Aalto University
School of Science
and Technology

PHYS-E0562 Nuclear Engineering, advanced course
Lecture 3 – Monte Carlo simulation

Jaakko Leppänen (Lecturer), Ville Valtavirta (Assistant)

Department of Applied Physics
Aalto University, School of Science
Jaakko.Leppanen@aalto.fi

Mar. 21, 2019

Topics of this lecture

Background:

- Transport theory revisited: deterministic and Monte Carlo approach
- General description of the Monte Carlo method
- Probability and statistics: basic concepts, sampling from distributions

Monte Carlo transport simulation:

- CSG geometry type: surfaces, cells, universes, lattices, boundary conditions
- Simulated random walk: sampling of particle path length, surface-tracking, delta-tracking
- Reactions: fission, capture, scattering

Simulating the particle population:

- External and criticality source simulations
- Collecting the results: analog and implicit flux estimators
- Implicit Monte Carlo techniques

Monte Carlo applications

What is expected to be known before the lecture

Probability and statistics:

- General idea of random variables and probability distributions
- Basic concepts: statistical mean, standard deviation, confidence intervals

Reactor physics:

- Interaction types
- Basic concepts: macroscopic and microscopic cross section, flux, reaction rate, particle mean-free-path
- Neutron chain reaction

Technical:

- Reactor operating principles
- Reactor geometries

The methods presented at this lecture correspond to what is used in the Serpent Monte Carlo neutron transport code developed at VTT. Even so, most of the topics are quite general and they apply to other codes and photon transport simulations as well.

Transport theory revisited: deterministic approach

All deterministic transport methods are based on the concept of neutron flux, $\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E)$, which is essentially a six-dimensional density-like function describing the collective behavior of the neutron population.

The neutron flux is needed for forming the transport equation:

$$\frac{1}{v}\frac{\partial}{\partial t}\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E, t) + \hat{\boldsymbol{\Omega}} \cdot \nabla\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E, t) + \Sigma(\boldsymbol{r}, E)\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E, t) = Q + S + F \tag{1}$$

where Q is the external source, S is the scattering source and F is the fission source. Even though the equation is linear,[1] it is difficult to solve because:

1) The geometries are complex and heterogeneous
2) Cross sections are complicated functions of neutron energy
3) The angular dependence of streaming and scattering source term is difficult to handle

When dealing with coupled problems, such as the modeling of an operating nuclear reactor, even the linearity assumption breaks down as reaction probabilities become dependent on reactor operating conditions (reactivity feedbacks) and changes in isotopic composition (burnup).

[1] With the assumption that cross sections are independent of flux.

Transport theory revisited: deterministic approach

Consequently, deterministic transport methods generally rely on at least three fundamental approximations:

1) The geometry is discretized into a number of homogeneous material regions
2) The continuous energy dependence of cross sections is condensed into a number of discrete energy groups
3) The angular dependence of double-differential scattering cross sections is represented by functional expansions, the directional dependence of flux is represented by functional expansions or discrete directions

The first two approximations are common to all deterministic transport methods, the treatment of angular dependence is what differentiates the methods from each other (S_n, P_n, method of characteristics, diffusion theory, etc.).

But it should be noted that, in the end, the flux itself is not a physical measurable quantity,[2] but rather the mathematical means to calculate physical reaction rates:

$$R_x = \int_V \int_{\hat{\boldsymbol{\Omega}}} \int_E \Sigma_x(\boldsymbol{r}, E)\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E) dV d\hat{\boldsymbol{\Omega}} dE \tag{2}$$

[2] This interpretation is subject to argument, but what cannot be argued is that flux can only be measured via physical reaction rates.

Transport theory revisited: Monte Carlo approach

The Monte Carlo approach to particle transport problems is fundamentally different – instead of dealing with the entire neutron population (flux), the transport process is viewed from the perspective of an *individual neutron*.

The procedure is based on a simulated random walk, in which the probability of each random event depends only on the position (material located at the collision point) and the energy of the transported neutron. Interactions are described by probability distributions, representing the “laws of physics”, from which the outcome is randomly sampled.

When the simulation is repeated for a very large number of neutron histories, various results describing the physical behavior of the system can be obtained by counting interactions, track lengths, boundary crossings, etc. and applying statistical methods on the collected results.

It is important to note that the Monte Carlo method does not solve the transport equation, but rather provides statistical estimators for integrals of the form:

$$\int_V \int_{\hat{\boldsymbol{\Omega}}} \int_E f(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E)\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E) dV d\hat{\boldsymbol{\Omega}} dE \tag{3}$$

where f can be an arbitrary response function depending on neutron coordinates in the six-dimensional phase space, most typically a cross section.

Transport theory revisited: Monte Carlo approach

Monte Carlo simulation has several fundamental advantages over the deterministic approach:

- Events occur at discrete locations, no need to integrate flux over space to obtain solution for the transport equation – complex geometries can be modeled at an arbitrary resolution
- Collisions occur at discrete energies, no need to integrate flux over energy to obtain solution for the transport equation – interaction data can be handled in continuous-energy form without group condensation
- Time dependence of reaction chains can be modeled explicitly without additional effort

But there are also a few fundamental disadvantages:

- Result estimates are based summation over simulated events, and they represent integrals over space, direction and energy (and time) – differential quantities are difficult to calculate
- All results are random variables, associated with a statistical uncertainty – accuracy of the simulation depends on the number of simulated neutron histories

The fact that interactions can be handled as independent events is based on the same linearity assumption that makes cross sections independent of flux in the formulation of the transport equation. Coupled simulations with the Monte Carlo method require similar iteration between different solvers.

Monte Carlo method: general

Computations based on random sampling date back to the 18th century,[3] but the theoretical basis (Markov's chains) was not formulated until the early 20th century. The method became practical along with the development of first electronic computers in the late 1940's, capable of producing random numbers and processing data in an automated manner.

The Monte Carlo method is typically used for solving complicated linear problems that can be divided into multiple simple and separate sub-tasks. Transport calculation is actually a very good example, since solving the particle flux (transport equation) is a complicated task, but simulating individual particle histories is relatively straightforward.

In practice, Monte Carlo particle transport simulation requires:

1) Geometry routine (simple linear algebra and vector calculus)
2) Physics model, handling the individual interactions with target nuclides (relatively simple nuclear physics and a lot of data)
3) Collecting the results (simple statistics)

The first Monte Carlo applications involved radiation transport problems in the aftermath of the Manhattan Project in the late 1940's. The method has been developed along with computing power, especially during the early years, and computer capacity is still one of the major factors limiting the scale of Monte Carlo simulations.

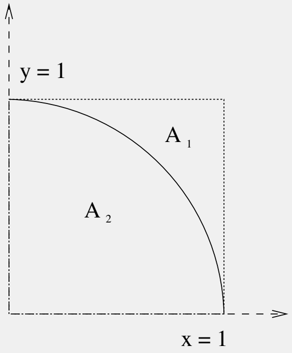
[3] See, for example, Buffon's needle problem for calculating the value of π

Monte Carlo method: simple example

Example 1: Monte Carlo estimate for π

Consider the quarter of a unit circle:

- Randomly sample M points between $(0 < x < 1, 0 < y < 1)$
- Count the number of points falling inside the circle (i.e. $x^2 + y^2 < 1$): m
- Based on the geometry it is known that the probability of a point falling inside the circle is the ratio of areas: $P = A_2/A_1 = (\pi 1^2/4)/(1^2) = \pi/4$
- From the simulation: $m/M \longrightarrow P$ when $M \longrightarrow \infty$
- Or: $4m/M \longrightarrow \pi$



Matlab example: `piex1.m`

Statistical mean and standard deviation

In the previous example the ratio of successful samples to total approached value $\pi/4$, but each repetition produced a slightly different, random result. The larger the number of samples, the closer the values were to the actual result.

From basic courses of probability and statistics it is recalled that the sample mean is given by:

$$\overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{4}$$

and the associated standard deviation:[4]

$$\sigma(\overline{x}) = \sqrt{\frac{1}{N(N-1)}\sum_{n=1}^{N}(x_n - \overline{x})^2} \tag{5}$$

or in a more practical form:

$$\sigma(\overline{x}) = \sqrt{\frac{1}{N(N-1)}\left[\sum_{n=1}^{N} x_n^2 - \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right)^2\right]} \tag{6}$$

The standard deviation is also associated to the variance, σ^2.

[4] Assuming that values x_n are not correlated.

Central limit theorem

The standard deviation σ is a measure of how much a value, randomly selected from distribution $f(x)$, differs (on the average) from the mean $\overline{x}$. This information has very little practical significance if the type of distribution $f(x)$ is unknown.[5]

Luckily, the central limit theorem states that:

> *The sum of arbitrarily distributed random variables is itself a random variable. Under certain conditions, the distribution of this variable approaches the normal distribution, as the sum of terms approaches infinity.*

The main conditions are:

1) The number of terms is sufficiently large
2) All terms are similarly distributed
3) The values are independent of each other (no correlations)

These conditions are easily met in Monte Carlo simulations, although the correlation between batches (violation of condition 3) may become a problem in criticality source simulations, as discussed later on.

[5] For example, for uniform distribution: $f(x) = \frac{1}{b-a}$ and $\sigma = \frac{b-a}{\sqrt{12}}$, but for exponential distribution: $f(x) = \lambda e^{-\lambda x}$ and $\sigma = \frac{1}{\lambda}$.

Confidence intervals

The significance of central limit theorem is that, since the type of the distribution is known for the simulated results, standard deviation can be associated with the more quantitative concept of confidence interval, which gives the range of values between a minimum and maximum positioned symmetrically about the mean.

For the normal distribution:

- 68% confidence interval: $\overline{x} \pm \sigma$
- 95% confidence interval: $\overline{x} \pm 1.96\sigma$
- 99% confidence interval: $\overline{x} \pm 2.58\sigma$

It is important to note that these confidence intervals apply only for the normal distribution.

The confidence intervals of Monte Carlo simulation are usually formed by assuming that the results follow the normal distribution, i.e. that the conditions of the central limit theorem are met. When this is not the case, statistical precision is either under- or over-estimated.[6]

[6] Correlations between values that should be independent also leads to the under-estimation of the standard deviation itself.

Confidence intervals

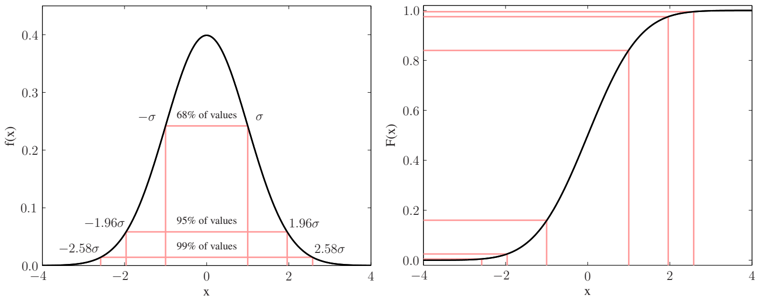


Figure 1: The (0,1) normal distribution with 68%, 95% and 99% confidence intervals. Left: probability density function (PDF), Right: cumulative distribution function (CDF). The PDF gives the probability of a value on interval $[x, x + dx]$. The CDF is obtained by integrating the distribution from $-\infty$ to x.

Monte Carlo method: simple example

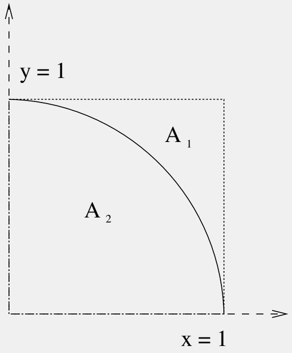
Example 2: Monte Carlo estimate for π with statistical errors

Consider the same quarter of a unit circle from the previous example, but instead of calculating the estimate directly, the problem is divided in two parts:

1) Randomly sample M points between $(0 < x < 1, 0 < y < 1)$, and calculate estimate $x_i = 4m/M$, where m is the number of points falling inside the circle
2) Repeat the procedure N times to obtain estimates $x_1, x_2, \ldots x_N$

Based on the central limit theorem, the values follow the normal distribution, which allows applying the known confidence intervals.

Matlab example: `piex2.m`



Monte Carlo method: simple example

Example 2: Monte Carlo estimate for π with statistical errors

Example results (M = 100):[a]

$N = 10$	$\pi \approx 3.20800 \pm 0.05053$	$[3.10897, 3.30703]$
$N = 100$	$\pi \approx 3.15320 \pm 0.01581$	$[3.12222, 3.18418]$
$N = 1000$	$\pi \approx 3.14476 \pm 0.00544$	$[3.13409, 3.15543]$
$N = 10000$	$\pi \approx 3.13973 \pm 0.00165$	$[3.13649, 3.14297]$
$N = 100000$	$\pi \approx 3.14141 \pm 0.00052$	$[3.14040, 3.14243]$
$N = 1000000$	$\pi \approx 3.14130 \pm 0.00016$	$[3.14098, 3.14162]$

[a] The value of π with five decimals is 3.14159

Figure-of-merit

The standard deviation in the previous example depends on:

1) The variation of values $x_1, x_2, \ldots x_N$, which depends on the number of points M sampled in the geometry
2) The number of repetitions N

If the number of sampled points M is kept constant, it is observed that:

1) The standard deviation falls as $\sigma \sim 1/\sqrt{N}$ (or variance as $\sigma^2 \sim 1/N$)
2) The computing time increases as $T \sim N$

This implies that product $\sigma^2 T$ is approximately constant regardless of N, which allows the definition of figure-of-merit:

$$FOM = \frac{1}{\sigma^2 T} \tag{7}$$

Since two algorithms performing the same task can have different running times and convergence rates,[7] the FOM provides a useful quantitative measure of computational efficiency. Since standard deviation depends the magnitude of the result, FOM is often calculated from the relative statistical error instead

[7] In Monte Carlo transport simulations it is very common that variance reduction techniques lead to both faster convergence and longer simulation time, so looking at the standard deviation alone does not give a fair estimate for the performance of the algorithm.

Notes on statistical precision

It is important to note that standard deviation or confidence intervals do not measure the accuracy of the simulation. The correct interpretation is rather related to statistical precisions, for example:

- The 95% confidence interval means that a random value selected from the distribution falls on interval $[\overline{x} - 1.96\sigma, \overline{x} + 1.96\sigma]$ with 95% probability
- The probability that the value is outside the range is 5%

An alternative interpretation is that if N repetitions gives a result with mean value $\overline{x}$ and standard deviation σ, the result obtained with a very large (infinite) number of repetitions falls on interval $[\overline{x} - 1.96\sigma, \overline{x} + 1.96\sigma]$ with 95% probability.

The results are also subject to systematic error resulting from approximations, errors and uncertainties in data, bugs in computer code, etc. The outcome of the simulation is only as good as the model: garbage in – garbage out.

Probability distributions

The laws of physics in Monte Carlo simulation are essentially represented by probability distributions, describing the outcome of random events. The *probability density function* (PDF), $f(x)$, is defined in such way that the probability of the event occurring between x and $x + dx$ is given by:

$$dP = f(x)dx \tag{8}$$

Consequently, the probability of the event occurring on interval $[x_0, x_1]$ of the variable is:

$$P(x_0 < x < x_1) = \int_{x_0}^{x_1} dP = \int_{x_0}^{x_1} f(x)dx \tag{9}$$

The probability that the event occurs before the variable reaches a certain value is given by the *cumulative distribution function* (CDF), $F(x)$, which is calculated from (8) by direct integration:

$$F(x) = P(x' < x) = \int_{-\infty}^{x} f(x')dx' \tag{10}$$

The PDF's are assumed to be properly normalized, i.e. the integration of $f(x)$ over all value space must yield $P = 1$:

$$\lim_{x \to \infty} F(x) = 1 \tag{11}$$

Sampling from probability distributions: inversion method

Probability distributions in Monte Carlo simulations are used for sampling the outcome of various events: distance to collision site, interaction after collision, direction and energy of scattered neutron, etc.

The simplest algorithm for sampling random values from distribution $f(x)$ is called the inversion method, which is based on the inverse of the CDF, $F^{-1}(x)$, and a uniformly distributed random variable ξ on the unit interval.[8]

The algorithm can be described as follows:

1) Calculate $F(x)$ and $F^{-1}(x)$ from $f(x)$
2) Sample a uniformly distributed random variable ξ on interval $[0, 1)$
3) Set $F(x) = \xi$
4) Value for variable x is obtained from the inverse of the CDF: $x = F^{-1}(\xi)$

Inversion sampling is an efficient way to produce samples from $f(x)$ if the distribution is provided in functional form and $F(x)$ and $F^{-1}(x)$ can be resolved.

This method is revisited when sampling neutron path lengths in homogeneous medium.

[8] The production of uniformly distributed (pseudo-)random numbers can be done with computational algorithms, but there is a surprising amount of science behind the methodology.

Sampling from probability distributions: inversion method

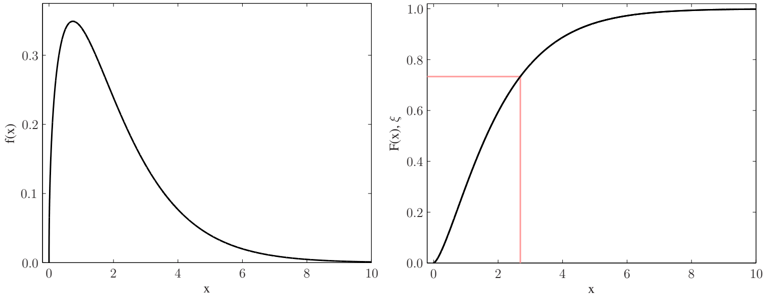


Figure 2: Sampling a random number from probability distribution using the inversion method. Left: Probability density function (PDF), $f(x)$, from which the value is sampled. Right: The corresponding cumulative distribution function (CDF), $F(x)$. A random number ξ is sampled on the unit interval, and the value is set $\xi = F(x)$. The distributed variable is obtained using the inverse function: $x = F^{-1}(\xi)$. In the example, ξ = 0.733, x = 2.69.

Sampling from probability distributions: rejection method

The inversion sampling method cannot be used if F or F^{-1} cannot be resolved in closed form. This is the case, for example, for the normal distribution and the Maxwell-Boltzmann distribution. In such case, one option is to obtain the value for F^{-1} numerically.

Another option is to use rejection sampling, which is based on two distribution functions, the original distribution $f(x)$ and a majorant function $g(x)$, which can be sampled using the inversion method, and for which:

$$g(x) \geq f(x) \tag{12}$$

for all values of x. The algorithm has two steps:

1) Sample random variable x from distribution $g(x)$ using the inversion method
2) Perform rejection sampling and accept the sample with probability $P(x) = f(x)/g(x)$

If the sample is rejected, the procedure is repeat from the beginning.

The method is best understood by considering the sampling of a uniform distribution of points inside a circle (see the previous example). Points are sampled uniformly on a square enveloping the circle, and accepted only if they fall inside the circle.

Another example is the Woodcock delta-tracking method used for sampling particle path lengths, discussed later on.

Sampling from probability distributions: tabular distributions

Energy and angular distributions of secondary neutrons are often given in tabular form, along with some interpolation rules for obtaining values between the tabulated points. In the case of linear-linear interpolation, the values of the PDF and the CDF are given by:

$$f(x) = \frac{x - x_{i-1}}{x_i - x_{i-1}}(f_i - f_{i-1}) + f_{i-1} \tag{13}$$

and

$$F(x) = \frac{x - x_{i-1}}{x_i - x_{i-1}}(F_i - F_{i-1}) + F_{i-1} \tag{14}$$

where i is the table index for which $x_{i-1} \leq x < x_i$, and f_{i-1}, f_i, F_{i-1} and F_i are the corresponding tabulated values of the distributions.

The sampling can be accomplished by selecting a uniformly distributed random number ξ on the unit interval, performing index search to obtain i such that $F_{i-1} \leq \xi < F_i$, and interpolating the value from:[9]

$$x = x_{i-1} + \frac{1}{a}\left(\sqrt{f_{i-1}^2 + 2a(\xi - F_{i-1})} - f_{i-1}\right) \tag{15}$$

where:

$$a = \frac{f_i - f_{i-1}}{x_i - x_{i-1}} . \tag{16}$$

[9] In practice, the procedure is not this simple, because the shape of the distribution may depend on neutron energy which requires additional interpolation between two distributions.

Geometry routine

The Monte Carlo simulation consists of a large number particle histories,[10] in which the random walk of an individual particle is followed, or tracked, through the geometry from its birth to eventual absorption or escape from the geometry.

From here on it is assumed that the tracking takes place in a three-dimensional Cartesian coordinate system. In vector notation, the position and direction of motion are defined by two vectors:

$$\boldsymbol{r} = x\hat{\boldsymbol{i}} + y\hat{\boldsymbol{j}} + z\hat{\boldsymbol{k}} \tag{17}$$

and

$$\hat{\boldsymbol{\Omega}} = u\hat{\boldsymbol{i}} + v\hat{\boldsymbol{j}} + w\hat{\boldsymbol{k}} \tag{18}$$

where $\hat{\boldsymbol{i}}$, $\hat{\boldsymbol{j}}$ and $\hat{\boldsymbol{k}}$ are the unit vectors defining the three-dimensional Cartesian coordinate system. Direction vector $\hat{\boldsymbol{\Omega}}$ is normalized to unity:

$$\hat{\boldsymbol{\Omega}} \cdot \hat{\boldsymbol{\Omega}} = 1 \tag{19}$$

or

$$u^2 + v^2 + w^2 = 1 \tag{20}$$

Coefficients u, v and w are the direction cosines, i.e. the cosines of the angle that vector $\hat{\boldsymbol{\Omega}}$ forms with the positive x-, y- and z-axis, respectively.

[10] The number of simulated histories is typically counted in millions, or in large-scale simulations, billions. This is several orders of magnitude less than the number of neutrons in an actual reactor, even at low power operation.

Geometry routine

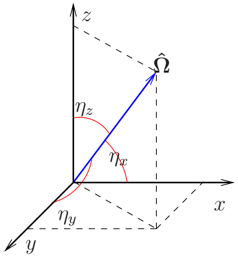


Figure 3: Direction vector $\hat{\mathbf{\Omega}}$ and direction cosines $u = \cos \eta_x$, $v = \cos \eta_y$ and $w = \cos \eta_z$ in the Cartesian coordinate system.

Geometry routine: surfaces

Monte Carlo transport codes are most typically based on the constructive solid geometry (CSG) type, in which the geometry is composed of homogeneous material cells, defined using combinations of elementary and derived surface types.

The most elemental building block is the surface, described using algebraic equations, typically of the quadratic type. The action that puts an arbitrary position $\boldsymbol{r}$ on one or the other side of a surface is based on a simple test carried out by substituting the coordinates into the surface equation:

$$S(\boldsymbol{r}) = S(x,y,z) \begin{cases} < 0 & \text{if the point is inside the surface} \\ = 0 & \text{if the point is on the surface} \\ > 0 & \text{if the point is outside the surface} \end{cases} \tag{21}$$

This surface test also fixes the concepts of “inside” and “outside” for each surface type, which is important when forming the cells from the surface combinations.

The general quadratic surface can be written in parametric form as:

$$S(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fzx + Gx + Hy + Iz + J \tag{22}$$

where $A, B, C, D, E, F, G, H, I$ and J are constants.[11]

[11] There are also non-quadratic surfaces, such as the torus.

Geometry routine: surfaces

Common examples of quadratic surfaces obtained from the parametrized quadratic equation include the plane perpendicular to x-axis at x_0:

$$S(x) = x - x_0 \tag{23}$$

sphere centered at (x_0, y_0, z_0) with radius r:

$$S(x,y,z) = (x - x_0)^2 + (y - y_0)^2 - (z - z_0)^2 - r^2 \tag{24}$$

and straight infinite cylinder parallel to z-axis centered at (x_0, y_0) with radius r:

$$S(x,y) = (x - x_0)^2 + (y - y_0)^2 - r^2 \tag{25}$$

Surface equation for general plane is obtained by dropping the second-order terms in Eq. (22):

$$S(x,y,z) = Gx + Hy + Iz + J \tag{26}$$

where G, H, I and J are constants. The plane can also be defined using three points: (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3), in which case the constants for Eq. (26) become:

$$\begin{aligned} G &= y_2 z_3 - y_3 z_2 - y_1(z_3 - z_2) + z_1(y_3 - y_2) \\ H &= z_2 x_3 - z_3 x_2 - z_1(x_3 - x_2) + x_1(z_3 - z_2) \\ I &= x_2 y_3 - x_3 y_2 - x_1(y_3 - y_2) + y_1(x_3 - x_2) \\ J &= -x_1(y_2 z_3 - y_3 z_2) + y_1(x_2 z_3 - x_3 z_2) - z_1(x_2 y_3 - x_3 y_2) \end{aligned} \tag{27}$$

The three points determine the inward surface direction according to the right-hand rule.

Geometry routine: surfaces

Surface equations are also used for determining the distance to the nearest material boundary in the direction of motion. The points where the particle path intersects the surface are obtained by solving Eq. (21) with condition:

$$S(\boldsymbol{r} + \delta\hat{\boldsymbol{\Omega}}) = S(x + \delta u, y + \delta v, z + \delta w) = 0 \tag{28}$$

i.e. by setting a point located at distance δ from position $\boldsymbol{r}$ in the direction of motion $\hat{\boldsymbol{\Omega}}$ on the surface, and solving for δ. When the equation has multiple solutions, the nearest point corresponds to the smallest positive value of δ. If all solutions are negative or no solution exists, the surface is away from the line-of-sight.

The distance to general quadratic surface (22) can be written as:

$$\delta = \frac{-L \pm \sqrt{L^2 - 4MK}}{2M} \tag{29}$$

where:

$$\begin{aligned}
K &= Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fxz + Gx + Hy + Iz + J \\
L &= 2(Aux + Bvy + Cwz) + D(vx + uy) + E(wy + vz) + F(wx + uz) \\
&\quad + Gu + Hv + Iw \\
M &= Au^2 + Bv^2 + Cw^2 + Duv + Evw + Fuw
\end{aligned}$$

Geometry routine: surfaces

For example, the distance to a plane perpendicular to x-axis at x_0 is obtained from

$$x + \delta u - x_0 = 0 \Longleftrightarrow \delta = \frac{x_0 - x}{u} \tag{30}$$

and the distance to a cylinder parallel to z-axis centered at (x_0, y_0) with radius r from:

$$\delta = \frac{-L \pm \sqrt{L^2 - 4MK}}{2M} \tag{31}$$

where:

$$\begin{aligned} K &= x^2 + y^2 - 2(x_0 x + y_0 y) x_0^2 - y_0^2 - r^2 \\ L &= 2(ux + vy) - 2(x_0 u + y_0 v) \\ M &= u^2 + v^2 \end{aligned}$$

Distance to general plane (26) is given by:

$$\delta = -\frac{Gx + Hy + Iz + J}{Gu + Hv + Iw} \tag{32}$$

which corresponds to the general quadratic equation (29) with coefficients of second order terms set to zero.

Geometry routine: surfaces

Monte Carlo codes often provide additional derived surface types, which are formed by combinations of elementary surfaces. For example, a cuboid with boundaries $[x_1, x_2]$, $[y_1, y_2]$, $[z_1, z_2]$ consists of six planes perpendicular to the coordinate axes:

$$S := \begin{cases} S_1(x) = x - x_1 \\ S_2(x) = x - x_2 \\ S_3(y) = y - y_1 \\ S_4(y) = y - y_2 \\ S_5(z) = z - z_1 \\ S_6(z) = z - z_2 \end{cases} \tag{33}$$

Other derived surface types convenient for reactor modeling include truncated cylinders, and square and hexagonal prisms.

The surface test and calculation of distance to nearest intersection point for derived surfaces involve multiple solutions to Eqs. (21) and (28). From the algorithmic point of view these operations should be seen as functions that return the result regardless of the surface type (elementary or derived).

Geometry routine: surfaces

It should be noted, however, that in case of derived surface types, a positive distance returned by (28) may not correspond to intersection with the actual surface, but rather the continuation of one of its constituents.

These “false” crossings can be accounted for by additional checks in the distance function or writing the tracking routine in such way that they can be ignored altogether.

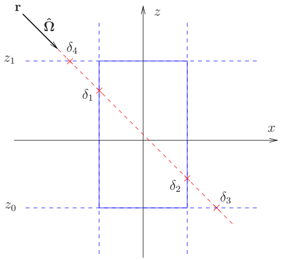


Figure 4: Particle path intersecting a truncated cylinder in the xz-plane demonstrating the false crossings. All surface distances in this example are positive and $\delta_4 < \delta_1 < \delta_2 < \delta_3$. Points 4 and 3 are not part of the boundary. The shortest actual distance is δ_1, even though the routine may return δ_4.

Geometry routine: surfaces

Third application for surface equations is the calculation of normal vectors, which are needed for various purposes, including:

- Calculating the surface flux estimator
- Applying reflective boundary conditions

The normal vector is given by the gradient of the surface equation:

$$\hat{\boldsymbol{n}}(\boldsymbol{r}) = \frac{\nabla S(\boldsymbol{r})}{|\nabla S(\boldsymbol{r})|} \,, \tag{34}$$

where

$$\nabla S(\boldsymbol{r}) = \nabla S(x,y,z) = \left(\frac{\partial S}{\partial x}\right)\hat{\boldsymbol{i}} + \left(\frac{\partial S}{\partial y}\right)\hat{\boldsymbol{j}} + \left(\frac{\partial S}{\partial z}\right)\hat{\boldsymbol{k}} \tag{35}$$

and

$$|\nabla S(\boldsymbol{r})| = |\nabla S(x,y,z)| = \sqrt{\left(\frac{\partial S}{\partial x}\right)^2 + \left(\frac{\partial S}{\partial y}\right)^2 + \left(\frac{\partial S}{\partial z}\right)^2} \tag{36}$$

For example, the normal vector on an infinite cylinder parallel to z-axis centered at (x_0, y_0) with radius r is given by:

$$S(x,y) = (x-x_0)^2 + (y-y_0)^2 - r^2 \Longrightarrow \hat{\boldsymbol{n}}(x,y) = \frac{(x-x_0)\hat{\boldsymbol{i}} + (y-y_0)\hat{\boldsymbol{j}}}{\sqrt{(x-x_0)^2 + (y-y_0)^2}} \tag{37}$$

Geometry routine: cells

The next building block in the CSG hierarchy is the cell, which is constructed from the combination of surfaces, and it defines a homogeneous material region. The construction is based on three operators:

Intersection: $S_1 \ \cap \ S_2$ – Point is inside the cell if it is inside *both* surface S_1 and S_2

Union: $S_1 \ \cup \ S_2$ – Point is inside the cell if it is inside *either* surface S_1 or S_2, or both

Complement: $\backslash S_1$ – Point is inside the cell if it is outside surface S_1

The intersection and union operator behave very similar to arithmetic multiplication and addition, respectively, and they share the properties of commutativity (order of operands is exchangeable):

$$\begin{aligned} S_1 \ \cup \ S_2 &= S_2 \ \cup \ S_1 \\ S_1 \ \cap \ S_2 &= S_2 \ \cap \ S_1 \end{aligned} \tag{38}$$

associativity (two or more similar operations can be grouped in an arbitrary manner):

$$\begin{aligned} (S_1 \ \cup \ S_2) \ \cup \ S_3 &= S_1 \ \cup \ (S_2 \ \cup \ S_3) \\ (S_1 \ \cap \ S_2) \ \cap \ S_3 &= S_1 \ \cap \ (S_2 \ \cap \ S_3) \end{aligned} \tag{39}$$

and distributivity (precedence of intersection over union):

$$(S_1 \ \cup \ S_2) \ \cap \ S_3 = (S_1 \ \cap \ S_3) \ \cup \ (S_2 \ \cap \ S_3) \tag{40}$$

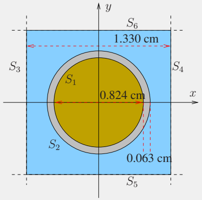
The use of the operators is best illustrated by an example.

Geometry routine: cells

Example 3: Cell definition in CSG geometry

Consider a 2D pin-cell model of a light water reactor fuel pin surrounded by coolant. The pin consist of a fuel pellet with an outer diameter of 0.824 cm, enclosed inside a 0.063 cm thick cladding. The square pitch of the unit cell is 1.330 cm. The surfaces are defined as:

$$\begin{aligned}
S_1(x,y,z) &= x^2 + y^2 - 0.412^2 \\
S_2(x,y,z) &= x^2 + y^2 - 0.475^2 \\
S_3(x,y,z) &= x + 0.665 \\
S_4(x,y,z) &= x - 0.665 \\
S_5(x,y,z) &= y + 0.665 \\
S_6(x,y,z) &= y - 0.665
\end{aligned} \tag{41}$$



These surfaces are used to define four cells using intersections, unions and complements:

$$\begin{array}{llll}
C_1 & : & S_1 & : \text{Fuel} \\
C_2 & : & (\backslash S_1) \cap S_2 & : \text{Cladding} \\
C_3 & : & (\backslash S_2) \cap (\backslash S_3) \cap S_4 \cap (\backslash S_5) \cap S_6 & : \text{Coolant} \\
C_4 & : & S_3 \cup (\backslash S_4) \cup S_5 \cup (\backslash S_6) & : \text{Outside world}
\end{array} \tag{42}$$

The last cell, described as the “outside world”, is not a part of the actual geometry, but it needs to be defined in order to tell the geometry routine that the particle has escaped the system, or that boundary conditions need to be invoked.

Geometry routine: cells

Testing cells that consist of only intersections and complements is straightforward:

- If all surfaces comprising the cell pass the surface test (taking into account the complement operators), the point is inside the cell
- If any of the surfaces fails the test, the point is outside the cell

The complement operator can be included by adding flags in the surface intersection list.

The problem with unions is that testing arbitrarily constructed cells with one or several unions becomes a non-trivial task, requiring either a tree-based or post-fix algorithm.

It is possible to construct geometries, even complicated ones, without using the union operator. This can be accomplished using derived surface types (square and hexagonal prisms, cubes, cuboids, etc.). In practice this means that any union operators are performed inside the surface functions instead of within the cell test algorithm.[12]

Most geometries consist of not a single, but of multiple cells, and in order to determine which cell fills the space at an arbitrary position, the cell search routine must loop over all possibilities until the conditions for the constituent surfaces are matched.

[12] Consider a cube comprised of six surfaces: $(\backslash S_1) \cap S_2 \cap (\backslash S_3) \cap S_4 \cap (\backslash S_5) \cap S_6$. The most obvious way to define the outside space is using unions: $S_1 \cup (\backslash S_2) \cup S_3 \cup (\backslash S_4) \cup S_5 \cup (\backslash S_6)$. Another option is to use a single derived surface type, which actually means that the unions are handled internally, as part of the surface test.

Geometry routine: cells

Algorithm 1 Cell search with intersections only

```
1:  for $m \leftarrow 1$ to $M$ do                                  ▷ Loop over candidate cells
2:      for $n \leftarrow 1$ to $N_m$ do                             ▷ Loop over intersection list
3:          $c \leftarrow S_n(x, y, z)$                              ▷ Call surface test routine
4:          if $S_n(x, y, z)$ is flagged with '\' then               ▷ Check complement flag
5:              $c \leftarrow \backslash c$                          ▷ Apply complement operator
6:          end if
7:          if $c = \mathrm{FALSE}$ then                             ▷ Check condition
8:              Break loop                                           ▷ Point is outside, test next cell
9:          end if
10:     end for
11:     if $n = N_m$ then                                            ▷ Check if all surfaces passed the test
12:         return $m$                                               ▷ Point is inside cell $m$
13:     end if
14: end for
15: return $\mathrm{ERROR}$                                          ▷ Geometry error: no cell at $(x, y, z)$
```

Geometry routine: universes and lattices

Handling of complex geometries can be considerably simplified by dividing the model into multiple levels. In reactor geometries the natural division is:

1) Core-level
2) Assembly-level
3) Pin-level

The reactor core consists of a regular array of fuel assemblies, many of which are identical and interchangeable. The same applies to fuel pins inside assemblies.

Instead of modeling each pin and assembly explicitly, they can be described as separate objects in their own universe. The same object can then be copied into multiple locations by filling cells with an entire universe.

The use of lattices helps the construction of regular structures, such as the pin layout in fuel assemblies or the loading pattern of assemblies in the reactor core. The geometry routine then automatically makes a coordinate transformation between the levels and centers the universe with respect to the lattice cell.

Structuring the geometry into multiple levels not only simplifies the input model, but may also lead to considerable increase in performance as the number of computationally expensive geometry operations is reduced.

Geometry routine: universes and lattices

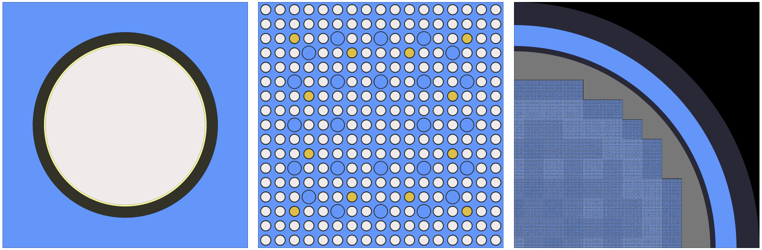


Figure 5: Pin-, assembly- and core-level geometries in a 1/4 core model of a PWR. Left: The first level is comprised of universes describing individual fuel pins, control rod guide tubes, etc. Center: The second level describes fuel assemblies as regular lattices of pins. Right: The last level is comprised of the reactor vessel, with an assembly lattice inside it. When the cell search routine finds itself inside a cell filled with an entire universe, it makes a coordinate transformation to the next level, and recursively locates the material cell at the position.

Geometry routine: universes and lattices

Most common lattice types are 2D square array and hexagonal “honeycomb”, but the concept can basically be extended to any regular (or even irregular) structure.

When the point is found inside a lattice, the geometry routine automatically makes a coordinate transformation between the levels and centers the universe with respect to the lattice cell.

This requires calculating the lattice indexes, which for a Cartesian type is written as:[13]

$$i = \text{floor}\left\{\frac{x}{p_x}\right\} \quad j = \text{floor}\left\{\frac{y}{p_y}\right\} \quad k = \text{floor}\left\{\frac{z}{p_z}\right\} \tag{43}$$

where i, j and k are the lattice indexes and p_x, p_y and p_z are the cell widths (pitches). The “floor” operation refers to truncation of the decimal part.

The universe filling the lattice cell is found from an input table using the lattice indexes. The coordinate transformation to the new origin is written as:

$$\begin{aligned} x' &= x - ip_x \\ y' &= y - jp_y \\ z' &= z - kp_z \end{aligned} \tag{44}$$

[13] It is assumed here that the lattice is centred at origin and has an odd number of cells in each direction.

Geometry routine: universes and lattices

Algorithm 2 Cell search with multiple universes and lattices

```
 1: $u \leftarrow u_0$                                      ▷ Start from root universe
 2: while $\mathrm{TRUE}$ do                                ▷ Loop over universes
 3:     $m \leftarrow C(u, \boldsymbol{r})$                 ▷ Call cell search routine for universe $u$ (algorithm 1)
 4:     if cell $m$ is a material cell then
 5:         return $m$                                      ▷ Point is inside cell $m$
 6:     else if cell $m$ is filled with universe $u'$ then
 7:         $u \leftarrow u'$                               ▷ Update universe
 8:     else if cell $m$ is filled with lattice $l$ then
 9:         $(u', \boldsymbol{r}_0) \leftarrow L(l, \boldsymbol{r})$   ▷ Get lattice cell universe and origin
10:         $\boldsymbol{r} \leftarrow \boldsymbol{r} - \boldsymbol{r}_0$ ▷ Update coordinates
11:         $u \leftarrow u'$                               ▷ Update universe
12:     else
13:         return $m_o$                                    ▷ Point is outside the geometry
14:     end if
15: end while
```

Geometry routine: boundary conditions

The geometry model is limited to an outer boundary, beyond which there are no more cells. What happens to a neutron crossing this boundary depends on the boundary conditions:

- Vacuum (or black) boundary condition – the neutron has escaped the geometry and the history is terminated
- Reflective (or mirror) boundary condition – the neutron is reflected back into the geometry at an angle symmetrical with respect to the surface normal
- Periodic boundary condition – the neutron is moved into a symmetry position on the other side of the geometry
- White boundary condition – the neutron is reflected back into the geometry isotropically (random direction)

Reflective and periodic boundary conditions can be used create infinite geometries in which the same structure is repeated over and over again. If the outer boundary is a square or hexagonal prism, the boundary conditions can also be invoked by a coordinate transformation.

With vacuum boundary conditions it is important that the geometry is non re-entrant, as the neutron is killed immediately after crossing the boundary without the possibility of returning to the geometry.

Geometry routine: boundary conditions

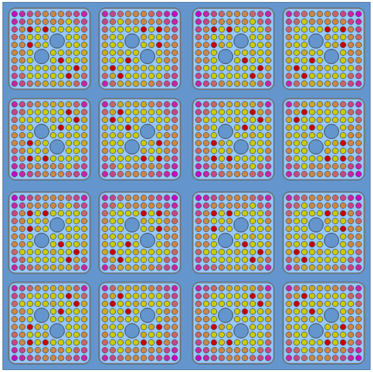
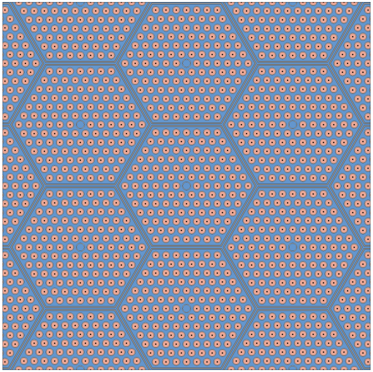



Figure 6: Infinite lattices of identical fuel assemblies generated using repeated boundary conditions. Left: BWR fuel lattice, with reflective boundary surrounding a single assembly. Right: VVER-440 fuel lattice with periodic boundary surrounding a single assembly. Animated neutron tracks illustrating the boundary conditions are plotted in `Lecture3_anim1.gif` (vacuum), `Lecture3_anim2.gif` (reflective), `Lecture3_anim3.gif` (periodic).

Simulated random walk

The simulated random walk proceeds from one interaction to the next, following a very simple procedure:

1) Sample path length (distance to next collision)
2) Transport particle to the collision point
3) Sample interaction

If the sampled interaction is scattering, the procedure restarts from beginning by sampling the distance to the next collision. The direction and energy are changed in the scattering event.

If the sampled interaction is fission, a number of new neutrons are produced with energy and direction sampled from the associated distributions.

The fact that the particle may cross the boundary between two material regions means that the interaction probability changes along the sampled path, i.e. the sample is not valid beyond the material boundary. There are two options:

1) Stop the track at the boundary crossing and sample a new path length according to the new interaction probability (surface-tracking)
2) Account for the changing probability by rejection sampling applied to each tentative collision site (delta-tracking)

Both options are introduced after deriving the equation for the sampled path length.

Simulated random walk: sampling the path length

By definition, the interaction probability per traveled path length is given by the macroscopic total cross section (denoted here as Σ). If it is assumed that the neutron travels through an infinite homogeneous medium characterized by constant total cross section, the probability that the next collision will occur within distance dl from the current position is:

$$dP = \Sigma dl \tag{45}$$

Let $P_0(l)$ be the probability that the neutron has reached position l without any interaction. When the neutron moves forward by distance dl from l, the reduction in $P_0(l)$ is equal to the conditional probability that the neutron will interact within the interval:

$$dP_0 = -P_0(l)dP = -P_0(l)\Sigma dl \tag{46}$$

The solution of this differential equation yields for the *non-interaction probability*:

$$P_0(l) = e^{-l\Sigma} \tag{47}$$

Using this result, the conditional probability that the neutron first moves distance l without interactions and then has its first interaction within the next dl is given by:

$$P_0(l)dP = P_0(l)\Sigma dl = \Sigma e^{-l\Sigma} dl \tag{48}$$

Simulated random walk: sampling the path length

The solution to (48) gives the probability density function (PDF) of the distance to the next collision:

$$f(l) = \Sigma e^{-l\Sigma} \tag{49}$$

from which the cumulative distribution function (CDF) is given by integration:

$$F(l) = \int_0^l \Sigma e^{-l'\Sigma} dl' = 1 - e^{-l\Sigma} \tag{50}$$

The probability distribution is exponential and the neutron distance to the next collision site can be sampled using the inversion method. The inverse of the CDF in (50) is

$$F^{-1}(\xi) = -\frac{1}{\Sigma}\log(1-\xi) \tag{51}$$

Since ξ is a uniformly distributed random number on the unit interval, so is $1-\xi$, and the collision distance can be sampled from:

$$l = -\frac{1}{\Sigma}\log\xi \tag{52}$$

It is important to note that the prerequisite of using (52) for sampling the distance to the next collision site is that the material is <u>infinite and homogeneous</u>. If this is not the case, the second equality in (50) does not hold, and path lengths sampled from (52) are not statistically valid.

Simulated random walk: sampling the path length

The most common case when the previous condition is violated is when the neutron crosses a boundary between two material regions.[14] In such case, the PDF of free path length is a piece-wise continuous function, which can be written in two parts, taking into account the conditional probability that the neutron reaches the boundary crossing at δ:

$$f(l) = \begin{cases} \Sigma_1 e^{-\Sigma_1 l} & \text{when } l \leq \delta \\ e^{-\Sigma_1 \delta} \Sigma_2 e^{-\Sigma_2 (l-\delta)} & \text{when } l > \delta \end{cases} \tag{53}$$

This approach becomes somewhat impractical in the general case, when the neutron crosses not one but several material boundaries and passes through multiple regions with different interaction probabilities.

A better option is to take advantage of the fact that the interaction probability within the next dl is independent of the distance l traveled so far. This essentially means that any point within the neutron's path can be considered a starting point of a new sample.

[14] The condition is also violated when the material is inhomogeneous or the microscopic cross sections are not constant. This is the case, for example, when the neutron travels through boiling coolant or a fuel pin with a steep temperature gradient. There are techniques to account for the continuous changes in material properties, but the conventional approach is to discretize the distribution into homogeneous sub-regions.

Simulated random walk: surface-tracking algorithm

If it is known that the neutron makes it to the next material boundary and interacts somewhere beyond the other side, the point of crossing can be taken as the starting point of a new path. This is the general idea in the surface-tracking algorithm, in which the neutron track is stopped at each boundary crossing, and a new path sampled using the cross section of the next material.

The algorithm requires calculating the distance to the nearest boundary surface. The only way to accomplish this is to loop over all candidate surfaces and pick the shortest value. The routine also needs to perform the cell test to obtain the material located on the other side of the boundary crossing.[15]

Surface tracking is considered the standard tracking algorithm and it is used by virtually every Monte Carlo neutron transport code. The method has a few drawbacks related to its efficiency in complex geometries:

- Determining the distance to the nearest boundary can become computationally expensive if the cells are comprised of a large number of surfaces
- The fact that the neutron has to be stopped at each boundary crossing becomes a computational bottleneck when the neutron mean-free-path is long compared to the dimensions

[15]The routine can be optimized to some extent by testing only cells that share the same boundary surface.

Simulated random walk: surface-tracking algorithm

Algorithm 3 Surface-tracking algorithm

```
1:  for $j \leftarrow 1$ to $\infty$ do                                                  ▷ Loop until collision
2:      Get cross section $\Sigma_j$ at current position $\boldsymbol{r}_j$                  ▷ Call cell search routine
3:      Get distance $\delta$ from $\boldsymbol{r}_j$ to nearest boundary in $\hat{\boldsymbol{\Omega}}$   ▷ Call surface distance routine
4:      $l \leftarrow -\log(\xi)/\Sigma_j$                                              ▷ Sample distance to collision
5:      if $l < \delta$ then                                                       ▷ Check distance
6:          $\boldsymbol{r}_{j+1} \leftarrow \boldsymbol{r}_j + l\hat{\boldsymbol{\Omega}}$                     ▷ Move neutron to collision site
7:          Break loop                                                         ▷ Proceed to collision routine
8:      else
9:          $\boldsymbol{r}_{j+1} \leftarrow \boldsymbol{r}_j + (\delta + \epsilon)\hat{\boldsymbol{\Omega}}$          ▷ Move neutron over boundary crossing[16]
10:     end if
11: end for
```

[16] A small extrapolation distance ϵ is added to the surface distance to avoid problems with limited floating point precision and to ensure that the cell search routine puts the neutron on the correct side of the surface.

Simulated random walk: delta-tracking algorithm

An alternative to surface-tracking is the Woodcock delta-tracking algorithm, which is based on the rejection sampling of neutron path lengths. The procedure relies on the concept of a *virtual collision*, which is a fictive interaction that preserves the energy and direction of the neutron.

Since virtual collisions do not change the random walk in any way, the material total cross section Σ can be adjusted with an arbitrary virtual collision cross section Σ_0:

$$\Sigma'(\boldsymbol{r}, E) = \Sigma(\boldsymbol{r}, E) + \Sigma_0(\boldsymbol{r}, E) \tag{54}$$

without changing the outcome of the simulation. It is then possible to adjust the cross sections of all material regions in the system such that:

$$\Sigma_1'(E) = \Sigma_2'(E) = \Sigma_3'(E) \cdots = \Sigma_\mathrm{m}(E) \tag{55}$$

where Σ_m is called the *majorant* cross section.

In practice, it is not necessary to define the virtual collision cross sections at all if the majorant is simply taken as the maximum of all material totals at each energy point:

$$\Sigma_\mathrm{m}(E) = \max\left[\Sigma(\boldsymbol{r}, E)\right] \tag{56}$$

Unlike the physical total cross section, which depends on the material located at the neutron position, the majorant cross section is completely independent of the spatial coordinates.

Simulated random walk: delta-tracking algorithm

The point of having a macroscopic cross section that is uniform throughout the geometry is that when used for sampling path lengths:

$$l = -\log(\xi)/\Sigma_{\mathrm{m}} \tag{57}$$

the values are statistically valid regardless of the number of material boundaries crossed.

At the end point of the sampled path the tracking routine performs rejection sampling. The probability to accept the collision is given by ratio of the physical total cross section to the majorant:

$$P = \frac{\Sigma(\boldsymbol{r}, E)}{\Sigma_{\mathrm{m}}(E)} \tag{58}$$

If the collision is rejected, a new path length is sampled from (57) and the neutron is moved to the next collision site candidate.

Since the majorant cross section is always larger than or equal to the total cross section, the path lengths sampled in delta-tracking are, on the average, shorter than those sampled with surface-tracking. The average physical distance between two collisions is preserved, as some paths are extended over multiple virtual collisions.

Simulated random walk: delta-tracking algorithm

Algorithm 4 Delta-tracking algorithm

```
 1: Get majorant cross section $\Sigma_\mathrm{m}$
 2: **for** $j \leftarrow 1$ to $\infty$ **do**                                ▷ Loop until collision
 3:     $l \leftarrow -\log(\xi)/\Sigma_\mathrm{m}$                          ▷ Sample distance to collision
 4:     $\boldsymbol{r}_{j+1} \leftarrow \boldsymbol{r}_j + l\hat{\boldsymbol{\Omega}}$   ▷ Move neutron to tentative collision site
 5:     Get cross section $\Sigma_{j+1}$ at current position $\boldsymbol{r}_{j+1}$   ▷ Call cell search routine
 6:     **if** $\xi < \Sigma_{j+1}/\Sigma_\mathrm{m}$ **then**               ▷ Rejection sampling
 7:         Break loop                                                       ▷ Proceed to collision routine
 8:     **else**
 9:         Virtual collision                                                ▷ Collision point rejected
10:     **end if**
11: **end for**
```

Simulated random walk: delta-tracking algorithm

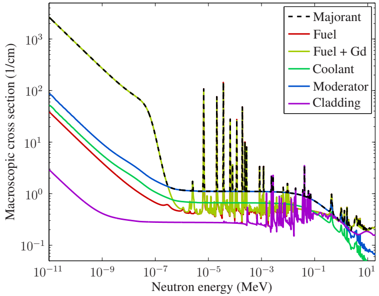
The advantage of delta-tracking over the surface-tracking algorithm is that there is no need to calculate the surface distances or stop the neutron at the material boundaries. This becomes significant for computational performance in geometries where the neutron mean-free-path is long compared to dimensions.[17]

Delta-tracking also has its drawbacks. Since the majorant cross section reflects the largest interaction probability within the system, the efficiency of the rejection sampling loop may become poor in the presence of localized heavy absorbers (control rods, burnable absorber pins, etc.) that dominate the majorant cross section, but occupy a relatively small volume in the geometry.

Another drawback is that delta-tracking rules out the use of the track-length estimate (TLE) of neutron flux, discussed later on, and reaction rate estimates need to be calculated using the potentially less efficient collision estimator (CFE).

[17]Since the majorant cross section used for sampling the path lengths does not depend on the spatial coordinates and the material total is needed only at discrete locations, variations of delta-tracking can be used for modeling inhomogeneous material compositions.

Simulated random walk: delta-tracking algorithm



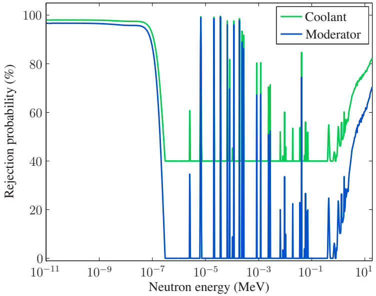


Figure 7: Left: Majorant and macroscopic cross sections in a system with localized heavy absorber (Gd-fuel pins in BWR assembly). The majorant is dominated by the high capture cross sections of ^{155}Gd and ^{157}Gd, even though the burnable absorber pins occupy a relatively small volume of the geometry. Right: Rejection probability in coolant and moderator where neutrons spend most of their lifetime. The efficiency of the rejection sampling scheme becomes poor especially at low energy.

Sampling the reaction mode

The physics routine in a Monte Carlo simulation is called when the particle has reached the collision site. The simulation proceeds to sampling the target nuclide. The probability of selecting nuclide n is given by:

$$P_n = \frac{\Sigma_n}{\Sigma} = \frac{N_n \sigma_{\mathrm{tot},n}}{\sum_i N_i \sigma_{\mathrm{tot},i}} \tag{59}$$

where N is the atomic density and σ is the microscopic cross section.

The selection is carried out by sampling a uniformly distributed random variable ξ on the unit interval and searching index n such that:

$$\sum_i^{n-1} \Sigma_i < \xi\Sigma \leq \sum_i^{n} \Sigma_i \tag{60}$$

Once the target nuclide is sampled, the reaction mode is selected in a similar way, using the microscopic cross sections. The probability of reaction mode x is given by:

$$P_x = \frac{\sigma_x}{\sigma_{\mathrm{tot}}} \tag{61}$$

In practice, the procedure is similar to (60).

Neutron interactions: capture

Neutron interactions can be roughly divided into three categories:

1) Capture
2) Fission
3) Scattering

Capture covers all reactions, in which the incident neutron is lost, and no secondary neutrons are emitted, for example:

- Radiative capture:
$$^{238}_{92}\mathrm{U} + ^{1}_{0}\mathrm{n} \longrightarrow ^{239}_{92}\mathrm{U} + \gamma \tag{62}$$

- Alpha emission:
$$^{10}_{5}\mathrm{B} + ^{1}_{0}\mathrm{n} \longrightarrow ^{7}_{3}\mathrm{Li} + ^{4}_{2}\mathrm{He} \tag{63}$$

- Proton emission:
$$^{3}_{2}\mathrm{He} + ^{1}_{0}\mathrm{n} \longrightarrow ^{3}_{1}\mathrm{H} + ^{1}_{1}\mathrm{H} \tag{64}$$

In analog Monte Carlo, capture terminates the neutron history. Capture can also be handled implicitly, by reducing the statistical weight of the neutron according to the capture probability.[18]

[18] This topic is revisited when discussing implicit Monte Carlo techniques.

Neutron interactions: fission

Fission terminates the history and a number of new neutrons are emitted. The typical way to sample the number of fission neutrons is to take the average fission neutron yield $\overline{\nu}$, truncate the value to the nearest integer n, and include one extra neutron if:

$$\xi < \overline{\nu} - n \tag{65}$$

The direction of emitted neutrons is sampled isotropically, and the energy distribution follows the Maxwell:

$$f(E) = C_0 \sqrt{E} e^{-E/T} \tag{66}$$

or Watt:

$$f(E) = C_0 e^{-E/a} \sinh \sqrt{bE} \tag{67}$$

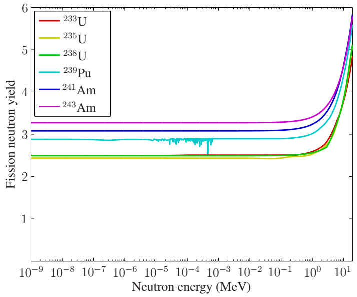
fission spectrum, where C_0 is a normalization constant and T, a and b depend on the fissioned nuclide and the incident neutron energy.[19] In most recent evaluated nuclear data files the fission spectra are given as tabular distributions.[20]

Fission neutrons are stored in a “bank”, which is a data structure that holds the position, direction, energy and time of each new particle. The transport routine retrieves particles from the bank, and moves on to the next source particle when the bank is empty.

[19] The dependence of emission energy on incident energy is not very strong at energies relevant for fission reactor applications. This dependence is usually ignored completely in deterministic calculations.

[20] Since prompt and delayed neutrons are produced in different processes, their spectra is also different. Delayed neutrons are born at considerably lower energy.

Neutron interactions: fission



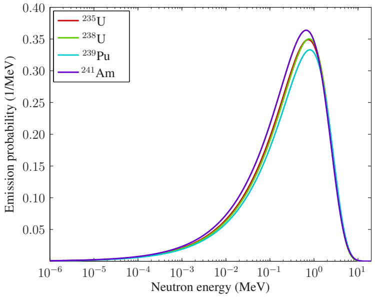


Figure 8: Left: Fission neutron yields of various actinides as function of neutron energy. The yield is actually a linear function of energy, which only appears to increase sharply in the MeV-range because of the logarithmic scale. The number of emitted neutrons increases along with actinide mass.
Right: The Maxwellian energy distributions of ^{235}U, ^{238}U, ^{239}Pu and ^{241}Am fission neutrons. The peak position varies slightly for different actinides, but is practically independent of neutron energy. The average energy for ^{235}U fission neutrons is around 2 MeV.

Neutron interactions: fission

If delayed neutron emission is included in the simulation, the selection between prompt and delayed neutrons is carried out based on the delayed and total nubars. Delayed neutron data is divided into 6 or 8 precursor groups,[21] and the probability of emitting a delayed neutron in group j is simply:

$$P = \frac{\overline{\nu}_{\mathrm{d},j}}{\overline{\nu}} \tag{68}$$

Delayed neutrons are produced in the radioactive decay chains of certain fission product isotopes, for example:

$$^{87}_{35}\mathrm{Br} \xrightarrow[55.7\mathrm{s}]{\beta^-} {}^{87}_{36}\mathrm{Kr}^* \longrightarrow {}^{86}_{36}\mathrm{Kr} + {}^{1}_{0}\mathrm{n} \tag{69}$$

The neutron release after the formation of an excited state of ^{87}Kr is practically instantaneous, and the delay between fission and neutron emission depends on the beta-decay of ^{87}Br.

The probability distribution (PDF) for radioactive decay is an exponential function:

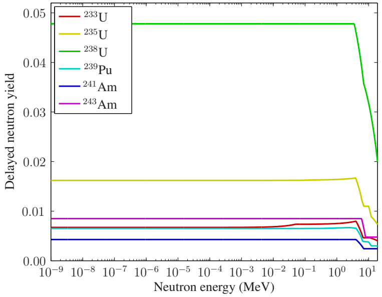
$$f(t) = \lambda e^{-\lambda t} \tag{70}$$

where λ is the decay constant. Similar to neutron path length, the decay time can be sampled from:

$$t = -\frac{1}{\lambda} \log \xi \tag{71}$$

[21] The European JEFF-3.1 and is successors use eight, other evaluations six precursor groups.

Neutron interactions: fission



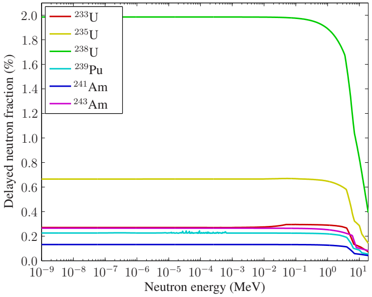


Figure 9: Delayed neutron yield (left) and fraction (right) of selected actinides as function of neutron energy. The yield depends on the probability of producing precursor isotopes, which depends on the fission product distribution of the actinide and the neutron energy. The fraction additionally depends on the prompt neutron yield, which varies from nuclide to nuclide and increases practically linearly as function of neutron energy.

Neutron interactions: scattering

Scattering includes all elastic and inelastic reaction modes in which the incident neutron is not lost. Additional neutrons may be produced in multiplying (n,2n), (n,3n), etc. reactions. In Monte Carlo simulation, scattering is handled by sampling a new direction and energy for the collided neutron and continuing the random walk.[22]

In two-particle collisions the scattering angle is coupled to energy transfer by conservation of energy and momentum. In such case, it is sufficient to sample the scattering angle and calculate the corresponding value for energy on-the-fly.

This is not the case for:

- Neutron-multiplying reactions: (n,2n), (n,3n), etc.
- Reactions where additional particles are emitted: (n,np), (n,nα)
- Continuum inelastic scattering (no discrete Q-value)

and the solution is to sample energy and angle from their own distributions, which preserve the conservation laws on the average.

[22] The additional neutrons emitted in multiplying scattering reactions are banked and their random walk is simulated once the original history is completed. In implicit Monte Carlo simulation the multiplication can be accounted for by increasing the neutron weight (see topic on implicit Monte Carlo).

Neutron interactions: scattering

The most common interaction in reactor applications is elastic scattering, which in LWR's constitutes over 90% of all interactions. Accurate modeling of collision kinematics is therefore absolutely essential for neutron slowing-down and thermalization.

Handling of elastic scattering can be divided into three categories:

1) Scattering from stationary nuclides
2) Scattering with free-gas model
3) Thermal scattering from bound moderator nuclides

The collision kinematics is best understood by defining three coordinate systems:

1) Laboratory frame-of-reference (L-frame), where the coordinates are fixed to the geometry
2) Target-at-rest frame-of-reference (T-frame), where the coordinates are fixed on the target
3) Center-of-mass frame-of-reference (C-frame), where the coordinates are fixed on the center-of-mass of the neutron-target system

The center-of-mass frame of reference is a coordinate system where the total momentum of the neutron-target system is zero, and neutron kinetic energy is preserved in elastic scattering.

Neutron interactions: scattering

When the neutron collides with a heavy nucleus at high energy, the velocity of the target can be approximated as zero.[23] The energy transfer is coupled to the scattering angle by preservation of energy and momentum. For elastic scattering this coupling can be written as:

$$E'_{\mathrm{T}} = \frac{E_{\mathrm{T}}}{(A+1)^2}\left[\mu_{\mathrm{C}} + \sqrt{A^2 - 1 + \mu_{\mathrm{C}}^2}\right]^2 \tag{72}$$

where E_{T} is the incident neutron energy in T-frame, μ_{C} is the scattering cosine in the C-frame and A is the atomic weight ratio (ratio of target to neutron mass). Similar coupling can be written for inelastic level scattering with discrete Q-value.

The reason to use a different frame-of-reference for the scattering angle is that elastic scattering is often isotropic in the C-frame. When this is the case, the scattering cosine can be sampled simply as:

$$\mu_{\mathrm{C}} = 2\xi - 1 \tag{73}$$

Scattering anisotropy increases with incident neutron energy, especially near resonances. In such case, the distribution is read from the interaction data, typically in the form of equi-probable cosine bins or tabular distribution, and the angle sampled using some set of scattering laws.

The final step of the procedure is to rotate the scattered direction vector over a random azimuthal angle (scattering always takes place in 3D)

[23] With this approximation, L-frame and T-frame become the same coordinate system.

Neutron interactions: scattering

The probability of up-scattering increases as the neutron is slowed down, and the stationary target approximation becomes poor when the neutron velocity is comparable to thermal motion. A better approximation is then the free-gas model, which essentially means that the target velocity is sampled from a Maxwellian-based distribution before the collision:

$$f(V, \mu) = C_0 v_{\mathrm{T}} V^2 e^{-\gamma^2 V^2} \tag{74}$$

where C_0 is a normalization constant,

$$\gamma = \sqrt{\frac{M}{2kT}} \tag{75}$$

and M is the target mass, k is the Boltzmann-constant and T is the temperature of the medium.

What differentiates this distribution from the Maxwell-Boltzmann distribution for speed is term v_{T}, which is the neutron speed in T-frame, i.e. the relative speed between the neutron and the target:

$$v_{\mathrm{T}} = \|\mathbf{v}_{\mathrm{L}} - \mathbf{V}\| = \sqrt{v_{\mathrm{L}}^2 + V^2 - 2v_{\mathrm{L}} V \mu} \tag{76}$$

where $\mu = \cos\theta$ is the cosine of the angle between the two velocity vectors.[24]

[24] This term reflects the fact that the neutron interaction probability per traveled path depends on the relative speed between the neutron and the nuclides in the medium, and it is best understood by considering a case when the neutron and target are moving in the same direction at same speed, in which case the reaction should not occur at all. In such case, $v_{\mathrm{T}} = 0$ and the probability given by (74) goes to zero.

Neutron interactions: scattering

After the target velocity is sampled, a transformation is made from the L-frame to T-frame:

$$\mathbf{v}_{\mathrm{T}} = \mathbf{v}_{\mathrm{L}} - \mathbf{V} \tag{77}$$

and a new velocity $\mathbf{v}'_{\mathrm{T}}$ obtained by sampling the scattering angle and applying Eq.(72). The velocity is then converted back to L-frame:

$$\mathbf{v}'_{\mathrm{L}} = \mathbf{v}'_{\mathrm{T}} + \mathbf{V} \tag{78}$$

The free-gas approximation is capable of modeling up-scattering and the collection of neutrons in the thermal region, but the model has two limitations:

1) Approximating scattering as a collision between two free particles fails when the neutron energy is comparable to the molecular and lattice binding energies of the target
2) The distribution of target velocity given by (74) assumes that the cross section is constant within the range of relative energy between the neutron and the target, which is a poor approximation near low-energy resonances

Molecular and lattice binding effects become important in moderator materials, such as light and heavy water and graphite, and they can be accounted for by using scattering laws with explicit cross sections and energy and angular distributions in the thermal region.

The second problem can be taken into account using the so-called Doppler-broadening rejection correction (DBRC) method, which is beyond the scope of this course.

Neutron interactions: scattering

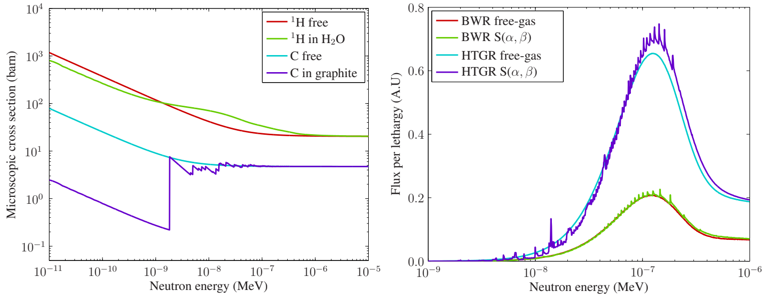


Figure 10: Left: Free-atom elastic scattering cross sections of ^{1}H and carbon compared to corresponding bound-atom cross sections in light water and graphite (all cross sections at 300 K). The secondary angular and energy distributions are given using so-called S(α, β) scattering laws. Right: thermal peak formed in water (BWR) and graphite (HTGR) in a Monte Carlo neutron transport simulation using the free-gas model and explicit thermal scattering laws (600 K temperature). The peaks in the distributions are not physical, but rather the result of sampling from discrete energy-angle distributions.

Simulating the neutron population

The simulated random walk of a single neutron history does not yet provide any useful results describing the behavior of the population. Instead, the transport simulation is repeated for a large number of histories, typically in the order of millions or even billions.

There are different ways in which the simulation can be carried out, corresponding to the different formulations of the transport equation, for example:

1) External source simulation
2) k-eigenvalue criticality source simulation (neutrons only)
3) α-eigenvalue criticality source simulation (neutrons only)

The first two are the most common simulation modes, and α-eigenvalue simulation is introduced briefly for the sake of completeness.

Simulating the neutron population: external source simulation

The most straightforward way to run the transport simulation is the external source mode, in which each neutron history is started from a user-specified source distribution. The random walk is carried out from beginning to end, and fission divides the history in multiple branches.

This simulation mode corresponds to solving the time-dependent transport equation:

$$\frac{1}{v}\frac{\partial}{\partial t}\psi(\boldsymbol{r},\hat{\boldsymbol{\Omega}},E,t) + \hat{\boldsymbol{\Omega}}\cdot\nabla\psi(\boldsymbol{r},\hat{\boldsymbol{\Omega}},E,t) + \Sigma(\boldsymbol{r},E)\psi(\boldsymbol{r},\hat{\boldsymbol{\Omega}},E,t) = Q + S + F \tag{79}$$

where Q is the external source, S is the scattering source and F is the fission source. External source transport simulation is basically always time-dependent, but if the external source is constant, the results can be integrated and averaged over time.

In its standard form the external source simulation is limited to non-multiplying and sub-critical systems, where the fission chains are finite in length. Applications include:

- Radiation shielding and dosimetry calculations
- Engineering applications (neutron diagnostics, oil well logging)
- Fusion neutronics
- Sub-critical accelerator-driven systems (ADS)

All transport simulations for photons, electrons, etc. are based on similar methods.

Simulating the neutron population: criticality source simulation

Simulation of a self-sustaining chain reaction is carried out using criticality source methods. The simulation is run in generations, or cycles, and the fission neutron distribution in the previous cycle forms the source distribution for the next cycle. The most common method is the k-eigenvalue simulation, which corresponds to solving the k-eigenvalue form of the transport equation:

$$\hat{\boldsymbol{\Omega}} \cdot \nabla \psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E) + \Sigma(\boldsymbol{r}, E)\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E) = S + \frac{1}{k}F \tag{80}$$

where k is the criticality eigenvalue, i.e. the neutron multiplication factor.

The most obvious difference between external and criticality source simulation is the way the neutron source is formed. In external source mode all neutrons are started from a user-defined distribution, while in criticality source mode the distribution is formed by iteration.

The source distribution starts with an initial guess, and it converges towards its final form cycle-by-cycle.[25] Before convergence is reached, the initial guess is reflected in the simulated neutron histories, and a number of initial cycles have to be skipped before starting the collection of results.

Slow source convergence can be a major problem in large geometries with high dominance ratio.[26] There are different ways to test and accelerate convergence, but it is not uncommon that the inactive cycles take a significant fraction of the overall running time.

[25] In deterministic transport theory the converged source distribution corresponds to the fundamental mode of the neutron flux, which begins to dominate after all transient modes have died out.

[26] The ratio of second to first eigenvalue.

Simulating the neutron population: criticality source simulation

If the neutron multiplication factor differs from unity, the source population increases or decreases from cycle to cycle.[27] To avoid this, the number of emitted fission neutrons is scaled by the multiplication factor from the previous cycle:[28]

$$\overline{\nu}' = \overline{\nu}/k \tag{81}$$

Since k is a random variable, so is the population size. The result is that the population oscillates about the initial size, but remains constant on the average.

In order to keep the results from each cycle consistently normalized, the population is fine-tuned before starting the next cycle. There are two options:

- In analog Monte Carlo, randomly selected neutrons are either killed ($k > 1$) or duplicated ($k < 1$) until the population matches the fixed size
- In implicit Monte Carlo, all neutrons are kept in the source population, but the total weight is preserved by renormalization, and individual neutron weight, $\mathcal{W}$ is included in the fission nubar:
$$\overline{\nu}' = \mathcal{W}\,\overline{\nu}/k \tag{82}$$

The neutron weight is revisited when discussing implicit Monte Carlo techniques.

[27] For example, if k = 1.3, source population starting with 1000 neutrons is multiplied to 1300, 1690, 2197, ..., and after 50 cycles 10^8 neutrons.

[28] Given by the ratio of new source neutrons to initial source population.

Simulating the neutron population: criticality source simulation

The fact that the source size can be adjusted is based on the linearity of the transport problem. Neutron histories within the cycle are (presumably) independent of each other, and simulating some partial sample of the population produces, on the average, the same result.

This would be the case if the adjustment was carried out in a completely random manner, but in fact, this is not what happens in the k-eigenvalue criticality source simulation. Instead, the adjustment is carried out at the fission event, when:

- All neutrons are located in the fuel
- All neutrons are at high energy

The result is that the contribution of fission source on reaction rates is either over- ($k < 1$) or under-estimated ($k > 1$), which introduces a bias in space and energy.[29]

Deterministic methods solving the eigenvalue form of the transport equation are subject to the same biases, as the time-dependence of flux is dropped, and balance between source and loss rates is obtained by modifying the average number of emitted fission neutrons. There is no easy way around this problem – the solution is biased whenever the system is away from criticality.[30]

[29] There is also a bias in time, which results from the fact that the simulation is run in source cycles, and the duration of a single history is not limited.

[30] It should be noted that the root cause of this issue is not in the way the transport problem is solved, but rather in its formulation: a time-dependent system is forced to steady-state condition by adjusting one of the source terms.

Simulating the neutron population: alpha-eigenvalue simulation

The α-eigenvalue method can be used for simulating the asymptotic (fundamental mode) solution of a time-dependent system, and it corresponds to solving the α-eigenvalue form of the transport equation:

$$\frac{\alpha}{v}\psi + \hat{\mathbf{\Omega}} \cdot \nabla\psi(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E) + \Sigma(\boldsymbol{r}, E)\psi(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E) = S + F \tag{83}$$

where the first term determines the time-absorption ($\alpha > 0$) or -production ($\alpha < 0$) rate needed to obtain balance between source and loss terms.

The simulation works similar to k-eigenvalue simulation, but the value of α is iterated in such way that the population size remains constant from cycle to cycle without adjustment in the number of emitted fission neutrons.

As discussed in Lecture 2, the α-eigenvalue method provides an unbiased solution for the fundamental mode flux in cases where $k \neq 1$, but method is not very commonly used.[31]

[31] It was noted in Lecture 2 that the modeled system itself is often an approximation of physical reality, for example, an infinite lattice of identical fuel assemblies. In such case, the asymptotic shape and spectrum of the time-dependent flux (solution to α-eigenvalue problem) may not be any closer to physical reality than the solution to the adjusted transport equation (solution to k-eigenvalue problem).

Collecting the results

The Monte Carlo transport simulation is run to obtain statistical estimates for integrals of the form:

$$F = \int_t \int_V \int_{\hat{\mathbf{\Omega}}} \int_E f(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E)\psi(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E, t) dV d\hat{\mathbf{\Omega}} dE dt \tag{84}$$

where f is a response function that can be evaluated at an arbitrary position of the phase space, most typically a reaction cross section. These estimates are based on the collection of simulated events (collisions, track-lengths, surface crossings, etc.) that occur during the course of the simulated random walk.

The estimates can be divided into:

- Analog estimates, based on recorded simulated physical events
- Implicit estimators, based the expected frequency of events

Implicit estimators are derived from analog estimators, with the purpose of obtaining better statistics. Even though the estimators introduced in the following can be used for calculating flux integrals, it should be noted that flux itself plays no role in Monte Carlo transport simulation.

Collecting the results: analog estimates

Analog estimates are the most straightforward way to obtain physical results from the Monte Carlo transport simulation. Each neutron history consists of a number of events containing relevant information on the transport process, which can be counted as-is:

- Collisions
- Sampled reactions
- Crossed surfaces

The integration domain in (84) is defined by separating the scores into different bins based on neutron position, energy and time.[32] For example:

- Fission rate in a specific fuel pin – count the number of simulated fission events in that fuel pin (integration over specific volume)
- Thermal neutron absorption in coolant – count the number of neutrons absorbed in the coolant with energy in the thermal region (integration over specific volume and energy)
- Total fission rate as function of time – count the number of fissions, and place the results in successive bins depending on the time of the event (integration over specific time)

These examples also illustrate the fact the results are always integrated over the variables.

[32] Similar binning can also be done for the direction of motion, but for most applications this is irrelevant.

Collecting the results: collision flux estimator

Implicit estimators are best understood by considering the collision estimate of neutron flux (CFE). When the neutron undergoes a collision at position $\boldsymbol{r}$ and energy E, the probability of sampling reaction x is the ratio of the reaction cross section to material total:

$$P_x(\boldsymbol{r}, E) = \frac{\Sigma_x(\boldsymbol{r}, E)}{\Sigma(\boldsymbol{r}, E)} \tag{85}$$

The probability is the same whether the reaction was actually sampled or not, so counting P_x as the result estimate means that the overall score reflects the *statistically expected* number of reactions x.

Since the total number of collisions is always greater than or equal to the number of sampled reactions, the implicit estimator gives better statistics. The overall score is given by the sum over all collisions:

$$x_n = \sum_i s_i \tag{86}$$

where the CFE is written as:

$$s_i = \frac{f(\boldsymbol{r}, E)}{\Sigma(\boldsymbol{r}, E)} \tag{87}$$

and f is the response function and Σ is the cross section that was used for sampling the path length.[33]

[33] In surface-tracking Σ is usually the material total, in delta-tracking it can be the total or the majorant.

Collecting the results: collision flux estimator

The relation between CFE and total collision rate:

$$R = \int_V \int_{\hat{\mathbf{\Omega}}} \int_E \Sigma(\boldsymbol{r}, E)\psi(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E) dV d\hat{\mathbf{\Omega}} dE \tag{88}$$

is easy to see. The response function is the macroscopic total cross section Σ, which means that $s_i = 1$ in (87). The sum in (86) is then reduced to the total number of collisions, as expected.

If the response function is set to 1, the result is the integral flux, and the value scored with the collision estimator is:

$$s_i = \frac{1}{\Sigma(\boldsymbol{r}, E)} \tag{89}$$

The connection is seen in that $1/\Sigma$ gives the neutron mean-free-path and the integral flux is equal to the sum of total path lengths traveled by neutrons in the medium.

The response function does not have to be a reaction cross section, and the only limitation is that the value has to be known at the points of collision. One example is the inverse neutron speed: $f = 1/v$, in which case the integral is written as:

$$\int_V \int_{\hat{\mathbf{\Omega}}} \int_E \frac{1}{v(E)} \psi(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E) dV d\hat{\mathbf{\Omega}} dE = \int_V \int_{\hat{\mathbf{\Omega}}} \int_E n(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E) dV d\hat{\mathbf{\Omega}} dE \tag{90}$$

The value gives the integral over neutron density, which can be used to calculate the average number of neutrons in a volume.

Collecting the results: track-length flux estimator

Another commonly used implicit estimator is the track-length estimate of neutron flux (TLE), which is based on the collection of neutron tracks. The overall score is given by the sum over all tracks:

$$x_n = \sum_i s_i \tag{91}$$

where the TLE is written as:

$$s_i = lf(E) \tag{92}$$

and l is the path length traveled by the neutron between collisions and surface crossings. The relation to flux is seen in that the integral flux is equal to the sum of total path lengths traveled by neutrons in the medium.

The TLE can be used similar to CFE, but there are a few differences:

- Since TLE is scored each time the neutron passes through a region, whether it collides or not, the number of scores is always greater than or equal to that of the CFE
- Since CFE is based on collisions that occur in discrete points in space, it can be used for calculating reaction rates in inhomogeneous material regions[34]

[34] This is seen in the fact that the response function in (87) may depend on the spatial coordinates, while that in (92) must be constant.

Collecting the results: track-length flux estimator

Most Monte Carlo codes rely on the use of track-length estimators, because of their superior performance. The differences are emphasized in a few specific cases:

- Calculation of flux integrals in optically thin regions (high probability to pass through, low probability to collide)
- Calculation of reaction rates with high threshold energy (fission neutrons exiting the fuel pin always contribute to TLE but only rarely to CFE)
- Calculation of reaction rates in low density or void regions (few or zero collisions for CFE, although the problem can be overcome by introducing virtual collisions)
- Calculation of reaction rates in regions located far or isolated from the active source (already poor statistics)

The main reason to use the collision flux estimator is that the transport routine is based on delta-tracking, which does not account for surface crossings needed for TLE.

Practical experience with Serpent has shown that there is no major difference between the two estimators in reactor physics applications, in which reaction rates are most typically scored in regions of high collision rate near the active source.

Collecting the results: tallies

There are different ways of combining the scores into statistical result estimates, or tallies. Perhaps the most intuitive way is to use batch statistics, meaning that the simulation of neutron histories is divided in multiple equal parts. In criticality source simulation the natural division is to use one batch per source cycle.

The idea is that all collisions or track lengths of all simulated neutron histories within a single batch are collected into a single batch-wise estimate:

$$x_n = \sum_i s_i \tag{93}$$

When these estimates are averaged over all batches, the sequence of values can be used to calculate the statistical mean:

$$\overline{x} = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{94}$$

and the associated standard deviation:

$$\sigma(\overline{x}) = \sqrt{\frac{1}{N(N-1)} \left[\sum_{n=1}^{N} x_n^2 - \frac{1}{N} \left(\sum_{n=1}^{N} x_n \right)^2 \right]} \tag{95}$$

which form the final result printed in the output.

Collecting the results: normalization of results

Since the number of scores (collisions, track-lengths, sampled reactions, etc.) in the batch-wise estimate (93) depends on the number of simulated neutron histories per batch, the absolute value of the result has no practical significance unless it is normalized to some physical variable.

Normalization is accomplished by fixing the value of one tally and calculating other reaction rates relative to it. If, for example, the fission and absorption rate estimates for batch n are x_n and y_n, respectively, and it is decided that the physical fission rate in the system is R_f, then the normalized capture rate can be obtained from:

$$\frac{R_\mathrm{f}}{x_n} = \frac{R_\gamma}{y_n} \Longleftrightarrow R_\gamma = \frac{R_\mathrm{f}}{x_n} y_n \tag{96}$$

Ratio R_f/x_n acts as the normalization coefficient for batch n, fixing the values of similarly normalized estimates to a user-specified reaction rate.

Some result estimates are calculated as ratios of two Monte Carlo integrals, in which case the normalization cancels out. This is the case, for example, for the implicit estimate of k_eff, defined formally as:

$$k_\mathrm{eff} = \frac{F}{T - S + L} \tag{97}$$

where F is the fission source rate, T is the total reaction rate S is the scattering source rate and L is the leakage rate, each calculated as Monte Carlo integrals.[35]

[35] The leakage rate can be calculated by an analog estimator counting the number of neutrons that escape the geometry.

Collecting the results: statistics

The standard deviation of tallies depends on:

1) The number of simulated histories per batch, which affects the variation of values $x_1, x_2, \ldots x_N$
2) The total number of simulated batches, which determines the number of terms N

There is no absolute truth to which is better, a large number of small batches or small number of large batches, but any extremes should be avoided.

Based on the central limit theorem it is assumed that the sequence of batch-wise estimates follows the normal distribution, but this assumption breaks down in the case of under-sampling, i.e. when the number of histories per patch is too low.

Another problem is related to the independence of batch-wise estimates. In criticality source simulation, the source distributions are formed from the fission distributions of the previous cycle, which means that the cycles are, in fact, correlated. This leads to incorrect estimates of standard deviation and violates the conditions of the central limit theorem.

In practice, the correlations are not very strong, but problems may occur in large geometries, in which the statistical errors of tallies scored in peripheral region are easily under-estimated.[36]

[36] There are studies showing that the inter-batch correlations in criticality source simulations can be reduced by including multiple source cycles within a single batch, which leads to better estimates of statistical error.

Implicit Monte Carlo methods

The previous description of Monte Carlo simulation relies mainly on analog methods, which means that the simulated random walk represents the history of a single neutron. There are also various implicit methods, which are essentially tricks that allow cheating the physical reality without introducing bias in the results.

Implicit methods rely on the concept of statistical weight, $\mathcal{W}$, which determines the contribution of an individual neutron to the overall results. If, for example, a neutron with weight 2 scores a tally, its contribution is equal to 2 scores made by neutron with weight 1. The use of statistical weights changes the batch wise scores (93) into:

$$x_n = \sum_i \mathcal{W}_i s_i \tag{98}$$

The concept of statistical weight was already introduced with the normalization of source population in criticality source simulation. In this context the weight is used to adjust the number of emitted fission neutrons (82):

$$\overline{\nu}' = \mathcal{W}\,\overline{\nu}/k \tag{99}$$

in order to keep the source size from diverging too far from the initial value.

The total weight of the population is also re-normalized before each cycle, to keep the batch-wise scores consistent. This is guaranteed by the fact that the weight is a multiplier for all result estimates and the total weight of the starting population remains constant from cycle-to-cycle.

Implicit Monte Carlo methods

Another commonly used technique is implicit capture, in which each collision reduces the neutron weight by a fraction defined by the capture probability:

$$\mathcal{W}' = \mathcal{W}\left(1 - \frac{\Sigma_\mathrm{c}}{\Sigma}\right) \tag{100}$$

and physical capture reactions are discarded in reaction sampling. Implicit capture allows neutron histories to extend further away from the source, which leads to improved statistics in far and isolated regions.

To avoid wasting CPU for the simulation of neutrons with very low weight,[37] implicit capture is often used with “Russian roulette”. If the neutron weight falls below a defined lower limit $\mathcal{W}_0$, the history is terminated with probability P_0. If the neutron survives, its weight is increase by factor:

$$\mathcal{W}' = \frac{1}{P_0}\mathcal{W} \tag{101}$$

The average weight of the population is preserved and the minimum weight is limited to $\mathcal{W}_0$.

Implicit methods become important in variance reduction techniques, which are used especially in radiation shielding calculations to get particles through structures that were designed to keep them out. This is an extensive topic, and beyond the scope of this course.

[37] Low weight is not so much a problem in itself, but if some region of the geometry is populated by neutrons with both high and low weight, the contribution of low-weight neutrons can become negligible.

Advantages and limitations of the Monte Carlo method

Continuous-energy Monte Carlo simulation has several advantages over deterministic transport methods:

- Transport simulation is inherently three-dimensional
- Capable of handling complex geometries, scalable to arbitrary resolution and level of spatial detail
- Capable of using the best available knowledge on neutron interactions without major approximations
- No application-specific limitations – same code and cross section data can be used for modeling any fuel or reactor type
- Time-dependent simulations can be run explicitly, without point-kinetics approximation
- Suitable to both small and large-scale problems
- In theory, anything that happens in the reactor core in reality can also be simulated

Most of these advantages result from the fact that neutron histories and events are simulated one-by-one, and integral reaction rates are calculated without obtaining a solution for the neutron flux.

Advantages and limitations of the Monte Carlo method

The method also has some major limitations:

- Running the transport simulation is computationally expensive
- All results are random variables, statistical precision is a function of running time
- Even though calculation of integral reaction rates is straightforward, other quantities require special tricks
- Not easy to apply approximations
- Not easily combined with deterministic transport methods
- Adjoint calculation requires special techniques

These limitations result from the same characteristic features as the advantages. Running time in particular is a practical limitation for the scale of applications.

Monte Carlo applications

Historically, Monte Carlo codes have been used for criticality safety, radiation shielding and validation of deterministic codes, but the development of CPU performance and especially parallel computing has broadened the scope of applications to more routine tasks in reactor physics.

Serpent is an example of a modern Monte Carlo code, and it has been developed at VTT since 2004 for reactor physics applications, including:

- Spatial homogenization and generation of group constants for deterministic reactor simulators
- Assembly-level burnup calculations
- Research reactor modeling

The development work is currently focused on three major topics:

1) Advanced methods for spatial homogenization
2) Coupled multi-physics applications
3) New applications beyond reactor physics

The second topic involves two-way coupling to system scale thermal hydraulics, CFD and fuel performance codes via a universal multi-physics interface.

For more information on Serpent, see the project website – http://montecarlo.vtt.fi

Summary of main topics

The Monte Carlo method in reactor physics is based on the simulated random walk of neutrons from one interaction to the next. The probability and outcome of each interaction is obtained from probability distributions, which represent the laws of physics for the transported neutron.

The main building blocks of the transport simulation are:

1) Tracking routine, handling the transport of neutrons through the geometry. Specific tasks include determining the material region at each collision site and determining the distance to the nearest material boundary.
2) Physics routine, handling the interactions, emission of secondary neutrons and collision kinematics of scattering events
3) Statistical methods, used for collecting the results from simulated events and forming the statistical means and associated standard deviations for the final output

The simulation can be carried out in external or criticality source mode, which in deterministic transport theory correspond to solving the time-dependent and eigenvalue form of the transport equation, respectively.

The Monte Carlo method can be used for a wide range of transport problems almost without approximations, and it has the potential to produce very accurate results. The method also has its limitations, in particular related to the high computational cost and the stochastic nature of the results.

Topics of next lecture

In the next lecture (14.3.2019), the viewpoint is switched back to deterministic methods with the introduction of diffusion theory.

Specific topics include:

- Transport theory revisited
- Derivation of diffusion theory
- Validity of diffusion approximation
- Two-group diffusion equation
- Solution in homogeneous medium