Todo list

Weighted Delta-Tracking with Scattering implemented in the Serpent 2 Monte Carlo Code

by

J.S. Rehak

A thesis submitted in partial satisfaction of the

requirements for the degree of

Masters of Science

in

Nuclear Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Rachael Slaybaugh, Chair
Professor ??
Professor ??

Spring 2017

The thesis of J.S. Rehak, titled Weighted Delta-Tracking with Scattering implemented in the Serpent 2 Monte Carlo Code, is approved:

Chair	______________________________	Date	____________
	______________________________	Date	____________
	______________________________	Date	____________

University of California, Berkeley

Weighted Delta-Tracking with Scattering implemented in the Serpent 2 Monte Carlo Code

Contents

List of Figures

List of Tables

Chapter 1

Method

The Serpent 2 Monte Carlo code uses a combination of ray tracing and delta-tracking to simulate the propagation of particles. The weighted delta-tracking (WDT) method modifies delta-tracking for an absorbing medium, replacing virtual collisions with a weight reduction. In this section, we will describe these neutral particle propagation methods, derive an extension of WDT to include scattering, and discuss how these methods are implemented in Serpent 2.

1.1 Particle Propagation Methods

As a particle moves through a medium, multiple forces and events will affect its trajectory. we will consider neutral elementary particles, neutrons, so that the effects of gravitational and electromagnetic force are negligible. It is therefore a good assumption that a neutron will travel in a straight line until a physical collision with another particle. Despite the large amount of neutrons present in our simulation, we will ignore improbable collisions between neutrons themselves. Neutron collisions with atoms results in an array of effects, including absorption and scattering. The relative probability of a collision occurring, and the resulting type of collision are dependent on atom that the neutron collides with and the energy of the neutron. Modeling the propagation of neutrons is accomplished through a variety of algorithms, some of which we described in this section.

Ray Tracing

The basic algorithm for simulating particle transport using Monte Carlo is ray tracing. This method follows a particle from collision to collision, assuming that it travels in a straight, statistically sampled path. Material properties determine the distance between collisions, and therefore material boundaries must be considered explicitly.

Particles propagating through a material have an interaction probability characterized by the material's experimentally determined total-cross section, a function of position and the incident neutron energy, $\Sigma_t(\mathbf{r}, E)$. The probability of an interaction occurring in a differential distance traveled ds is related to the cross-section:

$$\frac{dP}{ds} = \Sigma_t(\mathbf{r}, E) \tag{1.1}$$

We assume that each collision will remove the neutron from the incident flux, as it is absorbed or deflected out of the original path in either position, energy, or both. The medium therefore attenuates an incident mono-energetic neutron flux ϕ_0 as a function of distance:

$$\phi(s) = \phi_0 e^{-s\Sigma_t(\mathbf{r})} \tag{1.2}$$

The probability that a neutron has its first interaction in differential distance ds after traveling a distance s is found by dividing the flux at that position $\phi(s)$ by the total flux:

$$f(s)ds = \frac{\phi(s)}{\int_0^\infty \phi(s)ds} ds \tag{1.3}$$

The basis of Eq. (1.3) is clarified by an example: if the flux after a distance s is half the total flux, then half of the neutrons have undergone an interaction.

We consider the cross-section in a single material region, where we assume that the material is homogeneous. In this case, the cross-section is no longer a function of position. Using Eqs. (1.2) and (1.3), and assuming that the total cross-section is constant over $\mathbf{r}$[4]:

$$f(s)ds = \Sigma_t e^{-s\Sigma_t} ds \tag{1.4}$$

This is the PDF of the distance traveled before the first collision, s. As the distance a neutron travels increases, the value of $f(s)$ decreases; it is less probable that a neutron will travel further without collision. The PDF gives us the probability at a point s that a neutron undergoes a collision exactly there, integrating over the distance s provides us the probably that a neutron undergoes a collision between 0 and s. This is the CDF:

$$F(s) = \int_0^s f(s')ds' = 1 - e^{-s\Sigma_t} \tag{1.5}$$

As expected, increasing s causes the probability of any interaction, $F(s)$, to approach unity. A plot illustrating the behavior of the PDF and CDF are shown in Fig. 1.1.

As the CDF ranges from zero to unity, we can sample its value by a uniformly distributed random variable $\xi \in [0, 1)$. The distance traveled s, referred to as the path length, can then be expressed as a function of this sampled random variable:

$$\begin{aligned} F(s) = 1 - e^{-s\Sigma_t} &= \xi \\ \ln(e^{-s\Sigma_t}) &= \ln(1-\xi) \\ -\Sigma_t s &= \ln(\xi) \\ s(\xi) &= \frac{1}{\Sigma_t}\ln(\xi) \end{aligned}$$

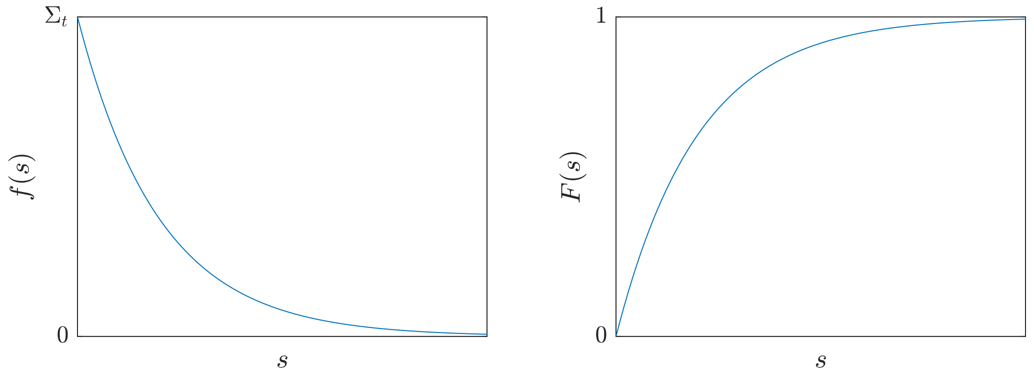


Figure 1.1: Plots of the CDF and PDF for collision probability per distance travelled s.

After sampling the path length of the neutron, its position is updated, based on its original position and direction. We assumed that the cross-section Σ_t was constant in a material region, so the sampled path length is only valid as long as the neutron remains in that region. If the neutron reaches the boundary between two material regions, a new path length must be sampled using the cross-section of the region it is entering. Each time a path length is sampled, the distance to the nearest boundary in the direction of motion is determined, and the neutron is moved to the boundary and path length is resampled if appropriate. This can become computationally expensive in complicated geometries and when the probability of crossing boundaries with each sample path length is high.

Woodcock Delta-tracking

As discussed in Section 1.1, the value of Σ_t at a given position depends on the material at that point. Therefore, $\Sigma_t(\mathbf{r})$ is a piece-wise discontinuous function that varies arbitrarily with position and the geometry of the problem [3]. Using the ray tracing method, neutrons must stop at boundaries to sample a new path length in a new material region. To avoid the computational inefficiency that arises in geometrically complicated regions, a rejection sampling technique known as Woodcock delta-tracking was developed [8].

Woodcock delta-tracking introduces the concept of the majorant cross-section, chosen to be the maximum of all material total cross-sections in the region of interest.

$$\Sigma_{\text{maj}} \equiv \max_{\mathbf{r}\in\mathbb{V}}\{\Sigma_t(\mathbf{r})\} \tag{1.6}$$

Where $\mathbb{V}$ is the volume of interest, ash shown in a one-dimensional region in Fig. 1.2.

The majorant cross-section can also be represented as the summation of the total cross-section and a delta cross-section:

$$\Sigma_{\text{maj}} = \Sigma_\delta(\mathbf{r}) + \Sigma_t(\mathbf{r}) \tag{1.7}$$

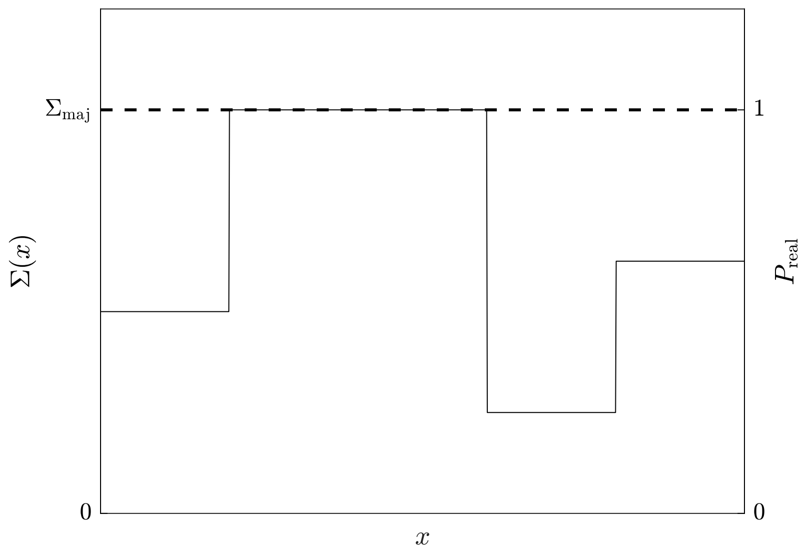


Figure 1.2: Total cross-section as a function of position in one dimension. The majorant cross-section is the largest value in the region of interest and determines the probability of a real collision.

Following from the definition of Σ_{maj} in Eq. (1.6), the function $\Sigma_\delta(\mathbf{r})$ is chosen such that Σ_{maj} is constant for the entire region of interest. At the position $\mathbf{r}$ where the maximum value of $\Sigma_t(\mathbf{r})$ occurs, the delta cross-section is zero.

The majorant cross-section is constant throughout the entire region of interest, so we can treat it as a single material. Following the same derivation in Section 1.1, the PDF of the first collision occurring after s in the region of interest using the majorant cross-section is given by:

$$\begin{aligned} f_{\text{maj}}(s) &= \Sigma_{\text{maj}} e^{-\Sigma_{\text{maj}} s} && (1.8) \\ &= (\Sigma_\delta(\mathbf{r}) + \Sigma_t(\mathbf{r})) e^{-\Sigma_{\text{maj}} s} && (1.9) \end{aligned}$$

In most of the region of interest, the majorant cross-section is not the real cross-section. We must use a technique called rejection sampling to simulate sampling the real $\Sigma_t(\mathbf{r})$ while actually sampling using Σ_{maj}.

Rejection Sampling

As described by Lux and Koblinger [4], rejection sampling requires a PDF of interest, $f(x)$, and a second PDF $g(x)$ for which:

$$f(x) \leq M g(x), \forall x \tag{1.10}$$

Sampling from $Mg(x)$ and accepting these samples with probability:

$$P = \frac{f(x)}{Mg(x)} \tag{1.11}$$

replicates sampling directly from $f(x)$.

For example, consider a PDF of interest $f(x) = x^2$ on the interval $x \in (0, 1]$ as shown in Fig. 1.3. As described in the previous section, we could sample $f(x)$ using the CDF $F(x)$

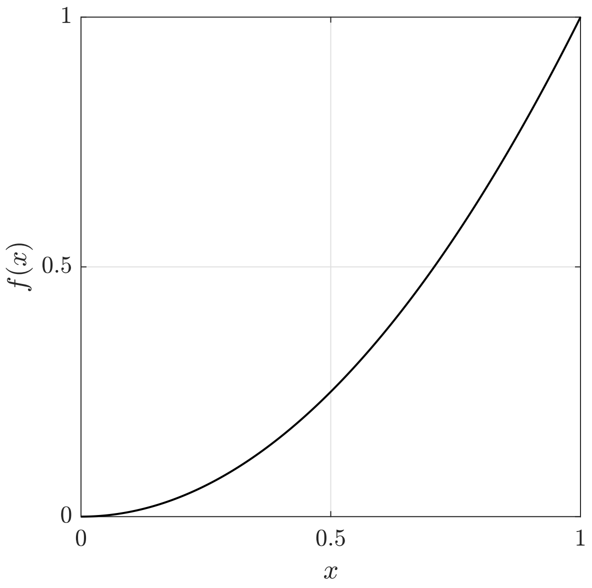


Figure 1.3: Example PDF function $f(x)$.

found by integrating and then inverting. Instead, we can sample from a different PDF that is always majorant of $f(x)$, as defined in Eq. (1.10). We can choose $g(x) = 1$, as $f(x) \leq 1$ on the interval of interest. The CDF of $g(x)$ is easy to determine, because it is a constant value, and is just $G(x) = x$. Therefore, we can sample $g(x)$ by simply sampling a random value $\xi \in (0, 1]$ and taking this as the value of x. To replicate sampling from $f(x)$, we then sample another random value $\xi_2 \in (0, 1]$ and accept our value of x using the probability defined in Eq. (1.11):

$$\xi_2 \leq \frac{f(x)}{g(x)} = x^2$$

Following this algorithm using the simple Matlab script:

```
a = [];                        % Accepted samples
for i = 1:1000
    x = rand;                  % Sample g(x)
    y = rand;                  % Gen random number xi2
    if y <= x.^2               % If xi2 <= f(x)/g(x)
        a(end+1) = x;          % Accept sample
    end
end
```

Generates the histogram shown in Fig. 1.4. As expected, the procedure has reproduced the desired PDF of $f(x) = x^2$. Although this requires sampling two random variables, it can be advantageous if $F(x)$ is difficult or impossible to invert.

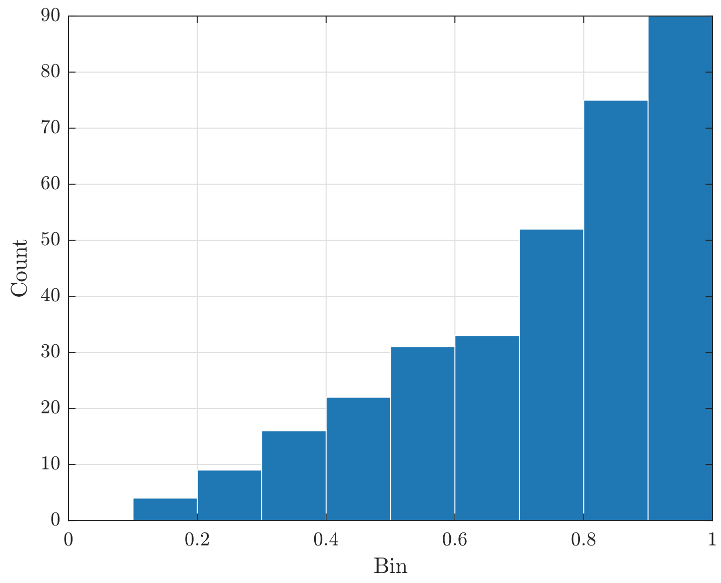


Figure 1.4: Histogram of results from sampling and applying the rejection sampling algorithm.

Application to Delta-tracking

As we saw in Fig. 1.2, the total cross-section Σ_t is a piecewise discontinuous function that depends on the geometry of our problem. Therefore, each region has a different CDF for sampling path length, and described in Section 1.1. Using rejection sampling, we can sample the real collision PDF, using a different, simpler PDF. In Woodcock delta tracking, the second PDF $g(x)$ is chosen to be the majorant PDF, Eq. (1.8). This is beneficial because the majorant cross-section is constant over the entire region.

These functions are both maximized at $x = 0$, where the inequality of Eq. (1.10) is satisfied by setting $M = 1$:

$$\Sigma_t(\mathbf{r}) \leq \Sigma_{\text{maj}}(\mathbf{r}), \forall \mathbf{r} \in \mathbb{V} \tag{1.12}$$

We sample path length using the constant majorant cross-section:

$$s_{\text{maj}}(\xi) = -\frac{1}{\Sigma_{\text{maj}}}\ln(\xi) \tag{1.13}$$

Which samples the PDF defined in Eq.(1.6):

$$\begin{aligned} f_{\text{maj}}(s) &= \Sigma_{\text{maj}} e^{-\Sigma_{\text{maj}} s} \\ &= (\Sigma_\delta(\mathbf{r}) + \Sigma_t(\mathbf{r}))e^{-\Sigma_{\text{maj}} s} \end{aligned}$$

We can see that this PDF is the sum of two different ones: one representing actual collisions based on the real total cross-section Σ_t and one representing the non-physical collisions based on Σ_δ. We call these non-physical collisions "virtual" collisions, and these should be eliminated by our rejection sampling. To apply rejection sampling, we therefore pick the PDF we want to sample $f(x)$ and the PDF we will actually sample $g(x)$ as such:

$$\begin{aligned} f(x) &= \Sigma_t(\mathbf{r})e^{-\Sigma_{\text{maj}} s} \\ g(x) &= \Sigma_{\text{maj}} e^{-\Sigma_{\text{maj}} s} \end{aligned}$$

We will therefore accept samples with the probability given in Eq. (1.11):

$$P_{\text{real}}(\mathbf{r}) = \frac{f(x)}{Mg(x)} = \frac{\Sigma_t(\mathbf{r})e^{-\Sigma_{\text{maj}} s}}{\Sigma_{\text{maj}} e^{-\Sigma_{\text{maj}} s}} = \frac{\Sigma_t(\mathbf{r})}{\Sigma_{\text{maj}}} \tag{1.14}$$

We will refer to this as the probability of a "real" collision, P_{real} to differentiate these physical collisions from the non-physical virtual collisions. It is important to note that the probability is independent of path length s, but is dependent on position $\mathbf{r}$. At each collision, the material region at the neutron position must be determined, but we do not need to explicitly track boundaries nor calculate their distance each time a path length is sampled. The algorithm for delta-tracking is shown in Fig. 1.5.

Now, the path length can be sampled across multiple material regions of varying Σ_t without explicitly stopping the neutron at a given boundary. This method can become computationally inefficient in regions where the total cross-section is much less than the majorant cross-section, leading to oversampling of virtual collisions. This is seen in geometries that include localized absorbers, such as control rods. Another downside is that the track-length estimator (TLE) for flux cannot be used. The TLE requires calculating the track-lengths within a particular material cell, and therefore does not work when the neutron path length can cross one or more material boundaries. The collision flux estimator (CFE) can be used in its place, but often results in inferior statistics as not every track length sampled ends in a collision [3].

Nonuniform Density Distributions

Rejection sampling can also be used when the total cross-section is not constant within a material region. As discussed by Leppänen [3], Serpent 2 conducts a rejection sampling routine

Algorithm 1 Delta-tracking

```
1: **Sample** path length
2: **Look up** location to get $\Sigma_t(\mathbf{r})$
3: $P_{\text{real}} \leftarrow \frac{\Sigma_t(\mathbf{r})}{\Sigma_{\text{maj}}}$
4: **Sample** random number $\xi \in [0, 1)$
5: **if** $\xi < P_{\text{real}}$ **then**                    ▷ Collision is real
6:     **Execute** real collision
7: **else**                                                  ▷ Collision is virtual
8:     **Execute** virtual collision
9: **end if**
```

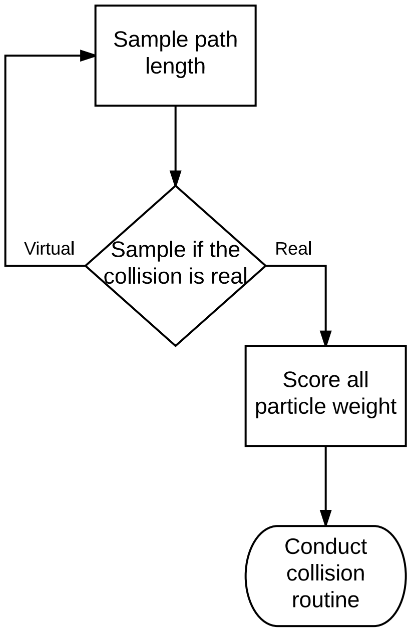

Sample path length
Virtual
Sample if the collision is real
Real
Score all particle weight
Conduct collision routine

Figure 1.5: Delta-tracking algorithm and flow chart.

similar to Woodcock delta-tracking in these regions. Instead of sampling from a majorant cross-section across multiple materials, a maximum cross-section for the single material region is used. Unlike delta-tracking, this requires stopping the neutron at boundaries and resampling path lengths. This algorithm is not modified by the implementation of WDT and none of the input files used to assess WDT use this feature.

1.2 Weighted delta-tracking

Morgan and Kotlyar [5] introduced a method to improve the inefficiencies of Woodcock delta-tracking in the presence of large absorbers. The method, WDT, replaces the rejection sampling algorithm of delta-tracking with a weight reduction algorithm. This process is similar to survival biasing, or implicit capture.

Implicit Statistical Events

We consider a process that can result in multiple outcomes, each with their own probability. One approach is to sample a random variable and determine which outcome actually occurred. If the event occurs many times, we can instead replace the statistical process by using the expected value of the random process [4]. The expected value of a random variable x that can take values $x_1 \dots x_n$ with probabilities $p_1 \dots p_n$, respectively, is given by:

$$E[x] = x_1 p_1 + x_2 p_2 \dots + x_n p_n \tag{1.15}$$

For example, imagine we have a bag with four coins, three worth \$0.25 and one worth \$0.10. The values and their probabilities are:

$$\begin{aligned} x_1 &= 0.25, \quad p_1 = \frac{3}{4} = 0.75 \\ x_2 &= 0.10, \quad p_2 = \frac{1}{4} = 0.25 \end{aligned}$$

For a given draw, we could sample a random value and use the probabilities p_1 and p_2 to determine the coin drawn. Or, we can describe the expected value of the drawn coin:

$$\begin{aligned} E[x] &= x_1 p_1 + x_2 p_2 \\ &= (0.25 \times 0.75) + (0.10 \times 0.25) \\ &= 0.2125 \end{aligned}$$

This value represents the average value of a coin drawn, if we perform many draws. The following Matlab code replicates this procedure:

```
v = 0;
n = 1000;
```

```
for i = 1:n
    xi = rand;
    v = v + (xi < 0.25)*0.1 + (xi >= 0.25)*0.25;
end
```

Although the exact values will vary due to the small sample size, `v/n` $\approx$ 0.2125, as we expected.

This framework is often applied to neutron propagation in a process called "survival biasing." At each collision location, the type of interaction is sampled and the appropriate action taken. In the case of an capture event the neutron is "killed," removed from the simulation. While this accurately reflects the physical reality, it does not produce good statistics. Assuming we are using a collision estimator, the neutron's contribution to our simulation is the history of collisions. Each of these provide a score to that particular collision, contributing to the overall simulation statistics. Capture events that kill neutrons, therefore, are removing the very particles we need to generate better statistics, resulting in the need for many more particles. Mitigating this issue is the goal of survival biasing.

Survival biasing avoids killing neutrons by replacing capture events with an expected outcome. As described above, we do not need to track the outcome of every single event, but can rely on the expected value. This will give us, on average, the outcome of our many events. To eliminate our explicit consideration of capture events, we therefore have two possible event types: capture events, and scattering events. The probabilities p will be given by the ratios of the cross-sections:

$$\begin{aligned} p_{\text{capture}} &= \frac{\Sigma_a}{\Sigma_t} \\ p_{\text{scattering}} &= \frac{\Sigma_s}{\Sigma_t} \end{aligned}$$

Where Σ_a and Σ_s are the macroscopic cross-sections for capture and scattering, respectively.

In the coin example, each had a different monetary value, giving us an expected value when we drew many times. We must introduce something similar for neutrons. It is convention to call this intrinsic value "weight" and represents the importance of the neutron. All neutrons are born with the same weight, usually unity, and are killed when they have zero weight. By that convention an capture event that kills a neutron immediately forces the final weight of the neutron to zero, $w_{f,\text{capture}} = 0$. We can then calculate the expected value of the interaction of a neutron with initial weight w_i:

$$\begin{aligned} E[w_f] &= w_{f,\text{scattering}} p_{\text{scattering}} + w_{f,\text{capture}} p_{\text{capture}} \\ &= w_{f,\text{scattering}} p_{\text{scattering}} \end{aligned}$$

A scattering event merely changes the location of the neutron in energy and angle phase space, leaving its importance unchanged. Therefore, $w_{f,\text{scattering}} = w_i$:

$$E[w_f] = w_i p_{\text{scattering}}$$

Now every collision is considered to be a non-capture event. The neutron continues with a lower weight, proportional to the probability that the event was scattering. The weight lost in the collision is scored as capture:

$$\begin{aligned} S_{\text{capture}} &= E[w_i - w_f] \\ &= E[w_i] - E[w_f] \\ &= w_i - w_i p_{\text{scattering}} \\ &= w_i(1 - p_{\text{scattering}}) \end{aligned}$$

A similar method will be used by weighted delta-tracking.

Russian Rouletting

When the survival biasing routine described in Sec. 1.2 is used, the loss of neutrons is entirely reliant on leakage from the problem or fission events. Neutrons will continue to undergo collisions and subsequent weight reduction until they have a very low weight. These particles will only contribute small amounts to our statistics, so tracking them is computationally inefficient. To mitigate this, a lower cutoff for the weight is introduced. Once neutrons are below this weight, they have a chance of being killed by a Russian Rouletting routine. The general algorithm is shown in Algorithm 2. Following a collision, if the weight of the

Algorithm 2 Rouletting Routine

```
1: if weight < weight threshold then
2:     $\xi \leftarrow$ random number $\in [0, 1)$
3:     if $\xi <$ roulette probability then
4:         Kill particle
5:     else
6:         $w_f \leftarrow w_i/$(roulette probability)
7:     end if
8: end if
```

colliding particle is below a defined weight threshold, a random number is sampled. If this number is below a defined rouletting probability, the particle is killed. If the particle survives rouletting, its weight is increased proportional to the rouletting probability. By killing low weight particles, or increasing their weight, the inefficiency of tracking low-weight particles can be reduced.

Weighted delta-tracking

Morgan and Kotlyar [5] introduced a method to improve the inefficiencies of Woodcock delta-tracking in the presence of large absorbers. The method, WDT, replaces the rejection sampling of delta-tracking with a weight reduction. This is similar to the process of implicit capture discussed in Section 1.2.

The WDT method samples the particle path length in the same fashion as Woodcock delta-tracking. As described in Section 1.1, after each path length sampled, the delta-tracking method accepts the collision as real with the probability shown in Eq. 1.11. The WDT method bypasses this rejection sampling by accepting all collisions as real with a subsequent reduction in weight. As discussed in Section 1.2, replacing a statistical event requires calculation of the expectated value. In this case, the two events are a real collision and a virtual collision.

$$E[w_f] = w_{f,\text{real}} P_{\text{real}} + w_{f,\text{virt}} P_{\text{virt}} \tag{1.16}$$

Morgan and Kotlyar examine a 1D test case with absorption. As an absorption event removes the particle, the resulting final weight of a real collision is zero. A virtual collision is rejected, and therefore leaves the weight unchanged. Inserting the appropriate values into Eq. (1.16) gives the expected value of the final weight for an absorption event.

$$\begin{aligned} E[w_f] &= w_{f,\text{real}} P_{\text{real}} + w_{f,\text{virt}} P_{\text{virt}} \\ &= 0 + w_i P_{\text{virt}} \\ &= w_i (1 - P_{\text{real}}) \\ &= w_i \left(1 - \frac{\Sigma_t}{\Sigma_{\text{maj}}}\right) \end{aligned}$$

The particle that is left following the collsion continues propegating as if it underwent a virtual collision. In this case, the absorption is then scored using the expectation value of the score.

$$\begin{aligned} S_{\text{absorption}} &= E[w_i - w_f] \\ &= E[w_i] - E[w_f] \\ &= w_i \left(\frac{\Sigma_t}{\Sigma_{\text{maj}}}\right) \end{aligned}$$

This is implemented by Kotlyar and Morgan in a 1D problem and the results are verified with the analytical solution. The authors point out that a rouletting routine should be implemented when this is used, to prevent the tracking of low-weight neutrons.

1.3 Weighted delta-tracking with scattering

In a scattering event, the weight of the incident particle does not change. Therefore, application of the expectation value as in Section 1.2 results in an expected value of the final weight equal to the initial weight.

$$\begin{aligned} E[w_f] &= w_{f,\text{real}}P_{\text{real}} + w_{f,\text{virt}}P_{\text{virt}} \\ &= w_i P_{\text{real}} + w_i P_{\text{virt}} \\ &= w_i(P_{\text{real}} + 1 - P_{\text{real}}) \\ &= w_i \end{aligned}$$

This doesn't model what we expect. We can view WDT as splitting the weighted neutron into two particles. One carries the real portion of the weight and experiences the collision. The other carries the virtual portion of the weight and continues propagating as if no collision occurred. This works fine for absorption, where the portion that experiences the collision does not propagate, it is immediately killed and scored. But, when the neutron is split into a scattering portion and a virtual portion, no weight is lost because neither of those events change the weight.

Therefore, extension of this methodology to scattering requires duplication of the particle at the point of collsion. The virtual portion of the weight is carried away by a particle that propagates as if no collision has occured, and the real portion is carried away by a particle that undergoes scattering. In problems with scattering, this results in a rapid multiplication of neutrons. When implemented into Serpent 2, this multiplication very quickly filled any available neutron buffer in simulations of a boiling water reactor (BWR), ending the simulation. Another method for handling scattering events must be developed.

Scattering Rejection Sampling

As described in the last section, the WDT method may result in an intractable simulation when applied to scattering. To maintain proper statistics, scattering must take into account the possibility of a virtual collision while using delta-tracking. To achieve this goal, the delta-tracking rejection sampling that had been supplanted by WDT was moved into the scattering subroutine. Therefore, the new routine uses both rejection-sampling and implicit events to account for the possibility of real and virtual collisions. The algorithm is shown in Alg. 3 and a flow chart of the routine is shown in Fig. 1.6

Note that there are two separate scoring events in each collision subroutine: scoring of the actual collision type for calculating specific reaction rates, and scoring of collision itself used by the collision flux estimator. In addition, scoring the fission reaction also encompasses generation of fission neutrons.

In the original delta-tracking routine, the rejection sampling takes place prior to collision type sampling, and collision scoring can occur between the two. This is the routine that Serpent 2 used, prior to modification for the above scheme. By moving the rejection sampling after the collision type sampling, the collision score is no longer agnostic of the type of collision that will occur. Therefore, in the implementation of this routine in Serpent 2, the collision scoring was moved after the collision type sampling.

Algorithm 3 Weighted delta-tracking with scattering

```
1: Sample path length
2: Sample collision type
3: if collision type == (capture or fission) then
4:     Score capture or fission ← $w_i P_{\text{real}}$
5:     Score collision ← $w_i P_{\text{real}}$
6:     $w_f \leftarrow w_i(1 - P_{\text{real}})$
7:     Execute virtual collision
8: else
9:     Sample random number $\xi \in [0, 1)$
10:    if $\xi < P_{\text{real}}$ then                          ▷ Collision is real
11:        Score scattering ← $w_i$
12:        Score collision ← $w_i$
13:        Execute scattering collision
14:    else                                                  ▷ Collision is virtual
15:        Execute virtual collision
16:    end if
17: end if
```

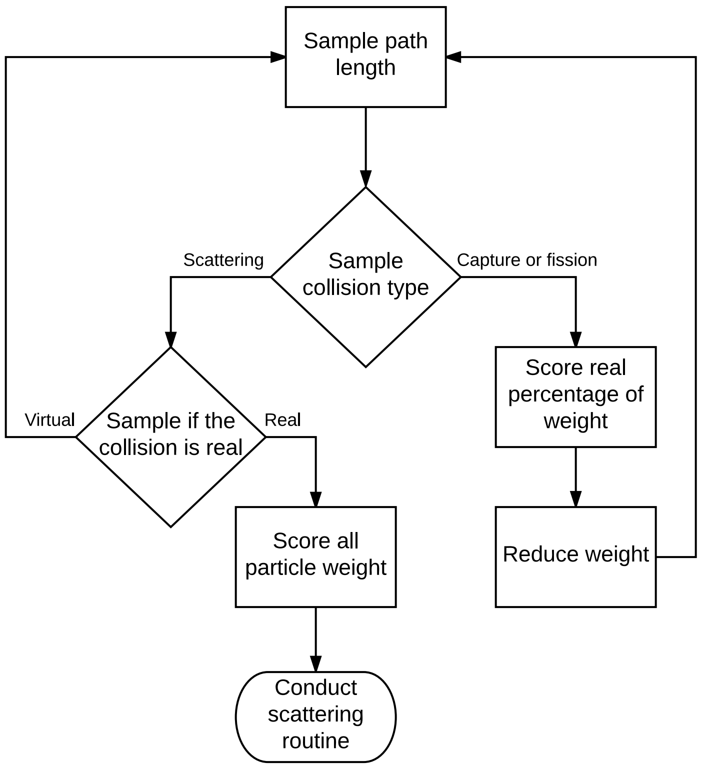

Sample path length
Scattering
Sample collision type
Capture or fission
Virtual
Sample if the collision is real
Real
Score real percentage of weight
Score all particle weight
Reduce weight
Conduct scattering routine

Figure 1.6: Weighted delta-tracking with scattering rejection sampling.

1.4 Implementation in Serpent 2

The Serpent 2 Monte Carlo Code uses a combination of surface tracking, Woodcock delta-tracking, and rejection sampling for non-uniform density distributions. Serpent 2 selects between surface tracking and delta-tracking by examining the ratio of total cross-section to majorant cross-section [2]. In regions where many virtual collisions would occur, the code preferentially switches to ray tracing. This is to avoid the computational inefficiency of processing virtual collisions that provide no statistics. This selection determined by a constant c and the inequality in Eq. (1.17).

$$\frac{\Sigma_t(\mathbf{r})}{\Sigma_{\text{maj}}} > 1 - c \tag{1.17}$$

If this inequality is true, delta-tracking is used, otherwise ray-tracing (referred to as surface-tracking in Serpent 2) is used. By default, the value of c is 0.9, as this was determined to produce the best improvement in run time [2]. We note that this inequality is identical to the value of P_{real}, and this scheme is summarized in Fig. 1.7. Prior to sampling path length, the code tests this ratio for the current neutron position and determines if surface tracking or delta-tracking should be used. If delta-tracking is used, the code then determines if the collision is virtual or real. The value of c can be set in the input file using `set dt`.

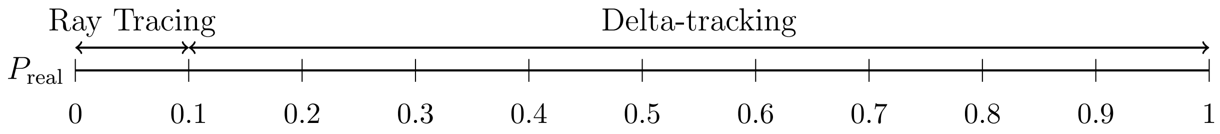


Figure 1.7: Ray-tracing and delta-tracking threshold values in standard Serpent 2.

Weighted delta-tracking threshold

WDT is designed to improve the effectiveness of delta-tracking when the change of a virtual collision is high. We can hypothesize that the algorithm will benefit in the regime where P_{real} is low. At high values of P_{real}, a majority of the weight of the incoming particle is scored. This leaves the particle that undergoes a virtual collision with a very low weight, relying on the rouletting routine to prevent causing computational inefficiency. These two situations imply that there is a region between low and high values of P_{real} where WDT may provide benefit.

A new input parameter line `set wdt` is used to indicate that WDT is to be used, and can also set the value of the WDT threshold t_{wdt}. This is the ratio below which WDT will be used. This scheme is summarized in Fig. 1.8.

$$\text{Mode} = \begin{cases} \text{Raytracing}, & P_{\text{real}} < 1 - c \\ \text{WDT}, & 1 - c \leq P_{\text{real}} < t_{\text{wdt}} \\ \text{Delta} - \text{tracking}, & P_{\text{real}} \geq t_{\text{wdt}} \end{cases}$$

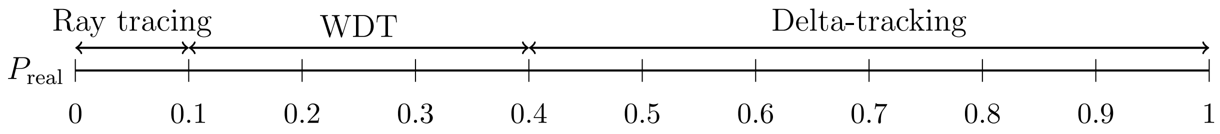


Figure 1.8: Implemented selection scheme for ray-tracing, weighted and normal delta-tracking. Shown using the values of $(1 - c) = 0.1$ and $t_{\text{wdt}} = 0.4$.

Chapter 2

Results

We expect that the WDT method will improve the statistics of Serpent 2 calculations. With normal delta-tracking, virtual collisions provide no statistical benefit, and are therefore an inefficient use of computational resources. Virtual collisions that results in absorption events do contribute to statistics when using WDT. Therefore, we expect an improvement in the results of a Serpent 2 simulation. In this section, we will describe how performance in Serpent 2 is quantified, describe the three test cases, and the results of those test cases.

2.1 Performance metrics

Figure of Merit

Monte Carlo codes such as Serpent 2 run many batches of a single simulation, calculating the mean of values of interest $\hat{x}$ across the many runs. By the central limit theorem, we know that the means across many simulations will form a normal distribution with variance $\sigma^2(\hat{x})$. We are interested in reducing the variance of the final value which will provide more confidence in the simulation results. In general, running the simulation more times will reduce this variance at the cost of requiring more computation time. The variance is proportional to the number of cycles n:

$$\sigma^2(\hat{x}) \propto \frac{1}{n} \tag{2.1}$$

Serpent 2 reports the standard deviation of the mean $\sigma(\hat{x})$, proportional to $n^{-1/2}$. The standard deviation plotted against cycles is shown in Fig. 2.1 on a log-log plot. As expected, we see a linear evolution with a slope of 1/2 once enough cycles have been executed to give good statistical calculations. We can always reduce the variance by running the simulation longer; a measure of our simulation quality, therefore, should be independent of cycles. We establish a figure of merit (FOM) that is independent of n. The standard FOM used by

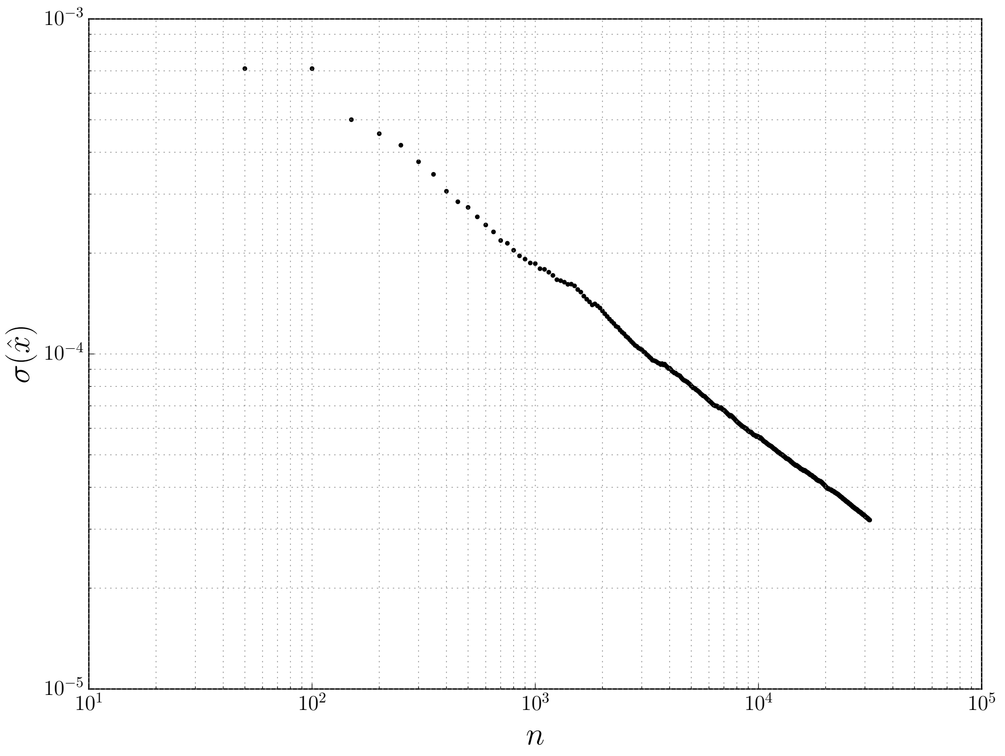


Figure 2.1: Variance in ϕ_∞ for the fast-neutron group as a function of Serpent 2 cycle, with WDT threshold of 0.2.

most simulations, described by Lewis and Miller [1], is shown in Eq. (2.2):

$$\text{FOM} = \frac{1}{\sigma(\hat{x})^2 T} \tag{2.2}$$

where T is the runtime of the simulation, proportional to the number of cycles $T \propto n$. Plugging this and Eq. (2.1) into Eq. (2.2):

$$\text{FOM} = \frac{1}{\sigma(\hat{x})^2 T} = \frac{n}{C_1} \cdot \frac{1}{C_2 \cdot n} = C_3$$

where C_1, C_2, C_3 are constants. As we can see, we expect the FOM to be a constant value, independent of the number of cycles n. A higher FOM indicates higher accuracy per computation time, and therefore a more efficient algorithm. Serpent 2 collects scores for each cycle (or batch of cycles) and outputs the sample mean and standard deviation [6]. This enables comparison of Serpent 2 running with and without WDT to determine the efficiency of the algorithm.

Convergence

As discussed in the previous section, the FOM should be independent of cycle number n. In reality, it takes enough cycles for good statistics to develop for the FOM to converge to its final value. As seen in Fig. 2.2, the FOM begins to converge around 1×10^4 cycles in this example. Statistical variations in the standard deviation seen in Fig. 2.1 are amplified by squaring and inverting to calculate FOM.

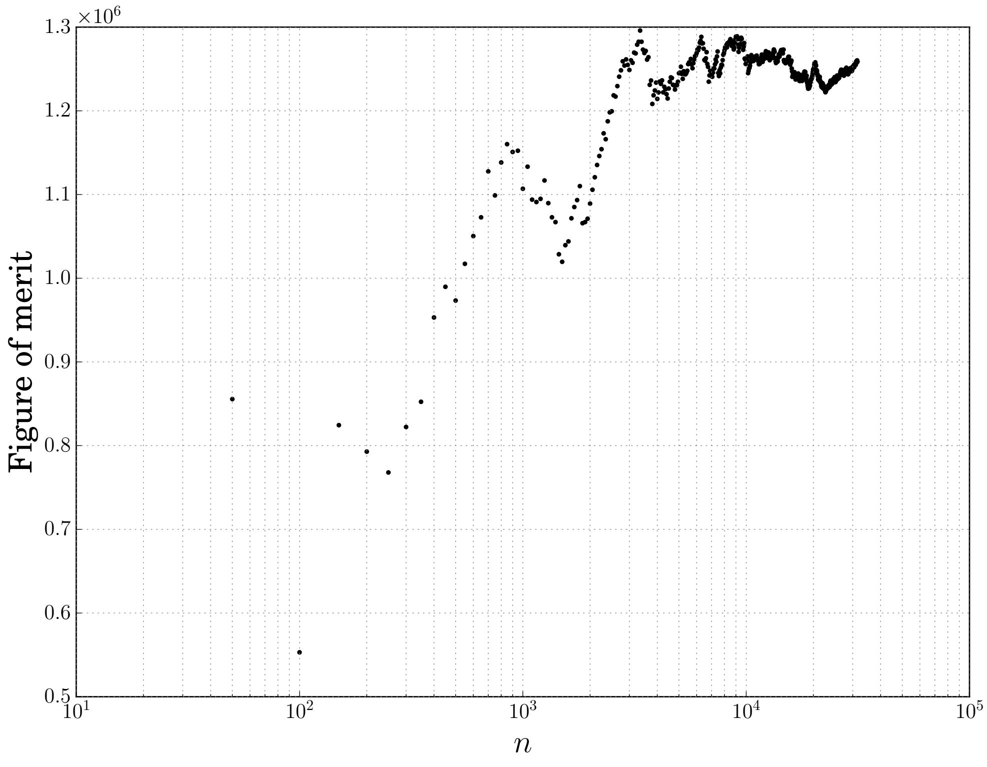


Figure 2.2: Figure of merit in ϕ_∞ for the fast-neutron group as a function of Serpent 2 cycle, with WDT threshold of 0.2.

Analysis Technique

Serpent 2 generates a single output file as the simulation runs. The parameters of interest are overwritten as their mean value and standard deviation are updated. This is sufficient to calculate the final value of the FOM which gives an indication of the quality of the simulation. In addition to this, figures such as those shown in Sec. 2.1 are useful to verify that the FOM has in fact converged, and that the variance is reducing as expecting. Therefore, in addition to changing the Serpent 2 source code to include the WDT method, it was modified to generate uniquely named output files at the end of each batch. Processing these files enables us to generate to above graphs.

As seen in Fig. 2.2, the final FOM is not a single value, but may oscillate for a large number of cycles. Therefore, we calculated an average FOM value for each run, to allow comparison of runs without requiring a visual inspection of each graph.

2.2 Test Cases

Three test cases were selected to test the FOM values for the WDT method. These are a pressurized water reactor (PWR) pin cell, a fast reactor pin cell, and a homogenous fuel element based on the composition of the Transient Reactor Test Facility (TREAT) fuel elements at the Idaho National Lab (INL). All test cases were run on the Abacus server at UC Berkeley, the specifications of the server are shown in Tab. 2.1.

Add table of abacus specs

Parameter	Specification
Place	Holder

Table 2.1: Abacus server specifications

Pressurized water reactor pin cell

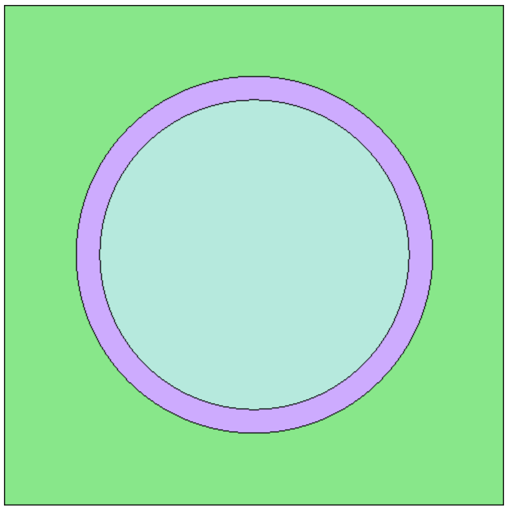
The PWR pin cell was chosen from the Serpent 2 validation input files provided on the VTT Technical Research Centre of Finland (VTT) Serpent webpage [7]. The geometry and physical parameters are shown in Fig. 2.3. The fuel is a 2.68 w/o enriched UO_2 mixture, with Zircalloy cladding, light water moderation, and reflective boundary conditions. We ran the simulation with two energy groups with the group boundary at 0.625 eV.

Fast reactor pin cell

We adapted the fast reactor pin cell from another example provided in the Serpent validation files [7]. This is a lead cooled pin cell with Mixed Oxide (MOX) fuel containing Uranium, Plutonium and a small amount of Americium. The cladding is stainless steel. The lattice is hexagonal, and has reflective boundary conditions. As with the PWR pin cell, we ran the simulation with two energy groups with the group boundary at 0.625 eV.

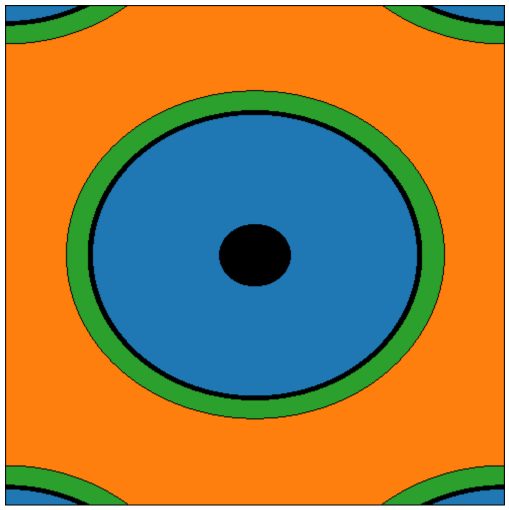
Homogenous fuel element

The third test case is derived from the TREAT reactor at INL. Work at INL focuses on using deterministic solvers to model the core, the Serpent 2 Monte Carlo code is used to generate the cross-sections for these solvers. We used a homogenized fuel element that replicates the material but not geometry of the TREAT fuel. The simulation uses eleven energy groups, which replicates the groups used by the deterministic solvers.



Parameter	**Value (cm)**
Fuel radius	0.412
Cladding radius	0.475
Pitch	1.33

Figure 2.3: Pressurized water reactor geometry and physical parameters. The fuel is shown in light blue, the cladding in purple and the coolant is green.



Parameter	**Value (cm)**
Void radius	0.1
Pellet radius	0.45
Inner cladding radius	0.465
Outer cladding radius	0.525
Pitch	1.789

Figure 2.4: Fast pin cell geometry and physical parameters. The pellet is shown in blue, the cladding green, and the coolant orange. Black regions are voids.

Bibliography

[1] E. E. Lewis and W.F. Miller, Jr. *Computational Methods of Neutron Transport.* American Nuclear Society, 1993.

[2] Jaakko Leppänen. Performance of Woodcock delta-tracking in lattice physics applications using the Serpent Monte Carlo reactor physics burnup calculation code. *Annals of Nuclear Energy*, 37:715–722, 2010.

[3] Jaakko Leppänen. Modeling of Nonuniform Density Distributions in the Serpent 2 Monte Carlo Code. *Nuclear Science and Engineering*, 174:318–325, 2013.

[4] Iván Lux and László Koblinger. *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations.* CRC Press, 1991.

[5] L.W.G. Morgan and D. Kotlyar. Weighted-delta-tracking for Monte Carlo particle transport. *Annals of Nuclear Energy*, 85:1184–1188, 2015.

[6] Toni Kaltiaisenaho. Statistical Tests and the Underestimation of Variance in Serpent 2. Technical Report VTT-R-00371-14, VTT Technical Research Centre of Finland, 2014.

[7] VTT Technical Research Centre of Finland. VTT Serpent 2. `http://http://montecarlo.vtt.fi/`. Accessed: 2017 April 18.

[8] E.R Woodcock et al. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. *ANL-7050. Argonne National Laboratory.*, 1965.