Aalto University
School of Science
and Technology

Monte Carlo method in particle transport simulations
Lecture 4 – Transport simulation

Jaakko Leppänen

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

Nov. 3, 2020

Topics of this lecture

Lecture topics:

- Sampling the path length
- Tracking algorithms:
  1. Surface-tracking
  2. Delta-tracking
- Simulating the particle population:
  1. External source simulation
  2. k-eigenvalue criticality source simulation
- Result estimators

4th programming exercise

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 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 in the following, 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 particle 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:

$$dP = \Sigma dl \tag{1}$$

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

$$dP_0 = -P_0(l)dP = -P_0(l)\Sigma dl \tag{2}$$

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

$$P_0(l) = e^{-l\Sigma} \tag{3}$$

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

$$P_0(l)dP = P_0(l)\Sigma dl = \Sigma e^{-l\Sigma} dl \tag{4}$$

Simulated random walk: sampling the path length

In other words, Eq. (4) gives the probability density function (PDF) of the distance to the next collision:

$$f(l) = \Sigma e^{-l\Sigma} \tag{5}$$

The probability distribution is exponential and the distance to the next collision site can be sampled using the inversion method (see Lecture 1).

The cumulative distribution function (CDF) is given by integration:

$$F(l) = \int_0^l \Sigma e^{-l'\Sigma} dl' = 1 - e^{-l\Sigma} \tag{6}$$

The inverse of the CDF in (6) is

$$F^{-1}(\xi) = -\frac{1}{\Sigma} \log(1 - \xi) \tag{7}$$

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

$$l = -\frac{1}{\Sigma} \log \xi \tag{8}$$

It is important to note that the prerequisite of using (8) for sampling the distance to the next collision site is that the material is infinite and homogeneous. If this is not the case, the second equality in (6) does not hold, and path lengths sampled from (8) are not statistically valid.

Simulated random walk: sampling the path length

The most common case when the previous condition is violated is when the particle crosses a boundary between two material regions.[1] In such case, the PDF of free path length is a piecewise continuous function, which can be written in two parts, taking into account the conditional probability that the particle reaches the boundary crossing at δ:

$$f(l) = \begin{cases} \Sigma_1 e^{-\Sigma_1 l} & \text{when } l \leq \delta \\ e^{-\Sigma_1 \delta} \Sigma_2 e^{-\Sigma_2 (l-\delta)} & \text{when } l > \delta \end{cases} \tag{9}$$

This approach becomes somewhat impractical in the general case, when the particle 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 particle's path can be considered a starting point of a new sample.

[1] 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 a 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 particle 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 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.[2]

Surface tracking is considered the standard tracking algorithm and it is used by virtually every Monte Carlo particle 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 particle has to be stopped at each boundary crossing becomes a computational bottleneck when the mean-free-path is long compared to the dimensions

[2] 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 1 Surface-tracking algorithm

1: **for** $j \leftarrow 1$ to ∞ **do** ▷ Loop until collision
2: Get cross section Σ_j at current position $\boldsymbol{r}_j$ ▷ Call cell search routine
3: Get distance δ 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 particle 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 particle over boundary crossing[3]
10: **end if**
11: **end for**

[3] 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 particle 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 particle 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 particle.

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:

$$\Sigma'(\boldsymbol{r}, E) = \Sigma(\boldsymbol{r}, E) + \Sigma_0(\boldsymbol{r}, E) \tag{10}$$

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:

$$\Sigma'_1(E) = \Sigma'_2(E) = \Sigma'_3(E) \cdots = \Sigma_\mathrm{m}(E) \tag{11}$$

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:

$$\Sigma_\mathrm{m}(E) = \max\left[\Sigma(\boldsymbol{r}, E)\right] \tag{12}$$

Unlike the physical total cross section, which depends on the material located at the particle 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:

$$l = -\log(\xi)/\Sigma_{\mathrm{m}} \tag{13}$$

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:

$$P = \frac{\Sigma(\boldsymbol{r}, E)}{\Sigma_{\mathrm{m}}(E)} \tag{14}$$

If the collision is rejected, a new path length is sampled from (13) and the particle 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 2 Delta-tracking algorithm

1: Get majorant cross section Σ_{m}
2: **for** $j \leftarrow 1$ to ∞ **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 particle to tentative collision site
5: Get cross section Σ_{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 particle at the material boundaries. This becomes significant for computational performance in geometries where the mean-free-path is long compared to dimensions.[4]

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 particle flux, discussed later on, and reaction rate estimates need to be calculated using the potentially less efficient collision estimator (CFE).

[4] 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

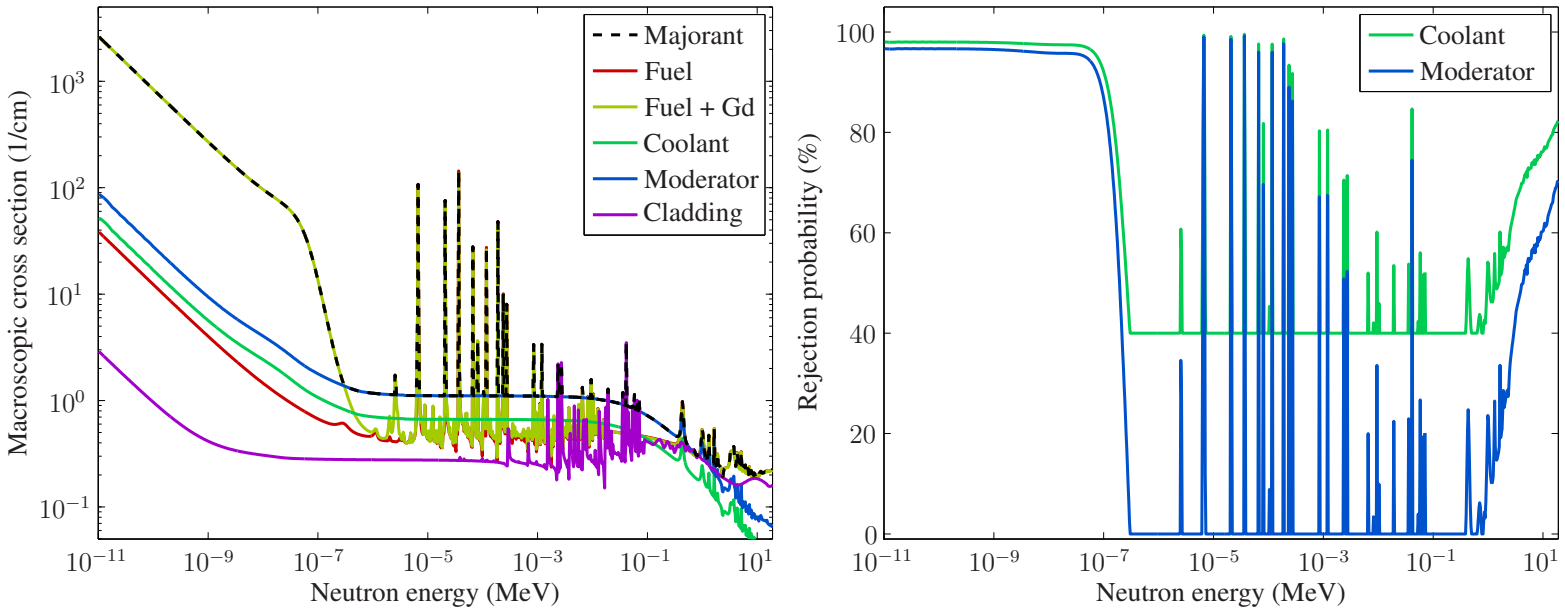

Figure 1: Left: Majorant and macroscopic neutron cross sections in a system with localized heavy absorber (Gd-fuel pins in BWR assembly). The majorant is dominated by the high capture cross sections of ^{155}Gd and ^{157}Gd, even though the burnable absorber pins occupy a relatively small volume of the 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.

Simulating the particle population

The simulated random walk of a single particle 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. α-eigenvalue simulations can be useful in some applications when studying the time-behavior of sub- or super-critical systems.

It is assumed that the simulated population size is divided into a number of equal size batches. The statistical estimators (mean + standard deviation) are collected by averaging over the batch-wise results.

Simulating the particle population: external source simulation

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

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

$$\frac{1}{v}\frac{\partial}{\partial t}\psi(\boldsymbol{r},\hat{\boldsymbol{\Omega}},E,t)+\hat{\boldsymbol{\Omega}}\cdot\nabla\psi(\boldsymbol{r},\hat{\boldsymbol{\Omega}},E,t)+\Sigma(\boldsymbol{r},E)\psi(\boldsymbol{r},\hat{\boldsymbol{\Omega}},E,t)=Q+S+F \qquad (15)$$

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 source term is constant, the results can be integrated and averaged over time.

In its standard form neutron external source simulations are 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 particle population: external source simulation

Algorithm 3 External source simulation

for $m \leftarrow 1$ to M **do** ▷ Loop over batches
  for $n \leftarrow 1$ to N **do** ▷ Loop over particles per batch
    Sample $\boldsymbol{r}$, E, $\hat{\boldsymbol{\Omega}}$ and t for source particle n ▷ Call source routine
    Move particle n to temporary bank
    while more particles in bank **do**
      Retrieve particle from bank (obtain $\boldsymbol{r}$, E, $\hat{\boldsymbol{\Omega}}$ and t)
      while particle is alive **do**
        Move particle to next collision site $\boldsymbol{r}'$ ▷ Call surface- or delta-tracking routine
        Sample reaction:
        if Capture (or particle escapes the geometry) **then**
          Terminate history
        else if Scattering **then**
          Sample new E and $\hat{\boldsymbol{\Omega}}$
        else if Fission **then**
          Store $\overline{\nu}$ new source particles in bank and terminate history
        end if
      end while
    end while
  end for
  Handle batch-wise statistics
end for

Simulating the particle 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:

$$\hat{\boldsymbol{\Omega}} \cdot \nabla \psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E) + \Sigma(\boldsymbol{r}, E)\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E) = S + \frac{1}{k}F \tag{16}$$

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.[5] 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.[6] 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.

[5] 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.

[6] The ratio of second to first eigenvalue.

Simulating the particle population: criticality source simulation

If the neutron multiplication factor differs from unity, the source population increases or decreases from cycle to cycle.[7] To avoid this, the number of emitted fission neutrons is scaled by the multiplication factor from the previous cycle:[8]

$$\overline{\nu}' = \overline{\nu}/k \tag{17}$$

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:

$$\overline{\nu}' = \mathcal{W}\,\overline{\nu}/k \tag{18}$$

[7] 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.

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

Simulating the particle 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.[9]

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.[10]

[9] 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.

[10] 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 particle population: criticality source simulation

Algorithm 4 Criticality source simulation

Sample $\boldsymbol{r}$, E and $\hat{\boldsymbol{\Omega}}$ for N source neutrons ▷ Call source routine to obtain initial guess
Move neutrons to source bank 1
for $m \leftarrow 1$ to M **do** ▷ Loop over cycles
    while more neutrons in source bank m **do**
        Retrieve neutron from bank m (obtain $\boldsymbol{r}$, E and $\hat{\boldsymbol{\Omega}}$)
        while neutron is alive **do**
            Move neutron to next collision site $\boldsymbol{r}'$ ▷ Call surface- or delta-tracking routine
            Sample reaction:
            if Capture (or neutron escapes the geometry) **then**
                Terminate history
            else if Scattering **then**
                Sample new E and $\hat{\boldsymbol{\Omega}}$
            else if Fission **then**
                Store $\overline{\nu}$ new source neutrons in source bank $m + 1$ and terminate history
            end if
        end while
    end while
    Handle batch-wise statistics
    Re-normalize source bank $m + 1$
end for

Simulating the particle population

Notes to algorithms 3 and 4:

- Escape terminates the history similar to capture, and it is assumed that the boundary conditions (vacuum, reflective or periodic) are handled by the tracking algorithm
- In criticality source simulation also fission terminates the history
- Non-fission multiplying reactions (e.g. (n,xn) for neutrons and pair production for photons) were omitted (handled by storing the extra particles in bank)
- External source simulations often include additional cut-offs (e.g. time cut-off in super-critical multiplying systems or energy-cut off in photon transport simulations)
- Criticality source simulation essentially integrates over all time, since the duration of a single generation is not fixed in any way.
- Delayed neutrons are usually handled similar to prompt neutrons (with different energy distribution and emission time).
- Handling of inactive cycles is not included in the description of algorithm 4 (collection of results is not started right away, but after cycle M_0).

Collecting the results

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

$$F = \int_t \int_V \int_{\hat{\boldsymbol{\Omega}}} \int_E f(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E)\psi(\boldsymbol{r}, \hat{\boldsymbol{\Omega}}, E, t) dV d\hat{\boldsymbol{\Omega}} dE dt \tag{19}$$

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 particle 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 (19) is defined by separating the scores into different bins based on particle position, energy and time.[11] 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.

[11] 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 flux (CFE). When the particle 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:

$$P_x(\boldsymbol{r}, E) = \frac{\Sigma_x(\boldsymbol{r}, E)}{\Sigma(\boldsymbol{r}, E)} \tag{20}$$

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:

$$x_n = \sum_i s_i \tag{21}$$

where the CFE is written as:

$$s_i = \frac{f(\boldsymbol{r}, E)}{\Sigma(\boldsymbol{r}, E)} \tag{22}$$

and f is the response function and Σ is the cross section that was used for sampling the path length.[12]

[12] In surface-tracking Σ is the material total, in delta-tracking it can be the total or the majorant, depending on whether all or only physical collisions are accounted for in the CFE.

Collecting the results: collision flux estimator

The relation between CFE and total collision rate:

$$R = \int_V \int_{\hat{\mathbf{\Omega}}} \int_E \Sigma(\boldsymbol{r}, E)\psi(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E)dV d\hat{\mathbf{\Omega}} dE \tag{23}$$

is easy to see. The response function is the macroscopic total cross section Σ, which means that $s_i = 1$ in (22). The sum in (21) 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:

$$s_i = \frac{1}{\Sigma(\boldsymbol{r}, E)} \tag{24}$$

The connection is seen in that $1/\Sigma$ gives the mean-free-path and the integral flux is equal to the sum of total path lengths traveled 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 particle speed: $f = 1/v$, in which case the integral is written as:

$$\int_V \int_{\hat{\mathbf{\Omega}}} \int_E \frac{1}{v(E)} \psi(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E)dV d\hat{\mathbf{\Omega}} dE = \int_V \int_{\hat{\mathbf{\Omega}}} \int_E n(\boldsymbol{r}, \hat{\mathbf{\Omega}}, E)dV d\hat{\mathbf{\Omega}} dE \tag{25}$$

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

Collecting the results: track-length flux estimator

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

$$x_n = \sum_i s_i \tag{26}$$

where the TLE is written as:

$$s_i = l f(E) \tag{27}$$

and l is the path length traveled by the particle 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 the particle in the medium.

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

- Since TLE is scored each time the particle 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[13]

[13] This is seen in the fact that the response function in (22) may depend on the spatial coordinates, while that in (27) must be constant over path length l.

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 scoring also 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 the Serpent code[14] 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.

[14] See - `http://montecarlo.vtt.fi`

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 particle 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 histories within a single batch are collected into a single batch-wise estimate:

$$x_n = \sum_i s_i \tag{28}$$

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

$$\overline{x} = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{29}$$

and the associated standard deviation:

$$\sigma(\overline{x}) = \sqrt{\frac{1}{N(N-1)} \left[\sum_{n=1}^{N} x_n^2 - \frac{1}{N} \left(\sum_{n=1}^{N} x_n \right)^2 \right]} \tag{30}$$

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 (28) depends on the number of simulated histories per batch, the absolute value of the result has no practical significance unless it is normalized to some physical variable.

Normalization can be accomplished, e.g. 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:

$$\frac{R_{\mathrm{f}}}{x_n} = \frac{R_\gamma}{y_n} \Longleftrightarrow R_\gamma = \frac{R_{\mathrm{f}}}{x_n} y_n \tag{31}$$

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:

$$k_{\mathrm{eff}} = \frac{F}{T - S + L} \tag{32}$$

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.[15]

[15] 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 the peripheral region are easily under-estimated.[16]

[16] 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.

4th programming exercise

The main goal in the 4th programming exercise is to combine the geometry routine developed in exercise 2 into the physics routine in exercise 3 in order to perform the transport simulation in a heterogeneous geometry. The tasks are relatively simple, and more realistic cases are included in the last round of exercises.

Mandatory tasks:

- Implement either the surface- or the delta-tracking algorithm to perform the particle transport simulation, and the functions to collect statistical estimates using the collision or track-length estimate of neutron flux.
- Implement the external source simulation algorithm.
- Demonstrate the particle transport simulation in a hollow cylinder partially filled with water:
  - Hollow cylindrical steel container with inner diameter of 40 cm, inner height 70 cm, wall thickness 0.5 cm, water surface at 35 cm from the bottom, upper part filled with air.[17]
  - Point-wise isotropic mono-energetic 1 MeV neutron source located at cylinder center-line, 10 cm from the bottom, emitting 10^6 neutrons per second.
  - Calculate total leak rate and analog and implicit estimators (CFE or TLE) of total collision and total absorption rate in water, air and container

[17] Steel is approximated by 100% ^{56}Fe and air with 100% ^{14}N. Cross sections are provided at `http://virtual.vtt.fi/virtual/montecarlo/misc/PHYS-E0565/`

4th programming exercise

Bonus tasks:

1. Implement both the surface- and the delta-tracking algorithm to perform the particle transport simulation, and the functions to collect statistical estimates using the collision and track-length estimate of neutron flux. Repeat the previous tasks with different options and compare results and FOM's. (+2 points)
2. Calculate the total collision rate distribution in the previous case on a 3D Cartesian mesh. Visualize the results. (+1 point)
3. Implement the criticality source simulation algorithm and estimate the critical surface level in the previous cylinder geometry, with water replaced with 50/50 molar mixture of water and 20% enriched uranium. Calculate k_{eff} for 25/75 and 75/25 solutions with the same surface level. (+3 points)
4. Calculate the average neutron density in the critical 50/50 molar mixture for 1 W power level, assuming that fission produces 200 MeV of energy. Compare the result to the total atomic density of the medium. (+1 point)

4th programming exercise

Table 1: Material compositions (atomic densities in $10^{24}/\mathrm{cm}^3$).

Material	Composition	Density
Water	^{1}H	6.68723E-02
	^{16}O	3.34362E-02
Steel	^{56}Fe	8.39767E-02
Air	^{14}N	5.16062E-05
50/50 uranium-water solution	^{1}H	3.34362E-02
	^{16}O	1.67181E-02
	^{235}U	4.81866E-03
	^{238}U	1.92746E-02

4th programming exercise

Milestones:

- Combination of geometry (Ex 2) and physics routines (Ex 3) into a routine capable of transporting particles through heterogeneous geometries.
- Implementation of result estimators capable of extracting physical reaction rates from the Monte Carlo simulation.

Notes and tips:

- The surface-tracking algorithm requires the capability to calculate distance to the nearest boundary surface within the line-of sight. Transporting particles through the geometry with surface-tracking is very similar to the calculation of cell volumes by drawing the lines in one of the exercises in round 2.
- When the particle is moved over a boundary surface, it is important to add a small extrapolation distance to the path length or otherwise the routine will fail because of limited numerical precision.
- The delta-tracking algorithm can be implemented without calculating the surface distances.
- The TLE requires using surface-tracking.
- The CFE can be applied to physical or all (physical + virtual) collisions, but the the cross section used in the estimator must be chosen correspondingly (total or majorant).
- Applying the CFE to calculate flux in void requires scoring virtual collisions. This also improves the statistics in low-density regions (e.g. air).