the sample
3. Choose a direction for the final wave vector, $\hat{\mathbf{k}}_{\rm f}$ within $\Delta\Omega$.
4. Calculate possible values of $k_{\rm f}$ so that the dispersion relation is fulfilled for the corresponding value of $\mathbf{k}_{\rm f}$. (There is always at least one possible $k_{\rm f}$ value [Bac75].)
5. Choose one of the calculated $k_{\rm f}$ values.
6. Propagate the neutron to the scattering point and adjust the neutron velocity according to $k_{\rm f}$.
7. Calculate and apply the correct weight factor correction, see below.

8.6.3. The weight transformation

Before making the weight transformation, we need to calculate the probability for scattering along one certain direction Ω from one phonon mode. To do this, we must integrate out the delta functions in the cross section (8.40). We here use that $\hbar\omega_q = \hbar^2(k_i^2 - k_f^2)/(2m_{\rm N})$, $\kappa = \mathbf{k}_{\rm i} - k_{\rm f}\hat{\mathbf{k}}_{\rm f}$, and the integration rule $\int\delta(f(x)) = (df/dx)(0)^{-1}$. Now, we reach

$$\left(\frac{d\sigma'}{d\Omega}\right)_j = \int\frac{d^2\sigma'}{d\Omega dE_{\rm f}}dE_{\rm f} = Nb^2\frac{k_{\rm f}}{k_{\rm i}}\frac{\hbar^2\kappa^2}{2Md_1(\kappa_j)J(k_{{\rm f},j})}\left\langle n_\kappa+\frac{1}{2}\pm\frac{1}{2}\right\rangle. \tag{8.41}$$

where the Jacobian reads

$$J = 1 - \frac{m_{\mathrm{N}}}{k_{\mathrm{f}}\hbar^2}\frac{\partial}{\partial k_{\mathrm{f}}}\left(d_1(\kappa)\right). \tag{8.42}$$

A rough order-of-magnitude consideration gives $\frac{k_{\mathrm{f},j}}{k_{\mathrm{i}}} \approx 1$, $J \approx 1$, $\langle n_\kappa + \frac{1}{2} \pm \frac{1}{2}\rangle \approx 1$, $\frac{\hbar^2\kappa^2}{2Md_1(\kappa)} \approx \frac{m}{M}$. Hence, $\left(\frac{d\sigma}{d\Omega}\right)_j \approx Nb^2\frac{m}{M}$, and the phonon cross section becomes a fraction of the total scattering cross section $4\pi Nb^2$, as it must be. The differential cross section per unit volume is found from (8.41) by replacing N with $1/V_0$.

The total weight transformation now becomes

$$\pi_i = a_{\mathrm{lin}} l_{\max} n_{\mathrm{s}} \Delta\Omega b^2 \frac{k_{\mathrm{f},j}}{k_{\mathrm{i}}} \frac{\hbar^2\kappa}{2V_0 M d_1(\kappa) J(k_{\mathrm{f},j})} \left\langle n_\kappa + \frac{1}{2} \pm \frac{1}{2} \right\rangle, \tag{8.43}$$

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

The `Test_Phonon` test/example instrument exists in the distribution for this component.

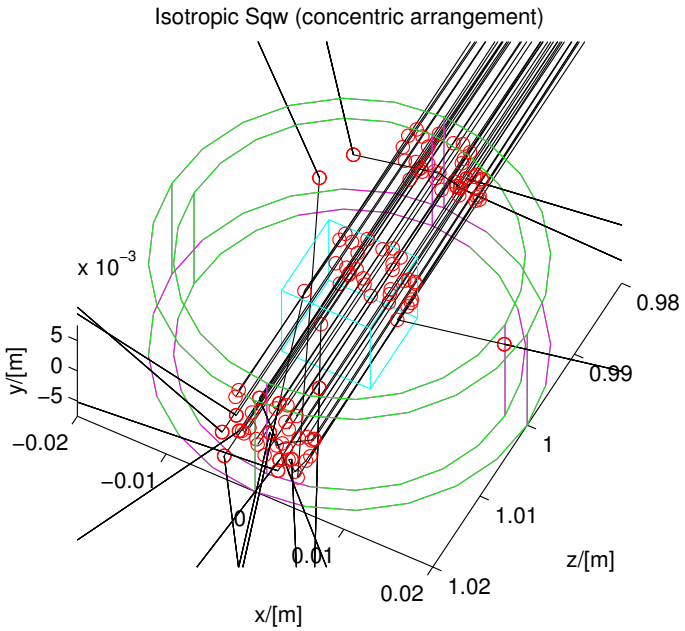


Figure 8.6.: An $l-^4$He sample in a cryostat, simulated with the Isotropic_Sqw component in concentric geometry.

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

Input Parameters for component `Isotropic_Sqw` from samples

```
1 <Parameter = value>, [Unit], Description
```

The sample component *Isotropic_Sqw* has been developed in order to simulate neutron scattering from any isotropic material such as liquids, glasses (amorphous systems), polymers and powders (currently, mono-crystals cannot be handled). The component treats coherent and incoherent neutron scattering and may be used to model most materials, including sample environments with concentric geometries. The structure and dynamics of isotropic samples can be characterised by the dynamic structure factor $S(q,\omega)$, which determines the interaction between neutrons and the sample and therefore can be used as a probability distribution of ω-energy and q-momentum transfers. It handles coherent and incoherent processes, both for elastic and inelastic interactions. The main input for the component is $S(q,\omega)$ tables, or powder structure files.

Usage examples of this component can be found in the

`Neutron site/tests/Test_Isotropic_Sqw`, the
`Neutron site/ILL/ILL_H15_IN6` and the `ILL_TOF_Env` instruments from the `mcgui`.

From McStas 2.4 we have decided to include the earlier version of Isotropic_Sqw from the McStas 2.0 under the name of Isotropic_Sqw_legacy, since some users reported that release being in better agreement with experiments. Note however that issues were corrected since 2.0 and fixed in today's component, that on the other hand exhibits other issues in terms of multiple scattering etc. We will try to rectify the problems during 2017.

8.7.1. Neutron interaction with matter - overview

When a neutron enters a material, according to usual models, it 'sees' atoms as disks with a surface equal to the total cross section of the material σ_{tot}. The latter includes absorption, coherent and incoherent contributions, which all depend on the incoming neutron energy. The transmission probability follows an exponential decay law accounting for the total cross section.

For the neutron which is not transmitted, we select a scattering position along the path, taking into account the secondary extinction and absorption probability. In this process, the neutron is considered to be a particle or an attenuated wave.

Once a scattering position has been assigned, the neutron interacts with a material excitation. Here we turn to the wave description of the neutron, which interacts with the whole sample volume. The distribution of excitations, which determines their relative intensity in the scattered beam, is simply the dynamic structure factor - or scattering law - $S(q, \omega)$. We shall build probability distributions from the scattering law in order to improve the efficiency of the method by favoring the (q, ω) choice towards high $S(q, \omega)$ regions.

The neutron leaves the scattering point when a suitable (q, ω) choice has been found to satisfy the conservation laws. The method is iterated until the neutron leaves the volume of the material, therefore allowing multiple scattering contributions, which will be considered in more details below.

No experimental method makes it possible to accurately measure the multiple scattering contribution, even though it can become significant at low q transfers (below the first diffraction maximum), where the single scattering coherent signal is weak in most materials. This is why attemps have been made to reduce the multiple scattering contribution by partitioning the sample with absorbing layers. However, this is not always applicable thus makiong the simulation approach very valuable.

The method presented here for handling neutron interaction with isotropic materials is similar in many respects to the earlier MSC [FMW72], Discus [Johrc] and MSCAT [Cop74] methods, but the implementation presented here is part of a more general treatment of a sample in an instrument.

8.7.2. Theoretical side

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

In the following, we consider an isotropic medium irradiated with a cold or thermal neutron beam. We ignore the possible thermal fission events and assume that the incoming neutron energy does not correspond to a Breit-Wigner resonance in the material. Furthermore, we do not take into account quantum effects in the material, nor refraction and primary extinction.

Following Squires [Squ78], the experimental counterpart of the scattering law $S(q,\omega)$ is the neutron double differential scattering cross section for both coherent and incoherent processes:

$$\frac{d^2\sigma}{d\Omega dE_f} = \frac{\sigma}{4\pi}\frac{k_f}{k_i}NS(q,\omega) \tag{8.44}$$

which describes the amount of neutrons scattered per unit solid angle $d\Omega$ and per unit final energy dE_f. In this equation, $N = \rho V$ is the number of atoms in the scattering volume V with atomic number density ρ, E_f, E_i, k_f, k_i are the kinetic energy and wavevectors of final and initial states respectively, σ is the bound atom scattering cross-section, Ω is the solid angle and q, ω are the wave-vector and energy transfer at the sample. In practice, the double differential cross section is a linear combinaison of the coherent and incoherent parts as:

$$\sigma S(q,\omega) = \sigma_{coh}S_{coh}(q,\omega) + \sigma_{inc}S_{inc}(q,\omega) \tag{8.45}$$

where the subscripts *coh* and *inc* stand for the coherent and incoherent contributions respectively.

We define its norm on a selected q range:

$$|S| = \iint S(q,\omega)dqd\omega. \tag{8.46}$$

The norm $\lim_{q\to\infty}|S| \simeq q$ for large q values, and can only be defined on a restricted q range.

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

$$\rho g(\vec{r}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq i}\langle\delta(\vec{r}+\vec{r}_i-\vec{r}_j)\rangle \tag{8.47}$$

$$S(\vec{q}) = \int S(\vec{q},\omega)d\omega \tag{8.48}$$

$$= 1+\rho\int_V [g(\vec{r})-1]e^{i\vec{q}.\vec{r}}d\vec{r} \tag{8.49}$$

$$= 1+\rho\int_0^\infty [g(r)-1]\frac{\sin(qr)}{qr}4\pi r^2 dr \text{ in isotropic materials.} \tag{8.50}$$

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

$$g(r) - 1 = \frac{1}{2\pi^2\rho} \int_0^\infty q^2[S(q) - 1] \frac{sin(qr)}{qr} dq \tag{8.51}$$

Both $g(r)$ and $S(q)$ converge to unity for large r and q values respectively, and they are representative of the atoms spatial distribution. In a liquid $\lim_{q\to 0} S(q) = \rho k_B T \chi_T$ where $\chi_T = (\frac{\partial \rho}{\partial P})_{V,T}$ is the compressibility [Ege67; FBS06]. In perfect gases, $S(q) = 1$ for all q. These quantities are obtained experimentally from diffractometers. In principle, $S_{inc}(q) = 1$ in all materials, but a q dependence is rather usual, partly due to the Debye-Waller factor $e^{-q^2\langle u^2\rangle}$. Anyway, $S_{inc}(q)$ converges to unity at high q.

The static pair correlation function $g(r)$ is the probability to find a neighbouring atom at a given distance (unitless). Since $g(0) = 0$, Eq. (8.51) provides a useful normalisation sum-rule for coherent $S(q)$:

$$\int_0^\infty q^2[S(q) - 1]dq = -2\pi^2\rho \text{ for coherent contribution.} \tag{8.52}$$

This means that the integrated oscillations (around 1) of $S_{coh}(q)$ are directly related to the density of the material ρ. In practice, the function $S(q)$ is often known on a restricted range $q \in [0, q_{max}]$, due to either limitations in the sample molecular dynamics simulation, or the measurement itself. In first approximation we consider that Eq. (8.52) can be applied in this range, i.e. we neglect the large q contributions provided $S(q) - 1$ converges faster than $1/q^2$. This is usually true after 2-3 oscillations of $S(q)$ in liquids. Then, in isotropic liquid-like materials, Eq. (8.52) provides a normalisation sum-rule for S.

8.7.3. Theoretical side - scattering in the sample

The Eq. 8.44 controls the scattering in the whole sample volume. Its implementation in a propagative Monte Carlo neutron code such as *McStas* can be summarised as follows:

1. Compute the propagation path length in the material by geometrical intersections between the neutron trajectory and the sample volume.

2. Evaluate the total cross section from the integration of the scattering law over the accessible dynamical range (Section 8.7.3).

3. Use the total cross section to determine the probability of interaction for each neutron along the path length, and select a scattering position.

4. Weight neutron interaction with the absorption probability and select the type of interaction (coherent or incoherent).

5. Select the wave vector and energy transfer from the dynamic structure factor $S(q, \omega)$ used as a probability distribution (Section 8.7.3). Apply the detailed balance.

6. Check whether selection rules can be solved (Section 8.7.3). If they cannot, repeat (5).

This procedure is iterated until the neutron leaves the sample. We shall now detail the key steps of this implementation.

Evaluating the cross sections and interaction probability

Following Sears [Sea75], the total scattering cross section for incoming neutrons with initial energy E_i is

$$\sigma_s(E_i) = \iint \frac{d^2\sigma}{d\Omega dE_f} d\Omega dE_f = \frac{N\sigma}{4\pi} \iint \frac{k_f}{k_i} S(q,\omega) d\Omega dE_f \tag{8.53}$$

where the integration runs over the entire space and all final neutron energies. As the dynamic structure factor is defined in the q, ω space, the integration requires a variable change. Using the momentum conservation law and the solid angle relation $\Omega = 2\pi(1 - cos\theta)$, were θ is the solid angle opening, we draw:

$$\sigma_s(E_i) = N \iint \frac{\sigma S(q,\omega) q}{2k_i^2} dq d\omega. \tag{8.54}$$

This integration runs over the whole accessible q, ω dynamical range for each incoming neutron. In practice, the knowledge of the dynamic structure factor is defined over a limited area with $q \in [q_{min}, q_{max}]$ and $\omega \in [\omega_{min}, \omega_{max}]$ which is constrained by the method for obtaining $S(q,\omega)$, i.e. from previous experiments, molecular dynamics simulations, and analytical models. It is desirable that this area be as large as possible, starting from 0 for both ranges. If we use $\omega_{min} \to 0$, $q_{min} \to 0$, $\omega_{max} > 4E_i$ and $q_{max} > 2k_i$, we completely describe all scattering processes for incoming neutrons with wavevector k_i [FMW72].

This means that in order to correctly estimate the total intensity and multiple scattering, the knowledge of $S(q,\omega)$ must be wider (at least twice in q, as stated previously) than the measurable range in the corresponding experiment. As a side effect, a self consistent iterative method for finding the true scattering law from the measurement itself is not theorically feasible, except for providing crude approximations. However, that measured dynamic structure factor may be used to estimate the multiple scattering for a further measurement using longer wavelength neutrons. In that case, extrapolating the scattering law beyond the accessible measurement ranges might improve substantially the accuracy of the method, but this discussion is beyond the scope of this paper.

Consequently, limiting the q integration in Eq. 8.54 to the maximum momentum transfer for elastic processes $2k_i$, we write the total scattering cross section as

$$\sigma_s(E_i) \simeq \frac{N}{2k_i^2} \int_0^{2k_i} q\sigma S(q) dq. \tag{8.55}$$

Using Eq. 8.45, it is possible to define similar expressions for the coherent and incoherent terms $\sigma_{coh}(E_i)$ and $\sigma_{inc}(E_i)$ respectively. These integrated cross sections are usually

quite different from the tabulated values [DL03] since the latter are bound scattering cross sections.

Except for a few materials with absorption resonances in the cold-thermal energy range, the absorption cross section for an incoming neutron of velocity $v_i = \sqrt{2E_i/m}$, where m is the neutron mass, is computed as $\sigma_{abs}(E_i) = \sigma_{abs}^{2200} \frac{2200m/s}{\sqrt{2E_i/m}}$, where σ_{abs}^{2200} is obtained from the literature [DL03].

We now determine the total cross section accounting for both scattering and absorption

$$\sigma_{tot}(E_i) = \sigma_{abs}(E_i) + \sigma_s(Ei). \tag{8.56}$$

The neutron trajectory intersection with the sample geometry provides the total path length in the sample d_{exit} to the exit. Defining the linear attenuation $\mu(E_i) = \rho\sigma_{tot}(E_i)$, the probability that the neutron event is transmitted along path d_{exit} is $e^{-\mu(E_i)d_{exit}}$.

If the neutron event is transmitted, it leaves the sample. In previous Monte Carlo codes such as DISCUSS [Johrc], MSC [FMW72] and MSCAT [Cop74], each neutron event is forced to scatter to the detector area in order to improve the sample scattering simulation statistics and reduce the computing time. The corresponding instrument model is limited to a neutron event source, a sample and a detector. It is equaly possible in the current implementation to 'force' neutron events to scatter by applying a correction factor $\pi_0 = 1 - e^{-\mu(E_i)d_{exit}}$ to the neutron statistical weight. However, the *McStas* instrument model is often build from a large sequence of components. Eventhough the instrument description starts as well with a neutron event source, more than one sample may be encountered in the course of the neutron propagation and multiple detectors may be positioned anywhere in space, as well as other instrument components (e.g. neutron optics). This implies that neutron events scattered from a sample volume should not focus to a single area. Indeed, transmitted events may reach other scattering materials and it is not desirable to force all neutron events to scatter. The correction factor π_0 is then not applied, and neutron events can be transmitted through the sample volume. The simulation efficiency for the scattering then drops significantly, but enables to model much more complex arrangements such as concentric sample environments, magnets and monochromator mechanical parts, and neutron filters.

If the neutron is not transmitted, the neutron statistical weight is multiplied by a factor

$$\pi_1 = \frac{\sigma_s(E_i)}{\sigma_{tot}(E_i)} \tag{8.57}$$

to account for the fraction of absorbed neutrons along the path, and we may in the following treat the event as a scattering event. Additionally, the type of interaction (coherent or incoherent) is chosen randomly with fractions $\sigma_{coh}(E_i)$ and $\sigma_{inc}(E_i)$.

The position of the neutron scattering event along the neutron trajectory length d_{exit} is determined by [MPC77; Johrc]

$$d_s = -\frac{1}{\mu(E_i)} \ln(1 - \xi[1 - e^{-\mu(E_i)d_{exit}}]) \tag{8.58}$$

where ξ is a random number in [0,1]. This expression takes into account secondary extinction, originating from the decrease of the beam intensity through the sample (self shielding).

Choosing the q and ω transfer from $S(q,\omega)$

The choice of the (q,ω) wavevector-energy transfer pair could be done randomly, as in the first event of the second order scattering evaluation in DISCUS [Johrc], but it is somewhat inefficient except for materials showing a broad quasi-elastic signal. As the scattering originates from structural peaks and excitations in the material $S(q,\omega)$, it is usual [Cop74] to adopt an importance sampling scheme by focusing the (q,ω) choice to areas where the intensity of $S(q,\omega)$ is high. In practice, this means that the neutron event should scatter preferably on e.g. Bragg peaks, quasielastic contribution and phonons.

The main idea to implement the scattering from $S(q,\omega)$ is to cast two consecutive Monte Carlo choices, using probability distribution built from the dynamic structure factor. We define first the probability $P_\omega(\omega)$ as the *unweighted* fraction of modes whose energy lies between ω and $\omega + d\omega$

$$P_\omega(\omega)d\omega = \frac{\int_0^{q_{max}} qS(q,\omega)dq}{|S|}, \tag{8.59}$$

where $|S| = \iint S(q,\omega)dqd\omega$ is the norm of $S(q,\omega)$ in the available dynamical range $q \in [q_{min}, q_{max}]$ and $\omega \in [\omega_{min}, \omega_{max}]$. The probability $P_\omega(\omega)$ is normalised to unity, $\int P_\omega(\omega)d\omega = 1$, and is a probability distribution of mode energies in the material. We then choose randomly an energy transfer ω from this distribution.

Similarly, in order to focus the wavevector transfer choice, we define the probability distribution of wavevector $P_q(q \mid \omega)$ for the selected energy transfer lying between ω and $\omega + d\omega$

$$P_q(q \mid \omega) = \frac{qS(q,\omega)}{S(q)}, \tag{8.60}$$

from which we choose randomly a wavevector transfer q, knowing the energy transfer ω. These two probability distributions extracted from $S(q,\omega)$ are shown in Fig. 8.7, for a model $S(q,\omega)$ function built from the l-^{4}He elementary excitation (Data from Donnelly).

Then a selection between energy gain and loss is performed with the detailed balance ratio $e^{-\hbar\omega/k_BT}$. In the case of Stokes processes, the neutron can not loose more than its own energy to the sample dynamics, so that $\hbar\omega < E_i$. This condition breaks the symmetry between up-scattering and down-scattering.

Solving selection rules and choosing the scattered wave vector

The next step is to check that the conservation laws

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

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

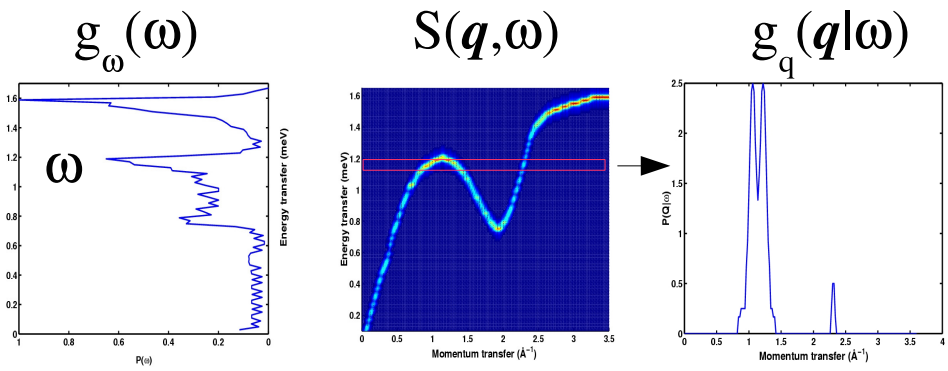


Figure 8.7.: *Centre*: Model of dynamic structure factor $S(q,\omega)$ for l-^{4}He ; *left*: probability distribution g_ω (horizontal axis) of energy transfers (vertical axis, density of states) ; *right* : probability distribution $g_q(\omega)$ (vertical axis) of momentum transfers (horizontal axis) for a given energy transfer $\hbar\omega \sim 1.1$ meV.

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

When the final wave vector has to be computed, the quantities $\vec{k_i}$, $\hbar\omega$ and $q = |\vec{q}|$ are known. We solve the energy conservation law Eq. (8.61) and we select randomly k_f as one of the two roots.

The scattering angle θ from the initial k_i direction is determined from the momentum conservation law $cos(\theta) = (k_i^2 + k_f^2 - q^2)/(2k_ik_f)$, which defines a scattering cone. We then choose randomly a direction on the cone.

If the selection rules can not be verified (namely $|cos(\theta)| > 1$), a new (q,ω) random choice is performed (see Section 8.7.3). It might appear inefficient to select the energy and momentum tranfers first and check the selection rules afterwards. However, in practice, the number of iterations to actually scatter on a high probability process and satisfy these rules is limited, usually below 10. Moreover, as these two steps are simple, the whole process requires a limited number of computer operations.

As mentioned in Section 8.7.3, previous multiple scattering estimation codes [FMW72; Cop74; Johrc] force the outgoing neutron event to come into the detector area and time window, thus improving dramatically the code efficiency. This choice sets the measurable energy and momentum transfers for the last scattering event in the sample, so that the choice of the scattering excitation actually requires a more complex sampling mechanism for the dynamic structure factor. As the present implementation makes no assumption on the simulated instrument part which is behind the sample, we can not apply this method. Consequently, the efficiency of the sample scattering code is certainly lower than previous codes, but on the other hand it does not depend on the type of instrument simulation. In particular, it may be used to model any material in the course of the

neutron propagation along the instrument model (filters, mechanical parts, samples, shields, radiation protections).

Once the scattering probability and position, the energy and momentum transfers and the neutron momentum after scattering have all been defined, the whole process is iterated until the neutron is transmitted and exits the sample volume.

Extension to powder elastic scattering

In principle, the component can work in purely elastic mode if only the $\omega = 0$ column is available in S. Anyway, in the diffractionists world, people do not usually define scattering with $S(q)$ (Eq. 8.48), but through the scattering vector $\boldsymbol{\tau}$, multiplicity $z(\tau)$ (for powders), and $|F^2|$ structure factors including Debye-Waller factors, as in Eq. 8.18.

When doing diffraction, and neglecting inelastic contribution as first approximation, we may integrate Eq. 8.44, keeping $k_i = k_f$.

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{coh.el.}}(|q|) = \int_0^\infty \frac{d^2\sigma_{coh}}{d\Omega dE_f} dE_f = \frac{N\sigma_{coh}}{4\pi} S_{coh}(q) \tag{8.63}$$

$$= N\frac{(2\pi)^3}{V_0}\sum_{\boldsymbol{\tau}} \delta(\boldsymbol{\tau} - \boldsymbol{q})|F_{\boldsymbol{\tau}}|^2 \text{ from Eq. (8.18)} \tag{8.64}$$

with $V_0 = 1/\rho$ being the volume of a lattice unit cell. Then we come to the formal equivalence, in the powder case [Squ78] (integration over Debye-Scherrer cones):

$$S_{coh}(q) = \frac{\pi\rho}{2\sigma_{coh}} \frac{z(q)}{q^2} |F_q|^2 \text{ in a powder.} \tag{8.65}$$

for each lattice Bragg peak wave vector q. The normalisation rule Eq. (8.52) can not usually be applied for powders, as the $S(q)$ is a set of Dirac peaks for which the $\int q^2 S(q) dq$ is difficult to compute, and $S(q)$ does not converge to unity for large q. Each F^2 Dirac contribution may be broaden when specifiying a diffraction peak width.

Of course, the component PowderN (see section 8.3) can handle powder samples efficiently (faster, better accuracy), but does not take into account multiple scattering, nor secondary extinction (which is significant for materials with large absorption cross sections). On the other side, the current Isotropic_Sqw component assumes a powder packing factor of 1 (massive sample). To change into a lower packing factor, use a lower powder density.

Important remarks and limitations

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

1. the two roots k_f^+ and k_f^- are imaginary, which means that conservation laws can not be satisfied and for instance the selected energy transfer is higher than the incoming neutron energy

2. the radius of the target circle is imaginary, that is $|cos(\theta)| > 1$.

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

The $S(q, \omega)$ data sets should be as wide a possible in q and ω range, else scattering conditions will be limited by the reduced data set (specially multiple scattering estimates). On the other hand, when q and ω ranges are too large, some Monte Carlo choices lead to scattering temptatives in non useful regions of S, which reduces dramatically the algorithm efficiency.

The best settings are:

1. to have the widest q and ω range for $S(q, \omega)$ data sets,
2. to either set $wmax$ and $qmax$ to the maximum scatterable energy and wavevectors,
3. or alternatively request the automatic range optimisation by setting parameter `auto_qw=1`. This is recommended, but may sometimes miss a few neutrons if the q, ω beam range has been guessed too small.

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

A bilinear interpolation for the q, ω determination is used to improve the accuracy on the scattered intensity, but it may be unactivated when setting parameter `interpolate=0`. This will often result in a discrete q, ω sampling.

As indicated in the previous section, the Isotropic_Sqw component is not as efficient as PowderN for powder single scattering, but handles scattering processes in a more accurate way (secondary extinction, multiple scattering).

8.7.4. The implementation

Geometry

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

- **box**: dimensions are $x_{width} \times y_{height} \times z_{depth}$.
- **box, hollow**: *idem*, and the side wall thickness is set with *thickness*.
- **cylinder**: dimensions are r for the radius and y_{height} for the height.
- **cylinder, hollow**: *idem*, and hollow part is set with *thickness*.

Parameter	type	meaning
Sqw_coh	string	Coherent scattering data file name. Use 0, NULL or "" to disable
Sqw_inc	string	Incoherent scattering data file name. Use 0, NULL or "" to scatter isotropically (Vanadium like)
sigma_coh	[barns]	Coherent scattering cross-section. -1 to disable
sigma_inc	[barns]	Incoherent scattering cross-section. -1 to disable
sigma_abs	[barns]	Absorption cross-section. -1 to disable
V_rho	[$Å^{-3}$]	atomic number density. May also be specified with molar weight *weight* in [g/mol] and material *density* in [g/cm^3]
T	[K]	Temperature. 0 disables detailed balance
xwidth	[m]	
yheight	[m]	dimensions of a box shaped geometry
zdepth	[m]	
radius_o	[m]	dimensions of a cylinder shaped geometry
radius_i	[m]	sphere geometry if radius_i=0
thickness	[m]	thickness of hollow shape
auto_qw	boolean	Automatically optimise probability tables during simulation
auto_norm	scalar	Normalize $S(q,\omega)$ when -1, use raw data when 0, multiply S by given value when positive
order	integer	Limit multiple scattering up to given order. 0 means all orders
concentric	boolean	Enables to 'enter' inside concentric hollow geometries

Table 8.2.: Main Isotropic_Sqw component parameters

- **sphere**: dimension is r for the radius.
- **sphere, hollow**: *idem*, and hollow part is set with *thickness*.

The AT position corresponds to the centre of the sample.

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

Dynamical structure factor

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

If you are lucky enough to have access to separated coherent and incoherent contributions (e.g. from material simulation), simply set Sqw_coh and Sqw_inc parameter to the files names. If on the other hand you have access to a global data set containing incoherent scattering as well (e.g. the result of a previous experiment), use Sqw_coh parameter, set the σ_{coh} parameter to the sum of both contributions $\sigma_{coh} + \sigma_{inc}$, and set $\sigma_{inc} = -1$. This way we only use one of the two implemented scattering channels. Such global data sets may originate from previous experiments, as far as you have applied all known corrections (multiple scattering, geometry, ...).

In any case, the accuracy of the $S(q,\omega)$ data limits the q and ω resolution of the simulation, eventhough a bilinear interpolation is performed in order to smooth binning. The sampling of data files should then be as thin as possible.

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

Anyway, as explained below, it is also possible to simulate the elastic scattering from a powder file (see below).

File formats: $S(q,\omega)$ inelastic scattering

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

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

Any line beginning with any character of `#;/%` is considered to be a comment, and lines which can not be read as vectors/matrices are ignored.

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

```
#V_0          35    cell volume [Angs^3]
#V_rho        0.07  atom number density [at/Angs^3]
#sigma_abs    5     absorption cross section [barns]
#sigma_inc    4.8   incoherent cross section [barns]
```

```
5  #sigma_coh    1    coherent cross section [barns]
6  #Temperature  10   for detailed balance [K]
7  #density      1    material density [g/cm^3]
8  #weight       18   material molar weight [g/mol]
9  #nb_atoms     6    number of atoms per unit cell
```

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

```
1 Isotropic_Sqw(<geometry parameters>, Sqw_coh="He4_liq_coh.sqw", T=4)
```

Example files are listed as `*.sqw` files in directory `MCSTAS/data`. A table of $S(q,\omega)$ data files for a few liquids are listed in Table 1.3 (page 16).

File formats: $S(q)$ liquids

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

```
1  powder_format=qSq
```

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

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

Such files can only handle elastic scattering.

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

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

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

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

```
1  powder_format=Crystallographica
2  powder_format=Fullprof
3  powder_Dd     =0
4  powder_DW     =1
```

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

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

The powder description is then translated into $S(q)$ by using Eq. (8.65). In this case, the density $\rho = n/V_0$ is the number of atoms in the inverse volume of the unit cell.

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

Such files can only handle elastic scattering. A list of common powder definition files is available in Table 1.2 (page 15).

Concentric geometries, sample environment

The component has been designed in a way which enables to describe complex imbricated set-ups, i.e. what you need to simulate sample environments. To do so, one has first to use hollow shapes, then keep in mind that each surrounding geometry should be first declared before the central position (usually the sample) with the `concentric=1` parameter, but also duplicated (with an other instance name) at a symmetric position with regards to the centre as in the example (shown in Fig. 8.6):

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

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

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

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

8.7.5. Validation

For constant incoherent scattering mode, V_sample, PowderN, Single_crystal and Isotropic_Sqw produce equivalent results, eventhough the two later are more accurate (geometry, multiple scattering). Execution times are equivalent.

Compared with the PowderN component, the $S(q)$ method is twice slower in computation time, and intensity is usually lower by typically 20 % (depending on scattering cross sections), the difference arising from multiple scattering and secondary extinction (not handled in PowderN). The PowderN component is intrinsically more accurate in q as each Bragg peak is handled separately as an exact Dirac peak, with optional Δq spreading. In Isotropic_Sqw, an approximated $S(q)$ table is built from the F^2 data, and is coarser. Still, differences in the diffraction pattern are limited.

The Isotropic_Sqw component has been benchmarked against real experiment for liquid Rubidium (Copley, 1974) and liquid Cesium (Bodensteiner and Dorner, 1989), and the agreement is excellent.

The `Test_Isotropic_Sqw` test/example instrument exists in the distribution for this component.

9. Monitors and detectors

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

Another important difference between computer simulations and real experiments is that one may allow the monitor to be sensitive to any neutron property, as *e.g.* direction, energy, and divergence, in addition to what is found in real-world detectors (space and time). One may, in fact, let the monitor record correlations between these properties, as seen for example in the divergence/position sensitive monitor in section 9.7.

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

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

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

9.1. TOF_monitor: The time-of-flight monitor

Input Parameters for component `TOF_monitor` from monitors

```
<Parameter = value>, [Unit], Description
```

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

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

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

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

Input Parameters for component `TOF2E_monitor` from monitors

```
<Parameter = value>, [Unit], Description
```

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

$$v_{\text{app}} = \frac{L_{\text{flight}}}{t - t_0}, \tag{9.1}$$

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**.

9.3. E_monitor: The energy-sensitive monitor

Input Parameters for component `E_monitor` from monitors

```
<Parameter = value>, [Unit], Description
```

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

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

9.4. L_monitor: The wavelength sensitive monitor

Input Parameters for component `L_monitor` from monitors

```
1 <Parameter = value>, [Unit], Description
```

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

9.5. PSD_monitor: The PSD monitor

Input Parameters for component `PSD_monitor` from monitors

```
1 <Parameter = value>, [Unit], Description
```

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.

9.6. Divergence_monitor: A divergence sensitive monitor

Input Parameters for component `Divergence_monitor` from monitors

```
1 <Parameter = value>, [Unit], Description
```

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.

9.7. DivPos_monitor: A divergence and position sensitive monitor

Input Parameters for component `DivPos_monitor` from monitors

```
1 <Parameter = value>, [Unit], Description
```

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 [Cus03]. **PSD_monitor** can easily be changed into being sensitive to y and vertical divergence by a 90 degree rotation around the z-axis.

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

Input Parameters for component `Monitor_nD` from monitors

```
1 <Parameter = value>, [Unit], Description
```

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`.

9.8.1. 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
$$\text{radius} = \max(\text{abs}\ [x_{\min}, x_{\max}, y_{\min}, y_{\max}, x_{width}/2, y_{height}/2]). \tag{9.2}$$
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
$$\text{radius} = \max(\text{abs}\ [x_{\min}, x_{\max}, x_{width}/2]), \tag{9.3}$$
and the height along y is
$$\text{height} = |y_{\max} - y_{\max}| \text{or} y_{height}. \tag{9.4}$$

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.

9.8.2. 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.

9.8.3. 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):

$$\Phi_c = \int_0^{0.5eV} \frac{d\Phi}{d\lambda} \frac{\lambda}{\lambda_{2200m/s}} d\lambda \tag{9.5}$$

The `auto` option is probably the most useful one: it asks the monitor to automatically determine 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. If no limits are set monitor_nd uses predefined limits that usually make sense for most neutron scattering simulations. Example: the default upper energy limit is 100 meV, but may be changed with an options string like `options="energy limits 0 200"`. Note that the limits also apply in list mode (see below).

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

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

9.8.4. 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 may require a lot of memory and generate huge files. Note that the limits to the measured parameters also apply in this mode. To exemplify, a monitor_nd instance with the option string `"list all v"` will *only* record those neutrons which have a velocity below 100000 m/s, whereas an instance with the option string `"list all vx vy vz 0 2000"` will record all neutrons with $|v_x, v_y| < 100$ m/s and $0 < v_z < 2000$ m/s. Thus, in this latter case, any neutron travelling in the negative z-direction will be disregarded.

9.8.5. Monitor equivalences

In the following table 9.1, we show how the Monitor_nD may substitute any other McStas monitor.

9.8.6. 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.

McStas monitor	Monitor_nD equivalent
Divergence_monitor	*options*="dx bins=$ndiv$ limits=[$-\alpha/2\alpha/2$], lambda bins=$nlam$ limits=[λ_0 λ_1] file=$file$"
DivLambda_monitor	*options*="dx bins=nh limits=[$-h_{max}/2h_{max}/2$], 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=[λ_0 λ_1], time bins=nt limits=[t_0, t_1]" *filename*=$file$
TOFlog_mon	*options*="log time bins=nt limits=[t_0, t_1]"
TOF_monitor	*options*="time bins=nt limits=[t_0, t_1]"

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

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

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

9.8.7. 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.

Directly setting the user variables (simple)

The first method uses directly the `user1` and `username1` component parameters to directly transfer 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.

Indirectly setting 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 transfer 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 B), 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 (...)
(...)
```

9.8.8. 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)
```

```
2 AT (...)
3
4   (... e.g. a guide system )
5
6 COMPONENT CorrelationMonitor  = Monitor_nD(
7    options="x dx, auto, all bins=30, premonitor")
8 AT (...)
```

which performs the same monitoring as the previous example, but with a spatial correlation constrain. Indeed, it records the position *vs* the divergence of neutrons at the correlation location, but only if they reach the monitoring position. All usual Monitor_nD variables may be used, except the user variables. These latter may be defined as described in section 9.8.7 in an EXTEND block.

10. Special-purpose components

The chapter deals with components that are not easily included in any of the other chapters because of their special nature, but which are still part of the McStas system.

One part of these components deals with splitting simulations into two (or more) stages. For example, a guide system is often not changed much, and a long simulation of neutron rays "surviving" through the guide system could be reused for several simulations of the instrument back-end, speeding up the simulations by (typically) one or two orders of magnitude. The components for doing this trick is **Virtual_input** and **Virtual_output**, which stores and reads neutron rays, respectively.

Other components perform the simulation of the instrument resolution functions. These are **Res_sample** and **TOF_Res_sample**, which are to be placed at the sample position, and **Res_monitor**, that should be localized at the position of the instrument detector.

Progress_bar is a simulation utility that displays the simulation status, but assumes the form of a component.

10.1. Virtual_output: Saving the first part of a split simulation

Input Parameters for component `Virtual_output` from sources

```
1 <Parameter = value>, [Unit], Description
```

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.

10.2. Virtual_input: Starting the second part of a split simulation

Input Parameters for component `Virtual_input` from sources

```
1 <Parameter = value>, [Unit], Description
```

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* McStas simulation events. We recommend to control the simulation on `repeat-count` by using a very larger ncount value.

10.3. Res_sample: A sample-like component for resolution calculation

Input Parameters for component `Res_sample` from misc

```
<Parameter = value>, [Unit], Description
```

The component **Res_sample** scatters neutron rays isotropically in direction and uniformly in energy. Regardless of the state of the incoming neutron ray, all directions and energies for the scattered ray have the same probability, within specified intervals.

The component is meant for computation of the resolution function, but may also be used for test and debugging purposes. For actual calculations of the resolution function, **Res_sample** should be used together with **Res_monitor**, described in section 10.5.

The shape of Res_sample is either a hollow cylinder or a rectangular box. The hollow cylinder shape is specified with the 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 10.5.

10.4. TOF_Res_sample: A sample-like component for TOF resolution calculation

Input Parameters for component `TOFRes_sample` from misc

```
<Parameter = value>, [Unit], Description
```

The component **TOF_Res_sample** scatters neutron rays isotropically in position within a specified angular range. As for **Res_sample**, this component is meant for computation of the resolution function, but in this case for one time bin in a time-of-flight (TOF) instrument. The component selects uniformly the neutron energy so that neutron arrival time at the TOF detector lies within one time bin, specified by t_0 and Δt. For actual calculations of the resolution function, **TOF_Res_sample** should be used together with **Res_monitor**, described in section 10.5.

The shape of **TOF_Res_sample** is either a hollow cylinder or a rectangular box. The hollow cylinder shape is specified with the inner and outer radius, r_{i} and r_{o}, respectively, and the height, h. If these parameters are unspecified, the shape is instead a box of dimensions x_w, y_h, and z_t.

The component only propagates neutron rays that are scattered; other rays are absorbed. As for **Res_sample**, the scattering probability is proportional to the neutron flight path length inside the sample. The point of scattering in the sample is chosen uniformly along the neutron flight path inside the sample, and the scattered direction is chosen in a user-specified range, either within a sphere of radius r_{foc}, within a rectangular target with measures $(x_{\mathrm{focus}}, y_{\mathrm{focus}})$ or in the specified angular range. This target is positioned at the x_{target}, y_{target}, z_{target} point in space, or using target_index.

This component stores complete information about the scattering event in the output parameter *res_struct*, see **Res_Sample**.

10.5. Res_monitor: The monitor for resolution calculation

Input Parameters for component `Res_monitor` from misc

```
<Parameter = value>, [Unit], Description
```

The component **Res_monitor** is used for calculating the resolution function of a particular instrument with detector of the given shape, size, and position. The shape of **Res_monitor** is by default rectangular, but can be a box, a sphere, a disk, or a cylinder, depending on the parameter "options". The component works like a normal monitor, but also records all scattering events and stores them to a file that can later be read by the McStas frontend tool `mcresplot`.

For time-of-flight (TOF) instruments, Res_monitor should be understood as giving the resolution of one time bin of the TOF-detector only; the bin properties being specified in the preceding **TOF_Res_sample**.

As described in section 10.3, the **Res_monitor** should be used in connection with one of the components **Res_sample** or **TOF_Res_sample**, the name of which should be passed as an input parameter to **Res_monitor**. For example

```
1    COMPONENT mysample = Res_sample( ... )
2    ...
3    COMPONENT det = Res_monitor(res_sample_comp = mysample, ...)
4    ...
```

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 $\mathrm{\AA}^{-1}$, the energy transfer in meV.

The output parameters from **Res_monitor** are the three count numbers, *Nsum*, *psum*, and *p2sum*, and the handle *file* of the output file.

10.6. Progress_bar: Simulation progress and automatic saving

Name:	Progress_bar
Author:	System
Input parameters	percent, flag_save, 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.

10.7. Beam_spy: A beam analyzer

Name:	Beam_spy
Author:	System
Input parameters	
Optional parameters	
Notes	should overlap previous component

This component is at the same time an Arm and a simple Monitor. It analyzes all neutrons reaching it, and computes statistics for the beam, as well as the intensity.

This component does not affect the neutron beam, and does not contain any propagation call. Thus it gets neutrons from the previous component in the instrument description, and should better be placed at the same position, with `AT (0,0,0) RELATIVE PREVIOUS`.

A. Polarization in McStas

P. Christiansen (Risø)
December 11, 2020

A.1. Introduction

In the current release of McStas there are components with polarization capabilities. At the moment all such components should be understood as under development as the amount of testing and debugging of these components is small, and there are known problems.

Here, we shall report on what have been done so far.

We first describe the polarization vector and how it is related to the neutron wavefunction (section A.2) and then the physics of simple components that we need in McStas is reviewed (section A.3). In the last two sections the actual McStas polarization components are first described (section A.4) and a list of test instruments in McStas is given (section A.5).

We rely heavily on the books [Lov84; Wil88] for the physics where the detailed calculations can be found.

The notation used here (and in [Wil88]) is P (scalar), $\mathbf{P}$ (vector), $\tilde{\mathbf{P}}$ (unit-vector), $\hat{\sigma}$ (operator), and $\hat{\boldsymbol{\sigma}}$ (vector of operators).

A.2. The Polarization Vector

The spin of the neutron is represented by an operator $\hat{\mathbf{s}}$ for which only a single component can be measured at one time. Each single measurement will give a value $\pm 1/2$, but if we could make a large number of measurements on the same neutron state, in each of the three axis directions, and then make the average we get $\langle \hat{\mathbf{s}} \rangle$. The polarization vector, $\mathbf{P}$, is then defined as:

$$\mathbf{P} = \frac{\langle \hat{\mathbf{s}} \rangle}{s}, \tag{A.1}$$

so that $-1 \leq |\mathbf{P}| \leq +1$

For a neutron beam which contains N neutrons, each with a polarization $\mathbf{P}_i$, the beam polarization is defined as:

$$\mathbf{P} = \frac{\sum_i \mathbf{P}_i}{N}. \tag{A.2}$$

If we have one common quantization direction (e.g. a magnetic field direction) each neutron will either be spin up, $\uparrow$, or spin down, $\downarrow$, and the polarization can be expressed as:

$$P = \frac{n_\uparrow - n_\downarrow}{n_\uparrow + n_\downarrow}, \tag{A.3}$$

where ν ($n_\downarrow$) is the number of neutrons with spin up (down).

For a given neutron the probability of the neutron being spin up, $P(\uparrow)$, is:

$$P(\uparrow) = \frac{n_\uparrow}{n_\uparrow + n_\downarrow} = \frac{n_\uparrow + (n_\downarrow - n_\downarrow)/2}{n_\uparrow + n_\downarrow} = \frac{1+P}{2}, \tag{A.4}$$

and $P(\downarrow) = 1 - P(\uparrow) = (1-P)/2$.

The expectation value of the 'spin' operator, $\hat{\boldsymbol{\sigma}}$, which can be expressed by the Pauli matrices, is the polarization vector $\mathbf{P}$, $\mathbf{P} = \langle\hat{\boldsymbol{\sigma}}\rangle \equiv \langle\chi|\hat{\boldsymbol{\sigma}}|\chi\rangle$. The most general form of the spin wave-function χ for a neutron (spin 1/2) is:

$$\chi = a\chi_\uparrow + b\chi_\downarrow, \tag{A.5}$$

where $\chi_\uparrow$ and $\chi_\downarrow$ are eigenfunction of $\hat{\sigma}^z$, and the complex coefficients a and b satisfy $|a|^2 + |b|^2 = 1$.

By calculation we find:

$$P_x = \langle\chi|\hat{\sigma}_x|\chi\rangle = 2\mathrm{Re}(a^*b) \tag{A.6}$$

$$P_y = \langle\chi|\hat{\sigma}_y|\chi\rangle = 2\mathrm{Im}(a^*b) \tag{A.7}$$

$$P_z = \langle\chi|\hat{\sigma}_z|\chi\rangle = |a|^2 - |b|^2 \tag{A.8}$$

This shows the relation of the polarization vector to the neutron wave function.

The neutron magnetic moment operator can be expressed in terms of $\hat{\boldsymbol{\sigma}}$, as:

$$\hat{\boldsymbol{\mu}}_\mathbf{n} = \mu_n\hat{\boldsymbol{\sigma}}, \tag{A.9}$$

which, as shown above, is related to the polarization vector.

In our simulation we represent the polarization by the vector $\mathbf{S} = (s_x, s_y, s_z)$ which is propagated through the different components so it has the correct relative orientation in each component. The probability for the spin to be parallel a given direction $\mathbf{n}$ is then:

$$P(\uparrow|\mathbf{n}) = \frac{1 + \mathbf{n}\cdot\mathbf{S}}{2}. \tag{A.10}$$

This equation (from [SD01]) is easy to understand. The average spin along $\mathbf{n}$ is $\mathbf{n}\cdot\mathbf{S}$ and the probability then follows from Eq. A.4.

For an unpolarized beam, $\mathbf{S} = \mathbf{0}$ and all directions are equally probable (50 %).

Note that in our approach we do not decide if the neutron is up or down after a given component, but instead keep track of as much information for as long as possible.

In the following we will use $\mathbf{P}$ to denote the polarization vector. The most important variables used are:

$\boldsymbol{\kappa}$	Scattering vector.
$\mathbf{P}$	Polarization before a component (ingoing).
$\mathbf{P}_\perp$	Polarization perpendicular to scattering vector, $\mathbf{P}_\perp = \tilde{\boldsymbol{\kappa}} \times (\mathbf{P} \times \tilde{\boldsymbol{\kappa}})$.
$\mathbf{P}'$	Polarization after a component (outgoing).
$\tilde{\boldsymbol{\eta}}$	Unit vector in direction of atomic spin ($\tilde{\boldsymbol{\eta}} \cdot \tilde{\mathbf{B}} = -1$ for a ferromagnet).
$F_N(\boldsymbol{\kappa})$	Unit cell nuclear structure factor.
$F_M(\boldsymbol{\kappa})$	Unit cell magnetic structure factor.

The unit cell nuclear structure factor is defined as:

$$F_N(\boldsymbol{\kappa}) = \sum_{\mathbf{d}} \exp(i\boldsymbol{\kappa} \cdot \mathbf{d}) \bar{b}_d, \tag{A.11}$$

where the $\mathbf{d}$ is the position of the d'th atom within the unit cell, and $\bar{b}_d$ is the average of b_d. In the simple case of a single atom Bravais crystal one finds $F_N(\boldsymbol{\kappa}) = \bar{b}$.

The unit cell magnetic structure factor is useful when the atoms in the crystal only have spin orbital angular momentum, and simple when the magnet is saturated (all spins are parallel or anti-parallel to *one* direction, $\sigma_d = \pm 1$). It is then given as:

$$F_M(\boldsymbol{\kappa}) = \gamma_n r_0 \sum_{\mathbf{d}} \exp(i\boldsymbol{\kappa} \cdot \mathbf{d}) \frac{1}{2} g_d F_d(\boldsymbol{\kappa}) \langle \hat{S}_d \rangle \sigma_d, \tag{A.12}$$

where $r_0 = \frac{\mu_0}{4\pi} \frac{e^2}{m_e} = 2.818 \times 10^{-15}$m, $g = 2$ is the Landé splitting factor, and $F_d(\boldsymbol{\kappa})$ is the magnetic form factor, which is the Fourier transform of the magnetization density (normalized so that $F_d(0) = 1$), and $\langle \hat{S}_d \rangle$ is the thermal average of the ordered atomic spin.

In the following the Debye-Weller factor ($\exp(-W_d)$) have been ignored in all cross sections.

A.2.1. Example: Magnetic fields

The magnetic moment operator of the neutron is $\hat{\boldsymbol{\mu}}_\mathbf{n} = \gamma_n \hat{\mathbf{s}}$, where $\gamma_n = 2\mu_n = -3.826$ is the gyromagnetic ratio (spin and magnetic moment is anti-parallel as for an electron) [1]

A magnetic field, $\mathbf{B}$, will exert a torque, $\boldsymbol{\sigma} = d\mathbf{s}/dt = (1/\gamma_n) d\boldsymbol{\mu}/dt$, on the neutron magnetic moment:

$$\frac{1}{\gamma_n} \frac{d\boldsymbol{\mu}}{dt} = \boldsymbol{\mu} \times \mathbf{B} \tag{A.13}$$

The magnetic moment μ can be related to the polarization as $\boldsymbol{\mu} = \gamma_n \mathbf{P}/2$, and inserting in Eq. A.13 we find:

$$\frac{d\mathbf{P}}{dt} = \gamma_n \mathbf{P} \times \mathbf{B} \tag{A.14}$$

[1] Note that if we had used S (with values $S = \pm 1$) to define γ_n we would get $\gamma_n = -1.913$ which is also commonly used.

In the simple case where $\mathbf{B} = (0, 0, B)$, we find the solution ([Wil88] p. 18) :

$$\begin{aligned} P_X(t) &= \cos(\omega_L t)P_X(0) - \sin(\omega_L t)P_Y(0) \\ P_Y(t) &= \sin(\omega_L t)P_X(0) + \cos(\omega_L t)P_Y(0) \\ P_Z(t) &= P_Z(0), \end{aligned} \tag{A.15}$$

where $\omega_L = -\gamma_n B/\hbar$ is the Larmor frequency.

> The equations above were checked against the equations in the "polarimetrie neutronique" notes by Francis Tasset and found to be consistent. There can be sign differences between different publications depending on whether they use a right-handed (like e.g. McStas) or a left-handed (like e.g. NISP) coordinate system.

A.3. Polarized Neutron Scattering

First we will give a short introduction to how calculations are done and then quote some results which are important for implementing the first McStas components.

All the potentials (nuclear, magnetic, and electric) we will be interested in can be written on the form:

$$\hat{v} = \hat{\beta} + \hat{\boldsymbol{\alpha}} \cdot \hat{\boldsymbol{\sigma}} \tag{A.16}$$

The first term does not affect the spin, while the second term can change the spin. Let us just remind here that:

$$\begin{array}{lll} \hat{\sigma}_x\chi_\uparrow = \chi_\downarrow, & \hat{\sigma}_y\chi_\uparrow = i\chi_\downarrow, & \hat{\sigma}_z\chi_\uparrow = \chi_\uparrow, \\ \hat{\sigma}_x\chi_\downarrow = \chi_\uparrow, & \hat{\sigma}_y\chi_\downarrow = -i\chi_\uparrow, & \hat{\sigma}_z\chi_\downarrow = -\chi_\downarrow. \end{array} \tag{A.17}$$

So that the interaction proportional to $\hat{\sigma}_x$ and $\hat{\sigma}_y$ results in spin flips, while the interactions with $\hat{\sigma}_z$ conserves the spin.

It turns out to be smart to define a density matrix operator:

$$\hat{\rho} = \chi\chi^\dagger = \begin{pmatrix} |a|^2 & ab^* \\ ba^* & |b|^2 \end{pmatrix} = \frac{1}{2}(\mathcal{I} + \mathbf{P} \cdot \hat{\boldsymbol{\sigma}}), \tag{A.18}$$

where χ is the neutron wave function (Eq. A.5), and $\mathcal{I}$ is the unit matrix.

Using the density matrix the elastic cross section can be written as ([Lov84], Eq. 10.31):

$$\frac{d\sigma}{d\Omega} = \mathrm{Tr}\hat{\rho}\hat{v}^\dagger\hat{v} = \sum_{\lambda,\lambda'} p_\lambda \mathrm{Tr}\hat{\rho}\langle\lambda|\hat{V}^\dagger(\boldsymbol{\kappa})|\lambda'\rangle\langle\lambda'|\hat{V}(\boldsymbol{\kappa})|\lambda\rangle\delta(E_\lambda - E_{\lambda'}), \tag{A.19}$$

where $\hat{V}$ is the interaction potential and it is understood that the trace is to be taken with respect only to the neutron spin coordinates. The outgoing polarization is given

as:

$$\mathbf{P}'\frac{d\sigma}{d\Omega} = \mathrm{Tr}\hat{\rho}\hat{v}^\dagger\hat{\boldsymbol{\sigma}}\hat{v} = \sum_{\lambda,\lambda'} p_\lambda \mathrm{Tr}\hat{\rho}\langle\lambda|\hat{V}^\dagger(\boldsymbol{\kappa})|\lambda'\rangle\hat{\boldsymbol{\sigma}}\langle\lambda'|\hat{V}(\boldsymbol{\kappa})|\lambda\rangle\delta(E_\lambda - E_{\lambda'}) \tag{A.20}$$

Inserting Eq. A.16 in Eq. A.19 and Eq. A.20 results in the two master equations for polarized neutron scattering:

$$\mathrm{Tr}\hat{\rho}\hat{v}^\dagger\hat{v} = \hat{\boldsymbol{\alpha}}^\dagger\cdot\hat{\boldsymbol{\alpha}} + \hat{\beta}^\dagger\hat{\beta} + \hat{\beta}^\dagger(\hat{\boldsymbol{\alpha}}\cdot\mathbf{P}) + (\hat{\boldsymbol{\alpha}}^\dagger\cdot\mathbf{P})\hat{\beta} + i\mathbf{P}\cdot(\hat{\boldsymbol{\alpha}}^\dagger\times\hat{\boldsymbol{\alpha}}), \tag{A.21}$$

and

$$\mathrm{Tr}\hat{\rho}\hat{v}^\dagger\hat{\boldsymbol{\sigma}}\hat{v} = \hat{\beta}^\dagger\hat{\boldsymbol{\alpha}}+\hat{\boldsymbol{\alpha}}^\dagger\hat{\beta}+\hat{\beta}^\dagger\hat{\beta}\mathbf{P}+\hat{\boldsymbol{\alpha}}^\dagger(\hat{\boldsymbol{\alpha}}\cdot\mathbf{P})+(\hat{\boldsymbol{\alpha}}^\dagger\cdot\mathbf{P})\hat{\boldsymbol{\alpha}}-\mathbf{P}(\hat{\boldsymbol{\alpha}}^\dagger\cdot\hat{\boldsymbol{\alpha}})-i\hat{\boldsymbol{\alpha}}^\dagger\times\hat{\boldsymbol{\alpha}}+i\hat{\beta}^\dagger(\hat{\boldsymbol{\alpha}}\times\mathbf{P})+i(\mathbf{P}\times\hat{\boldsymbol{\alpha}}^\dagger)\hat{\beta}. \tag{A.22}$$

Based on these two equations and the interaction potentials all the results presented in the following are derived in [Lov84].

A.3.1. Example: Nuclear scattering

The nuclear scattering potential for a crystal is:

$$\hat{V}_N(\boldsymbol{\kappa}) = \sum_{\mathbf{l},\mathbf{d}} \exp(i\boldsymbol{\kappa}\cdot\mathbf{R}_{ld})(A_{ld} + \frac{1}{2}B_{ld}\hat{\boldsymbol{\sigma}}\cdot\hat{\mathbf{I}}_{ld}), \tag{A.23}$$

so that

$$\hat{\boldsymbol{\alpha}} = \sum_{\mathbf{l},\mathbf{d}} \exp(i\boldsymbol{\kappa}\cdot\mathbf{R}_{ld})\frac{1}{2}B_{ld}\hat{\mathbf{I}}_{ld} \tag{A.24}$$

$$\hat{\beta} = \sum_{\mathbf{l},\mathbf{d}} \exp(i\boldsymbol{\kappa}\cdot\mathbf{R}_{ld})A_{ld}, \tag{A.25}$$

where $\hat{\mathbf{I}}$ is the nuclear spin operator and the constants A and B are related to the nuclear scattering lengths b^+ and b^- as $A = ((I+1)b^+ + Ib^-)/(2I+1)$ and $B = (b^+ + b^-)/(2I+1)$.

To calculate the polarization cross section and outgoing polarization we have to average over the nuclear spin (which we assume is random oriented), so that terms linear in $\hat{\boldsymbol{\alpha}}$ (three last terms in Eq. A.21) disappears. The scattering cross section ends up being (see [Lov84] p. 159):

$$\frac{d\sigma}{d\Omega} = \sum_{\mathbf{l},\mathbf{d},\mathbf{l}',\mathbf{d}'} \exp(i\boldsymbol{\kappa}\cdot(\mathbf{R}_{ld} - \mathbf{R}_{l'd'}))(|\overline{A}_d|^2 + \delta_{\mathbf{l},\mathbf{l}'}\delta_{\mathbf{d},\mathbf{d}'}[\overline{|A_d|^2} - |\overline{A}_d|^2 + \frac{1}{4}\overline{|B_d|^2}I_d(I_d+1)]) \tag{A.26}$$

where the first term is the coherent cross-section and the second term is the site-incoherent cross-section. Both terms are independent of $\mathbf{P}$ as expected for a system without a preferred internal direction.

The polarization in the final state is:

$$\mathbf{P}'\frac{d\sigma}{d\Omega} = \sum_{\mathbf{l},\mathbf{d},\mathbf{l}',\mathbf{d}'} \exp(i\boldsymbol{\kappa}\cdot(\mathbf{R}_{ld}-\mathbf{R}_{l'd'}))\mathbf{P}'(|\overline{A}_d|^2+\delta_{\mathbf{l},\mathbf{l}'}\delta_{\mathbf{d},\mathbf{d}'}[\overline{|A_d|^2}-|\overline{A}_d|^2-\frac{1}{12}\overline{|B_d|^2}I_d(I_d+1)]) \tag{A.27}$$

Comparing Eq. A.26 and Eq. A.27 we find that: 1) The nuclear coherent polarization is the same as the initial polarization. 2) The same is true for the incoherent scattering due to the random isotope distribution. 3) The nuclear incoherent scattering due to the random nuclear spin orientations has polarization $\mathbf{P}' = -1/3\mathbf{P}$ (for a random nuclear spin the associated Pauli matrix will 2/3 of the time point in the direction of $\hat{\sigma}_x$ and $\hat{\sigma}_y$ which according to Eq. A.17 flips the spin).

For Vanadium, where there is only one isotope and coherent scattering is negligible, we find $\mathbf{P}' = -1/3\mathbf{P}$. There is however one catch. If the probability for multiple scattering is large one has to take into account that after two scattering one has: $\mathbf{P}'(2) = 1/9\mathbf{P}$, and so forth. The average polarization after a thick vanadium target is therefore a sum of different contributions.

A.3.2. Example: Polarizing Monochromator and Guides

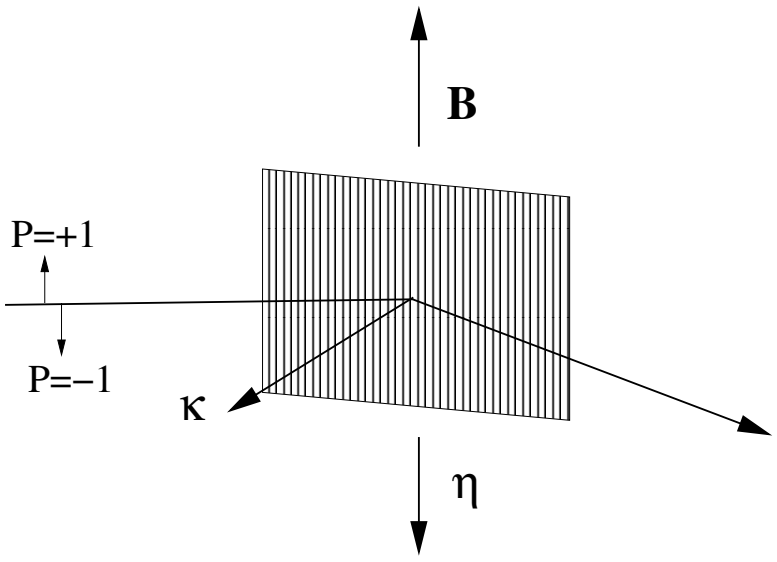


Figure A.1.: Principle and geometry of a polarizing monochromator.

In a polarized monochromator and polarizing guides we have a ferromagnetic crystal in an external magnetic field. The scattering potential is now both nuclear (no internal direction) and magnetic (internal direction), so in general the outgoing polarization can be quite complex. However, as illustrated in Figure A.1, the typical setup has many geometrical constraints: $\tilde{\boldsymbol{\eta}} \cdot \tilde{\boldsymbol{\kappa}} = 0$, $\tilde{\boldsymbol{\eta}} \cdot \mathbf{P}_\perp = \tilde{\boldsymbol{\eta}} \cdot \mathbf{P}$, and $\boldsymbol{\kappa} \times (\tilde{\boldsymbol{\eta}} \times \boldsymbol{\kappa}) = \tilde{\boldsymbol{\eta}}$, which simplifies the problem.

In [Lov84] the calculation for a centrosymmetric ferromagnetic crystal is done, and inserting the constraints above one finds ([Lov84], Eq. 10.96 and Eq. 10.110):

$$d\sigma/d\Omega = F_N(\boldsymbol{\kappa})^2 + 2F_N(\boldsymbol{\kappa})F_M(\boldsymbol{\kappa})(\mathbf{P} \cdot \tilde{\boldsymbol{\eta}}) + F_M(\boldsymbol{\kappa})^2 \quad \text{(A.28)}$$

$$\mathbf{P}' d\sigma/d\Omega = \mathbf{P}[F_N(\boldsymbol{\kappa})^2 - F_M(\boldsymbol{\kappa})^2] + \tilde{\boldsymbol{\eta}}[2F_N(\boldsymbol{\kappa})F_M(\boldsymbol{\kappa}) + 2(\tilde{\boldsymbol{\eta}} \cdot \mathbf{P})F_M(\boldsymbol{\kappa})^2] \quad \text{(A.29)}$$

NB! Note that in [Wil88] Eq. 2.2.25 there is a minus in front of the second term in Eq. A.28. We have not been able to understand this discrepancy, which is probably due to notation. Most other authors agree with the minus in front of the second term (e.g. Squires and Francis Tasset).

For a beam which is initially unpolarized we find the outgoing polarization to be:

$$\mathbf{P}' = \frac{\tilde{\boldsymbol{\eta}} 2F_N(\boldsymbol{\kappa})F_M(\boldsymbol{\kappa})}{d\sigma/d\Omega} = \frac{2F_N(\boldsymbol{\kappa})F_M(\boldsymbol{\kappa})}{F_N(\boldsymbol{\kappa})^2 + F_M(\boldsymbol{\kappa})^2}\tilde{\boldsymbol{\eta}}, \quad \text{(A.30)}$$

so that the beam is fully polarized along $\tilde{\boldsymbol{\eta}}$ if $F_N(\boldsymbol{\kappa}) = \pm F_M(\boldsymbol{\kappa})$.

What we use to characterize the polarizing monochromator in practice is not $F_N(\boldsymbol{\kappa})$ and $F_M(\boldsymbol{\kappa})$, but instead the reflection probabilities $R_\uparrow$ and $R_\downarrow$ (for the reflection of interest).

If we assume that the reflection probabilities are directly proportional to the cross sections (with proportionality constant k), i.e., $R_\uparrow = kd\sigma/d\Omega(\mathbf{P} = +\tilde{\boldsymbol{\eta}})$ and $R_\downarrow = kd\sigma/d\Omega(\mathbf{P} = -\tilde{\boldsymbol{\eta}})$ then we can use Eq. A.28 to determine $F_N(\boldsymbol{\kappa})$ and $F_M(\boldsymbol{\kappa})$:

$$R_\uparrow = k(F_N(\boldsymbol{\kappa}) + F_M(\boldsymbol{\kappa}))^2, \quad \text{(A.31)}$$

$$R_\downarrow = k(F_N(\boldsymbol{\kappa}) - F_M(\boldsymbol{\kappa}))^2. \quad \text{(A.32)}$$

The values of $\sqrt{k}F_N(\boldsymbol{\kappa})$ and $\sqrt{k}F_M(\boldsymbol{\kappa})$ are then between -1 and +1 and unit less like the reflection probabilities. In the following we ignore k and just talk about $F_N(\boldsymbol{\kappa})$ and $F_M(\boldsymbol{\kappa})$.

In principle there are four solutions for $F_N(\boldsymbol{\kappa})$ and $F_M(\boldsymbol{\kappa})$, so in the code we currently choose the values where $F_N(\boldsymbol{\kappa}) + F_M(\boldsymbol{\kappa}) = +\sqrt{R_\uparrow}$ and $F_N(\boldsymbol{\kappa}) - F_M(\boldsymbol{\kappa}) = +\sqrt{R_\downarrow}$ (so that $F_N(\boldsymbol{\kappa}) > 0$ and $F_N(\boldsymbol{\kappa}) > F_M(\boldsymbol{\kappa})$). We then find:

$$F_N(\boldsymbol{\kappa}) = \frac{\sqrt{R_\uparrow} + \sqrt{R_\downarrow}}{2}, \quad \text{(A.33)}$$

$$F_M(\boldsymbol{\kappa}) = \frac{\sqrt{R_\uparrow} - \sqrt{R_\downarrow}}{2}. \quad \text{(A.34)}$$

When $F_N(\boldsymbol{\kappa})$ and $F_M(\boldsymbol{\kappa})$ are determined from these equations, Eq. A.28 and Eq. A.29 can easily be used to handle any situation.

This solution is both used for monochromators and guides.

It is not clear that this solution is correct. If we make a simple example with $R_\uparrow = 1$ and $R_\downarrow = 0.25$ then we could in principle have four solutions, but let us just quote the two where $F_N(\boldsymbol{\kappa})$ is positive since the last two are found by inserting a minus before all the solutions and this does not change the physics. The two solutions are $F_N(\boldsymbol{\kappa}) = 0.75, F_M(\boldsymbol{\kappa}) = 0.25$ and $F_M(\boldsymbol{\kappa}) = 0.75, F_N(\boldsymbol{\kappa}) = 0.25$. All solutions gives the same cross section, but if the incoming beam is polarized (and only then) the outgoing beam will have two different polarization values, since $\mathbf{P}[F_N(\boldsymbol{\kappa})^2 - F_M(\boldsymbol{\kappa})^2]$ and $\tilde{\boldsymbol{\eta}} 2(\tilde{\boldsymbol{\eta}} \cdot \mathbf{P}) F_M(\boldsymbol{\kappa})^2$ are different for the two solutions. It seems that one needs some additional information to choose between the two solutions.

> NB! The simplifying geometry shown in Figure A.1 only applies for the sides of the guide wall and not the top and bottom (assuming that the magnetizing field is pointing up or down), so there another set of equations should really be used.

The same physics could also be used for a polarizing powder or single crystal sample if $F_N(\boldsymbol{\kappa})$ and $F_M(\boldsymbol{\kappa})$ can be calculated with some other program, but one would have to use the general form of Eq. A.28 and Eq. A.29 without the simplifying geometrical constraints for monochromators and guides.

A.4. New McStas Components

The components written so far can be divided into four groups:

- **Polarizers:** Components used to make the beam polarized.
- **Monitors:** Unphysical detectors that can measure the polarization of the neutrons.
- **Magnetic fields:** Components used to handle magnetic fields.
- **Samples:** Samples that affects the polarization.

A.4.1. Polarizers

Some of the most common ways of polarizing a beam have been implemented.

- **Set_pol:** This unphysical component can be used in two ways. Either to hard code the polarization to the vector (px, py, pz) or when randomOn!=0 to set the polarization vector to a random vector on the unit sphere.

- **Monochromator_pol:** A monochromator that only does the $n = 1$ reflection. For each neutron it calculates the wavelength which would give Bragg reflection, λ_{Bragg}, and it then calculates, based on one mosaicity and one d-spread, the reflection probability given the neutrons actual λ. The reflection probability is a Gaussian in $\Delta\lambda = \lambda - \lambda_{\text{Bragg}}$, with the peak reflectivity and polarization calculated as described in section A.3.2.

  NB! Note that this monochromator reflects the neutrons billiard-like. In **Monochromator_flat** the mosaicity of the reflecting crystal is taken into account, but the d-spread is not taken into account. One should implement d-spread and mosaicity in a way similar to what is done in **Single crystal**.

- **Pol_mirror:** Plane with a reflection probability for up and down. There are 3 options: always reflect, always transmit, or random select transmit/reflect.

  NB! Note that at the moment the plane only reflects from one side (because it uses PROP_Z0.

- **Pol_bender:** Curved guide with the possibilities to insert multiple slits, and have the end gap parallel to the entrance or following the guide angle. It is possible to select different coatings (mirror parameters) for each of the four sides.

- **Pol_guide_vmirror:** Straight guide with non-polarizing coatings with two polarizing super mirrors sitting in a V shape inside.

Note that for all the polarizing guides it is possible to define analytical functions or use tables for the up and down reflectivity descriptions.

A.4.2. Detectors

- **Pol_monitor:** One defines a vector $\mathbf{m} = (mx, my, mz)$ for the monitor and measures the projection of the spin along this vector i.e. $\mathbf{m} \cdot \mathbf{S}$.

- **PolLambda_monitor:** Measures the projection of the spin along the defined vector $\mathbf{m}$ (see **Pol_monitor**) as a function of the wavelength λ.

- **MeanPolLambda_monitor:** Measures the *average* projection of the spin along the defined vector $\mathbf{m}$ (see **Pol_monitor**) as a function of the wavelength λ.

  NB! currently the error on the mean is shown ($\sigma/\sqrt{(N)}$), but it might make more sense to show the spread (σ).

A.4.3. Magnetic fields

Much inspiration for the components and the tests have been found in [SD01].

- **Pol_constBfield:** A rectangular box with a constant magnetic field in the y-direction. The x- and z-components of the spin precess with the Larmor frequency ω_L. It is possible to define the field in terms of a wavelength so that the spin will precess 180 degrees for the given wavelength. The component can be rotated to have the field along another axis.

- **Pol_simpleBfield:** The first attempt at a component for handling general magnetic fields. It is a concentric component where you define a start and stop component for each field, but this allows other components, e.g. monitors, to be put inside the field. The component overloads the propagation routines so that numerical spin propagation is done for analytical magnetic fields.

  NB! At the moment both components does not really check the boundaries of the field on the sides, but merely assumes that the field starts at the entrance plane and stops at the exit plane.

  Also, some optimization remains for the numerical component and it would be nice to support tabulated magnetic field files. However, the framework developed for **Pol_simpleBfield** is very general and should easily facilitate these changes.

A.4.4. Samples

- **V_sample:** Modified the sample so that the scattered neutron has $\mathbf{P}' = -1/3\mathbf{P}$. Note that this component does not handle multiple scattering, so this approach is correct. If the components handled multiple scattering the polarization should be set to $\mathbf{P}' = (-1/3)^n\mathbf{P}$, where n is the number of scatterings.

A.5. Tests With New Components

All the test instruments can be found in the McStas examples folder (go to "Neutron site/tests" in mcgui).

There are basically two kind of tests. The first kind of tests shows that the polarizing component can reproduce the same results as a similar non-polarizing component:

- *Test_Monochromators.instr* : Intercomparison of **Monochromator_flat** and **Monochromator_pol**.

- *Test_Pol_Bender_Vs_Guide_Curved.instr* : Intercomparison of **Guide_curved** and **Pol_bender**.

The second type of test illustrates the polarizing capabilities of the component:

- *Test_Magnetic_Constant.instr* : Constant magnetic field.
- *Test_Magnetic_Majorana.instr* : Linearly decreasing field with small transverse component.
- *Test_Magnetic_Rotation.instr* : Rotating magnetic field.
- *Test_Magnetic_Userdefined.instr* : Example of how to make a user defined analytic magnetic field that can also depend on time.
- *Test_Pol_Bender.instr* : Illustrates beam polarization with the **Pol_bender**.
- *Test_Pol_Set.instr* : Tests **Pol_set**.
- *Test_Pol_Guide_Vmirror.instr* : Illustrates beam polarization with the **Pol_guide_vmirror**.
- *Test_Pol_Mirror.instr* : Illustrates beam polarization with the **Pol_mirror**.
- *Test_Pol_TripleAxis.instr* : An example of a triple axis spectrometer with polarizing monochromators, a vanadium sample, and a spin flipper.

B. Libraries and constants

The McStas Library contains a number of built-in functions and conversion constants which are useful when constructing components. These are stored in the `share` directory of the `MCSTAS` library.

Within these functions, the 'Run-time' part is available for all component/instrument descriptions. The other parts are dynamic, that is they are not pre-loaded, but only imported once when a component requests it using the `%include` McStas keyword. For instance, within a component C code block, (usually SHARE or DECLARE):

```
1  %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,

```
1  %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.

B.1. Run-time calls and functions (`mcstas-r`)

Here we list a number of preprogrammed macros which may ease the task of writing component and instrument definitions.

B.1.1. Neutron propagation

Propagation routines perform all necessary operations to transport neutron rays from one point to an other. Except when using the special `ALLOW_BACKPROP;` call prior to executing any `PROP_*` propagation, the neutron rays which have negative propagation times are removed automatically.

- **ABSORB**. This macro issues an order to the overall McStas simulator to interrupt the simulation of the current neutron history and to start a new one.
- **PROP_Z0**. Propagates the neutron to the $z = 0$ plane, by adjusting (x, y, z) and t accordingly from knowledge of the neutron velocity (vx, vy, vz). If the propagation time is negative, the neutron ray is absorbed, 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_DT` call.

- **PROP_DT**(dt). Propagates the neutron through the time interval dt, adjusting (x, y, z) and t accordingly from knowledge of the neutron velocity. This macro automatically calls PROP_GRAV_DT when the `--gravitation` option has been set for the whole simulation.

- **PROP_GRAV_DT**(dt, Ax, Ay, Az). Like **PROP_DT**, but it also includes gravity using the acceleration (Ax, Ay, Az). In addition to adjusting (x, y, z) and t, also (vx, vy, vz) is modified.

- **ALLOW_BACKPROP**. Indicates that the next propagation routine will not remove the neutron ray, even if negative propagation times are found. Subsequent propagations are not affected.

- **SCATTER**. This macro is used to denote a scattering event inside a component. It should be used e.g to indicate that a component has interacted with the neutron ray (e.g. scattered or detected). This does not affect the simulation (see, however, **Beamstop**), and it is mainly used by the `MCDISPLAY` section and the `GROUP` modifier See also the SCATTERED variable (below).

B.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_NEUTRON**($index, x, y, z, vx, vy, vz, t, sx, sy, sz, p$) stores the current neutron state in the trace-history table, in local coordinate system. *index* is usually INDEX_CURRENT_COMP. This is automatically done when entering each component of an instrument.

- **RESTORE_NEUTRON**($index, x, y, z, vx, vy, vz, t, sx, sy, sz, p$) restores the neutron state to the one at the input of the component *index*. To ignore a component effect, use RESTORE_NEUTRON (INDEX_CURRENT_COMP,
  $x, y, z, vx, vy, vz, t, sx, sy, sz, p$) at the end of its TRACE section, or in its EXTEND section. These neutron states are in the local component coordinate systems.

- **SCATTERED** is a variable set to 0 when entering a component, which is incremented each time a SCATTER event occurs. This may be used in the `EXTEND` sections to determine whether the component interacted with the current neutron ray.

- **extend_list**(n, &*arr*, &*len*, *elemsize*). Given an array *arr* with *len* elements each of size *elemsize*, make sure that the array is big enough to hold at least n elements, by extending *arr* and *len* if necessary. Typically used when reading a list of numbers from a data file when the length of the file is not known in advance.

- **mcset_ncount**(n). Sets the number of neutron histories to simulate to n.

- **mcget_ncount**(). Returns the number of neutron histories to simulate (usually set by option `-n`).

- **mcget_run_num**(). Returns the number of neutron histories that have been simulated until now.

B.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 $t.a$

B.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_1, &t_2 A, B, C). Solves the 2^{nd} order equation $At^2+Bt+C = 0$ and returns the solutions into pointers $*t_1$ and $*t_2$. if t_2 is specified as `NULL`, only the smallest positive solution is returned in t_1.

(* The experienced c-programmer may be puzzled that these routines can return information without the use of *pass by reference*, the reason is that these calls are implemented as macros / `#define` wrapped functions.)

B.1.5. Output from detectors

Details about using these functions are given in the McStas User Manual.

- **DETECTOR_OUT_0D**(...). Used to output the results from a single detector. The name of the detector is output together with the simulated intensity and estimated statistical error. The output is produced in a format that can be read by McStas front-end programs.
- **DETECTOR_OUT_1D**(...). Used to output the results from a one-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.

B.1.6. Ray-geometry intersections

- **inside_rectangle**(x, y, 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**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, d_x, d_y, d_z). Calculates the (0, 1, or 2) intersections between the neutron path and a box of dimensions d_x, d_y, and d_z, centered at the origin for a neutron with the parameters (x, y, z, v_x, v_y, v_z). The times of intersection are returned in the variables t_1 and t_2, with $t_1 < t_2$. In the case of less than two intersections, t_1 (and possibly t_2) are set to zero. The function returns true if the neutron intersects the box, false otherwise.
- **cylinder_intersect**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, r, h). Similar to **box_intersect**, but using a cylinder of height h and radius r, centered at the origin.
- **sphere_intersect**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, r). Similar to **box_intersect**, but using a sphere of radius r.

B.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. [Pre+86, 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 neutron is returned in $d\Omega$. This routine was previously called **randvec_target_sphere** (which still works).

- **randvec_target_rect_angular**(&v_x, &v_y, &v_z, &$d\Omega$, aim_x, aim_y, aim_z,h, w, Rot) does the same as randvec_target_circle but targetting at a rectangle with angular dimensions h and w (in **radians**, not in degrees as other angles). The rotation matrix *Rot* is the coordinate system orientation in the absolute frame, usually ROT_A_CURRENT_COMP.

- **randvec_target_rect**(&v_x, &v_y, &v_z, &$d\Omega$, aim_x, aim_y, aim_z,$height, width, Rot$) is the same as randvec_target_rect_angular but *height* and *width* dimensions are given in meters. This function is useful to e.g. target at a guide entry window or analyzer blade.

B.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 get element [i,j] */
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 concatenate 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 'MCSTAS/data/BeO.trm' (Transmission of a Berylium filter) looks like:

```
# BeO transmission, as measured on IN12
# Thickness: 0.05 [m]
# [ k(Angs-1) Transmission (0-1) ]
# wavevector multiply
1.0500  0.74441
1.0750  0.76727
1.1000  0.80680
...
```

Binary files should be of type "float" (i.e. REAL*32) and "double" (i.e. REAL*64), and should *not* contain text header lines. These files are platform dependent (little or big endian).

The *filename* is first searched into the current directory (and all user additional locations specified using the `-I` option, see the 'Running McStas' chapter in the User Manual), and if not found, in the `data` sub-directory of the `MCSTAS` library location. This way, you do not need to have local copies of the McStas Library Data files (see table 1.1).

A usage example for this library part may be:

```
t_Table Table;        // declare a t_Table structure
char file[]="BeO.trm"; // a file name
double x,y;

Table_Read(&Table, file, 1); // initialize and read the first numerical
    block
Table_Info(Table);           // display table informations
...
x = Table_Index(Table, 2,5); // read the 3rd row, 6th column element
                             // of the table. Indexes start at zero in C
                                .
y = Table_Value(Table, 1.45,1); // look for value 1.45 in 1st column (x
    axis)
                             // and extract 2nd column value of that row
Table_Free(&Table);          // free allocated memory for table
```

Additionally, if the block number (3rd) argument of **Table_Read** is 0, all blocks will be concatenated. 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;
```

```
3   double y;
4
5   Table = Table_Read_Array("file.dat", &n); // initialize and read the all
        numerical block
6   n = Table_Info_Array(Table);      // display informations for all blocks (
        also returns n)
7
8   y = Table_Index(Table[0], 2,5);   // read in 1st block the 3rd row, 6th
        column element
9                                     // ONLY use Table[i] with i < n !
10  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.

B.3. Monitor_nD Library

This library gathers a few functions used by a set of monitors e.g. Monitor_nD, Res_monitor, Virtual_output, etc. It may monitor any kind of data, create the data files, and may display many geometries (for `mcdisplay`). Refer to these components for implementation examples, and ask the authors for more details.

B.4. Adaptive importance sampling Library

This library is currently only used by the components **Source_adapt** and **Adapt_check**. It performs adaptive importance sampling of neutrons for simulation efficiency optimization. Refer to these components for implementation examples, and ask the authors for more details.

B.5. Vitess import/export Library

This library is used by the components **Vitess_input** and **Vitess_output**, as well as the `mcstas2vitess` utility. Refer to these components for implementation examples, and ask the authors for more details.

B.6. Constants for unit conversion etc.

The following predefined constants are useful for conversion between units

Name	Value	Conversion from	Conversion to
DEG2RAD	$2\pi/360$	Degrees	Radians
RAD2DEG	$360/(2\pi)$	Radians	Degrees
MIN2RAD	$2\pi/(360 \cdot 60)$	Minutes of arc	Radians
RAD2MIN	$(360 \cdot 60)/(2\pi)$	Radians	Minutes of arc
V2K	$10^{10} \cdot m_{\mathrm{N}}/\hbar$	Velocity (m/s)	**k**-vector ($\mathrm{\AA}^{-1}$)
K2V	$10^{-10} \cdot \hbar/m_{\mathrm{N}}$	**k**-vector ($\mathrm{\AA}^{-1}$)	Velocity (m/s)
VS2E	$m_{\mathrm{N}}/(2e)$	Velocity squared ($\mathrm{m}^2\ \mathrm{s}^{-2}$)	Neutron energy (meV)
SE2V	$\sqrt{2e/m_{\mathrm{N}}}$	Square root of neutron energy ($\mathrm{meV}^{1/2}$)	Velocity (m/s)
FWHM2RMS	$1/\sqrt{8\log(2)}$	Full width half maximum	Root mean square (standard deviation)
RMS2FWHM	$\sqrt{8\log(2)}$	Root mean square (standard deviation)	Full width half maximum
MNEUTRON	$1.67492 \cdot 10^{-27}$ kg	Neutron mass, m_{n}	
HBAR	$1.05459 \cdot 10^{-34}$ Js	Planck constant, $\hbar$	
PI	3.14159265...	π	
FLT_MAX	3.40282347E+38F	a big float value	

C. The McStas terminology

This is a short explanation of phrases and terms which have a specific meaning within McStas. We have tried to keep the list as short as possible with the risk that the reader may occasionally miss an explanation. In this case, you are more than welcome to contact the McStas core team.

- **Arm** A generic McStas component which defines a frame of reference for other components.
- **Component** One unit (*e.g.* optical element) in a neutron spectrometer. 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 McStas components defining a neutron spectrometer.
- **Kernel** The McStas meta-language definition and the associated compiler `mcstas`.
- **McStas** Monte Carlo Simulation of Triple Axis Spectrometers (the name of this package). Pronunciation ranges from *mex-tas*, to *mac-stas* and *m-c-stas*.
- **Output parameter** An output parameter for a component. For example the counts in a monitor. An output parameter may be accessed from the instrument in which the component is used using `MC_GETPAR`.
- **Run-time** C code, contained in the files `mcstas-r.c` and `mcstas-r.h` included in the McStas distribution, that declare functions and variables used by the generated simulations.
- **Setting parameter** Similar to a definition parameter, but with the restriction that the value of the parameter must be a number.

Bibliography

[Mcs] See http://www.mcstas.org (cit. on pp. 10, 17).

[Git] See https://github.com/McStasMcXtrace/McCode/issues (cit. on pp. 10, 17).

[Nmi] See http://neutron-eu.net/en (cit. on p. 10).

[Mcna] See http://mcnsi.risoe.dk (cit. on p. 10).

[Ts2] See http://ts-2.isis.rl.ac.uk (cit. on p. 10).

[Ess] See http://www.ess-europe.de (cit. on p. 10).

[Sin] See http://sine2020.eu (cit. on p. 10).

[Ics] ICSD, Inorganic Crystal Structure Database. See http://icsd.ill.fr (cit. on pp. 15, 16, 93).

[Tri] See http://www.nea.fr/html/dbprog/tripoli-abs.html (cit. on p. 39).

[Mcnb] See http://mcnpx.lanl.gov and http://mcnp.lanl.gov (cit. on p. 39).

[Vit] See http://www.hmi.de/projects/ess/vitess (cit. on p. 39).

[Cry] Crystallographica, Oxford Cryosystem, 1998. See http://www.crystallographica.com (cit. on p. 92).

[Ful] Fullprof powder refinement. See http://www-llb.cea.fr/fullweb/fp2k/fp2k.htm (cit. on p. 93).

[Bac75] G.E. Bacon. *Neutron Diffraction*. Oxford University Press, 1975 (cit. on pp. 75, 82, 97).

[Bla83] Y. Blanc. In: *ILL Report* 83BL21G (1983) (cit. on p. 59).

[Cla+98] K. N. Clausen et al. "The RITA spectrometer at Risø - design considerations and recent results". In: *Physica B* 241-243 (1998), pp. 50–55 (cit. on p. 46).

[Cop74] J.R.D Copley. In: *Comput. Phys. Commun.* 7 (1974), p. 289 (cit. on pp. 100, 104–106).

[Cop93] J.R.D. Copley. In: *J. Neut. Research* 1 (1993), p. 21 (cit. on p. 18).

[Cus03] L. D. Cussen. In: *J. Appl. Cryst.* 36 (2003), p. 1204 (cit. on p. 117).

[DL03] A.-J. Dianoux and G. Lander. *ILL Neutron Data Booklet*. OCP Science, 2003 (cit. on pp. 15, 16, 94, 104).

[Ege67] P.A. Egelstaff. *An introduction to the liquid state*. Academic Press, London, 1967 (cit. on p. 102).

[FMW72] Bischoff FG, Yeater ML, and Moore WE. In: *Nuclear Science and Engineering* 48 (1972), p. 266 (cit. on pp. 100, 103, 104, 106).

[Far+02] E. Farhi et al. In: *Appl. Phys.* A 74 (2002), S1471 (cit. on p. 18).

[FBS06] H.E. Fischer, A.C. Barnes, and P.S. Salmon. In: *Rep. Prog. Phys.* 69 (2006), p. 233 (cit. on p. 102).

[GRR92] Grimmett, G. R., and Stirzakerand D. R. *Probability and Random Processes, 2nd Edition*. Clarendon Press, Oxford, 1992 (cit. on p. 18).

[Jam80] F. James. In: *Rep. Prog. Phys.* 43 (1980), p. 1145 (cit. on pp. 18, 22).

[Johrc] M.W. Johnson. In: *Harwell report* AERE - R 7682 (March 1974) (cit. on pp. 100, 104–106).

[LN99] K. Lefmann and K. Nielsen. "McStas, a general software package for neutron ray-tracing simulations". In: *Neutron News* 10 (1999), pp. 20–23. DOI: 10.1080/10448639908233684 (cit. on p. 10).

[Lie05] K. Lieutenant. In: *J. Phys.: Condens. Matter* 17 (2005), S167 (cit. on p. 18).

[Lov84] S.W. Lovesey. *Theory of neutron scattering from condensed matter*. Oxford Clarendon Press, 1984 (cit. on pp. 133, 136, 137, 139).

[MPC77] D.F.R. Mildner, C.A. Pellizari, and J.M. Carpenter. In: *Acta. Cryst. A* 33 (1977), p. 954 (cit. on p. 104).

[Pre+86] W. H. Press et al. *Numerical Recipes in C*. Cambridge University Press, 1986 (cit. on p. 149).

[Pre+02] W.H. Press et al. *Numerical Recipes (2nd Edition)*. Cambridge University Press, 2002 (cit. on p. 59).

[Sch+04] C. Schanzer et al. In: *Nucl. Instr. Meth.* A 529 (2004), p. 63 (cit. on p. 18).

[Sea75] V.F. Sears. In: *Adv. Phys.* 24 (1975), p. 1 (cit. on p. 103).

[SD01] P. A. Seeger and L. L. Daemen. "Numerical Solution of Bloch's Equation for Neutron Spin Precession". In: *Nucl. Instr. Meth.* 457 (2001), p. 338 (cit. on pp. 134, 142).

[Squ78] G.L. Squires. *Thermal Neutron Scattering*. Cambridge University Press, 1978 (cit. on pp. 83, 85, 96, 101, 107).

[Wil+14] Peter Kjær Willendrup et al. "McStas: Past, present and future". In: *Journal of Neutron Research* 17.1 (2014), pp. 35–43. ISSN: 1023-8166. DOI: 10.3233/JNR-130004 (cit. on p. 10).

[Wil+05] Peter Willendrup et al. *User and Programmers Guide to the Neutron Ray-Tracing Package McStas, Version 1.9*. Risoe Report, 2005 (cit. on pp. 10, 17).

[Wil88] W. Gavin Williams. *Polarized Neutrons*. Oxford Clarendon Press, 1988 (cit. on pp. 133, 136, 139).

[ZLa04] G. Zsigmond, K. Lieutenant, and S. Manoshin et al. In: *Nucl. Instr. Meth.* A 529 (2004), p. 218 (cit. on p. 18).

Index