PH3010 MSci Skills Project
X-ray simulation with the Monte Carlo method

1 Introduction

In this project you will investigate the production of x-rays and their interactions with matter using the Monte Carlo method. This is a technique for simulating processes using sequences of random numbers. The first part of the problem is to simulate the production of individual photons, each having a different energy. The energies are chosen 'at random' in such a way that the values follow a given distribution.

We then allow each photon to hit a layer of matter (in the example below, 1 mm beryllium). There is a certain probability for the photon to be absorbed, and this probability depends on the photon's energy. So for each photon we 'toss a coin' (in the computer, of course) to decide whether the photon makes it through. We can then make a histogram of the energies of the photons which penetrate the layer.

Your assignment is to simulate the production of x-ray photons from a synchrotron radiation source and to simulate their passage through a layer of matter. The main objectives include the following:

- simulate random numbers uniformly distributed between 0 and 1;
- use the uniformly distributed numbers to simulate photon energies using the acceptance–rejection technique;
- simulate the absorption of photons in matter;
- make histograms of the energies of the photons which are not absorbed;
- determine the mean energy of photons versus the thickness of the absorber.

In addition, the project can be extended in a number of ways and for a first-class mark it is expected that you achieve some of the extension objectives as well. These will be described further along. Background information on the statistical methods used for the project can be found in Refs. [1, 2, 3] and in the chapters on Probability, Statistics and Monte Carlo from Ref. [4].

2 C++ framework for the project

The project should be implemented in the C++ language using the linux computer linappserv0 of the Particle Physics group. To access linappserv0, you need an X11-capable terminal window on a networked computer. You can get this using the program cygwin from the PCs in the teaching lab, or you can install cygwin on your own laptop (if you have windows) or you can install X11 on a Macintosh. Some further information is provided on the course page for PH4515 (Statistical Data Analysis):

http://www.pp.rhul.ac.uk/~cowan/stat_course.html

The computing notes on the PH4515 page also contain some information on how to program with C++ in the linux environment and on how to use the data analysis package ROOT. ROOT is both a C++ class library as well as an executable program that allows one to display and manipulate histograms. Acquiring some degree of familiarity with ROOT is an important element of this project. The statistics notes from the PH4515 web page also contain useful background information on the Monte Carlo method.

3 The Warm-up project: simpleMC

Before beginning to write your own Monte Carlo program you should first carry out the "warm-up" project simpleMC. The ingredients are located here:

http://www.pp.rhul.ac.uk/~cowan/ph3010/xray/simpleMC/

You should download the files into a separate working directory on linappserv0 along with the "makefile" GNUMakefile. You can copy files directly from the web server into your working directory on linappserv0 using, e.g.,

```
cp ~cowan/WWW/ph3010/xray/simpleMC/* ./
```

Then from your working directory, enter

```
gmake
```

followed by return. This will create the executable program simpleMC. To run the program, type

```
./simpleMC
```

followed by return. This will create an output file called sipleMC.root, which contains a histogram of uniformly distributed random values.

The program simpleMC uses the random number generator supplied by the data analysis package ROOT. ROOT includes a C++ class called TRandom3. As with all ROOT classes it begins with the letter T, and the easiest way to find the web page with the class definition is through google, which will direct you to

http://root.cern.ch/root/html/TRandom3.html

Here you will see that TRandom3 contains a member function Rndm, which generates random numbers that are uniformly distributed between zero and one. In a similar way you can find information about the class TFile, which is used to create the output file and TH1D, which is used for creating histograms.

To look at the histogram stored in the output file simpleMC.root, you can use the executable program root. To run the program on linappserv0, simply type root. From here you can enter commands at the root prompt, but it is much more efficient to store the commands in a file (a "macro"). An example is shown in the file plotUniform.C. The root commands are essentially the same as statements in the C++ language. To execute the macro, type

```
.x plotUniform.C
```

To quit from root type `.q`. As you work through the project you will want to create more sophisticated macros to create different kinds of plots. Further information on root can be found from Ref. [5], which contains links to numerous tutorials and examples.

4 Generating random numbers

Simulation of random processes by the Monte Carlo method can be broken down into two basic steps. First, random numbers are generated which follow a uniform distribution between 0 and 1. Next, these are used to generate random numbers which correspond to physical quantities (such as photon energies), which in general have a different frequency distribution. This second step is described in Sec. 5.

In the program `simpleMC`, uniformly distributed random numbers are generated using the class `TRandom3`, and you can simply continue to use this generator for the project. One of the possible extensions to the project, however, is to write your own random number generator, and so in this section some additional information on this is supplied.

There are many computer algorithms available for producing uniformly distributed random numbers. They produce a sequence of numbers according to a certain rule, and would produce the same sequence if you repeat the procedure. In this sense they are not truly random, and therefore are often called pseudorandom. For our purposes this is just as good.

A simple but effective algorithm is the multiplicative linear congruential generator (MLCG). This generates a sequence of integer values $n_1, n_2, \ldots$ according to the rule

$$n_{i+1} = (an_i) \bmod m. \tag{1}$$

Here the *multiplier* a and *modulus* m are integer constants and the mod (modulo) operator means that you take the remainder of an_i divided by m. The values n_i follow a periodic sequence in the range $[1, m-1]$. The initial value n_0 is called the *seed*. The transformation

$$r_i = n_i/m \tag{2}$$

then gives numbers in the interval $(0, 1)$.

The art of random number generation is to choose a and m so that the resulting values r perform well in various tests of randomness. For a 32-bit integer representation, for example, $m = 2147483399$ and $a = 40692$ have been shown to give good results [6].

You may wish to try implementing your own random number generator using, for example, the MLCG algorithm. Notice, however, that if you implement the MLCG rule given by Eq. (1), the quantity an_i can easily be larger than the maximum value that you can store in a variable of type `int`. There are some tricks needed to avoid an arithmetic overflow, and these are described, e.g., in the book by Brandt [1].

5 Simulation of x-ray photons from synchrotron radiation

When a charged particle such as an electron traverses a magnetic field, it emits electromagnetic radiation (photons) tangential to its direction of motion. The energy spectrum of these photons has a certain derivable form which depends on the electron's energy and the strength of the magnetic field. This is difficult to calculate, but fortunately we don't have to; spectra can be obtained from the Center for X-ray Optics at the Lawrence Berkeley National Laboratory at `http://www.cxro.lbl.gov/`. Follow the links to 'x-ray database', and then 'synchrotron bend magnet radiation'. You can choose various parameters for the electron beam to produce energy spectra such as the one shown in Fig. 1. The data can be downloaded into a file for use in your project.

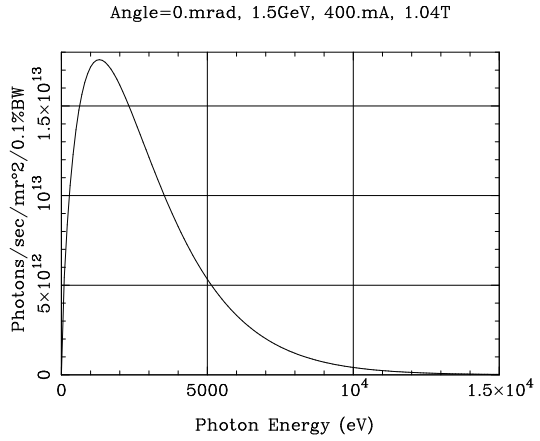

Figure 1: A photon energy spectrum from a synchrotron radiation source.

Once you have obtained a spectrum like Fig. 1, the next step is to generate photon energies such that the probability to obtain a given energy is proportional to the height of the curve. The easiest way to do this is with the *acceptance–rejection* technique.

- First generate two independent random numbers, r_1 and r_2, both uniformly distributed between 0 and 1.
- Use r_1 and r_2 to produce two more numbers: $E = r_1 E_{\max}$ and $f = r_2 f_{\max}$, where $E_{\max}$ is the maximum photon energy and $f_{\max}$ is at least as high as the maximum of the energy spectrum. For Fig. 1, for example, use $E_{\max} = 1.5 \times 10^4$ and $f_{\max} = 1.8 \times 10^{13}$. The two numbers E and f will be represented by a point somewhere on the energy spectrum plot.
- If the point (E, f) is below the curve, accept E as the generated photon energy, otherwise, reject the value and repeat the procedure until a value is accepted.

The probability to accept the point is proportional to the height of the curve, so the accepted energies will have the desired distribution.

As a check of this part of the calculation, you should show a histogram of your generated photon energies on a plot with the original energy spectrum superimposed. To make a meaningful comparison, the area under the two plots must be scaled to the same value. For this it is most convenient to normalize both to unit area. You will need to consider carefully how to do this for both distributions.

6 Interaction of x-rays with matter

We will imagine that the x-rays hit a layer of material of a given thickness x. We would like to know how many of the photons make it out the other side and what their energies are. The probability for a photon to be absorbed by the photoelectric effect before penetrating a distance x can be expressed as

$$P(\gamma \text{ of energy } E \text{ absorbed before } x) = 1 - e^{-x/\lambda(E)} \tag{3}$$

where $\lambda(E)$ is the attenuation length, which is in general a complicated function of the photon's energy. Attenuation lengths measured as a function of energy can be downloaded from the Center for X-ray Optics for a variety of materials; an example for beryllium is shown in Fig. 2. (Follow the links 'x-ray interactions with matter', 'attenuation length'.)

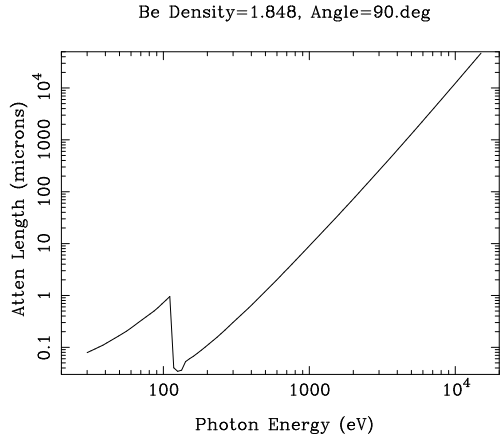

Figure 2: Attenuation length for x-rays in beryllium as a function of energy.

The strategy is thus to start by generating a photon with a certain energy. For this energy, determine the attenuation length $\lambda(E)$ and from (3) the probability P for the photon to be absorbed before making it through a given thickness of material, e.g. $x = 1$ mm beryllium. Then generate another random number r between 0 and 1. If $r < P$, we say that the photon is absorbed, if not, it makes it through. (In fact, a real photon may make it through but its energy can be degraded by Compton scattering. For now we will ignore this effect.)

You will then have a subset of the photons which make it through the layer of material, and you should make a histogram of their energies. Generate enough events so that the energy spectrum of the photons getting through is reasonably well determined.

As a minimum you should determine the mean photon energy as a function of the thickness of a given absorber and show on an appropriate plot. As the mean energy will be determined with a finite number of simulated photons, it will have a statistical error which you should calculate for each point and display. Should time permit, you can try varying the parameters of the problem such as the thickness of the composition, the nature of the photon source, etc.

7 Software implementation

This section gives some advice on how to implement the calculation described above in a C++ program. At different stages it is necessary to read in columns of numbers from a file that you get from the website `www.cxro.lbl.gov`. The numbers can be thought of as x and y values, i.e., you read in pairs of numbers (x_i, y_i) for $i = 1, \ldots, n$. Using these one needs an approximation for the function $y(x)$ for arbitrary x, that is, even at values of x that do not coincide with any of the x_i that were initially read in.

This task arises, for example, when obtaining the spectrum $f(E)$ of the photon energies by using the numbers (f_i, E_i) read in from a file. And one has essentially the same problem when obtaining the attenuation length $\lambda(E)$. So to carry out this task, it is useful to create a C++ class that can read in a file containing two columns of numbers, say, x and y, and the class should contain a method that returns the function $y(x)$, which can be based on an interpolation between the (x_i, y_i) points from the file. The basic idea is illustrated in Fig. 3.

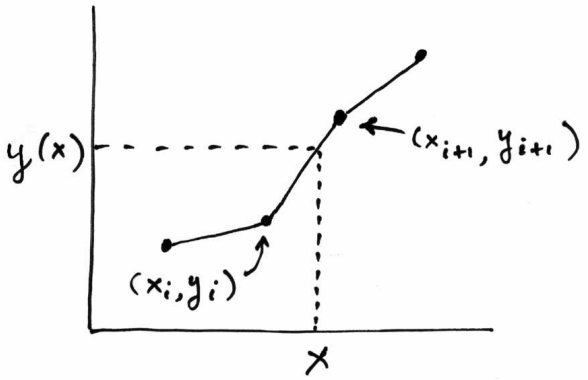

Figure 3: Illustration of the interpolation between points to approximate the function $y(x)$.

The suggested method for dealing with this problem is to create a class called, say, `FunctionFromTable`, since its purpose is to take as input a table of numbers, i.e., the points (x_i, y_i), $i = 1, \ldots, n$ and to provide a function $y(x)$, that could be evaluated with any value of x. An example of such a class is provided in

`http://www.pp.rhul.ac.uk/~cowan/ph3010/xray/FunctionFromTable/`

A test program `testFunctionFromTable` together with a sample data file `energy_spectrum.txt` illustrates how the class is used. To begin the project you can simply use this class. As extension objectives, you could introduce at least two improvements.

First, to find the to points x_i and x_{i+1} which bracket the input value x, the class currently loops over all values of x_i starting from $i = 0$ and determines whether the input value x lies in the range $x_i \leq x < x_{i+1}$. As x can be anywhere in the range of allowed values, the average number of computing steps required to locate the bracketing pair (x_i, x_{i+1}) is proportional to the number of points in the table.

A more elegant and much faster method would be to use a binary search, where the list of x_i values is first cut in half and one determines in which of the two halves the input value x is located. The half containing x is then redefined as the range of x where one searches and the procedure is iterated until the range consists of just two adjacent points. One can show that

the number of computing steps required to bracket a given x value grows only logarithmically with the number of points in the table.

A second improvement would be to use a `spline` function to interpolate between points rather than a straight line. A simple version of this called a cubic spline can be understood again from Fig. 3 by considering the set of four x values x_{i-1}, x_i, x_{i+1} and x_{i+2}, where the input x value lies in the range $x_i \leq x < x_{i+1}$. The set of four (x, y) points (called *knots*) uniquely determine a cubic equation, and it is this equation that is used to provide the interpolated value.

The cubic spline of course requires that there be two values in the table on each side of the input x value, so special treatment must be provided for the edges of the range. In the regions between the first and last pairs of points, for example, one can simply do a linear interpolation. For the purposes of the present project, such a spline fit does not bring a significant improvement but it is an interesting and useful numerical method and if you have time you can either implement this yourself or use an external routine such as `TSpline3` from the root library.

8 Including Compton scattering

Monte Carlo simulations like the one in this project are an important element of many experiments. For particle physics detectors such as those at the Large Hadron Collider, the simulations can model complex interactions of many types of particles and have millions of lines of code.

As a very small step in this direction, a possible extension to the x-ray simulation is to allow for processes other than photo-electric absorption as the photon passes through matter. For higher photon energies, there is a significant probability that the photon will undergo Compton scattering with an atomic electron. In this way, the photon is scattered (in effect re-emitted) with a reduced energy and changed direction. The simulation must continue to track the progress of the new photon until it either exits the material or is finally absorbed.

This is a very ambitious extension and will not be described in detail in this project script. Some brief notes are included on the project web page; for more information please discuss with your supervisor.

9 Project report

Your project report should contain the following elements:

- A brief overview of the physical processes that are being modeled.
- A description of the important mathematical techniques that you implement in software, including generation of random numbers and the Monte Carlo method. If, for example, you include a spline fit or binary search, then this should be described.
- A brief description of how the mathematical concepts were implemented in your code. For this you may want to include small portions of code in the body of the report.
- A description of the error analysis (e.g., the statistical errors for the average photon energy).

- Plots of all relevant distributions containing appropriate labels and captions.
- All of your code should be included as an appendix.

Try to organise your report in a meaningful way. The introduction should state what the project covers and give a road map to the rest of the report. Consider the order in which information is presented.

Think carefully about what information the reader already knows and what needs to be explained and defined. For this you should assume your reader has a general background in physics but no specialist knowledge about the Monte Carlo method or x-ray interactions.

Consider carefully the precise meaning of your words. If you write "calculate" than this should be exactly what you mean (not estimate, measure, compute).

In your conclusions, summarise what one can conclude from the numerical investigations you have carried out and discuss briefly how they could be extended.

10 References

This script and links to other resources can be found on the project web page: `www.hep.ph.rhul.ac.uk/~cowan/ph3010/xray/`

X-ray data can be obtained from the web site of the Center for X-ray Optics at the Lawrence Berkeley National Laboratory: `www.cxro.lbl.gov/`.

Further references are listed below:

References

[1] S. Brandt, *Statistical and Computational Methods in Data Analysis*, Springer, New York (1997).

[2] G. Cowan, *Statistical Data Analysis*, Clarendon Press, Oxford (1998).

[3] L. Lyons, *Statistics for Nuclear and Particle Physicists*, Cambridge University Press, Cambridge (1986).

[4] K. Nakamura et al. (Particle Data Group), J. Phys. G 37, 075021 (2010); `pdg.lbl.gov`.

[5] Weg page of the root data analysis framework `root.cern.ch`.

[6] P.L. L'Ecuyer, *Efficient and portable combined random number generators*, Commun. ACM **31** (1988) 742.