Chapter 3 Event-based Particle Transport

The choices that we will be normally making in the course of following a particle in a Monte Carlo transport calculation are

1. Particle initial position
2. Particle initial energy
3. Particle initial direction
4. Distance to next collision
5. Type of collision
6. Outcome of a scattering event (new energy and directions)

3.1 Choosing initial position

The basic mathematical approach to picking the initial position in three-dimensional space is to relate three random numbers (ξ_1, ξ_2, ξ_3) to the three position coordinates—usually either Cartesian (x, y, z) or cylindrical (r, θ, z) or spherical (r, θ, ϕ)).

To do this a "volume" in random number space is equated with a fractional volume in the physical space, e.g., in Cartesian:

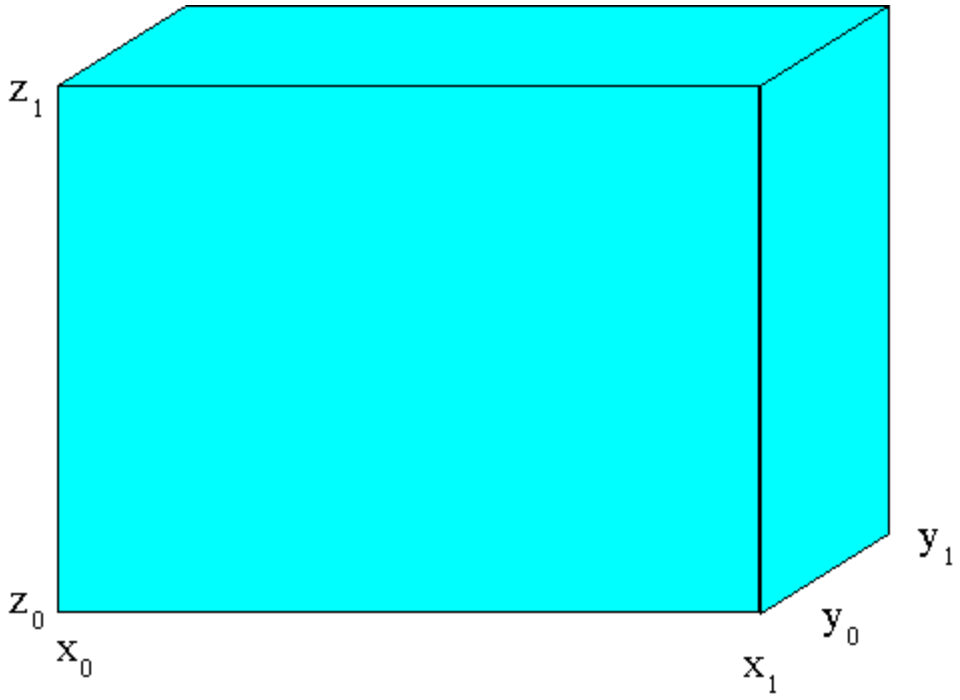
$$d\xi_1\, d\xi_2\, d\xi_3 = \frac{dV}{V_{total}} = \pi(x)dx \cdot \pi(y)dy \cdot \pi(z)dz \tag{3-1}$$

to obtain a separable set of PDFs for sampling the three coordinates (e.g., *x*, *y*, and *z*).

So the hard work comes from precisely defining the volume element dV in terms of differentials of the three position coordinates. Theoretically, this could be done in any of the 13 orthogonal geometries used in mathematical physics, but we will restrict ourselves to the three that MCNP uses: Cartesian, cylindrical, and spherical.

Cartesian coordinate system

The classic shape in Cartesian coordinate systems is a right parallelpiped (i.e., 3D rectangle) in (x,y,z) with upper and lower limits of x_0 and x_1 in x, y_0 and y_1 in y, and z_0 and z_1 in z:



A differential volume element would be defined by:

$$dV = dx\,dy\,dz \tag{3-2}$$

Using our mathematical form gives us:

$$d\xi_1\,d\xi_2\,d\xi_3 = \frac{dV}{V_{total}} = \frac{dx}{(x_1 - x_0)} \cdot \frac{dy}{(y_1 - y_0)} \cdot \frac{dz}{(z_1 - z_0)} \tag{3-3}$$

Taking advantage of the separability, we can assign ξ_1 to x, ξ_2 to y, and ξ_3 to z to get:

$$d\xi_1 = \frac{dx}{(x_1 - x_0)} = \pi(x)\,dx$$

$$d\xi_2 = \frac{dy}{(y_1 - y_0)} = \pi(y)\,dy$$

and

$$d\xi_3 = \frac{dz}{(z_1 - z_0)} = \pi(z)\,dz \tag{3-4}$$

Looking at the rightmost equal sign of the three above equations, we immediately have the PDF (with an obvious upper and lower limit for each variable):

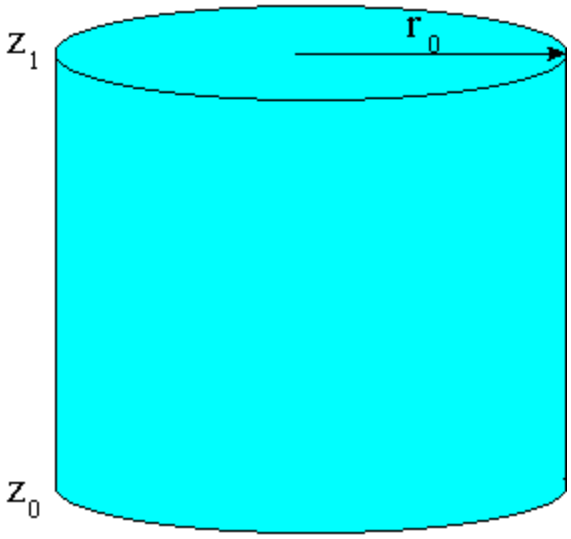
$$\begin{aligned} \pi(x) &= \frac{1}{(x_1 - x_0)} \quad ; x_0 < x < x_1 \\ \pi(y) &= \frac{1}{(y_1 - y_0)} \quad ; y_0 < y < y_1 \\ \pi(z) &= \frac{1}{(z_1 - z_0)} \quad ; z_0 < z < z_1 \end{aligned} \tag{3-5}$$

This means, of course, that each of the coordinates x, y, and z would be chosen according to constant distributions over their respective domains, giving us:

$$\begin{aligned} x &= x_0 + \xi_1 (x_1 - x_0) \\ y &= y_0 + \xi_2 (y_1 - y_0) \\ z &= z_0 + \xi_3 (z_1 - z_0) \end{aligned} \tag{3-6}$$

Cylindrical coordinate system

For a cylinder of radius r_0 and z limits z_0 and z_1:



The volume element is:

$$dV = \left(rd\theta\right)drdz = rdr\,d\theta\,dz \tag{3-7}$$

Where θ is the azimuthal angle, which goes from 0 to 2π .

Following the same path as before gives us:

$$d\xi_1\,d\xi_2\,d\xi_3 = \frac{dV}{V_{total}} = \frac{rdr\cdot d\theta\cdot dz}{\pi r_0^2(z_1 - z_0)} \tag{3-8}$$

The distributions for each of the dimensions are:

$$\begin{aligned}
\pi\left(r\right) &= \frac{r}{r_0^2/2} \quad ;\, 0 < r < r_0 \\
\pi\left(\theta\right) &= \frac{1}{2\pi} \quad ;\, 0 < \theta < 2\pi \\
\pi\left(z\right) &= \frac{1}{(z_1 - z_0)} \quad ;\, z_0 < z < z_1
\end{aligned} \tag{3-9}$$

(where normalization of the PDFs guided us to how much of the denominator was "assigned" to each of the variables).

Use of these PDFs over the domains of the dimensions would result in the following equations for choosing the position variables:

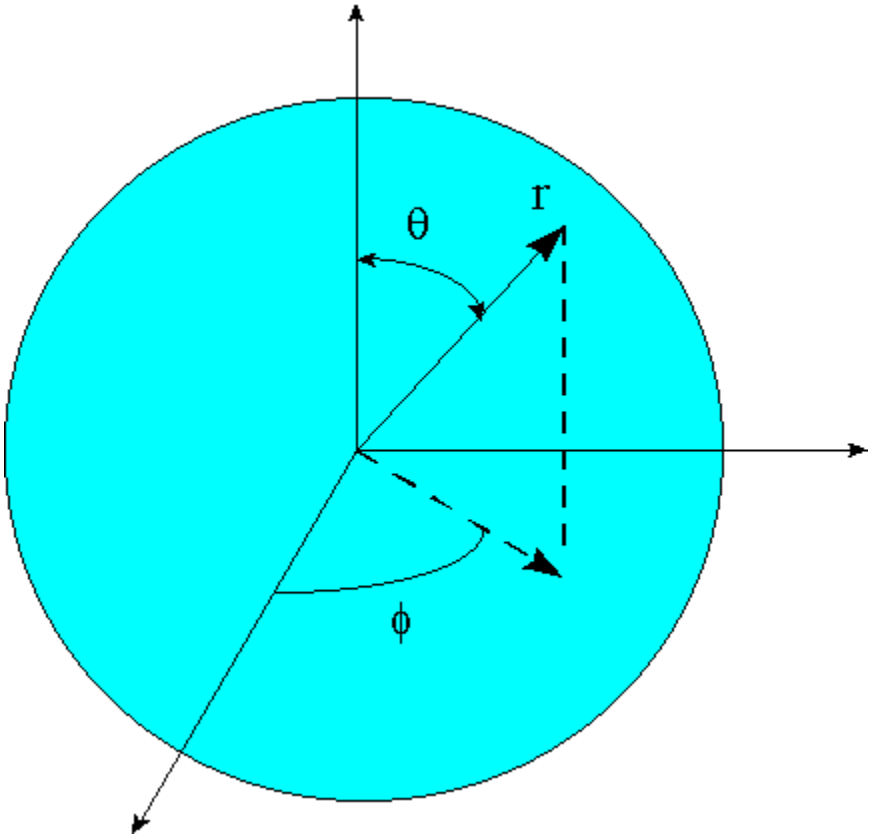
$$\begin{aligned}
r &= r_0\sqrt{\xi_1} \\
\theta &= 2\pi\xi_2 \\
z &= z_0 + \xi_3\left(z_1 - z_0\right)
\end{aligned} \tag{3-10}$$

In the Cartesian coordinate system (that most Monte Carlo codes use) these would be translated into:

$$\begin{aligned}
x &= r\cos\theta \\
y &= r\sin\theta \\
z &= z
\end{aligned} \tag{3-11}$$

Spherical coordinate system

For a sphere of radius r_0 with dimensions r (distance from origin), θ (polar angle), and ϕ (azimuthal angle):



The volume element is:

$$dV = \left(r\sin\theta d\phi\right)\left(rd\theta\right)\left(dr\right) = r^2 dr\sin\theta d\theta\, d\phi \tag{3-12}$$

Analogous to the previous two examples, we have:

$$d\xi_1\, d\xi_2\, d\xi_3 = \frac{dV}{V_{total}} = \frac{r^2 dr\cdot\sin\theta d\theta\cdot d\phi}{\frac{4}{3}\pi r_0^3} \tag{3-13}$$

The distributions for each of the dimensions, then are:

$$\pi(r) = \frac{r^2}{r_0^3/3} \quad ; 0 < r < r_0$$
$$\pi(\theta) = \frac{\sin\theta}{2} \quad ; 0 < \theta < \pi \tag{3-14}$$
$$\pi(\phi) = \frac{1}{2\pi} \quad ; 0 < \phi < 2\pi$$

Use of these PDFs over the domains of the dimensions would result in the following equations for choosing the position variables:

$$r = r_0 \sqrt[3]{\xi_1}$$
$$\theta = \cos^{-1}(1 - 2\xi_2) \tag{3-15}$$
$$\phi = 2\pi\xi_3$$

In the Cartesian coordinate system, these would be translated into:

$$x = r\sin\theta\cos\phi$$
$$y = r\sin\theta\sin\phi \tag{3-16}$$
$$z = r\cos\theta$$

Choosing an initial point from multiple sources

For a situation in which source particles are chosen from multiple source (possibly of various shapes, sizes, and source rate density), the user should apply a probability mixing strategy whereby:

7. A source is chosen from the multiple sources using the fractional source rate in each source (in units of particles/sec) as a discrete PDF used to choose one source.

   [NOTE: Pay attention to the fact that the choice between sources is not based on particles/volume/sec but on particles/sec. You have to multiply by the volume of each volumetric source (and area of each area source and length of each line source).]

8. A point within the chosen source is picked using the appropriate shape's equations from above.

Non-uniform spatial distributions

One additional consideration is what should be done if the spatial source distribution is not uniform within a volumetric source. In this case, the PDFs for the individual dimensions would be multiplied by the non-uniform distribution.

Example: How would you choose a point inside a spherical source if the source is distributed in volume according to $f(\vec{r}) = \dfrac{r^2}{\sin\theta}$?

Answer: In this case, the probability for a differential volume element would be:

$$f(\vec{r})\,dV = \frac{r^2}{\sin\theta} r^2 dr \sin\theta d\theta\, d\phi$$
$$= r^4 dr\, d\theta\, d\phi$$

The (unnormalized) PDFs for each of the dimensions, then would become:

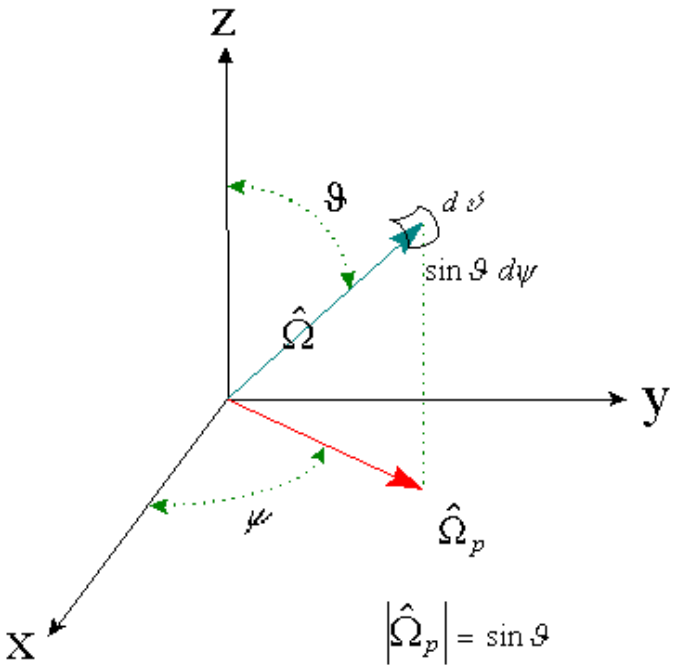
$$\pi(r) = r^4$$
$$\pi(\theta) = 1$$
$$\pi(\phi) = 1$$

Use of these PDFs over the domains of the dimensions would result in the following equations for choosing the position variables.

$$r = r_0 \sqrt[5]{\xi_1}$$
$$\theta = \pi\xi_2$$
$$\phi = 2\pi\xi_3$$

3.2 Choosing initial energy and distribution

The choice of direction uses a 2D version of the technique we used in the previous section, based on probabilities on $d\Omega$, which is a differential element of solid angle on the surface of a unit sphere:



with the value:

$$d\Omega = \sin\theta d\theta\, d\phi \tag{3-17}$$

where we note that the specification of the polar axis to be the z axis in this figure is **completely arbitrary**. The polar axis can be oriented in any direction that the analyst desires.

If we define $\omega \equiv \cos\theta$, this becomes:

$$d\Omega = -\, d\omega\, d\phi \tag{3-18}$$

where the minus sign is present because ω decreases as θ increases. (We will reverse it from here on and let ω vary from its minimum to maximum, like the other variables (i.e., effectively redefining the polar angle to go from south pole to north pole).

Equating a two dimensional "area" of "random number space" to this relative area:

$$d\xi_1\, d\xi_2 = \frac{d\Omega}{4\pi} = \frac{d\omega}{2}\frac{d\phi}{2\pi} \tag{3-19}$$

gives us a dimensional PDFs of:

$$\pi(\omega) = \frac{1}{2} \; ; \; -1 < \omega < 1$$
$$\pi(\phi) = \frac{1}{2\pi} \; ; \; 0 < \phi < 2\pi \tag{3-20}$$

Since ω varies from -1 to 1 and ϕ from 0 to 2π, the resulting equations for the variables are:

$$\omega = 2\xi_1 - 1$$
$$\phi = 2\pi\xi_2 \tag{3-21}$$

Generally, Monte Carlo methods require directions in the form of direction cosines, which would be:

$$u = \Omega_x = \sqrt{1-\omega^2}\cos\phi$$
$$v = \Omega_y = \sqrt{1-\omega^2}\sin\phi \tag{3-22}$$
$$w = \Omega_y = \omega$$

Particle initial energy

Generally, choice of the initial particle energy is based on either a continuous, discrete, or multigroup source spectrum.

If the source distribution is **continuous**, the particular distribution (in units of eV^{-1} or MeV^{-1}) must be dealt with in the usual ways -- either by a direct approach (if the distribution can be integrated and inverted) or with a rejection method.

If the source distribution is **discrete** - which is common for photon production from decay of radioactive sources - the data is usually in the form of particular particle energies coupled with the **yield** (i.e., the percentage of decay events that produce a gamma of the particular energy). In this situation, the yield values for the various possible emitted energies serve as the probabilities (π_i) used for choosing from a discrete distribution.

Similarly, if the source distribution is in **multigroup** form, the individual source contribution in a given group is the integrated source over the group (and therefore is not a distribution in "per unit energy" units). Therefore, the individual group source values are exactly analogous to discrete yields, so would be used as the π_i probabilities in a discrete distribution.

3.3 Choosing distance to next collision

For an infinite medium (constant cross section), the probability of collision inside a differential path element dx a distance *x* from the previous collision is:

$$\begin{aligned}\pi(x)\,dx = &\text{ Probability of next collision in dx}\\ =&\text{(Probability particle travels distance x without}\\ &\text{collision) } \times \text{ (Probability it subsequently collides in dx)}\\ =&\left(e^{-\Sigma_t x}\right)\times\left(\Sigma_t\,dx\right)\end{aligned} \tag{3-23}$$

Therefore the PDF is:

$$\pi(x) = \Sigma_t\, e^{-\Sigma_t x} \tag{3-24}$$

which is already normalized over the domain $(0,\infty)$.

The associated CDF is:

$$\Pi(x) = \int_0^x \pi(x')\,dx' = \int_0^x \Sigma_t\, e^{-\Sigma_t x'}\,dx' = 1 - e^{-\Sigma_t x} = \xi \tag{3-25}$$

which inverts to give us the formula:

$$x = \frac{-\ln(1-\xi)}{\Sigma_t} \tag{3-26}$$

Notice that if we define the "optical path length", τ ,as:

$$\tau = \Sigma_t x \tag{3-27}$$

(which corresponds to the number of **mean free paths** traveled) we can use:

$$\tau = -\ln(1-\xi) \tag{3-28}$$

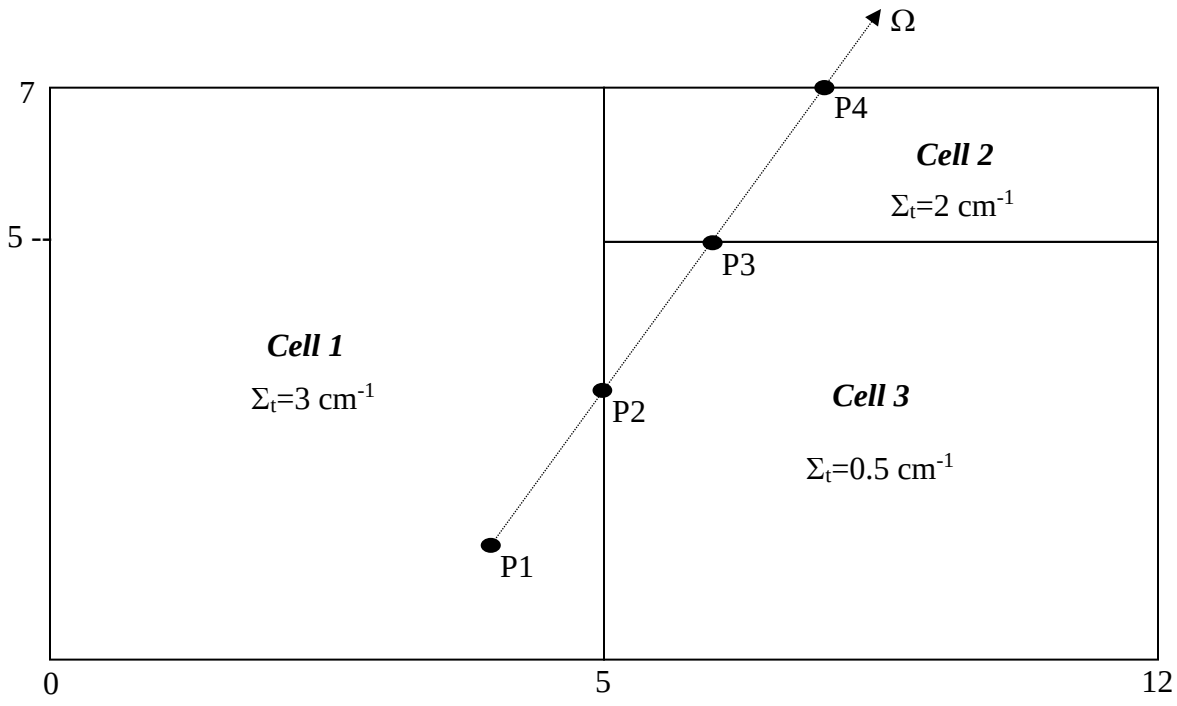
This can be shown to be true for finite media (i.e., with changing total cross section). Once the number of mean free paths have been chosen using this relation, the translation into actual distance is done piece-wise (as the particle travels through different media).

Since $1-\xi$ is "as random" as ξ, it is possible to simplify this to:

$$\tau = -\ln \xi \tag{3-29}$$

(Actually, though, I don't usually do this. As we saw in an earlier section, the current generation of linear congruential generators can give a zero but not a one; therefore, subtracting the random number from one keeps me from having to deal with the natural logarithm of 0.)

Example: A particle emerges from a collision at point $\vec{r} = (4,1,0)$ travelling in direction $\vec{\Omega} = (0.6, 0.8, 0.0)$ (see figure below) with the indicated total cross sections. It is determined that it will travel 8 mean free paths (a rare event) to the next collision. At what (x,y,z) point will this collision take place?



Answer: The path of the particle can be described using the initial position and direction cosines as:

$$x = 4 + 0.6d$$
$$y = 1 + 0.8d$$
$$z = 0 + 0.0d$$

where *d* is the distance that the particle has traveled (in THREE dimensions, although I have defined this problem to have the particle travel in the x-y plane).

We will attack this problem in segments with the path through each homogeneous material being a segment.

Segment 1

The particle begins its journey in Cell 1 with total cross section of 4. In the direction it is travelling, it will enter the next region at the point of intersection of its line of travel and the plane with x equal to 5, therefore we have two equations for x:

$$x = 5 = 4 + 0.6d$$

and *0.6* is the x-direction cosine. Therefore the point of intersection corresponds to *d=1.666…*, which is at the point:

$$P_2 : \left(4 + 0.6d, 1 + 0.8d, 0 + 0d\right) = (5, 2.33\overline{3}, 0)$$

Travelling 1.666… cm in a medium with a total cross section of 3 cm^{-1} corresponds to 5 mean free paths; since it was initially determined that the particle would travel 8 mean free paths, it has 3 mean free paths to go.

Segment 2

Once the particle reaches P2, its distance to the next intersection is the point where the line of travel intersections the plane *y=5*, which will be after traveling:

$$y = 5 = 1 + 0.8d$$
$$d = 5$$

which is at the point:

$$P_3 : \left(4 + 0.6d, 1 + 0.8d, 0 + 0d\right) = (7, 5, 0)$$

The travel of (5-1.666…=3.333…) cm in a Cell 3, with I total cross section of 0.5 is a distance of 1.666… mean free paths; since the particle had 3 mean free paths yet to go, it now has 1.333… mean free paths yet to go.

Segment 3

Having reached Cell 2, the next boundary crossing would occur when the line of travel intersections with the plane with the equation y=7; this crossing will occur when:

$$y = 7 = 1 + 0.8d$$
$$d = 7.5$$

This corresponds to 2.5 cm of travel in Cell 2, with its total cross section of 2 cm^{-1}; this amount of travel WOULD HAVE used up 5 mean free paths. Since the particle only had 1.333… mean free paths left, the particle will only travel a fraction of the distance across the cell equal to its fraction of the mean free path.

That is, the particle will make it only a fraction:

$$f = \frac{1.3\overline{3}}{5} = 0.26\overline{6}$$

This fraction of a distance of 2.5 cm is (0.266…)x(2.5)=0.66… cm; so the total distance traveled is 5.666…, which would put the point of intersection at (7.4,5.5333…,0).

Although this seems like a long calculation, we—as humans—were able to shorten it because we had a figure to go by. We knew from the figure that the first crossing would occur on the surface y=5. MCNP does <u>not</u> process geometry in such a way as to know this fact, so it would have to find the distance to intersection to all four of the surfaces that bound the cell it is in (Cell 1, in this case). It would then determine that line of travel would cross:

- The left boundary (x=0) at a distance given by:

$$x = 0 = 4 + 0.6d \;;\; d = -6\,{}^{2}\!/_{3} \text{ cm}$$

- The bottom boundary (y=0) at a distance given by:

$$y = 0 = 1 + 0.8d \;;\; d = -1\,{}^{1}\!/_{4} \text{ cm}$$

- The right boundary (x=5) at the distance $d = 1\,{}^{1}\!/_{3}$ cm we already calculated; and
- The top boundary (y=7) at a distance given by:

$$y = 7 = 1 + 0.8d \;;\; d = 7.5 \text{ cm}$$

Now, you and I know that this would involve crossing several other cells, but MCNP does <u>not know this</u>. Among these four choices, it picks the smallest positive distance as the surface it will next hit, which turns out to be the right boundary.

The algorithm for this would be:

Choose τ, the total number of mean free paths to travel to the next collision, using:

$$\tau = -\ln \xi \quad \text{or} \quad \tau = -\ln(1-\xi) \tag{3-30}$$

9.

10. Find τ_0, the number of mean free paths to reach the edge of the cell, by finding the minimum positive distance to intersect a surface that bounds the cell, and multiplying this distance by the total cross section in the cll.

11. Are there enough mean free paths left to reach the edge of the cell (i.e., is $\tau > \tau_0$?)
    1. If yes, has the particle reached the edge of the whole problem?
        i. If yes, the particle escaped the problem, so end the particle history.
        ii. If no, subtract τ_0 from τ , find out what the new cell is, and return to step 2 for the new cell.
    2. If no, the next collision occurs a fraction $\frac{\tau}{\tau_0}$ of the distance to the edge of the cell.

3.4 Choosing type and outcome of next collision

Once a collision is known to have occurred, the choice of reaction type is based on the reaction macroscopic cross sections at the particle energy. Generally, the reactions of interests are scattering, fission, and capture, with relative probabilities based on ratios with the total cross section:

$$\Sigma_t(E) = \Sigma_s(E) + \Sigma_c(E) + \Sigma_f(E), \tag{3-31}$$

which gives us probabilities of:

$$\begin{aligned} \pi_{scatter} &= \frac{\Sigma_s(E)}{\Sigma_t(E)} \\ \pi_{capture} &= \frac{\Sigma_c(E)}{\Sigma_t(E)} \\ \pi_{fission} &= \frac{\Sigma_f(E)}{\Sigma_t(E)} \end{aligned} \tag{3-32}$$

We make the choice between reaction types by using these probabilities as a discrete distribution.

Outcome of scattering event

The outcome of a scattering event by a particle with initial energy E is given (formally) by the double differential cross section:

$$\Sigma_s^M\left(E \to E', \mu_0\right) \tag{3-33}$$

where M = material, the primed variables are associated with the particle after the collision, and $\mu_0 = \hat{\Omega} \cdot \hat{\Omega}'$ (the deflection cosine of the particle).

The associated two-dimensional distribution is given by:

$$\pi\left(E', \mu_0\right) = \frac{\Sigma_s^M\left(E \to E', \mu_0\right)}{\Sigma_s^M\left(E\right)} \tag{3-34}$$

The most common way to handle this 2D distribution is to reduce it to a one dimensional choice by integrating over the other dimension. In general, then, a distribution $\pi\left(x, y\right)$ can be reduced to a distribution over *x* by integrating out *y*:

$$\pi\left(x\right) = \int_{-\infty}^{\infty} \pi\left(x, y\right) dy \tag{3-35}$$

and x_i chosen from this distribution.

Now comes the tricky part. Once a value x_i has been chosen from this 1D distribution, we do NOT then go back to the original distribution and integrate out *x* to get a distribution for *y*.

Instead, we SUBSTITUTE the chosen value of x into the original 2D distribution to get the y distribution. That is:

$$\pi\left(y\right) \propto \pi\left(x_i, y\right) \tag{3-36}$$

where I have left in the ~ ("proportional to") because when you plug it in, the result is most likely NOT properly normalized.

Applying this to the particle scatter distribution, our knowledge of the kinematics of scatter generally makes it more convenient for us to integrate out the deflection cosine to get:

$$\pi\left(E'\right) = \int_{-1}^{1} \frac{\Sigma_s^M\left(E \to E', \mu_0\right)}{\Sigma_s^M\left(E\right)} d\mu_0 \tag{3-37}$$

After this exiting energy E_i' is determined, we then choose the deflection angle using:

$$\pi\left(\mu_0\right)=\frac{\Sigma_s^M\left(E\rightarrow E_i',\mu_0\right)}{\Sigma_s^M\left(E\right)} \tag{3-38}$$

It should be noted that for elastic scattering events (and inelastic scattering from known nuclear levels) there is a unique relationship among scattering deflection angle, initial energy, and final energy. This situation would, of course, mean that once the outgoing energy from a scattering event is chosen, the deflection angle can be calculated from the scattering kinematics, with the azimuthal component of scatter chosen uniformly from 0 to 2π and the result direction cosines found from spherical trigonometry relations.

Example: Choose the outgoing energy, $\hat{E}_i'$, and direction cosine, $\hat{\mu}_{0i}$, for an isotropic elastic collision of a neutron of energy *E* with a nucIide *i* with atomic mass A_i (expressed in units of neutron mass).

Answer: For isotropic elastic scattering, the joint distribution for outgoing energy (*E'*) in the LAB system and deflection angle (μ_0) in the COM system is given by:

$$\pi\left(E',\mu_0\right)=\frac{\sigma_s^i\left(E\rightarrow E',\mu_0\right)}{\sigma_s^i\left(E\right)}=\begin{cases}\dfrac{\delta\left(\mu_0-\tilde{\mu}_0\right)}{E\left(1-\alpha_i\right)} & ,\ \alpha_i E<E'<E\\ 0 & ,\ \text{otherwise}\end{cases}$$

where:

$$\alpha_i=\left(\frac{A_i-1}{A_i+1}\right)^2$$

$$\tilde{\mu}_0=\frac{2\dfrac{E'}{E}-1-\alpha}{1-\alpha}$$

So, if we follow the rules for multi-dimensional distributions and integrate the above distribution over μ_0, we get a 1D energy distribution:

$$\pi(E') = \int_{-1}^{1} \pi(E', \mu_0)\, d\mu_0$$

$$= \begin{cases} \dfrac{1}{E(1-\alpha_i)} & , \ \alpha_i E < E' < E \\ 0 & , \text{otherwise} \end{cases}$$

and the resulting μ_0 distribution is found from plugging in the *E'* value chosen (which is a formality, since *E'* is not even IN the distribution, but at least we avoid the "two cases" problem because we know that *E'* it has to fall between *aE* and *E*). After this substitution, we formally get:

$$\pi(\mu_0) \square \ \pi(\hat{E}_i', \mu_0) = \frac{\delta(\mu_0 - \tilde{\mu}_0)}{E(1-\alpha_i)}$$

$$\pi(\mu_0) = \delta(\mu_0 - \tilde{\mu}_0)$$

This Dirac delta simply means that there is really no "choice" involved. A Dirac delta distribution means that the value of the variable that makes the Dirac delta argument equal to zero is the only variable available; it is a "one outcome" discrete distribution and:

$$\hat{\mu}_{0i} = \tilde{\mu}_0$$

For **multigroup** energy representations in which the angular dependence of the group-to-group scattering is represented by a Legendre expansion in deflection angle, the energy group of the outgoing distribution is generally determined –rst -- using the 0th order Legendre cross sec–ons -- and then the direction is chosen from the angular expansion of the chosen groups, using the particular group-to-group Legendre coefficients. (You will get the chance to research this further later.)

3.5 Material crossing variation

According to our long example in a previous section, the distance to next collision is found by first selecting the number of mean free paths that the particle will travel and then (painstakingly) translating this into distance by tracing the path of the particle, possibly through the multiple material boundaries it crosses, and at each crossing subtracting the number of mean free paths used in the previous boundary from the original mean free path "account" until it is "used up."

In practice, however, codes like MCNP use a slightly different approach (because of the cell weighting and weight windows variance reduction techniques you will study later).

Our previous algorithm is modified to:

1. Choose τ , the total number of mean free paths to travel to the next collision, using:

$$\tau = -\ln \xi \quad \text{or} \quad \tau = -\ln(1-\xi) \tag{3-39}$$

2. Find τ_0 , the number of mean free paths to reach the edge of the cell, by finding the minimum positive distance to intersect a surface that bounds the cell, and multiplying this distance by the total cross section in the cell.

3. Are there enough mean free paths left to reach the edge of the cell (i.e., is $\tau > \tau_0$?)
    - If yes, has the particle reached the edge of the whole problem?
        - If yes, the particle escaped the problem, so end the particle history.
        - If no, ~~subtract τ_0 from τ~~ find out what the new cell is, and return to step ~~2~~ 1 for the new cell.
    - If no, the next collision occurs a fraction $\frac{\tau}{\tau_0}$ of the distance to the edge of the cell.

That is, at boundary crossings a new number of mean free paths to the next collision is chosen, rather than depleting the original choice.

The mathematical reason that we have this flexibility is that the exponential distribution used for Decision 4 (distance to next collision) is a unique one: If you start at any value of x other than 0, but normalize the distribution from that point to infinity, you will end up with the same as the original distribution. This idea is often expressed either as the comment "Neutrons do not age" or "Neutrons have no memory". What it means is that the exponential distribution (uniquely) has the property that, as the particle travels, the probability it will collide in the next mean free path is always 1/e; the fact that a particle has "survived" a certain distance into a medium has no bearing at all on the probability of how much farther it will travel before colliding.

3.6 Cell flux estimation techniques

Knowledge of the cell-averaged flux (where a "cell" is a contiguous region containing a single material) is often desired in nuclear engineering applications because of the fact that so many physical effects of interest can be found using it—doses, reactions rates, deposited power, point kinetics parameters, etc.

There are two principal techniques used to estimate the cell average flux: reaction estimators and track-length estimators.

Reaction estimators

In reaction estimators, contribution to the cell flux is triggered when the particle being tracked has an identifiable reaction in the cell of interest. There can, theoretically, be as many varieties of this as there are reaction types, but in practice the predominant ones are collision estimators and absorption estimators.

The mathematical treatment is based on the fact that when a given reaction occurs in the cell, a contribution has been made to the reaction rate in the cell. We might later want to reproduce the reaction rate using a cell-averaged flux (times macroscopic cross section times cell volume), so we convert the reaction rate contribution into a cell flux contribution using the relation:

$$\begin{aligned} \Delta R_{x,cell} &\equiv \text{Cell reaction rate contribution of reaction event of type x} \\ &= \text{w} \equiv \text{Current weight of particle undergoing reaction x} \\ &= \Delta\phi_{cell}\Sigma_{x,cell}V_{cell} \end{aligned} \tag{3-40}$$

or

$$\Delta\phi_{cell} = \frac{\Delta R_x}{\Sigma_{x,cell}V_{cell}} = \frac{w}{\Sigma_{x,cell}V_{cell}} \tag{3-41}$$

So, the cell averaged flux contribution is found by dividing the particle weight by the macroscopic cross section times the cell volume.

Different reaction rates can be used with this formula. The two most common, as previously mentioned, are the collision estimator—in which all collisions "score" and the denominator has the total cross section in it—and the absorption estimator—in which only absorption events "score" and the absorption macroscopic cross section is in the denominator.

Different reaction types *x* will have different statistics, but the same expected value. Compared to the total reaction rate, for example, reaction type *x* will occur Σ_x/Σ_t times less frequently but contribute Σ_t/Σ_x times as much each time it occurs, leaving the expected score unchanged.

Track-length estimator

In a track-length estimator, the flux is estimated based on an alternate definition of flux as "track-length per unit volume."

The basis for this definition can be deduced from a units balance on the same familiar cell reaction rate equation:

$$R_{t,cell} = \phi_{cell} \Sigma_{t,cell} V_{cell} \tag{3-42}$$

If we assign the following units to the non-flux variables:

$R_{t,cell}$ = Reaction rate (units of "reactions/unit time")

V_{cell} = Cell volume (units of "unit volume")

$\Sigma_{t,cell}$ = Reactions per distance travelled (units of "reactions/distance travelled by particles")

Performing a units balance results in the fact that the flux units must be:

ϕ_{cell} = Cell flux (units of "distance travelled by particles/unit volume/unit time")

At first (or even second) glance, this seems like a strange unit for flux, but when you think about it, if we have a total cross section of 0.1 cm-1, then we must expect one reaction per ever 10 cm of neutron travel. So, if the expected reaction rate is 10 reactions/sec in a region (no matter what its size is), then it must have been caused by particles travelling 100 cm (on average) in the volume each second. So, total "path segments" laid down by all particles traversing a region divided by the cell volume equals the cell flux.

Another way of looking at the track-length estimator is as a collision estimator that replaces the actual number of collisions with the expected number of collisions. That is, instead of scoring the average distance travelled (a.k.a. "mean free path", $1/\Sigma_t$) divided by volume each time there is a collision, it keeps up with the actual distance travelled (a.k.a. sum of "track lengths" in the material) divided by the volume, which is proportional to how many collisions there should have been.

This is a recurring theme that we will encounter later in our study of variance reduction schemes (i.e., techniques to improve Monte Carlo efficiency): You usually get an increased efficiency if you score the expectation of events occurring rather than scoring actual events. Not always, but most often.

Time

This might be a good time to discuss the issue of time in Monte Carlo calculations. For the most part (except for kinetic Monte Carlo, which keeps up with the instant of birth, time of flight, etc., to solve time-dependent problems), Monte Carlo is a timeless calculation—the distinction

between time-integrated and time-rate tallies is left to the user to keep up with in the numbers specified for the source.

For example, assume a Monte Carlo case is run with a source normalized to 1 (which is automatically done by MCNP and the other standard Monte Carlo codes) and the resulting cell average flux in some cell is 0.1 +/- 0.005. In such a case, it is up to the user to interpret this answer appropriately. If the source strength is one curie (3.7×10^{10} particles/sec) then the cell flux should be interpreted as 3.7×10^{9} particles/cm^2/sec. On the other hand, if the source represents burst that releases 1×10^{17} neutrons, then the results should be interpreted as a fluence of 1×10^{16} particles/cm2. The flux "inherits" the time unit from the source time unit—the Monte Carlo code doesn't care.

3.7 Surface and point fluxes

In addition to the volumetric (3D) cell averaged fluxes, Monte Carlo codes can also calculate the average flux on a surface (2D) or the average flux at a point (0D).

> *[NOTE: MCNP also has a ring tally that is, formally, 1D, but it is really a rotated point tally and can only be used on problems with rotational symmetry. We will not work out the details here.]*

Surface flux tally

The surface flux tally is based on a straight-forward application of the track-length estimator on a surface "cell" of area A and infinitesimal thickness t:

Because the particle passes through at an oblique angle, its path through the thin surface element is stretched by the secant (inverse of cosine). The resulting contribution to the surface flux is then:

$$\Delta\phi = \frac{\text{Track length}}{\text{Volume}} = \frac{\Delta t \Big/ \left|\vec{n}\cdot\vec{\Omega}\right|}{A\Delta t} = \frac{1}{A\left|\vec{n}\cdot\vec{\Omega}\right|} \tag{3-43}$$

Where $\vec{\Omega}$ is the unit vector in the direction of travel and $\left|\vec{n}\cdot\vec{\Omega}\right|$ is a unit vector perpendicular to the surface at the point of crossing.

Point flux tally

Determining the flux at a point is an impossible task for a cell flux tally because the point has no volume. Although it is possible to let the average flux in a small cell (which include the desired point) approximate this value, another approach is taken.

The theoretical approach is similar to the idea of a track-length estimator using (particle distance travelled)x(probability of interactions per cm travelled) to represent the expected number of collision along a path travelled by a particle (in place of actual collisions observed). Once again we are goint to score expected contributions to flux rather than actual collisions or path length contributions.

Consider the case that we want to calculate the flux at a point P.

When the particle begins its life we make three initial choices

- Original position, $\vec{r}_s$
- Original energy, E_s
- Original direction, $\vec{\Omega}_s$

The point flux estimator does its thing between the second and third decisions (which is why energy goes before direction).

Once a position $\vec{r}_s$ and energy E_s have been selected, we score the expected contribution to the flux at point P, which is computed using the traditional equation for the flux from a monoenergetic point source of strength $\Delta\phi_p$:

$$\Delta\phi_p = \frac{\pi\left(\vec{\Omega}_{\vec{r}_s\to\vec{r}_P}\right)}{4\pi\left|\vec{r}_s-\vec{r}_P\right|^2} e^{-\tau\left(\vec{r}_s\to\vec{r}_P,E_s\right)} \tag{3-44}$$

where

$\vec{\Omega}_{\vec{r}_s\to\vec{r}_P}$ ≡ unit direciton vector that points from $\vec{r}_s$ to $\vec{r}_P$

$\pi\left(\vec{\Omega}_{\vec{r}_s\to\vec{r}_P}\right)$ ≡ Probability that particle starts in direction $\vec{\Omega}_{\vec{r}_s\to\vec{r}_P}$

$\left|\vec{r}_s-\vec{r}_P\right|$ ≡ Distance between $\vec{r}_s$ and $\vec{r}_P$

$\tau\left(\vec{r}_s\to\vec{r}_P,E_s\right)$ ≡ Number of mean free paths between $\vec{r}_s$ and $\vec{r}_P$ for particle of energy E_s

$e^{-\tau\left(\vec{r}_s\to\vec{r}_P,E_s\right)}$ ≡ Probability that particle travels from $\vec{r}_s$ to $\vec{r}_P$ without colliding

But we are not through. We must repeat this calculation every time the particle scatters. For example, if the particle travels to $\vec{r}_1$ and has a collision that results in a particle of weight w_1 and energy E_1, we then make another contribution of:

$$\Delta\phi_p = \frac{\pi\left(\vec{\Omega}_{\vec{r}_1\to\vec{r}_P}\right)}{4\pi\left|\vec{r}_1-\vec{r}_P\right|^2} e^{-\tau\left(\vec{r}_1\to\vec{r}_P,E_1\right)} \tag{3-45}$$

This is conceptually complete but has a practical shortcoming: The distance squared in the denominator creates disruptively large contributions from collisions close to $\vec{r}_p$. MCNP deals with this problem by having the user specify an "exclusion radius" within which a track-length estimator is used. Since the cell flux tally previously discussed is formally defined as:

$$\phi_{cell} = \int_{\Delta V} dV \int_0^\infty dE\, \phi\left(\vec{r},E\right) \tag{3-46}$$

(i.e., the same except for the weight function), these cell tallies are computed exactly like the cell-averaged flux tallies but with the tally contribution multiplied by $R_x\left(E_1\right)$ evaluated at the particle energy).

The very same analogy holds for modifying the surface flux tallies and point flux tallies to tally responses or reaction rates: Multiply by the appropriate response function $R_x\left(E\right)$. This works for either cell, surface, or point flux tallies.

3.8 Adding response functions to flux tallies

General tallies build on the flux tallies by making applying specific desired responses, with the total response defined by:

$$\begin{aligned} &R_x\left(E\right)=1 \text{ for cell-averaged flux tally} \\ &R_x\left(E\right)=\Sigma_x\left(E\right) \text{ for tally of reaction rate in cell i, reaction type x} \\ &R_x\left(E\right)=\text{General pre-determined response function for cell i} \\ &\qquad\qquad \text{dose, detector response, kerma, etc.} \end{aligned} \tag{3-47}$$

with the type of tally defined by the $R_x\left(E\right)$ weight function used:

- 1 for cell-averaged flux tallies.
- $\Sigma_x\left(E\right)$ for the tally of a reaction in cell i, reaction type x;
- $R_x\left(E\right)$ for general pre-determined response functions for cell i dose, detector response, kerma, etc.

3.9 K-effective calculations

The particle lifetime described previously assumed a source with known (and unchanging) spatial, energy, and direction distributions; furthermore, subsequent lessons assumed that we wanted to know one or more tallies on a "per source particle" basis. Monte Carlo calculations of k-effective differ from this in several respects:

4. There is no fixed and unchanging source to select from. The source particles are fission neutrons whose spatial distribution depends on where fission events occur. (Lucky for us, the energy and directional distributions of fission neutrons are known and unchanging.) Therefore, the source spatial distribution depends on the neutron flux, so must be determined while the code runs.
5. The main tally of interest is k-effective, the ratio of fission neutrons in a generation to the number in the previous generation.

Because of this generation-to-generation dependency, k-effective calculations proceed somewhat differently from fixed source calculations:

- The fission neutrons are released in "batches" or "generations" of several hundred or more. This is done in order to (hopefully) get a good number of new fission events from which to obtain a good estimate of k-effective from the generation and a good spatial distribution of new fission sites.
- The code gets one estimate of the k-effective per batch, rather than one per neutron history.
- The starting location of fission neutrons in one generation are selected from the fission sites of the preceding generation. This practice has several ramifications:
    1. The user must specify the number of batches to run and the number of neutrons in each batch (instead of just specifying the total number of histories as is true in a fixed source calculation).
    2. The first generation has no "previous generation" to choose sites from, so some method must be implemented in the code to choose the initial fission sites or the user must provide the original fission locations.
    3. As a result, the first fission site distribution is unlikely to give a good estimate of k-effective. Nor is the second. Nor the third, etc. The assumption is made that the fission site distribution will eventually settle down to a realistic distribution. Although it is difficult to determine, the user has to specify a number of generations to skip before the code really starts to keep statistical estimates of k-effective.
    4. The resulting generation-to-generation dependence violates the LLN assumption that each k-effective sample is independent from the other samples.
    5. Because each fission event releases, on average, more than two neutrons, an "analog" simulation would result in fewer fission sites than fission neutrons (~40 sites per 100 fission neutrons). So, either fission sites must be reused regularly (which degrades the statistics of the fission distribution) or some technique must be used to increase the number of fission sites.

3.10 Example: Writing a 2-group transport code.

Actually, I have made it easy for you by giving the all the algorithms in this lesson. Code it and verify that your results are statistically consistent with my results. Be able to justify each line of the code!

Our first step into actual Monte Carlo programming of neutral particle transport will be the development of a simple Java code to solve a particular problem. The problem will be the simplest that I can think of—a 1D slab transport penetration problem in two energy groups.

The analog transport solution for this problem is a series of probabilistic choices that are made following the guidance of Lesson 6 using the problem data.

Our description will be in the following sections:

- A problem description and setup
- An accounting of the overhead parts of the computer code, including statistical setup and treatments
- Details of the coding for the individual choices made during an individual particle history
- Results of running the resulting program

Description of Problem and problem setup

The problem that we will attack is a 50-cm thick 1D slab problem that has a single material throughout, but the source region is only the left-most 5 cm. This distance corresponds to 5 mean free paths in Group 1 and 10 mean free paths in Group 2. The cross sections were chosen so that there is no upscatter from Group 2 to group 1, but downscatter dominates the Group 1 cross sections. The first section of the coding includes a crude drawing of the problem and variable declarations and definition.

```
import java.util.Scanner;
class Slab
{
  public static void main(String[] args)
  {
    int nhist;
//*********************************************************************
//                                                                    *
//       Monte Carlo calculation of a slab:                           *
//                                                                    *
//       |      |                 g                  |                *
//       |      |         -----------------           |                *
//       |      |         Grp 1       Grp 2           |                *
//       |      | Total    0.1        0.2             |                *
//       |      | G1->g    0.05       0.04            |                *
//       |S1 4  | G2->g    0          0.1             |                *
//       |S2 6  |                                     |                *
//       |      |                                     |                *
//       |      |                                     |                *
//       |      |                                     |                *
//       |      |                                     |                *
//       | 5 cm |              45 cm                  |                *
//       |<---->|<----------------------------------->|                *
//       |      |                                     |                *
//                                                                    *
//*********************************************************************
//                                                                    *
```

```
//     Variable definitions:                                              *
//       totxs(ig)   Total cross section in group ig, 1/cm                *
//       scat(ig,jg) Scattering cross section from group ig to jg, 1/cm *
//       totscat(ig) Total scattering in group ig (i.e., sum of scat   *
//                      for all other groups jg                           *
//       sour(ig)    Source in group ig, #/cm3/sec                        *
//       bin(ib)     Bin values                                           *
//                   ib = 1 Left leakage for group 1                      *
//                      = 2 Left leakage for group 2                      *
//                      = 3 Right leakage for group 1                     *
//                      = 4 Right leakage for group 2                     *
//                      = 5 Flux for group 1                              *
//                      = 6 Flux for group 2                              *
//       ig          Current energy group of the particle                 *
//       mu          Current direction cosine of the particle, (-1,1)  *
//       x           Current position of particle                         *
//       dd          Distance to next collision                           *
//       dx          x dimension distance to next collision = dd*mu    *
//       mfp         Mean free paths to next collision                    *
//                                                                        *
//************************************************************************
//                                                                        *
//       Set the source and cross sections                                *
//                                                                        *
//************************************************************************
    double[] totxs={0.1,0.2};
    double[][] scat=new double[2][2];
    scat[0][0]=.05;
    scat[0][1]=0.04;
    scat[1][0]=0.;
    scat[1][1]=0.1;
    double[] sour={4.,6.};
```

In addition to the crude drawing, the variables that will be used in the problem are defined and the cross section data is set.

Problem overhead -- Statistical setup and treatment

In this section, the source and scattering cross section data for the problem is translated into CDFs for use in the discrete probability testing that will be done later.

```
//************************************************************************
//                                                                        *
//       Find total scattering cross section for each group               *
//                                                                        *
//************************************************************************
    int ng=2;
    double[] totscat=new double[2];
    for(int ig=0;ig<ng;ig++)
    {
      totscat[ig]=0.;
      for(int jg=0;jg<ng;jg++)
      {
        totscat[ig]+=scat[ig][jg];
      }
    }
//************************************************************************
//                                                                        *
//       Convert source to CDF                                            *
//                                                                        *
//************************************************************************
    for(int ig=1;ig<ng;ig++)
    {
      sour[ig]+=sour[ig-1];
    }
    for(int ig=0;ig<ng;ig++)
    {
      sour[ig]/=sour[ng-1];
```

```
    }
//*************************************************************************
//                                                                        *
//       Convert scattering cross sections to CDF                         *
//                                                                        *
//*************************************************************************
    for(int ig=0;ig<ng;ig++)
    {
      for(int jg=1;jg<ng;jg++)
      {
        scat[ig][jg]+=scat[ig][jg-1];
      }
      for(int jg=0;jg<ng;jg++)
      {
        scat[ig][jg]/=scat[ig][ng-1];
      }
    }
```

In addition, the bins for the problem are emptied. Because of the Neumann process that is used for particle transport Monte Carlo, we need PROBLEM bins and HISTORY bins. The former are the bins that collect the binned data for the problem as a whole (i.e., that each history contributes to); these use the variables BIN (for the bin data itself) and BIN2 (for the squared data that are needed for determining the standard deviations.

The set of HISTORY bins are needed to collect data as a single particle moves around the problem, possibly leaving and re-entering regions, a given particle's history may involve several "pieces" that add up to the particle's contributions to the statistics.

We use 6 different bins in the problem:

- Bins 1 and 2 for the group 1 and group 2 (respectively) LEFT LEAKAGE.
- Bins 3 and 4 for the two group's RIGHT LEAKAGE
- Bins 5 and 6 for the two group's average flux rates in the problem.

In this section of the coding, we also begin each particle's history; we begin a history by emptying the HISTORY bins, which are kept in the variable TBIN.

```
//*************************************************************************
//                                                                        *
//       Empty the total bins                                             *
//                                                                        *
//*************************************************************************
    double[] bin=new double[6];
    double[] bin2=new double[6];
    for(int ib=0;ib<6;ib++)
    {
      bin[ib]=0.;
      bin2[ib]=0.;
    }
//*************************************************************************
//                                                                        *
//       For each history:                                                *
//                                                                        *
//*************************************************************************
    System.out.println(" No. of histories?");
    Scanner sc=new Scanner(System.in);
    nhist=sc.nextInt();
    for(int ih=0;ih<nhist;ih++)
    {
//*************************************************************************
```

```
//                                                                      *
//         Empty the history bins                                       *
//                                                                      *
//**********************************************************************
    double[] tbin=new double[6];
      for(int ib=0;ib<6;ib++)
      {
        tbin[ib]=0.;
      }
```

Java coding for the individual decisions

Choosing an initial particle position

Since the source is only in the left 5 cm of the problem, we pick the original source location uniformly over this domain.

```
//**********************************************************************
//                                                                      *
//         Find the original position                                   *
//                                                                      *
//**********************************************************************
      double x=Math.random()*5.;
```

Choosing an initial particle direction

Since the source is isotropic, we pick a direction by choosing the cosine of the polar angle uniformly over the domain -1 to 1. Note that the coordinate system has been chosen so that the polar axis points to the right, eliminating the need to keep up with an azimuthal angle.

```
//**********************************************************************
//                                                                      *
//         Find the original direction                                  *
//                                                                      *
//**********************************************************************
      double mu=Math.random()*2.-1.;
```

Since the Group 1 source is 0.4 and the group 2 source is 0.6, we choose the initial group discretely from the two choices with a 40%/60% bias toward group 2. Note that Monte Carlo results are usually on a "per particle" basis. (So, in this problem, the user would have to know that the source adds up to ten, and multiply the answers by 10.)

```
//**********************************************************************
//                                                                      *
//         Find the original energy group                               *
//                                                                      *
//**********************************************************************
      double xsi=Math.random();
      int ig=0;
      for(ig=0;ig<ng;ig++)
      {
        if(xsi < sour[ig])break;
      }
```

Choosing the distance traveled to next collision site and scoring escape events

The choice of distance traveled to the next collision site is divided (as is traditional) into a selection of the number of mean free paths chosen and a translation of this into distance by dividing the number of mean free paths by the total cross section.. In this coding, this distance traveled is multiplied by the direction cosine in the x dimension to get distance traveled in the x axis; this is then added to the current x position to get the new position.

Since all of our bins are triggered by particle movement, coding follows to score:

- a left leakage if x < 0
- a right leakage if x > the problem width (50 cm)

If a leakage occurs, this constitutes the end of the current particle history.

```
//*********************************************************************
//                                                                    *
//         Find how many mean free paths it travels to next position  *
//                                                                    *
//*********************************************************************
      boolean historyThrough=false;
      while(!historyThrough)
      {
        double mfp=-Math.log(1.-Math.random());
//*********************************************************************
//                                                                    *
//           Convert mfp to distance in cm                            *
//                                                                    *
//*********************************************************************
        double dd=mfp/totxs[ig];
//*********************************************************************
//                                                                    *
//           Translate into a new position                            *
//                                                                    *
//*********************************************************************
        double dx=dd*mu;
//*********************************************************************
//                                                                    *
//         If particle has exited the left boundary:                  *
//                                                                    *
//*********************************************************************
        if(x+dx < 0.)
        {
//*********************************************************************
//                                                                    *
//           Contribute to bins 1 or 2                                *
//                                                                    *
//*********************************************************************
          tbin[ig]+=1.;
//*********************************************************************
//                                                                    *
//           Contribute to flux bins 5 or 6                           *
//           End the history                                          *
//                                                                    *
//*********************************************************************
          historyThrough=true;
        }
//*********************************************************************
//                                                                    *
//         If particle has exited the right boundary                  *
//                                                                    *
//*********************************************************************
        else if(x+dx > 50.)
        {
//*********************************************************************
//                                                                    *
//           Contribute to bins 3 or 4                                *
//                                                                    *
//*********************************************************************
          tbin[ig+2]+=1.;
//*********************************************************************
//                                                                    *
//           Contribute to flux bins 5 or 6                           *
//           End the history                                          *
//                                                                    *
```

```
//*********************************************************************
          historyThrough=true;
        }
```

Choosing the collision type and scoring absorptions

If the particle has not leaked by this time, the total path length divided by cell volume is contributed to the flux tally and a decision is made about the collision type. This decision is made based on the collision being a scatter event if a test random number is less than the ratio of scattering to total cross section in the current energy group; if the test random number is greater than the scattering ratio, then an absorption has occurred and the particle history is terminated.

```
//*********************************************************************
//                                                                    *
//          If you got this far, must have been a collision.          *
//            Contribute to the flux bins 5 or 6                      *
//          If it was an absorption collision:                        *
//                                                                    *
//*********************************************************************
        else if(Math.random() > totscat[ig]/totxs[ig])
        {
//*********************************************************************
//                                                                    *
//            End the history                                         *
//                                                                    *
//*********************************************************************
          historyThrough=true;
        }
```

For scattering events, choosing the new direction

If the event was a scatter (which must be true if this part of the coding is reached), a new direction for the outgoing particle is chosen, again uniformly (in cosine) over the domain -1 to 1.

```
//*********************************************************************
//                                                                    *
//          If it was a scattering collision:                         *
//            Pick the new direction                                  *
//                                                                    *
//*********************************************************************
        else
        {
          x+=dx;
          mu=Math.random()*2.-1.;
```

Choosing the new particle energy

The post-scatter energy is chosen by choosing from the discrete distribution determined from the group-to-group scattering cross section PDFs for particles born in the current group. (For example, if the scattering even occurred in group1, there is a 0.05/(0.05+0.05) or 5/9 probability that the particle will emerge in group 1 and a 4/9 probability that it will emerge in group 2.)

After the new particle characteristics are known, we loop back in the coding to the point where the distance traveled is determined and continue.

```
//*********************************************************************
//                                                                    *
//            Pick the new energy group                               *
//                                                                    *
//*********************************************************************
          xsi=Math.random();
          int jg=0;
          for(jg=0;jg<ng;jg++)
```

```
          {
            if(xsi < scat[ig][jg])break;
          }
          ig=jg;
//*********************************************************************
//                                                                     *
//           Return to the point where we find the # of mfps travelled *
//                                                                     *
//*********************************************************************
        }
```

Statistical wrap-up and presentation of final results

At the end of the history, we dump the temporary HISTORY bin information for the just-ended particle history into the PROBLEM bins that are collecting the total information for the Monte Carlo run.

Likewise, once all particle histories have been run, we compute the final best estimate of the value for each bin (using BIN data) and the standard deviation of our result (using both the BIN and BIN2 data).

```
C//*********************************************************************
//                                                                     *
//         If the history is over, dump the history bins into the total *
//                  bins                                               *
//                                                                     *
//*********************************************************************
        for(int ib=0;ib<6;ib++)
        {
          bin[ib]+=tbin[ib];
          bin2[ib]+=tbin[ib]*tbin[ib];
        }
      }
    }
//*********************************************************************
//                                                                     *
//       When the problem is over, print the results for each bin:     *
//                                                                     *
//*********************************************************************
    for(int ib=0;ib<6;ib++)
    {
      double xhat=bin[ib]/(double)(nhist);
      double x2hat=bin2[ib]/(double)(nhist);
      double sample_variance=(double)(nhist)/(double)(nhist-1)*(x2hat-xhat*xhat);
      double sample_sd=Math.sqrt(sample_variance);
      double mean_variance=sample_variance/(double)(nhist-1);
      double mean_sd=Math.sqrt(mean_variance);
      double fsd=mean_sd/xhat;
      System.out.printf(" For bin  %1$1d ==> %2$8.6f +/- %3$8.6f "+
                        "(FSD=%4$8.4f)\n",ib,xhat,mean_sd,fsd);
    }
  }
}
```

Results of problem execution”

Here is the result from running a 1,000,000 history calculation. (The code does not yet have flux estimates—bins 4 and 5—coded; you will have that chance.)

```
 No. of histories?
 For bin  0 ==> 0.153845 +/- 0.000361 (FSD=  0.0023)
 For bin  1 ==> 0.208604 +/- 0.000406 (FSD=  0.0019)
 For bin  2 ==> 0.001108 +/- 0.000033 (FSD=  0.0300)
```

```
For bin  3 ==> 0.000324 +/- 0.000018 (FSD=  0.0555)
For bin  4 ==> 0.000000 +/- 0.000000 (FSD=     NaN)
For bin  5 ==> 0.000000 +/- 0.000000 (FSD=     NaN)
```

Chapter 3 Exercises

3-1. Code and test an algorithm for choosing points uniformly in a 10x20x30 cm cube. Bin your results in 100 equal probability bins.

3-2. Code and test an algorithm for choosing points uniformly in a radius=10, ht=20 cm cylinder. Bin your results in 100 equal probability bins.

3-3. Code and test an algorithm for choosing points uniformly in a radius=10 sphere. Bin your results in 100 equal probability bins.

3-4. Code and test an algorithm for choosing energy from a Watt spectrum. Bin your results in 0.1 MeV wide bins.

3-5. Code and test an algorithm for choosing distance to next collision along a path consisting of (in order) the following segments:

- 1 cm with total cross section of 3 cm^{-1}
- 2 cm with total cross section of 2 cm^{-1}
- 3 cm with total cross section of 1 cm^{-1}
- Rest of infinite medium with total cross section of 0.1 cm^{-1}

3-6. For a neutron with an initial energy of 1 keV, find the expected distribution of energies for neutrons after 3 elastic collisions (Use A=12).

3-7. For an angular distribution given by:

$$S(\mu) = S_0 + S_1\mu + S_2\left(\frac{3\mu^2 - 1}{2}\right)$$

code and test an algorithm for choosing the deflection angles. Set the S coefficients all to 1.

3-8. Get a version of the 2-group example problem up and running. Compare your results to those from the exercise.

3-9. Modify your version of the program to include a collision estimator of the group 1 and group 2 fluxes. Use the particle balance to check your results (i.e., source=leakage+absorption).

3-10. Modify your version of the program to include a track-length estimator of the group 1 and group 2 fluxes. Again use the particle balance to check your results.

3-11. Modify your 2-group code to calculate k-effective. Make the material nu*fission cross sections be:

Group 1: 0.005 cm^{-1}
Group 2: 0.12 cm^{-1}

Answers to selected exercises

Chapter 3
3-11. $k_{eff} = 0.875$