Computing the Solid Angle of Two Detectors Using Monte Carlo Methods

Sean O’Brien

August 15, 2006

Abstract

The solid angle between two detectors arranged in coincidence was computed using a Monte Carlo simulation program written in C++ [1]. This was done in order to allow an accurate measurement of cosmic ray flux through the detector setup.

1 Solid Angle

For a system involving a series of overlapping detectors, it is critical that their solid angle, $\Omega$, be known as it is needed in making certain measurements. The solid angle is a way of describing how an object appears to an observer at a given point. Given an object in three dimensional space one takes the object and projects the image of it onto the sphere incapsulating the object and the solid angle is then proportional to the surface area, $S$, of that projection, then divided by the radius, $R$, of the sphere squared, $(\Omega = \frac{kS}{R^2})$. The proportionality constant, $k$, is related to the surface area of a sphere in much the same way as the circumference is related to the radian. If we choose $k = 1$, then the solid angle of a sphere of radius equal to one, will have a solid angle of $4\pi$ steradians (sr) when measured from its center. Therefore the solid angle can be no greater than $4\pi$ sr. [2].

2 Monte Carlo Method

In science and mathematics, many problems are to complex to solve using conventional methods, such as by integral calculus or other numerical methods. The Monte Carlo method is generally useful in calculating complex integrals and problems that involve several variables. The Monte Carlo method involves the use of a large number of calculations involving many random numbers to populate the function in question to arrive at a statistical solution [1].
3 Determining the Solid Angle of Two Detectors

A single detector has a solid angle of $4\pi$, when the source is coming from all directions and a solid angle of $2\pi$ when the source comes from a single hemisphere as in cosmic ray detection as essentially none are detected coming up through the earth. However, when two detectors are used together, and separated by a distance, the solid angle is determined by the projection of what passes through both detectors. This calculation is extremely difficult to solve analytically but rather requires the statistical Monte Carlo method. In which many random numbers are used to test the problem. In the case of determining the solid angle of two detectors in coincidence, random values were generated for the three spatial dimensions $x$, $y$, and $z$ and the angles $\theta$, the polar, and $\phi$, the azimuthal. Once generated, these values were used to compute vectors that were tested to see if they passed through both of the detectors.

![Diagram of detectors and simulated test vectors.](image)

After a number of random sets of values are used and determined if they passed through both detectors or not, one can then determine the solid angle. Given the above formula for the solid angle, $\Omega = \frac{kS}{R^2}$, and having $k = 2\pi$, as we are only considering our detectors solid angle from a single hemisphere, we can
arrive at the solid angle. This calculation of the solid angle is done by considering the number of random vectors that passed through both the detectors and the total number of vectors generated. Using these values one can then determine the percentage of the solid angle that the detectors have by $\frac{\text{passed}}{\text{total}}$. This fraction is then multiplied by $k$ in this case $2\pi$ to determine the solid angle.

$$\Omega = 2\pi \frac{\text{passed}}{\text{total}} \quad (1)$$

4 Using C++ to Compute the Solid Angle

The solid angle program, written in C++, first requests an input for the number of random sets to use as test values. Since the compiler comes with a pseudo random number generator, because it uses a strange function to generate its numbers, the program must compensate for this, as random numbers are required for a Monte Carlo simulation, and does so in two ways; first the program asks for an input on how to seed the random number generator. This seed changes the starting point of the generator. Next, using the random seed, the program fills a 100x100 matrix with random numbers, this is done to ensure even more random results. The program calls this matrix everytime it needs a new random number, which it selects by producing two random numbers, one for the row and the other for the column of the matrix, and takes that number and assigns it to variable, such as x, and then replaces it with another random number. This segment of code controls the above mentioned processes.

```c++
cout<<"Input the seed to seed the array: \n";
cin>>seed; //value to seed the random number generator
getchar();
srand((unsigned)seed); //the random number generator is seeded
for(first=0;first<=99;first++) { //these two loops populate the array with values
  second=0;
  for(second=0;second<=99;second++) {
    matrix[first][second]=rand();
  }
}
for(loops==0;loops<random_sets;loops++) { //this loop assigns random numbers to variables
  pick_1=rand()%100; //picks a random point in the matrix
  pick_2=rand()%100;
  rand_value_x=matrix[pick_1][pick_2]; // assigns value to variable
  matrix[pick_1][pick_2]=rand(); //reassigns value to used point in matrix
  pick_1=rand()%100;
  pick_2=rand()%100;
  rand_value_z=matrix[pick_1][pick_2];
```
These random numbers then must be adjusted to the required size for each variable. This is done by simply knowing the maximum random value generated by the computer and then dividing it by the appropriate value to give one the range of values needed. This maximum random number varies by compiler, in this case, using a g++ compiler on linux, the number was 2,147,483,647. This segment of code controls the above mentioned processes.

To improve the efficiency of the program all random values were chosen to originate on bottom surface of the bottom plate. All x values are between 0 and .276, z values between 0 and .128, y (the vertical direction) was always zero (x, y and z correspond to the physical dimensions of the detector), θ ranged between 0 and π and phi between 0 and 2π. It is important for the effectiveness of the Monte Carlo simulation that the ranged random values to be used are distributed evenly throughout the detectors dimensions. If these values are not evenly distributed this can lead to a skewed result. To ensure that this was not
the case all random values generated by the program were made into histograms. Below is an example of one such histogram but all were distributed in a similar manner corresponding to their range.

![Distribution of x](image)

**Figure 2:** Distribution of random values in x.

From here the program uses vectors to calculate a series of points to test against the top plates conditions. The separation of the top and bottom detectors can be altered by user input. If all tests are passed for a single set, then one is added to the number of total sets passed and the loop is exited and another series of random numbers is tested. This segment of code controls the above mentioned processes.

```plaintext
hyp=(seperation -.1);  
for(hyp== (seperation - .1);hyp<(seperation + .1);hyp+=.00001){ //loop that varies hyp  
x_test=x_random+(hyp*cosine_theta); // vectors that are generated until all pass  
y_test=y_random+(hyp*sine_theta);
```
z_test = z_random + (hyp * sine_phi);

if (x_test >= 0 && x_test <= .276) // tests value of x
x_pass++;

if (y_test >= seperation && y_test <= (seperation + .01)) // test value of y
y_pass++;

if (z_test >= 0 && z_test <= .128) // tests value of z
z_pass++;

if (x_pass == 1 && z_pass == 1) // compares tests of x and z
x_and_z_pass++;

if (y_pass == 1 && x_and_z_pass == 1) // compares tests of y and x and z
total_passed_in++;

if (total_passed_in == 1) // counts and resets passed values
{ total_passed++;
total_passed_in = 0;
// writes passed values to file
ofstream out_file_passed("solid_angle_passed_x_y_z_ct_sp.txt", ios::app);
    out_file_passed << x_test << ' ' << y_test << ' ' << z_test << ' ' << cosine_theta << ' ' << sine_phi << '
';

x_pass = 0; // resets counters if all pass
y_pass = 0;
z_pass = 0;
x_and_z_pass = 0;
hyp = (seperation + 1);
}

x_pass = 0; // resets if hyp has not reached max
y_pass = 0;
z_pass = 0;
x_and_z_pass = 0;
}
hyp = (seperation - .01); // resets if hyp reaches max with no passes and restarts loop with new num
x_pass = 0;
z_pass = 0;
x_and_z_pass = 0;
y_pass = 0;
This process is repeated until the initial number of random sets to test has been satisfied. The program then outputs the solid angle given by \( \frac{\text{numberpassed}}{\text{numbertested}} \cdot 2\pi \).

Then using a goto statement the program asks the user whether they would like to rerun the program. In addition throughout the program there are write and read file commands that were later used to analyze the data produced.

5 Analysis of Program Data

As mentioned above, throughout the program nearly all values produced by the program are written to a file for later analysis. In these files the values written to file are all the random numbers produced by the program, to insure that the numbers are generated uniformly throughout and the values for all the random numbers that eventually pass the programs conditions, so the acceptance of the detectors can be understood. Here is one such example of an acceptance.

![Acceptance of Detectors](plot_ct_sp.txt)

Figure 3: The acceptance of the detector system in \( \cos(\theta) \) and \( \sin(\phi) \).
In addition to knowing the acceptance and solid angle of the detectors for a certain separation, it is useful to be able to calculate the solid angle at any separation of the two detector plates. This allows one to easily change the separation of the detectors to make new measurements as well as understand the sensitivity of the solid angle’s value as related to the plates separation.

![Graph showing solid angle of detectors with increasing separation](solid-angle-delta-r.png)

Figure 4: The Solid Angle with increasing plate separation.

6 Conclusion

With the combined use of the Monte Carlo method and computer programing in C++, the solid angle for a system of detectors was successfully calculated and utilized in the measurement of the angular flux of cosmic rays. By using a well known value to measure, cosmic ray flux, we were able to experimentally verify the calculated solid angle of the detector setup (see ”Detector Data and Analysis of Cosmic Radiation” by Lori Jackson for details).
References


7 Solid Angle Source Code

#include<iostream>
#include<cstdlib>
#include<math.h>
#include<fstream>
long double x_random;
long double y_random;
long double z_random;
long double theta;
long double phi;
long double sine_theta;
long double cosine_theta;
long double sine_phi;
long double loops;
long double random_sets;
long double hyp;
long double x_test;
long double y_test;
long double z_test;
long double x_pass;
long double y_pass;
long double z_pass;
long double x_and_z_pass;
float total_passed;
float total_passed_in;
long double seed;
int first;
int second;
long double rand_value_x;
long double rand_value_z;
long double rand_value_theta;
long double rand_value_phi;
int pick_1;
int pick_2;
lonw double matrix[100][100];
int again;
int x_range;
long double data_x_array[100000];
long double data_z_array[100000];
long double data_ct_array[100000];
long double data_st_array[100000];
long double data_sp_array[100000];
float seperation;
float sep_min;
float sep_max;
float sep_inc;
using namespace std;
int main()
{
    loops=0;
x_and_z_pass=0;
x_pass=0;
z_pass=0;
x_range=0;
total_passed_in=0;
total_passed=0;
cout<<"Enter number of random sets to test:\n";
cin>>random_sets;
getchar();
cout<<"Enter the initial seperation between the plates:\n";
cin>>sep_min;
getchar();
cout<<"Enter the maximum seperation between the plates:\n";
cin>>sep_max;
getchar();
cout<<"Input the incrementation between seperation min and max:\n";
cin>>sep_inc;
getchar();
hyp=(seperation - 1);
cout<<"Input the seed to seed the array: \n";
cin>>seed;
getchar();
srand((unsigned)seed);
for(first=0;first<=99;first++){
    second=0;
    for(second=0;second<=99;second++){
        matrix[first][second]=rand();
    }
    seperation=sep_min;
    for(seperation==sep_min; seperation <= sep_max ; seperation= seperation + sep_inc){
        total_passed=0;
        total_passed_in=0;
        x_pass=0;
    }
y_pass=0;
z_pass=0;
x_and_z_pass=0;
for(loops==0;loops<random_sets;loops++){
pick_1=rand()%100;
pick_2=rand()%100;
rand_value_x=matrix[pick_1][pick_2];
matrix[pick_1][pick_2]=rand();
pick_1=rand()%100;
pick_2=rand()%100;
rand_value_z=matrix[pick_1][pick_2];
matrix[pick_1][pick_2]=rand();
pick_1=rand()%100;
pick_2=rand()%100;
rand_value_theta=matrix[pick_1][pick_2];
matrix[pick_1][pick_2]=rand();
pick_1=rand()%100;
pick_2=rand()%100;
rand_value_phi=matrix[pick_1][pick_2];
matrix[pick_1][pick_2]=rand();
x_random=rand_value_x/7780737851.45;
y_random=0;
z_random=rand_value_z/16777215992.2;
theta=rand_value_theta/1073741823.5;
phi=rand_value_phi/1073741823.5;
sine_phi=phi-1;
cosine_theta=theta-1;
sine_theta=sqrt(1-(cosine_theta*cosine_theta));
ofstream out_file_rand("solid_angle_data_rand_x_z_costheta_sinephi.txt",ios::app);
out_file_rand<<x_random(" ");
out_file_rand<<z_random(" ");
out_file_rand<<cosine_theta(" ");
out_file_rand<<sine_phi(" ");
out_file_rand<<endl;
}
x_pass=0;
y_pass=0;
z_pass=0;
x_and_z_pass=0;
hyp=(seperation -.1);
for(hyp== (seperation - .1);hyp<(seperation + .1);hyp+=.00001){
x_test=x_random+(hyp*cosine_theta);
y_test=y_random+(hyp*sine_theta);
z_test=z_random+(hyp*sine_phi);
}
if(x_test>=0 && x_test<=.276)
x_pass++;

if(y_test>=seperation && y_test<=(seperation + .01))
y_pass++;

if(z_test>=0 && z_test<=.128)
z_pass++;

if(x_pass==1 && z_pass==1)
x_and_z_pass++;

if(y_pass==1 && x_and_z_pass==1)
total_passed_in++;

if(total_passed_in==1){
total_passed++;
total_passed_in=0;
}

ofstream out_file_passed("solid_angle_passed_x_y_z_ct_sp.txt", ios::app);
    out_file_passed<<x_test<<' '<<'	';
    out_file_passed<<y_test<<' '<<'	';
    out_file_passed<<z_test<<' '<<'	';
    out_file_passed<<cosine_theta<<' '<<'	';
    out_file_passed<<sine_phi<<endl;

x_pass=0;
y_pass=0;
z_pass=0;
x_and_z_pass=0;
hyp=(seperation + 1);
}
x_pass=0;
y_pass=0;
z_pass=0;
x_and_z_pass=0;
}
hyp=(seperation - .01);
x_pass=0;
z_pass=0;
x_and_z_pass=0;
y_pass=0;