

MPhys Project Report

Multiple Scattering By Aerosols

1 Abstract

As part of the current research into aerosol properties with an aerosol cell at the Rutherford Appleton Laboratory, it was required to investigate the effect of photon scattering within the cell. This was to be done by modifying an existing Monte Carlo computer program that was written to model scattering in a simple cylinder cell. The changes include alteration to describe the current cell set-up, and take into account other parameters previously not included in the code, such as absorption of the photons by interaction with the aerosol. The new program was used to simulate the sending of a beam with a large number of photons to determine how the amount of multiple scattering varies with changes in the optical depth, asymmetry parameter and single scatter albedo. This enabled the use of data about the variation of the asymmetry parameter and single scatter albedo with wavenumber to show how the optical depth at which multiple scattering becomes a significant contribution to the total number of photons detected.

2 Definitions and Introduction

When a beam of light is incident through a chamber containing an aerosol, the photons in the beam interact with the particles. This interaction is described by what is known as Mie Theory, which is discussed below. Each photon in the beam travels in the cell until it either reaches an aerosol particle, where it is then either absorbed or scattered, or until it hits the cell wall, or until it is detected. Before proceeding further, it is necessary to define a number of terms, which will be used throughout the text: -

- Transmission – This is defined by $\text{Transmission} = I/I_0$ (varies from 0 to 1). Where I and I_0 are defined as the number of photons leaving and entering the cell respectively.
- Optical depth – This is defined as a factor which measures the amount of extinction a beam of light experiences travelling between two points. Mathematically, its definition is given by $\text{Optical depth} = -\log_e(\text{Transmission})$. It varies from minus to plus infinity.
- Single scatter albedo (called Omega in the programs) – This is a parameter which determines the chance of a photon being absorbed in any one collision with an aerosol particle, rather than being scattered. It can take any value between 0 and 1, with 1 being no chance of absorption, and 0 being certain absorption.
- Asymmetry parameter (denoted by G) – This variable (again taking values between 0 and 1) determines the angular distribution of scattered particles as a function of incident angle. For example whether they are scattered isotropically (0), or only forward (1 is the delta function in the forward direction).
- Acceptance Angle – This is the angle of transmission from the cell, within which the photons must fall if they are to be detected by the apparatus.
- Multiple scattering – is defined as scattering whereby the photon concerned has undergone more than one interaction with the aerosol in the cell set-up.

Three properties are required to exactly specify the radiative effects of particles, these are the optical depth, single scatter albedo and asymmetry parameter. These properties depend on the mass, size and composition of the particle species. The solution of Maxwell's equations for the geometry of the aerosol (usually considered to be spherical) in the medium being studied yields the connection between the refractive index, n and the three single scattering properties. The complete solution to this is Mie Theory, which predicts the optical efficiencies of particles of a given size at given wavelength. The optical efficiency Q_x (where x represents e, s or a) for each of the processes (extinction, scattering and absorption respectively) is the optical depth of a single particle due to the process divided by the cross sectional area of the particle. These

optical efficiencies define the three important properties that are used extensively in this project. They are not independent of one another, since extinction is the sum of absorption and scattering. The mathematical definitions of optical depth, single scatter albedo and the asymmetry parameter respectively are shown below: -

$$\tilde{\tau}_e(z, \lambda) = \int_0^{\infty} \pi r^2 Q_e(r, \lambda) n(r, z) dr$$

$$\tilde{\omega}(z, \lambda) = \left(\int_0^{\infty} \pi r^2 Q_s(r, \lambda) n(r, z) dr \right) / \tilde{\tau}_e$$

$$\tilde{g}(z, \lambda) = \frac{\int_0^{\infty} \pi r^2 g(r, \lambda) Q_s(r, \lambda) n(r, z) dr}{\int_0^{\infty} \pi r^2 Q_s(r, \lambda) n(r, z) dr}$$

An existing program modelled the interaction of photons in a beam of light entering a small cylindrical cell containing aerosol using the IDL (Interactive Data Language) programming language. Consequently, the first task was to learn how to use IDL and understand thoroughly the workings of the current simulation program, in order that it would be possible to tackle the difficult problem of modifying the code to allow the aims of the project to be achieved.

After fully understanding the existing program, the next task was to change the program so that code was added to take into account the chance that on collision with the aerosol in the cell, a photon is absorbed. Altering the value of the relevant variable (the single scatter albedo) could then change the probability of absorption.

The next step was to add code so that the program wrote to a file data relevant to obtaining the final result. At this point, loops were also added in the program to vary optical depth and single scatter albedo. By running this program for different values of G (The asymmetry parameter), contour plots could be obtained (by writing program code to do so) showing lines of constant percentage enhancement due to multiple scattering.

It was then necessary to alter the program to reflect the dimensions of the new cell set-up. This involved quite a lot of thought, as there were a number of problems with writing the code correctly to test whether or not the photon had escaped from the cell, as will be described later.

The final major change to the program was required to be implemented, where lines of code were needed to determine if the photons leaving the cell reached the detector, or whether they were outside the acceptance angle. Once all these changes had been made the program could be run with a loop so that it ran for all the different asymmetry parameters, and stored all the corresponding data on file. After having to re-run the program with different output data due to statistical problems with the original results, it was then possible to use the stored information to obtain the final results of how the optical depth at which multiple scattering becomes significant varies with wavenumber.

3 Technical Details

3.1 The Cell Being Modelled

The cell containing the aerosol consists, for the purposes of the modelling program, of two cylindrical cells, arranged so that one is contained within the other, at right angles to it, so it sticks out of the side. The diagram of the apparatus is shown here, in figure 3.1, where the cells modelled by the computer simulation are the large vertical cell within the outside casing (with winding round it on the diagram) and the cylinder horizontally within it. Photons enter this horizontal cell from the left-hand side, and are detected by the detector box on the right. This contains a mirror to focus the beam onto a detector; this will be further discussed later when the optical information is added to the simulation.

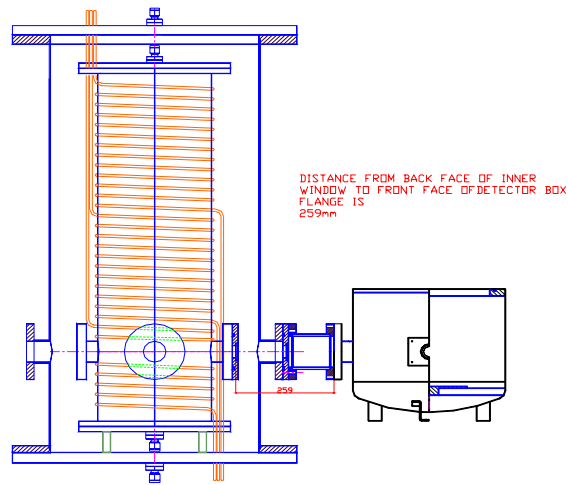


Figure 3.1: The aerosol Cell to be modelled

This contrasts widely from the cell set-up in the original program, which only looked at a single cylinder where photons entered from the left and exited to the right.

3.2 The Computer set-up

The computer used for carrying out this investigation was a Linux based machine, using the IDL programming language. Due to limited experience with the operating system, initially the basic commands on Unix that would be needed in the day to day use of the computer to adapt the code had to be learnt. The main problem was in understanding the IDL language itself.

4 Experimental Method

4.1 Understanding the Original Program

The original program was called beam2.pro. This program uses Monte Carlo modelling to find the exit points of photons sent through the simple cylindrical cell. This method of modelling reconstructs real events by selecting a random sample of particles (in this case photons), and tracks them through the computer model, calculating everything that happens to them. Thus the Monte Carlo algorithms determine the photons' exit points by following the path of representative particles as they travel through the cell. The probabilistic laws of physics prevent us from knowing the exact fate of each particle, but allow us to predict the distribution at the exit by recording the output of thousands of sample photons sent through the simulation. An accurate representation of what happens is thus built up, but like all models involving probability, the Monte Carlo model induces statistical errors the results.

The most effective way to attempt to gain a better understanding of how the whole code worked was to run it a few times, and attempt to see what it did.

The program was executed with different values of optical depth, and it was easy to see that the speed of running was much slower with a value of 10 than with a value of 2 for example. This was to be expected, as a greater optical depth gives rise to more collisions with particles in the aerosol, and so a more time consuming process as photons are scattered more

and more. It was also noticeable that on the output plot that a large optical depth led to there being little difference between the number of photons in the beam radius compared to the number outside. Again, the reason for this is because multiple scattering has occurred, and there are very few direct photons. Further experimentation with the program showed that when the beam radius was increased to a more realistic value, the program failed to run, saying “Unable to allocate memory to make array, not enough space”. It was discovered later that this is because by enlarging the beam radius, the number of photons sent is also increased, so more memory than previously allocated to the program is required.

Another observation made during this experimentation was that when plotting was being carried out of the output photons’ exit position on the screen, the beam appeared to “fill” from left to right. The reason for this is that the program works by sending the photons into the cell using a grid that sends a set number of photons from each point on a grid that makes up the area of the entrance window. This means that the photons that go directly through will appear to fill in this regular pattern when plotted as each column of the grid is filled in turn.

The IDL help facility was then utilised to find out what the function was of many of the commands used in the code of beam2.pro as they were not immediately obvious to a relative novice in the language; this information was noted down for future reference.

In attempting to determine further what individual parts of the program code did, “Stops” were inserted into the code, and then IDL was instructed to continue after each, so that an insight could be gained into the function of each part. This information, together with discussion with the original programmer about the parts of the program that were still not fully understood, allowed a fairly good initial understanding of the program code to be developed. See Appendix 7.1 for the code listing of the original code, beam2.pro. This has been divided up into six parts, for the purpose of explanation as to what each part does.

The first part, labelled 7.1.1, defines constants that are used throughout the program, and sets the number of photons to be sent per grid point. Some of these constants will become variables in the adapted code, e.g. Optical_Depth and G (The asymmetry parameter).

Section 7.1.2 sets up the graphics for the display of plots of photon exit points, including plotting the circle which represents the radius of the cell window, and the hashed circle showing the original beam radius. The colour code for showing the number of scatters a photon has gone through before emerging is also plotted onto the graphical display at this stage. As well as this, the grid that forms a crucial part of the programming model is defined. The cell radius is split up into a number of rows and columns, and it is from the points where these grid lines meet that each set of photons is sent into the simulated cell. Arrays representing this grid set-up are defined here.

The next part (7.1.3) sets up the number of photons being sent, and defines arrays for storing the exit point, number of scatters and exit angle of each photon. Also defined here are the variables that store different position and movement vectors for the individual photons.

After this, in 7.1.4, the main simulation of photon interaction with the aerosol in the cell is carried out. Loops are followed for each photon within each grid point, so that the process is carried out once for each photon sent. Firstly, the initial direction and distance travelled before the first scatter are found, using the extinction value set using the optical depth, and a random number generated between 0 and 1. A repeat loop is then entered, which continues until the photon is outside the cell (tested by the use of a true/false flag). Within this, there is an IF loop, which has two different results, depending on whether the photon is within the cell or not. If it is, then a new scatter occurs, and the program code calculates the new location of the photon, and adds one to the count of the number of scatters encountered by that particular photon. The loop is then repeated, again testing to see if the photon is inside the cell or not. If it is outside the cell, then the number of scatters is stored in an array, and the flag determining whether it is outside or not is set. A test is consequently carried out to find out what happened to the photon – whether it was reflected back out of the entrance window, whether it was absorbed by hitting the side of the cell, or whether it was successfully transmitted through the exit window. This is carried out by looking at where along the final vector the walls are crossed. One is added to the correct counter, depending upon the fate of the photon. If the photon was transmitted, then the

program stores the exit point information and plots on the display graph on the screen the exit point, in a colour determined by the number of scatters which the photon has had.

The next section, 7.1.5, stores the exit point, angle and number of scatters undergone by the photon, and displays on the screen information about the total number reflected, lost and transmitted. A test is at this point carried out to ensure none of the photons have been lost, and that they are all counted.

The final part of the code (7.1.6) carries out a number of plots displaying information about the simulation. Most of this part of the code is not needed in the adaptation of the code to achieve the aims of this project, so there is no real need to explain the workings of the code in detail. To sum up, this part of the program first displays on the screen a plot similar to the one carried out while simulation takes place, showing exit points of the photons. The expected and actual number of direct photons is printed on the screen, as is the maximum number of scatters encountered by any photon. The final part of 7.1.6 plots four diagrams on the screen, displaying different information such as all the photons, those only within Acceptance angle etc.

4.2 Adding Absorption Code

The first task in adapting the code was to include program code to account for the single scatter albedo. This is where there is a chance (which varies with the value of the parameter) that on interaction with the aerosol, a photon is absorbed, and not just simply scattered. To do this, code needed to be inserted to generate a random number from 0 to 1, then see if this was greater than the single scatter albedo, to determine whether the photon is absorbed or not. Consequently, if the single scatter albedo is set to be 1, the random number generated will never be bigger than this, so absorption never occurs, and vice versa, if it is set to be 0, then all interactions between photons and aerosol particles will result in absorption.

As well as doing this, it was also necessary to add in a flag, similar to that used to determine whether the photons are outside of the cell, to set if a photon is absorbed. To keep count of the number that have been absorbed, a variable was also added, named “absorbed_photons” which is incremented as a photon is absorbed. Whether or not the flag is set determines which path the program follows. The initial attempt to write in the extra code failed, because the condition for testing whether absorption had occurred was put in the wrong place. On correcting this and running the program, it was discovered that despite initially appearing to run successfully, the program was not in fact working, since it was not counting all the photons sent as being either reflected, transmitted, escaped or absorbed. After much thought, it became apparent that the reason for this was that those photons absorbed after being multiply scattered were not being counted, as the computer skips over much of the code in this instance, including the part which counts the number absorbed. To try to rectify this problem, firstly an extra flag was inserted, “absorbedp”, to attempt to test whether photons were absorbed correctly even if they had been multiply scattered. Despite a large amount of time spent analysing the code, it was not possible to see why the program did not count all the photons.

It was decided that the best thing to do in an attempt to solve the problem would be to start the changes again, but this time first drawing a flow chart to see what exactly the program was required to do, and in what order, as shown in figure 4.1 on the next page.

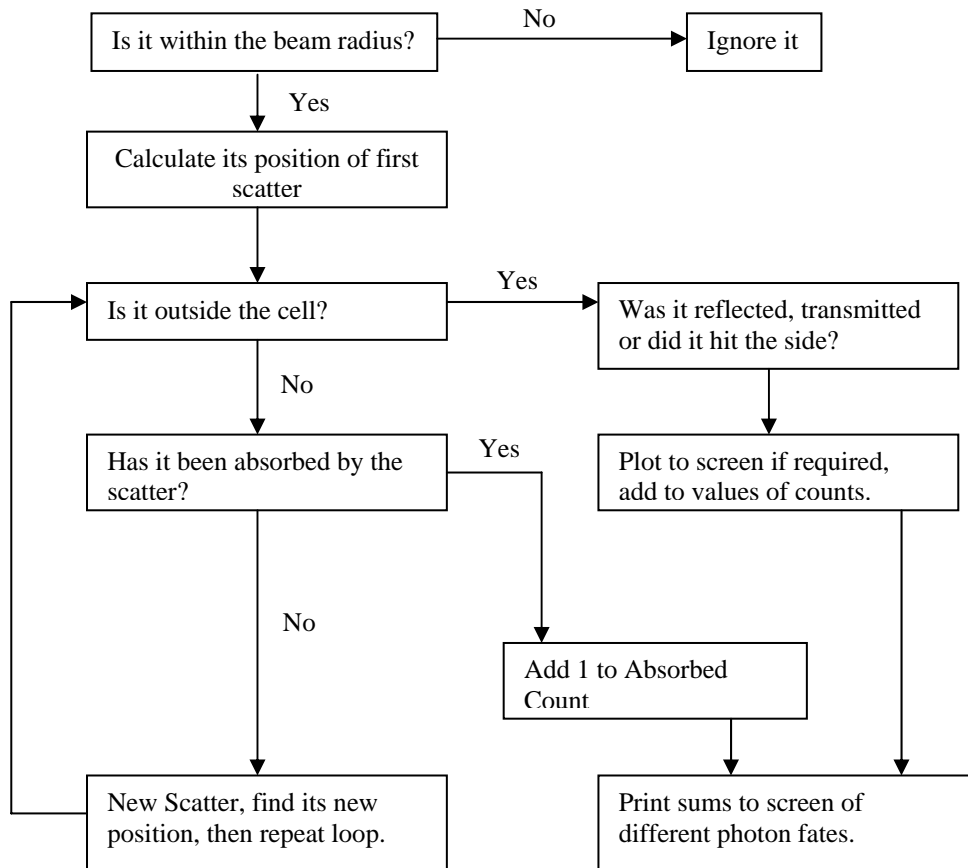


Figure 4.1: Flow Chart illustrating program adaptation to include absorption

This flow chart was then used to change the program code, and after a few minor problems, it ran successfully. Thus, the single scatter albedo can now be altered so as to give different amounts of photon absorption. This new program is called beam4.pro (beam3.pro was my unsuccessful attempt).

4.3 Adding loops to vary optical depth & asymmetry parameters

The next step was to use this new program, along with new code added, to make a program which runs the beam4 code for a variety of different values of G (the asymmetry parameter), optical depth and ω (single scatter albedo). The aim was to get plots, for each G , of ω vs Optical depth, with contours of constant values of enhancement due to multiple scattering. This program would need commands for changing the variables and lines for saving the results to file, with the same basic code inside the loops.

The results would need to be calculated for a grid of single scatter albedo values from 0 to 1, in 0.05 increments, and of optical depth varying exponentially from 0.3 to 30 in 10 steps. The code for the later plots (most of appendix 7.1.6) was at this point commented out (IDL was instructed to ignore it), as this was not needed for my plots, so could be ignored when the program compiled.

Including the code to calculate the data that needed to be saved to file (to be used to create the plots) completed the changes. These were the expected transmission if only extinction was a contributing factor (only direct photons), the actual transmission and finally, the deviation due to multiple scattering from the extinction-only value. (Found by subtracting the value for extinction only from the actual transmission value, and dividing this by the extinction-only value).

This new program was then tested for set single scatter albedo and optical depth values to get an estimate of the time taken for one run of the program. This enabled the number of

photons that should be sent to be calculated (and the corresponding grid resolution) so as to allow the program to run overnight.

Unfortunately, following the initial run of the program, the data file was empty, as an arithmetic error had occurred. A large amount of time was then spent attempting to sort out this problem by gradually sending a smaller number of photons, to see at what number of photons the problems started happening. The error could not be found at this stage, despite these measures, so a change was made: the single scatter albedo and optical depth were varied using integer counters, which were then used to get the value of the variables, in an attempt to ensure that, for example, Omega went up to 1.000000 exactly.

On running the program again, an error occurred due to the array being too small for the data. This time the data had been written up to Omega (single scatter albedo) = 0.85, so the data that had been written was used to create a test contour plot (although this would have to be sorted out later to obtain the correct final results). To do this, another program needed to be written which included a correctly sized array to read in the data from the file. This array was 5 by 198 (the same dimensions as the data). The last column of this array was then stored in the array Z, which is consequently an array containing the percentage deviation from what the transmission would be without multiple scattering. Next, this was reformed, using the IDL Reform command, to arrange the data in columns and rows according to the single scatter albedo and optical depth values. This then allowed the IDL Contour command to be used to plot the data. This contour plot was for one G (= 0.6) – see later for a more complete version of this plot with all data (all Omega values) included.

Now that the procedure for contour plot creation had been developed, the program needed to be corrected so it would execute for all values of single scatter albedo from 0 to 1, then get a complete set of data for all G (asymmetry parameter) values. A significant change to the program was then made to eradicate the arithmetic errors that were preventing a full set of results from being obtained. The array named “Photons_to_Count” was set to be equal to “Photons_to_send”, to ensure that the array was big enough to avoid the error. On running the program again, a full set of data for all the required single scatter albedo and optical depth values was obtained. Despite this, an error was still returned on the screen after the program finished, displaying “Arithmetic Error : Floating illegal Operand”, but after discussion with a more experienced IDL programmer, it was realised that this was due to the computer dealing with a very big/small number, and didn’t affect the program’s ability to write the data to the file, so could be ignored.

4.4 Creating Contour plots

With the program now running correctly, the next task was to create a good set of contour plots using all the data for each G value. The first thing to do was to make changes to the code of the contour program (contour.pro) to take into account the increase in data in the file. This was carried out, but the plot produced was not satisfactory, due to gaps in the contours. It was necessary to send more photons than the number currently being sent, in order to make sure that at least one on average gets transmitted for given variables. To calculate how many needed to be sent, the transmission formula, $T = \exp(-t)$, was used to work out how many photons needed to be sent for the maximum optical depth to get an average of 1-2 transmitted. This was approximately 300,000 photons. Another change that was made at this point, in order to try to achieve a better plot, was to increase the number of optical depths for which data was obtained by a factor of two.

After running the program again, a full set of data was obtained, which was then plotted after making the appropriate changes to the plotting program to reflect changes to the data. The plot was very good (see later for example plots), with clear contour lines without the gaps that had previously been experienced. As a result, it was now possible to begin creating plots for all the different values of the asymmetry parameter (G). To do this, the code in the current program (now beam5.pro) was changed to vary G, and output the data to a file whose name varied depending on the value of G in the FOR loop. Due to the large number of photons sent, and

large number of data values needing to be obtained, it would take a number of days for this program to execute.

On its completion, the contour program was altered to give two plots for each G value, one for the percentage deviation from the extinction model due to multiple scattering and one for the percentage error in this value. The data used to generate these plots would be used later to obtain the final results for the simple model (single cylinder cell).

4.5 Changing the program to represent the new cell set-up

The next change which was required to be implemented in the program code was that which took into account the fact that the aerosol cell was not simply one single cylinder, but two, as discussed in section 3.1. To do this, amongst other things, a co-ordinate transform would be needed, and used to determine whether a photon was within the large cylinder. This turned out to be relatively simple to accomplish: -

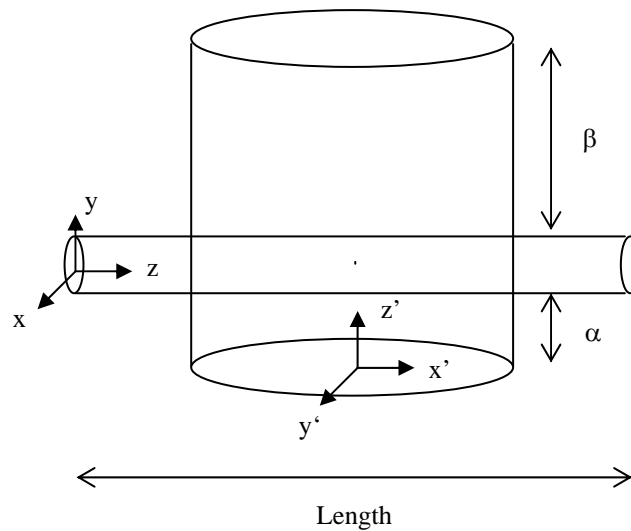


Figure 4.2: The co-ordinate system of the big cell

The relation between the co-ordinates in the big cell to those in the small cell is:-

$$\begin{aligned}x' &= z - 0.5(\text{Length}) \\y' &= x \\z' &= y + 0.5(\text{Cell_Radius}) + \alpha\end{aligned}$$

The new code, which uses this transform, would need to follow the flow chart shown below in figure 4.3: -

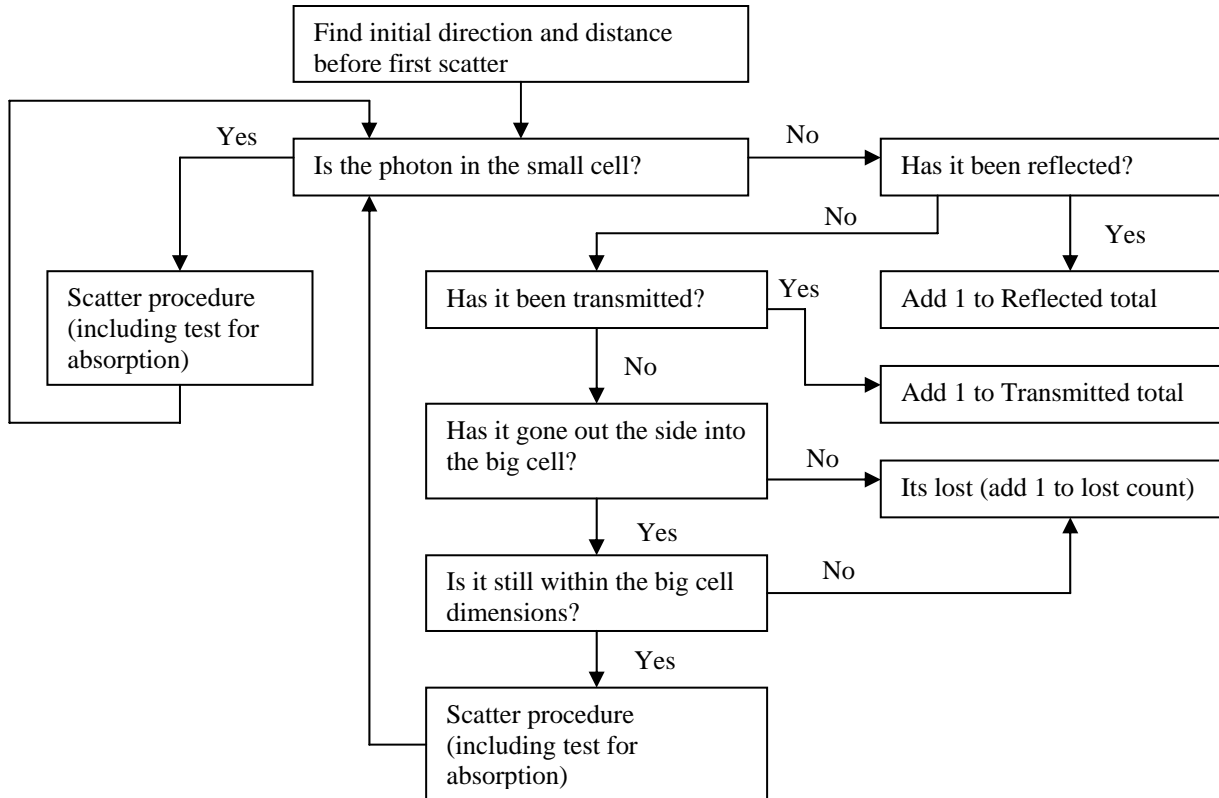


Figure 4.3: Flow chart showing program logic to include code for the new cell set-up

This was then written into the program code, in the following way: -

- Added variables for BigCell_Radius, S_out (the distance the small cell sticks out) and α and β (as defined above).
- An extra array, $K = \text{fltarr}(3)$ (a 3 dimensional array), was defined as the position of a photon in the co-ordinates of the big cell.
- The program tests as before to see if a photon is in the small cell. If so, it repeats as before, repeating the loop until it is outside or absorbed.
- If it is outside the small cell, tests are done as on the flow chart above to see what happened to it. For the case where its z position < 0 in the small cell co-ordinate frame, the program tests as before to find out if it left through the side wall (if exit point was between $z = 0$ and S_{out}) or was reflected. Flags are set accordingly.
- If $z > (\text{length})$ it again finds if it left through the side (if the exit point was between $z = (\text{length} - S_{\text{out}})$ and (length)) or went out of the end window.
- Otherwise, the photon may have gone out the side into the big cylinder. Before it can be established whether it did, it needs to be checked whether or not the exit point from the small cell meant it entered the big cell, or escaped.
- If it did enter the big cell, then the co-ordinates are transformed to those of the big cylinder, and a test is carried out to see if it is still inside or whether it escaped. If inside, then the scatter procedure is executed, and the loop returns to the start and begins again.

A counter was then put into the program to keep a record of the number of transmitted photons that had entered the big cell. This did not work, and always gave a value of zero, so something was obviously wrong with the program. After a great deal of consideration, it was apparent that the program was using something in the original code called `Exit_Point`, to test where the exit point from the small cell was. This was always equal to zero, since the part of the program where it is defined is not executed in the new version when the program follows the loop where a photon is in the big cell. Also, at this point, no code had been written to test for the situation where a photon enters the big cell by going out the side of the small cell between $z = 0$ and S_{out} and outside first, which needs to be forbidden. (This is shown below, in figure 4.4).

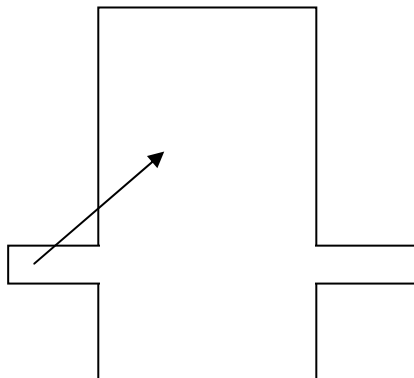


Figure 4.4: A forbidden situation

It was decided that the method that had been employed was not the best way to implement the changes – there is a much neater way, as described below.

The alternative solution is to test whether a photon is in the big cell or the small cell at the same time, then scatter. To do this, a neat trick is used where the following flags are used: -

- `In_small_cell`
- `In_big_cell`
- `Was_in_big_cell`
- `Was_in_small_cell`

This allows the above problem to be solved quickly and easily; for example, if a photon was in the big cell and not in the small cell, and is now in the small cell and not the big cell, then it is lost, as it must have hit the side. There is no longer a need to find out where the boundary was crossed in this part of the code. The latter two flags are used by the code to determine where the photon was before the last movement, and so this information combined with the set state of the other two flags allows the path to be determined, and the appropriate action to be taken.

These changes were then inserted into the program, most of which were within the main repeat loop, along with a counter, to indicate how many transmitted photons entered the big cell at some point during their path.

4.6 Adding code to represent actual detection set-up

The final change to the program that was required was to alter the code to take the actual optical detection set-up into account, and not simply determine what would be observed at the exit window of the cell. After exiting the window, the photons hit a mirror, some distance from the cell window (see figure 4.5 on the next page), where the light is reflected to a detector.

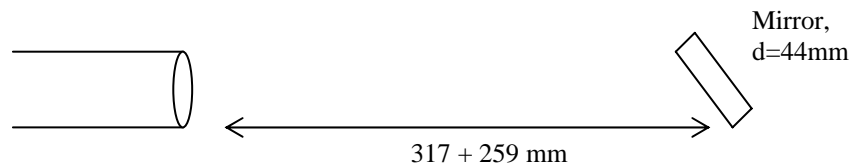


Figure 4.5: The Optical Detector set-up

The program needs to take this into consideration, so that only those photons with an exit angle small enough to hit the mirror will be detected and so counted as transmitted.

The best (and simplest) way of doing this was to extend the vector of motion of each photon, and see if its projection would hit the mirror or not. The main changes made were: -

- Variables were introduced which stored the lengths `Mirror_Radius`, `Cell_to_detector_box` and `Detector_Box_to_mirror`.
- The code was changed for the case where the z co-ordinate of position is greater than the length of the small cell so that it checks the photon is within the mirror radius when it reaches it (length of cell + `Cell_to_detector_box` + `Detector_Box_to_mirror`), as well as checking it is within the cell radius when it reaches the exit window.

This completed the changes that needed to be made to the main code so that the new program code could then be used to obtain results.

5 Results and Analysis

5.1 Writing the code to analyse data

The first step in analysis, both for the simple data (single cylinder with absorption included) and for the complex data (using the finished program) was to integrate the data file containing information about the variation of asymmetry parameter and single scatter albedo with Wavenumber with the data files obtained.

Initially, the data was read in containing the variation of G and Ω with wavenumber, and stored in an array. Then the files containing the multiple scattering deviation percentages for different G values were read in, and likewise were stored in an array. These values were then rearranged into a three-dimensional matrix, with all the percentage deviation values for each of the 3 variables - G , Ω and optical depth. This is represented graphically below, in figure 5.1, where each axis represents one of the variables, and the data is stored in the element corresponding to the G , Ω (single scatter albedo) and t (optical depth) values that it represents.

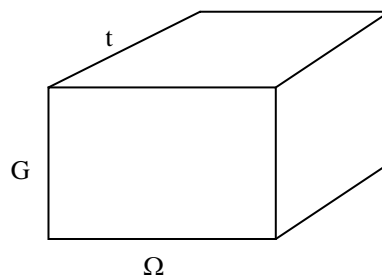


Figure 5.1: A graphical representation of the matrix used to store the results

So, for a specified G and Ω (and so wavenumber), we have a line of values in the array, giving the variation of percentage deviation due to multiple scattering for each optical depth value. From this, the program needs to find the value of optical depth where there is a 1% enhancement in detection due to multiple scattering. Using the variation of G and Ω with wavenumber, it was then possible to see how the optical depth at which there is a 1% enhancement varies with wavenumber (the aim of the project). This results program is included in Appendix 7.4. The IDL Interpol command is used to determine the optical depth at which there is a 1% enhancement for each wavenumber in the spectrum (G , Ω vs. wavenumber) file. Lines were then added to create the plots that were required.

5.2 Analysis of Simple Cell data

The contour plots obtained using the data from the simple model (with no code to take into account the optical acceptance due to the mirror or the large cell's existence) were good. They had clear contour lines, which became smoother as the asymmetry parameter increased. Two examples of the plots (including error plots) are shown below, for $G=0.3$ and $G=0.9$.

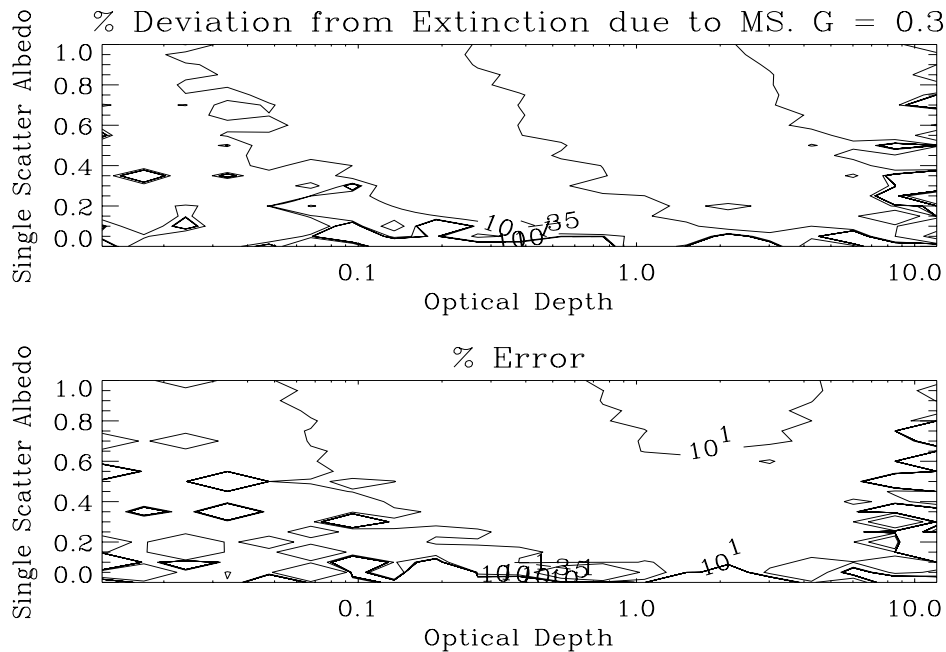


Figure 5.2: Contour plot showing lines of constant percentage enhancement due to multiple scattering with varying Optical Depth and Single Scatter Albedo for Asymmetry Parameter = 0.3

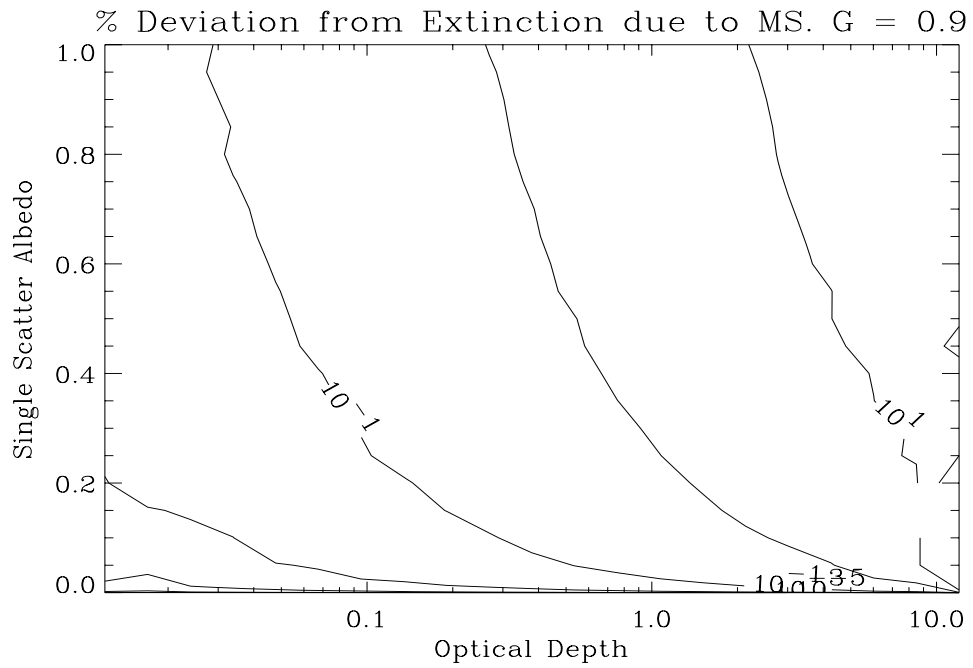


Figure 5.3: Contour plot showing lines of constant percentage enhancement due to multiple scattering with varying Optical Depth and Single Scatter Albedo for Asymmetry Parameter = 0.9

The final thing to do with my simple data was to run the results program in order to analyse (as described in section 5.1 above) the final result for the simple model. The final results graph is shown below, along with graphs showing the variation of asymmetry parameter and Omega with wavelength: -

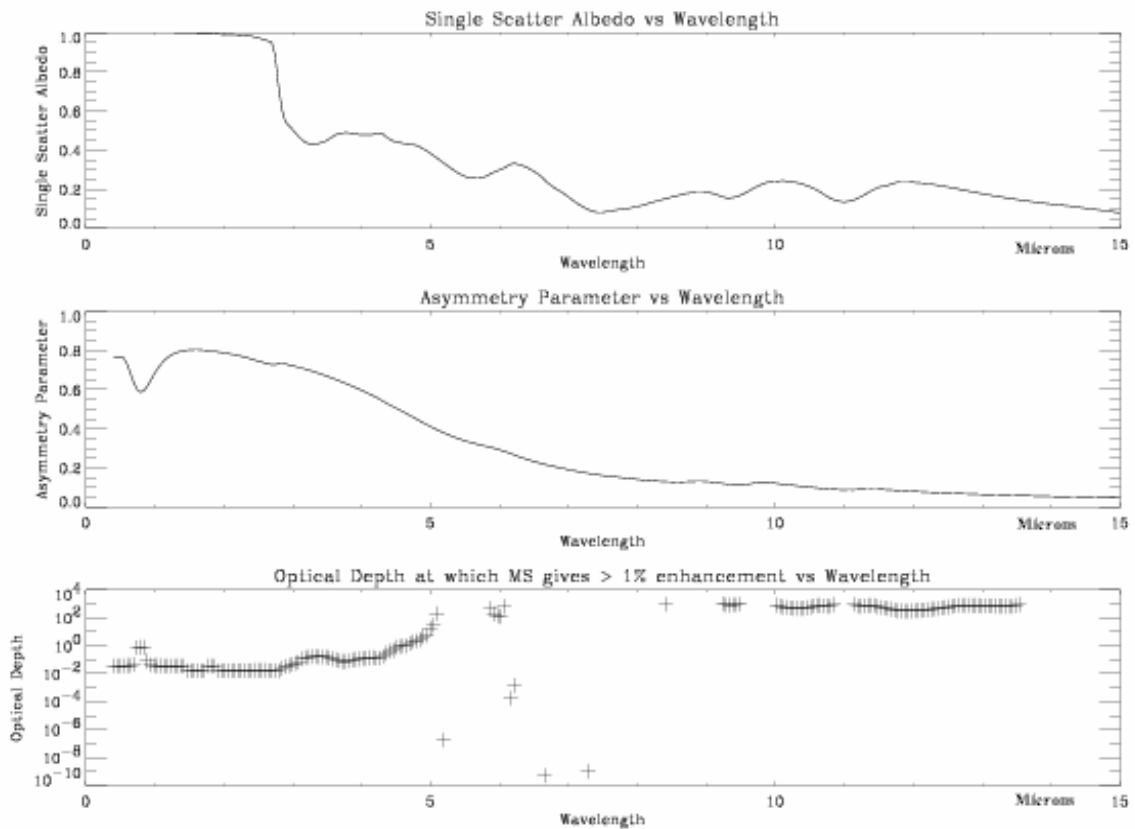


Figure 5.4: The final results graph for the simple (single cell) model, showing the Optical Depth at which multiple scattering contributes > 1% of photons vs wavelength.

This could also be plotted vs. wavenumber.

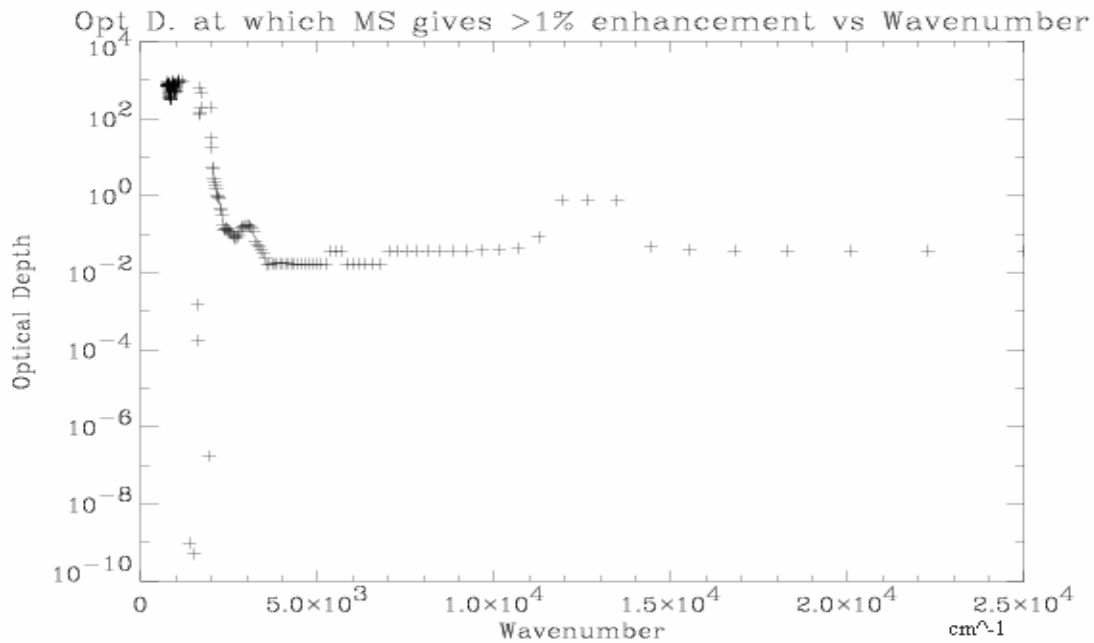


Figure 5.5: The final results graph for the simple (single cell) model, showing the Optical Depth at which multiple scattering contributes > 1% of photons vs wavenumber.

The optical depth at which multiple scattering becomes significant (a 1% enhancement to total transmission count) varies with wavelength such that at small wavelengths it is pretty much constant. It rises up steeply before the graph faults (due to statistical noise in the data) between wavelengths 5-9, before becoming a clear line again above this. As will be discussed later with reference to the “complex” cell data, the problems that result in the first attempt at the complex data results being abandoned also cause this simple model result to be not particularly useful. Therefore further analysis of this result will not be considered, and the remainder of the project analysis will focus on that of the results formed using the complete (complex) cell model.

5.3 Analysis of Complex Cell data

When attempting to run the results program for the new complex cell data, the results were not at all as expected. The plots were just a noisy mess, and when looking at the contour plots for individual G values, these were also seen to be clearly quite a mess. An example follows for $G = 0.5$.

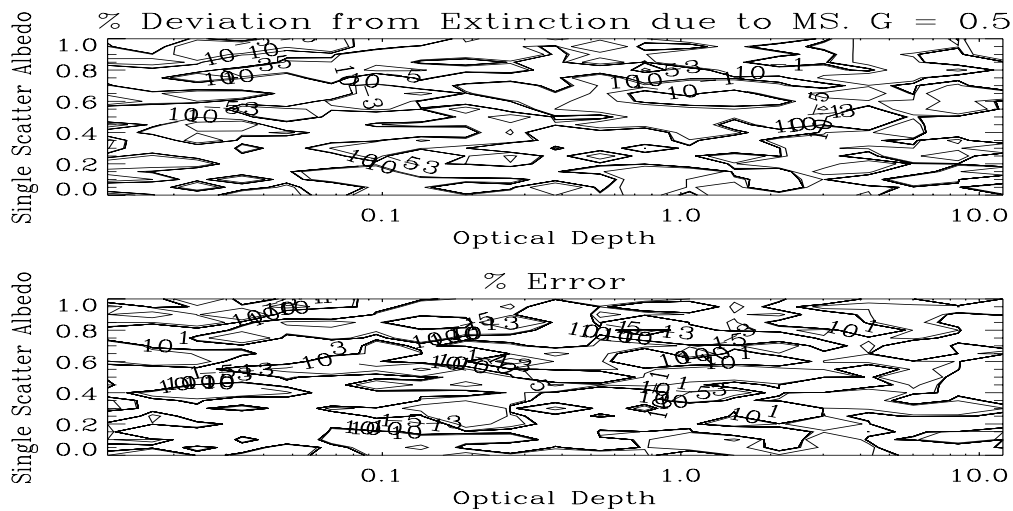


Figure 5.6: Contour plot for the complex cell model, attempting to showing lines of constant percentage enhancement due to multiple scattering with varying Optical Depth and Single Scatter Albedo for Asymmetry Parameter = 0.5

Following discussion, it was decided that statistical noise was causing the plot to be so messy, and that the only solution to being able to obtain a result would be to re-run the entire simulation with different output data, such that it would not be able to have a negative percentage enhancement or silly values. Instead of what was previously saved to file, the output was changed to save the number of photons transmitted, the number which were multiply scattered, and the number which were directly transmitted. This would still enable the percentage enhancement to be calculated, but in a simpler way which should eradicate the noise which was ruining the plots. The final programs in the appendices represent this method of handling the data.

After running the program again, with this new output data, it was possible to attempt to create contour plots again, and to generate the graph that was to be the final result. The contour plots were significantly better this time. Below are two examples, for $G = 0.1$ and 0.9 .

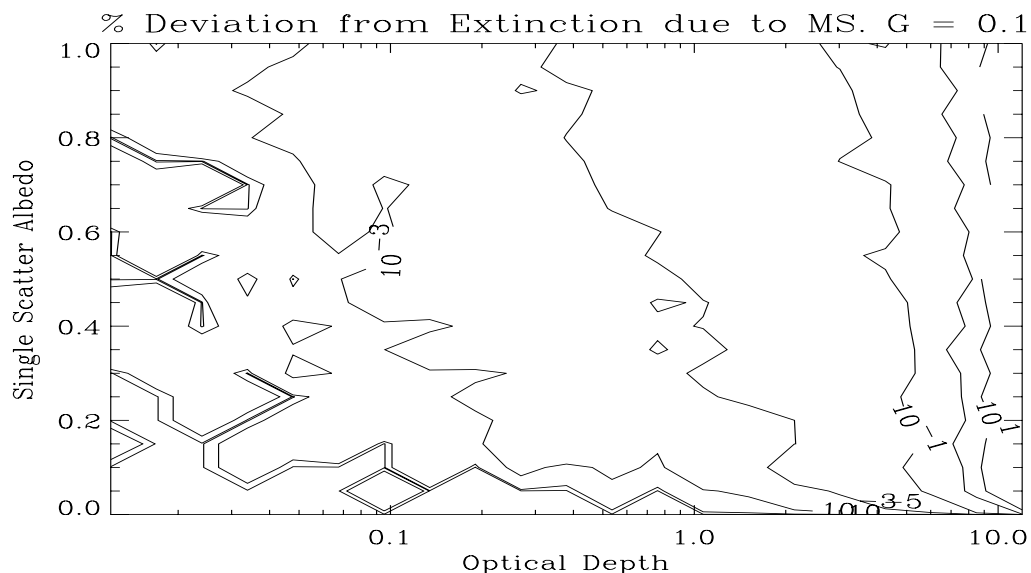


Figure 5.7: Contour plot showing lines of constant percentage enhancement due to multiple scattering with varying Optical Depth and Single Scatter Albedo for Asymmetry Parameter = 0.1

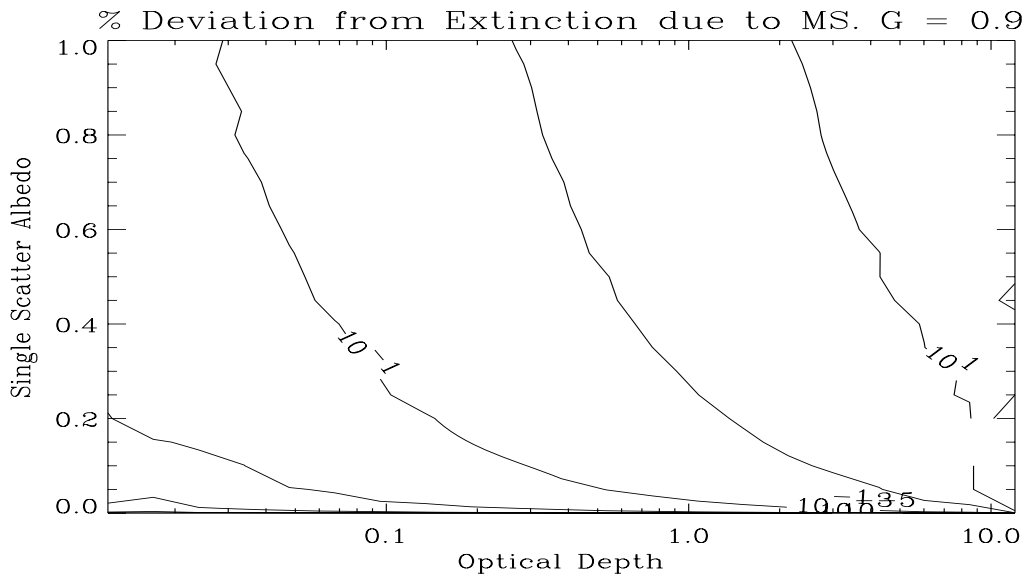


Figure 5.8: Contour plot showing lines of constant percentage enhancement due to multiple scattering with varying Optical Depth and Single Scatter Albedo for Asymmetry Parameter = 0.9

Clearly these plots are much better than those produced previously. Even for the plot with the asymmetry parameter equal to 0.1 we have fairly clear contours. This indicated that there was a good chance of obtaining a good final results graph this time, and on running the program (after changing the relevant code to reflect the new data stored in the files), the following graph was obtained: -

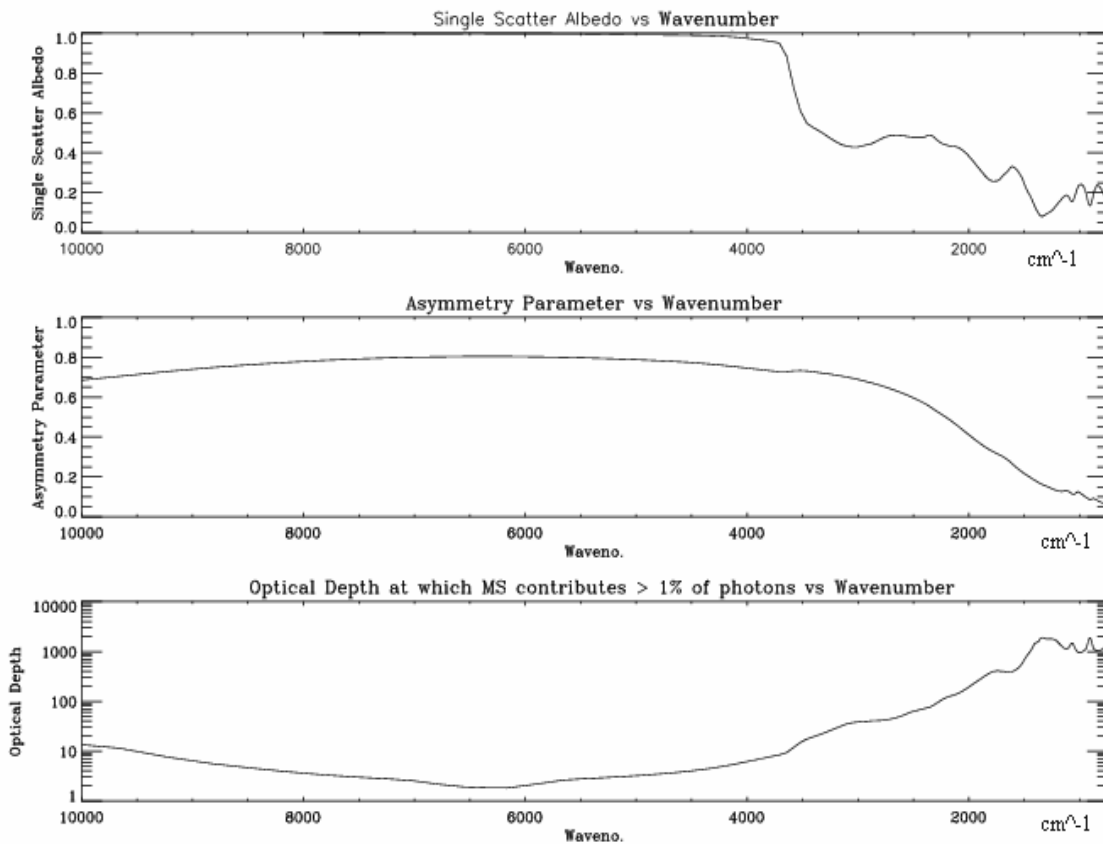


Figure 5.9: Final result graph for the complex cell model, showing how the Optical Depth at which multiple scattering contributes >1% of photons varies with Wavenumber, and the variation of Single Scatter Albedo and Asymmetry Parameter with Wavenumber.

This final graph for the new cell set-up is very pleasing. It is a smooth curve, with the optical depth at which multiple scattering becomes significant decreasing initially with increasing wavenumber, then reaching a minimum before gradually rising again. Below are different versions of this final data, vs. wavelength, and vs. wavenumber with the axis the other way round.

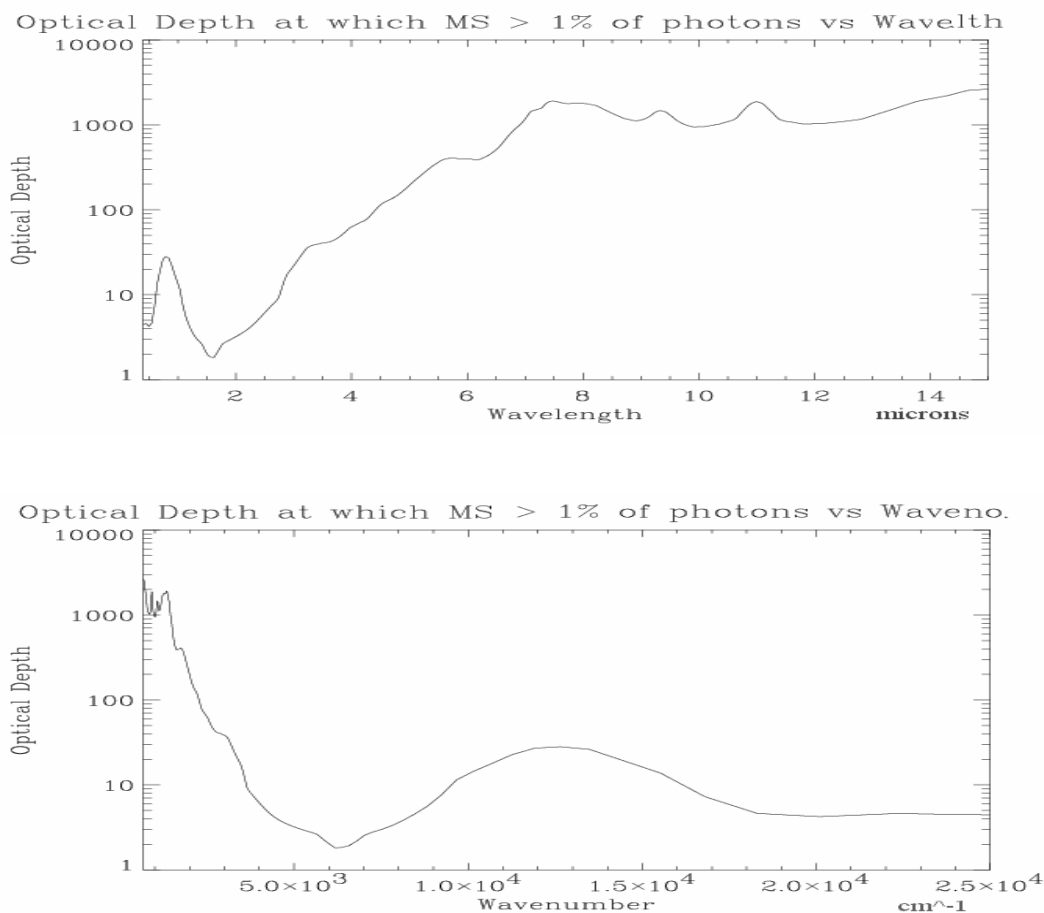


Figure 5.10: Different representations of the final results graph, with the axis reversed, showing variation with both wavelength and wavenumber.

When creating this final plot, the IDL program looks at each value of wavenumber in the spectrum file (of asymmetry parameter and single scatter albedo vs. wavenumber) and sees at what optical depth for that wavenumber that the contribution due to multiple scattering is 1%. Temporarily lines were then added to the final program in order to print out two examples of the method of calculating this, which are shown below. In the first example, the optical depth at which the percentage enhancement is 1% is outside the range of data points found by the simulation code, so the computer extrapolates the line to see where it would occur for this wavenumber. In the second example, the optical depth at which multiple scattering becomes significant is seen to be about 10, and little extrapolation is needed, as the data generated goes up to this value.

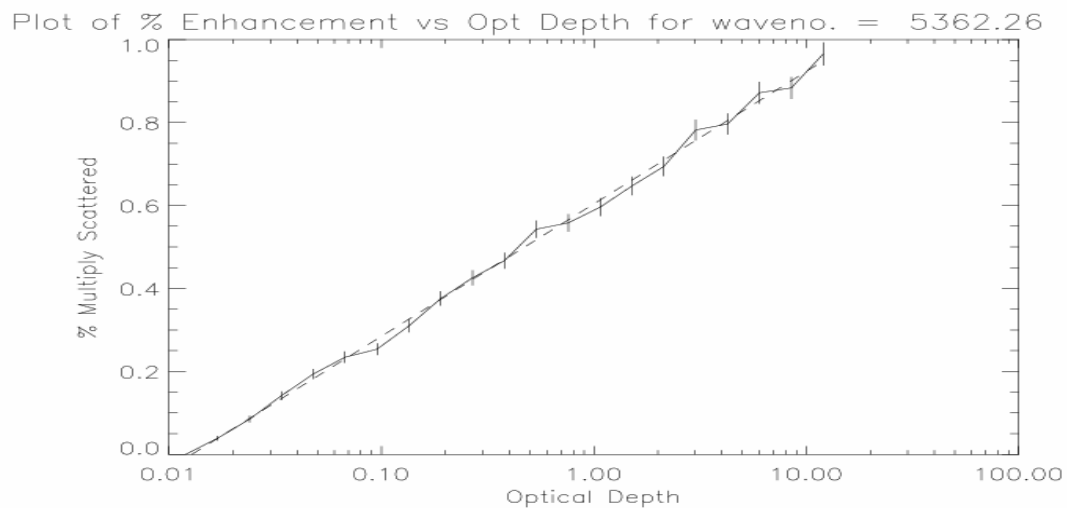
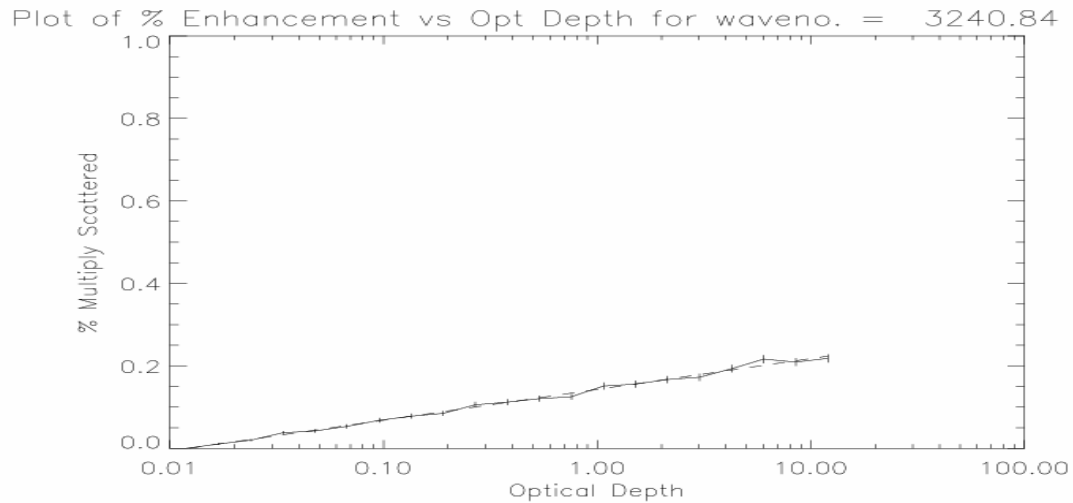


Figure 5.11: Graphs showing the method for calculating the final results graph, by analysing the Optical Depth at which multiple scattering contributes >1% for each wavenumber value.

The final part of the analysis was to compare the final result graph that had been plotted with actual measured data for where multiple scattering gives 1% enhancement. To do this, a small program was written to plot them both on the same axis. The plot is shown below, in figure 5.12, with the smooth line (which starts much higher) being the predicted line of 1% enhancement (my graph) and the other line being that of the measured data. This clearly shows that at small wavenumbers, the measured optical depth is very much less (it is a log scale) than the optical depth at which it has been found that multiple scattering is significant. However, at a wavenumber of approximately 5500, the measured optical depth becomes quickly comparable in magnitude to that at which multiple scattering is very significant, and indeed the measured value rises above the 1% predicted line from 6000-7000 wavenumbers. After this the measured optical depth is fairly erratic, for most wavenumbers, it is below the 1% multiple scattering contribution line.

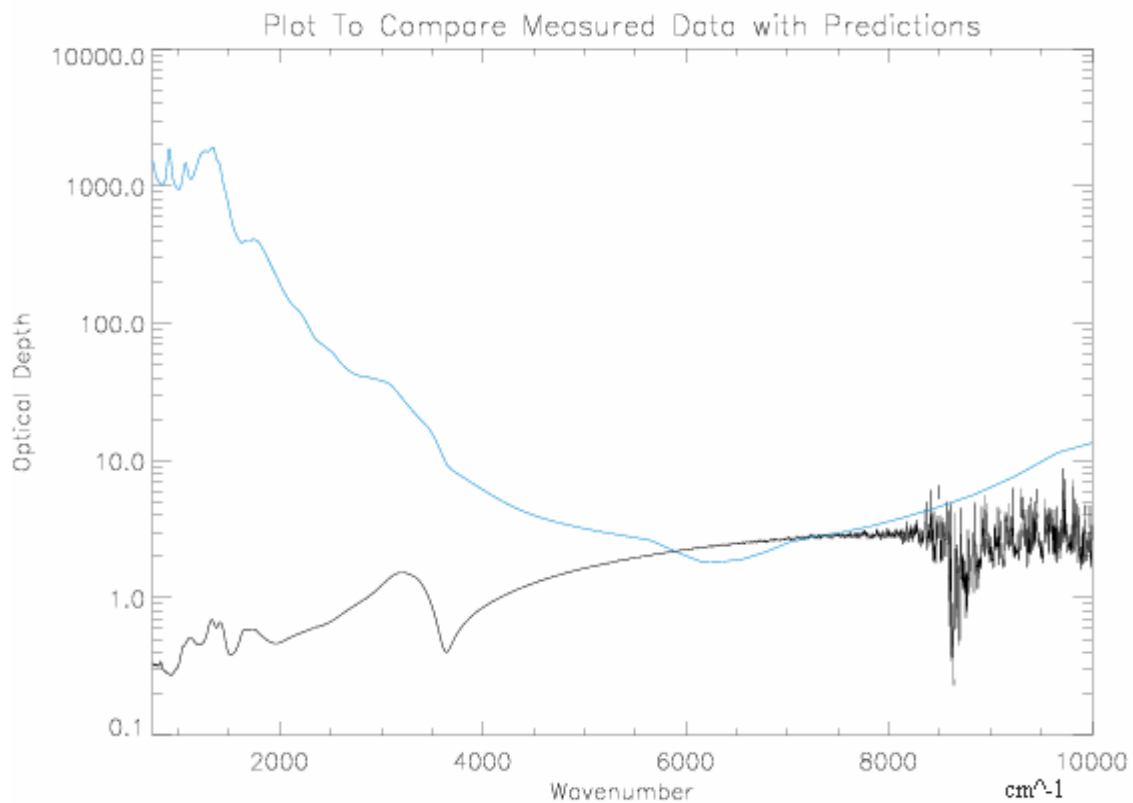


Figure 5.12: Graph showing the comparison of measured variation of Optical Depth with Wavenumber, and the predicted Optical Depth at which multiple scattering is significant vs. Wavenumber. The lighter line (which begins much higher) is that showing the predicted variation.

6 Conclusion

The significance of the contribution of multiple scattering to the measurements obtained at the Rutherford Appleton Laboratory aerosol cell has been predicted in this computer model. A previously existing computer simulation program has been modified to take into account a number of additional factors not previously included in the more basic model, and has been adapted to give these final results. The fact that the aerosol can absorb photons in the beam was added, and the code was altered to take into account the new more complex dimensions of the new cell set-up, including the actual detector arrangement. Programs were then written to plot contours, and finally to vary the optical depth and asymmetry parameter, in order to create the final results graphs. After quite a large number of problems in obtaining it, a good final graph has been created to show the variation of the optical depth at which multiple scattering becomes significant against wavenumber, so as to achieve the main aim of the project. When compared to measured variation, this result shows that for most wavenumbers, multiple scattering will not effect measurements in the Rutherford Appleton Laboratory aerosol cell. Only for wavenumbers above 5600 does the multiple scattering contribution become significant, and even then, it only does so for a small number of wavenumbers, before falling back below the predicted line. Further work could involve the adaptation of the code further, to include the possibility of reflection by the inner walls of the cell, and not to simply assume photons are absorbed and lost. Other extensions to the project could involve the sending of a greater number of photons, which would increase the simulation time, but reduce error, and investigating other percentages of multiple scattering enhancement.

7 Appendices

7.1 The Original Program, beam2.pro

7.1.1

```
; Version 2 of this program only takes note of photons that are transmitted
window,0
Cell_Radius = 0.0165 ; m
Cell_Radius2 = Cell_Radius*Cell_Radius ; m^2
Beam_Radius = 0.006 ; 0.014* .125/.263 ; m
Beam_radius2=Beam_Radius*Beam_Radius
Length = 0.263 ; m
Optical_Depth = 2.
Extinction = Optical_Depth/Length ; m^-1
G = 0.70 ; or 0.5
Tg = 2*g
Opg = 1 + g
Opgg = 1 + g * g
Omgg = 1 - g * g
Tpi = 2 * !pi
Photons= 355 ; 355 = 10^6 total
; Colour_ps,10,/back
```

7.1.2

```
!p.multi=0
!p.region=squareregion(region)

dX = 0.0002 ; m
dY = 0.0002 ; m

BXN = 2*Beam_Radius/dX + 1
BXV = -Beam_Radius + Indgen(BXN) * dX
BYN = 2*Beam_Radius/dY + 1
BYV = -Beam_Radius + Indgen(BYN) * dY
; If (CrossHair) Then BXV(where(BXV Eq 0)) = 2 * CellRadius ; don't start any photons on the yaxis if
cross hair is set

plot,[-cell_radius,cell_radius], [-cell_radius,cell_radius], $
xtitle='!6X (mm)',ytitle='!6Y (mm)', Title = '!6Exit Beam', /nodata
Circle = !Pi * findgen(361) / 180
oplot,cell_radius*sin(Circle),cell_radius*cos(Circle)
oplot,beam_radius*sin(Circle),beam_radius*cos(Circle),linestyle=2

T = !Pi * Findgen(21) / 10
usersym,2*sin(t),2*cos(t),/fill
for i = 0,7 do oplot,[-Cell_radius + 0.002*i],[0.018], psym=8 ; ,color=i+1
for i = 0,7 do xyouts,[-Cell_radius + 0.002*i],[0.019] ,align=0.5,string(i,format='(I1)')
Empty
```

7.1.3

```
; Photon values
Radius2 = BXV^2 # Replicate(1.0,BYN) + Replicate(1.0,BXN) # BYV^2
Q = where (Radius2 Le Beam_Radius2, Count)
Photons_To_Send = Count * Photons
print,' Photons To Send:', Photons_To_Send
```

Photons_to_Count = Photons_to_Send * Exp(-Optical_Depth)*101 ; can handle a diffuse to direct ratio up to 100

P_NS = LonArr(Photons_To_Count)

P_Ex = FltArr(3,Photons_To_Count)

P_Th = FltArr(Photons_To_Count)

Reflected_Photons = 0l

Lost_Photons = 0l

X = fltarr(3) ; position of the photon in cartesian coordinates (reference)

Y = fltarr(3) ; scatter vector in cartesian coordinates (primed)

T = fltarr(3) ; scatter vector in spherical coordinates (primed)

Z = fltarr(3) ; scatter vector in cartesian coordinates (reference)

U = fltarr(3) ; scatter vector in sperical coordinates (reference)

Count = 0L

7.1.4

For Photon = 1, Photons do begin

For I = 0, BXN - 1 Do Begin

For J = 0, BYN - 1 do begin

X(0) = BXV(I) ; Initial X location of the photon

X(1) = BYV(J) ; Initial Y location of the photon

If ((X(0)^2 + X(1)^2) Lt Beam_Radius2) Then Begin

X(2) = 0 ; Initial Z location of the photon

OutSide = 0

Scatter = 0

T(0) = -Alog(1-Randomu(Seed))/Extinction & T(1) = 0 & T(2) = 0 ; Initial direction and distance

before 1st scatter

Z = S_to_C(T) ; delta position in reference system

U = T ; direction vector in reference system

Repeat Begin

V = X ; save last position for output angle calculation

X = X + Z ; New position in (reference) cartesian coordinates

If ((X(2) Gt 0) And (X(2) Lt Length) And \$

((X(0)^2 + X(1)^2) Lt Cell_Radius2)) Then Begin

;

New scatter

Scatter = Scatter + 1

;

Calculate new location in primed coordinates (ie incident travelling direction with scatter at origin)

T = [-Alog(1-Randomu(Seed)) / Extinction , \$

Acos((Opgg - (Omgg / (Opg-Tg*Randomu(Seed)))^2)/Tg) , \$

RandomU(Seed) * Tpi]

Y = S_to_C(T) ; new location in cartesian coordinates (still primed)

;

Rotate the primed coordinates back to the reference system using the direction vector u

SinT = sin(u(1)) & CosT = Cos(u(1))

SinP = sin(u(2)) & CosP = Cos(u(2))

M = [[CosT*CosP, CosT*SinP,-SinT] , \$

[-SinP, CosP, 0] , \$

[SinT*CosP, SinT*SinP, CosT]]

Z = M # Y ; delta position in reference system

U = C_to_S(Z) ; direction vector in reference system

EndIf Else Begin

Outside = 1

P_NS(Count) = Scatter

```

Side = 0 ; assume exit not out the side of the cylinder
; Determine exit point
Case 1 of
(X(2) Le 0): Begin
M = (Z(2) - X(2)) / z(2) ; Distance along final vector that the Entry wall is crossed
Exit_Point = X - Z + M * Z
If (Exit_Point(0)^2+Exit_Point(1)^2 Gt Cell_Radius2) Then $ ; went out side not Entry wall
Side = 1 $
Else $
Reflected_Photons = Reflected_Photons + 1
End
(X(2) Ge Length): Begin
M = (Length + Z(2) - X(2)) / z(2) ; Distance along final vector that the exit wall is crossed
Exit_Point = X - Z + M * Z
Exit_Point(2) = Length ; Force to be identically length
If (Exit_Point(0)^2+Exit_Point(1)^2 Gt Cell_Radius2) Then $ ; went out side not exit wall
Side = 1 $
Else Begin
P_Ex(*,Count) = Exit_Point
P_Th(Count) = Total([0.0,0.0, 1.0] * Z)/Sqrt(Total(Z^2)) ; Use dot product to determine
cos(theta)
oplot,[Exit_Point(0)],[Exit_Point(1)],psym=3 ;,color=scatter+1 ; As we go plot
Count = Count + 1
EndElse
End
Else: Side = 1 ; went out the side of the cylinder
EndCase
If (Side) Then Lost_Photons = Lost_Photons + 1
If ((Count+Reflected_Photons+Lost_Photons) Mod 1000 Eq 0) Then begin
Sent = 'Photons Sent:'+string(Count+Reflected_Photons+Lost_Photons,Format='(I9)')
iF(N_Elements>Last Ne 0) Then $
xyouts,.0,.018>Last,color=0,align=0
xyouts,.0,.018,Sent,color=9,align=0
Last = Sent
Empty
EndIf
EndElse

EndRep Until (Outside)
EndIf
EndFor
EndFor
EndFor

```

7.1.5

```

P_Ex = P_Ex(*,0:Count)
P_NS = P_NS(0:Count)
P_Th = P_Th(0:Count)

```

```

Filename =
'Beam2_T'+string(Optical_Depth*100,Format='(I4.4)')+ '_G'+string(G*100,Format='(I4.4)')+'.IDL'
Save,Filename=Filename

```

```

Sum = Count+Lost_Photons+Reflected_Photons
print,Reflected_Photons, 100.*Reflected_Photons/Float(Sum), Format = "(' Photons Reflected:', I8,'
(,F4.1,' %))'"
print, Lost_Photons, 100.*Lost_Photons /Float(Sum), Format = "(' Photons out wall:', I8,' (,F4.1,'
%))'"

```

```

print, Count, 100.*Count /Float(Sum), Format = "('Photons Transmitted:', I8, ' (',F4.1,' %)'"
If (Count+Lost_Photons+Reflected_Photons Ne Photons_To_Send) Then Stop,'We have a problem: Not
all the boys came home'

```

7.1.6

```

; Plots
!p.region=0
!p.multi=0
Max_Scatters = Max(P_NS)
print,'Maximum number of scatters:',Max_Scatters
Colour_ps,Max_Scatters+3,/back
ODS = string(Optical_Depth,Format='(F5.2)')

; Transmitted Photons
plot,[-cell_radius,cell_radius], [-cell_radius,cell_radius], $
  xtitle='X (mm)',ytitle='Y (mm)', Title = 'Transmitted Photon Exit Points Tau =' +ODS, /nodata
oplot,cell_radius*sin(Circle),cell_radius*cos(Circle)
oplot,beam_radius*sin(Circle),beam_radius*cos(Circle),linestyle=2
for i = 0,Max_Scatters do oplot,[Cell_radius*(-1. + 0.12*i)],[1.10*Cell_Radius], psym=8 ,color=i+1
for i = 0,Max_Scatters do xyouts,[Cell_radius*(-1. + 0.12*i)],[1.15*Cell_Radius]
,align=0.5,string(i,format='(I1)')
plots,P_Ex(0,*),P_Ex(1,*),psym=3,color=P_NS+1

Temp = where(P_NS Eq 0, DCount)
print,Sum*Exp(-Optical_Depth),100*Exp(-Optical_Depth), $
  Format = "('Expectation Number of direct photons:', I8, ' (',F5.2,'%)"
print,DCount, 100.*DCount/Sum, $
  Format = "(' Actual Number of direct photons:', I8, ' (',F5.2,'%)"

!P.multi=[0,2,2]
If (!d.name Eq 'PS') then device,/landscape,Filename =
'Beam2_T'+string(Optical_Depth*100,Format='(I4.4)')+'_G'+string(G*100,Format='(I4.4)')+'.PS'

X = Indgen(Max_Scatters+1)
Y = histogram(P_NS,Min= 0,Max=Max_Scatters,Binsize=1)/Float(Count)
plot,x,y,psym=10,xtitle='Number of scatters',ytitle='Normalized Frequency',title='All Transmitted
Photons Tau =' +ODS

Rn = 165
dR = Cell_Radius/Rn
X = Findgen(Rn)* dR
Y = Fltarr(RN,Max_Scatters+1)

Acceptance_Angle = [0, Cos(1!/radeg),Cos(.1!/radeg) ] ; Optics acceptance

For J = 0,2 do begin
  QS = where(P_Th Ge Acceptance_Angle(j), C2S) &$
  IR = Long(Sqrt(P_Ex(0,QS)^2+P_Ex(1,QS)^2)/dR) &$
  IS = P_NS(QS) &$
  Y(*,*) = 0 &$
  for i = 0I, C2S-1 do Y(IR(I),IS(I)) = Y(IR(I),IS(I)) + 1 &$
  A = Total (Y(*,0)) &$
  B = Total (Y(*,1:*)) &$
  If (B Ne 0) Then SN = A / B &$
  If (B Ne 0) Then SNE = SN * Sqrt(A/A^2 + B/B^2) &$
  plot_io,X # Replicate(1,Max_Scatters+1),y>1, $
  xtitle='Cell Radius (mm)',ytitle='Photon Count', $

```

```

    Title = 'Transmitted Photons within '+string(Acos(Acceptance_Angle(j))*!radeg,Format='(F5.2)')+ '
degrees!CTau =' + ODS, /nodata  &$
    If (B Eq 0) Then SNS = 'Signal to noise: !C >' + string(A,Format='(F9.2)') Else SNS = 'Signal to
noise: !C '+string(SN,SNE,Format="(F9.2,'!9+!6',F9.2)") &$
    xyouts,beam_radius,Max(y)/2,SNS                                     &$
    for i = 0!,Max_Scatters do oplot,x,y(*,i) ,color=i+1                &$
EndFor
If (!d.name Eq 'PS') then device,/close

End

```

7.2 The Final Simulation program, beam10.pro

; Beam10 (Describes new cell setup, includes Absorption code for Photons and Finds how much deviation multiple scattering causes to the transmission value, as Optical depth and Omega are varied for specified G. Includes code for Limited Angle Acceptance. Also Plots correctly (As seen by mirror not Cell).)

```

window,0
Cell_Radius = 0.0175 ; m
Cell_Radius2 = Cell_Radius*Cell_Radius ; m^2
Beam_Radius = 0.006 ; 0.014* .125/.263 ; m
Beam_radius2=Beam_Radius*Beam_Radius
Length = 0.434 ; m
BigCell_Radius = 0.151 ;m
BigCell_Radius2 = BigCell_Radius*BigCell_Radius ;m^2
; Optical_Depth = 2.
S_out = 0.055 ; m
alpha = 0.214 - (0.5 * 0.025)
beta = 0.815
Big_Length = Alpha + 2*Cell_Radius + Beta
Cell_To_Detector_Box = 0.259
Detector_Box_To_Mirror = 0.317
Mirror_Radius = 0.022
Mirror_Radius2 = Mirror_Radius*Mirror_Radius

LUN = 1

For CntG = 1,9 Do Begin
    G = 0.1 * CntG

Print, 'G=',G

; G = 0.70 ; or 0.5
Tg = 2*g
Opg = 1 + g
Oppg = 1 + g * g
Omgg = 1 - g * g
Tpi = 2 * !pi

Photons=120 ; 355 = 10^6 total

Filenam = 'Compnew_G_'+string(CntG,Format='(I4.4)')+'.dat'
Openw, LUN, Filenam

For CntOmega = 0, 20 Do Begin
    For CntOpt = 0, 20 Do Begin

        Omega = 0.05*CntOmega

```

```

Optical_Depth = 0.12*10^(-1 + 3*CntOpt/20.)
Extinction = Optical_Depth/Length ; m^-1
Colour_ps,10,/back

!p.multi=0
!p.region=squareregion(region)
dX = 0.0002 ; m
dY = 0.0002 ; m

BXN = 2*Beam_Radius/dX + 1
BXV = -Beam_Radius + Indgen(BXN) * dX
BYN = 2*Beam_Radius/dY + 1
BYV = -Beam_Radius + Indgen(BYN) * dY
; If (CrossHair) Then BXV(where(BXV Eq 0)) = 2 * CellRadius ; don't start any photons on the yaxis if
cross hair is set

plot,[-mirror_radius,mirror_radius], [-mirror_radius,mirror_radius], $
  xtitle='!6X (mm)',ytitle='!6Y (mm)', Title = '!6Mirror Capture', /nodata

; plot,[-cell_radius,cell_radius], [-cell_radius,cell_radius], $
;  xtitle='!6X (mm)',ytitle='!6Y (mm)', Title = '!6Exit Beam', /nodata
Circle = !Pi * findgen(361) / 180
oplot,mirror_radius*sin(Circle),mirror_radius*cos(Circle)
oplot,cell_radius*sin(Circle),cell_radius*cos(Circle),linestyle=2
oplot,beam_radius*sin(Circle),beam_radius*cos(Circle),linestyle=2

T = !Pi * Findgen(21) / 10
usersym,2*sin(t),2*cos(t),/fill
for i = 0,7 do oplot,[-Mirror_radius + Mirror_radius*0.002/.0165 *i],[Mirror_radius*0.018/.0165],
psym=8 ,color=i+1
for i = 0,7 do xyouts,[-Mirror_radius + Mirror_radius*0.002/.0165 *i],[Mirror_radius*0.019/.0165]
,align=0.5,string(i,format='(I1)')
Empty

; Photon values
Radius2 = BXV^2 # Replicate(1.0,BYN) + Replicate(1.0,BXN) # BYV^2
Q = where (Radius2 Le Beam_Radius2, Count)
Photons_To_Send = Count * Photons
print,' Photons To Send:', Photons_To_Send
Photons_to_Count = Photons_to_Send ; * Exp(-Optical_Depth)*101 ; can handle a diffuse to direct ratio
up to 100

Photons_to_count = Photons_to_count ; Is this needed > 400000

P_NS = LonArr(Photons_To_Count)
P_Ex = FltArr(3,Photons_To_Count)
P_Th = FltArr(Photons_To_Count)

Reflected_Photons = 0l
Lost_Photons = 0l
Absorbed_Photons = 0l
Photons_in_big_C = 0l

X = fltarr(3) ; position of the photon in cartesian coordinates (reference)

Y = fltarr(3) ; scatter vector in cartesian coordinates (primed)
T = fltarr(3) ; scatter vector in spherical coordinates (primed)

Z = fltarr(3) ; scatter vector in cartesian coordinates (reference)
U = fltarr(3) ; scatter vector in sperical coordinates (reference)

```

K = fltarr(3) ; position of photon in frame of Big_cell

Count = 0L

For Photon = 1, Photons do begin

For I = 0, BXN - 1 Do Begin

For J = 0, BYN - 1 do begin

X(0) = BXV(I) ; Initial X location of the photon

X(1) = BYV(J) ; Initial Y location of the photon

If ((X(0)^2 + X(1)^2) Le Beam_Radius2) Then Begin

X(2) = 0 ; Initial Z location of the photon

In_Small_Cell = 1

In_Big_Cell = 0

OutSide = 0

Scatter = 0

Absorbed = 0

Big_C = 0

T(0) = -Alog(1-Randomu(Seed))/Extinction & T(1) = 0 & T(2) = 0 ; Initial direction and distance before 1st scatter

Z = S_to_C(T) ; delta position in reference system

U = T ; direction vector in reference system

Repeat Begin

V = X ; save last position for output angle calculation

Was_In_Small_Cell = In_Small_Cell

Was_In_Big_Cell = In_Big_Cell

X = X + Z ; New position in (reference) cartesian coordinates

K = S_To_B(x,Length,Cell_radius,Alpha) ; Find coordinates in Big Cell frame

In_Small_Cell = (X(2) Gt 0) And (X(2) Lt Length) And ((X(0)^2 + X(1)^2) Lt Cell_Radius2)

In_Big_Cell = (K(2) Gt 0) And (K(2) Lt Big_Length) And ((K(0)^2 + K(1)^2) Lt

BigCell_Radius2)

Escaped = (In_Big_Cell And Not In_Small_Cell And Was_In_Small_Cell And Not

Was_In_Big_Cell) Or \$

(In_Small_Cell And Not In_Big_Cell And Was_In_Big_Cell And Not Was_In_Small_Cell)

If (In_Small_Cell Or In_Big_Cell And Not Escaped) Then Begin

If Randomu(seed) Gt Omega then begin

Absorbed = 1

Absorbed_photons = Absorbed_photons + 1

EndIf Else Begin

; New scatter

Scatter = Scatter + 1

If in_Big_Cell And Not in_Small_cell Then Big_C = 1 Else Big_C = Big_C

; Calculate new location in primed coordinates (ie incident travelling direction with scatter at origin)

T = [-Alog(1-Randomu(Seed)) / Extinction , \$

Acoss((Opgg - (Omgg / (Opg-Tg*Randomu(Seed)))^2)/Tg) , \$

RandomU(Seed) * Tpi]

Y = S_to_C(T) ; new location in cartesian coordinates (still primed)

; Rotate the primed coordinates back to the reference system using the direction vector u

SinT = sin(u(1)) & CosT = Cos(u(1))

SinP = sin(u(2)) & CosP = Cos(u(2))

M = [[CosT*CosP, CosT*SinP,-SinT], \$

[-SinP, CosP, 0], \$

```

[ SinT*CosP, SinT*SinP, CosT ]

Z = M # Y ; delta position in reference system
U = C_to_S(Z) ; direction vector in reference system
EndElse

EndIf Else Begin
  Outside = 1
  P_NS(Count) = Scatter
  Side = 0 ; assume exit not out the side of the cylinder
; Determine exit point
  Case 1 of
    (X(2) Le 0): Begin
      M = (Z(2) - X(2)) / z(2) ; Distance along final vector that the Entry wall is crossed
      Exit_Point = X - Z + M * Z
      If (Exit_Point(0)^2+Exit_Point(1)^2 Gt Cell_Radius2) Or (Escaped) Then $ ; went out
side not Entry wall
        Side = 1 $
      Else $
        Reflected_Photons = Reflected_Photons + 1
      End
    (X(2) Ge Length): Begin
      S = (Length + Cell_To_Detector_Box + Detector_Box_To_Mirror + Z(2) - X(2)) / z(2) ;
Distance along final vector that the mirror plane is reached
      Mirror_Point = X - Z + S * Z
      Mirror_Point(2) = Length + Cell_To_Detector_Box + Detector_Box_To_Mirror
      If (Mirror_Point(0)^2+Mirror_Point(1)^2 Gt Mirror_Radius2) Or (Escaped) Then $ ; went
out side or missed mirror
        Side = 1 $
      Else Begin

        M = (Length + Z(2) - X(2)) / z(2) ; Distance along final vector that the exit wall is crossed
        Exit_Point = X - Z + M * Z
        Exit_Point(2) = Length ; Force to be identically length
        If (Exit_Point(0)^2+Exit_Point(1)^2 Gt Cell_Radius2) Or (Escaped) Then $ ; went out
side not exit wall
          Side = 1 $
        Else Begin
          P_Ex(*,Count) = Mirror_Point ; was Exit_Point
          P_Th(Count) = Total([0.0,0.0, 1.0] * Z)/Sqrt(Total(Z^2)) ; Use dot product to determine
cos(theta)
          oplot,[Mirror_Point(0)],[Mirror_Point(1)],psym=3 ,color=scatter+1
; oplot,[Exit_Point(0)],[Exit_Point(1)],psym=3 ,color=scatter+1 ; As we go plot
          Count = Count + 1
          If (Big_C) then (Photons_in_big_c) = (Photons_in_big_c) + 1
          EndElse
        EndElse
      End
    Else: Side = 1 ;went out the side

  EndCase

  If (Side) Then Lost_Photons = Lost_Photons + 1
  If ((Count+Reflected_Photons+Lost_Photons+Absorbed_Photons) Mod 1000 Eq 0) Then
begin
  Sent = 'Photons
Sent:'+string(Count+Reflected_Photons+Lost_Photons+Absorbed_Photons,Format='(I9)')
  iF(N_Elements(Last) Ne 0) Then $
  xyouts,0,Mirror_Radius*.018/.0165,Last,color=0,align=0
  xyouts,0,Mirror_Radius*.018/.0165,Sent,color=9,align=0

```

```

        Last = Sent
        Empty
    EndIf
EndElse

EndRep Until (Outside Or Absorbed)

EndIf
EndFor
EndFor
EndFor

P_Ex = P_Ex(*,0:Count)
P_NS = P_NS(0:Count)
P_Th = P_Th(0:Count)

; Filename =
'Beam2_T'+string(Optical_Depth*100,Format='(I4.4)')+ '_G'+string(G*100,Format='(I4.4)')+'.IDL'
; Save,Filename=Filename

Sum = Count+Lost_Photons+Reflected_Photons+Absorbed_Photons
print,Reflected_Photons, 100.*Reflected_Photons/Float(Sum), Format = "(' Photons Reflected:', I8,'
(',F4.1,' %))"
print, Lost_Photons, 100.*Lost_Photons /Float(Sum), Format = "(' Photons out wall:', I8,' (',F4.1,'
%))"
print, Count, 100.*Count /Float(Sum), Format = "('Photons Transmitted:', I8,' (',F4.1,' %))"
print, Absorbed_Photons, 100.*Absorbed_Photons /Float(Sum), Format = "(' Photons Absorbed:', I8,'
(',F4.1,' %))"
If (Count+Lost_Photons+Reflected_Photons+Absorbed_Photons Ne Photons_To_Send) Then Stop,'We
have a problem: Not all the boys came home'

; Plots
!p.region=0
!p.multi=0
Max_Scatters = Max(P_NS)
print,'Maximum number of scatters:',Max_Scatters
print, Photons_in_big_C, 100.*Photons_in_big_C /Float(Sum), Format = "('Transmitted Photons That
Entered Big Cell:', I8,' (',F4.1,' %))"
Colour_ps,Max_Scatters+3,/back
ODS = string(Optical_Depth,Format='(F5.2)')

; Transmitted Photons
plot,[-cell_radius,cell_radius], [-cell_radius,cell_radius], $
    xtitle='X (mm)',ytitle='Y (mm)', Title = 'Transmitted Photon Exit Points Tau =' +ODS, /nodata
oplot,cell_radius*sin(Circle),cell_radius*cos(Circle)
oplot,beam_radius*sin(Circle),beam_radius*cos(Circle),linestyle=2
for i = 0,Max_Scatters do oplot,[Cell_radius*(-1. + 0.12*i)],[1.10*Cell_Radius], psym=8 ,color=i+1
for i = 0,Max_Scatters do xyouts,[Cell_radius*(-1. + 0.12*i)],[1.15*Cell_Radius]
,align=0.5,string(i,format='(I1)')
plots,P_Ex(0,*),P_Ex(1,*),psym=3,color=P_NS+1

Temp = where(P_NS Eq 0, DCount)
print,Sum*Exp(-Optical_Depth),100*Exp(-Optical_Depth), $
    Format = "('Expectation Number of direct photons:', I8,' (',F5.2,'%))"
print,DCount, 100.*DCount/Sum, $
    Format = "(' Actual Number of direct photons:', I8,' (',F5.2,'%))"

; !P.multi=[0,2,2]

```

```

; If (!d.name Eq 'PS') then device,/landscape,Filename =
'Beam2_T'+string(Optical_Depth*100,Format='(I4.4)')+ '_G'+string(G*100,Format='(I4.4)')+'.PS'

; X = Indgen(Max_Scatters+1)
; Y = histogram(P_NS,Min= 0,Max=Max_Scatters,Binsize=1)/Float(Count)
; plot,x,y,psym=10,xtitle='Number of scatters',ytitle='Normalized Frequency',title='All Transmitted
Photons Tau ='+ODS

; Rn = 165
; dR = Cell_Radius/Rn
; X = Findgen(Rn)* dR
; Y = Fltarr(RN,Max_Scatters+1)

; Acceptance_Angle = [0, Cos(1/!radeg),Cos(.1/!radeg) ] ; Optics acceptance

; For J = 0,2 do begin                                &$
;   QS = where(P_Th Ge Acceptance_Angle(j), C2S)    &$
;   IR = Long(Sqrt(P_Ex(0,QS)^2+P_Ex(1,QS)^2)/dR)    &$
;   IS = P_NS(QS)                                    &$
;   Y(*,*) = 0                                       &$
;   for i = 0l, C2S-1 do Y(IR(I),IS(I)) = Y(IR(I),IS(I)) + 1    &$
;   A = Total (Y(*,0))                                &$
;   B = Total (Y(*,1:*))                              &$
;   If (B Ne 0) Then SN = A / B                        &$
;   If (B Ne 0) Then SNE = SN * Sqrt(A/A^2 + B/B^2)    &$
;   plot_io,X # Replicate(1,Max_Scatters+1),y>1, $
;   xtitle='Cell Radius (mm)',ytitle='Photon Count', $
;   Title ='Transmitted Photons within '+string(Acos(Acceptance_Angle(j))*!radeg,Format='(F5.2)')+ '
degrees!CTau ='+ODS, /nodata &$
;   If (B Eq 0) Then SNS = 'Signal to noise:!C >' + string(A,Format='(F9.2)') Else SNS = 'Signal to
noise:!C '+string(SN,SNE,Format="(F9.2,!9+!6',F9.2)") &$
;   xyouts,beam_radius,Max(y)/2,SNS                                     &$
;   for i = 0l,Max_Scatters do oplot,x,y(*,i),color=i+1                &$
; EndFor
; If (!d.name Eq 'PS') then device,/close

Trans_ext = Exp(-Optical_Depth)
Trans_ms = Float(Count)/(Photons_To_Send)
Trans_Dev = ((Trans_ms)-(Trans_ext))/Trans_ext
Trans_DevP= 100*Trans_Dev

; printf, LUN, Optical_Depth, Omega, Trans_ext, Trans_ms, Trans_DevP
; print, 'Opt D Omega Tran ext Tran ms Tran DevP'
; print, Optical_Depth, Omega, Trans_ext, Trans_ms, Trans_DevP
printf, LUN, Optical_Depth, Omega, Photons_to_Send, Count, Dcount

EndFor

EndFor

Close, LUN

EndFor

print, 'Program has finished successfully!!'

End

```

7.3 Program to create contour plots

; Program to plot Contours using data giving how the Transmission varies due to multiple scattering for different Single Scatter Albedo and Optical Depth values

; Read data in from file, then close the file

For TenG=1,9 Do Begin

; !P.Multi=[0,1,2]

If (!d.name eq 'PS') then device,file='ContComp'+string(TenG,Format='(I4.4)')+'.ps'

Data = FltArr(5,441)

LUN = 1

OpenR, LUN, 'Compnew_G_'+string(TenG,Format='(I4.4)')+'.dat'

ReadF, LUN, Data

Close, LUN

Yp=Indgen(21)

Xp=Indgen(21)

Y=0.12*10^(-1 + 3*Float(Yp)/20)

X=0.05*Xp

Tot = Data(3,*)

Dir = Data(4,*)

MS = Tot - (Dir-1)

PMS = 100*MS/Tot ;

AA = Reform (PMS,21,21)

levels=exp(-20 + (alog(max(aa))+20)*findgen(28)/27.)

levels=10^(-5+findgen(11))

Contour, AA, Y,X, levels=levels,/follow,c_ysize=1.2,xtitle='!6Optical Depth',ytitle='Single Scatter Albedo',ysize=1.2,/xlog,xstyle=1,ystyle=1,title= '% Deviation from Extinction due to MS. G = 0.'+string(TenG,Format='(I1.1)')

; Uncertainty= (((Data(3,*) * 314000)^(0.5))/314000)/Z/Data(2,*)*100*100

; BB = Reform (Uncertainty,21,21)

; Contour, BB, Y,X, levels=levels,/follow,c_ysize=1.2,xtitle='!6Optical Depth',ytitle='Single Scatter Albedo',ysize=1.2,/xlog,xstyle=1,ystyle=1,title= '% Error'

If (!d.name eq 'PS') then device,/close

EndFor

End

7.4 Program to create Complex Result

Loadxyz,'spectrum.dat',0,1,2,wave,ssa,G

!p.multi=0

Od=fltarr(N_elements(wave))

Data = FltArr(5,441)

LUN = 1

Enhance=wave

```

Omega =FltArr(21,21,9)
Tau =FltArr(21,21,9)
DirectC=FltArr(21,21,9)
Count = FltArr(21,21,9)
For TenG=1,9 Do Begin
  OpenR, LUN, 'Compnew_G_'+string(TenG,Format='(I4.4)')+'.dat'
  ReadF, LUN, Data
  Close, LUN
  Tau(*,*,TenG-1)=Data(0,*)
  Omega(*,*,TenG-1)=Data(1,*)
  Count(*,*,TenG-1)=Data(3,*)
  DirectC(*,*,TenG-1)=Data(4,*)
EndFor

MS = Count - (DirectC-1)
ES = sqrt(Count - (DirectC-1))

PMS = 100*MS/Count ;
PME = sqrt((ES/MS)^2 + (Sqrt(Count)/Count)^2)
PMP = PMS * PME

Xp=Indgen(21)
Yp=Indgen(21)
Zp=Indgen(9)

X=0.12*10^(-1 + 3*Float(Yp)/20)
Y=0.05*Xp
Z=(Zp+1)*.1

; Initialise optical depth vector
AA=X
BB=X
CC=X

For I=0,N_elements(wave)-1 Do Begin

  II = Interpol(Xp,X,SSA(I))
  KK = Interpol(Zp,Z,G(I))
  For J=0,N_elements(BB)-1 Do begin
    BB(J)=Interpolate(PMS,II,J,KK)
    CC(J)=Interpolate(PMP,II,J,KK)
  EndFor

  Result = POLY_FIT(alog(aa), bb, 1)
  od(i) = (1-Result(0))/Result(1)

plot_oi,aa,bb,yrange=[0,1],title=string(1e4/wave(i)),charsize=1.5
oplote, aa,bb,cc
plot,aa,Result(0)+Result(1)*alog(aa),linestyle=2

empty

EndFor

openw,1,'don.dat'
for i = 0,n_elements(wave)-1 do printf,1,wave(i),od(i)
close,1

!p.multi=[0,1,3]

```

```
If (!d.name eq 'PS') then device,file='Eg_per_cVSOpt.ps'
```

```
plot,10000/wave,ssa,xtitle='!6Waveno.',ytitle='!6Single Scatter Albedo',charsize=1.2,title='Single Scatter Albedo vs Wavenumber',xstyle=1,xrange=[10000,750]
```

```
plot,10000/wave,g,xtitle='!6Waveno.',ytitle='!6Asymmetry Parameter',charsize=1.2,title='Asymmetry Parameter vs Wavenumber',xstyle=1,xrange=[10000,750]
```

```
plot_io,10000/wave,od,xtitle='!6Waveno.',ytitle='!6Optical Depth',charsize=1.2,title='Optical Depth at which MS contributes > 1% of photons vs Wavenumber',xstyle=1,xrange=[10000,750]
```

```
If (!d.name eq 'PS') then device,/close
```

```
End
```

8 References

[1] Radiative Transfer in The Earth System, Charlie Zender

[2] IDL manual, Research Systems Ltd.

[3] The Physics of the Atmosphere, John T. Houghton