## Computational Astrophysics – Radiative Transfer:

1a.) A linear congruential generator, based on sequences of the following form, is used to generate pseudo-random numbers:

$$X_n = (aX_n + c) \bmod m \tag{1}$$

where $X_0$ is the starting value, known as the seed; a the 'multiplier'; m the modulus and c the 'increment', are all integers. A well-known minimal standard generator is used with $a = 16807, c = 0, m = 2^{31} - 1$. Other known multipliers: $a = 48271, a = 69621$ are used to generate different random number sequences as these have good 'randomness' and very long sequence lengths. Different prime seeds $X_0$= 107, $X_0 = 313$ are used to ensure the starting random number is different. These were checked to ensure that the combinations work and the random numbers to do not demonstrate any repeating patterns. The 'random_number()' function performs the equation from (1), and the output is then divided by m to get a random number between 0 and 1.

In the first part of the exercise the focus is on non-isotropic scattering processes. The following distribution over angles is

$$P(\mu)\, d\mu = \frac{3}{8}(1 + \mu^2) d\mu \tag{2}$$

where $\mu = cos\theta$ and $\theta$ is the angle measured from the ray direction and -1 ≤ $\mu$ ≤ 1. The distribution from equation 1 is normalised as:

$$\frac{3}{8}\int_{-1}^{1}(1 + \mu^2)\, d\mu = \frac{3}{8}\left[1 + \frac{1}{3} - \left(-1 - \frac{1}{3}\right)\right] = 1 \tag{3}$$

Figure 1 provides evidence that the random draws obtained in the implemented code do approach the required distribution for $N = 5000$. The random numbers appear randomly spaced over the whole range, all lying within the required distribution.

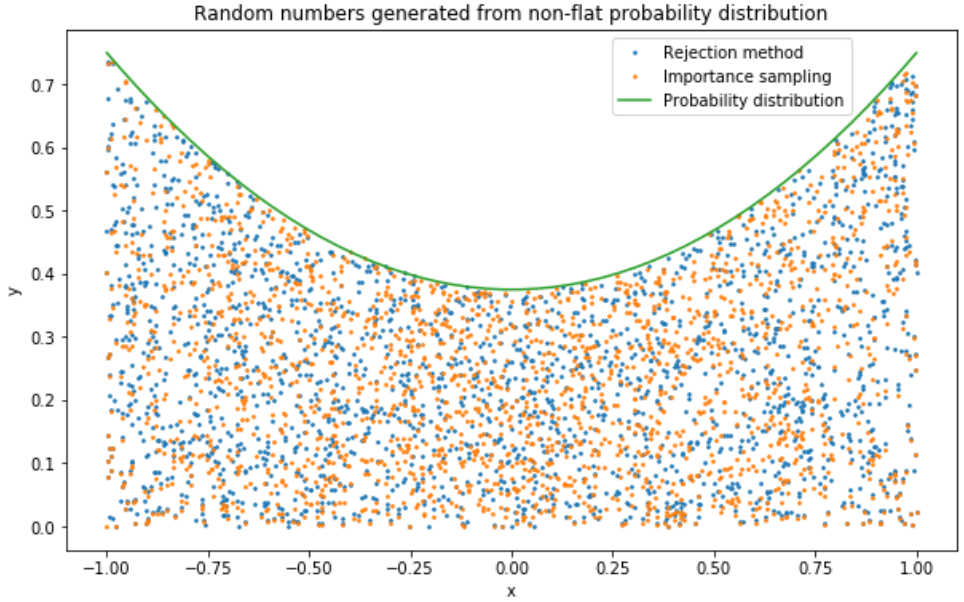


Figure 1: Random draws from given probability distribution using the rejection and importance sampling methods.

The rejection sampling and importance sampling methods were both used to generate random numbers from the non-uniform distribution. The random numbers between 0 and 1 were generated using the minimal standard generator. For the x values, these were scaled between the limits -1 and 1, and the y values scaled between 0 and the peak at 0.75. The two methods are both very simple and robust. The similarities and differences between methods will be discussed.

The rejection method draws random x and y values, and only accept those points which lie below the distribution. The importance sampling method can be easier to implement as it doesn't reject any points, but instead applies a weighting function,

$$w_i = \frac{P(x_i)}{Q(x_i)} \tag{4}$$

where $Q(x_i) = 0.75$, a flat distribution that is always greater than probability distribution $P(x_i)$. As an example, 5000 random numbers are required. The rejection method requires 7429 iterations, implying 2429 draws were wasted. The importance sampling method is less wasteful, requiring 5000 iterations. No draws are wasted, but instead a weight factor is applied to each draw. Both methods are computationally wasteful. The importance sampling method is only effective where $Q(x_i)$ closely follows $P(x_i)$ and can lead to a very high variance, especially where Q and P differ. As a result, we shall implement random numbers for the isotropic scattering process using the rejection method. The importance sampling method could have been improved by using a $Q(x_i)$ that follows the probability distribution more closely.

1b.) In this section, the random walk approach is used to simulate Monte Carlo radiative transfer of photons through an atmosphere. The first while loop: 'while(count_above<N_photons)' ensures the simulation is repeating until there are the required number of photons that have escaped from above the atmosphere. Initially, the photon coordinates and angles are set to zero.

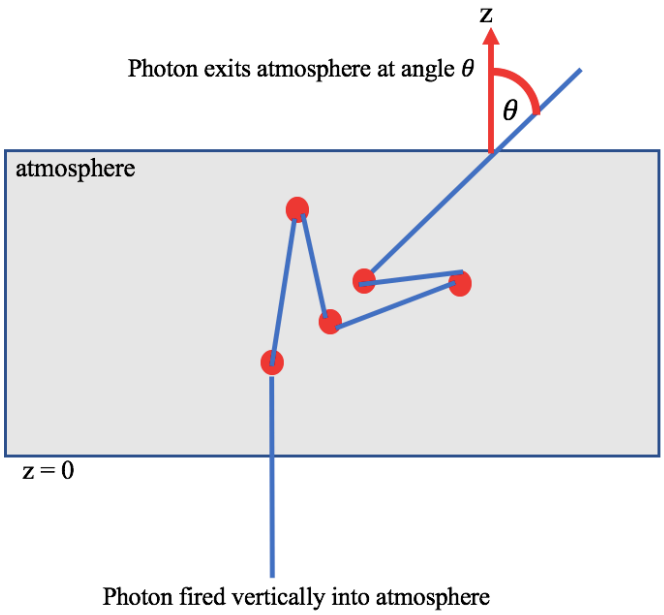


*Figure 2: Simplified diagram of a photon's path where the red dots represent a scattering*

The second while loop 'while(z>=z_min && z<=z_max)' loops through the scattering process (outlined below), whilst the photon is still situated within the atmosphere:

**1. Photon moved a random distance tau.** The distance tau is a random number drawn using the cumulative method from the probability distribution:

$$P(\tau) = e^{-\tau} \tag{5}$$

The cumulative distribution function:

$$P_{cum}(\tau) = \int_0^{\tau} e^{-\tau'} d\tau' = 1 - e^{-\tau} = y \tag{6}$$

To find the random tau value, random numbers y are drawn from a flat distribution and scaled so they lie between 0 and $(1 - \frac{1}{e})$. This ensures the tau value lies between 0 and 1 when calculated using the inverse of the cumulative function:

$$\tau = -\ln(1 - y) \tag{7}$$

The atmosphere is infinite in the x and y planes and we are only interested in the z direction. However, three dimensions are included for completeness. A random number, labelled 'rand', is generated between 0 and 1. The angles are calculated as below:

$$\phi = 2\pi * \text{rand}, \ \mu = (2 * \text{rand}) - 1, \ \theta = \cos^{-1}(\mu) \tag{8}$$

where $\mu = \cos\theta$. The co-ordinates of the photon are updated using the spherical polar coordinate system as below:

$$x = x + \left(\frac{z_{max}}{\tau_{max}}\tau\right)\sin\theta \ \cos\phi, \ y = y + \left(\frac{z_{max}}{\tau_{max}}\tau\right)\sin\theta \ \sin\phi, x = x + \left(\frac{z_{max}}{\tau_{max}}\tau\right)\cos\theta \tag{9}$$

The random value tau is converted to a physical length by dividing by the maximum tau value, in this case $\tau_{max} = 10$ and multiplying by $z_{max} = 1$. Therefore, if the photon was fired directly upwards, you would expect the photon so take 10 steps to escape from the atmosphere.

**2. Absorb or scatter.** A random number is drawn from a flat distribution between 0 and 1, labelled 'rand'. A simple if loop is included 'if(rand < a)' where 'a' is the albedo. If the random number is less than the albedo, scattering occurs, else, absorption occurs. As the albedo = 1, only scattering is present.

**3. Draw random direction.** Isotropic scattering implies that there is equal likelihood the photon will scatter in any direction. The scattering angle theta is measured from the vertical z axis and is a random number between 0 and $\pi$. Initially, the photon is fired vertically upwards, hence the initial theta for each photon is 0.

If 'z' is not within the maximum and minimum z values, it implies that the photon has left the atmosphere. If $z > z_{max}$ the photon has escaped above the atmosphere and we bin this photon according to its angle. The index of the angle, between 0 and 9 (as we have 10 bins), is calculated using 'int index = (int) (theta/d_theta)'. Where d_theta is the width of the bin.

If z $<$ $z_{min}$ then the photon has escaped below the atmosphere and we ignore this photon. For 1,000,000 photons escaping above the atmosphere, for this simulation, there are roughly 900,000 photons escaping below the atmosphere. This is approximately as you would expect. For isotropic scattering the scattering is random so roughly half of the photons will overall travel up, roughly half will travel down. As the photons are initially fired vertically upwards, it is expected more photons to travel upwards overall.

As each photon has identical energy, the normalised intensity is the number of photons within that bin, divided by the total number of photons. Figure 3 shows the output for N=1,000,000 photons. As expected, there is a larger proportion of photons exiting from the top of the atmosphere with $\theta = 0$. The smaller the angle, the closer the photon is travelling vertically upwards, hence cos$\theta$ is greater implying a larger jump in the z component and the photon is more likely to exit the atmosphere. When $\theta = \frac{\pi}{2}$ and the photon is travelling horizontally there is a very small contribution to the z component.

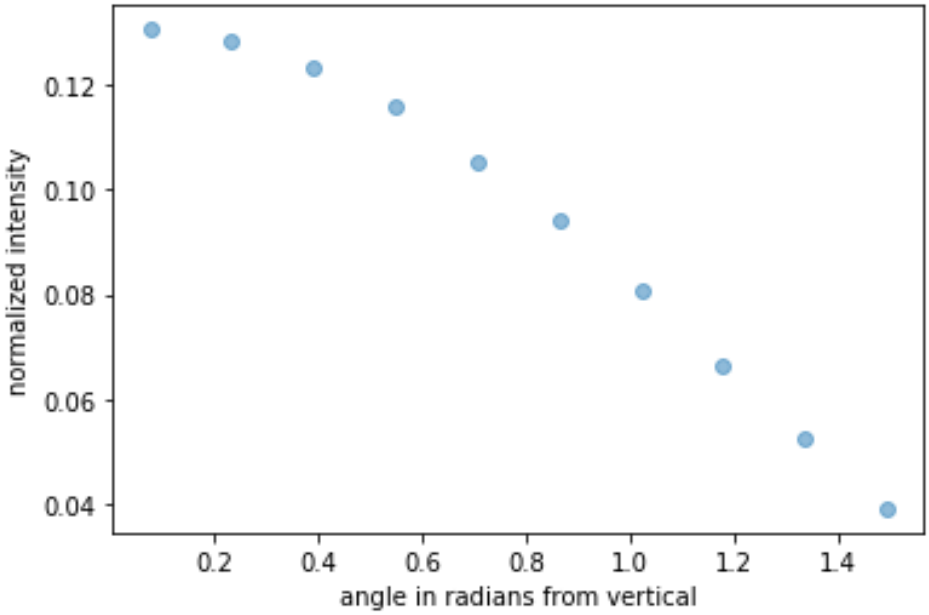


Figure 3: Normalized intensity against angle theta from z axis for isotropic scattering

1c.) In this final section, the scattering is non-isotropic. The angle of scattering follows the distribution:

$$P(\hat{\theta})d\Omega = \frac{3}{4}(1 + \cos^2\hat{\theta})\,\frac{d\Omega}{4\pi} \tag{9}$$

Substituting $d\Omega = \int_0^{2\pi} d\phi \int_0^{\pi} \sin\theta\, d\theta = 2\pi$ and replacing $\cos\hat{\theta} = \mu$ we end up with $P(\mu)\, d\mu = \frac{3}{8}(1 + \mu^2)d\mu$. This is the same probability distribution as in section 1a). However, here the angle $\hat{\theta}$ is the angle between the incoming and outgoing photon direction, rather than the angle relative to the z-axis. To simplify the problem, we shall incorrectly assume that $\hat{\theta}$ is instead relative to the z-axis, resulting in a non-physical solution.

The method for this non-isotropic scattering is similar to part 1b), but the scattering angle is not drawn from a flat probability distribution but from the probability distribution as seen in figure 1. This probability distribution implies that the photons are more likely to scatter either straight forwards or backwards.

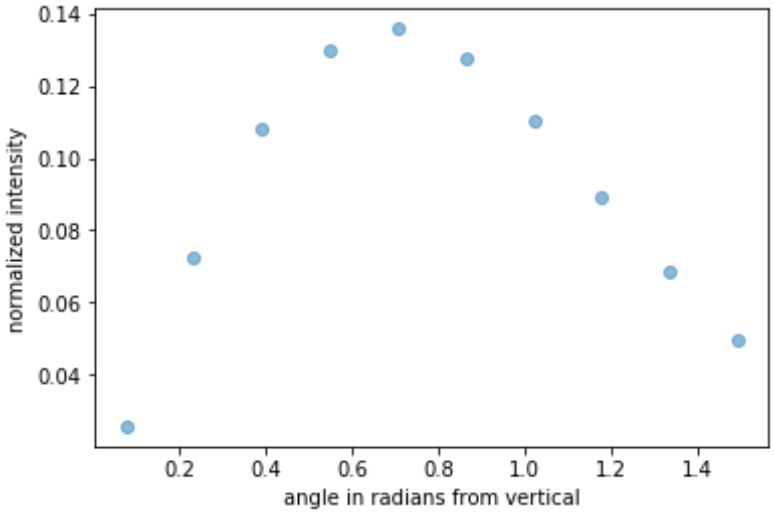


*Figure 4: Normalized intensity against angle theta from z axis for non-isotropic scattering*

To implement choosing an angle from a non-flat probability distribution, the rejection method will be used. We use the while loop: 'while(accept<1)' where 'accept' is the number of $\mu$ values that have been accepted, to ensure that new random numbers are drawn until they fit the probability distribution. The same rejection method is applied from part 1a.

Figure 4 shows the output of photon angle distribution from the non-flat probability distribution. Unlike isotropic scattering, the non-isotropic scattering has output photons peaking at around $\theta = \frac{\pi}{4}$ rather than $\theta = 0$. In addition, for every 1,000,000 photons that escape from above the atmosphere, 757,823 photons escape from below the atmosphere, which is less than in the isotropic scattering process. With non-isotropic scattering the process is less random, the photon is more likely to travel along a straight line.

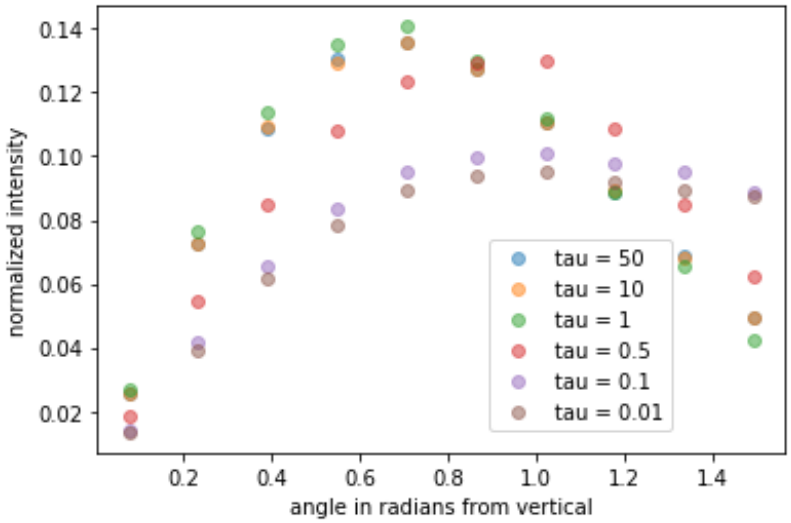


*Figure 5: Normalized intensity against angle theta from z axis for non-isotropic scattering*

Figure 5 shows the photon output for the non-isotropic scattering process for a range of total optical depth (tau) values. For larger tau values, where the atmosphere is in the optically thick range the angle at peak intensity remains around $\theta = \frac{\pi}{4}$. Decreasing tau, entering the optically thin range, the distribution of intensity against angle seems to broaden, with a larger proportion of photons exiting the atmosphere with angles closer to $\theta = \frac{\pi}{2}$.

# Appendix:

## Appendix part A – initialize variables and functions:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define res 1000

long long random_number(long long seed, int a, int m, int c)  //random number
function
{
  return (seed*a + c) % m;
}

double probdist(double x)  //normalized probability distribution function
{
  return (double) (3./8.)*(1.+(x*x));
}

double f(double y)  //inverse cumulative function where y is random number: 0 to 1.
y_in is scaled: 1/e to 1 so that corresponding tau is between 0 and 1
{
  double y_min, y_length, y_in;
  double e = 2.7182818284590;
  y_min = (1./e);
  y_length = 1 - y_min;
  y_in = (double)  y*y_length;
  return (double) -1.*log(1-y_in);
}

//INITIALIZE VARIABLES
double x_reject, y_reject, fx;  //rejection method variables
double x_im, y_im, w; //importance sampling variables
int i, j; //loop counter
long long seed = 1; long long seed2 = 107;  long long seed3 = 313; long long m =
2147483647;  //variables for random number generation
```

## Appendix B – section 1a:

```
int part_1a()
{
  FILE *fptr_reject;
  fptr_reject = fopen("rejection_method.text", "w");
```

```
  int N_accept = 0; //accepted x values counter
  int target = 5000; //target number of x values
  int count = 0;  //count iterations for rejection method

  //********************REJECTION METHOD**********************************
  while (N_accept<target)
  {
    count++;  //counter for number of interations
    seed = random_number(seed, 16807, m, 0);      //generates random numbers from 0
to 1
    seed2 = random_number(seed2, 48271, m, 0);       //generates random numbers
from 0 to 1

    x_reject = (double) seed*2./m -1.;  //x range of random numbers from -1 to +1
    y_reject = (double) seed2*0.75/m;  //y range of random numbers from 0 to peak
(+0.75)
    fx = (double) probdist(x_reject);   //p(x)

    double x_accept[N_accept];   double y_accept[N_accept];   //create two arrays

    if(y_reject<=fx)
    {
      N_accept++;
      x_accept[N_accept] = x_reject;
      y_accept[N_accept] = y_reject;
      fprintf(fptr_reject, "%.3f, %.3f\n", x_accept[N_accept], y_accept[N_accept]);
//print accepted (x,y) values to text file
    }
  }
  //**************IMPORTANCESAMPLING*********************************
  int count_im = 0;   //count iterations for importance sampling method
  FILE *fptr_importance;
  fptr_importance = fopen("importance_sampling.text", "w");

  for(i=0; i<target; i++)
  {
    count_im++;   //counter for number of interations
    seed = random_number(seed, 16807, m, 0);
    seed2 = random_number(seed2, 48271, m, 0);

    double x_values[target];    double y_values[target];   //create two arrays for
importance sampling method

    x_im = (double) seed*2./m -1.;  //x range of random numbers from -1 to +1
    y_im = (double) seed2*0.75/m;  //y range of random numbers from 0 to peak +0.75
    fx = (double) probdist(x_im);

    w = (double) fx/0.75;   //weighting factor

    x_values[i] = x_im;
```

```
    y_values[i] = (double) (y_im*w);

    fprintf(fptr_importance, "%.3f,%.3f\n", x_values[i], y_values[i]);    //print
(x,y) values to text file
  }
  printf("\nPart 1a\n");
  printf("number of iterations (rejection) = %d, number of iterations (importance)
= %d\n", count, count_im);
  return 0;
}
```

**Appendix part C – section 1b:**

```
//INITIALIZE VARIABLES
int z_min = 0; //bottom of atmosphere
int z_max = 1; //top of atmosphere
double a = 1; //albedo = 1 meaning all scattering and no absorption
double x, y, z, dx, dy, dz, tau, rand1,rand2, theta, phi, mu; //variables
double tau_max = 10;  //total optical depth
int N_photons = 1000000;  //required number of photons
int N_bins = 10; //number of bins

int part_1b()
{
  FILE *fptr;
  fptr = fopen("part_1b.text", "w");

  //create array of bins and set values to zero
  int array[N_bins];
  for (j=0; j<N_bins; j++)
  {
    array[j] = 0;
  }

  double d_theta = (double) (M_PI/2/N_bins);  //bin width

  int count_below = 0;
  int count_above = 0;     //counters for number of photons above/below atmosphere
  while (count_above<N_photons)
  {
    z = 0; x = 0; y = 0;  //each new photon starts at origin
    theta = 0;  phi = 0;  mu = 1; //each new photon originally fired upwards

    while (z>=z_min && z<=z_max)
    {  //the photon has NOT escaped and is within atmosphere boundaries
      seed = random_number(seed, 16807, m, 0);
      rand1 = (double) seed/m;  //random number between 0 and 1
      tau = (double) f(rand1);  //random distance tau between 0 to 1 from
exponential func using cumulative method
      z = z + tau*1/tau_max;  //update positions
      x = x + tau*1/tau_max;
      y = y + tau*1/tau_max;
```

```
      //should we scatter or absorb?
      if(rand1 < a)
      {
        seed2 = random_number(seed2, 48271, m, 0);  //second random number for
random new photon direction between 0 and pi
        rand2 = (double) seed2/m ; //random number between 0 and 1

        phi = 2*M_PI*rand2;
        theta =  M_PI*rand2;

        z = z + cos(theta)*1/tau_max;     //update positions
        x = x + sin(theta)*cos(phi)*1/tau_max;
        y = y + sin(theta)*sin(phi)*1/tau_max;
      }

      else
      {
        break;     //absorb and remove photon
      }

    }

    if(z<z_min)
    {
      count_below++; //photon escapes from below atmosphere - restart, new photon
    }

    if(z>z_max)
    {
      //photon escapes from above atmosphere - bin photon
      int index = (int) (theta/d_theta);  //index of bin
      array[index] = array[index]+1;  //add one for every photon escaping with
angle within that bin
      count_above++;
    }
  }
  printf("\nPart 1b\n");
  printf("No. photons escaped above: %d, No. photons escaped below: %d\n",
count_above, count_below);

  for (j=0; j<N_bins; j++)
    {
    double mu_mid = (double) (d_theta*j) + d_theta/2;    //angle from midpoint of
bin
    double norm_int = (double) array[j]/N_photons;  //normalized intensity
    fprintf(fptr, "%d, %e, %e\n", j, norm_int, mu_mid);   //print normalized
intensity against midpoint of bin
    }
    return 0;
}
```

**Appendix part D – section 1c:**

```
int part_1c()
{
  FILE *fptr;
  fptr = fopen("part_1c.text", "w");

  //create array of bins and set to zero
  int array[N_bins];
  for (j=0; j<N_bins; j++) {
    array[j] = 0;
  }

  double d_theta = (double) (M_PI/2/N_bins);  //bin width

  int count_below = 0;
  int count_above = 0;

  while (count_above<N_photons)
  {

    z = 0; x = 0; y = 0;  //each new photon starts at origin
    theta = 0;  phi = 0;  mu = 1; //each new photon originally fired upwards


    while (z>=z_min && z<=z_max)
    {   //the photon has NOT escaped
      seed = random_number(seed, 16807, m, 0);
      rand1 = (double) seed/m;  //random number between 0 and 1
      tau = (double) f(rand1);  //random distance tau between 0 to 1 from
exponential func using cumulative method
      z = z + tau*1/tau_max;   //update positions
      x = x + tau*1/tau_max;
      y = y + tau*1/tau_max;

      //should we scatter or absorb?
      if(rand1 < a)
      {
        int accept = 0; //counter- want one accepted value for mu (from rejection
method) for each loop
        while(accept<1)
        {
          seed2 = random_number(seed2, 48271, m, 0);
          seed3 = random_number(seed3, 69621, m, 0);

          double x = (double) seed2*2./m -1.;  //x range of random numbers from -1
to +1
          double y = (double) seed3*0.75/m;   //y range of random numbers from 0 to
0.75
          double fx = (double) probdist(x);   //p(x)
```

```
          if (y<fx)
          {
            mu = x;
            accept++;
          }

        }
        theta = acos(mu);
        z = z + cos(theta)*1/tau_max;    //update positions
        x = x + sin(theta)*cos(phi)*1/tau_max;
        y = y + sin(theta)*sin(phi)*1/tau_max;
      }

      else
      {
        break;     //absorb and remove photon
      }
    }

    if(z<z_min)
    {
      count_below++; //photon escapes from below atmosphere - restart, new photon
    }

    if(z>z_max)
    {
      //photon escapes from above atmosphere - bin photon
      int index = (int) (theta/d_theta);  //index of bin
      array[index] = array[index]+1;  //add one for every photon escaping with
angle within that bin
      count_above++;
    }

  }
  printf("\nPart 1c\n");
  printf("No. of photons escaped above: %d, No. of photons escaped below: %d\n\n",
count_above, count_below);

  for (j=0; j<N_bins; j++)   //calculating normalized intensity and mid of bin of
angle for plotting
  {
    double mu_mid = (double) (d_theta*j) + d_theta/2;    //angle from midpoint of
bin
    double norm_int = (double) array[j]/N_photons;  //normalized intensity
    fprintf(fptr, "%d, %e, %e\n", j, norm_int, mu_mid);   //print normalized
intensity against midpoint of bin
  }
  return 0;
}
```