Risø–R–1xyz(EN)

Component Manual for the Neutron Ray-Tracing Package McStas, Version 1.8

Peter Kjær Willendrup, Kim Lefmann, and Emmanuel Farhi

Risø National Laboratory, Roskilde, Denmark
October 2004

Abstract

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

This report constitutes the component manual for McStas, and, together with the manual for the McStas system, it contains full documentation of all aspects of the program. It covers a description of all official components of the McStaspackage with some theoretical background. Further, it describes selected test instruments and show representative McStassimulations performed with these instruments.

This library component report documents McStasversion 1.8, released May 15th, 2003

The authors are:

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

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

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

Mark Hagen `<mhz@ansto.gov.au>`
Australian Nuclear Science and technology Organization, Sydney, Australia

as well as authors who left the project:

Per-Olof Åstrand `<per-olof.aastrand@risoe.dk>`
Materials Research Department, Risø National Laboratory, Roskilde, Denmark

Kristian Nielsen `<kn@sifira.dk>`
Materials Research Department, Risø National Laboratory, Roskilde, Denmark
Present address: Sifira A/S, Copenhagen, Denmark,

and many users that have contributed to components of the McStaslibrary.

ISBN 87–550–2929–9
ISBN 87–550–2930–2 (Internet)
ISSN 0106–2840
Pitney Bowes Management Services Denmark A/S · Risø National Laboratory · 2004

Contents

Preface and acknowledgements

This document contains information on the neutron scattering components which are the building blocks for describing instruments in the Monte Carlo neutron ray-tracing program McStas version 1.8. This is an update to the initial release in October 1998 of version 1.0 as presented in Ref. [1]. The reader of this document is not supposed to have specific knowledge of neutron scattering, but some basic understanding of the underlying physics is helpful in understanding the theoretical background for the component functionality. For details about simulation techniques, we refer to the McStas system manual [2]. We assume familiarity with the use of the C programming language.

We especially like to thank Kristian Nielsen for laying a solid foundation for the McStas system, which the authors of this manual are using daily. It is also a pleasure to thank Prof. Kurt N. Clausen for his continuous support to McStas and for having initiated the project. We have also 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 any errors, questions, suggestions, do not hesitate to contact the authors at `mcstas@risoe.dk` or consult the McStas WWW home page [3].

Important developments on the component side in McStas version 1.8 as compared to version 1.4 (the last version of the component manual) are listed in section 2.4.

This project has been supported by the European Union, initially through the “XENNI” network and “Cool Neutrons” RTD program, later through the “SCANS” network, and today the “MCNSI“ network package.

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.

Chapter 1

Introduction to McStas

Efficient design and optimization of neutron spectrometers are formidable challenges. Monte Carlo techniques are well matched to meet these challenges. When McStas version 1.0 was released in October 1998, except for the NISP/MCLib program [4], no existing package offered a general framework for the neutron scattering community to tackle the problems currently faced at reactor and spallation sources. The McStas project was designed to provide such a framework.

McStas is a fast and versatile software tool for neutron ray-tracing simulations. It is based on a meta-language specially designed for neutron simulation. Specifications are written in this language by users and automatically translated into efficient simulation codes in ANSI-C. The present version supports both continuous and pulsed source instruments, and includes a library of standard components with in total around 90 components. These enable to simulate all kinds of neutron scattering instruments (diffractometers, triple-axis, reflectometers, time-of-flight, small-angle, back-scattering,...).

The McStas package is written in ANSI-C and is freely available for down-load from the McStas web-page [3]. The package is actively being developed and supported by Risø National Laboratory and the Institut Laue Langevin. The system is well tested and is supplied with several examples and with an extensive documentation, including a separate component manual.

1.1 Background

The McStas project is the main part of a major effort in Monte Carlo simulations for neutron scattering at Risø National Laboratory. Simulation tools were urgently needed, not only to better utilize existing instruments (*e.g.* RITA-1 and RITA-2 [5–7]), but also to plan completely new instruments for new sources (*e.g.* the Spallation Neutron Source, SNS [8] and the European Spallation Source, ESS [9]). Writing programs in C or Fortran for each of the different cases involves a huge effort, with debugging presenting particularly difficult problems. A higher level tool specially designed for the needs of simulating neutron instruments is needed. As there was no existing simulation software that would fulfill our needs, the McStas project was initiated. In addition, the ILL required an efficient and general simulation package in order to achieve renewal of its instruments and guides. A significant contribution to both the component library and the McStas kernel itself was developed at the ILL and included in the package, in agreement with the original McStas authors.

1.1.1 The goals of McStas

Initially, the McStas project had four main objectives that determined the design of the McStas software.

Correctness. It is essential to minimize the potential for bugs in computer simulations. If a word processing program contains bugs, it will produce bad-looking output or may even crash. This is a nuisance, but at least you know that something is wrong. However, if a simulation contains bugs it produces wrong results, and unless the results are far off, you may not know about it! Complex simulations involve hundreds or even thousands of lines of formulae, making debugging a major issue. Thus the system should be designed from the start to help minimize the potential for bugs to be introduced in the first place, and provide good tools for testing to maximize the chances of finding existing bugs.

Flexibility. When you commit yourself to using a tool for an important project, you need to know if the tool will satisfy not only your present, but also your future requirements. The tool must not have fundamental limitations that restrict its potential usage. Thus the McStas systems needs to be flexible enough to simulate different kinds of instruments (e.g. triple-axis, time-of-flight and possible hybrids) as well as many different kind of optical components, and it must also be extensible so that future, as yet unforeseen, needs can be satisfied.

Power. "*Simple things should be simple; complex things should be possible*". New ideas should be easy to try out, and the time from thought to action should be as short as possible. If you are faced with the prospect of programming for two weeks before getting any results on a new idea, you will most likely drop it. Ideally, if you have a good idea at lunch time, the simulation should be running in the afternoon.

Efficiency. Monte Carlo simulations are computationally intensive, hardware capacities are finite (albeit impressive), and humans are impatient. Thus the system must assist in producing simulations that run as fast as possible, without placing unreasonable burdens on the user in order to achieve this.

1.2 The design of McStas

In order to meet these ambitious goals, it was decided that McStas should be based on its own meta-language, specially designed for simulating neutron scattering instruments. Simulations are written in this meta-language by the user, and the McStas compiler automatically translates them into efficient simulation programs written in ANSI-C.

In realizing the design of McStas, the task was separated into four conceptual layers:

1. Modeling the physical processes of neutron scattering, *i.e.* the calculation of the fate of a neutron that passes through the individual components of the instrument (absorption, scattering at a particular angle, etc.)
2. Modeling of the overall instrument geometry, mainly consisting of the type and position of the individual components.
3. Accurate calculation, using Monte Carlo techniques, of instrument properties such as resolution function from the result of ray-tracing of a large number of neutrons. This includes estimating the accuracy of the calculation.
4. Presentation of the calculations, graphical or otherwise.

Though obviously interrelated, these four layers can be treated independently, and this is reflected in the overall system architecture of McStas. The user will in many situations be interested in knowing the details only in some of the layers. For example, one user may merely look at some results prepared by others, without worrying about the details of the calculation. Another user may simulate a new instrument without having to reinvent the code for simulating the individual components in the instrument. A third user may write an intricate simulation of a complex component, e.g. a detailed description of a rotating velocity selector, and expect other users to easily benefit from his/her work, and so on. McStas attempts to make it possible to work at any combination of layers in isolation by separating the layers as much as possible in the design of the system and in the meta-language in which simulations are written.

The usage of a special meta-language and an automatic compiler has several advantages over writing a big monolithic program or a set of library functions in C, Fortran, or another general-purpose programming language. The meta-language is more *powerful*; specifications are much simpler to write and easier to read when the syntax of the specification language reflects the problem domain. For example, the geometry of instruments would be much more complex if it were specified in C code with static arrays and pointers. The compiler can also take care of the low-level details of interfacing the various parts of the specification with the underlying C implementation language and each other. This way, users do not need to know about McStas internals to write new component or instrument definitions, and even if those internals change in later versions of McStas, existing definitions can be used without modification.

The McStas system also utilizes the meta-language to let the McStas compiler generate as much code as possible automatically, letting the compiler handle some of the things that would otherwise be the task of the user/programmer. *Correctness* is improved by having a well-tested compiler generate code that would otherwise need to be specially written and debugged by the user for every instrument or component. *Efficiency* is also improved by letting the compiler optimize the generated code in ways that would be time-consuming or difficult for humans to do. Furthermore, the compiler can generate several different simulations from the same specification, for example to optimize the simulations in different ways, to generate a simulation that graphically displays neutron trajectories, and possibly other things in the future that were not even considered when the original instrument specification was written.

The design of McStas makes it well suited for doing "what if..." types of simulations. Once an instrument has been defined, questions such as "what if a slit was inserted", "what if a focusing monochromator was used instead of a flat one", "what if the sample was offset 2 mm from the center of the axis" and so on are easy to answer. Within minutes the instrument definition can be modified and a new simulation program generated. It also makes it simple to debug new components. A test instrument definition may be written containing a neutron source, the component to be tested, and whatever detectors are useful, and the component can be thoroughly tested before being used in a complex simulation with many different components.

The McStas system is based on ANSI-C, making it both efficient and portable. The meta-language allows the user to embed arbitrary C code in the specifications. *Flexibility* is thus ensured since the full power of the C language is available if needed.

1.3 Overview

This McStas Component Manual consists of the following major parts:

- A short list of new features introduced in this McStas release appears in chapter 2
- A list of available components from the library in chapter 3, sorted by categories.
- A list of example instruments from the library in chapter 12.
- The McStas library functions and definitions that aid in the writing of simulations and components are descripbed in appendix A.

An explanation of McStas terminology can be found in appendix B, and a list of library calls that may be used in component definitions appears in appendix A of the Manual.

Plans for future extensions are presented on the McStas web-page [3].

Chapter 2

New features in McStas versions 1.7 - 1.8

Many new features have been implemented in McStas version 1.8 since version 1.5 (no version 1.6. manual was written, the most important being that McStas can now compile and run in a Windows enviroment. The list of changes given here may be particularly useful to experienced McStas users who do not wish to read the whole manual, searching only for new features. A global upward compatibility exists for all changes, and thus all previous components and instruments should work as before. Changes are labelled as 1.7 or 1.8 to indicate in which McStas version they appeared.

Important note for Windows users: It is a known problem that some of the McStas tools do not support filenames / directories with spaces. We are working on a more general approach to this problem, which will hopefully be solved in the next release.

2.1 General

As of McStas version 1.8 the license covering the McStas package is the GNU General Pubclic License. See `http://www.gnu.org` for details.

2.2 Kernel

The following changes concern the 'Kernel' (i.e. the McStas meta-language and program). See the dedicated chapter in the *User manual* for more details.

- McStas can now compile for Windows!
- Instrument parameters may now have default values, the same way as components do (appeared in 1.8).
- An instrument source file may contain `EXTEND` %{ ...}% C blocks just after the usual `AT` ...`ROTATED` ...keywords, to extend the behaviour of existing components, without touching their code. All local component variables are available. This may for instance be used to add a new 'color' to neutrons, i.e. assign a new characteristic variable to the neutron (appeared in 1.7).

- Component instances in an instrument file may be GROUP'ed into exclusive assembly, i.e. only one component of the group will intercept the neutron ray, the rest will be skipped. This is useful for multi monochromators multi detectors, multiple collimators, etc. After the ROTATED keyword, the keyword GROUP should be added followed by a group name (e.g. GROUP MyGroup) (appeared in 1.7).
- The instrument and components may have string (`char*`) setting parameters. For components, their length is limited to 1024 characters (appeared in 1.7).
- In both components and instruments, the FINALLY section, that is executed at the end of simulations, has been supplemented with a new SAVE section. This latter is executed at simulation end (just before the FINALLY section), but also each time an intermediate save is required (e.g. when a 'kill -USR2 $pid' is used under Unix, see section 2.3) (appeared in 1.7).
- Components may have a SHARE section, which is imported only once per type of component. SHARE has the same role as DECLARE, but is useful when several instances of the same component is used in a single simulation (appeared in 1.7).
- The component files may have some `%include` inside '%{ }%' DECLARE or SHARE C blocks. The files to include are searched locally, and then in the library. If an extension is found, only the specified file is included, else both .h and .c are embedded unless the –no-runtime has been specified. As in previous releases, the instrument files can embed external files, both in C blocks and in the instrument parts (DECLARE, etc.) (appeared in 1.7).
- The PREVIOUS keyword, to be used after RELATIVE in place of component instance names refers to the preceeding component, and does not require to actually know its name. Similarly the PREVIOUS(n) keyword refers to the n-th preceeding component (appeared in 1.8).

2.3 Run-time

Some important modifications were done to the 'Run-time' library (i.e. the functions used in the instrument program). Some details may be found in *Running* chapter of the *User manual* as well as in the appendix A.

- A global gravitation handling is now available, by setting the `-g` flag.
- Many output formats are available for data. Use the `--format="format"` flag, e.g. `--format="Scilab"`. The default format is McStas/PGPLOT, but may be specified globally using the MCSTAS_FORMAT environment variable. See section **??** for details (appeared in 1.7).
- It is possible to save 3D data arrays, by calling the DETECTOR_OUT_3D macro. (handled as 2D by `mcplot` by ignoring the 3rd dimension) (appeared in 1.7).
- The C type of the 'number of events' array in monitors (usually named L_N) was changed from `int` to `double`, to avoid overflow. All 'home-made' monitors should be updated accordingly (appeared in 1.7).

- The USR2 signal generates an intermediate save for all monitors, during the simulation (executes the SAVE section). The USR1 signal still gives informations (appeared in 1.7).
- New randvec_target_rect and randvec_target_rect_angular functions now focus on a rectangle (more efficient than the former randvec_target_sphere) (appeared in 1.7 and 1.8).

2.4 Components and Library

We here list some of the new components (found in the McStas lib directory) which are detailed in the *Component manual*, also mentioned in the *Component Overview* of the *User Manual*.

- All components now have a valid header for mcdoc automatic documentation, as well as usage example.
- contrib This directory now contains contributed components. Those that will be found to be highly stable will then go into the other component categories. Contributed components (Guide_honeycomb, Guide_tapering, Guide_curved, ISIS_moderator) have been placed there (appeared in 1.7 and 1.8).
- data This directory now contains example data files mainly for Laue diffraction, transmission and reflection files.
- doc The documentation is now included in the doc directory.
- example A new example directory contains instrument examples, available from the mcgui 'Neutron site' menu Tool.
- misc Updated Vitess_input and Vitess_output components according to Vitess ¿= 2.3 Neutron structure.
- misc A Progress_bar component may be positioned anywhere in the simulation and displays the simulation percentage. It may also save temporary data files regularly so that one can have a look (using mcplot) at the results on-the-fly before the end of the simulation.
- monitors The Monitor_nD component may be used as a replacement for all monitors. It may have automatic limits mode for either all or selected monitored variables. It may also plot banana monitors for mcdisplay and monitor something else than the intensity, e.g. the mean energy on a XY psd (appeared in 1.7). A bug (atan2) was corrected in PSD_monitor_4PI.
- obsolete Some components were renamed by categories for better sorting. Old components are still available, but not maintained anymore. We highly encourage to avoid using these when writing new instrument definitions.
- optics components were renamed by categories, starting with Guide_..., Monochromator_..., Filter_...etc so that sorting is easier (appeared in 1.7).

- `optics` Monochromator_curved can read reflectivity and transmission tables (appeared in 1.7).

- `optics` Guide_gravity can handle a 2D array of channels, and has options for subdivisions in length, chamfers and wavyness.

- `optics` Filter_gen can read a table from a file and affect the neutron beam (replaces the obsolete `Flux_adapter`). It may act as a filter or a source (appeared in 1.7).

- `optics` Bugs were corrected in the following components: Bender, Chopper, Guide_channeled. Some similar components were gathered. New components were added: Guide_gravity, Guide_wavy, Filter_gen.

- `samples` can now target towards any component, given its index (no need to compute `target_x|y|z` vector, use e.g. `target_index=1`). Position an `Arm` at the focusing position when targetting to centered components (appeared in 1.7).

- `samples` Rectangular focusing has been implemented (instead of circular) in most components. More sample shapes are available. A bug has been corrected in Single_crystal. A Powder2 component (2 rings) has been added. Sans_spheres is a new sample component for small angle scattering. Phonon_simple is a new sample for phonon scattering. (appeared in 1.8).

- `share` Many dedicated libraries are now available as shared code for reading tables, handling data files and monitors. These are C functions to be `%included` into components (see e.g. `MCSTAS/monitors/Monitor_nD.comp`) (appeared in 1.7).

- `sources` The `Source_gen` and `Source_Maxwell_3` components may now simulate all kinds of continuous neutron sources. Some bugs were corrected in most sources in order to have an isotropic neutron emission. The `Virtual_input` and `Virtual_output` may load and save neutron event files (beware the size of the generated files !). Format may be text or binary (appeared in 1.7).

2.5 Documentation

As of McStas 1.8 an improved tutorial has been integrated into to the graphical user interface of McStasṪhe content of the tutorial is also available in appendix **??**.

2.6 Example instruments

As mentioned above, McStas 1.8 has various improvements regarding distribution of instruments. Firstly, `mcgui` has a new special menu for instruments distributed with McStasand secondly instruments can now have default values and thirdly a new McStas tutorial has integrated into the user interface. These three new features together proves a solid starting point for new users of McStasṪo further increase the value of the package, we have decided to include selveral example instruments contributed by McStas users. The full list of example instruments are

- `vanadium_example`, improved version used in the McStas tutorial.
- `linup-1-linup-7`, legacy instruments of the Risø TAS1 instrument from a historical release of McStas.
- `h8_test`, model of the Brookhaven H8 triple-axis, written by McStas team member Emmanuel Farhi
- `ILL_D9`, a model of the ILL D9 hot four-circle diffractometer, contributed by Chris Ling
- `Hetfull`, a model of the ISIS HET spectrometer, contributed by Dickon Champion
- `focus_psi`, a model of the PSI FOCUS spectrometer, contributed by Uwe Filges
- `prisma2`, a model of the ISIS prisma2 spectrometer, written by Kristian Nielsen and Mark Hagen from a historical release of McStas.
- Test instruments `ISIStest` for testing the new `ISIS_moderator` component, `SANS` for testing the new `Sans_spheres` component and `Test_Phonon` for testing the new `Phonon_simple` component, all contributed by the corresponding component authors.

2.7 Tools, installation

A renewal of most McStas Tools, used to edit and display instrument or results, has been undertaken, aiming at proposing alternatives to the Perl+PerlTk+PGPLOT+PDL libraries.

Quite a lot of work was achieved in order to solve the installation problems that have been encoutered so far. A fully working McStas distribution now only requires a C compiler, perl, perl-Tk and one of Matlab, Scilab and (PGPLOT, perl-DL). The Plotlib Scilab library has been included in the package, and does not need to be installed separately anymore.

This has improved significantly the portability of McStas and thus simplified the installation of the package. Details about the installation and the available tools are given in the *Installing McStas* chapter of the *User Manual*.

- The list of required packages for a complete McStas installation is now a C compiler, Perl, PerlTk and Scilab or Matlab.
- Matlab, Scilab and IDL may read directly McStas results if the simulation was executed with the `--format="..."` option (see 2.3 changes). The former PGPLOT interface is still supported (appeared in 1.7).
- `mcgui` can now perform `mcrun`-like parameter scans directly from the gui (appeared in 1.8).
- `mcgui` has now a 'Neutron site' menu which enables to load directly one of the instrument examples (appeared in 1.8).

- mcrun can generate scan results in all formats (appeared in 1.8).
- mcrun has a McStas self test procedure available (appeared in 1.8).
- mcplot, mcdisplay, mcgui are now less dependent on the perl/PDL/pgplot installed versions and fully work with Matlab/Scilab (appeared in 1.7).
- mcplot can plot a single simulation data file.
- mcplot, mcresplot, mcdisplay can output GIF, PS and color PS files. They also have integrated help (-h options), and may generate output files in a non interactive mode (read data, create output file, exit) (appeared in 1.7).
- mcplot and mcdisplay work with Matlab, PGPLOT and Scilab plotters (depends on the MCSTAS_FORMAT environment variable, or -pPLOTTER option, or PGPLOT if not set) (appeared in 1.7).
- mcplot may display parameter scan step results in all formats (appeared in 1.8).
- mcstas2vitess enables to convert a McStas component into a Vitess [10] one (appeared in 1.7).
- mcresplot can plot projections of the 4D resolution function of an instrument obtained from the Res_sample and Res_monitor components. In version 1.8, it only works with the McStas/PGPLOT format, but a port for Scilab/Matlab is under way (appeared in 1.7).
- mcdoc can now display the pdf version of the manual, an HTML catalog of the current library, as well as help for single components. The mcdoc functions have been closely integrated into mcgui (appeared in 1.7).
- mcdoc can document automatically instruments in the same way as components (appeared in 1.8).
- mcconvert is a new tool to convert McStas result text files from Matlab to Scilab, and from Scilab to Matlab formats (appeared in 1.8).

2.8 Future extensions

The following features are planned for the oncoming releases of McStas:

- Support for Matlab and Scilab in mcresplot.
- Support for MPI (parallel processing library) in the runtime
- Language extension 'JUMP' for enabeling loops, 'teleporting' etc. in instrument descriptions.
- Concentric components.
- Polarised components and magnetic field computation components.
- Optimize a set of parameters for a better flux and/or resolution on a given monitor.

Chapter 3

About the component library

The component library is maintained by the McStas system group. A number of basic components "belongs" the McStas system, while other are contributed by specific authors, mentioned along with each component.

Users are encouraged to send contributions to us for inclusion in future releases.

In the description of the theory behind the component functionality we will use the usual symbols **r** for the position (x, y, z) of the particle (unit m), and **v** for the particle velocity (v_x, v_y, v_z) (unit m/s). Another frequently used symbol is the wave vector $\mathbf{k} = m_{\mathrm{N}}\mathbf{v}/\hbar$, where m_{N} is the neutron mass. **k** is usually given in Å^{-1}, while neutron energies are given in meV. 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 a component. Note that all mentioning of component geometry refer to the local coordinate system of the individual component, in which the z axis is along propagation axis, the y axis is vertical up, and the x completes the direct coordinate system.

Source code for all component may be found in the `lib/mcstas/` 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 can be changed to something else using the `MCSTAS` environment variable.

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

```
mcdoc --show
```

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

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

The first line will search for all components matching the *name*, and display their help section as text, whereas the second example will display the help corresponding to the *file.comp* component, using your BROWSER setting, or as text if unset. The `--help` option will display the command help, as usual.

An overview of the component library is also given at the McStas home page...

Chapter 4

Source components

The main function of the source components is to determine a set of initial parameters $(\mathbf{r}, \mathbf{v})$, or equivalent $(\mathbf{r}, v, \mathbf{\Omega})$, for each neutron. This is done by Monte Carlo choices. In the current sources no polarization dependence is implemented, whence we let $\mathbf{s} = (0, 0)$.

The sources to be presented in the following all make their Monte Carlo choices for initial position and direction on the basis of a uniform distribution, with the exception of Souce_Div, where the distribution of initial divergence (deviation between the nominal and actual direction) is Gaussian. The shape of the source may be a disk (Moderator, Source_adapt, Source_flat and Source_flux), a rectangle (Source_div), a cube (ESS sources, Source_Maxwell_3)or even a box/cylinder (Source_gen). The choice of energy distribution varies from a flat distribution in Source_flat, to the more realistic weighted sum of three Maxwellians in Source_Maxwell_3 and Source_gen. As neutron energy is in principle unlimited, all sources require energy/wavelength windows to be specified.

For time-of-flight sources, the initial neutron time, t (in s), is an essential parameter. The choice of this is being made on basis of detailed analytical expressions, see the respective components. For other sources, the initial neutron time is set to zero.

The flux of the sources deserves special attention. The total flux is defined as the sum of weights of all emitted neutron rays during one simulation (the unit of weight is thus neutrons per second), if all neutron energies were allowed. In most sources, the total flux is a required parameter. For the simple sources Source_flat and Source_flat_lambda (mostly used for test purposes), the flux may be omitted. In this case the total integrated flux is set to unity.

4.1 Source_flat: A circular continuous source with a flat energy spectrum

Name:	Source_flat
Author:	System
Input parameters	r_s, z_f, w, h, E_0, ΔE
Optional parameters	None

This component is a simple continuous source with a energy distribution which is uniform in the range $E_0 - \Delta E$ to $E_0 + \Delta E$. The component is not used for 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 within a solid angle, defined by a rectangular target of width w, height h, parallel to the xy plane placed at $(0, 0, z_f)$.

The total weight of the neutron beam is set to the solid angle of the focusing opening divided by 4π.

4.2 Source_flux: A circular continuous source with a flat energy spectrum and absolute flux

4.3 Source_flat_lambda: A continuous source with flat wavelength spectrum

Name:	Source_flat_lambda
Author:	System
Input parameters	*radius, dist, xw, yh, lambda_0, d_lambda*
Optional parameters	None

The component **Source_flat_lambda** is similar to the Source_flat component, except that the spectrum is flat in wavelength, rather than in energy.

The input parameters for Source_flat_lambda are *radius* to set the source radius; *dist*, *xw*, and *yh* to set the focusing as for Source_flat; and *lambda_0* and *d_lambda* to set the range of wavelength emitted (the range will be from *lambda_0* − *d_lambda* to *lambda_0* + *d_lambda*).

IN GENERAL: HOW DO WE HANDLE MATH SYMBOLS IN THE COMPONENT DESCIPTIONS? THE FIRST TWO COMPONENT DESCRIPTIONS ARE WIDELY DIFFERENT!

4.4 Source_flux_lambda: A continuous source with a flat wavelength spectrum and absolute flux

The component **Source_flux_lambda** is a variation on the Source_flat_lambda component. The only difference is the possibility to specify the absolute flux of the source. The specified flux is used to adjust the initial neutron weight so that the intensity in the detectors is directly comparable to a measurement of one second on a real source with the same flux. This also makes the simulated detector intensities independent of the number of neutron histories simulated, easing the comparison between different simulation runs (though of course more neutron histories will give better statistics).

The flux Φ is the number of neutrons emitted per second from a one cm^2 area on the source surface, with direction within a a one steradian solid angle, and with wavelength within a one Ångstrøm interval. The total number of neutrons emitted towards a given diaframe in one second is therefore

$$N_{\mathrm{total}} = \Phi A \Omega \Delta\lambda$$

where A is the source area, Ω is the solid angle of the diaframe 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). If N_{sim} denotes the number of neutron histories to simulate, the initial neutron weight p_0 must be set to

$$p_0 = \frac{N_{\mathrm{total}}}{N_{\mathrm{sim}}} = \frac{\Phi}{N_{\mathrm{sim}}} A \Omega \Delta\lambda$$

The input parameters for Source_flux_lambda are *radius* to set the source radius in meters; *dist*, *xw*, and *yh* to set the focusing as for Source_flat; *lambda_0* and *d_lambda* to set the range of wavelength emitted (the range will be from *lambda_0* $-$ *d_lambda* to *lambda_0* $+$ *d_lambda*); and *flux* to set the source flux in units of $\mathrm{cm}^{-2}\mathrm{st}^{-1}\AA$.

4.5 Source_div: A divergent source

Source_div is a rectangular source which emits a beam of a certain divergence around the main exit direction (the direction of the z axis). The beam intensity and divergence are uniform over the whole of the source, and the energy distribution of the beam is uniform.

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

The input parameters for Source_div are the source dimensions w and h (in m), the divergencies δ_h and δ_v (FWHM in degrees), and the mean energy E_0 and the energy spread dE (both in meV). The neutron energy range is $(E_0 - dE; E_0 + dE)$.

4.6 Moderator: A time-of-flight source

The simple time-of-flight source component **Moderator** resembles the source component **Source_flat** described in **??**. Like **Source_flat**, **Moderator** is circular and focuses on a rectangular target. Further, 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,

$$\Psi(E,t) = \exp(-t/\tau(E)), \tag{4.1}$$

where the decay constant is given by

$$\tau(E) = \begin{cases} \tau_0 & ; E < E_c \\ \tau_0/[1 + (E - E_c)^2/\gamma^2] & ; E \geq E_c \end{cases} \tag{4.2}$$

The input parameters for **Moderator** are the source radius, r_{s}, the minimum and maximum energies, E_0 and E_1 (in meV), the distance to the target, z_{f}, the dimensions of the target, w and h, and the decay parameters τ_0 (in μs), E_c, and γ (both in meV).

4.7 Source_adapt: A neutron source with adaptive importance sampling

The **Source_adapt** component 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 achive 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.

The physical characteristics of the source are similar to those of Source_flat (see section ??). The source is a thin rectangle in the X-Y plane with a flat energy spectrum in a user-specified range. The flux per area per steradian per Ångstrøm per second is specified by the user; the total weight of neutrons emitted from the source will then be the same irrespectively of the number of neutron histories simulated, corresponding to one second of measurements.

The initial neutron weight is given by (see section 4.4 for details)

$$p_0 = \frac{N_{\mathrm{total}}}{N_{\mathrm{sim}}} = \frac{\Phi}{N_{\mathrm{sim}}} A\Omega\Delta\lambda$$

Here $\Delta\lambda$ is the total wavelength range of the source; since the spectrum is flat in energy (but not in wavelength), the flux will actually be different for different energies. A later version of this component will probably adapt (in a backward-compatible way) a more sensible way to specify the flux. For now, an energy or wavelength monitor (see sections 8.7 and 8.8) placed just after the source will show the actual energy-dependent flux.

4.7.1 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

$$N_{\mathrm{bin}} = N_{\mathrm{eng}} N_{\mathrm{pos}} N_{\mathrm{div}}$$

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

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

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

$$w_{\mathrm{min}} = \frac{\beta}{N_{\mathrm{bin}}} \sum_{j=1}^{N_{\mathrm{bin}}} w_j, \qquad 0 \le \beta \le 1$$

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 by the amount

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

In order to set the criteria for adaption, the Adapt_check component is used (see section 8.14). The source attemps to sample only from bins from which neutrons are not absorbed prior to the position in the instrument at which the Adapt_check component 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 $\pi = p_1/p_0$ 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 ψ 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 ψ_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 ψ_i/n_i at the Adapt_source position:

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

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

$$F(j) = C\frac{y_0}{y_0 + (1-y_0)e^{-r_0 j}} \tag{4.3}$$

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

$$\begin{aligned}
y_0 &= \frac{2}{N_{\text{bin}}} && (4.4)\\
r_0 &= \frac{1}{\alpha}\frac{1}{N_{\text{hist}}}\log\left(\frac{1-y_0}{y_0}\right) && (4.5)\\
C &= 1 + \log\left(y_0 + \frac{1-y_0}{N_{\text{hist}}}e^{-r_0 N_{\text{hist}}}\right) && (4.6)
\end{aligned}$$

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:

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

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

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

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

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

4.7.2 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 $v_i^{\rm new} = v_i^{\rm old} + a$. A *search* finds, for given input b, the minimum i such that

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

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

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

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; how this is achieved will be described elsewhere.

The input parameters for Source_adapt are *xmin*, *xmax*, *ymin*, and *ymax* in meters to set the source dimensions; *dist*, *xw*, and *yh* to set the focusing as for Source_flat (section **??**); *E0* and *dE* to set the range of energies emitted, in meV (the range will be from $E0 - dE$ to $E0 + dE$); flux to set the source flux Φ in $\text{cm}^{-2}\text{st}^{-1}\text{Å}\text{s}^{-1}$; $N_{\rm eng}$, $N_{\rm pos}$, and $N_{\rm div}$ to set the number of bins in each dimensions; *alpha* and *beta* to set the parameters α and β as described above; and *filename* to give the name of a file in which to output the final sampling destribution.

A good general-purpose value for α and β is $\alpha = \beta = 0.25$. 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.

4.8 Source_Optimizer: A general Optimizer for McStas

This component was contributed by Emmanuel Farhi, Institute Laue-Langevin.

The component **Source_Optimizer** optimizes the whole neutron flux in order to achieve better statistics at each **Monitor_Optimizer** location(s) (see section 8.15 for this latter component). It can act 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 *via* various *options*.

The optimizer efficiency makes it easy to increase the number of events at optimization criteria locations by a factor of 20, and thus decreases the signal error bars by a factor 4.5. Higher factors can often be achieved in practise. Of course, the overall flux remains the same as without optimizer.

4.8.1 The optimization algorithm

When a neutron reaches the **Monitor_Optimizer** location(s), the component records its 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. Anyway, 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* source to the **Source_Optimizer**. However, '*keep*' percent of the original *Reference* source is sent unmodified (default is 10 %). The *Optimized* source is thus:

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

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:

$$p_{optimized} = p_{initial} \frac{\text{Reference}}{\text{Optimized}}, \tag{4.7}$$

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 than 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 $|\vec{v}|$ 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. Howver, 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.

4.8.2 Using the Source_Optimizer

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

```
/* where to act on neutrons */
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* = 10, *bins* = 10, *smooth* spikes, and do *not free* energy during optimization".

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.

Chapter 5

Simple optical components: Arms, slits, collimators, filters

Below we list a number of simple optical components which require only a minimum of explanation.

5.1 Arm: The generic component

The component **Arm** is empty; is resembles an optical bench and has no effect on the neutron. The function of this component is only to set up a local frame of reference within the instrument definition. Other components of the same arm/optical bench 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. For more about the use of this component, see the sample instrument definitions listed in Appendix **??**.

5.2 Slit: The rectangular slit

The component **Slit** is a very simple construction. It sets up a rectangular opening at $z = 0$, and propagates the neutrons onto the plane of this rectangle by the kernel call PROP_Z0.

Neutrons within the slit opening are unaffected, while all other neutrons (no matter how far from the opening their paths intersect the plane) are discarded by the kernel call ABSORB. By this simplification, 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 that clear one of 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.

5.3 Circular_slit: The circular slit

The component **Circular_slit** defines a circle in the $z = 0$ plane, centered in the origin. In analogy with **Slit**, neutrons are propagated to this plane, and those which intersect the plane outside the circle are ABSORB'ed.

The only input parameter of **Circular_slit** is the radius, r, of the circle.

5.4 Beamstop_rectangular: The rectangular beam stop

The component **Beamstop_rectangular** models a thin, infinitely absorbing rectangle in the X-Y plane, centered on the origin. The input parameters are *xmin*, *xmax*, *ymin*, and *ymax* defining the edges of the slit in meters.

5.5 Beamstop_circular: The circular beam stop

The component **Beamstop_circular** models a thin, infinitely absorbing circular disk in the X-Y plane, centered on the origin. It takes a single input parameter *radius* to define the circle radius in meters.

5.6 Soller: The simple Soller blade collimator

The component **Soller** defines two rectangular openings like the one in **Slit**. Neutrons not clearing both these openings are ABSORB'ed, see the discussion in 5.2. The collimating effect is taken care of by employing an ideal triangular transmission through the collimator, as explained below. For a more detailed Soller collimator simulation the Channeled_guide component can be employed, see section 6.4.

Let the collimation angle be $\delta = \tan^{-1}(d/L)$, where L is the length of the collimator and d is the distance between the blades, and let ϕ be the divergence angle between the neutron path and a vertical plane along the collimator axis, see Fig. 5.1. Neutrons with a large divergence angle $|\phi| \geq \delta$ will always hit at least one collimator blade and will thus be absorbed. For smaller divergence angles, $|\phi| < \delta$, 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 somewhat uncertain (and also unimportant). A simple statistical consideration now shows that the transmission probability is $T = 1 - \tan|\phi| / \tan\delta$.

We simulate the collimator by transmitting all neutrons with $|\phi| < \delta$, but adjusting their weight with the amount

$$\pi_i = T = 1 - \tan|\phi| / \tan\delta, \tag{5.1}$$

while all others are discarded by the kernel call ABSORB.

The input parameters for **Soller** are the coordinates $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$, defining the identical entry and exit apertures, the length, l, between the slits, and the collimator divergence δ. If $\delta = 0$, the collimating effect is disabled, so that $\pi_i = 1$ whenever the neutron clears the two apertures.

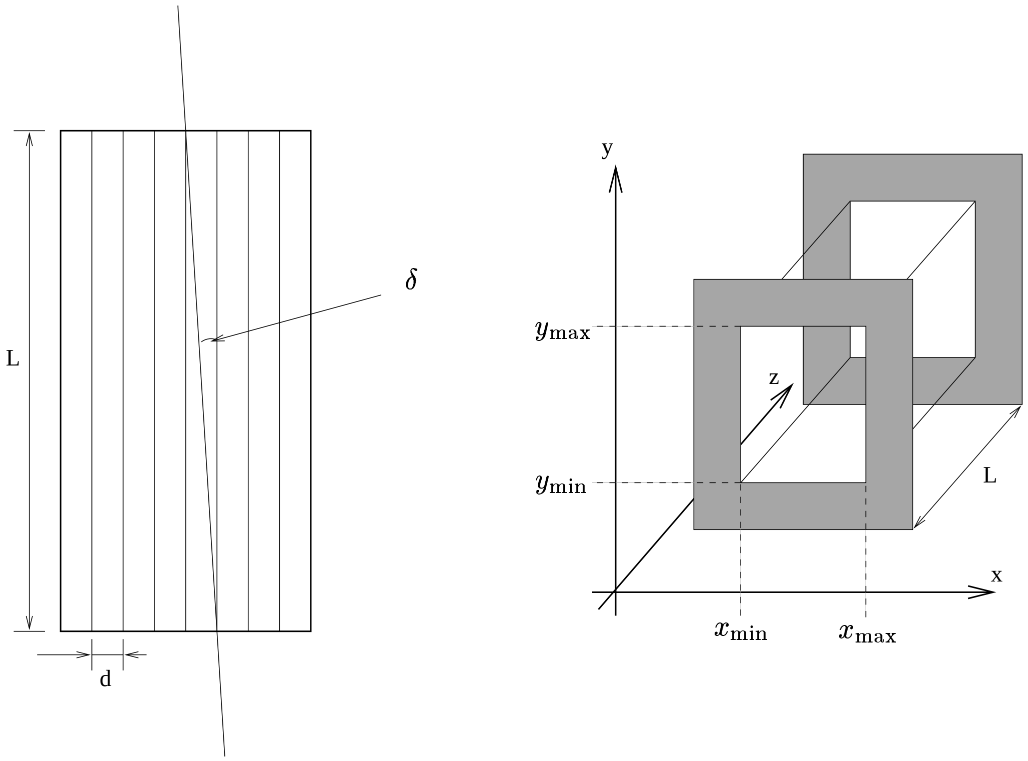

Figure 5.1: The geometry of a simple Soller blade collimators: The real Soller collimator, seen from the top (left), and a sketch of the component **Soller** (right). The symbols are defined in the text.

5.7 Filter: A transmission filter

A neutron transmission filter act in much of the same way as two identical slits, one after the other. The only difference is that the transmission of the filter varies with the neutron energy.

In the simple component **Filter**, we have not tried to simulate the details of the transmission process (which includes absorption, incoherent scattering, and Bragg scattering in a polycrystalline sample, *e.g.* Be). Instead, the transmission is parametrised to be $\pi_i = T_0$ when $E \leq E_{\rm min}$, $\pi_i = T_1$ when $E \geq E_{\rm max}$, and linearly interpolated between the two values in the intermediate interval.

$$\pi_i = \begin{cases} T_0 & E \leq E_{\rm min} \\ T_1 + (T_0 - T_1)\frac{E_{\rm max}-E}{E_{\rm max}-E_{\rm min}} & E_{\rm min} < E < E_{\rm max} \\ T_1 & E \geq E_{\rm max} \end{cases} \tag{5.2}$$

If $T_1 = 0$, the neutrons with $E > E_{\rm max}$ are ABSORB'ed.

The input parameters are the four slit coordinates, $(x_{\rm min}, x_{\rm max}, y_{\rm min}, y_{\rm max})$, the distance, l, between the slits, and the transmission parameters T_0, T_1, $E_{\rm min}$, and $E_{\rm max}$. The energies are given in meV.

Chapter 6

Advanced optical components: mirrors and guides

This section describes advanced neutron optical components such as supermirrors and guides. The first subsection, however, contains only a description of the reflectivity of a supermirror.

6.1 Mirror reflectivity

To compute the reflectivity of the supermirrors, we use an empirical formula derived from experimental data (see figure 6.1). The reflectivity is given by the following formula

$$R = \begin{cases} R_0 & \text{if } Q \leq Q_c \\ \frac{1}{2}R_0(1-\tanh[(Q-mQ_c)/W])(1-\alpha(Q-Q_c)) & \text{if } Q > Q_c \end{cases} \tag{6.1}$$

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

$$Q = |\mathbf{k_i} - \mathbf{k_f}| = \frac{m_n}{\hbar}|\mathbf{v_i} - \mathbf{v_f}|, \tag{6.2}$$

m_n being the neutron mass. The value m is a parameter determined by the mirror materials, the bilayer sequence, and the number of bilayers. As can be seen, $R = R_0$ for $Q < Q_c$, which 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 cut-off at $Q = mQ_c$. The width of the cut-off is denoted W. For the curve in figure 6.1, the values are

$$m = 4 \qquad R_0 = 1 \qquad Q_c = 0.02\ \text{Å}^{-1} \qquad \alpha = 6.49\ \text{Å} \qquad W = 1/300\ \text{Å}^{-1}$$

As a special case, if $m = 0$ then the reflectivity is zero for all Q, *ie.* the surface is completely absorbing.

In the components, the neutron weight is adjusted with the amount $\pi_i = R$. 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.

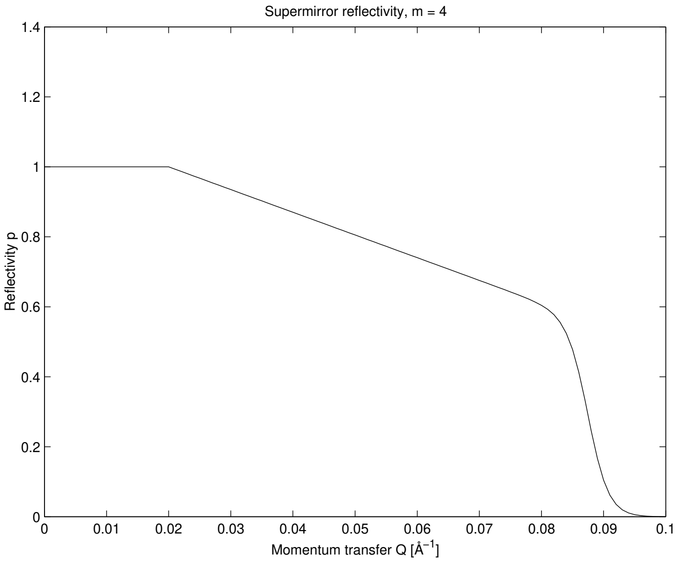

Figure 6.1: A typical reflectivity curve for a supermirror, Eq. (6.2).

6.2 Mirror: The single mirror

The component **Mirror** models a single rectangular neutron mirror plate. It can be used 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.

The mirror is assumed to lie in the first quadrant of the x-y plane, with one corner at $(0, 0, 0)$. If the neutron trajectory intersects the mirror plate, it is reflected, otherwise it is left untouched. Since the mirror lies in the x-y plane, an incoming neutron with velocity $\mathbf{v}_\mathrm{i} = (v_x, v_y, v_z)$ is reflected with velocity $\mathbf{v}_\mathrm{f} = (v_x, v_y, -v_z)$. The computation of the reflectivity is handled as detailed in section 6.1.

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.

6.3 Guide: The guide section

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 6.2. Neutrons not clearing the guide entrance are ABSORB'ed. For a more general guide simulation, see the Channeled_guide component in section 6.4.

For computations on the guide geometry, we define the planes of the four guide sides

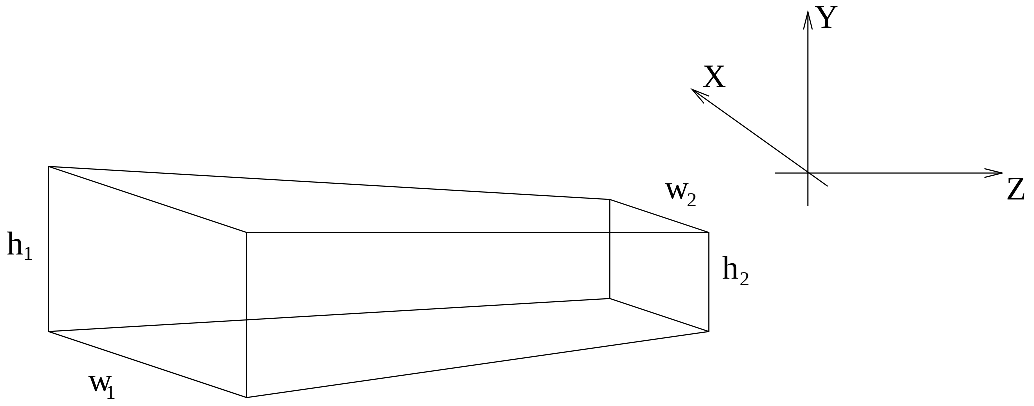

Figure 6.2: The geometry used for the guide component.

by giving their normal vectors (pointing into the guide) and a point lying in the plane:

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

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

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

$$t_1 = \frac{(\mathbf{O} - \mathbf{r}_0) \cdot \mathbf{n}}{\mathbf{v} \cdot \mathbf{n}} \tag{6.3}$$

For a neutron that leaves the guide through the guide exit we have

$$t_2 = \frac{l - z_0}{v_z} \tag{6.4}$$

To compute the interaction of the neutron with the guide, the neutron is initially propagated to the $z = 0$ plane of the guide entrance. If it misses the entrance, it is ABSORB'ed. Otherwise, we repeatedly compute the time of intersection with the four mirror sides and the guide exit. 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). The neutron is propagated to this point, the reflection from the side is computed and the process is repeated until the neutron leaves the guide.

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

$$\mathbf{v}_\mathrm{f} = \mathbf{v}_\mathrm{i} - 2\frac{\mathbf{n} \cdot \mathbf{v}_\mathrm{i}}{|\mathbf{n}|^2}\mathbf{n} \tag{6.5}$$

This expression is arrived at by again considering the projection onto the mirror normal (see figure 6.3). The reflectivity of the mirror is taken into account as explained in section 6.1.

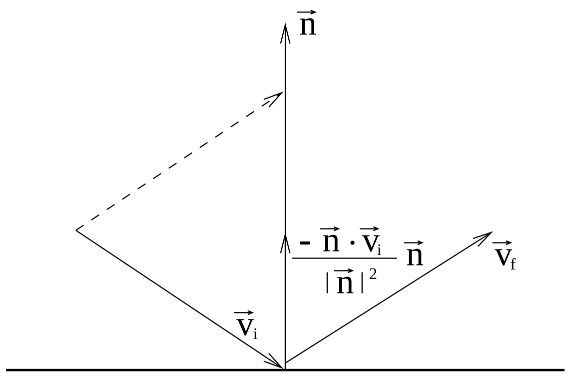

Figure 6.3: Neutron reflecting from mirror. $\mathbf{v}_\mathrm{i}$ and $\mathbf{v}_\mathrm{f}$ are the initial and final velocities, respectively, and $\mathbf{n}$ is a vector normal to the mirror surface.

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.

The input parameters of this component are the opening sizes of the entry and exit point of the guide, (w_1, h_1) and (w_2, h_2), respectively, the guide length, l, and the values of R_0, m, Q_c, W, and α for the mirror.

6.4 Channeled_guide: A guide section component with multiple channels

The component Channeled_guide is a more flexible variation of the Guide component 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 to simulate a Soller collimator.

The channel walls are assumed to be infinitely absorbing. The implementation is basen on that of the Guide component. Initially, the channel which the neutron will enter is computed. The x coordinate is then shifted so that the channel can be simulated as a single instance of the Guide component. Finally the coordinates are restored when the neutron exits the guide or is absorbed.

The input parameters are *w1*, *h1*, *w2*, *h2*, and l to set the guide dimensions in meters as for the Guide component (entry window, exit window, and length); k to set the number of channels; d to set the thickness of the channel walls, in meters; and *R0*, *W*, *Qcx*, *Qcy*, *alphax*, *alphay*, *mx*, and *my* to set the supermirror parameters as described above (the names with x denote the vertical mirrors, and those with y denote the horizontal ones).

Chapter 7

Chopper-like components

In this section, we will present rotating components such as chopppers, velocity selectors, *etc.*

7.1 V_selector: The rotating velocity selector

The component **V_selector** models a rotating velocity selector constructed from a number of collimator blades arranged radially on an axis. Two identical slits 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 any neutrons; the total twist angle is denoted ϕ. By rotating the selector you allow transmittance of neutrons around certain velocities, given by

$$V_0 = \omega L/\phi, \tag{7.1}$$

which means that the selector has turned the twist angle ϕ during the neutron flight time L/V_0.

Neutrons having a velocity slightly smaller or larger than 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 assuming infinitely thin blades), we arrive at

$$T = \begin{cases} 1 - (N/2\pi)|\phi - \omega L/V| & \text{if } -1 < (N/2\pi)(\phi - \omega L/V) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

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 amount

$$dT = (Nt)/(2\pi r), \tag{7.3}$$

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.

The input parameters for V_selector are the slit dimensions, *width*, *height* (in m), the distance between apertures, L_0 (in m), the length of the collimator blades, L_1 (in m), the height from rotation axix to the slit centre, r_0 (in m), the rotation speed ω (in rpm) the twist angle ϕ (in degrees), the blade thickness t (in m), and the number of blades, N.

The local coordinate system is centered at the slit centre.

7.2 Chopper: The disc chopper

This component was contributed by Philipp Bernhardt, Lehrstuhl für Kristallographie und Strukturphysik.

To cut a continuous neutron beam into short pulses, you can use a disc chopper (figure 7.1). 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 there are slits through which the neutrons can pass.

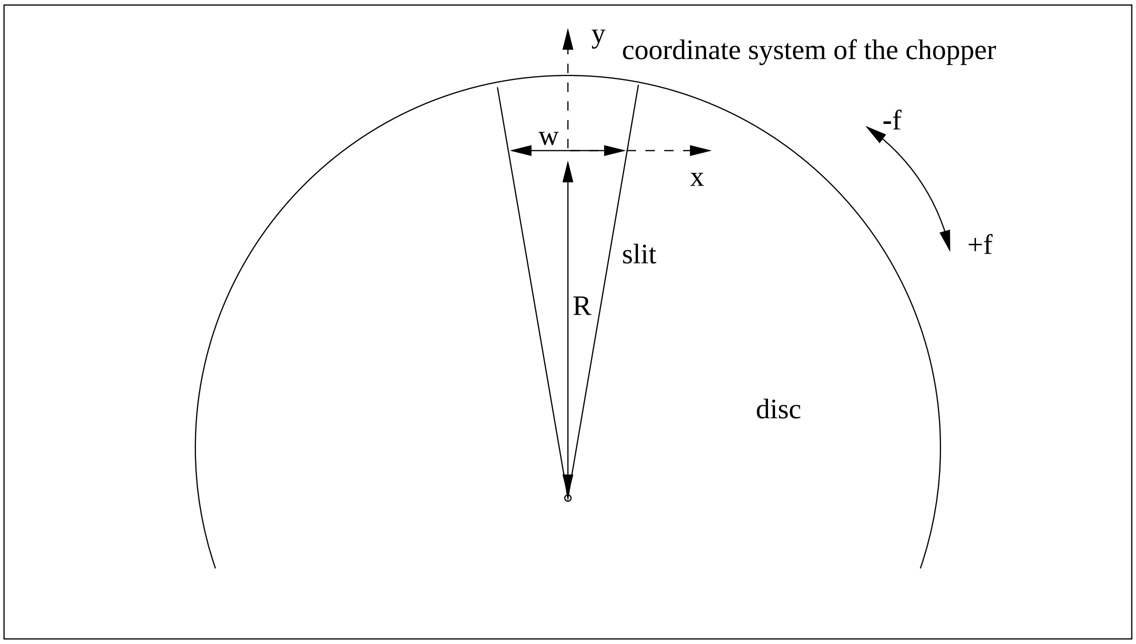

Figure 7.1: disc chopper

This component simulates choppers with more than one slit. The slits are symmetrically disposed on the disc. You can set the direction of rotation, which allows to simulate double choppers. You can also define the phase by setting the time at which one slit is positioned at the top. The sides of the slits are pointing towards the center of the disc. The thickness of the disc is neglected. There is no parameter for the height of the slits, so if you like to limit the neutrons in the y-direction, just use a slit component in front of the chopper.

If you use a rectangular shaped beam and the beam has nearly the same size as the slit, you will get an almost triangular shape of the transmission curve (figure 7.2).

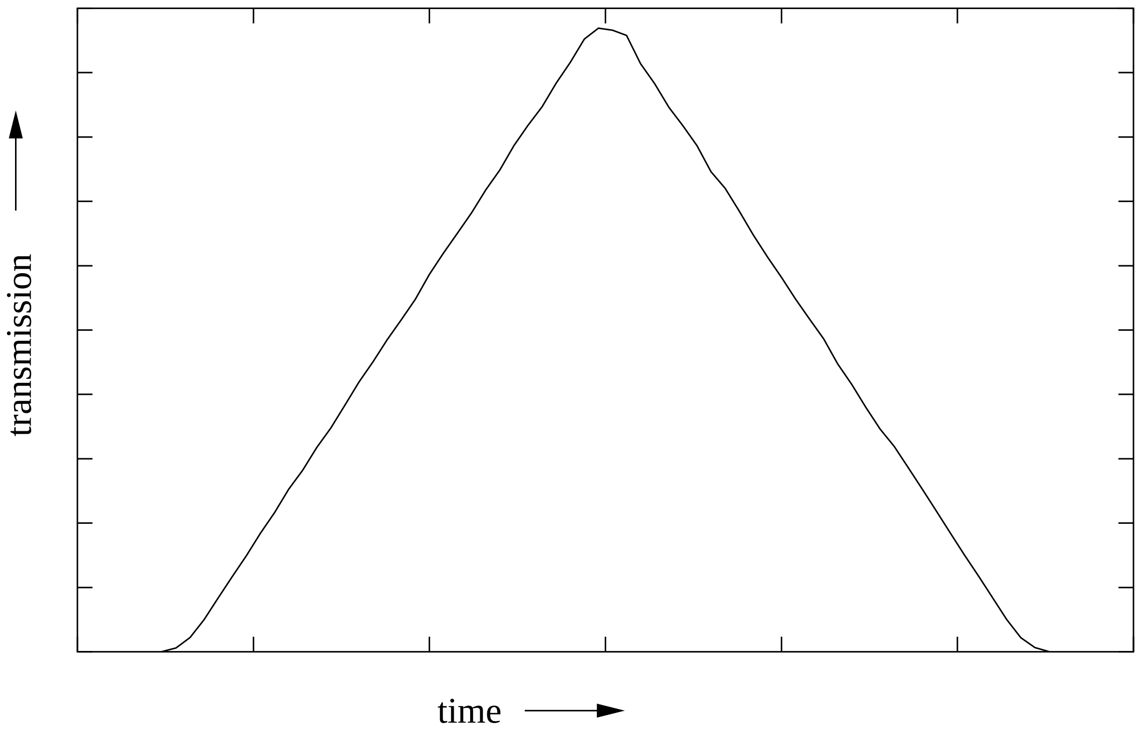

Figure 7.2: example transmission curve for the disc chopper

The input parameters for this component are the width w of the slit at the radius R of the disc, the phase pha, the number of slits n and the angular frequency f. The sign of f defines the direction of rotation, as can be seen in figure 7.1.

7.3 First_chopper: The first disc chopper

This component was contributed by Philipp Bernhardt, Lehrstuhl für Kristallographie und Strukturphysik.

The disadvantage of the component 'Chopper' is the bad statistic, because most of the neutrons of a continuous beam are absorbed. Furthermore TOF-instruments define the starting time of the neutrons at the position of the first chopper and not at the source. Therefore this component is useful. This 'first disc chopper' has the same geometrical and physical attributes as the normal disc chopper before. But it does not check if the neutron can pass the disc chopper, it instead gives the neutron a time at which it is possible to pass. There is no absorption in this component, all neutrons will be used.

Because the value t of the incoming neutron will be overwritten, this chopper can only be used as a first chopper.

The input parameters are, again, the width w of the slit at the radius R of the disc, the phase pha of the chopper, the number of slits n and the angular frequency f (sign defines direction of rotation). With the additional parameter a you can set the number of pulses. This is useful if you want to investigate frame overlaps.

Chapter 8

Detectors and monitors

In real neutron experiments, detectors and monitors play quite different roles. One wants the detectors to be as efficient as possible, counting all neutrons (and 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 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, since they do not absorb the neutron.

Another 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 polarization, in addition to what is found in advanced existing monitors (space and time). One may, in fact, let the monitor have several of these properties at the same time, as seen for example in the energy sensitive monitor in section 8.7.

8.1 Monitor: The single monitor

The component **Monitor** consists of a rectangular opening — like that for **slit**. The neutron is propagated to the plane of the monitor by the kernel call PROP_Z0. Any neutron that passes within the opening is counted — the number counting variable is incremented: $N_i = N_{i-1}+1$, 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: $M_{2,i} = M_{2,i-1} + p_i^2$. The input parameters for **Monitor** are the opening coordinates $x_{\rm min}, x_{\rm max}, y_{\rm min}, y_{\rm max}$, and the output parameters are the three count numbers, N, I, and M_2.

8.2 Monitor_4PI: The 4π monitor

The component **Monitor_4PI** does not model any physical monitor but may be thought of as a spherical monitor completely surrounding the previous component. It simply detects all neutrons that have not been absorbed at the position in the instrument in which it is placed. If this monitor is placed in the instrument file after another component, *e.g.*

a sample, it will count any neutron scattered from this component. This may be useful during tests.

The output parameters for **Monitor_4PI** are the three count numbers, N, I, and M_2.

8.3 PSD_monitor: The PSD monitor

The component **PSD_monitor** closely resembles **Monitor**. In the PSD monitor, though, the rectangular monitor window is divided into $n \times m$ pixels, each of which acts like a single monitor.

The input parameters for **PSDmonitor** are the opening coordinates $x_{\text{min}}, x_{\text{max}}, y_{\text{min}}$, y_{max}, the array dimensions (n, m), and a name of a file in which to store $I(x, y)$. The output parameters are three two-dimensional arrays of counts: $N(x, y), I(x, y), M_2(x, y)$.

8.4 PSD_monitor_4PI: The 4π PSD monitor

The component **PSD_monitor_4PI** represents a PSD monitor shaped as a sphere, much like **Monitor_4PI**. It subdivides the surface of the sphere into pixels of equal area (using a projection onto a cylinder with an axis which is vertical in the local coordinate system) and distributes the incoming neutron counts into the respective pixels.

The 4π PSD monitor is typically placed around another component. Used in this way, the 4π PSD monitor is very useful for debugging components.

The input parameters for **PSD_monitor_4PI** are the monitor radius, the number of pixels, (n_x, n_y) – where y is the vertical direction, and the name of the file in which to store $I(x, y)$. The output parameters of the component are the three count arrays $N(x, y), I(x, y)$, and $M_2(x, y)$.

8.5 PSD_monitor_4PI_log: The 4π PSD monitor with log scale

The component **PSD_monitor_4PI_log** is the same as PSD_monitor_4PI described in the previous section, except that the output histograms contain the base-10 logarithm of the intensities rather than the intensities themselves. Currently, this does not work well together with the McStas mechanism to output detector results (see section **??**), so the total intensity as output by McStas from this detector component will be wrong. However, the component was sufficiently useful in Laue-type diffraction instruments to be included here nevertheless. A future version of McStas may implement a better way to get log-scale in output files.

The input parameters for PSD_monitor_4PI_log are the same as for PSD_monitor_4PI.

8.6 TOF_monitor: The time-of-flight monitor

TOF_monitor is a rectangular single monitor which is sensitive to the absolute time, where the neutron is hits the component. Like in a real time-of-flight detector, the time dimension is binned into small time intervals of length dt, whence this monitor updates a one-dimensional array of counts.

The input parameters for **TOF_monitor** are the opening coordinates $x_{\min}, x_{\max}, y_{\min}$, $y_{\max}$, the number of time bins (beginning from $t = 0$), n_{chan}, the time spacing between bins, dt (in μs), and the name of the output file. Output parameters of the component are the three count arrays $N(i), I(i)$, and $M_2(i)$, where i is the bin number.

8.7 E_monitor: The energy sensitive monitor

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

The input parameters for **E_monitor** are the opening coordinates $x_{\min}, x_{\max}, y_{\min}$, $y_{\max}$, the total energy interval given by $E_{\min}$ and $E_{\max}$ (in meV), and *nchan* and the name of the output file. Output parameters of the component are the three count arrays $N(i), I(i)$, and $M_2(i)$, i being the bin number.

8.8 L_monitor: The wavelength sensitive monitor

The component **L_monitor** is a rectangular monitor with an opening in the x-y plane which is sensitive to the neutron wavelength. The wavelength spectrum is output in a one-dimensional histogram. Only neutrons with wavelength $\lambda_0 < \lambda < \lambda_1$ are detected.

The input parameters for **L_monitor** are the opening coordinates *xmin*, *xmax*, *ymin*, and *ymax* defining the edges of the slit in meters; the lower and upper wavelength limit *Lmin* and *Lmax* in Ångstrøm; the number of histogram bins *nchan*; and *filename*, a string giving the name of the file to store the data in.

8.9 Divergence_monitor: The divergence sensitive monitor

The component **Divergence_monitor** is a rectangular monitor with an opening in the x-y plane, which is sensitive to the neutron divergence, *i.e.* the angle between the neutron path and the monitor surface normal.

The divergence is divided into horisontal and vertical divergencies, which are calculated as $\delta_h = \tan^{-1}(v_x/v_z)$ and $\delta_v = \tan^{-1}(v_y/v_z)$, respectively. Only neutrons within a divergence window of $\delta_h = (-\delta_{\text{h,max}}; \delta_{\text{h,max}})$, $\delta_v = (-\delta_{\text{v,max}}; \delta_{\text{v,max}})$ are detected. The counts are binned in an array of $n_h \times n_v$ pixels.

The input parameters for the Divergence_monitor component are the opening coordinates $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$, the number of pixels (n_h, n_v), the parameters $(\delta_{\text{h,max}}, \delta_{\text{v,max}})$ defining the divergence interval, and a name of the file in which to store the detected intensities.

Note that a divergence sensitive monitor with a small opening may be thought of as a non-reversing pinhole camera.

8.10 DivPos_monitor: The divergence-position sensitive monitor

The component **DivPos_monitor** is a rectangular monitor with an opening in the x-y plane, which is sensitive to both the horizontal neutron divergence and the horizontal neutron position. The neutron intensity as a function of position and divergence is output in a two-dimensional histogram. This output may be directly compared to an acceptance diagram, an analytical technique that is sometimes used to calculate neutron guide performances.

The horizontal divergence is calculated as $\delta_h = \tan^{-1}(v_x/v_z)$. Only neutrons within a divergence window of $\delta_h = (-\delta_{\rm h,max};\delta_{\rm h,max})$ are detected.

The input parameters for the DivPos_monitor component are the opening coordinates *xmin*, *xmax*, *ymin*, and *ymax* in meters; the number of histogram bins *npos* and *ndiv* in position and divergence; the maximum divergence *maxdiv* to detect, in degrees; and *filename*, a string giving the name of the file to store the data in.

8.11 DivLambda_monitor: The divergence-wavelength sensitive monitor

The component **DivLambda_monitor** is a rectangular monitor with an opening in the x-y plane, which is sensitive to both the horizontal neutron divergence and the wavelength. The neutron intensity as a function of wavelength and divergence is output in a two-dimensional histogram.

The horizontal divergence is calculated as $\delta_h = \tan^{-1}(v_x/v_z)$. Only neutrons within a divergence window of $\delta_h = (-\delta_{\rm h,max};\delta_{\rm h,max})$ and with wavelength $\lambda_0 < \lambda < \lambda_1$ are detected.

The input parameters for the DivLambda_monitor component are the opening coordinates *xmin*, *xmax*, *ymin*, and *ymax* in meters; the number of histogram bins *nlam* and *ndiv* in wavelength and divergence; the maximum divergence *maxdiv* to detect, in degrees; *lambda_0* and *lambda_1* to define the wavelength window, in Ångstrøm; and *filename*, a string giving the name of the file to store the data in.

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

This component was contributed by Emmanuel Farhi, Institute Laue-Langevin.

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

The input parameters for **Monitor_nD** are its dimensions $x_{\rm min}, x_{\rm max}, y_{\rm min}$, $y_{\rm max}$ (in meters) and an *options* string describing what to detect, and what to do with the signals, 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 *no* or *not* option modifier will revert next option. The *all* option can also affect a set of monitor configuration parameters (see below).

The Monitor_nD geometry

The monitor shape can be selected among four geometries:

1. (*square*) The default geometry is flat rectangular in (xy) plane with dimensions $x_{\min}, x_{\max}, y_{\min}, y_{\max}$.
2. (*disk*) When choosing this geometry, the detector is a flat disk in (xy) plane. The radius is then
$$\text{radius} = \max(\text{abs } [x_{\min}, x_{\max}, y_{\min}, y_{\max}]). \tag{8.1}$$
3. (*sphere*) The detector is a sphere with the same radius as for the *disk* geometry.
4. (*cylinder*) The detector is a cylinder with revolution axis along y (vertical). The radius in (xz) plane is
$$\text{radius} = \max(\text{abs } [x_{\min}, x_{\max}]), \tag{8.2}$$
and the height along y is
$$\text{height} = |y_{\max} - y_{\max}|. \tag{8.3}$$

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

At last, the *slit* or *absorb* option will ask the component to absorb the neutrons that do not intersect the monitor.

The neutron parameters that can be monitored

There are 25 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) Wavevector norm or coordinates
vx vy vz v            (m/s)    Velocity norm or coordinates
x y z radius          (m)      Position and radius in (xy) plane
t time                (s)      Time of Flight
energy omega          (meV)    Neutron energy
lambda wavelength     (Angs)   Neutron wavelength
p intensity flux      (n/s) or (n/cm^2/s) The neutron weight
ncounts               (1)      The number of events detected
sx sy sz              (1)      Spin of the neutron
vdiv                  (deg)    vertical divergence
hdiv divergence       (deg)    horizontal divergence
angle                 (deg)    divergence from <z> direction
theta longitude       (deg)    longitude (x/z)
phi   lattitude       (deg)    lattitude (y/z)
```

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.

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

Each monitoring will record the flux (sum of weights p) versus the given variables. The *per cm2* option will ask to normalize the flux to the monitor section surface.

Some examples ?

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

2. `options="x y, all bins=30, all limits=[-0.05 0.05]"`
   will do the same, but set limits and bins for x and y.

3. `options="x y, auto limits"`
   will determine itself the required limits for x and y to monitor passing neutrons with default *bins*=20.

The output files

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

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

The 2D output

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

The 1D output

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

The List output

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

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*=`"x y, all bins=50"`
  will monitor the neutron x and y in a single 2D file (same as PSD_monitor)

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

- *options*=`"x y z kx ky kz, auto limits"`
  will monitor theses variables in six 1D files

- *options*=`"x y z kx ky kz, list all, auto limits"`
  will monitor all theses neutron variables in one long list

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

8.13 Res_monitor: The resolution monitor

The component **Res_monitor** is used together with the **Res_sample** component (described in section 11.1) and the `mcresplot` front-end (described in section **??**). It works like a normal single detector, but also records all scattering events in the resolution sample and writes them to a file that can later be read by `mcresplot`.

The instrument definition should contain an instance of the **Res_sample** component, the name of which should be passed as an input parameter to **Res_monitor**. For example

```
COMPONENT mysample = Res_sample( ... )
...
COMPONENT det = Res_monitor(res_sample_comp = mysample, ...)
...
```

The output file is in ASCII format, one line per scattering event, with the following columns:

- $\mathbf{k}_\mathrm{i}$, the three components of the initial wave vector.
- $\mathbf{k}_\mathrm{f}$, the three components of the final wave vector.
- $\mathbf{r}$, the three components of the position of the scattering event in the sample.
- p_i, the neutron weight just after the scattering event.
- p_f, the relative neutron weight adjustment from sample to detector (so the total weight in the detector is $p_\mathrm{i}p_\mathrm{f}$).

From $\mathbf{k}_\mathrm{i}$ and $\mathbf{k}_\mathrm{f}$, we may compute $\mathbf{Q} = \mathbf{k}_\mathrm{i} - \mathbf{k}_\mathrm{f}$ and $\omega = (2.072\ \mathrm{meV}\cdot\mathrm{Å}^2)(\mathbf{k}_\mathrm{i}^2 - \mathbf{k}_\mathrm{f}^2)$.

The vectors are given in the local coordinate system of the resolution sample component. The wave vectors are in units of $\mathrm{Å}^{-1}$, the scattering position in units of meters.

The input parameters for **Res_monitor** are the opening coordinates $x_\mathrm{min}, x_\mathrm{max}, y_\mathrm{min}$, y_max as for the single monitor component, the name of the file to write in *filename*, and *res_sample_comp* which should be set to the name of the resolution sample component used in the instrument. The output parameters are the three count numbers, *Nsum*, *psum*, and *p2sum*, and the handle *file* of the output file.

8.14 Adapt_check: The simple adaptive importance sampling monitor

The component **Adapt_check** is used together with the Source_adapt component — see section 4.7 for details. When placed somewhere in an instrument using Source_adapt, the source will optimize for neutrons that reach that point without being absorbed (regardless of neutron position, direction, wavelength, *etc*).

The Adapt_check component takes a single input parameter *source_comp*. This should be set to the name given to the Source_adapt component in the instrument, for example

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

8.15 Monitor_Optimizer: Optimization locations for the Source_Optimizer component

This component was contributed by Emmanuel Farhi, Institute Laue-Langevin.

The **Monitor_Optimizer** component works with the **Source_Optimizer** component. See section 4.8 for usage.

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

Chapter 9

Bragg scattering single crystals, monochromators

In this class of components, we are concerned with elastic Bragg scattering from single crystals. The Mosaic_anisotropic component models a thin mosaic crystal with a single scattering vector perpendicular to the surface. It is a replacement for the Monochromator component from previous releases; it uses a better algorithm that works in some cases where the old component would give wrong results. The Mosaic_simple component is similar, but has an isotropic mosaic and allows a scattering vector that is not perpendicular to the surface. The Single_crystal component is a general single crystal sample that allows the input of an arbitrary unit cell and a list of structure factors, and also allows anisotropic mosaic and $\Delta d/d$ lattice space variation.

9.0.1 Mosaic_simple: An infinitely thin mosaic crystal with a single scattering vector

The component **Mosaic_simple** simulates an infinitely thin single crystal with a single scattering vector and a mosaic spread. A typical use for this component is to simulate a monochromator or an analyzer.

The physical model used in the component 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 of the crystal. The mosaic spread is assumed to be large compared to the Bragg width of the scattering vector.

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 can be used as a parameter for the model rather than the atomic scattering cross section. 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 9.0.3 for that).

When a neutron trajectory intersects the crystal, the first step in the computation is to determine the probability of scattering. This probability is then used in a Monte Carlo choice deciding whether to scatter or transmit the neutron. The scattering probability is

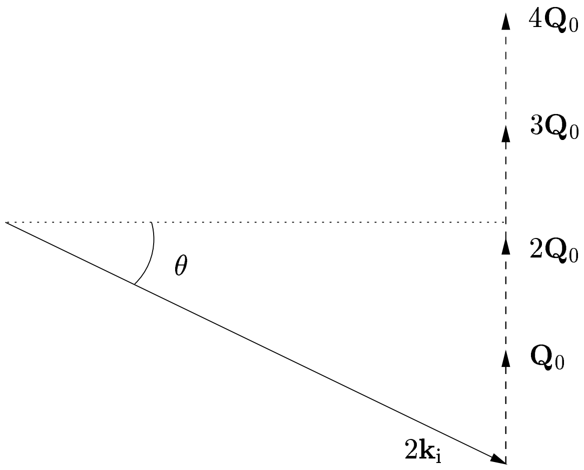

Figure 9.1: Selection of the Bragg order ("2" in this case).

the sum of the probabilities of first-order scattering, second-order, ..., up to the highest order that permits Bragg scattering at the given neutron wave length. 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. Bragg's law is

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

Thus, the scattering order is obtained simply as the integer multiple n of the nominal scattering vector $\mathbf{Q}_0$ which is closest to the projection of $2\mathbf{k}_i$ onto $\mathbf{Q}_0$ (see figure 9.1). Once n has been determined, the Bragg angle θ can be computed. The angle d that the nominal scattering vector $\mathbf{Q}_0$ makes with the closest scattering vector $\mathbf{q}$ that admits Bragg scattering is then used to compute the probability of reflection from the mosaic

$$p_\mathrm{reflect} = R_0 e^{-d^2/2\sigma^2},$$

where R_0 is the reflectivity at the Bragg angle (see figure 9.2). The probability p_reflect is used in a Monte Carlo choice to decide whether the neutron is transmitted or reflected.

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

$$\mathbf{k}_\mathrm{f} = \mathbf{k}_\mathrm{i}\cos 2\theta + \sin 2\theta(\mathbf{c}\cos\varphi + \mathbf{b}\sin\varphi), \qquad \varphi \in [-\pi;\pi], \tag{9.1}$$

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 9.3). 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 σ of $\mathbf{q}$ from the nominal scattering vector $n\mathbf{Q}_0$ corresponds to a Δq of

$$\Delta q \approx \sigma 2k \sin\theta.$$

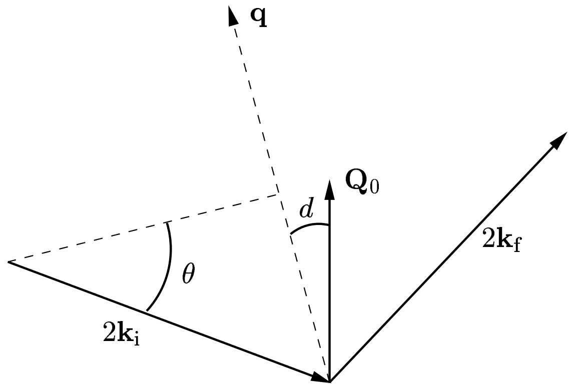

Figure 9.2: Computing the deviation d from the nominal scattering direction.

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

$$\Delta q \approx \varphi k \sin(2\theta)$$

(see figure 9.3). Hence we may sample φ from a Gaussian with standard deviation

$$\sigma \frac{2k\sin\theta}{k\sin(2\theta)} = \sigma \frac{2k\sin\theta}{2k\sin\theta\cos\theta} = \frac{\sigma}{\cos\theta}$$

to get good statistics.

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

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

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

$$\Pi(\varphi) = e^{\frac{-d(\varphi)^2}{2\sigma^2}} / \int_{-\pi}^{\pi} e^{\frac{-d(\varphi)^2}{2\sigma^2}} d\varphi \tag{9.2}$$

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

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

In the implementation, the integral in (9.2) 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.

The input parameters for Mosaic_simple are *zmin*, *zmax*, *ymin*, and *ymax* to define the surface of the crystal in the Y-Z plane; *mosaic* to give the FWHM of the mosaic spread; *R0* to give the reflectivity at the Bragg angle, and *Qx*, *Qy*, and *Qz* to give the scattering vector.

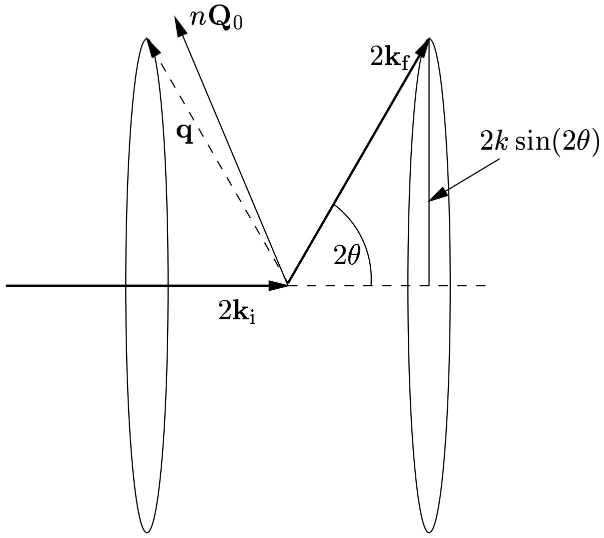

Figure 9.3: Scattering into the part of the Debye-Scherrer cone covered by the mosaic.

9.0.2 Mosaic_anisotropic: The crystal with anisotropic mosaic

The component **Mosaic_anisotropic** is a modified version of the Mosaic_simple component, intended to replace the Monocromator component from previous releases. It restricts the scattering vector to be perpendicular to the crystal surface, but extends the Mosaic_simple component by allowing different mosaics in the horizontal and vertical direction.

The code is largely similar to that for Mosaic_simple, and the documentation for the latter should be consulted for details. The differences are mainly due to two reasons:

- Some simplifications have been done since two of the components of the scattering vector are known to be zero.
- The computation of the Gaussian for the mosaic is done done using different mosaics for the two axes.

The input parameters for the component Mosaic_anisotropic are *zmin*, *zmax*, *ymin*, and *ymax* to define the size of the crystal (in meters); *mosaich* and *mosaicv* to define the mosaic (in minutes of arc); *r0* to define the reflectivity (no unit); and Q to set the length of the scattering vector (in Å^{-1}).

9.0.3 Single_crystal: The single crystal component

The physical model

The textbook expression for the scattering cross-section of a crystal is [**?**]:

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

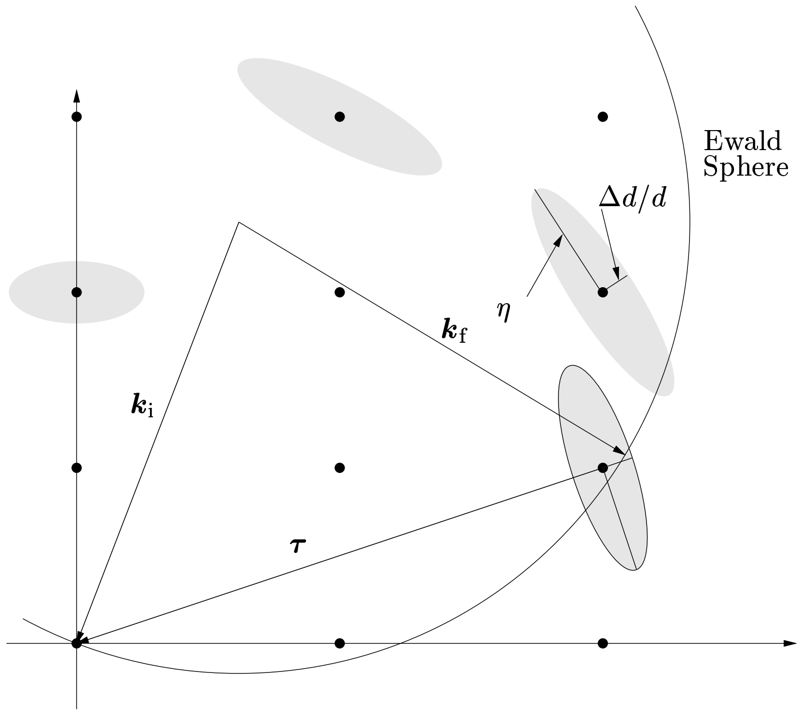

Figure 9.4: Ewald sphere construction for a single neutron showing the Gaussian broadening of reciprocal lattice points in their local coordinate system.

Here $|F_{\boldsymbol{\tau}}|^2$ is the structure factor, 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 a real crystal, however, reflections are not perfectly sharp. Because of imperfection and finite-size effects, there will be a small region around $\boldsymbol{\tau}$ in reciprocal space of possible scattering vectors.

The Single_crystal component 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 9.4). 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, the Single_crystal component uses a model which is at least mathematically plausible and which works as expected in the two common cases: (1) isotropic mosaic, and (2) two mosaic directions ("horizontal and vertical mosaic") perpendicular to a scattering vector.

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

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

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

$$G(x_1, x_2, x_3) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\frac{1}{2}\left(\frac{x_1^2}{\sigma_1^2} + \frac{x_2^2}{\sigma_2^2} + \frac{x_3^2}{\sigma_3^2}\right)} \tag{9.3}$$

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 (9.3) can be written as

$$G(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\boldsymbol{x}^{\mathrm{T}} D \boldsymbol{x}} \tag{9.4}$$

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

$$G(\boldsymbol{y}) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-(U(\boldsymbol{y}+\boldsymbol{\tau}))^{\mathrm{T}} D(U(\boldsymbol{y}+\boldsymbol{\tau}))} \tag{9.5}$$

The elastic coherent cross-section is then given by

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{coh.el.}} = N\frac{(2\pi)^3}{V_0}\sum_{\boldsymbol{\tau}} G(\boldsymbol{\tau}-\boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 \tag{9.6}$$

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

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

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 made whether the neutron is transmitted or not. 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 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, $dist \leq 5\max(\sigma_1,\sigma_2,\sigma_3)$.

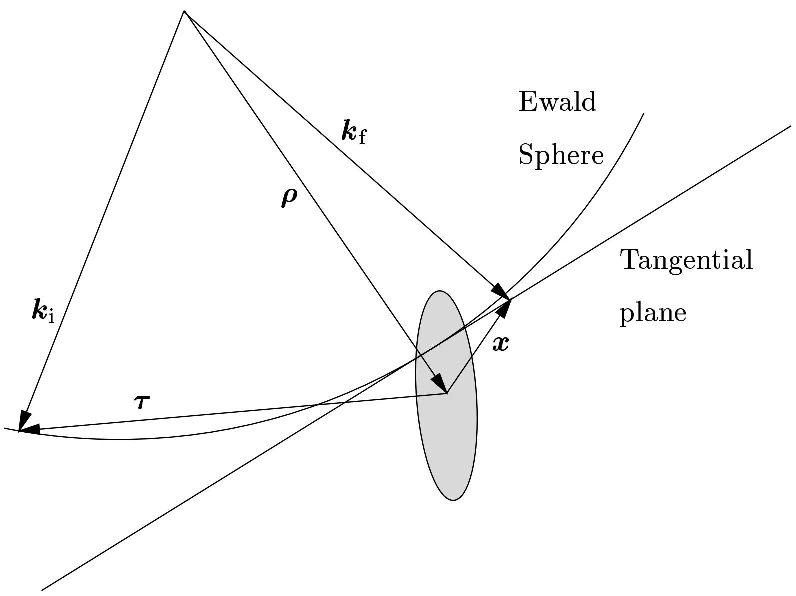

Figure 9.5: The scattering triangle in the single crystal.

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 9.5. 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

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

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

The equation for the plane is

$$\boldsymbol{P}(\boldsymbol{t}) = \boldsymbol{o} + B\boldsymbol{t}, \qquad \boldsymbol{t} \in \mathbb{R}^2 \tag{9.7}$$

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

$$G(\boldsymbol{x}(\boldsymbol{t})) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\boldsymbol{x}(\boldsymbol{t})^{\mathrm{T}} D \boldsymbol{x}(\boldsymbol{t})} \tag{9.8}$$

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 (9.8) can be re-written as

$$G(\boldsymbol{x}(\boldsymbol{t})) = \frac{1}{(\sqrt{2\pi})^3} \frac{1}{\sigma_1\sigma_2\sigma_3} e^{-\alpha} e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}} M (\boldsymbol{t}-\boldsymbol{t}_0)} \tag{9.9}$$

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 (9.9). 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

$$\begin{aligned} P(\boldsymbol{t})d\boldsymbol{t} &= \frac{1}{2\pi}e^{-\frac{1}{2}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}}d\boldsymbol{g} && (9.10)\\ &= \frac{1}{2\pi}\frac{1}{\det L}e^{-\frac{1}{2}(L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0))^{\mathrm{T}}(L^{-1}(\boldsymbol{t}-\boldsymbol{t}_0))}d\boldsymbol{t} && (9.11)\\ &= \frac{1}{2\pi}\frac{1}{\det L}e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}}M(\boldsymbol{t}-\boldsymbol{t}_0)}d\boldsymbol{t} && (9.12) \end{aligned}$$

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 (9.9). 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.

Computing the total coherent cross-section To get 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 (9.6) and (9.9):

$$\begin{aligned} \sigma_{\mathrm{coh},\boldsymbol{\tau}} &= \int N\frac{(2\pi)^3}{V_0}G(\boldsymbol{\tau}-\boldsymbol{\kappa})|F_{\boldsymbol{\tau}}|^2 d\Omega && (9.13)\\ &= \frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^3}{V_0}\frac{1}{(\sqrt{2\pi})^3}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2\int e^{-(\boldsymbol{t}-\boldsymbol{t}_0)^{\mathrm{T}}M(\boldsymbol{t}-\boldsymbol{t}_0)}d\boldsymbol{t} && (9.14)\\ &= \det(L)\frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{3/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2\int e^{-\frac{1}{2}\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}}d\boldsymbol{g} && (9.15)\\ &= 2\pi\det(L)\frac{1}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{3/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2 && (9.16)\\ &= \frac{\det(L)}{\boldsymbol{k}_i^2}N\frac{(2\pi)^{5/2}}{V_0}\frac{e^{-\alpha}}{\sigma_1\sigma_2\sigma_3}|F_{\boldsymbol{\tau}}|^2 && (9.17)\\ \sigma_{\mathrm{coh}} &= \sum_{\boldsymbol{\tau}}\sigma_{\mathrm{coh},\boldsymbol{\tau}} && (9.18) \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

The implementation

The equations describing the Single_crystal simulation 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 $|F_\tau|^2$, in barns. The middle three numbers are not used; they are nevertheless required since this makes the file format compatible with the output from the Crystallographica program [**?**].

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

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

Chapter 10

Powder-like sample components

In this section, we consider elastic coherent and incoherent scattering from polycrystalline samples. We have chosen to simulate the correct physical processes within the powder samples on a quite detailed level.

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* [**?**], but the effect is also important in powders. 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. For simplicity we ignore multiple scattering, which will be implemented in a later version of McStas.

10.0.4 Weight transformation in samples; focusing

Let us look in detail on how to simulate the physics of the scattering process within the powder. 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 10.1. 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 beam along this path is attenuated according to

$$P(l) = \exp(-l(\mu^s + \mu^a)), \tag{10.1}$$

where the normalization is taken to be $P(0) = 1$.

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

$$\Pi(l_1)dl = \mu^s P(l_1)dl, \tag{10.2}$$

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

$$\Pi(l_1, \Omega)dld\Omega = \mu^s P(l_1)P(l_2)\gamma(\Omega)d\Omega dl, \tag{10.3}$$

where $\gamma(\Omega)$ is the directional distribution of the scattered neutrons, and l_2 is determined by l_1, Ω, and the sample geometry, see figure 10.1.

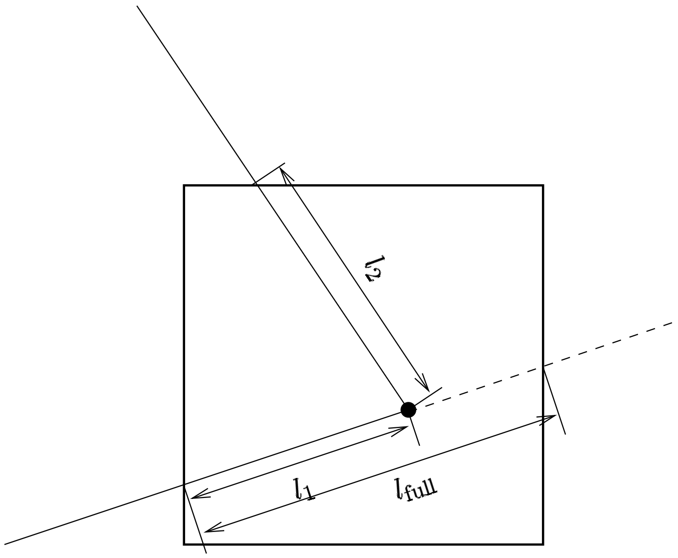

Figure 10.1: The geometry of a scattering event within a powder sample.

In our Monte-Carlo simulations, we will often choose the scattering parameters by making a Monte-Carlo choice of l_1 and Ω from a distribution different from $\Pi(l_1, \Omega)$. 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

$$f(l_1, \Omega) = g(\Omega)/l_{\text{full}}, \tag{10.4}$$

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

$$\pi_i(l_1, \Omega) = \mu^s l_{\text{full}} \exp\left[-(l_1 + l_2)(\mu^a + \mu^s)\right] \frac{\gamma(\Omega)}{g(\Omega)}. \tag{10.5}$$

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 (10.5), 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 of the powder type have been constructed with a focusing and a non-focusing option.

10.1 V_sample: An incoherent scatterer, the V-sample

A vanadium sample is frequently being used for calibration purposes, as almost all of the scattering from the sample occurs incoherently.

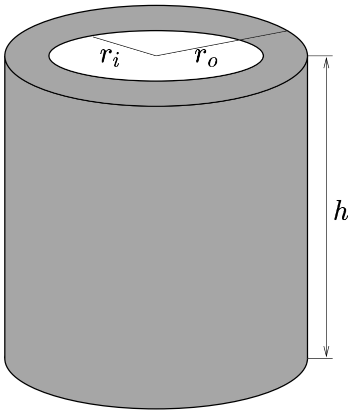

Figure 10.2: The geometry of the hollow-cylinder vanadium sample.

In the component **V_sample**, shown in **??** we assume *only* absorption and incoherent scattering. For the sample geometry, we have assumed the shape of 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 10.2.

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

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

$$g(\Omega) = \frac{1}{4\pi} \frac{x_\mathrm{t}^2 + y_\mathrm{t}^2 + z_\mathrm{t}^2}{(\pi r_\mathrm{t}^2)}. \tag{10.6}$$

The input parameters for the component **V_sample** are the sample dimensions (r_i, r_o, and h), the packing factor for the V-sample (`pack`), and the focusing parameters ($x_\mathrm{t}, y_\mathrm{t}, z_\mathrm{t}$, and r_t) for the target sphere. The relevant material parameters for V (σ_c^s, σ_c^a, and the unit cell volume V_c) are contained within the component.

Note: When simulating a realistic V-sample of this geometry 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 Appendix **??**.

10.2 Powder1: A general powder sample

10.2.1 General considerations

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

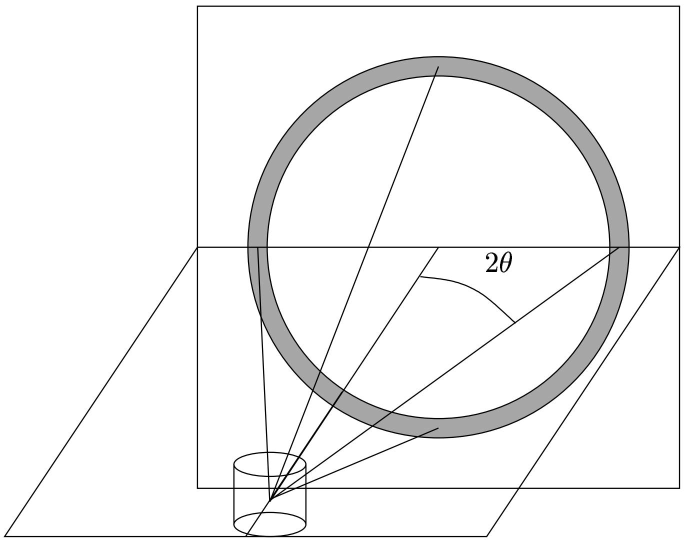

Figure 10.3: The scattering geometry of a powder sample showing the Debye-Scherrer cone and the Debye-Scherrer circle.

Bragg condition

$$n\lambda = 2d\sin\theta, \tag{10.7}$$

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 10.3. As all crystal orientations are realised in a powder sample, the neutrons are scattered within a *Debye-Scherrer cone* of opening angle 4θ [**?**].

Equation (10.7) may be cast into the form

$$|\mathbf{Q}| = 2|\mathbf{k}|\sin\theta, \tag{10.8}$$

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 textbook expression for the scattering intensity from one reflection in a slab-shaped powder sample, much larger than the beam cross section, reads [**?**]

$$\frac{P}{P_0} = \frac{\lambda^3 l_s}{4\pi r}\frac{\rho'}{\rho} tjN_c^2|F(\mathbf{Q})|^2\exp(-2W)\frac{\exp(-\mu^{\mathrm{a}}t/cos\theta)}{\sin^2(2\theta)} \tag{10.9}$$

$$|F(\mathbf{Q})|^2 = \left|\sum_j b_j\exp(\mathbf{R}_j\cdot\mathbf{Q})\right|^2, \tag{10.10}$$

where the sum in the structure factor runs over all atoms in one unit cell. The meanings and units of the symbols are

P_0	s^{-1}	Incoming intensity of neutrons
P	s^{-1}	Detected intensity of neutrons
l_s	m	Height of detector
r	m	Distance from sample to detector
ρ'/ρ	1	Packing factor of the powder
t	m	Slab thickness
j	1	Multiplicity of the reflection
N_c	m^{-3}	Density of unit cells in bulk material
$\lvert F(\mathbf{Q})\rvert^2$	m^2	Structure factor
$\exp(-2W)$	1	Debye-Waller factor
μ^a	m^{-1}	Linear attenuation factor due to absorption.

In analogy with this, the textbook expression for a cylinder shaped powder sample, completely illuminated by the beam, reads [**?**]

$$\frac{P}{\Psi_0} = \frac{V\rho'}{\rho} N_c^2 |F(\mathbf{Q})|^2 j \exp(-2W) \frac{A_{hkl}}{\sin(\theta)\sin(2\theta)} \frac{l_s}{2\pi r} \frac{\lambda^3}{4}, \tag{10.11}$$

where the new symbols are

Ψ_0	$\mathrm{s}^{-1}\mathrm{m}^{-2}$	Incoming beam flux
V	m^3	Sample volume
A_{hkl}	1	Attenuation factor.

Eq. (10.9) for a slab shaped sample may be cast into the form of the cylinder expression above by using the substitutions

Incoming flux	$P_0/(wh\cos\theta)$	$\rightarrow$	Ψ_0
Sample volume	wht	$\rightarrow$	V
Absorption factor	$\exp(-\mu^a t/\cos\theta)$	$\rightarrow$	A_{hkl},

where h and w are the height and width of the sample, respectively. Often, one defines the *scattering power* as

$$Q \equiv N^2 \frac{|F(\mathbf{Q})|^2 \lambda^3}{V \sin(2\theta)} = N_c^2 V \frac{\rho'}{\rho} \frac{|F(\mathbf{Q})|^2 \lambda^3}{\sin(2\theta)}, \tag{10.12}$$

where N is the number of unit cells.

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) only counts a fraction $f_d = l_s/(2\pi r \sin(2\theta))$ of the scattered neutrons. One may now calculate the linear attenuation coefficient in the material due to scattering (from one $\mathbf{Q}$-value only):

$$\mu^s \equiv -\frac{1}{P_0} \frac{d(P/f_d)}{dl} = \frac{Q}{V} j \exp(-2W)\cos(\theta). \tag{10.13}$$

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

10.2.2 This implementation

For component **Powder1**, we assume that the sample has the shape of a solid cylinder. Further, the incoherent scattering is only taken into account by the attenuation of the beam, given by (10.13) and σ_c^a. The incoherently scattered neutrons are not propagated through to the detector, but rather not generated at all. Focusing is performed by only scattering into one angular interval, $d\phi$ of the Debye-Scherrer circle. The center of this interval is located at the point where the Debye-Scherrer circle intersects the half-plane defined by the initial velocity, $\mathbf{v}_\mathrm{i}$, and a user-specified vector, $\mathbf{f}$. Multiple scattering is not implemented.

The input parameters for this component are

r	m	Radius of cylinder
h	m	Height of cylinder
σ_c^a	fm^2	Absorption cross section per unit cell (at 2200 m/s)
$\sigma_{i,c}^s$	(fm)^2	Incoherent scattering cross section per unit cell
ρ'/ρ	1	Packing factor
V_c	Å^3	Volume of unit cell
$\mathbf{Q}$	Å^{-1}	The reciprocal lattice vector under consideration
$\lvert F(\mathbf{Q}_j)\rvert^2$	(fm)^2	Structure factor
j	1	Multiplicity of reflection
$\exp(-2W)$	1	Debye-Waller factor
$d\phi$	deg	Angular interval of focusing
f_x	m	
f_y	m	Focusing vector
f_z	m	

The source text for the component is shown in Appendix **??**.

In a later version, more reciprocal lattice vectors will be allowed. Further, we intent to include the effect of multiple scattering.

Chapter 11

Inelastic scattering kernels

In this section, samples with inelastic scattering are described. Currently, only a single sample is available that scatters uniformly in $(\mathbf{Q}, \omega)$ and is used for computing resolution functions in tripple-axis instruments.

11.1 Res_sample: A uniform scatterer for resolution calculation

The component **Res_sample** models an inelastic sample that scatters completely homogeneous in position and energy; regardless of the state of the incoming neutron, all directions and energies for the scattered neutron have the same probability. This clearly does not correspond any physically realizable samples, but the component is very useful for computation of the resolution function and may also be used for test and debugging purposes. The component is designed to be used together with the **Res_monitor** component, described in section 8.13.

The shape of the sample is either a hollow cylinder (like the vanadium sample described in section 10.1) or a rectangular box. The hollow cylinder shape is specified with inner and outer radius *radius_i* and *radius_o* and height h. If *radius_o* is negative, the shape is instead a box of width *radius_i* along the X axis, height h, and thickness $-$*radius_o* along the Z axis, centered on the Z axis and with the front face in the X-Y plane. See figure 11.1.

The component only propagates the neutrons that are scattered; neutrons that would pass through or miss the sample are absorbed. There is no modeling of the cross section of the sample, secondary extinction *etc.*; the scattering probability is proportional to the neutron flight path length inside the sample, with the constant of proportionality arbitrarily set to $1/(2|radius\_o|)$. The reason for this is that the component is designed for computing the resolution function of an instrument, including the sample size but independent of any sample properties such as scattering and absorbtion cross sections.

The point of scattering in the sample is chosen at a random position along the neutron flight path inside the sample, and the scattered neutron is given a random energy and direction. The energy is selected in a user-specified interval $[E_0 - \Delta E; E_0 + \Delta E]$ which must be chosen large enough to cover all interesting neutrons, but preferably not excessively large for reasons of efficiency. Similarly, the direction is chosen in a user-specified range; the range is such that a sphere of given center and radius is fully illuminated.

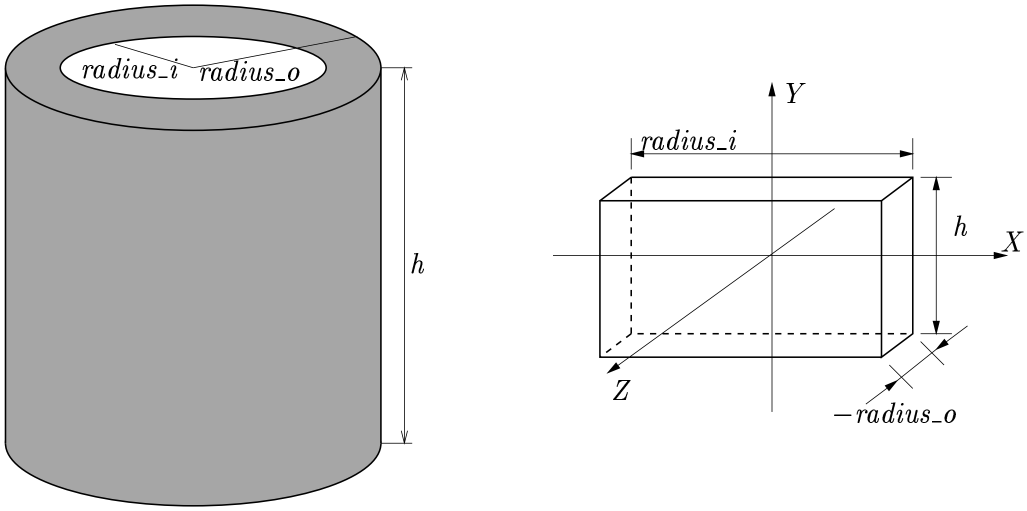

Figure 11.1: The two possible shapes of the **Res_sample** component.

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 $(\mathbf{Q}_i, \omega_i)$ for every scattering event i may be recorded and used to compute the resolution function of an instrument, as explained below. For an example of how to use the information in the output parameter, see the description of the **Res_monitor** component in section 8.13.

The input parameters to the **Res_sample** components are the sample dimensions *radius_i*, *radius_o*, and h, all in meters; the center of the scattered energy range *E0* and the energy spread *dE* in meV; and the target sphere position in the local coordinate system *target_x*, *target_y*, *target_z*, and radius *focus_r*, in meters. The only output parameter is *res_struct* containing information about the scattering event, with all vectors given in the local coordinate system of the component in units of meter.

11.1.1 Background

In an experiment, as well as in the simulation, the expected intensity is by definition of the resolution function given by

$$I = \int R(\mathbf{Q}, \omega)\sigma(\mathbf{Q}, \omega) d\mathbf{Q} d\omega$$

Here $I(\mathbf{Q}_0, \omega_0)$ is the measured or simulated intensity in the detector, R is the resolution function for the instrument in a given setup, σ is the scattering cross section of the sample, and $(\mathbf{Q}, \omega)$ denote the scattering vector and energy transfer in the sample. For the uniform scatterer, $\sigma(\mathbf{Q}, \omega) = 1/V_0$ is a constant, so we have

$$I = 1/V_0 \int R(\mathbf{Q}, \omega) d\mathbf{Q} d\omega$$

If we instead consider only the intensity contributed by scattering with parameters $(\mathbf{Q}, \omega)$ that lie within a small part $\Delta\Omega$ of the total phase space and has volume ΔV,

$$I_{\Delta\Omega} = 1/V_0 \int_{\Delta\Omega} R(\mathbf{Q}, \omega) d\mathbf{Q} d\omega = \frac{\Delta V}{V_0} R(\Delta\Omega)$$

(where $R(\Delta\Omega)$ denotes the average value of R over $\Delta\Omega$), we get a good approximation of the value of R provided that $\Delta\Omega$ is sufficiently small. This is useful with the output from the simulations, since $I_{\Delta\Omega}$ is approximated by

$$I_{\Delta\Omega} \approx \sum_{(\mathbf{Q_i}, \omega_i) \in \Delta\Omega} p_i$$

This can be used to histogram the resolution function or visualize it in different ways. The 3D visualization of the resolution function produced by the `mcresplot` program for example uses this by displaying a cloud of dots, the local density of which is proportional to the resolution function.

The `mcresplot` program also computes the covariance and resolution matrices. Letting $(x_i^1, x_i^2, x_i^3, x_i^4)$ denote the $(\mathbf{Q_i}, \omega_i)$ values obtained from the scattering events in the simulation and $\mu^j = (\sum_i p_i x_i^j)/(\sum_i p_i)$ the mean value of x_i^j, the covariance matrix is computed as

$$\mathbf{C}_{jk} = \Big(\sum_i p_i (x_i^j - \mu_j)(x_i^k - \mu_k)\Big) / \Big(\sum_i p_i\Big)$$

This covariance matrix is given in the local coordinate system of the sample component. The `mcresplot` program actually outputs the covariance matrix in another coordinate system which is rotated around the Y axis so that the projection to the X-Z plane of the average scattering vector $\mathbf{Q}_{\text{avg}} = (\sum_i p_i \mathbf{Q}_i)/(\sum_i p_i)$ is parallel to the X axis.

The resolution matrix $\mathbf{M}$ is the inverse of the covariance matrix and is also output in the rotated coordinate system by `mcresplot`. The 4-dimensional gaussian distribution, defined by

$$f(\mathbf{X}) = e^{-\frac{1}{2}\mathbf{X}^T \mathbf{M} \mathbf{X}} \tag{11.1}$$

where $\mathbf{X} = (\mathbf{Q}, \omega)$, has covariance matrix $\mathbf{C}$ and thus defines the gaussian resolution function with the same covariance as the resolution computed by the simulation.

The `mcresplot` program provides for the simultaneous visualization of the computed and the gaussian resolution function by obtaining an appropriate number of random points with the statistical distribution (11.1). Each point $\mathbf{X}$ is obtained as follows: A vector $\mathbf{Y}$ is generated of four individually gaussian distributed random numbers with mean zero and variance one. Using the Cholesky decomposition of $\mathbf{C}$, $\mathbf{C} = \mathbf{L}\mathbf{L}^T$, we have

$$\mathbf{X} = \mathbf{L}\mathbf{Y}.$$

Chapter 12

Instrument examples

Here, we give a short description of three selected instruments. We present the McStas versions of the Risø standard triple axis spectrometer TAS1 (12.2) and the ISIS time-of-flight spectrometer PRISMA (12.3). Before that, however, we present one example of a component test instrument: the instrument to test the component **V_sample** (12.1). These instrument files are included in the McStas distribution in the `examples/` directory. All the instrument examples there-in may be executed automatically throught the McStas self-test procedure (see section **??**). It is also our intention to extend the list of instrument examples extensively and perhaps publish them in a separate report.

12.1 A test instrument for the component V_sample

This instrument is one of many test instruments written with the purpose of testing the individual components. We have picked this instrument both because we would like to present an example test instrument and because it despite its simplicity has produced quite non-trivial results, also giving rise to the McStas logo.

The instrument consists of a narrow source, a 60' collimator, a V-sample shaped as a hollow cylinder with height 15 mm, inner diameter 16 mm, and outer diameter 24 mm at a distance of 1 m from the source. The sample is in turn surrounded by an unphysical 4π-PSD monitor with 50×100 pixels and a radius of 10^6 m. The set-up is shown in figure 12.1.

12.1.1 Scattering from the V-sample test instrument

In figure 12.2, we present the radial distribution of the scatting from an evenly illuminated V-sample, as seen by a spherical PSD. It is interesting to note that the variation in the scattering intensity is as large as 10%. This is an effect of attenuation of the beam in the cylindrical sample.

12.2 The triple axis spectrometer TAS1

With this instrument definition, we have tried to create a very detailed model of the conventional cold-source triple-axis spectrometer TAS1 at Risø National Laboratory. Un-

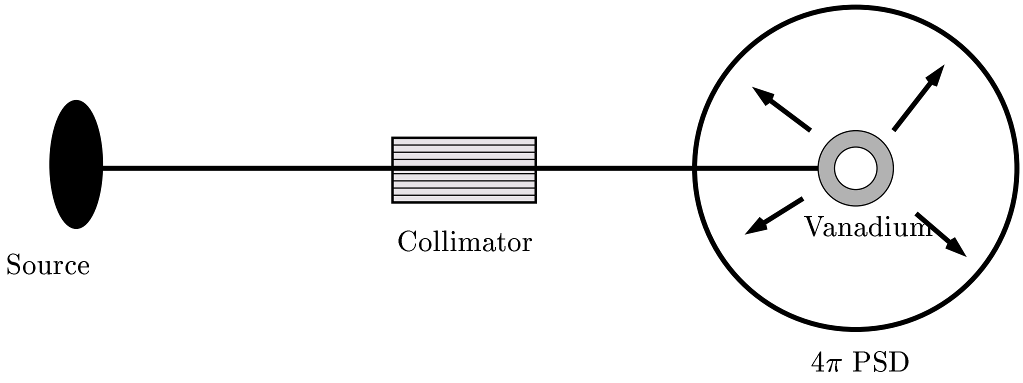

Figure 12.1: A sketch of the test instrument for the component V_sample.

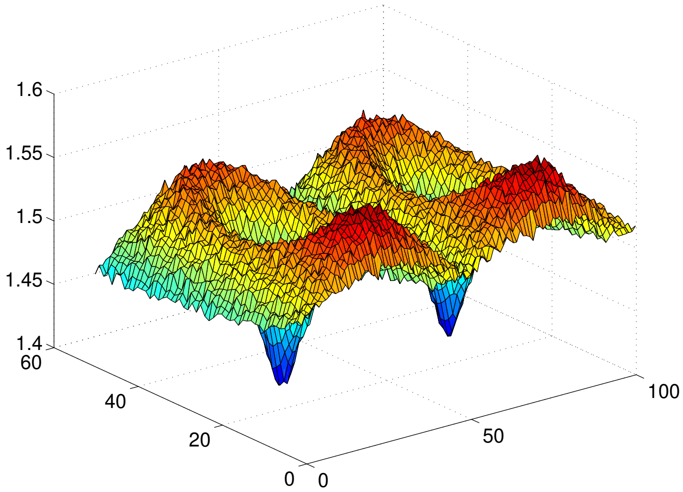

Figure 12.2: Scattering from a V-sample, measured by a spherical PSD. The sphere has been transformed onto a plane and the intensity is plotted as the third dimension. A colour version of this picture is found on the title page of this manual.

fortunately, no neutron scattering is performed at Risø anymore, but it still serves as a good example. Except for the cold source itself, all components used have quite realistic properties. Furthermore, the overall geometry of the instrument has been adapted from the detailed technical drawings of the real spectrometer. The TAS 1 simulation is the first detailed work performed with the McStas package.For further details see reference [11].

At the spectrometer, the channel from the cold source to the monochromator is asymmetric, since the first part of the channel is shared with other instruments. In the instrument definition, this is represented by three slits. For the cold source, we use a flat energy distribution (component **Source_flat**) focusing on the third slit.

The real monochromator consist of seven blades, vertically focusing on the sample. The angle of curvature is constant so that the focusing is perfect at 5.0 meV (20.0 meV for 2nd order reflections) for a 1×1 cm^2 sample. This is modeled directly in the instrument definition using seven **Monochromator** components. The mosaicity of the pyrolytic graphite crystals is nominally 30' (FWHM) in both directions. However, the simulations indicated that the horisontal mosaicities of both monochromator and analyser were more likely 45'. This was used for all mosaicities in the final instrument definition.

The monochromator scattering angle, in effect determining the incoming neutron energy, is for the real spectrometer fixed by four holes in the shielding, corresponding to the energies 3.6, 5.0, 7.2, and 13.7 meV for first order neutrons. In the instrument definition, we have adapted the angle corresponding to 5.0 meV in order to test the simulations against measurements performed on the spectrometer.

The exit channel from the monochromator may on the spectrometer be narrowed down from initially 40 mm to 20 mm by an insert piece. In the simulations, we have chosen the 20 mm option and modeled the channel with two slits to match the experimental set-up.

In the test experiments, we used two standard samples: An Al_2O_3 powder sample and a vanadium sample. The instrument definitions use either of these samples of the correct size. Both samples are chosen to focus on the opening aperture of collimator 2 (the one between the sample and the analyser). Two slits, one before and one after the sample, are in the instrument definition set to the opening values which were used in the experiments.

The analyser of the spectrometer is flat and made from pyrolytic graphite. It is placed between an entry and an exit channel, the latter leading to a single detector. All this has been copied into the instrument definition, where the graphite mosaicity has been set to 45'.

On the spectrometer, Soller collimators may be inserted at three positions: Between monochromator and sample, between sample and analyser, and between analyser and detector. In our instrument definition, we have used 30', 28', and 67' collimators on these three positions, respectively.

An illustration of the TAS1 instrument is shown in figure 12.3. Test results and data from the real spectrometer are shown in Appendix 12.2.1.

12.2.1 Simulated and measured resolution of TAS1

In order to test the McStas package on a qualitative level, we have performed a very detailed simulation of the conventional triple axis spectrometer TAS1, Risø. The measurement series constitutes a complete alignment of the spectrometer, using the direct beam and scattering from V and Al_2O_3 samples at an incoming energy of 20.0 meV, using

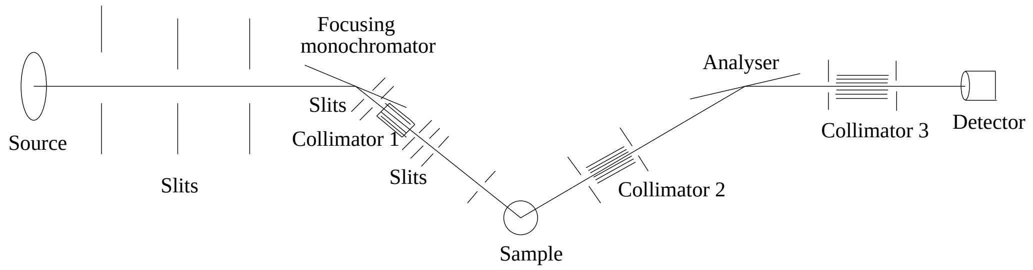

Figure 12.3: A sketch of the TAS1 instrument.

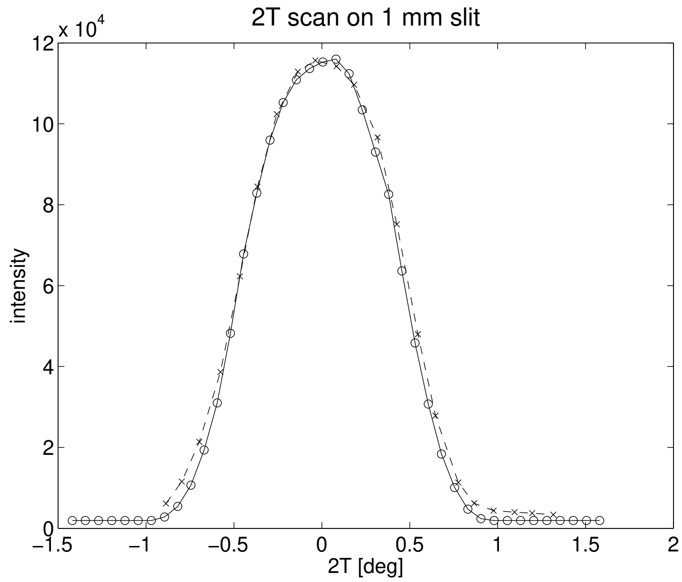

Figure 12.4: Scans of $2\theta_s$ in the direct beam with 1 mm slit on the sample position. "×": measurements, "o": simulations Collimations: open-30'-open-open.

the second order scattering from the monochromator. In the instrument definitions, we have used all available information about the spectrometer. However, the mosaicities of the monochromator and analyser are set to 45' in stead of the quoted 30', since we from our analysis believe this to be much closer to the truth.

In these simulations, we have tried to reproduce every alignment scan with respect to position and width of the peaks, whereas we have not tried to compare absolute intensities. Below, we show a few comparisons of the simulations and the measurements.

Figure 12.4 shows a scan of $2\theta_s$ on the collimated direct beam in two-axis mode. A 1 mm slit is placed on the sample position. Both the measured width and non-Gaussian peak shape are well reproduced by the McStas simulations.

In contrast, a simulated $2\theta_a$ scan in triple-axis mode on a V-sample showed a surprising offset from zero, see Figure 12.5. However, a simulation with a PSD on the sample position showed that the beam center was 1.5 mm off from the center of the sample, and this was

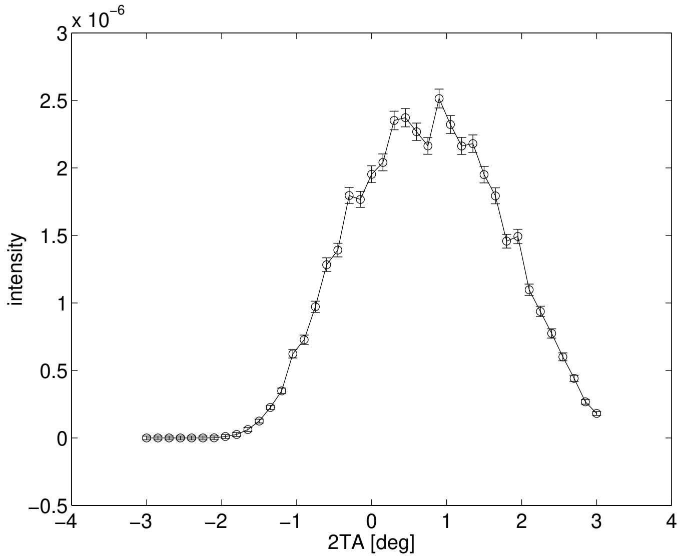

Figure 12.5: First simulated $2\theta_a$ scan on a vanadium sample. Collimations: open-30'-28'-open.

important since the beam was no wider than the sample itself. A subsequent centering of the beam resulted in a nice agreement between simulation and measurements. For a comparison on a slightly different instrument (analyser-detector collimator inserted), see Figure 12.6.

The result of a $2\theta_s$ scan on an Al_2O_3 powder sample in two-axis mode is shown in Figure 12.7. Both for the scan in focusing mode $(+ - +)$ and for the one in defocusing mode $(+ + +)$ (not shown), the agreement between simulation and experiment is excellent.

As a final result, we present a scan of the energy transfer $E_a = \hbar\omega$ on a V-sample. The data are shown in Figure 12.8.

12.3 The time-of-flight spectrometer PRISMA

In order to test the time-of-flight aspect of McStas, we have in collaboration with Mark Hagen, ISIS, written a simple simulation of a time-of-flight instrument loosely based on the ISIS spectrometer PRISMA. The simulation was used to investigate the effect of using a RITA-style analyser instead of the normal PRISMA backend.

We have used the simple time-of-flight source **Tof_source**. The neutrons pass through a beam channel and scatter off from a vanadium sample, pass through a collimator on to the analyser. The RITA-style analyser consists of seven analyser crystals that can be rotated independently around a vertical axis. After the analysers we have placed a PSD and a time-of-flight detector.

To illustrate some of the things that can be done in a simulation as opposed to a real-life experiment, this example instrument further discriminates between the scattering off each individual analyser crystal when the neutron hits the detector. The analyser component is modified so that a global variable `neu_color` keeps track of which crystal scatters the

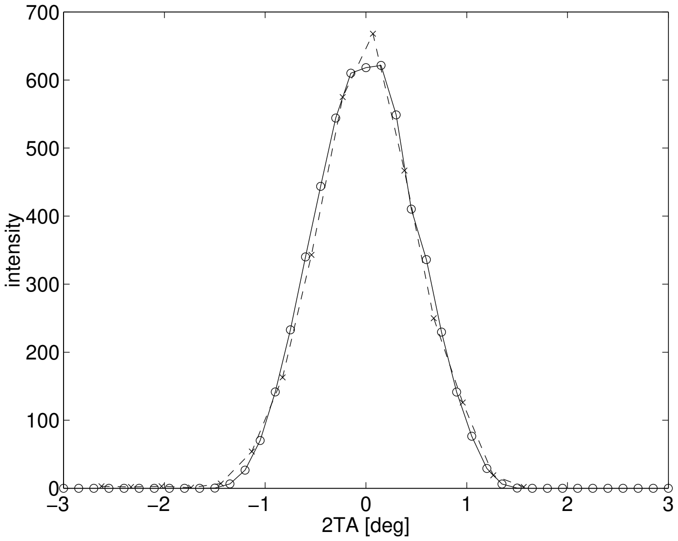

Figure 12.6: Corrected $2\theta_a$ scan on a V-sample. Collimations: open-30'-28'-67'. "$\times$": measurements, "o": simulations.

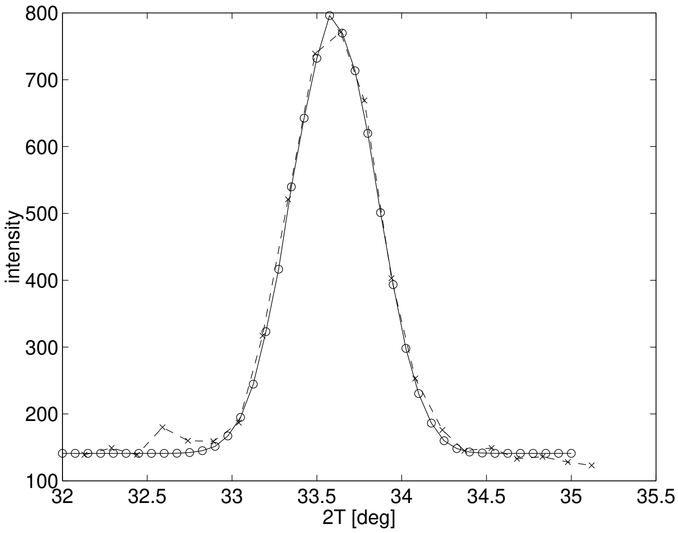

Figure 12.7: $2\theta_s$ scans on Al_2O_3 in two-axis, focusing mode. Collimations: open-30'-28'-67'. "$\times$": measurements, "o": simulations. A constant background is added to the simulated data.

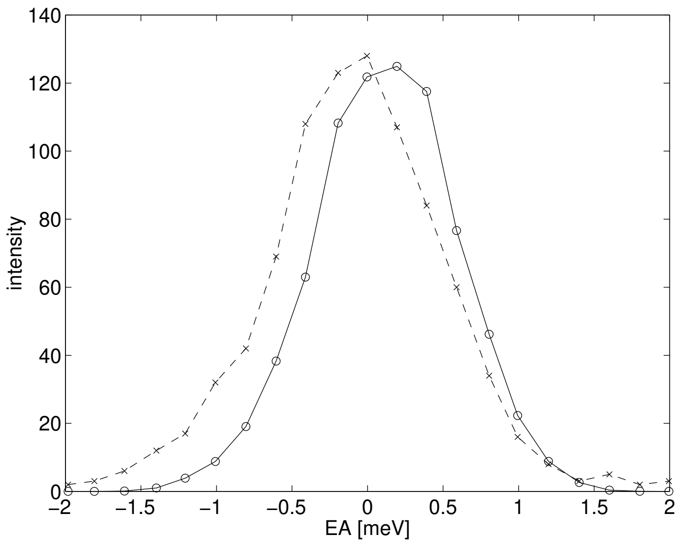

Figure 12.8: Scans of the analyser energy on a V-sample. Collimations: open-30'-28'-67'. "$\times$": measurements, "o": simulations.

neutron. The detector component is then modified to construct seven different time-of-flight histograms, one for each crystal (see the source code for the instrument for details). One way to think of this is that the analyser blades paint a color on each neutron which is then observed in the detector. An illustration of the instrument is shown in figure 12.9. Test results are shown in Appendix 12.3.1.

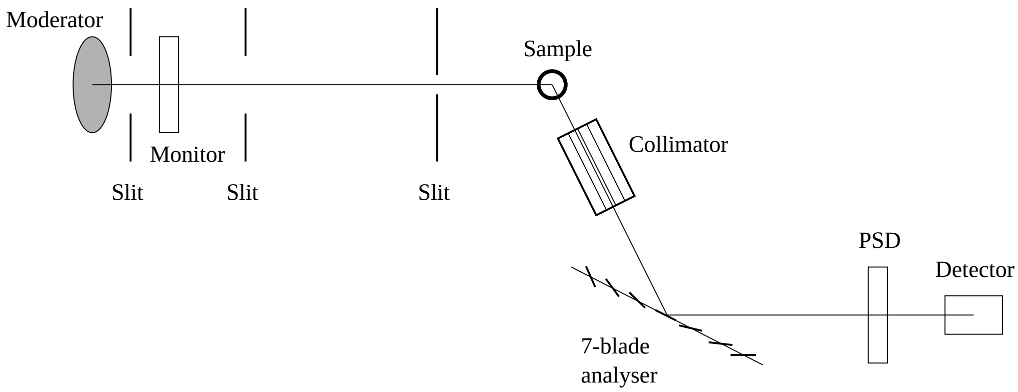

Figure 12.9: A sketch of the PRISMA instrument.

12.3.1 Simple spectra from the PRISMA instrument

A plot from the detector in the PRISMA simulation is shown in Figure 12.10. These results were obtained with each analyser blade rotated one degree relative to the previous one. The separation of the spectra of the different analyser blades is caused by different

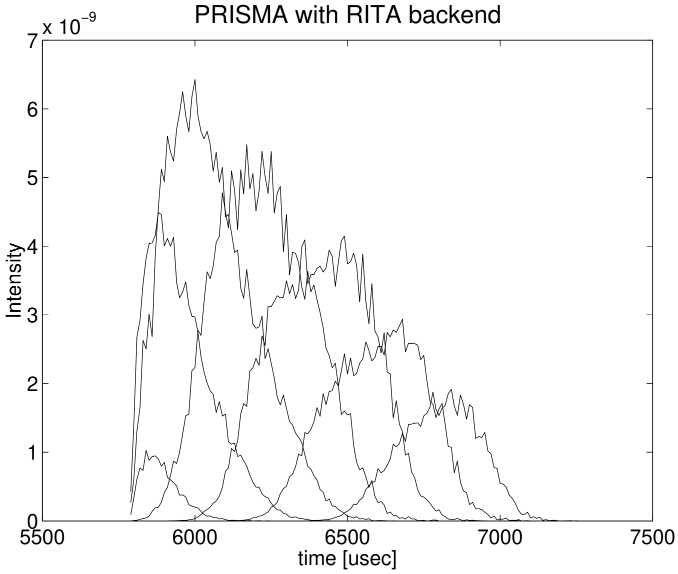

Figure 12.10: Test result from PRISMA instrument using "coloured neutrons". Each graph shows the neutrons scattered from one analyser blade.

energy of scattered neutrons and different flight path length from source to detector. We have not performed any quantitative analysis of the data at this time.

Appendix A

Libraries and conversion constants

The McStas Library contains a number of built-in functions and conversion constants which are useful when constructing components. These are stored in the `share` directory of the `MCSTAS` library.

Within these functions, the 'Run-time' part is available for all component/instrument descriptions. The other parts (see table **??**) are dynamic, that is they are not pre-loaded, but only imported once when a component requests it using the `%include` McStas keyword. For instance, within a component C code block, (usually SHARE or DECLARE):

```
%include "read_table-lib"
```

will include the 'read_table-lib.h' file, and the 'read_table-lib.c' (unless the `--no-runtime` option is used with `mcstas`). Similarly,

```
%include "read_table-lib.h"
```

will *only* include the 'read_table-lib.h'. The library embedding is done only once for all components (like the SHARE section). For an example of implementation, see the Res_monitor component.

Here, we present a short list of both each of the library contents and the run-time features.

A.1 Run-time calls and functions

Here we list a number of preprogrammed macros which may ease the task of writing component and instrument definitions.

A.1.1 Neutron propagation

- **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. If the neutron velocity points away from the $z = 0$ plane, the neutron is absorbed. If component is centered, in order to avoid the neutron to be propagated there, 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.

- **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.

- **SCATTER**. This macro is used to denote a scattering event inside a component, see section **??**. It should be used e.g to indicate that a component has 'done something' (sctattered or detected). This does not affect the simulation at all, and is mainly used by the `MCDISPLAY` section and the `GROUP` modifier (see **??** and **??**). See also the SCATTERED variable (below).

A.1.2 Coordinate and component variable retrieval

- **MC_GETPAR**(). This may be used in the finally section of an instrument definition to reference the output parameters of a component. See page **??** for details.

- **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 matrice is referred to as ROT_A_CURRENT_COMP$[m][n]$, where m and n are 0, 1, or 2.

- **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, 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 (always executed when existing) to branch action depending if the component acted or not on the current neutron.

- **extend_list**(n, &*arr*, &*len*, *elemsize*). Given an array *arr* with *len* elements each of size *elemsize*, make sure that the array is big enough to hold at least n elements, by extending *arr* and *len* if necessary. Typically used when reading a list of numbers from a data file when the length of the file is not known in advance.

- **mcset_ncount**(n). Sets the number of neutron histories to simulate to n.

- **mcget_ncount**(). Returns the number of neutron histories to simulate (usually set by option `-n`).

- **mcget_run_num**(). Returns the number of neutron histories that have been simulated until now.

A.1.3 Coordinate transformations

- **coords_set**(x, y, z) returns a Coord structure (like POS_A_CURRENT_COMP) with x, y and z members.

- **coords_get**(P, &x, &y, &z) copies the x, y and z members of the Coord structure P into x, y, z variables.

- **coords_add**(a, b), **coords_sub**(a, b), **coords_neg**(a) enable to operate on coordinates, and return the resulting Coord structure.

- **rot_set_rotation**(*Rotation t*, ϕ_x, ϕ_y, ϕ_z) Get transformation for rotation first ϕ_x around x axis, then ϕ_y around y, then ϕ_z around z. t should be a 'Rotation' ([3][3] 'double' matrix).

- **rot_mul***(Rotation t1, Rotation t2, Rotation t3)* performs $t3 = t1.t2$.

- **rot_copy***(Rotation dest, Rotation src)* performs *dest* = *src* for Rotation arrays.

- **rot_transpose**(*Rotation src, Rotation dest*) performs $dest = src^t$.
- **rot_apply**(*Rotation t, Coords a*) returns a Coord structure which is $t.a$

A.1.4 Mathematical routines

- **NORM**(x, y, z). Normalizes the vector (x, y, z) to have length 1.
- **scalar_prod**$(a_x, a_y, a_z, b_x, b_y, b_z)$. Returns the scalar product of the two vectors (a_x, a_y, a_z) and (b_x, b_y, b_z).
- **vecprod**$(a_x, a_y, a_z, b_x, b_y, b_z, c_x, c_y, c_z)$. Sets (a_x, a_y, a_z) equal to the vector product $(b_x, b_y, b_z) \times (c_x, c_y, c_z)$.
- **rotate**$(x, y, z, v_x, v_y, v_z, \varphi, a_x, a_y, a_z)$. Set (x, y, z) to the result of rotating the vector (v_x, v_y, v_z) the angle φ (in radians) around the vector (a_x, a_y, a_z).
- **normal_vec**(&n_x, &n_y, &n_z, x, y, z). Computes a unit vector (n_x, n_y, n_z) normal to the vector (x, y, z).

A.1.5 Output from detectors

- **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. See section **??** for details.
- **DETECTOR_OUT_1D**(...). Used to output the results from a one-dimentional detector. See section **??** for details.
- **DETECTOR_OUT_2D**(...). Used to output the results from a two-dimentional detector. See section **??** for details.
- **DETECTOR_OUT_3D**(...). Used to output the results from a three-dimentional detector. Arguments are the same as in DETECTOR_OUT_2D, but with the additional z axis (the signal). Resulting data files are treated as 2D data, but the 3rd dimension is specified in the *type* field.
- **mcheader_out**(*FILE *f,char *parent, int m, int n, int p, char *xlabel, char *ylabel, char *zlabel, char *title, char *xvar, char *yvar, char *zvar, double x1, double x2, double y1, double y2, double z1, double z2, char *filename*) appends a header file using the current data format setting. Signification of parameters may be found in section **??**. Please contact the authors in case of perplexity.
- **mcinfo_simulation**(*FILE *f, mcformat, char *pre, char *name*) is used to append the simulation parameters into file f (see for instance the Res_monitor component). Internal variable $mcformat$ should be used as specified. Please contact the authors in case of perplexity.

A.1.6 Ray-geometry intersections

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

- **cylinder_intersect**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, r, h). Similar to **box_intersect**, but using a cylinder of height h and radius r, centered at the origin.

- **sphere_intersect**(&t_1, &t_2, x, y, z, v_x, v_y, v_z, r). Similar to **box_intersect**, but using a sphere of radius r.

A.1.7 Random numbers

- **rand01**(). Returns a random number distributed uniformly between 0 and 1.

- **randnorm**(). Returns a random number from a normal distribution centered around 0 and with $\sigma = 1$. The algorithm used to get the normal distribution is explained in [12], chapter 7.

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

- **randvec_target_circle**(&v_x, &v_y, &v_z, &$d\Omega$, aim$_x$, aim$_y$, aim$_z$, r_f). Generates a random vector (v_x, v_y, v_z), of the same length as (aim$_x$, aim$_y$, aim$_z$), which is targeted at a *disk* centered at (aim$_x$, aim$_y$, aim$_z$) with radius r_f (in meters), and perpendicular to the *aim* vector.. All directions that intersect the sphere are chosen with equal probability. The solid angle of the sphere 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$_z$,$height, width, Rot$) does the same as randvec_target_circle but targetting at a rectangle with angular dimensions *height* and *width* (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 target at a guide entry window.

A.2 Reading a data file into a vector/matrix (Table input)

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. Available functions are:

- **Table_Init**(&$Table$) and **Table_Free**(&$Table$) initialize and free allocated memory blocks
- **Table_Read**(&$Table$, $filename$, *block*) reads numerical block number *block* (0 for all) data from *text* file $filename$ into $Table$. 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.
- **Table_Rebin**(&$Table$) rebins $Table$ rows with increasing, evenly spaced first column (index 0), e.g. before using Table_Value.
- **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_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.

The format of text files is free. Lines starting by '`# ; % /`' characters are considered to be comments. 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 plateform 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 section **??**), and if not found, in the `data` subdirectory of the `MCSTAS` library location. This way, you do not need to have local copies of the McStas Library Data files (see table **??**).

A usage example for this library part may be:

```
t_Table rTable;       % declares a t_Table structure
char file="BeO.trm";  % a file name
double x,y;

Table_Init(&rTable);  % initialize the table to empty state
Table_Read(&rTable, file, 1); % reads the first numerical block
Table_Info(rTable);   % display table informations
...
x = Table_Index(rTable, 2,5);  % reads the 3rd row, 6th column element
                               % of the table. Indexes start at zero in C.
y = Table_Value(rTable, 1.45,1);  % looks for value 1.45 in 1st column (x axis)
                                  % and extract 2nd column value of that row
Table_Free(&rTable);  % free allocated memory for table
```

Additionally, if the block number (3rd) argument of **Table_Read** is 0, all blocks will be catenated. The **Table_Value** function assumes that the 'x' axis is the first column (index 0). Other functions are used the same way with a few additional parameters, e.g. specifying an offset for reading files, or reading binary data.

You may look into, for instance, the Monochromator_curved component, or the Virtual_input component for other implementation examples.

A.3 Monitor_nD Library

This library gathers a few functions used by a set of monitors e.g. Monitor_nD, Res_monitor, Virtual_output, ... It may monitor any kind of data, create the data files, and may display many geometries (for `mcdisplay`). Refer to these components for implementation examples, and ask the authors for more details.

A.4 Adaptative importance sampling Library

This library is currently only used by the components Source_adapt and Adapt_check. It performs adaptative importance sampling of neutrons for simulation efficiency optimization. Refer to these components for implementation examples, and ask the authors for more details.

A.5 Vitess import/export Library

This library is used by Vitess_input, Vitess_output components, as well as the `mcstas2vitess` utility (see section **??**). Refer to these components for implementation examples, and ask the authors for more details.

A.6 Constants for unit conversion etc.

The following predefined constants are useful for conversion between units

Name	Value	Conversion from	Conversion to
DEG2RAD	$2\pi/360$	Degrees	radians
RAD2DEG	$360/(2\pi)$	Radians	degrees
MIN2RAD	$2\pi/(360 \cdot 60)$	Minutes of arc	radians
RAD2MIN	$(360 \cdot 60)/(2\pi)$	Radians	minutes of arc
V2K	$10^{10} \cdot m_{\mathrm{N}}/\hbar$	Velocity (m/s)	**k**-vector (Å^{-1})
K2V	$10^{-10} \cdot \hbar/m_{\mathrm{N}}$	**k**-vector (Å^{-1})	Velocity (m/s)
VS2E	$m_{\mathrm{N}}/(2e)$	Velocity squared (m^2 s^{-2})	Neutron energy (meV)
SE2V	$\sqrt{2e/m_{\mathrm{N}}}$	Square root of neutron energy (meV$^{1/2}$)	Velocity (m/s)
FWHM2RMS	$1/\sqrt{8\log(2)}$	Full width half maximum	Root mean square (standard deviation)
RMS2FWHM	$\sqrt{8\log(2)}$	Root mean square (standard deviation)	Full width half maximum
MNEUTRON	$1.67492E-27kg$	Neutron mass	
HBAR	$1.05459E-34Js$	Planck constant	
PI	3.14159265358979323846	π	

Further, we have defined the constants **PI**= π and **HBAR**= $\hbar$.

Appendix B

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 authors.

- **Arm** A generic McStas component which defines a frame of reference for other components.
- **Component** One unit (*e.g.* optical element) in a neutron spectrometer.
- **Definition parameter** An input parameter for a component. For example the radius of a sample component or the divergence of a collimator.
- **Input parameter** For a component, either a definition parameter or a setting parameter. These parameters are supplied by the user to define the characteristics of the particular instance of the component definition. For an instrument, a parameter that can be changed at simulation run-time.
- **Instrument** An assembly of McStas components defining a neutron spectrometer.
- **McStas** Monte Carlo Simulation of Triple Axis Spectrometers (the name of this project).
- **Output parameter** An output parameter for a component. For example the counts in a monitor. An output parameter may be accessed from the instrument in which the component is used using `MC_GETPAR`.
- **Run-time** C code, contained in the files `mcstas-r.c` and `mcstas-r.h` included in the McStas distribution, that declare functions and variables used by the generated simulations.
- **Setting parameter** Similar to a definition parameter, but with the restriction that the value of the parameter must be a number.

Bibliography

[1] K. Lefmann and K. Nielsen. *Neutron News*, **10**, 20–23, 1999.

[2] Kim Lefmann Per-Olof Astrand Marc Hagen Peter Willendrup, Emmanuel Farhi and Kristian Nielsen. *User and Programmers Guide to the Neutron Ray-Tracing Package McStas, Version 1.8.* Risoe Report, 2004.

[3] See http://neutron.risoe.dk/mcstas/.

[4] See http://strider.lansce.lanl.gov/NISP/Welcome.html.

[5] T. E. Mason, K. N. Clausen, G. Aeppli, D. F. McMorrow, and J. K. Kjems. *Can. J. Phys.*, **73**, 697–702, 1995.

[6] K. N. Clausen, D. F. McMorrow, K. Lefmann, G. Aeppli, T. E. Mason, A. Schröder, M. Issikii, M. Nohara, and H. Takagi. *Physica B*, **241-243**, 50–55, 1998.

[7] K. Lefmann, D. F. McMorrow, H. M. Rønnow, K. Nielsen, K. N. Clausen, B. Lake, and G. Aeppli. *Physica B*, **283**, 343–354, 2000.

[8] See http://www.sns.gov/.

[9] See http://www.ess-europe.de.

[10] See http://www.hmi.de/projects/ess/vitess/.

[11] A. Abrahamsen, N. B. Christensen, and E. Lauridsen. McStas simulations of the TAS1 spectrometer. Student's report, Niels Bohr Institute, University of Copenhagen, 1998.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1986.

Index

Bibliographic Data Sheet			**Risø–R–1xyz(EN)**
Title and author(s)			
Component Manual to the Neutron Ray-Tracing Package McStas, Version 1.8			
Peter Kjær Willendrup, Kim Lefmann, Emmanuel Farhi			
ISBN			ISSN
87–550–2929–9; 87–550–2930–2 (Internet)			0106–2840
Dept. or group			Date
Materials Research Department			May 15th, 2003
Groups own reg. number(s)		Project/contract No.	
—		—	
Pages	Tables	Illustrations	References
86	2	15	10

Abstract (Max. 2000 char.)

The software package McStas is a tool for carrying out Monte Carlo ray-tracing simulations of neutron scattering instruments with high complexity and precision. The simulations can compute all aspects of the performance of instruments and can thus be used to optimize the use of existing equipment, design new instrumentation, and carry out virtual experiments. McStas is based on a unique design where an automatic compilation process translates high-level textual instrument descriptions into efficient ANSI-C code. This design makes it simple to set up typical simulations and also gives essentially unlimited freedom to handle more unusual cases.
This report constitutes the reference manual for McStas, and, together with the manual for the McStas components, it contains full documentation of all aspects of the program. It covers the various ways to compile and run simulations, a description of the meta-language used to define simulations, and some example simulations performed with the program.

Descriptors

Neutron Instrumentation; Monte Carlo Simulation; Software

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