PH3010 - X-ray Simulation with the Monte Carlo Method

Aleksas Girenas

December 10, 2020

1 Abstract

We will demonstrate the use of the Monte Carlo method to investigate the production of X-rays and their interaction with matter. By generating uniformly distributed random numbers with CERN-root [1] and applying an acceptance-rejection technique to obtain photon energies we simulated the production of X-Ray radiation from a synchrotron source [2]. We then found the mean energy of the photons that penetrated a Beryllium absorber of a set thickness by testing against the attenuation lengths of Be [2].

Repeating this for different thicknesses and plotting in **Python** we found a logarithmic correlation of photon energy with the thickness of Beryllium. The project was completed in C++ using a Scientific Linux (SL) [3] remote server that includes the CERN-root frameworks required. The final absorption plot was produced locally in Python using the mean energies obtained at varying thicknesses.

2 Introduction

First we must use X-ray radiation data produced by using known synchrotron radiation from [2]. The photons are emitted when a charged particle such as an electron travels at relativistic speeds perpendicular to a magnetic field [4]. Energy spectra can be obtained for the photons and we use this to simulate our experiment. As these are randomly distributed emissions we must interpolate between the values from the synchrotron radiation data in order to get a random distribution of photon energies. We will use a pre-written class for loading and interpolating between y values at a set x value **FunctionFromTable**; this was used repeatedly for both loading and using the initial data from synchrotron radiation data and to find the absorption of photons by Be using Be attenuation length data. Using this data we can randomly produce our photons for the simulation and so have a random distribution of photons at random energies.

Next we will see whether the X-ray photon passes through a sample of matter using attenuation length data from [2]. Photons interact with matter through the photoelectric affect where the inner electron shell of a particle causes the absorption of the photon leading to the release of the electron from that shell. The electron then moves from a higher energy level to fill the vacancy in the inner shell causing the release of a photon that has energy equal to the difference in energy levels. Measuring the mean photon energies vs the thickness from the simulation we can find how photons interact with the matter.

A mostly functional approach was used to write the code as it was easier to write in nano on the remote server and re-use/scalability of the program was not a requirement for this project. The main.cc file code can be seen in the appendix and includes the core functionality.

As each time the program is run a set of new random numbers are generated in order to simulate photons the following graphs may not match perfectly with each other; however, for the purposes of this report the graphs show the general trends and so are sufficient in showing X-Ray simulation and interactions with matter.

3 X-ray photon simulation

For this simulation we used synchrotron data with: an electron energy of 1.9 GeV; a the storage ring currant of 400 mA; and a magnetic field strength of 1.27 T. This spectrum is used to determine a set of random photon energies that we can simulate photon interaction with matter.

The simulation of photon energies and their interaction with matter is most easily done using the Monte Carlo method, a technique that involves using random numbers and probability to produce random values for given data.

By first generating two independent random numbers, $r1$ and $r2$, both uniformly distributed between 0 and 1 we can produce two more numbers: $E = r1Emax$ and $f = r2fmax$, where $Emax$ is the maximum photon energy and $fmax$ is at least as high as the maximum of the energy spectrum. If the point (E, f) is below the curve, E is accepted as the generated photon energy, otherwise we can reject the value and repeat the procedure until a value is accepted. This was done 800,000 times to have a large distribution of photon energies.

Uniformly distributed random numbers can be generated using Trandom3 in the CERN-root framework [1] and by finding an E value and f we can see if the coordinates lied below the curve of the downloaded Bend Magnetic Spectrum data using **FunctionFromTable** that interpolates between the data points. In **FunctionFromTable** energy points from the downloaded data are compared to determine which x values the generated energy is between. A straight line can be estimated between the points that the generated energy was between and the frequency can be obtained. The frequency obtained from **FunctionFromTable** is compared to the one generated from $r2fmax$. If the generated frequency is lower than the one obtained from interpolation, then it falls below the curve and the energy is accepted and we add it to our own histogram. This data was stored in a $vector < double >$ to maintain accuracy and to allow for the vector to grow as data is added and so save memory.

To see if our random photon energies match the Bend Magnetic

Spectrum data that was downloaded we can create a histogram and overlap the data for comparison. In CERN-root we can create a Tfile that includes a histogram class:

```
TH1D h_Uniform("h_Uniform", "Angle=0.mrad,␣1.9GeV,␣
    ↪ 400.mA,␣1.27T", numBins, 0, 10000);
```

Creating two histograms, one of the original data and one of the accepted energies, we can use a CERN-root macro to superimpose the two graphs.

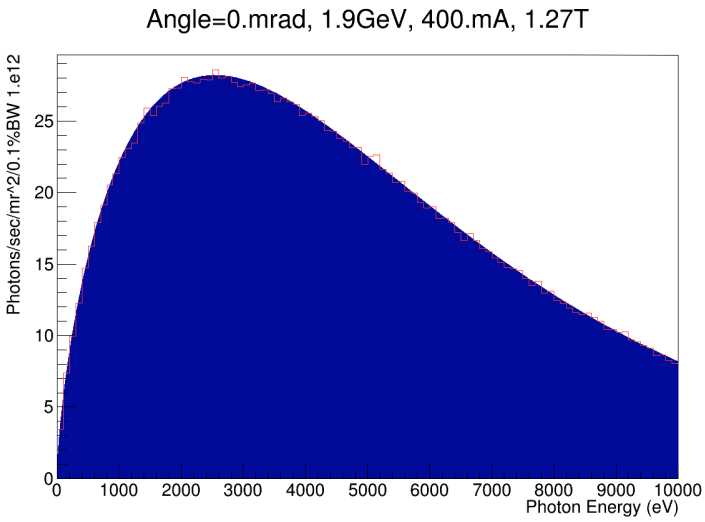

Figure 1: Overlay of accepted photons (red) over Bend Magnetic Spectrum data (blue)

The probability of accepting an energy is proportional to the height of the original X-ray photon energy spectrum at that energy so the accepted photon energies would also have the same shape as the downloaded energy spectrum.

Figure 1 displays the superimposed histograms. Both histograms were scaled to make comparison meaningful, and clearly both follow the same shape showing that the acceptance-rejection method worked.

4 X-ray interaction with matter

Now we can use our random photons to observe their interaction with matter. For this simulation we use a Beryllium sample which has a density of 1.848 gm/cm3.

The probability of a photon being absorbed by the photoelectric effect is given by:

$$P(\gamma) = 1 - e^{-x/\lambda(E)} \tag{1}$$

where x is the thickness of the material and $\lambda(E)$ is the attenuation length. $\lambda(E)$ is the depth of the material where the intensity of the radiation falls to $1/e$ of its values measured as a normal to the surface. We can find the Be Attenuation Length data at a fixed angle of 90 degrees from [2].

By using equation (1) we can obtain the probability of each accepted photon that is re-emitted. For each accepted photon energy, we can again use $FunctionFromTable$ to interpolate and to obtain attenuation lengths for set photon energies. The probabilities calculated for each photon energy were again compared to a uniformly distributed random number between 0 and 1 using TRandom3. If the probability was greater than the random number, the photon is considered to be absorbed, otherwise it has been re-emitted.

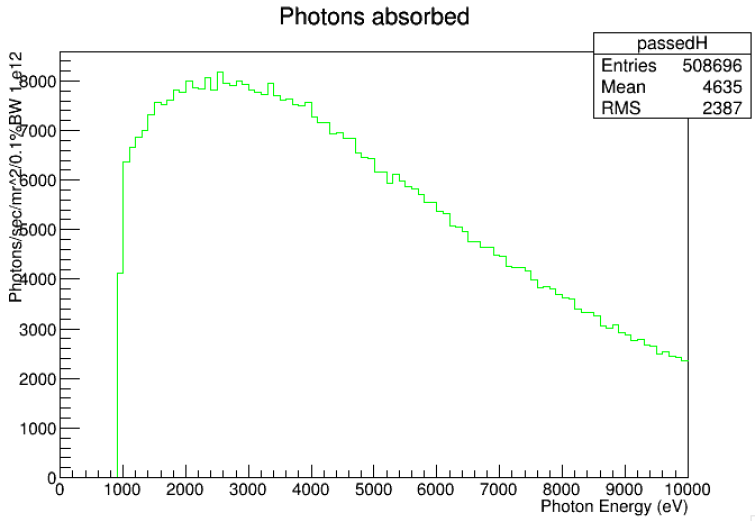

Figure 2: Example photons passed by 1micron thickness Be

Figure 2 shows the histogram of the photons that were not absorbed for a thickness of 1 microns and at this thickness the mean photon energy was found to be 4635 eV. This is easily calculated by summing the not absorbed photon energies and dividing it by the total number of photons obtained from the synchrotron simulation.

To obtain the mean photon energy at multiple thicknesses we can create a $for()$ loop that runs the absorption calculation for increasing thicknesses. Figure 3 shows the relationship of mean photon energy against the thickness of Beryllium in microns.

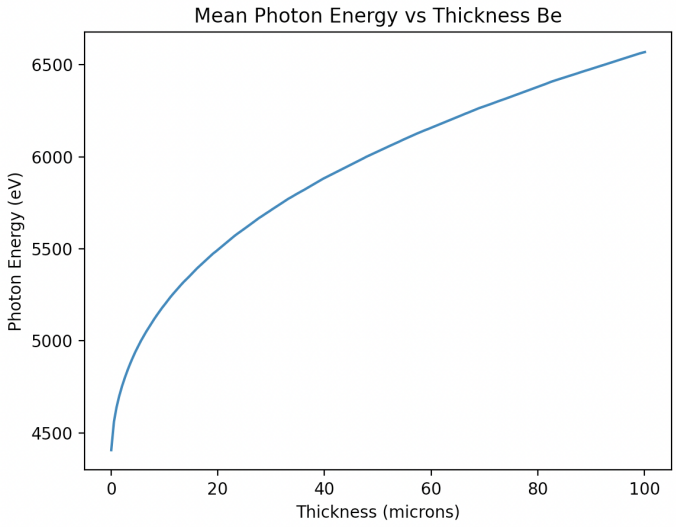

Figure 3: Plot of mean photon energies absorbed vs thickness of Be

We can see that there is a logarithmic relation; as the thickness increases the photon energy emitted grows much more quickly.

5 Conclusion

The Monte Carlo method was successfully used in order to simulate the production of photons from synchrotron radiation and we simulated their interaction with matter to evaluate the energy of re-emitted photons. By generating uniformly distributed random numbers with CERN-root [1] and applying an acceptance-rejection technique to obtain photon energies we simulated the production of X-Ray radiation from a synchrotron source [2].

The acceptance-rejection was accurate when used to determine the accepted energies as comparing the histogram of accepted energies had the same shape as the original spectrum.

The interaction of X-rays with matter of a chosen thickness was also simulated through random numbers. We found the mean energy of the photons that penetrated a Beryllium absorber of a set thickness by testing against the attenuation lengths of Be [2]. Repeating this for different thicknesses and plotting in **Python** we found a logarithmic correlation of photon energy with the thickness of Beryllium.

The only photon interaction with matter that was discussed was the photoelectric effect. Further effects that were not accounted for include Compton scattering which would also affect photon interaction with matter.

These techniques are important in many fields including the understanding of medical imaging [5].

References

[1] CERN - root Data analysis framework https://root.cern.ch/

[2] X-Ray Interactions With Matter https://henke.lbl.gov/optical_constants/

[3] Scientific Linux (SL) https://scientificlinux.org/

[4] What is a Synchrotron https://www.esrf.eu/about/synchrotron-science/synchrotron

[5] Radiation penetration http://www.sprawls.org/ppmi2/RADPEN/

Appendices

C++ script used. (CERN-root macros and FunctionFromTable class not included). The complete code can be seen on linappserv1of the RHUL Physics Department in the directory agirenas/Xrays/Project/

```
#include <iostream>
#include <cmath>
#include <TH1D.h>
#include <TFile.h>
#include <TRandom3.h>
#include "FunctionFromTable.h"

using namespace std;

const int numBins = 20000;
int numARIterations = 800000;
void Histogram(FunctionFromTable edist);
vector<double> AcceptanceRejection(FunctionFromTable
    ↪ edist);
void HistogramOverlay(vector<double> acceptedE);
vector<double> FinalPhotons(vector<double> acceptedE,
    ↪  float thickness);
void PassedPhHist(vector<double> passedPh, const char
    ↪ * fileName);


int main(){

// Using FunctionFromTable
    double scale = 1.e12;
    FunctionFromTable edist("energy_spectrum.txt",
        ↪ scale);

    vector<double> acceptedE = AcceptanceRejection(
        ↪ edist);


    Histogram(edist);
    HistogramOverlay(acceptedE);

    for(float i = 0.05; i < 100.0; i+=0.5){
      vector<double> finalPh = FinalPhotons(acceptedE,
          ↪  i);
    }

    vector<double> finalPh = FinalPhotons(acceptedE,
        ↪ 1.0);

    PassedPhHist(finalPh, "PhotonsPassed.root");
  return 0;
}

void Histogram(FunctionFromTable edist){

// Open output file

    double scale = 1.e12;
    FunctionFromTable edist("energy_spectrum.txt",
        ↪ scale);

    vector<double> acceptedE = AcceptanceRejection(
        ↪ edist);


    Histogram(edist);
    HistogramOverlay(acceptedE);

    for(float i = 0.005; i < 50.0; i+=0.05){
      vector<double> finalPh = FinalPhotons(acceptedE,
          ↪  i);
    }

    vector<double> finalPh = FinalPhotons(acceptedE,
        ↪ 1.0);

    PassedPhHist(finalPh, "PhotonsPassed.root");
  return 0;
}
```

```
void Histogram(FunctionFromTable edist){

// Open output file

  TFile file("XrayPlot.root", "recreate");

// Book histograms, i.e., create TH1D objects // 1st
    ↪ argument is a string; by convention same as
    ↪ object name // 2nd argument is histogram tit$
// 3rd argument is number of bins (type int) // 4th
    ↪ and 5th arguments: upper and lower limits of
    ↪ histogram (type double)

  TH1D h_Uniform("h_Uniform", "Angle=0.mrad,␣1.9GeV,␣
      ↪ 400.mA,␣1.27T", numBins, 0, 10000);

  for (int i=0; i<numBins; ++i){
       double r = edist.val(i);
       h_Uniform.Fill(i, r);
  }

// Store all histograms in the output file and close
    ↪ up

  file.Write();
  file.Close();
}

void HistogramOverlay(vector<double> acceptedE){
// Open output file

  TFile file("XrayPlot.root", "recreate");

// Book histograms, i.e., create TH1D objects // 1st
    ↪ argument is a string; by convention same as
    ↪ object name // 2nd argument is histogram tit$
// 3rd argument is number of bins (type int) // 4th
    ↪ and 5th arguments: upper and lower limits of
    ↪ histogram (type double)

  TH1D h_Uniform("h_Uniform", "Angle=0.mrad,␣1.9GeV,␣
      ↪ 400.mA,␣1.27T", numBins, 0, 10000);

  for (int i=0; i<numBins; ++i){
       double r = edist.val(i);
       h_Uniform.Fill(i, r);
  }

// Store all histograms in the output file and close
    ↪ up

  file.Write();
  file.Close();
}

void HistogramOverlay(vector<double> acceptedE){

  TFile file("XrayPlotOverlay.root", "recreate");

  TH1D acceptedH("acceptedH","Xrays␣after␣AR␣
      ↪ technique" , 100, 0, 10000);

  for (int i=0; i<acceptedE.size(); ++i){
       double r = acceptedE[i];
       acceptedH.Fill(r);
  }

  // Store all histograms in the output file and
      ↪ close up

  file.Write();
  file.Close();
}

void PassedPhHist(vector<double> passedPh, const char
    ↪ * fileName){
 TFile file(fileName, "recreate");

  TH1D passedH("passedH","Photons␣absorbed" , 100, 0,
      ↪  10000);

  for (int i=0; i<passedPh.size(); ++i){


  TFile file("XrayPlotOverlay.root", "recreate");

  TH1D acceptedH("acceptedH","Xrays␣after␣AR␣
      ↪ technique" , 100, 0, 10000);

  for (int i=0; i<acceptedE.size(); ++i){
       double r = acceptedE[i];
       acceptedH.Fill(r);
  }

  // Store all histograms in the output file and
      ↪ close up

  file.Write();
  file.Close();
}

void PassedPhHist(vector<double> passedPh, const char
    ↪ * fileName){
 TFile file(fileName, "recreate");

  TH1D passedH("passedH","Photons␣absorbed" , 100, 0,
      ↪  10000);

  for (int i=0; i<passedPh.size(); ++i){
       double r = passedPh[i];
       passedH.Fill(r);
  }

  // Store all histograms in the output file and
      ↪ close up
  file.Write();
  file.Close();

}

vector<double> AcceptanceRejection(FunctionFromTable
    ↪ edist){
  vector<double> acceptedE;
  TRandom3 rand(99998);
  TRandom3 rand1(12345);
```

```
  for(int i = 0; i < numARIterations; ++i){
        double E = rand.Rndm()*10000;
        double f = rand1.Rndm()*edist.maxVal();
        if(f < edist.val(E)){
               acceptedE.push_back(E);
        }
  }

  for (int i=0; i<passedPh.size(); ++i){
        double r = passedPh[i];
        passedH.Fill(r);
  }

  // Store all histograms in the output file and
     ↪ close up
  file.Write();
  file.Close();

}

vector<double> AcceptanceRejection(FunctionFromTable
    ↪ edist){
  vector<double> acceptedE;
  TRandom3 rand(99998);
  TRandom3 rand1(12345);
  for(int i = 0; i < numARIterations; ++i){
        double E = rand.Rndm()*10000;
        double f = rand1.Rndm()*edist.maxVal();
        if(f < edist.val(E)){
               acceptedE.push_back(E);
        }
  }
  return acceptedE;

}

vector<double> FinalPhotons(vector<double> acceptedE,
    ↪  float thickness){
  // Using FunctionFromTable
    double scale = 1.0;
//Check function from table cout to see if scale
    ↪ correct!!!
    FunctionFromTable edist("attenLength_Be.txt",
        ↪ scale);
    vector<double> photons;
    for(int i = 0; i<acceptedE.size(); i++){
        double P = 1 - exp(-thickness/edist.val(
            ↪ acceptedE[i]));
        TRandom3 rand(123456);
        if(rand.Rndm() > P){
             //photon makes it through
             photons.push_back(acceptedE[i]);
        }
    }
    double total = 0;

    for(int i = 0; i<photons.size(); i++){
       total += photons[i];
    }
    cout << total/photons.size() << endl;
    return photons;
}
```