

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

**SIMULATION OF SMALL-ANGLE SCATTERING
PATTERNS VIA A MONTE-CARLO TECHNIQUE**

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

**Bryan Carl McAlister
Norman, Oklahoma
2000**

UMI Number: 9977954

UMI[®]

UMI Microform 9977954

Copyright 2000 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.


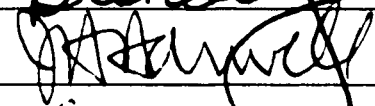
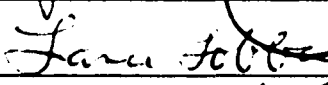

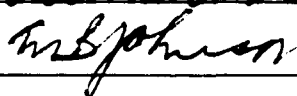
Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

**© Copyright by Bryan Carl McAlister 2000
All Rights Reserved**

SIMULATION OF SMALL-ANGLE SCATTERING PATTERNS VIA A MONTE-CARLO TECHNIQUE

A Dissertation APPROVED FOR THE
SCHOOL OF CHEMICAL ENGINEERING
AND MATERIAL SCIENCE

BY

Dedicated to

My family:

**Kenneth and Rosemary McAlister
Laurie Westlake and Denton McAlister**

My wife:

Shon Bower

ACKNOWLEDGEMENTS

I worked full time as a chemical engineer through most of my graduate school experience. This was in no way a trivial task because of the demands that job and school make on your time. I would not suggest this to anyone because of the various pitfalls that serving two masters brings. Because of this I feel three key figures deserve much credit; Dr. Brian Grady, Dr. Jeff Harwell and Howard Caplinger. I also need to make mention of the “coffee breaks” with my brother Denton.

I cannot say enough about Dr. Brian Grady. He has been an excellent choice for a graduate advisor. Brian has given me a lot of opportunity to grow as both a person and as an engineer. Under him I have been a teaching assistant, I have presented at many national conferences and I have taught a class. Brian’s approach with me has been more of a mentor and less as a boss. He has shown a lot of confidence in me and the other students that have worked for him. I need to point out that although I was his first graduate student, I was by no means his first to graduate. I sincerely appreciate his patience.

I want to say “thank you” to Dr. Jeff Harwell for allowing me to work full time while going to graduate school. As I mentioned earlier, I would not suggest this to anyone else, but I have survived and it has made me a much more complete person and has given me a very unique understanding about both industry and academia. Dr. Harwell showed his wisdom by having me to finish all course work before attempting any research. If I had tried to do things the normal route, course work and research simultaneously, I doubt I would have ever completed this degree.

Howard Caplinger was the Senior Chemical Engineer I worked with while at Dayton Tire. Not only has been very helpful in allowing my schooling to continue, he has also been very instrumental in my successes at Dayton Tire. Very few people are fortunate enough to work with someone who offers so much freedom in the workplace especially to a newly graduated engineer. I am very glad I was able to work with him and learn from his experience.

Some of my best memories here at the University of Oklahoma are the times Denton and I met for coffee. These “coffee breaks” normally entailed the discussion of a whole gambit of topics. It was a great experience. Not only did we discuss every day topics, but also we were able to discuss each other’s research topics and see them through each other’s eyes. I honestly believe that he and I could solve all the world’s problems over a cup of coffee and maybe the occasional large chocolate chip cookie.

TABLE OF CONTENTS

1. Fundamentals of Small-Angle Scattering and Monte-Carlo Simulation	1
1.1. Introduction	1
1.2. General Scattering Theory	4
1.3. Analysis of Scattering Data and Goals of this Research	8
1.4. General Monte-Carlo Theory	13
1.5. Experimental Constraints	17
1.6. References	18
 2. SAXS Simulation of Single Particle Systems	 20
2.1. Introduction	20
2.2. Theoretical Background	22
2.3. Simulation Technique and Technical Considerations	23
2.4. Comparisons	26
2.5. Objects and Orientations Without Analytical Solution	33
2.6. References	38
 3. SAXS Simulation of Particles with More Than One Electron Density	 39
3.1. Introduction	39
3.2. Theoretical Background and Description of Method	40
3.3. Results	42
3.4. References	54
 4. SAXS Simulation of Multiple Particle Systems	 55
4.1. Introduction	55
4.2. Dense Multiple Particle Systems	56
4.3. References	65
 5. Conclusions And Future Work	 67
5.1. Single Particle Systems	67
5.2. Multiple Particle Systems	69
5.3. Analyzing Experimental Data	70
5.4. References	71

Appendix A. Sample Random Number Generators	72
A.1. Sample Random Number Function 1	72
A.2. Sample Random Number Function 2	73
A.3. Sample Random Number Function 3	74
A.4. Sample Random Number Function 4	75
 Appendix B. FORTRAN Programs Used in Chapter 2	 76
B.1. Sphere	76
B.2. Prolate Spheroid	82
B.3. Cylinder	88
B.4. Tilted Cylinder	94
B.5. Elongated Hexagon	102
B.6. Bundled Cylinders	114
 Appendix C. FORTRAN Programs Used in Chapter 3	 123
C.1. Core/Corona	123
C.2. Core/Shell	130
C.3. Layered Cylinder	138

LIST OF FIGURES

1.1. Scattering Experimental Setup	3
1.2. Representation of \underline{S} and \underline{S}_0	4
1.3. Illustration of system for Equation 1.1	6
1.4. Illustration of system for Equation 1.3	7
1.5. SAXS photograph of drawn Linear Polyethylene	13
1.6. Sphere Visualization	14
2.1. Sphere: $R_s = 2.0$ nm: $R_L = 0.985$ for simulation time of 30 minutes.	28
2.2. R_L as a function of simulation run time and the number of scattering points generated. Sphere A: $R_s = 0.5$ nm. Sphere B: $R_s = 1.0$ nm.	29
2.3. Picture of a scattering object, coordinate axes and the scattering vector, \underline{q} .	31
2.4. Prolate Spheroid: major axis = 2.0 nm, minor axis = 1.0 nm, $\phi = \pi/6, \pi/3$: $R_L = 0.998, 0.972$ respectively and simulation run times were each 30 minutes.	31
2.5. Cylinder: $H = 0.5$ nm, $R_c = 0.5$ nm, $\phi = \pi/6, \pi/3$: $R_L = 1.0, 1.0$ respectively and simulation run times were each 30 minutes.	33
2.6. Tilted Cylinder: $H = 4.0$ nm, $R_c = 2.0$ nm, $\phi = \pi/4, \sigma = \pi/6, \pi/3$: simulation times were 30 minutes.	35
2.7. A. Elongated Hexagon: $H = 2.0$ nm, $R_{\text{hex}} = 2.0$ nm, $\phi = \pi/6, \pi/3$: simulation times were each 30 minutes. B. Seven Bundled Cylinders: $H = 4.0$ nm, $R_{\text{bundled}} = 2.0$ nm (each cylinder), $\phi = \pi/6, \pi/3$: simulation times were each 30 minutes. Small diagrams of each morphology are included.	36

2.8. Elongated Hexagon compared to Seven Bundled Cylinders: Elongated Hexagon: $H = 3.0$ nm, $R_{\text{hex}} = 3.0$ nm, $\varphi = \pi/4$: Seven Bundled Cylinders: $H = 3.0$ nm, $R_{\text{bundled}} = 1.0$ nm (each cylinder), $\varphi = \pi/4$: simulation times were each 30 minutes. Small diagrams of each morphology are included.	37
3.1. Visual interpretations: a. Core/Corona Sphere Model; b. Core/Shell Sphere Model; Centro-Symmetric Layered Cylinder.	44
3.2. Core/Corona: $R_{\text{core}} = 2.0$ nm, $R_{\text{corona}} = 2.4$ nm, $\varpi = -0.1$: $R_L = 0.972$ 45 minute simulation run time.	45
3.3. Core/Shell: $R_{\text{core}} = 1.5$ nm, $R_{\text{shell-inner}} = 2.0$ nm, $R_{\text{shell-outer}} = 2.5$ nm, $\varpi = 0.75$: $R_L = 0.980$, 30 minute simulation run time.	45
3.4. Diagram of cylindrical scattering object, coordinate axes.	47
3.5. Layered Cylinder: $R_c = 1.0$ nm, $H = 3.0$ nm, $H_{\text{primary}} = 0.200$ nm, $H_{\text{secondary}} = 0.185$ nm, $\varphi = \pi/4$, $\varpi = 0.00, 0.25, 0.50, 0.75, 1.00$: 30 minute simulation run time for each curve.	49
3.6. Layered Cylinder: $R_c = 1.0$ nm, $H = 3.0$ nm, $H_{\text{primary}} = 0.200$ nm, $H_{\text{secondary}} = 0.185$ nm, $\varphi = \pi/4$, $\varpi = 0.00, -0.25, -0.50, -0.75, -1.00$: 45 minute simulation run time for $\varpi = -0.25, -0.50, -0.75, -1.00$; 30 minute simulation run time for $\varpi = 0.00$ (extended run time because of “negative scattering”).	50
3.7. Layered Cylinder: $R_c = 1.2$ nm, $H = 2.6$ nm, $H_{\text{primary}} = 0.20$ nm, $H_{\text{secondary}} = 0.05$ nm ($z = 0.25$), 0.10 nm ($z = 0.50$), 0.15 nm ($z = 0.75$), $\varphi = \pi/8$, $\varpi = 0.25$: 30 minute simulation run time for each curve.	53
3.8. Layered Cylinder: $R_c = 1.2$ nm, $H_{\text{primary}} = 0.20$ nm ($z = 1.0$), 0.10 nm ($z = 2.0$), 0.025 nm ($z = 8.0$), $H_{\text{secondary}} = 0.20$ nm, $\varphi = 3\pi/8$, $\varpi = 0.25$: 30 minute simulation run time for each curve.	53
5.1. One “cell” of the double diamond surface.	68
5.2. Three-dimensional double diamond array.	69

CHAPTER 1

Fundamentals of Small-Angle Scattering and Monte-Carlo Simulation

1.1 Introduction

Electromagnetic radiation can be used to obtain information about materials whose dimensions are on the same order as the radiation wavelength, λ . An ordinary glass of milk vividly illustrates this principle. In normal visible light, i.e. that emitted from a household fluorescent bulb, a glass of milk appears as a continuous, milky white fluid, thus the name milk. However, place the same glass of milk in an ultraviolet “black” light and it appears as an emulsion of particles because the wavelength of the “black” light is similar to the dimensions of the butterfat that is emulsified in milk. Another example of how radiation illuminates length scales similar to that of the radiation wavelength is the blue appearance of the sky. Density fluctuations in the earth’s atmosphere are small and are therefore closer to the wavelength of the blue portion of the visible light spectrum. Thus, the color of scattered sunlight and the appearance of the sky are blue. If no gases were present in the earth’s atmosphere at all then our sky would appear black like that of the moon (Guinier, 1984).

Around the turn of the 20th century, Röntgen discovered radiation with wavelength much smaller than that of visible light. Röntgen named this high-energy radiation “x-rays” because of their unknown nature (Glasser, 1945). Soon after this

discovery, von Laue and his associates discovered that crystals scatter x-rays in distinct patterns (von Laue, 1950). It was quickly recognized that these patterns give direct insight into the structure of the materials that caused the scattering. Since these early discoveries, many technical advances made x-ray scattering one of the most powerful characterization tools available for both homogenous and heterogeneous materials. Today, scattering from x-rays, neutrons and light is used by scientists in many different disciplines to study a vast range of materials ranging from polymers to proteins.

Small-Angle Scattering (SAS) experiments commonly appear as shown in Figure 1.1 and generally follow this procedure; irradiate a sample with some type of radiation (x-rays, neutrons or light), measure the resulting scattering pattern, then determine the structure that caused the observed pattern. Scattering patterns are caused by the interference of secondary waves that are emitted from various structures (electrons for x-rays and light, or nuclei for neutrons) when irradiated. Scattering of x-rays is caused by differences in electron density, scattering of neutrons is caused by differences in scattering power of different nuclei and scattering of light is caused by differences in refractive index. Since the larger the diffraction angle the smaller the length scale probed, wide angle x-ray scattering (WAXS) is used to determine crystal structure on the atomic length scale while small-angle x-ray scattering (SAXS) or small-angle neutron scattering (SANS) is used to explore microstructure on the colloidal length scale (Kratky & Porod, 1949). Light is used similarly but because the wavelength of light is much greater than that of x-rays or neutrons, light scattering is

used for much larger structures like the phases in blends of elastomers or particle size distributions.

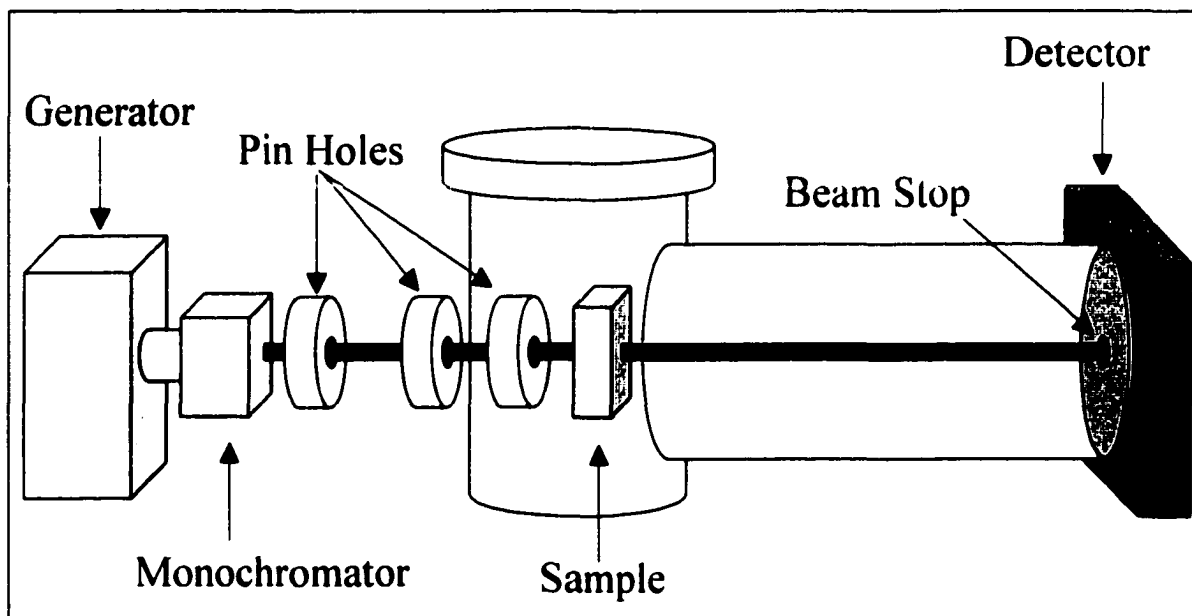


Figure 1.1. Scattering Experimental Setup

Unlike an electron micrograph, small-angle x-ray scattering patterns do not give morphological information directly. The result of a SAXS experiment is essentially the intensity of the Fourier transform of the electron density and must be interpreted in order to determine morphology. One fundamental problem with any scattering experiment is that two different morphologies can, in theory, give identical scattering patterns. Generally, one cannot reconstruct the exact microstructure uniquely from a SAXS pattern because in a scattering experiment only the scattered radiation intensity can be measured and all phase information is lost. Therefore, one cannot be absolutely sure that a scattering pattern is due to a particular morphology.

Still however, usually something is known about the system in question, so that it is often (but not always!) reasonable to assume that if a particular model is shown to fit the scattering pattern, then the model is a correct description of the morphology. Nevertheless, many different approaches exist to extract morphological information from a SAXS pattern.

1.2 General Scattering Theory

When x-rays of known wavelength are scattered, a scattering vector, \underline{q} , can be defined that is equal to $\frac{2\pi}{\lambda}(\underline{S} - \underline{S}_0)$. This important definition is based on the wavelength of radiation, λ , and unit vectors in the incident and scattered x-ray directions, respectively, \underline{S}_0 and \underline{S} . As shown in Figure 1.2, the angle between \underline{S}_0 and \underline{S} is 2θ . Thus the resulting magnitude of the scattering vector, $|\underline{q}|$, is equal to $\frac{4\pi}{\lambda} \sin \theta$.

The scattering vector is the basis for all scattering equations.

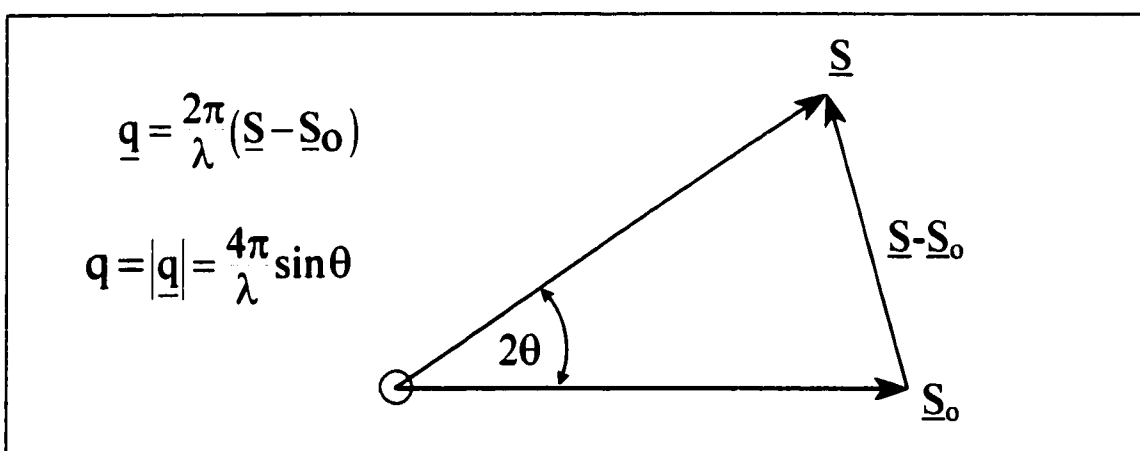


Figure 1.2. Representation of \underline{S} and \underline{S}_0

Incoherent or Compton scattering, which is virtually nonexistent at small angles, refers to scattered waves that have changed phase and wavelength (Alexander, 1969). Neglecting Compton scattering (no phase change and no wavelength change), the coherent scattering of x-rays $[I(\underline{q})]$ by a single fixed particle is mathematically represented by Equation 1.1 which is the fundamental scattering equation (Schmidt, 1995). In Equation 1.1, $I_e(\underline{q})$ is the scattered intensity of a single electron measured in identical conditions as $I(\underline{q})$. As shown in Figure 1.3, f_j and f_k are the scattering powers of the j^{th} and k^{th} atoms respectively. Also in Figure 1.3, \underline{r}_j and \underline{r}_k are vectors from some arbitrarily chosen origin to the center of the j^{th} and k^{th} atoms respectively.

$$\frac{I(\underline{q})}{I_e(\underline{q})} = \sum_k^n \sum_j^n f_k f_j \cos[\underline{q} \cdot (\underline{r}_j - \underline{r}_k)] \quad \text{Equation 1.1}$$

The square root of the right hand side of Equation 1.1 is known as the form factor, F_k . Form factors have been derived analytically for many simple geometries. For example, the form factor for a sphere of radius R_s , is given in Equation 1.2 (Guinier & Fournet, 1955). This expression was derived by Lord Rayleigh and is often denoted as $\Phi(qR)$ when used in complex scattering expressions. Some of the form factors with analytical expressions include cylinders (circular and oval cross sections), spheroids (prolate and oblate), spherical shells, concentric spherical shells, parallelepipeds, infinitely thin rods and infinitely flat circular disks. A more exhaustive list can be found in the work of Pedersen (1997).

$$\frac{I(\underline{q})}{I_c(\underline{q})} = [F_k(\underline{q}, R_i)]^2 = \left[3 \frac{\sin(qR_i) - (qR_i)\cos(qR_i)}{(qR_i)^3} \right]^2 \quad \text{Equation 1.2}$$

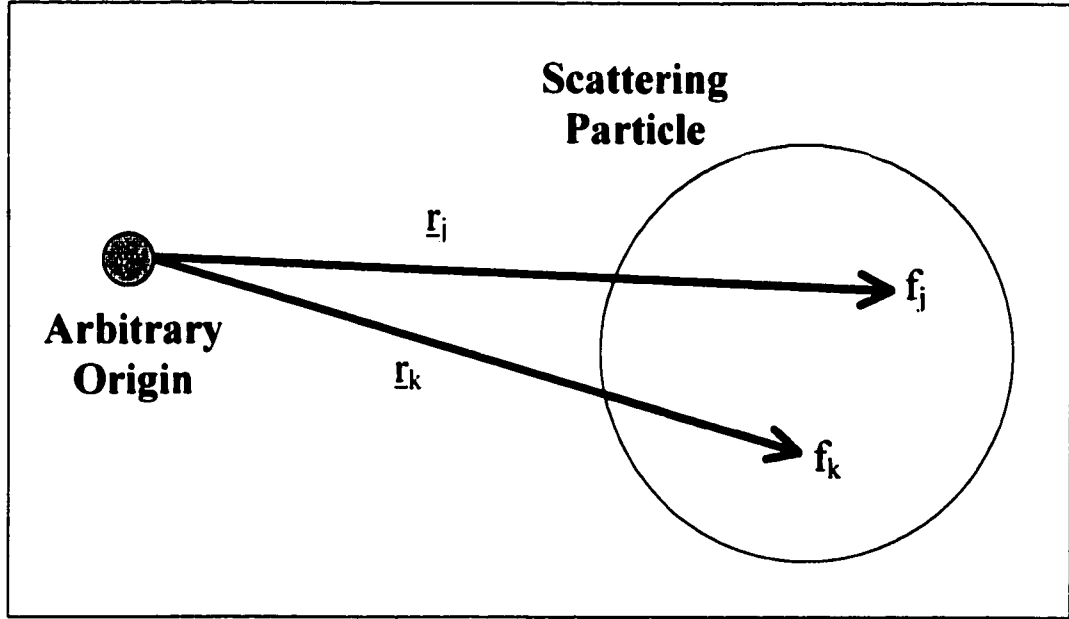


Figure 1.3. Illustration of system for Equation 1.1

As particle density increases (particle volume divided by total volume), inter-particle interference becomes a factor, i.e. the scattering pattern depends on the locations of the individual particles. At high particle densities inter-particle interference can dominate the scattering curve. The Debye Equation describes scattering from multiple particle systems with inter-particle interference included. The Debye Equation is analogous to Equation 1.1 and is given in Equation 1.3. Equation 1.3 is based on the form factors of N different particles and a vector from the centers of the j^{th} and k^{th} scattering particles, $\underline{R}_k - \underline{R}_j$, as shown in Figure 1.4.

Equation 1.1 is based on scattering “points” and Equation 1.3 is based on scattering particles. Scattering “points” have no finite volume while scattering particles do have finite volume. Therefore when simulating scattering by choosing points, scattering “points” can be chosen randomly while the centers of scattering particles cannot, since the latter is limited by the fact that no two particles can occupy the same space.

$$\frac{I(\underline{q})}{I_c(\underline{q})} = \sum_k \sum_j F_k(\underline{q}) F_j(\underline{q}) \cos[\underline{q} \cdot (\underline{R}_k - \underline{R}_j)] \quad \text{Equation 1.3}$$

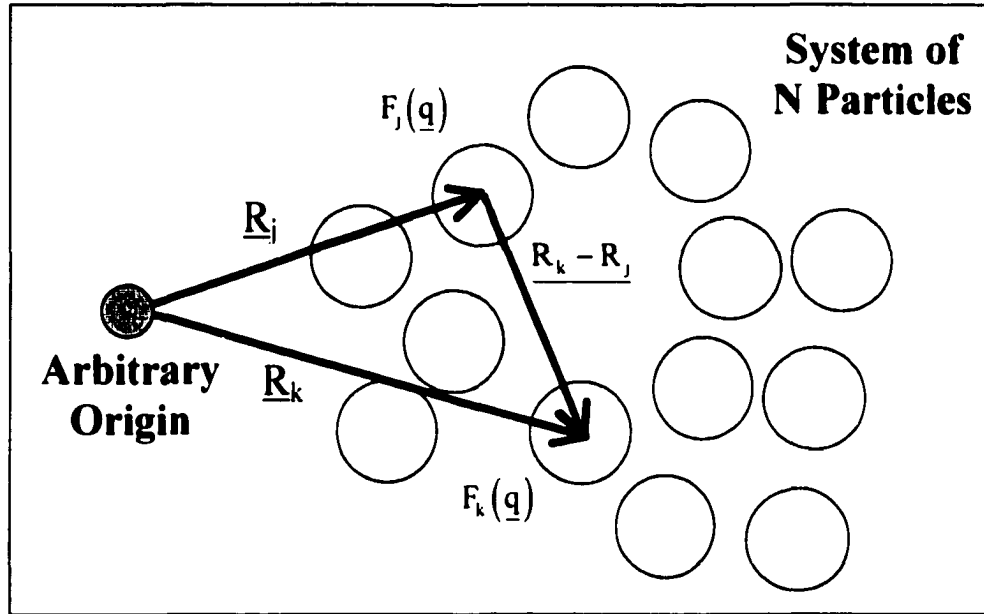


Figure 1.4. Illustration of system for Equation 1.3

Occasionally, it is desired determine scattering from a collection of anisotropic particles whose orientation is random. An example of this would be a dilute,

quiescent solution of particles. Equations 1.4 and 1.5 result from Equations 1.1 and 1.3 respectively, after orientation averaging. In these equations q is the magnitude of the scattering vector, $|\underline{q}|$. r_{jk} is the magnitude of the “point” to “point” vector, $|\underline{r}_j - \underline{r}_k|$, and R_{jk} is the magnitude of the particle center to particle center vector, $|\underline{R}_k - \underline{R}_j|$.

$$\frac{I(q)}{I_e(q)} = \sum_k \sum_j f_k f_j \frac{\sin(qr_{jk})}{qr_{jk}} \quad \text{Equation 1.4}$$

$$\frac{I(q)}{I_e(q)} = \sum_k \sum_j F_k(q) F_j(q) \frac{\sin(qR_{jk})}{qR_{jk}} \quad \text{Equation 1.5}$$

1.3 Analysis of Scattering Data and Goals of this Research

There are many analytical methods used to analyze scattering patterns from both single and multiple particle systems. At low q , the Guinier Law can be used to estimate the radius of gyration of the scattering particle (Glatter & Kratky, 1982). The Guinier Law works well with most single particles with the exception of very anisotropic particles. At high q , the Porod Law can be used to estimate the total surface area of all scattering particles regardless of shape (Guinier, 1994). The Porod Law assumes a two-phase system with sharp interfaces and works for multiple particle systems as well.

Generally, analyzing an entire scattering pattern requires significant effort, and a number of different methods are used to analyze data. The purpose here is not to familiarize the reader with all these methods but rather to describe the most commonly used approaches. The reader should note that the Monte-Carlo methods described in

this thesis can be used to calculate entire scattering patterns, and Monte-Carlo methods are applicable to almost any type of system; however these methods are very time consuming and hence should only be used when alternatives do not exist.

Morphologies without Interparticle Interference

Structural parameters from simple single-particle spherical morphologies can be determined in a matter of seconds using a least squares fit of Equation 1.2. In other words, a scattering curve is calculated from Equation 1.2, then it is smeared (see Section 1.5) and compared to the experimental scattering pattern. This process is repeated until the calculated scattering curve most closely matches the experimental one, i.e. the sum of the squares of the differences between the experimental scattering points and the calculated scattering points is minimized. Essentially all real examples of identical single particles with perfect alignment are objects that also have analytical solutions, i.e. cylinders or spheroids, and can be solved with essentially the same procedure.

The first step in this research was to simulate scattering from these simple systems to prove that a Monte-Carlo method can in fact be used to simulate scattering. The most important outcome of this part of the research was in learning how to properly perform these simulations. The existence of analytical solutions allowed me to pinpoint errors in the code, allowed for the determination of simulation times required, and also aided in the choice of a random number generator. Chapter 2

describes this effort in detail, and was the basis for a paper published in the Journal of Applied Crystallography.

If single particles are allowed to have a distribution of sizes or orientations, then the situation becomes quite complicated. Analytical methods can be used if spheres have a distribution of sizes and such methods have been applied to systems of Al-Li precipitates (Pedersen, 1993). Anisotropic objects with random orientation fall into two categories. Scattering patterns from some objects with random orientation (discs, cylinders, spheroids) have analytical solutions. Cylinders for example have been used to model dilute solutions of hemoglobin (Fournet, 1951). Some objects with random orientation do not have analytical descriptions of the scattering pattern; these systems require Monte-Carlo simulations developed elsewhere and described in more detail in Chapters 2 and 4. If the orientation of anisotropic objects is non-random, i.e. a fibrinogen system under shear, then there are no other approaches other than the Monte-Carlo approach given in this thesis to describe the complete scattering pattern. Chapter 3 describes this effort in detail, along with an effort to describe single particle systems with more than 2 electron densities. Other than spherical objects, systems with more than 2 electron densities do not have analytical solutions.

The ultimate goal of this research is to develop a Monte-Carlo method that is able to fit real experimental data using the approach given in this thesis. For the simplest systems, i.e. identical single particles that are infinitely symmetric (spheres) or identical single particles with perfect alignment, this goal could currently be reached if a suitable least-squares routine were developed. The former could probably

be coded and run on a fast PC, while the latter is obviously more time-consuming because an entire two-dimensional pattern is necessary. On a PC, many weeks would be required to fit a two-dimensional scattering pattern with a Monte-Carlo model, which is too far long to be of practical use. However a high-speed supercomputer could perform this function in a reasonable amount of time. Practically, there is no reason to develop such a routine for these systems because simpler methods exist to analyze scattering data from these systems.

For both multiple-electron density systems and anisotropic objects with preferred orientation, there would be definite practical benefit in developing least-square routines to fit real experimental data. Systems such as elongated micelles, and voids in elongated polymers presumably would scatter without interparticle interference, and currently there are no good methods to analyze data from such systems. However, the many weeks of simulation time for a perfectly oriented system could possibly increase by an order of magnitude for a system with non-perfect orientation.

Morphologies with Interparticle Interference

Dense multiple particle systems (both with and without preferred orientation) are covered in Chapter 4. Dense multiple particle systems are the most common example of “real” systems and have two-dimensional SAXS patterns like that shown in Figure 1.5. Analysis of scattering from dense multiple particle systems is extremely difficult. Normally, if the dense multiple scattering systems are comprised

of spheres then the analytical expression derived by Zernicke and Prins can be used as described in Chapter 4. Because no other approaches exist, scattering from systems of dense anisotropic scattering particles has to be analyzed using a Monte-Carlo method. Monte-Carlo data analysis for systems of dense anisotropic scattering particles is done using Equation 1.3 or Equation 1.5. One problem in simulating scattering from dense anisotropic systems is that Equation 1.3 is valid for only a single arrangement of particles and thus, a large number of particles are required in the simulation. Equation 1.3 is also a function of a double summation over all scattering particles. Comparing small systems of 100 particles and 1000 particles, the 1000 particle systems would take 100 times longer to calculate using Equation 1.3. To generate the entire scattering pattern from a single arrangement of particles would take nearly a year for system with tens of millions of particles, which I think is the size required for accurate statistics. Another problem is that it can be very difficult to get high particle densities by merely placing particles. In our lab it has taken several days to get particle densities near 0.4 for approximately 1 million spheres by random placement. An alternative to random placement is given in Chapter 4, but the procedure is still quite time-consuming. These demands make computer simulation of scattering from these kinds of systems impossible.

To give the reader a better understanding why a Monte-Carlo methods might be a good tool to analyze scattering data, the general ideas of Monte-Carlo techniques need to be explored. This section should help the reader understand why a Monte-Carlo technique is a much more time-consuming approach to SAXS data analysis.

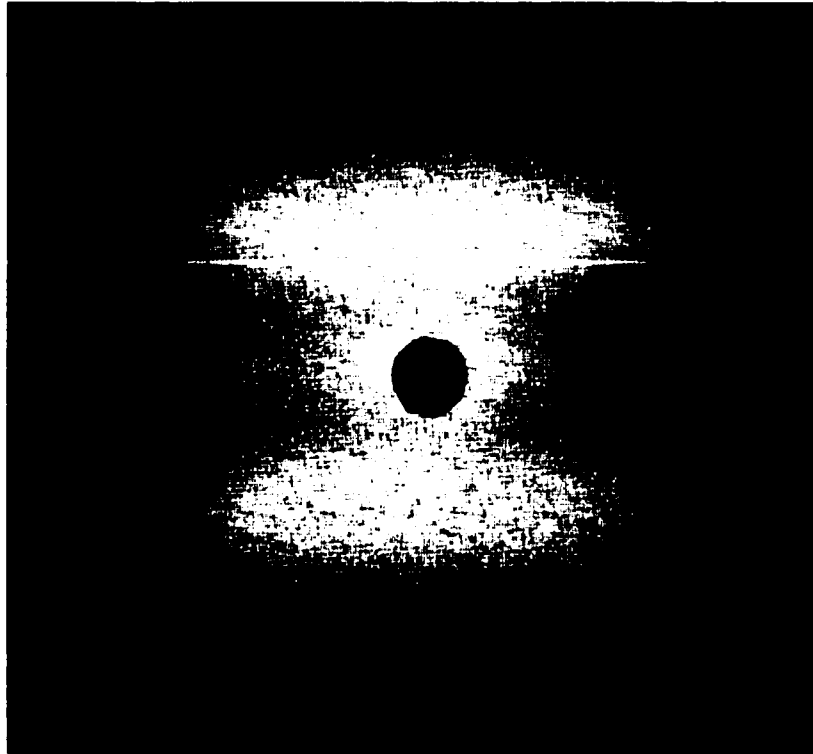


Figure 1.5. SAXS photograph of drawn Linear Polyethylene

1.4 General Monte-Carlo Theory

Generally, a Monte-Carlo method is a technique to solve a complex problem by the observation of a random process whose parameters are based on the complex problem (Buslenko et al., 1966; Niederreiter, 1992). Monte-Carlo methods are therefore based on the sampling of the devised random process. The following description should clarify this definition.

Consider the task of choosing points in space to simulate a sphere with a radius equal to 10 cm. The points in space, once chosen, will then be used as scattering points. One quickly realizes that a real sphere is a continuous object while a sphere simulated via “points in space” is nothing more than a concentration of discrete points. At first thought, it may seem easiest to accomplish this goal by either building a sphere from spherical shells or from a grid of points, however, a Monte-Carlo method can also be employed. Figure 1.6 is a visualization of these three ideas.

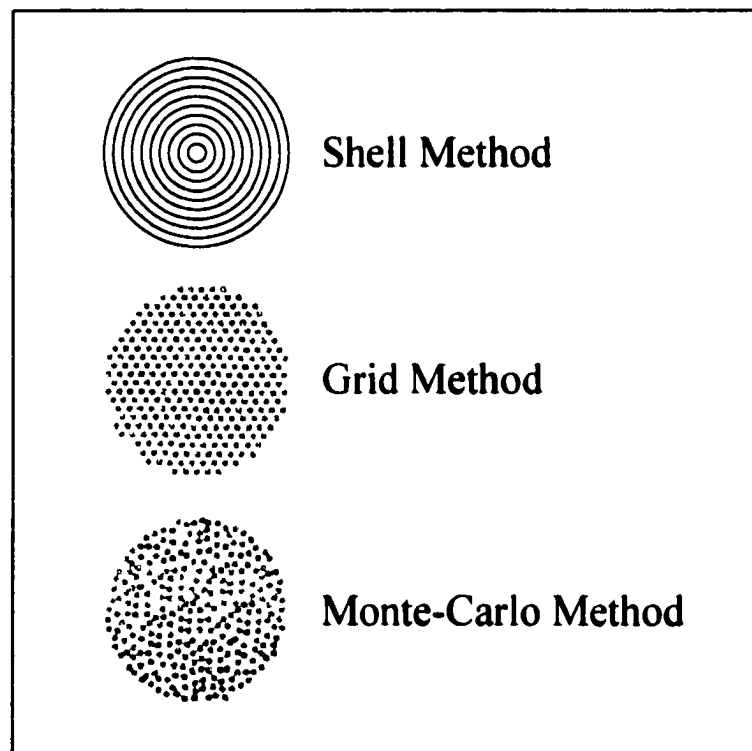


Figure 1.6. Sphere visualization

To build a sphere from spherical shells the following must be done.

1. Select an origin.
2. Build a very small shell around this origin with points in space.
3. Continue to build larger shells that are equidistant and concentric about the origin until the radius of 10 cm is met.

To use a grid approach the following must be done.

1. Arrange a grid of points larger than 20 cm x 20 cm x 20 cm and choose the center of the grid as the origin.
2. Choose every point whose distance from the origin less than or equal to 10 cm.
3. Use the chosen points in space as scattering points.

Using a Monte-Carlo method, this problem would be approached in the following manner.

1. For some box larger than 20 cm x 20 cm x 20 cm, choose the center as the origin.
2. Randomly select points inside (or on the surface) the box.
3. If the distance between the origin and random point is less than or equal to 10 cm then use the random point as a scattering point.

There are problems with the “shell” and “grid” methods that the Monte-Carlo method overcomes. In the “shell” method the problem is that the distance between the discrete shells causes artifacts in the scattering pattern. Also, it is very difficult to get a constant point density with this method. In other words, as the shells get smaller the

number of points that are needed per shell is a function of the cube root of the radius, hence how does one maintain that the number of points is a whole number. In the “grid” method, artifacts caused by the regular grid can be seen in the scattering. Also, the exterior of the sphere is very rough because the sphere is made from a cubic grid of points, this gives poor scattering results because the scattering curve is very sensitive to what happens at the edge. Therefore, the act of choosing random points to simulate the sphere is by far the most appropriate choice.

However, true randomness is impossible to achieve with random number generators normally available on computers. To best illustrate the non-random behavior, one could use a spreadsheet program to draw four random numbers between zero and 100. The expected average of the four numbers should be approximately 50 with an approximately even distribution. If the first number drawn is roughly 20, the second number drawn is roughly 80 and the third number drawn is roughly 60, the spreadsheet will force the fourth number to be approximately 40. In other words, the (not so random) random number generator approximately forces the correct average and expected distribution. Using FORTRAN, random number generation is much better but of course there is the tradeoff with computer run time. In other words the better the randomness of the number generator the longer the computer run time.

It should be fairly obvious that using a Monte-Carlo approach to analyze scattering data takes significant more time than using an available theoretical model. For this reason, Monte-Carlo approaches should only be used as a “last resort”.

1.5 Experimental Constraints

Whatever the method of data analysis, there are a number of practical experimental constraints that must be considered; for example, only a certain range of q values can be utilized. The SAXS bound at lower q values is the beam because the beam has a finite cross section. In other words, the intensity of the main beam is much larger than the scattered intensity at very small q values. Actually, extending the usable angular range to angles very close to the beam is quite difficult and is the subject of a significant body of literature on camera design. The upper SAXS bound is usually between $\theta = 3^\circ$ and $\theta = 5^\circ$ $\left(q = \frac{4\pi}{\lambda} \sin \theta \right)$. The upper SAXS bound is usually dependent on the dynamic range of the detector because the intensity of the SAXS pattern typically decreases as a function of q^4 . This upper bound can be extended by moving the detector and collecting for longer times, but for most morphologies the effort is not worth the benefit.

In a SAXS experiment, smearing of the scattered radiation occurs which results in a loss of resolution. This smearing is the result of basically three things; the finite size of the x-ray beam, the finite size of the “pixels” on the detector and the polychromaticity of the beam. The size of the beam is the most important effect and is determined by the collimation geometry; both slits and pinholes are commonly used. Thus, corrections must be made for the pinholes or the width and heights of the slits. Smearing can also occur because of the finite size of the detection element i.e. a two-dimensional detector has pixels with finite area. In a scattering experiment, each of

these pixels “sees” the intensity over a small range of q values. Smearing of these two types is easily corrected with equations given in the literature (Lake, 1967; Register & Cooper, 1988; Barker & Pedersen, 1995). Finally, any deviation from mono-chromaticity can smear the scattered radiation. This type of smearing is usually ignored for two reasons. First, most beams use crystal monochromators and hence this effect is small. Second, it is extremely difficult to measure a wavelength polydispersity even in systems with filters rather than crystals, and hence it is ignored.

Scattering patterns given throughout this thesis are not smeared because smearing can be easily incorporated into curve fitting routines. Therefore, the curves in this thesis must be smeared to match real data. Or conversely, real data must be desmeared to match the scattering curves presented in this thesis (of course smearing is much easier than desmearing!).

1.6 References

- Alexander, L. E. (1969) *X-Ray Diffraction Methods in Polymer Science*. New York: John Wiley & Sons, Inc.
- Barker, J. G. & Pedersen, J. S. (1995). *J Appl Cryst* 28, 105
- Buslenko, N. P., Golenko, D. I., Shreider, Yu. A., Sobol, I. M. & Sragovich, V. G. (1966). *The Monte Carlo Method*, translated by G. J. Tee, translation edited by D. M. Parkyn. Oxford: Pergamon Press Ltd.
- Fournet, G. (1951). *Bull Soc Franç Minéral Et Crist* 74, 39
- Glasser, O. (1945). *Dr. W. C. Röntgen*. Springfield: C. C. Thomas
- Glatter, O. & Kratky, O. (1982). Editors. *Small-Angle X-Ray Scattering*. New York: Academic Press

- Guinier, A. (1984). *The Structure of Matter*. London: Edward Arnold Ltd.
- Guinier, A. (1994). *X-Ray Diffraction In Crystals, Imperfect Crystals, and Amorphous Bodies*, translated by P. Lorrain & D. Lorrain. New York: Dover Publications.
- Guinier, A. & Fournet, G. (1955). *Small-Angle Scattering Of X-Rays*, translated by C. B. Walker. New York: John Wiley & Sons, Inc.
- Kratky, O. & Porod, G. (1949). *J Coll Sci* 4, 35
- Lake, J. A. (1967). *Acta Cryst* 23, 191
- von Laue, M. (1950). *History of Physics*, translated by R. Oesper. New York: Academic Press
- Niederreiter, H. (1992) *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: Society For Industrial And Applied Mathematics
- Pedersen, J. S. (1993). *Phys Rev B* 47, 657
- Pedersen, J. S. (1997). *Adv Coll Interface Sci* 70, 171
- Register, R. A. & Cooper, S. L. (1988) *J Appl Cryst* 21, 550
- Schmidt, P. W. (1995). *Modern Aspects of Small-Angle Scattering*, edited by H. Brumberger. Boston: Kluwer Academic Publishers

2.1 Introduction

One method to extract morphological information from a SAXS pattern is to use a Monte-Carlo method to model the scattering from a given object, smear the modeled scattering and then compare it to real scattering data. Using this approach, one assumes a morphology and calculates the scattering pattern by using a Monte-Carlo method to generate random scattering points (Stöckel et al., 1980; Hansen, 1990; Henderson, 1996). Previously, modeling methods have focused on statistically isotropic systems (systems where a random distribution of orientations exist) that are found in solution and powder diffraction. This type of system allows the use of rotationally averaged scattering curves, greatly simplifying the simulation by requiring the generation of only a one dimensional electron distribution. These previous simulation methods have calculated the rotationally averaged scattering curves by generating the correlation function, $\gamma(r)$ (Debye and Bueche, 1949), or the pair-distance distribution function, $P(r)$ (Guinier & Fournet, 1955; Glatter, 1979), which are related by the following equation, $P(r)=r^2\gamma(r)$. These functions were then integrated with their associated intensity functions, shown below [$q=(4\pi/\lambda) \sin \theta$], (Debye and

Bueche, 1949; Feigin and Svergun, 1987) to calculate the rotationally averaged scattered intensity curves.

$$I(q) = 4\pi V \int_0^{\infty} r^2 \gamma(r) \frac{\sin(qr)}{qr} dr$$

Intensity Function (correlation function)

$$I(q) = \frac{1}{4\pi} \int_0^{\infty} P(r) \frac{\sin(qr)}{qr} dr$$

Intensity Function (pair distribution function)

The following chapter describes a novel method of modeling SAXS data that is similar to the previously mentioned Monte-Carlo methods. Although inherently more time consuming because of the requirement for non-rotationally averaged scattering curves, the method presented here is distinct in that it simulates scattering for oriented systems. This method directly calculates the general scattering function from basic small angle scattering (SAS) principles and hence eliminates the use of simplifications such as a particle's inhomogeneity distribution (Feigin and Svergun, 1987) or chord distribution (Glatter & Kratky, 1982) which are advantageous to rotationally averaged scattering pattern simulations. This method's only requirement for the calculation of the entire scattering curve in a two dimensional scattering plane is that the scattering object must be centrosymmetric. However, exact centrosymmetric match between the actual scattering object and the model is not required as long as the deviations from centrosymmetry are small compared to the scattering object's dimensions (Glatter & Kratky, 1982). Thus, this technique is ideal for morphologies or orientations where an exact analytical solution for the scattering does not exist. Furthermore, the technique

presented here can be applied to small angle neutron scattering (SANS) with only minor changes in nomenclature.

In order to establish a foundation for this technique, this chapter presents simulations of scattering from single particle systems. Accuracy will be verified by calculating scattering patterns using different simulation times for objects that possess analytical solutions. Simulated scattering curves will then be compared to the respective exact theoretical solution, thus providing evidence as to the accuracy and accessibility of this method. Furthermore, scattering for three objects without theoretical scattering solutions will be given to demonstrate the capabilities of this technique.

2.2 Theoretical Background

Equation 1.1 could be used to simulate x-ray scattering from single particles systems, however, the resulting simulations would be very time consuming because of the double summation to be evaluated. Instead, by assuming that the scattering object is centrosymmetric with respect to electron density and the origin is chosen as the center of symmetry, Equation 1.1 can be simplified as shown below (Guinier & Fournet, 1955). This simplification arises from the fact that for every vector from the centrosymmetric center to the k^{th} point, \underline{Ox}_k , there exists an equal and opposite vector within the system, $-\underline{Ox}_k$. A discrete version of Equation 2.1 was used to simulate scattering from all oriented single particles given within this chapter.

$$\frac{I(\underline{q})}{I_c(\underline{q})} = \left(\int_V (\rho_k - \rho_o) \cos(\underline{q} \bullet \underline{Ox}_k) \right)^2 = [F_k(\underline{q}, \text{shape})]^2 \quad \text{Equation 2.1}$$

Where ρ_k and ρ_o are the electronic densities of the k^{th} atom and the media in which the single particle is immersed. $F_k(\underline{q})$ is designated the form factor of the scattering particle and is a function of both \underline{q} and the shape of the scattering object.

2.3 Simulation Technique and Technical Considerations

As was pointed out, past Monte-Carlo simulations were developed by either generating the correlation function or the pair distribution function and then integrating the associated intensity function. Scattering from rotationally averaged particles allows for integration in the θ and ϕ directions before requiring the implementation of the r direction distribution function in the scattering equation, which greatly simplifies simulation techniques by requiring a distribution only in a single direction. Therefore, random points inside the scattering object need only to be chosen until the r direction distribution function becomes numerically continuous. The technique presented here however, is for oriented systems. Oriented systems demand the development of a distribution in all directions, namely, the r , θ and ϕ directions.

This technique simulates scattering by the following method.

- A. Draw a box around the scattering object.
- B. Generate x , y , z random coordinates inside the box through a Monte-Carlo method.

- C. Test to see if the random coordinates are inside or on the surface of the particular morphology being investigated.
- D. Coordinates which pass the test given in part B are used as scattering points.
These points are used to generate the scattering array which is comprised of 401 q -points equally distributed between 0.0 and 7.1 nm^{-1} . Coordinates which fail the test given in part B are rejected.
- E. This iteration is repeated for a given amount of time.
- F. When the predetermined run time is over, the scattering array is then normalized by the intensity at $q=0$ which is equal to the total number of scattering points used to calculate the scattering array. Finally, the \log_{10} is taken of the scattering array.

Simulation accuracy is dependent on scattering point density and how smooth and continuous the scattering point distribution is developed. Because the scattering pattern is being simulated by scattering from many discrete scattering points and not a continuum (solid particle), simulation accuracy improves with increased scattering point density. Increasing scattering point density in the model is achieved by simply increasing the computer run time, which increases the number of iterations and thus the number of scattering points. The smoothness of the scattering point distribution is dependent upon the quality of the random number generator.

The accuracy of scattering simulation is also dependent on the smoothness and continuity of the scattering point distribution relative to the shape of the object. A sphere for instance is infinitely symmetrical and requires a more fully developed

scattering point distribution than a “less” symmetrical object such as a flat disk.

Additionally, because the scattering object is assumed to be centrosymmetric for any scattering point chosen its symmetrical “sister” is also chosen to eliminate any variation caused by nonsymmetrical scattering points. A scattering pattern that has a bigger range of intensity variation over the chosen angular range necessarily also requires more simulation run time in order to obtain better agreement between the simulated and actual data.

A quantitative measure of the each simulation’s accuracy was calculated using the coefficient of determination. The coefficient of determination approaches unity as a simulated scattering curve approaches the theoretical scattering curve. The coefficient of determination is designated R and has the following definition (Mendenhall & Sincich, 1992).

$$R = 1 - \frac{\sum (y_i - \bar{y}_i)^2}{\sum y_i^2 - \frac{(\sum y_i)^2}{n}} \quad \text{Equation 2.2}$$

In this equation, y_i is the simulated i^{th} data point, \bar{y}_i is the analytical solution for the i^{th} data point and n is the total number of data points. In this thesis $\log_{10}(y_i)$ and $\log_{10}(\bar{y}_i)$ has been substituted for y_i and \bar{y}_i , respectively. To distinguish between the coefficient of determination and the logarithmic expression used in this thesis, R_L will be used when referring to the latter.

Although using the logarithmic form of this expression changes the meaning of the coefficient of determination, this substitution was made because almost all scattering

curves are presented and analyzed in a semi-logarithmic format. Using the logarithmic form also has the advantage of providing a better representation of the error in the simulation over the entire q -range because the scattering intensity can vary over many orders of magnitude. The disadvantage of using the logarithmic form is that this presents difficulties for scattering curves that have zero intensity at finite q , such as a sphere. If one of the 401 simulated q -points is near a zero intensity point, an extremely large number of simulated points will be required to perfectly simulate a scattering curve. Hence in a plot of R_L vs. simulation run-time, R_L will seem to asymptote at a value which depends on the number of simulated q -points and is not equal to unity.

All simulations presented in this chapter were compiled with Microsoft's FORTRAN Powerstation Version 4.1. The computer utilized was an IBM clone computer equipped with a 200Mhz MMX Pentium chip and 72 megabytes of RAM running Microsoft's Windows95. The particles that were simulated and compared to analytical scattering curves were spheres, prolate spheroids and cylinders. These geometries were chosen primarily because of their symmetry and because each morphology presents unique problems in modeling. Additionally, these morphologies are commonly found in many systems.

2.4 Comparisons

Sphere

The analytical solution for small angle scattering from a sphere of radius R_s , can be derived by rotationally averaging Equation 2.1.

$$\frac{I(\underline{q})}{I_e(\underline{q})} = \int_0^{R_s} \rho(r) \frac{\sin(qr)}{qr} 4\pi r^2 dr \quad \text{Equation 2.3}$$

In Equation 2.3, r is the distance from the sphere's center and $\rho(r)$ is the electronic density function which is a constant for $r \leq R_s$ and zero for $r > R_s$. The resulting analytical form factor can be derived (Rayleigh, 1911).

$$F_k(q, R_s) = \frac{3[\sin(qR_s) - (qR_s)\cos(qR_s)]}{(qR_s)^3} \quad \text{Equation 2.4}$$

Figure 2.1 compares simulated scattering data with the analytical solution. Clearly, the simulated data agrees very well with the analytical solution. Figure 2.1 represents scattering from a sphere where R_s equals 2.0 nm. The run time for this simulation was approximately 30 minutes and R_L was equal to 0.985.

Although the scattering from a sphere is not anisotropic, scattering from spheres was simulated because this morphology requires the greatest number of scattering points and hence provides the most rigorous test of this technique. Therefore, this simulation was used to set the minimum run-time required in order to produce an acceptable fit for all of the objects simulated in this thesis. According to Figure 2.2 and using an arbitrary safety factor of 2, 30 minutes of run-time or roughly 1,000,000 simulated points are required to adequately simulate the scattering pattern from an arbitrary object. Just to be sure, this run-time was verified with simulations of both prolate spheroids and cylinders.

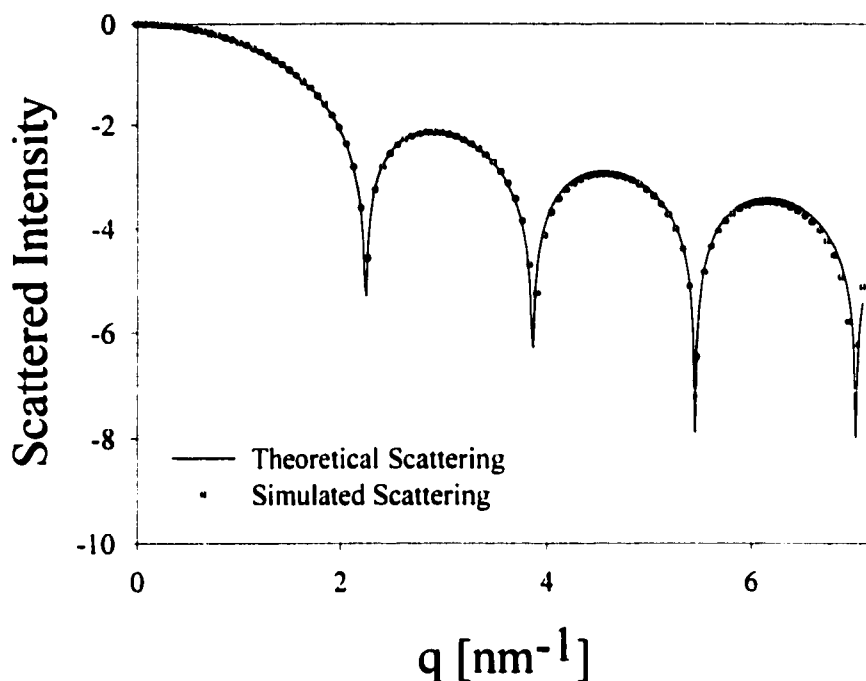


Figure 2.1. Sphere: $R_s = 2.0$ nm: $R_L = 0.985$ for simulation time of 30 minutes.

Spheres of two different radii are shown in Figure 2.2 to illustrate the concept discussed previously, the presence of zero-intensity q-points will cause the R_L value to be far-removed from one and this curve does not seem to approach unity. The scattering curve for a 0.5 nm sphere does not go to zero in this angular range, hence the much higher R_L values and the clear slow increase with simulation time of R_L . Although the data points are not shown, simulations of the scattering pattern for the 1.0 nm radius sphere for many days shows that the R_L values seem to approach unity.

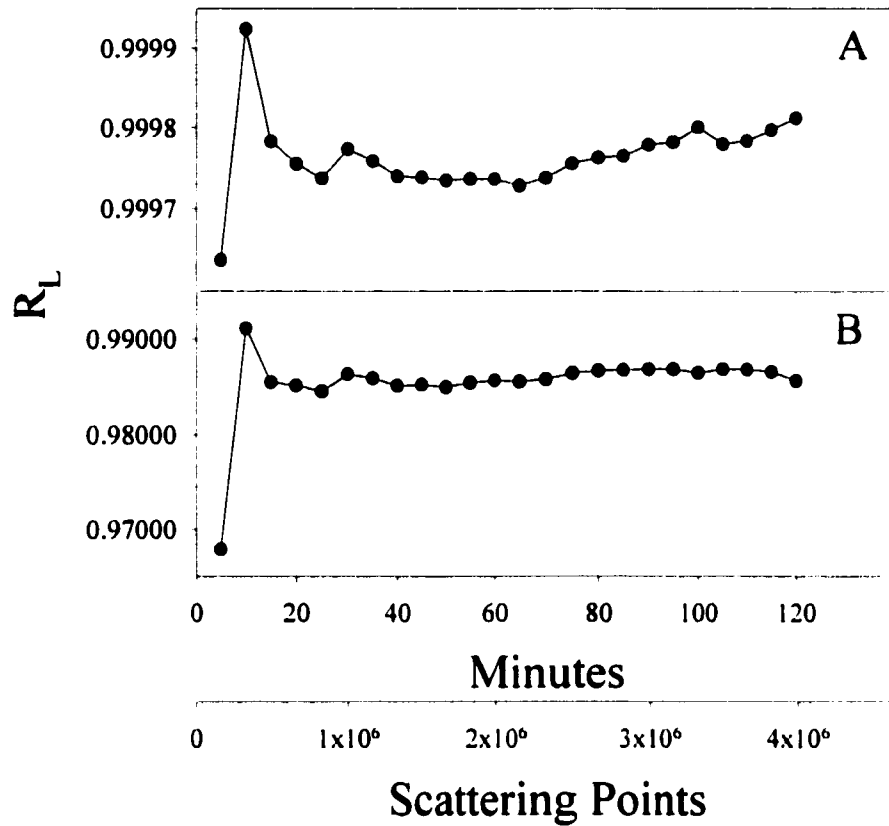


Figure 2.2. R_L as a function of simulation run time and the number of scattering points generated. Sphere A: $R_s = 0.5$ nm. Sphere B: $R_s = 1.0$ nm.

Prolate Spheroid

The analytical solution for small angle scattering from a prolate spheroid comes from direct integration of Equation 4 using a geometrical expression for the radius. If the prolate spheroid is defined to have a major axis, $2\nu\alpha$, and two minor axes, 2α , with one of the minor axis aligned with the beam (x direction), then the following solution exists (Guinier & Fournet, 1955).

$$\Psi = \alpha \sqrt{\cos^2(\phi) + v^2 \sin^2(\phi)} \quad \text{Equation 2.5}$$

$$F_k(q, \Psi) = \frac{3[\sin(q\Psi) - (q\Psi)\cos(q\Psi)]}{(q\Psi)^3} \quad \text{Equation 2.6}$$

Referring to Figure 2.3, a prolate spheroid is oriented such that the minor axes coincide with the **x** and **y** directions while the major axis coincides with the **z** direction. Also the angle in the detector plane (\angle) is designated ϕ . In the above equation and as shown in Figure 2.3, ϕ is the angle between the minor axis (**y** direction) and the projection of **q** on the detector plane. Thus for any ϕ chosen, scattering in that particular “slice” can be calculated. If many angles for ϕ are simulated the two dimensional scattering pattern can be generated. However, only ϕ values from 0 to $\pi/2$ need to be considered since the scattering in the other quadrants can be easily determined from symmetry.

The simulations shown in Figure 2.4 are of the same prolate spheroid, having a major axis equal to 2.0 nm and a minor axis equal to 1.0 nm. The only difference between the two curves is the “slice” of the two dimensional scattering pattern which is calculated. The two “slices” shown in Figure 2.4 are ϕ equals $\pi/6$ and $\pi/3$. These simulations each represent approximately 30 minutes of computer run time with R_L values equaling 0.998 and 0.972 respectively.

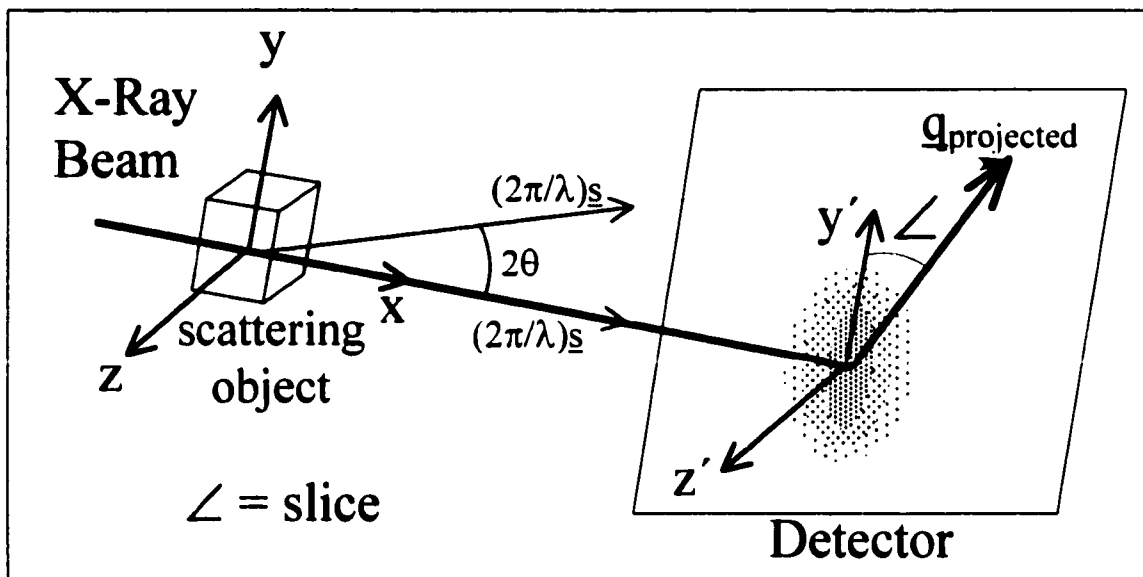


Figure 2.3. Picture of a scattering object, coordinate axes, scattering vector and slice angle, \angle .

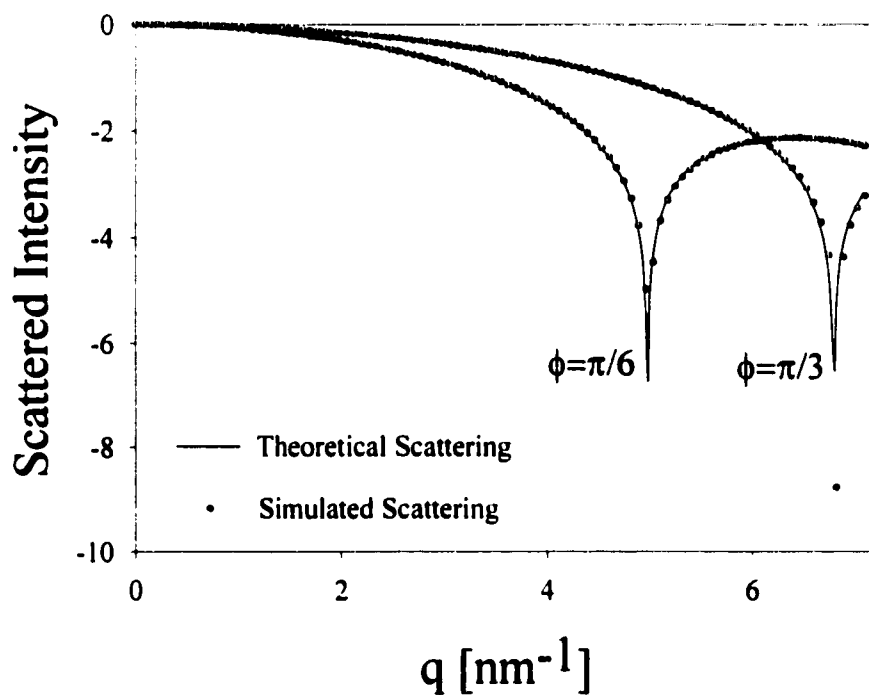


Figure 2.4. Prolate Spheroid: major axis = 2.0 nm, minor axis = 1.0 nm, $\phi = \pi/6, \pi/3$: $R_L = 0.998, 0.972$ respectively and simulation run times were each 30 minutes.

Cylinder

If a cylinder is defined to have a radius R_c , length $2H$ and its cylindrical axis parallel to the detector plane, then the following equation is the form factor (Guinier & Fournet, 1955).

$$F_k(q, R_c, H, \varphi) = \frac{2 \sin[qH \sin(\varphi)] J_1[qR_c \cos(\varphi)]}{q^2 R_c H \sin(\varphi) \cos(\varphi)} \quad \text{Equation 2.7}$$

Referring to Figure 2.3, a cylinder is oriented such that the cylinder axis coincides with the y direction. Also the angle in the detector plane (\angle) is designated φ . In the above equation and as shown in Figure 2.5, φ is the angle between the cylinder axis (y direction) and the projection of q in the detector plane. Only φ values from 0 to $\pi/2$ need to be considered since the scattering in the other quadrants can be easily determined from symmetry. Figure 2.5 shows that the simulated scattering data correlates extremely well with the analytical result for a cylinder where R_c equals 0.5 nm and H equals 0.5 nm. The two “slices” represented in Figure 2.5 are φ equals $\pi/6$ and $\pi/3$. Again, each curve represents computer run times of approximately 30 minutes and both simulations R_L values are practically equal to unity. Additionally, the cylinder simulation routine could easily be used to simulate scattering from infinitely long cylinders ($H=\infty$) and infinitely thin disks ($H=0$).

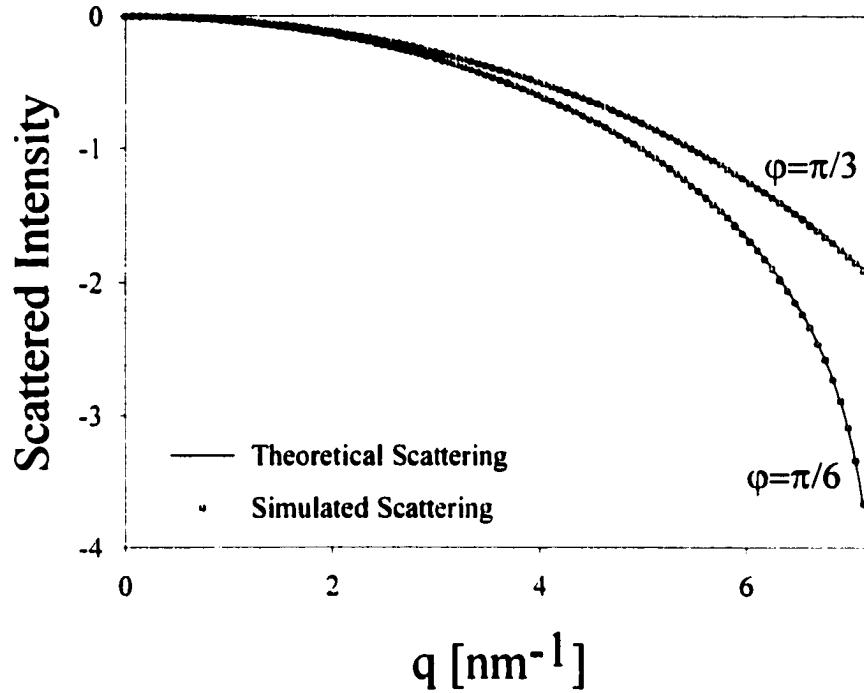


Figure 2.5. Cylinder: $H = 0.5$ nm, $R_c = 0.5$ nm, $\varphi = \pi/6, \pi/3$; $R_L = 1.0$, 1.0 respectively and simulation run times were each 30 minutes.

2.5 Objects and Orientations Without Analytical Solution

This technique can be used to simulate scattering from centrosymmetric objects whose orientation or geometry make the analytical solution extremely difficult or impossible to derive. Three examples have been chosen to demonstrate this; a tilted cylinder, an elongated uniaxial hexagon and a bundle of seven cylinders of the same radius and length.

Tilted Cylinder

The tilted cylinder's orientation is such that the tilting causes one end of the cylinder to be closer to the detector than the other end of the cylinder. The cylinder modeled in this manner had R_c equal to 1.0 nm and H equal to 2.0 nm. With ϕ having the same definition as the "ideal" cylinder as earlier, σ is defined as the angle of tilt, which is the angle between the "ideal" orientation (cylinder axis coincides with the y axis) and the actual cylinder axis orientation. In Figure 2.6, both curves represent ϕ equals $\pi/4$ with differing angles of tilt (σ). The two values for σ are $\pi/6$ and $\pi/3$. Both of these simulations required approximately 30 minutes of computer run time.

Elongated Hexagon and Bundled Cylinders

Fibers or fibrils could be modeled as an elongated hexagon or as bundled cylinders. For both of these simulations the only deviation from the nomenclature of a tilted cylinder is definition of the radius (R_c). R_{hex} for an elongated hexagon is the radius of a circle circumscribed about the hexagon and R_{bundled} for bundled identical cylinders is the radius of one of the cylinders. Figure 2.7 shows simulations of two different ϕ "slices" of both an elongated hexagon and seven identical bundled cylinders along with a small diagram of each scattering object. The two "slices" for both models are ϕ equals $\pi/6$ and $\pi/3$. The elongated hexagon modeled had no tilt (σ equals zero), R_{hex} equal to 2.0 nm and H equal to 2.0 nm. The bundled cylinders modeled also had no tilt, H equal to 8.0 nm and R_{bundled} equal to 4.0 nm. Simulations were also done of an elongated hexagon and a dimensionally comparable bundle of

seven cylinders. Figure 2.8 shows these simulations which were both of an untilted (σ equals zero) ϕ “slice” equaling $\pi/4$. For the elongated hexagon, R_{hex} equals 3.0 nm and H equals 3.0 nm. The bundled cylinders had R_{bundled} equal to 1.0 nm and H equal to 3.0 nm. As expected the simulations deviate only at higher values of q . All of these simulations represent approximately 30 minutes of computer run time.

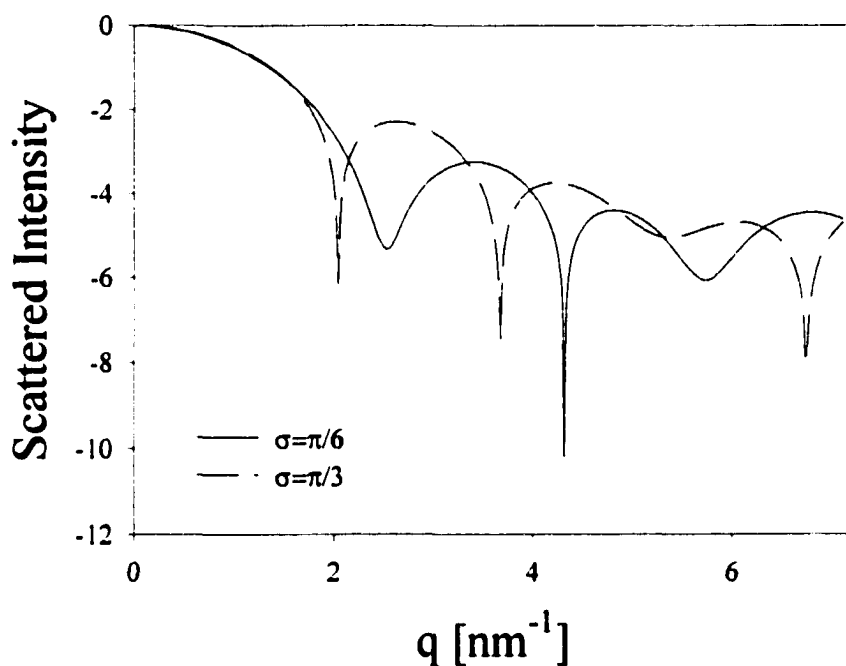


Figure 2.6. Tilted Cylinder: $H = 4.0$ nm, $R_c = 2.0$ nm, $\phi = \pi/4$, $\sigma = \pi/6$, $\pi/3$: simulation times were 30 minutes.

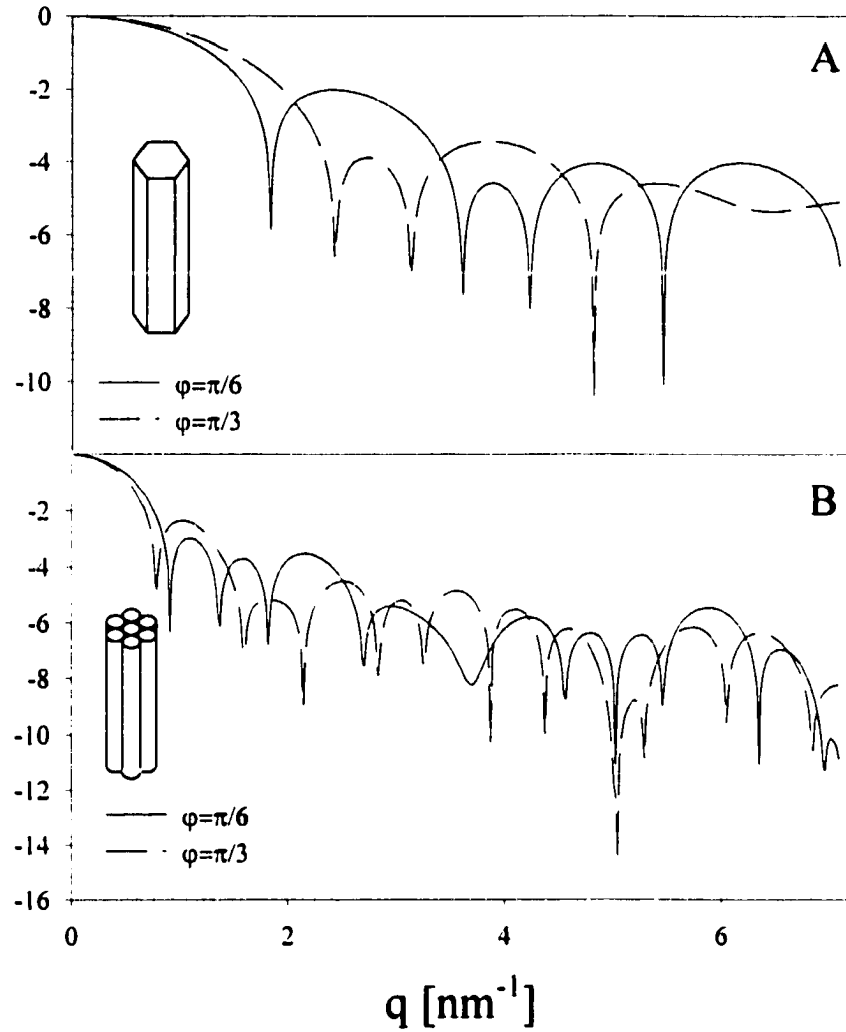


Figure 2.7. A. Elongated Hexagon: $H = 2.0$ nm, $R_{\text{hex}} = 2.0$ nm, $\varphi = \pi/6, \pi/3$: simulation times were each 30 minutes. B. Seven Bundled Cylinders: $H = 4.0$ nm, $R_{\text{bundled}} = 2.0$ nm (each cylinder), $\varphi = \pi/6, \pi/3$: simulation times were each 30 minutes. Small diagrams of each morphology are included.

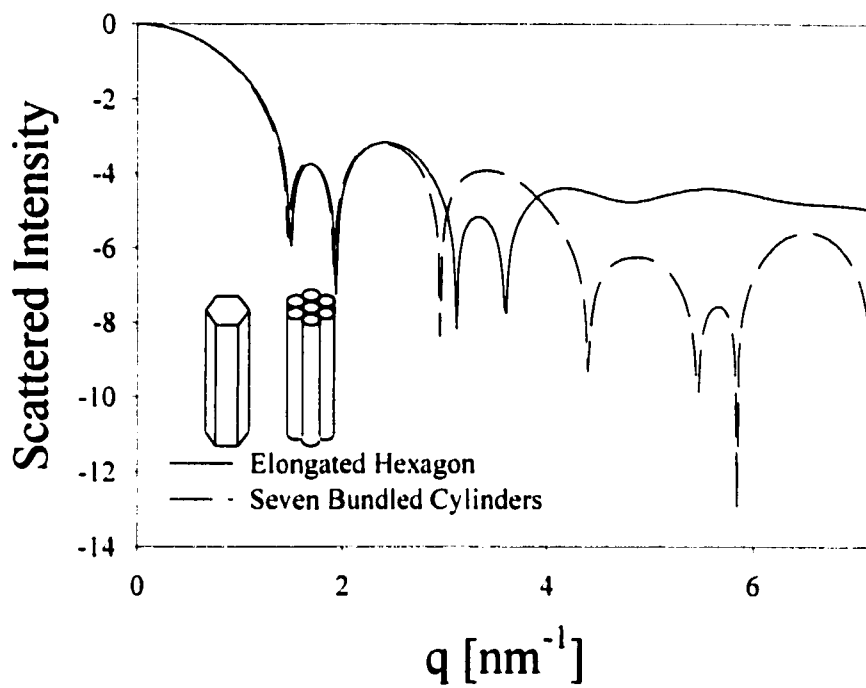


Figure 2.8. Elongated Hexagon compared to Seven Bundled Cylinders:
Elongated Hexagon: $H = 3.0$ nm, $R_{\text{hex}} = 3.0$ nm, $\varphi = \pi/4$; **Seven Bundled Cylinders:** $H = 3.0$ nm, $R_{\text{bundled}} = 1.0$ nm (each cylinder), $\varphi = \pi/4$; simulation times were each 30 minutes. Small diagrams of each morphology are included.

2.6 References

- Debye, P. & Bueche, A. M. (1949). *J Appl Phys* 20, 518
- Feigin, L. A. & Svergun, D. I. (1987). *Structural Analysis by Small-Angle X-Ray and Neutron Scattering*. New York: Plenum Press
- Glatter, O. (1979). *J Appl Cryst* 12, 166
- Glatter, O. & Kratky, O. (1982). Editors. *Small-Angle X-Ray Scattering*. New York: Academic Press
- Guinier, A. & Fournet, G. (1955). *Small-Angle Scattering Of X-Rays*, translated by C. B. Walker. New York: John Wiley & Sons, Inc.
- Hansen, S. (1990). *J Appl Cryst* 23, 344
- Henderson, S. (1996). *Biophy J* 70, 1618
- Mendenhall, W. & Sincich, T. (1992). *Statistics for Engineering and the Sciences*. San Francisco: Dellen Publishing Company
- Rayleigh, Lord (1911). *Proc Roy Soc A*-84, 25
- Stöckel, P., May, R., Strell, I., Cejka, Z., Hoppe, W., Heumann, H., Zillig, W. & Crespi, H. (1980). *Eur J Biochem* 112, 411

CHAPTER 3 SAXS Simulation of Particles with More Than One Electron Density

3.1 Introduction

In polymer science, cylindrical bodies such as fibers and fibrils with periodic axial structure or regular paracrystalline lattice distortions have been of interest since the 1950's and much effort has been devoted to collecting and understanding small-angle scattering (SAS) data from these systems (Bear & Bolduan, 1950; Hay & Keller, 1967; Pope & Keller, 1975; Gottlicher et al., 1983; Shibayama & Hashimoto, 1986; Stribeck, 1989; Rule et al., 1995; Murthy et al., 1996). Many approaches have been utilized in these previous studies, but most have generated scattering curves from paracrystalline macrolattice methodology. Paracrystalline macrolattice methodology calculates scattering from close packed scattering clusters, which are finite lamellar or cylindrical crystallites, and then averages them about the fiber axis. The approach here assumes widely separated systems of layered cylinders, so that the effect of morphological changes in the primary scattering objects (individual lamellae) can be qualitatively understood. Thus, the intent of this chapter is to establish the ability to model SAS from heterogeneous materials and help give the researcher insight to scattering phenomena from fibrillar systems.

As in Chapter 1, scattering curves are generated with the use of random scattering points and the single centrosymmetric particle equation (Equation 2.1). The

other requirements are the knowledge of the electron density ratio between the phases, a mathematical description of how the phases vary along the fiber axis and the fiber dimensions. To use Equation 2.1 there must also be a point that is the center of symmetry with respect to electron density. For a layered cylinder, this assumption only requires that the cylinder radius and lamellar thickness vary in some ordered, symmetric way (no variation, sinusoidal etc.). Strictly speaking, to satisfy centrosymmetry the ends of the fibril must also be identical. However, if the cylinder is longer than the length scale being probed by the radiation (i.e. the cylinder is 100 nm or greater), then the influence of the fibril ends is small and can be ignored and the assumption of centrosymmetry is satisfied in practice. Since essentially all fibril morphologies would be expected to be longer than 100 nm, the assumption of centrosymmetry should not be a significant restriction. Another important note is that, although all the cross sections of layered cylinders presented in this chapter are circular, this technique can also be easily adapted for any cross section geometry desired.

3.2 Theoretical Background and Description of Method

As in Chapter 2, Equation 2.1 was used to simulate scattering patterns presented in this chapter. However, the scattering object is an oriented cylinder that contains the alternating lamellae like that of a fibril.

$$\frac{I(\underline{q})}{I_e(\underline{q})} = \left(\int_V (\rho_k - \rho_o) \cos(\underline{q} \cdot \underline{Ox}_k) \right)^2 = [F_k(\underline{q}, \text{shape})]^2 \quad \text{Equation 2.1}$$

Oriented systems require the development of a scattering point distribution in three dimensions. The scattering point distribution is generated by randomly choosing (x,y,z) coordinates, then scattering intensity is calculated for each point. Scattering patterns are usually presented normalized to the scattering at zero angle. The simulation technique and code was able to simulate scattering from systems of multiple electron densities with only minor modifications from the code used in Chapter 2. A scattering array was developed for each electron density separately (i.e. from each lamellar phase in the cylinder). The array for the secondary phase was scaled at the end of the simulation as seen in Equation 3.1 and added to the array for the primary phase, this made the final scattering array.

$$\frac{I(\underline{q})}{I_e(\underline{q})}\bigg|_{\text{final}} = \frac{I(\underline{q})}{I_e(\underline{q})}\bigg|_{\text{primary}} + \left\{ \varpi \left[\frac{(I(\underline{q}=0)_{\text{primary}} V_{\text{primary}})}{(I(\underline{q}=0)_{\text{secondary}} V_{\text{secondary}})} \right] \right\} \frac{I(\underline{q})}{I_e(\underline{q})}\bigg|_{\text{secondary}} \quad \text{Equation 3.1}$$

Where ϖ is defined as follows.

$$\varpi = \frac{(\rho_2 - \rho_0)}{(\rho_1 - \rho_0)} = \frac{\rho'_2}{\rho'_1}$$

The primary phase, ρ_1 , is selected as the phase with the largest absolute value of the electron density difference relative to the surrounding media, i.e. $|\rho_k - \rho_0|$.

Thus, ϖ varying from -1 to 1 encompasses all possible morphologies of this type. ϖ constant equal to 1 the scattering pattern should be that of a regular cylinder.

Scaling at the end, according to Equation 3.1, significantly reduced the computational time required vs. scaling scattering from each random coordinate

continuously through the simulation. The term in brackets in Equation 3.1 was required to ensure that the random coordinates were chosen in the two phases according to the proper volumetric proportions. This term noticeably affected the simulated pattern only in those cases where a volumetrically small phase had a large electron density difference relative to the other phases, or when the scattering intensities from phases with electron density differences of opposing sign were approximately equal.

As in Chapter 2, the goodness of fit between simulated scattering curves and their analytical counterparts was quantitatively determined by using a deviant of the coefficient of determination (R_L) and all scattering simulations utilized $\text{CuK}\alpha$ as the wavelength of radiation (15.4242 Å). The FORTRAN code was compiled with Microsoft® Fortran Powerstation 4.1 running on an IBM compatible computer. Unlike Chapter 2, the computer utilized in Chapter 3 was comprised of a 450Mhz Pentium II™ processor with 128 Mbytes of RAM.

3.3 Results

Spherical Models

Two different scattering models with scattering patterns having analytical solutions were simulated to demonstrate that this technique can accurately predict scattering patterns from objects with greater than two electron densities. The spherical models are the Core/Corona model (also known as the Ionic Cluster and the Depleted

Zone Core/Shell model) and the Core/Shell model. Schematics are shown in Figure 3.1 and their respective analytical solutions are given in Equations 3.2 and 3.3 (MacKnight et al., 1974; Roche et al., 1980; Yarusso & Cooper, 1980). Equation 3.2 and Equation 3.3 are based on the form factor for a sphere, $\Phi(x)$, which is given in Equation 3.4, the volume of the sample illuminated by the x-ray beam, V , and the average sample volume per scattering particle, v_p .

$$\frac{I(\underline{q})}{I_e(\underline{q})} = \frac{V}{v_p} \left\{ \frac{4\pi}{3} \left[(\rho'_1 - \rho'_2) R_1^3 \Phi(\underline{q}R_1) + \rho'_2 R_2^3 \Phi(\underline{q}R_2) \right] \right\}^2 \quad \text{Equation 3.2}$$

$$\frac{I(\underline{q})}{I_e(\underline{q})} = \frac{V}{v_p} \left(\frac{4\pi}{3} \left\{ \rho'_1 R_1^3 \Phi(\underline{q}R_1) + \rho'_2 \left[R_3^3 \Phi(\underline{q}R_3) - R_2^3 \Phi(\underline{q}R_2) \right] \right\} \right)^2 \quad \text{Equation 3.3}$$

$$\Phi(x) = \frac{3[\sin(x) - (x)\cos(x)]}{(x)^3} \quad \text{Equation 3.4}$$

Figure 3.2 and Figure 3.3 compare simulated scattering data to analytical scattering data for the Core/Corona and the Core/Shell models. In Figure 3.2 the core/corona has values for $R_1 = 2.0$ nm and $R_2 = 2.4$ nm. The value for ρ_2 was chosen to be negative, thus the value for ϖ is also a negative number, $\varpi = -0.1$. The simulation run time for Figure 3.2 was 45 minutes because of the “negative” electron density which essentially adds in negative scattering. This is an important result which is believed to be general: an object which has electron density differences of opposing sign(s) can significantly increase run-times necessary for an accurate result. The

resulting $R_L = 0.972$. In Figure 3.3 the core/shell has values of $R_1 = 1.5$ nm, $R_2 = 2.0$ nm and $R_3 = 2.5$ nm while $\varpi = 0.75$. The simulation run time was 30 minutes and gave a resulting $R_L = 0.980$.

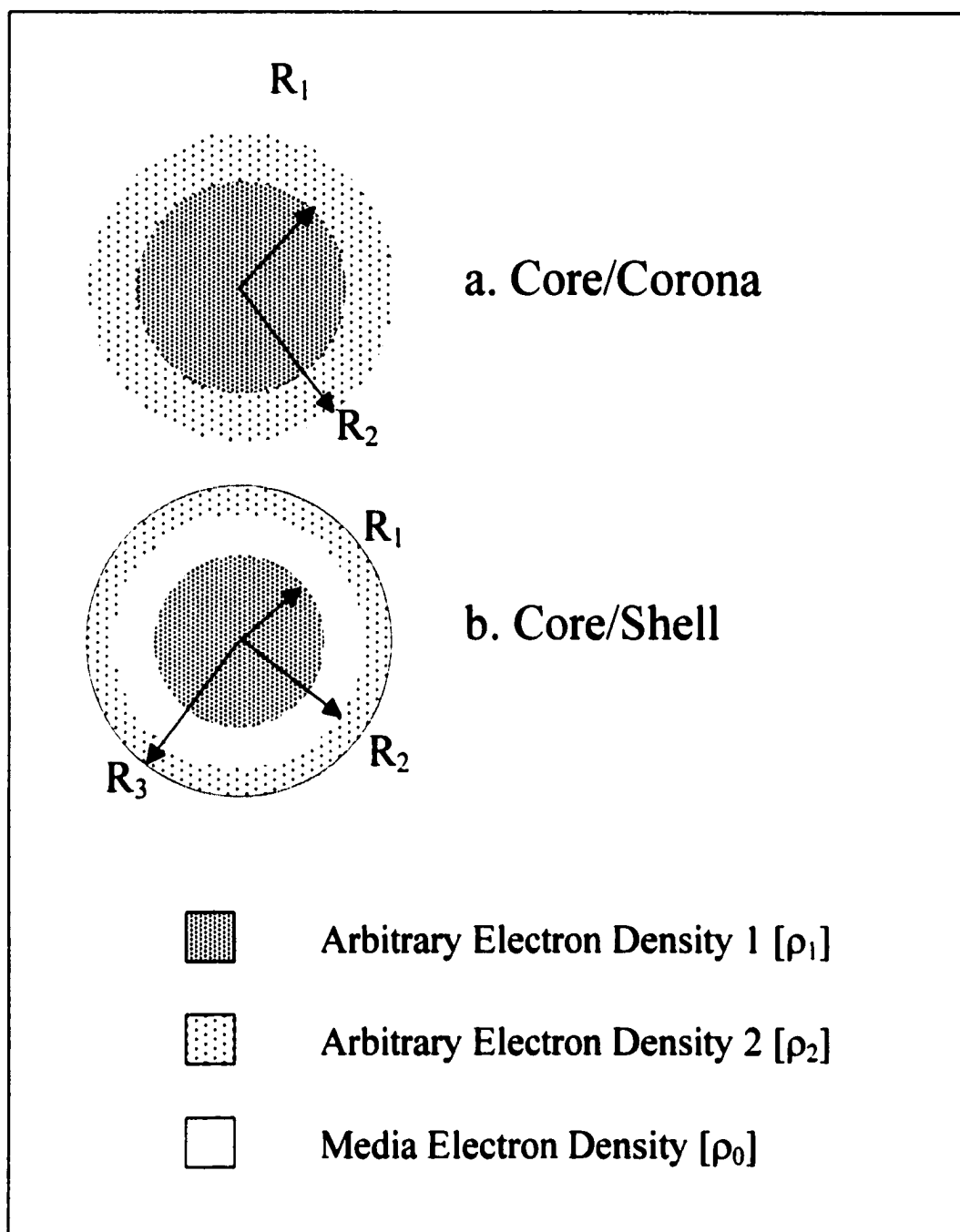


Figure 3.1. Visual interpretations: a. Core/Corona Sphere Model; b. Core/Shell Sphere Model; Centro-Symmetric Layered Cylinder.

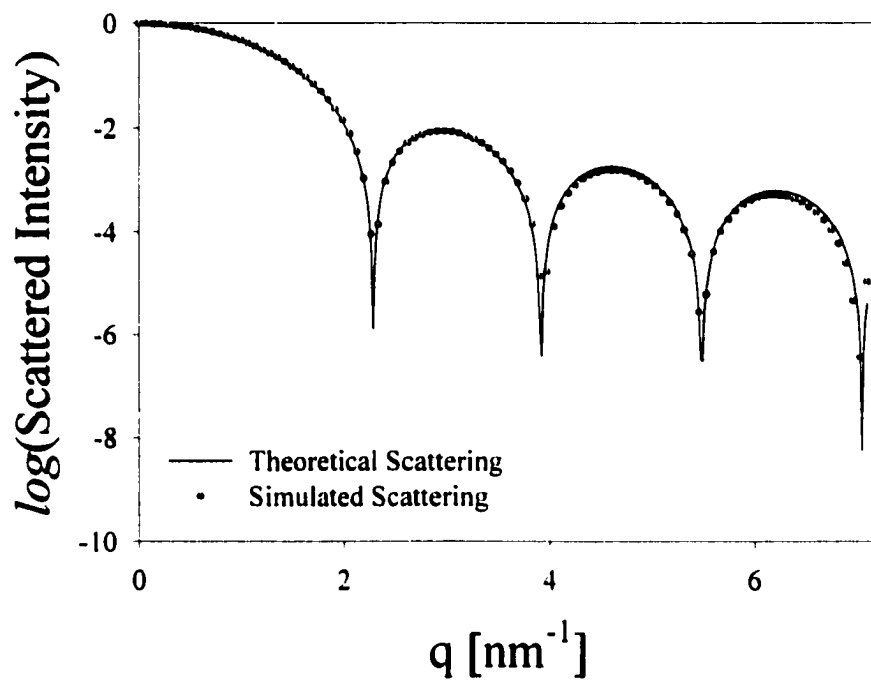


Figure 3.2. Core/Corona: $R_{\text{core}} = 2.0$ nm, $R_{\text{corona}} = 2.4$ nm, $\varpi = -0.1$: $R_L = 0.972$ 45 minute simulation run time.

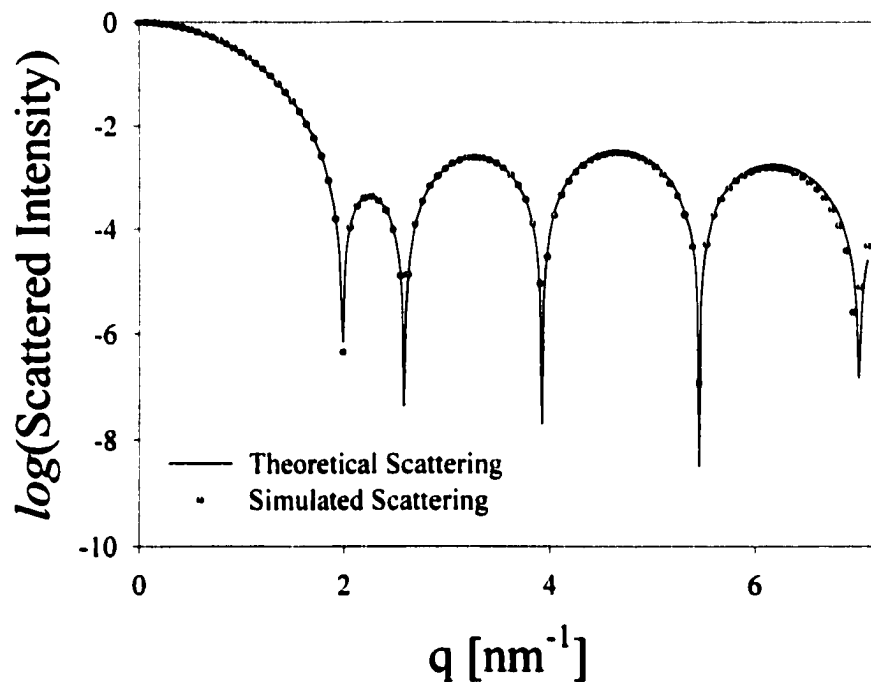


Figure 3.3. Core/Shell: $R_{\text{core}} = 1.5$ nm, $R_{\text{shell-inner}} = 2.0$ nm, $R_{\text{shell-outer}} = 2.5$ nm, $\varpi = 0.75$: $R_L = 0.980$, 30 minute simulation run time.

Cylindrical Model

Many dimensions must be defined when simulating scattering from a layered cylinder. The analytical solution for a “regular” cylinder requires the following information; the cylinder radius, R_c , the half-length of the cylinder, H , and the angle in the detector plane between the projection of the cylindrical axis and the projection of q . This angle is referred to as the “slice” and is given the symbol φ . In addition to these quantities, knowledge of the electron density ratio and the half-lengths of the primary and secondary phases are required to simulate scattering for a layered cylinder. Visual interpretations of these variables are given in Figure 3.4 except for φ which was given in Chapter 2 (“slice” angle in Figure 2.3).

The rest of this section is devoted to comparing different changes in morphology, and is designed to give the experimenter important insights into determining morphological characteristics if it is known that the scattering object is of a layered-cylinder type.

Case 1: Changing ϖ

In patterns of semicrystalline polymer fibers that are often described by this type of morphology, the electron density of the one phase (the crystalline phase) is typically known, while the electron density of the second phase and the matrix are often unknown. Scattering cannot be used to determine these two values independently, but allowing the latter to assume some value, this method can be used to differentiate the electron density of the second lamellar phase. Likewise, knowing

the electron density of the second lamellar phase will allow for determining the difference in the electron density between the crystalline and the matrix phase.

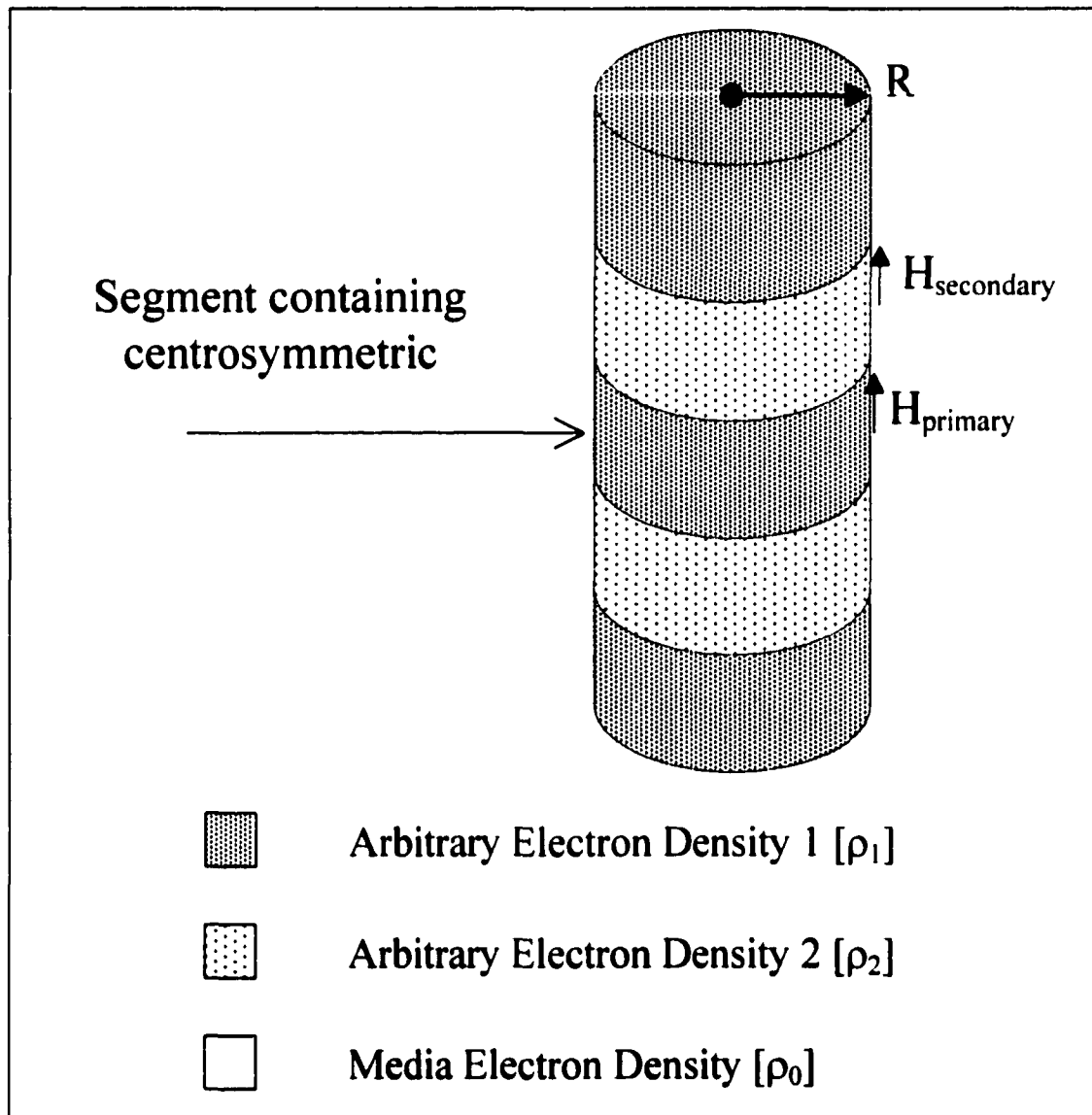


Figure 3.4. Diagram of cylindrical scattering object.

Figure 3.5 illustrates that the general shape of the normalized patterns (intensity at zero scattering angle is set to one) are similar at low q but show increasing variation as q increases. These variations along the scattering curves are caused by two different components; φ and ϖ . In scattering pattern variation caused by φ , variation is nearly absent when φ equals $\pi/2$ but becomes very pronounced as the φ approaches 0. Hence, if a difference in pattern shape at $\pi/2$ occurs between two samples, it is not simply a case of different lamellar electron density, some change in morphology must have occurred (different lamellar width or different arrangement of fibrils relative to one another). Similarly in scattering pattern variation caused by ϖ , the variations become more prominent as ϖ decreases, i.e. variation for negative ϖ values is substantially greater than for positive ϖ values. Absolute intensities scale with the difference in the relevant electron density differences as expressed by ϖ , i.e. as ϖ decreases absolute intensities decrease also.

Comparisons of Figure 3.5 and Figure 3.6 show that the introduction of "negative scattering" ($\varpi < 0$) i.e. the case where one lamellar phase has an electron density greater than the surrounding matrix and the other lamellar phase has an electron density less than the surrounding matrix, makes the scattering pattern extremely sensitive to the actual electron density differences. Hence, if what one thinks to be identical samples have very different scattering patterns, then this could be indicative of "negative scattering".

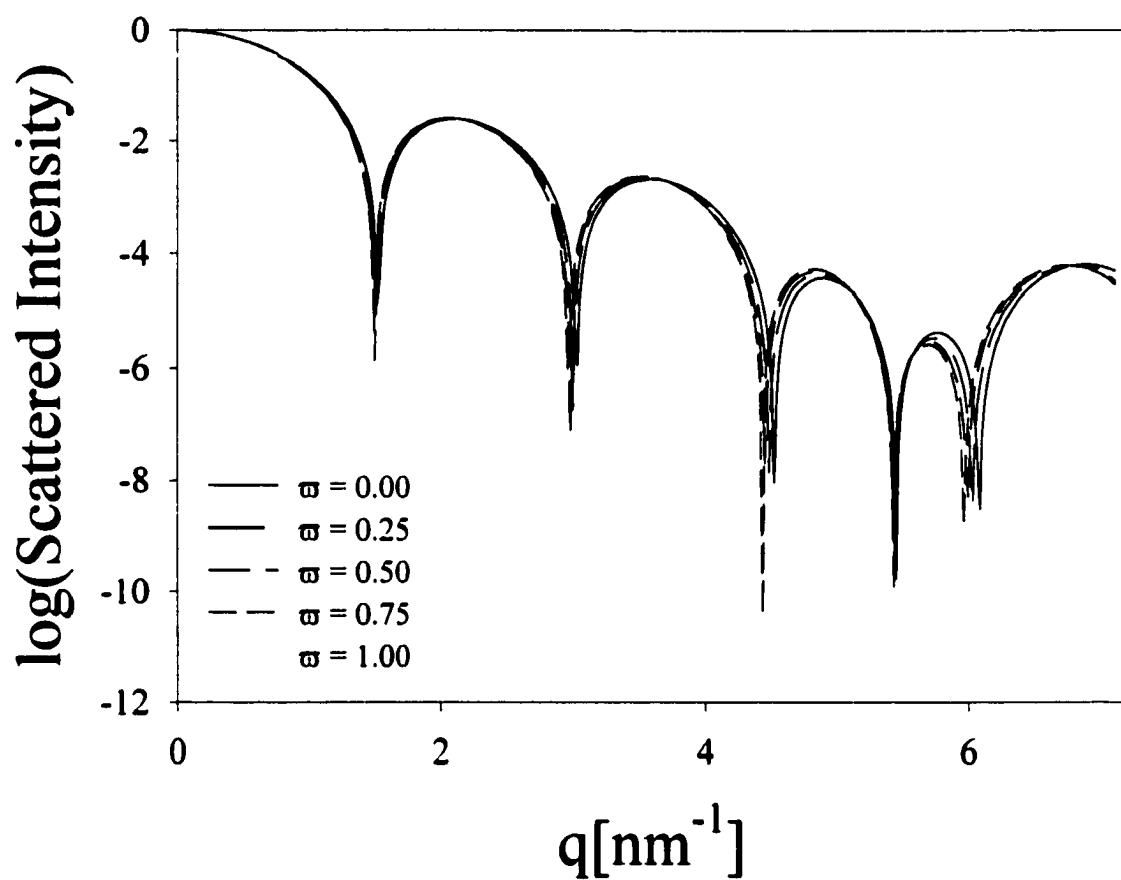


Figure 3.5. Layered Cylinder: $R_c = 1.0$ nm, $H = 3.0$ nm, $H_{\text{primary}} = 0.200$ nm, $H_{\text{secondary}} = 0.185$ nm, $\varphi = \pi/4$, $\varpi = 0.00, 0.25, 0.50, 0.75, 1.00$:
30 minute simulation run time for each curve.

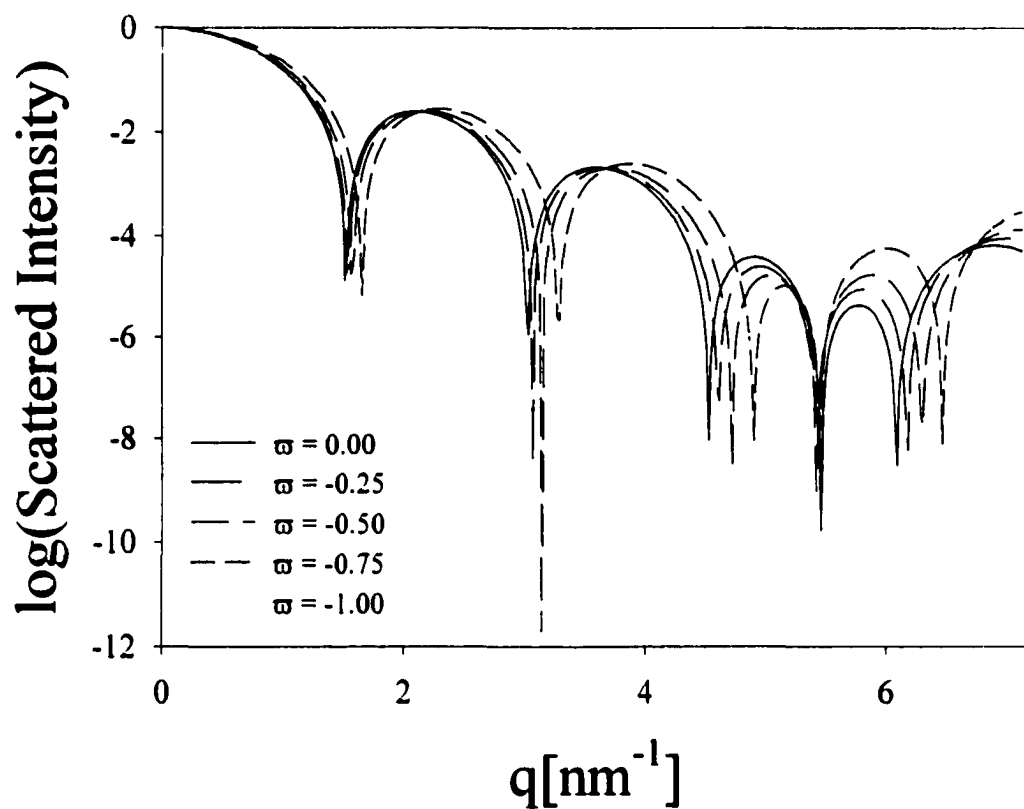


Figure 3.6. Layered Cylinder: $R_c = 1.0$ nm, $H = 3.0$ nm, $H_{\text{primary}} = 0.200$ nm, $H_{\text{secondary}} = 0.185$ nm, $\phi = \pi/4$, $\omega = 0.00, -0.25, -0.50, -0.75, -1.00$: 45 minute simulation run time for $\omega = -0.25, -0.50, -0.75, -1.00$; 30 minute simulation run time for $\omega = 0.00$ (extended run time because of “negative scattering”).

Case 2: Cylinder Radius

If scattering for a cylinder of unknown dimensions is being analyzed and the cylinders are aligned in the same direction, the scattering slice of $\varphi = \pi/2$ can be used to determine the radius of the cylinder by simply matching the curves because changes in electron density affect this curve only very slightly, as was stated previously.

Case 3: Distribution of Cylinder Orientations

To simulate a distribution of cylinder orientations, one can simply add the scattering patterns from individually simulated cylinders, since it is assumed that the cylinders are widely separated. This method allows one to choose a distribution of cylinder axes where each cylinder has a different tilt angle. In fact, one can also alter the tilt angle relative to the detector plane randomly as well, although all the simulations in this chapter were for cylinder axes parallel to the detector plane. Only highly oriented systems were considered, a range of distributions was developed from an orientation function equal to one ($f = 1.0$) to an orientation function equal to 0.923 ($f = 0.923$) as defined in Equation 3.5.

$$f = \frac{3\langle \cos^2 \theta \rangle - 1}{2} \quad \text{Equation 3.5}$$

When otherwise identical cylinders have different orientations, the result is similar to smearing; i.e. the position of the maximums and the poles do not change but the features become less abrupt. The changes that do occur are dependent on the distribution and φ . Of course, as the orientation function decreases the amount of

broadening increases. More surprising perhaps, broadening becomes more pronounced as φ approaches $\pi/2$ although some broadening still occurs at $\varphi = 0$.

Case 4: Changing Thickness

The parameter z was defined so that changes in lamellar thickness can be analyzed. This parameter is simply equal to the half-length of the secondary phase divided by the half-length of the primary phase. Although z can vary from zero to infinity, in the simulations z was varied from 0 to 8.0. Lamellar thicknesses were assumed to be identical, although using unequal lamellar thickness would not be a problem as long as centrosymmetry is maintained.

As can be seen in Figure 3.7 and Figure 3.8 the effect of changing the lamellar thickness of either phase can be seen to increase with increasing q . Further, changes in the scattering curves increase as φ approaches 0. As before, changes are also more pronounced with decreasing ω .

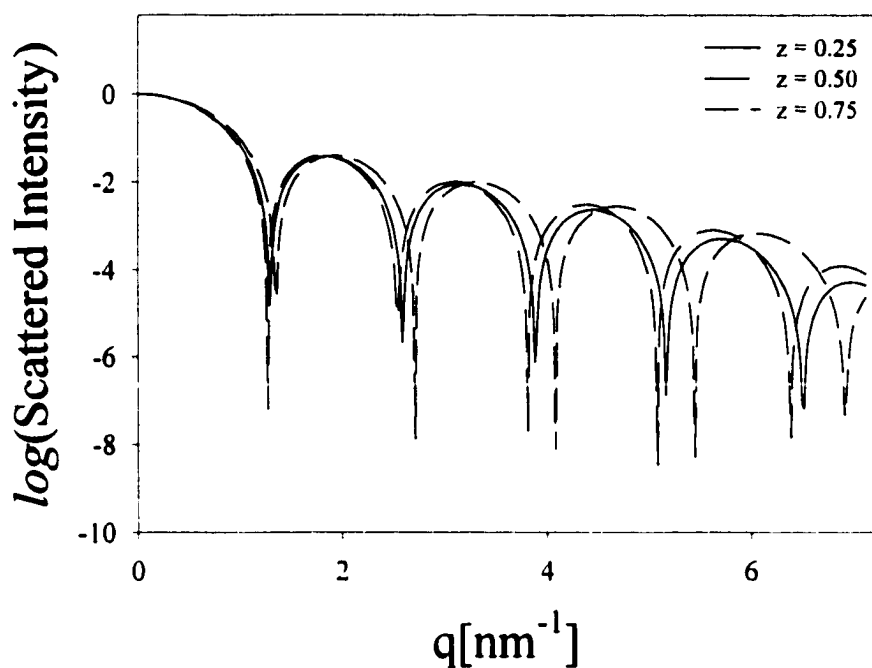


Figure 3.7. Layered Cylinder: $R_c = 1.2$ nm, $H = 2.6$ nm, $H_{\text{primary}} = 0.20$ nm, $H_{\text{secondary}} = 0.05$ nm ($z = 0.25$), 0.10 nm ($z = 0.50$), 0.15 nm ($z = 0.75$), $\phi = \pi/8$, $\varpi = 0.25$: 30 minute simulation run time for each curve.

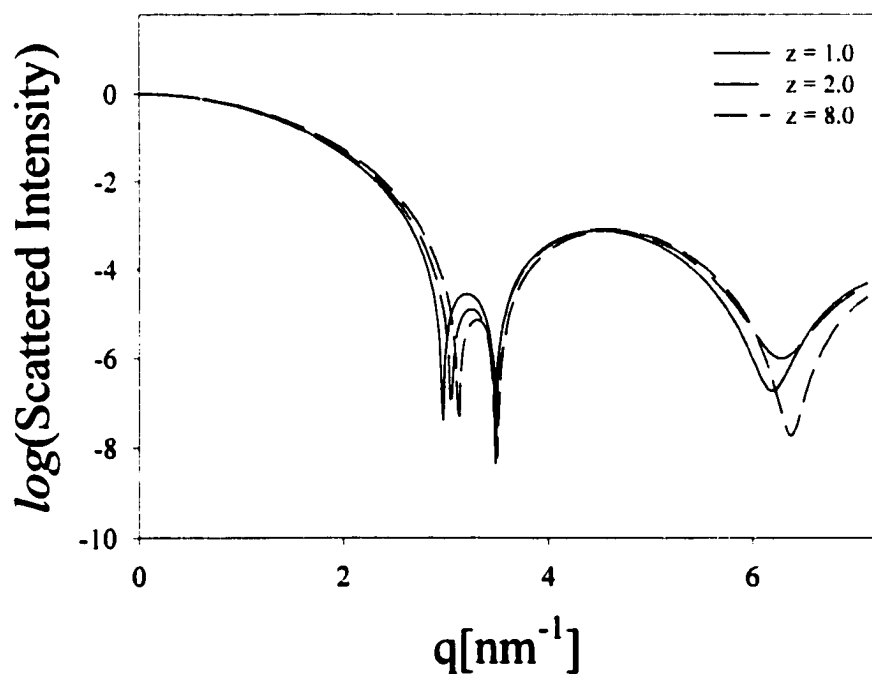


Figure 3.8. Layered Cylinder: $R_c = 1.2$ nm, $H_{\text{primary}} = 0.20$ nm ($z = 1.0$), 0.10 nm ($z = 2.0$), 0.025 nm ($z = 8.0$), $H_{\text{secondary}} = 0.20$ nm, $\phi = 3\pi/8$, $\varpi = 0.25$: 30 minute simulation run time for each curve.

3.4 References

- Bear, R. S. & Bolduan, O. E. A. (1950). *Acta Cryst* 3, 236
- Göttlicher, K., Fronk, W. & Wilke, W. (1983). *Coll & Polym Sci* 261, 126
- Hay, I. L. & Keller, A. (1967). *J Mat Sci* 2, 538
- MacKnight, W. J., Taggart, W. P. & Stein, R. S. (1974) *J Polym Sci, Polym Symp* 45, 113
- Murthy, N. S., Bednarczyk, C., Moore, R. A. F. & Grubb, D. T. (1996). *J Polym Sci B, Polym Phys* 34, 821
- Pope, D. P. & Keller, A (1975). *J Polym Sci, Polym Phys Ed* 13, 533
- Roche, E. J., Stein, R. S., Russell, T. P. & MacKnight, W. J. (1980) *J Polym Sci, Polym Phys Ed* 18, 1497
- Rule, R. J., MacKerron, D. H., Mahendrasingam, A., Martin, C. & Nye, T. M. W. (1995). *Macromolecules* 28, 8517
- Shibayama, M. & Hashimoto, T. (1986). *Macromolecules* 19, 740
- Stribeck, N. (1989) *Coll & Polym Sci* 267, 301
- Yarusso, D. J. & Cooper, S. L. (1983) *Macromolecules* 16, 1871

4.1 Introduction

The underlying purpose of this research is to basically “fill in the gaps” of the current SAXS modelling techniques. As an example, the Centro-Symmetric method can calculate non-rotationally averaged scattering curves, while the Pair-Distance and Correlation-Function methods cannot. Thus, this method can be used with oriented systems to give more structural information (no information is lost because of artificial rotational averaging of the experimental pattern). Also, unlike the Correlation Function and the Pair-Distance methods, this technique can also be used with morphologies with more than two electron densities as illustrated in Chapter 3. This intent to “fill in the gaps” should therefore be extended to multiple particle systems.

At the end of Chapter 3, multiple particle systems were introduced. The treatment of scattering from systems with more than one particle is very similar to treatment of scattering from single particles when the concentration of particles is low enough that inter-particle interference can be ignored. The scattering from a system of perfectly aligned identical particles or spheres with like radii is just the number of particles times the scattering from one particle. In this situation a normalized scattering curve would be identical to that of one particle. However, most “real” systems have a distribution of both sizes and alignments.

This complication significantly lengthens simulation time but does not really complicate simulating SAS for widely separated particles. Like the distribution of widely separated, uniaxially oriented cylinders given in Chapter 3, a random sample size (the larger the better) with the appropriate distribution(s) must be numerically generated. The scattering from each object (or rotationally averaged object) is calculated and then averaged. If the scattering pattern is anisotropic, then this procedure must be followed for the desired “slice” of the overall two-dimensional scattering pattern.

What can be done to simulate scattering patterns from systems where interparticle interference cannot be ignored? Models exist to compute the intensity in various angular regions, i.e. the Guinier model at low scattering angles and the Porod model at high scattering angles. These types of models will not be explored in this thesis; the interested reader is referred to the books by Guinier and Fournet (1955) and Glatter and Kratky (1982) for a complete description. This chapter only considers methods that take a given morphology, and calculate the scattering pattern using numerical techniques.

4.2 Dense Multiple Particle Systems

Inter-particle interference becomes a factor at moderate concentration and then dominates at high concentration. Simulating scattering from more densely concentrated particle systems is more difficult than simulating scattering from single particle systems because as the particle concentration increases, the positions of the

particles become less random. Thus, when the system of particles becomes dense (liquid like) the positions have very little randomness (Compton & Allison, 1935).

One equation to describe scattering from densely concentrated particle systems can be written if the scattering objects are spherically symmetric. Zernicke and Prins (Zernicke & Prins, 1927; Debye & Menke, 1930) developed an expression, which is shown in Equation 4.1, to calculate scattering patterns from an analytical expression if the radial distribution function $P(r)$ is known. $P(r)$ describes the average number of particles whose centers lie between the distance r and $r + dr$ from the center of a random particle. The number of particles is $4\pi r^2 P(r) dr$ (Guinier, 1963).

$$\frac{I(\underline{q})}{I_c(\underline{q})} = N[F_k(q, R)]^2 \left[1 + \frac{N}{V} \int_0^\infty (P(r) - 1) 4\pi r^2 \frac{\sin(qr)}{qr} dr \right] \quad \text{Equation 4.1}$$

Without an assumed sphere interaction model, the Zernicke and Prins Equation is limited to calculating the radial distribution function from scattering data as in the work of Gingrich (1943). The radial distribution function itself can be useful; however this approach holds little predictive abilities without a thermodynamic model for sphere interaction. In fact, a significant body of literature exists on the simulation of radial distribution functions given a thermodynamic model of sphere interaction; the application of these models to scattering problems is described in the paragraph below.

Percus-Yevick hard sphere interactions (Thiele, 1963; Wertheim, 1963) have been used effectively by researchers to describe scattering from multi-particle systems

(Hayter & Penfold, 1981; Kinning & Thomas, 1984; Pedersen, 1994; Pedersen & Gerstenberg, 1996; Bertram, 1996). Using the Percus-Yevick formulation, morphological parameters are determined from a SAS scattering pattern by simply fitting data to an analytical expression. Other spherical systems have also been modeled including permeable spheres and two component mixtures (Lebowitz, 1964; Blum & Stell, 1979; Blum & Stell, 1980; Salacuse & Stell, 1982). Systems that can also be accommodated are those with polydispersity of radii and systems that change from monodisperse radii to polydisperse radii (Vrij, 1979; Pedersen, 1993). However, the Zernicke and Prins Equation cannot be extended to systems of particles that are not spherical because the radial distribution function is a function of distance *and* orientation (Glatter & Kratky, 1982).

The Debye Equation (Equation 1.3) and the Zernicke and Prins Equation (Equation 4.1) are related as shown in Equation 4.2. The first term on the right hand side of Equation 4.2 is an “extra” term that does not appear in Equation 4.1. This term is the intensity from a particle with volume equal to the overall scattering volume and electron density equal to the average electron density in this scattering volume. The fact that Equation 4.1 does not contain this term does not limit the effectiveness of the Zernicke and Prins approach, since the resulting scattered intensity from this “particle volume” is effectively zero for all scattering angles because this “particle volume” is much larger than the particles of interest (Fournet, 1951; James, 1982). This term is critically important in understanding just how scattering from multi-particle systems is simulated as described below.

$$\sum_k^N \sum_j^N F_k(q, R) F_j(q, R) \frac{\sin(qR_{jk})}{(qR_{jk})} = \text{Equation 4.2}$$

$$\left[\langle F_i(q, R) \rangle \right]^2 \int_V \int_V \frac{\sin(qR_{jk})}{qR_{jk}} \frac{dv_j}{v_l} \frac{dv_k}{v_l} + N \left[\langle F_i(q, R) \rangle \right]^2 \left[1 + \frac{N}{V} \int_0^\infty (P(r) - 1) 4\pi r^2 \frac{\sin(qr)}{qr} dr \right]$$

Numerical simulation of systems with inter-particle interference using the Debye Equation is complicated not only by the fact that some sort of expression for interparticle distance and particle orientation must be generated using a thermodynamic model, but also by the fact that in some cases the particles move in relation to one another, i.e. in solution. These movements are by no means random, which means the local arrangements are also not random. In fact, it is not necessary for the particles to move in order for this problem to arise, since spatial variations are in this case no different than temporal variations since the characteristic dimension of the x-ray beam is usually orders of magnitude larger than the characteristic dimension of the scattering object. However, if the positions of the particles are fixed with respect to one another and the radial distribution function goes to something other than exactly 1 at $r=\infty$, then the Debye Equation can be used without any adjustments to numerically simulate the scattering pattern. In fact, if the radial distribution function goes to zero at infinite distance, i.e. the collection of small particles forms a large object with finite size, then either the Debye approach or any of the three approaches given earlier (the Correlation Function Method, the Pair-Distance Method and the Centro-Symmetric Method) can be used if the appropriate assumptions are met. If the

Debye approach is used, Equation 7 is particularly well suited, since many of these systems are in solution and require rotational averaging.

Equation 1.3 is rather easy to model but, because of the double integral, can be very time consuming if many particles are used. Some procedures can be used to substantially shorten computer run time (Glatter & Kratky, 1982; Perkins & Sims, 1986; Pantos & Bordas, 1994). Additionally, this technique is able to simulate scattering from morphologies with more than two electron densities (Pantos et al., 1996). Finally, the Debye Equation has been used to calculate a form factor that then was used with the Zernicke and Prins Equation to give the analytical scattering curve for high-density micelles (Oster & Riley, 1951).

Using the Debye Equation to simulate scattering from systems with random placement and orientations of particles is a difficult and time-consuming task. The first step is to place and orient the particles, which can be done using some random procedure according to a given thermodynamic interaction (hard sphere etc.). The fundamental problem with random systems is that an extremely large box is required to truly have a random system, i.e. all oscillations of $P(r)$ are eliminated. In other words, the placement of the first few particles will have a large influence on the final morphology. For obvious reasons, it is much more efficient to use many different simulation boxes to generate the scattering pattern than to use one large box. Even so, the numerical simulation time goes up at least 3-4 orders of magnitude versus that required for the case of non-random systems. Still however, some excellent examples

of using this type of procedure have been described in the literature (Sjöberg, 1999; Sjöberg & Mortensen, 1994; Sjöberg & Mortensen, 1997).

Simply stated, the approach of Sjöberg and Mortensen has the following steps. For some chosen volume, particles are randomly placed one at a time so that they are not overlapping. To account for any inhomogeneities near the volume surface, periodic boundary conditions are used (Metropolis et al., 1953; Wood & Parker, 1957). To accommodate for the many different arrangements of particles, Equation 1.5 is not used. Rather Equation 1.3, the Debye Equation is modified so it is no longer comprised of a double summation. The modification is simply done with the use of the cosine difference identity. The resulting equation is a function based on a single summation and is given in Equation 4.3. In this modified Debye Equation, \underline{R}_k is the vector from the center of the chosen scattering volume to the center of the k^{th} particle. Equation 4.3 is then calculated for one “slice” of the two-dimensional scattering pattern. The result from Equation 4.3 is then used with Equation 4.4 to give the rotationally averaged scattering curve for this configuration of particles. This process is repeated many times ($\sim 10^5$) so that many particle arrangements can be generated. Each time, the same number of particles, N , is selected in the scattering volume and the same “slice” of the three dimensional scattering curve is generated. The rotationally averaged scattering curves for each particle arrangement are then averaged to give a “total scattering” curve.

$$\frac{I(\underline{q})}{I_c(\underline{q})} = \left\{ \sum_k^N F_k(\underline{q}) \cos(\underline{q} \cdot \underline{R}_k) \right\}^2 + \left\{ \sum_k^N F_k(\underline{q}) \sin(\underline{q} \cdot \underline{R}_k) \right\}^2 \quad \text{Equation 4.3}$$

$$\frac{I(\underline{q})}{I_c(\underline{q})} = \frac{1}{4\pi} \int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} \left[\frac{I(\underline{q})}{I_c(\underline{q})} \right] d\phi \sin \theta d\theta \quad \text{Equation 4.4}$$

Since the box is of finite size, the “volume scattering” must be subtracted from the calculated “total scattering” curve (the average of all results from Equation 4.4). To make this correction, the same “slice” of scattering from a particle shaped like the original chosen scattering volume must be calculated. For example, if the chosen scattering volume is a cube with edge length X , the scattering from the same “slice” that was used with the volume of particles must be calculated for a cube with edge length X . This calculated “volume scattering” is then multiplied by the number of particles squared (N^2) and then subtracted from the “total scattering” to give the “corrected scattering” curve. This “corrected scattering” curve is usually then normalized by N , the number of particles.

The result at this point is a corrected scattering curve that has an intensity of zero at $\underline{q} = 0$ because the intensity of the “total scattering” at $\underline{q} = 0$ is N^2 while the intensity of the “volume scattering” at $\underline{q} = 0$ is one, $N^2 - (N^2 * 1) = 0$. This result is not correct and the reason represents a fundamental limitation of numerically simulated patterns. In all cases, scattering is calculated from morphologies where the positions of individual particles are rigidly fixed. In real systems, this is not true except for perfect crystals at absolute zero. Hence, Sjöberg and Mortensen (1994) have recalculated the scattering at $\underline{q} = 0$ with a known isothermal compressibility to give a more correct value. However, at best this result is approximate since the simulation

itself is only valid for values of $q > 2\pi/L$, where L is the characteristic scattering length, i.e. the edge of the cube.

One significant complication not considered by these authors is the closer packing of particles. This problem significantly complicates the initial placement of particles. Numerical experiments in our lab show that the maximum packing density that can be achieved with random placement of identically-sized hard spheres is approximately 0.40, which is far below what is commonly encountered in many densely packed systems. One simple technique which has been used successfully in our lab to achieve a higher packing density is to start with a perfectly ordered set of spheres (i.e. BCC or FCC), slightly expand the box used to achieve a given packing density, and then move the particles randomly. After sufficient movement time, a scattering pattern can be calculated. This type of approach is also ideal for multiple simulations, since one can simply move the particles again for long enough to erase the memory of the previous arrangement, and recalculate the scattering patterns.

Code was written to perform simulations, i.e. scattering from densely packed spheres. The difficulty in this “motion” approach is that at high concentrations, particle movements require much computer run time because of the very small step size required. Thus, sufficient movements to erase similarities to previously chosen particle arrangements took several days on a personal computer. Since approximately 10^5 simulations would be necessary to obtain proper statistics, no results are presented because the simulation time was much too long. The process to obtain non-identical random distributions could probably be refined. However we don't believe the time

could be reduced by a factor of approximately 10^7 , which is what is required to calculate the scattering pattern in a reasonable time.

Rotationally averaged scattering curves are not always desired. For instance, consider a system such as several fibers. In essence this is a close packed system of many cylinders (fibrils). A scattering experiment with these fibers stretched between two fixed points would obviously give a very anisotropic scattering curve.

To calculate the scattering from oriented systems, the technique just reviewed should be used with a few modifications. Each time the particles are placed in the scattering volume, their orientation will be approximately known. In the fiber example, this means that the cylinders might have a Gaussian distribution along the axis of orientation. Equation 4.3 could then be used to calculate the scattering for a single “slice” of the two-dimensional scattering curve for this arrangement of particles. Obviously, Equation 4.4 would be ignored. After repeating the particle movement/scattering “slice” calculation process many times ($\sim 10^5$), the scattering would be averaged. The result is one “slice” of the anisotropic, two-dimensional scattering curve. The intensity of the “volume scattering” would still have to be subtracted to yield the scattering curves. To generate the whole pattern this entire process would have to be repeated for many “slices” (many different scattering vector arrays would have to be used). However, the difficulties described for densely-packed spheres become even more important, because the concentration where objects cannot be randomly placed is much lower for anisotropic objects than for spheres.

4.3 References

- Blum, L. & Stell, G. (1979). *J Chem Phys* 71, 42
- Blum, L. & Stell, G. (1980). *J Chem Phys* 72, 2212
- Bertram, W. K. (1996). *J Appl Cryst* 29, 682
- Compton, A. H. & Allison, S. K. (1935). *X-Rays in Theory and Experiment*. New Jersey: D. Van Nostrand Company, Inc.
- Debye, P. & Menke, H. (1930). *Physik Z* 31, 419
- Fournet, G. (1951). *Bull Soc Franç Minéral Et Crist* 74, 39
- Gingrich, N. S. (1943). *Rev Mod Phys* 15, 90
- Glatter, O. & Kratky, O. (1982). Editors. *Small-Angle X-Ray Scattering*. New York: Academic Press
- Guinier, A. (1963). *X-Ray Diffraction in Crystals, Imperfect Crystals, and Amorphous Bodies*. San Francisco: W. H. Freeman and Company
- Guinier, A. & Fournet, G. (1955). *Small-Angle Scattering Of X-Rays*. New York: John Wiley & Sons, Inc.
- Hayter, J. B. & Penfold, J. (1981). *Molec Phys* 42, 109
- James, R. W. (1982). *The Optical Principles Of The Diffraction Of X-Rays*. Connecticut: Ox Bow Press
- Kinning, D. J. & Thomas, E. L. (1984). *Macromolecules* 17, 1712
- Lebowitz, J. L. (1964). *Phys Rev* 133, A895
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. (1953). *J Chem Phys* 21, 1087
- Oster, G. & Riley, D. P. (1951). *Acta Cryst* 5, 1
- Pantos, E. & Bordas, J. (1994). *Pure Appl Chem* 66, 77
- Pantos, E., van Garderen, H. F., Hilbers, P. A.J., Beelen, T. P.M. & van Santen, R. A. (1996). *J Molec Struct* 383, 303

- Pedersen, J. S. (1993). *Phys Rev B* 47, 657
- Pedersen, J. S. (1994). *J Appl Cryst* 27, 595
- Pedersen, J. S. & Gerstenberg, M. C. (1996). *Macromolecules* 29, 1363
- Perkins, S. J. & Sims, R. B. (1986). *Eur J Biochem* 157, 155
- Salacuse, J. J. & Stell, G. (1982). *J Chem Phys* 77, 3714
- Sjöberg, B. (1999). *J Appl Cryst* 32, 917
- Sjöberg, B. & Mortensen, K. (1994). *Biophys Chem* 52, 131
- Sjöberg, B. & Mortensen, K. (1997). *Biophys Chem* 65, 75
- Thiele, E. (1963). *J Chem Phys* 39, 474
- Vrij, A. (1979). *J Chem Phys* 71, 3267
- Wertheim, M. S. (1963). *Phys Rev Lett* 10, 321
- Wood, W. W. & Parker, F. R. (1957). *J Chem Phys* 27, 720
- Zernicke, F. & Prins, J. A. (1927). *Z Physik* 41, 184

5.1 Single Particle Systems

The Centro-Symmetric method is a very powerful SAXS simulation tool for single particle systems. Unlike other methods, the centro-symmetric method has the ability to simulate scattering from oddly-shaped or oddly-oriented morphologies. Additionally, relatively short computer run times of approximately 30 minutes on a standard personal computer are needed to fully calculate an isotropic scattering pattern from one spherically symmetric object. If the particle is not spherically symmetric, then the simulation time will be governed by the number of “slices” required; a day would be required to simulate a pattern in 2° increments. If a system has particles with a distribution of sizes and or orientations, longer times are required; still though an entire scattering pattern can be accurately simulated from these types of systems in a few weeks at most.

Bi-continuous morphologies were mentioned in the text, but no simulations were performed. Many different types of materials show bi-continuous morphologies, including surfactants and block copolymers. Block copolymers exhibit bi-continuous morphologies like the double diamond (cubic phase Q_{224}), gyroid* (cubic phase Q_{230}) and possibly the Schoen surface (cubic phase Q_{229}) which has not yet been observed in copolymer systems but has in surfactant systems (Bates &

Fredrickson, 1999; Benedicto & O'Brien, 1997). These systems are difficult to classify (Hajduk et al., 1995) and pose a challenge to SAXS simulations because of their unusual repeat structures. For example, Figure 5.1 is one "cell" of a double diamond surface. In a phase separated copolymer system, this surface could represent the interface between the two phases. Looking at it on a larger scale, this "cell" would repeat three dimensionally as shown in Figure 5.2. Thus, the question in simulating SAXS from these types of systems is how many "cells" are needed to correctly model the morphology.

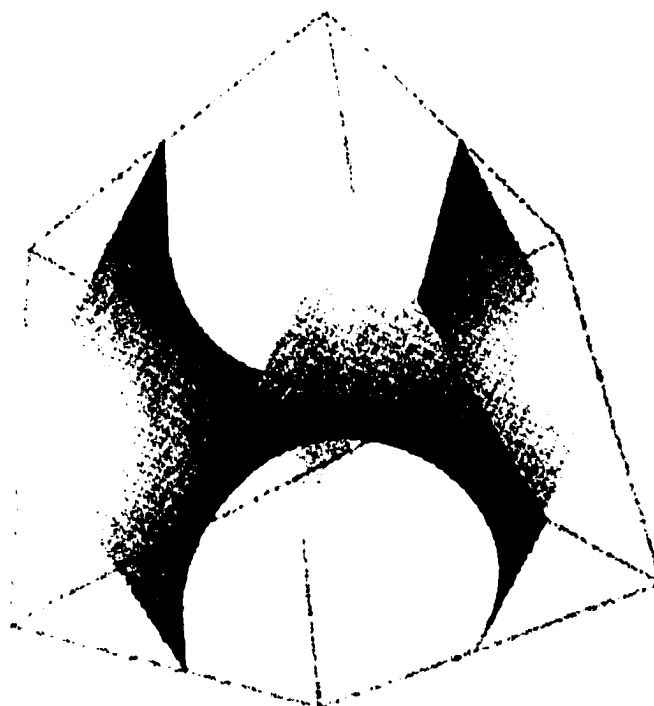


Figure 5.1. One "cell" of the double diamond surface.

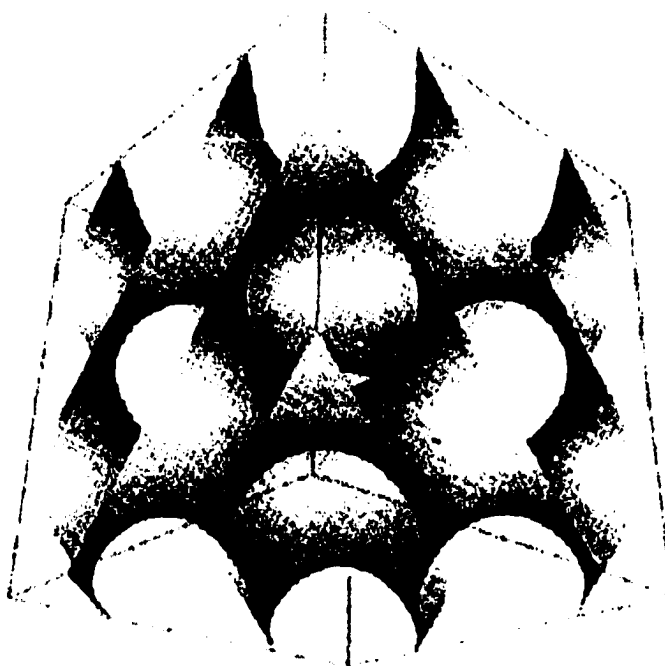


Figure 5.2. Three-dimensional double diamond array.

5.2 Multiple Particle Systems

Obviously the largest shortcoming in this research is the current inability to simulate dense multiple particle systems. However, the future might remedy this situation. Personal computers should have the ability to simulate SAXS from dense multiple particle simulations efficiently in 10 to 15 years.

5.3 Analyzing Experimental Data

The overall goal of this research is to develop a new method to analyze experimental data based on the approach given in this thesis. The approach given in this thesis is missing two steps, one of which is trivial. As was pointed out at the end of Chapter 1, smearing was not included in the simulations. Using simulated SAXS data to analyze “real” scattering data demands that smearing be included. Either the method of Lake (Lake, 1967) or Glatter’s enhanced version of Lake’s method (Glatter, 1974) is a simple way to incorporate smearing into these routines.

The second non-trivial step would be to develop a least-squares fitting routine to fit simulated data to real experimental data. Except in the simplest of systems, $I(q)$ is almost certainly a very complicated function of morphological variables. Hence, finding the absolute minimum rather than a relative minimum would be non-trivial and would almost certainly require some randomly directed search algorithm. Further, even the single-particle approach is too long to implement on a PC, because of the multiple simulations involved. However, on today’s supercomputers, one could write code to fit real data using Monte-Carlo simulations of the scattering pattern from morphologies with no interparticle interference, especially if the random number routine were adjusted. If the speed of PC’s continues to double every 18 months, then this code could probably run on a desktop computer in 5-10 years. However, to fit experimental data from systems where $P(r) \rightarrow 1$ at $r \rightarrow \infty$ using Monte-Carlo techniques will probably not be possible for many years.

5.4 References

Bates, F. S. & Fredrickson, G. H. (1999). *Phys Today* 52-2, 32

Benedicto, A. D. & O'Brien, D. F. (1997). *Macromolecules* 30, 3395

Glatter, O. (1974). *J Appl Cryst* 7, 147

Hajduk, D. A., Harper, P. E., Gruner, S. M., Honeker, C. C., Thomas, E. L. & Fetters, L. J. (1995). *Macromolecules* 28, 2570

Lake, J. A. (1967). *Acta Cryst* 23, 191

APPENDIX A

Sample Random Number Generators

A.1. Sample Random Number Function 1

```
FUNCTION ran0(idum)
  INTEGER idum,IA,IM,IQ,IR,MASK
  REAL ran0,AM
  PARAMETER (IA=16807,IM=2147483647,AM=1./IM,IQ=127773,IR=2836,
    *MASK=123459876)
  INTEGER k
  idum=ieor(idum,MASK)
  k=idum/IQ
  idum=IA*(idum-k*IQ)-IR*k
  if (idum.lt.0) idum=idum+IM
  ran0=AM*idum
  idum=ieor(idum,MASK)
  return
END
```

A.2. Sample Random Number Function 2

```
FUNCTION ran1(idum)
  INTEGER idum,IA,IM,IQ,IR,NTAB,NDIV
  REAL ran1,AM,EPS,RNMX
  PARAMETER (IA=16807,IM=2147483647,AM=1./IM,IQ=127773,IR=2836,
*NTAB=32,NDIV=1+(IM-1)/NTAB,EPS=1.2e-7,RNMX=1.-EPS)
  INTEGER j,k,iv(NTAB),iy
  SAVE iv,iy
  DATA iv /NTAB*0/, iy /0/
  if (idum.le.0.or.iy.eq.0) then
    idum=max(-idum,1)
    do 11 j=NTAB+8,1,-1
      k=idum/IQ
      idum=IA*(idum-k*IQ)-IR*k
      if (idum.lt.0) idum=idum+IM
      if (j.le.NTAB) iv(j)=idum
11 continue
    iy=iv(1)
    endif
    k=idum/IQ
    idum=IA*(idum-k*IQ)-IR*k
    if (idum.lt.0) idum=idum+IM
    j=1+iy/NDIV
    iy=iv(j)
    iv(j)=idum
    ran1=min(AM*iy,RNMX)
  return
END
```

A.3. Sample Random Number Function 3

```
FUNCTION ran2(idum)
  INTEGER idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
  REAL ran2,AM,EPS,RNMX
  PARAMETER (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
*IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,IR2=3791,
*NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2e-7,RNMX=1.-EPS)
  INTEGER idum2,j,k,iv(NTAB),iy
  SAVE iv,iy,idum2
  DATA idum2/123456789/, iv/NTAB*0/, iy/0/
  if (idum.le.0) then
    idum=max(-idum,1)
    idum2=idum
    do 11 j=NTAB+8,1,-1
      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      if (j.le.NTAB) iv(j)=idum
11 continue
    iy=iv(1)
    endif
    k=idum/IQ1
    idum=IA1*(idum-k*IQ1)-k*IR1
    if (idum.lt.0) idum=idum+IM1
    k=idum2/IQ2
    idum2=IA2*(idum2-k*IQ2)-k*IR2
    if (idum2.lt.0) idum2=idum2+IM2
    j=1+iy/NDIV
    iy=iv(j)-idum2
    iv(j)=idum
    if(iy.lt.1)iy=iy+IMM1
    ran2=min(AM*iy,RNMX)
  return
END
```

A.4. Sample Random Number Function 4

```
FUNCTION ran3(idum)
  INTEGER idum
  INTEGER MBIG,MSEED,MZ
  REAL ran3,FAC
  PARAMETER (MBIG=1000000000,MSEED=161803398,MZ=0,FAC=1./MBIG)
  INTEGER i,iff,ii,inext,inextp,k
  INTEGER mj,mk,ma(55)
  SAVE iff,inext,inextp,ma
  DATA iff /0/
  if(idum.lt.0.or.iff.eq.0)then
    iff=1
    mj=MSEED-iabs(idum)
    mj=mod(mj,MBIG)
    ma(55)=mj
    mk=1
    do 11 i=1,54
      ii=mod(21*i,55)
      ma(ii)=mk
      mk=mj-mk
      if(mk.lt.MZ)mk=mk+MBIG
      mj=ma(ii)
11  continue
    do 13 k=1,4
      do 12 i=1,55
        ma(i)=ma(i)-ma(1+mod(i+30,55))
        if(ma(i).lt.MZ)ma(i)=ma(i)+MBIG
12  continue
13  continue
    inext=0
    inextp=31
    idum=1
    endif
    inext=inext+1
    if(inext.eq.56)inext=1
    inextp=inextp+1
    if(inextp.eq.56)inextp=1
    mj=ma(inext)-ma(inextp)
    if(mj.lt.MZ)mj=mj+MBIG
    ma(inext)=mj
    ran3=mj*FAC
    return
  END
```


APPENDIX B

FORTRAN Programs Used in Chapter 2

B.1. Sphere

CC This file will model a sphere through random numbers.

```
      use portlib
      real*8 s, sl, timespent
      Integer*4 iter, l, number
      Integer*4 m, n, count, seed
      Integer ia1, ia2, iat, ib1, ib2, ibt,
+      ic1, ic2, ict, id1, id2, idt,
+      ie1, ie2, iet, if1, if2, ift,
+      ja1, ja2, jat, jb1, jb2, jbt,
+      jc1, jc2, jct, jd1, jd2, jdt,
+      je1, je2, jet, jf1, jf2, jft,
+      ka1, ka2, kat, kb1, kb2, kbt,
+      kc1, kc2, kct, kd1, kd2, kdt,
+      ke1, ke2, ket, kf1, kf2, kft
      Real*8 OR, A(1,401), Pi, lamda, Qi, dev,
+      Qk, Bragg, zeta, final, B(1,118), ran2,
+      x1, x2, x3, x4, x5, x6, y1, y2, y3, y4,
+      y5, y6, z1, z2, z3, z4, z5, z6, R1, R2,
+      R3, R4, R5, R6
```

CC Initilization of some of the variables and the constants.

```
      Do 1 n=0,400
      A(1,n)=0.0
1      Continue
      Pi=3.1415927
      lamda=.154242
      Bragg=2*Pi/lamda
      count=0
```

CC Input statements.

```
      Print*, 'Sphere Scattering Simulation Through Random Numbers.'
      Print*, 'Enter the overall radius of the sphere [nm].'
      Read*, OR
      Print*, 'Enter the number of iterations desired.'
      Read*, iter
      Print*, 'Enter a seed for the random numbers.'
      Read*, seed
```

```

s=rtc()
CC Iteration Loop
  Do 10 l=1,iter
    number=seed+l

CC Draw Random Numbers
  Do 2 m=0,117
    B(1,m)=ran2(number)
  2 continue

  ia1=idint(10*B(1,100))
  ia2=idint((100*B(1,100))-(10*ia1))
  iat=10*ia2+ia1

  ib1=idint(10*B(1,101))
  ib2=idint((100*B(1,101))-(10*ib1))
  ibt=10*ib2+ib1

  ic1=idint(10*B(1,102))
  ic2=idint((100*B(1,102))-(10*ic1))
  ict=10*ic2+ic1

  id1=idint(10*B(1,103))
  id2=idint((100*B(1,103))-(10*id1))
  idt=10*id2+id1

  ie1=idint(10*B(1,104))
  ie2=idint((100*B(1,104))-(10*ie1))
  iet=10*ie2+ie1

  if1=idint(10*B(1,105))
  if2=idint((100*B(1,105))-(10*if1))
  ift=10*if2+if1

  ja1=idint(10*B(1,106))
  ja2=idint((100*B(1,106))-(10*ja1))
  jat=10*ja2+ja1

  jb1=idint(10*B(1,107))
  jb2=idint((100*B(1,107))-(10*jb1))
  jbt=10*jb2+jb1

  jc1=idint(10*B(1,108))
  jc2=idint((100*B(1,108))-(10*jc1))

```

jct=10*jc2+jc1

jd1=idint(10*B(1,109))
jd2=idint((100*B(1,109))-(10*jd1))
jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1

jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1

ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1

kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
kct=10*kc2+kc1

kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1

ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1

kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1

x1=2.00002*OR*B(1,iat)-1.00001*OR
x2=2.00002*OR*B(1,ibt)-1.00001*OR
x3=2.00002*OR*B(1,ict)-1.00001*OR
x4=2.00002*OR*B(1,idt)-1.00001*OR
x5=2.00002*OR*B(1,iet)-1.00001*OR
x6=2.00002*OR*B(1,ift)-1.00001*OR

```

y1=2.00002*OR*B(1,jat)-1.00001*OR
y2=2.00002*OR*B(1,jbt)-1.00001*OR
y3=2.00002*OR*B(1,jct)-1.00001*OR
y4=2.00002*OR*B(1,jdt)-1.00001*OR
y5=2.00002*OR*B(1,jet)-1.00001*OR
y6=2.00002*OR*B(1,jft)-1.00001*OR
z1=2.00002*OR*B(1,kat)-1.00001*OR
z2=2.00002*OR*B(1,kbt)-1.00001*OR
z3=2.00002*OR*B(1,kct)-1.00001*OR
z4=2.00002*OR*B(1,kdt)-1.00001*OR
z5=2.00002*OR*B(1,ket)-1.00001*OR
z6=2.00002*OR*B(1,kft)-1.00001*OR

```

```

R1=dsqrt((x1**2)+(y1**2)+(z1**2))
R2=dsqrt((x2**2)+(y2**2)+(z2**2))
R3=dsqrt((x3**2)+(y3**2)+(z3**2))
R4=dsqrt((x4**2)+(y4**2)+(z4**2))
R5=dsqrt((x5**2)+(y5**2)+(z5**2))
R6=dsqrt((x6**2)+(y6**2)+(z6**2))

```

CC Test points

```

    if (R1.LE.OR) then
      Do 3 n=0,400
        zeta=Pi*n/(36*400)
        Qi=Bragg*(dcos(2*zeta)-1)
        Qk=Bragg*dsin(2*zeta)
        A(1,n)=A(1,n)+2*dcos(Qi*x1+Qk*z1)
3      continue
        count=count+1
      endif

      if (R2.LE.OR) then
        Do 4 n=0,400
          zeta=Pi*n/(36*400)
          Qi=Bragg*(dcos(2*zeta)-1)
          Qk=Bragg*dsin(2*zeta)
          A(1,n)=A(1,n)+2*dcos(Qi*x2+Qk*z2)
4        continue
          count=count+1
        endif

```

```

if (R3.LE.OR) then
Do 5 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x3+Qk*z3)
5  continue
count=count+1
endif

if (R4.LE.OR) then
Do 6 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x4+Qk*z4)
6  continue
count=count+1
endif

if (R5.LE.OR) then
Do 7 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x5+Qk*z5)
7  continue
count=count+1
endif

if (R6.LE.OR) then
Do 8 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x6+Qk*z6)
8  continue
count=count+1
endif

10 Continue

print*, '-----'
print*, 'Radius'

```

```

print*, OR
print*, '-----'
print*, 'Iterations'
print*, iter
print*, '-----'
print*, 'Number of Points'
print*, count
print*, '-----'

final=A(1,0)

Do 101 n=0,400
  A(1,n)=dlog10((A(1,n)/final)**2)
101 Continue

  sl=rtc()
  timespent=sl-s
  dev=(dble(iter))/timespent
  print*, 'CPU Time (seconds)'
  print*, timespent
  print*, 'Iterations Per Seconds'
  print*, dev

  Do 102 n=0,400
  write (15,*) A(1,n)
102 Continue

  dev=1
End

```

C

Random Number Function Goes Here

B.2. Prolate Spheroid

CC This file will model spheroidal scattering through random numbers.

```
use portlib
real*8 s, sl, timespent
Integer*4 iter, l, number
Integer*4 m, n, count, seed
Integer ial, ia2, iat, ib1, ib2, ibt,
+ ic1, ic2, ict, id1, id2, idt,
+ ie1, ie2, iet, if1, if2, ift,
+ jal, ja2, jat, jbl, jb2, jbt,
+ jcl, jc2, jct, jdl, jd2, jdt,
+ jel, je2, jet, jf1, jf2, jft,
+ kal, ka2, kat, kbl, kb2, kbt,
+ kcl, kc2, kct, kd1, kd2, kdt,
+ kel, ke2, ket, kfl, kf2, kft
Real*8 OR, A(7,401), Pi, lamda, Qi, dev, length,
+ Qj, Qk, Bragg, zeta, B(1,118), ran2, cut1, cut2,
+ x1, x2, x3, x4, x5, x6, y1, y2, y3, y4, y5,
+ y6, z1, z2, z3, z4, z5, z6, R1, R2, R3, R4,
+ R5, R6, final
```

CC Initilization of some of the variables and the constants.

```
Do 1 n=0,400
  A(1,n)=0.0
1 Continue
Pi=3.1415927
lamda=.154242
Bragg=2*Pi/lamda
count=0
```

CC Input statements.

```
Print*, 'Prolate Spheroid Scattering Simulation(Random Numbers.)'
Print*, 'Enter A [nm] (half of the major axis).'
Read*, length
Print*, 'Enter B [nm] (half of the minor axis).'
Read*, OR
Print*, 'Enter the slice desired of the cylinder [nm].'
Print*, '(Pi/3 times this number.)'
Read*, cut1
Print*, 'Enter the number of iterations desired.'
Read*, iter
Print*, 'Enter a seed for the random numbers.'
Read*, seed
```

```

        cut2=cut1*Pi/3.0
        s=rtc()
CC Iteration Loop
        Do 10 l=1,iter
            number=seed+l

CC Draw Random Numbers
        Do 2 m=0,117
            B(1,m)=ran2(number)
2        continue

        ia1=idint(10*B(1,100))
        ia2=idint((100*B(1,100))-(10*ia1))
        iat=10*ia2+ia1

        ib1=idint(10*B(1,101))
        ib2=idint((100*B(1,101))-(10*ib1))
        ibt=10*ib2+ib1

        ic1=idint(10*B(1,102))
        ic2=idint((100*B(1,102))-(10*ic1))
        ict=10*ic2+ic1

        id1=idint(10*B(1,103))
        id2=idint((100*B(1,103))-(10*id1))
        idt=10*id2+id1

        ie1=idint(10*B(1,104))
        ie2=idint((100*B(1,104))-(10*ie1))
        iet=10*ie2+ie1

        if1=idint(10*B(1,105))
        if2=idint((100*B(1,105))-(10*if1))
        ift=10*if2+if1

        ja1=idint(10*B(1,106))
        ja2=idint((100*B(1,106))-(10*ja1))
        jat=10*ja2+ja1

        jb1=idint(10*B(1,107))
        jb2=idint((100*B(1,107))-(10*jb1))
        jbt=10*jb2+jb1

```



```

jc1=idint(10*B(1,108))
jc2=idint((100*B(1,108))-(10*jc1))
jct=10*jc2+jc1

jd1=idint(10*B(1,109))
jd2=idint((100*B(1,109))-(10*jd1))
jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1

jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1

ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1

kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
ket=10*kc2+kc1

kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1

ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1

kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1

x1=2.00002*OR*B(1,iat)-1.00001*OR
x2=2.00002*OR*B(1,ibt)-1.00001*OR
x3=2.00002*OR*B(1,ict)-1.00001*OR
x4=2.00002*OR*B(1,idt)-1.00001*OR

```

```

x5=2.00002*OR*B(1,iet)-1.00001*OR
x6=2.00002*OR*B(1,ift)-1.00001*OR
y1=2.00002*length*B(1,jat)-1.00001*length
y2=2.00002*length*B(1,jbt)-1.00001*length
y3=2.00002*length*B(1,jct)-1.00001*length
y4=2.00002*length*B(1,jdt)-1.00001*length
y5=2.00002*length*B(1,jet)-1.00001*length
y6=2.00002*length*B(1,jft)-1.00001*length
z1=2.00002*OR*B(1,kat)-1.00001*OR
z2=2.00002*OR*B(1,kbt)-1.00001*OR
z3=2.00002*OR*B(1,kct)-1.00001*OR
z4=2.00002*OR*B(1,kdt)-1.00001*OR
z5=2.00002*OR*B(1,ket)-1.00001*OR
z6=2.00002*OR*B(1,kft)-1.00001*OR

```

```

R1=((x1/OR)**2)+((y1/length)**2)+((z1/OR)**2)
R2=((x2/OR)**2)+((y2/length)**2)+((z2/OR)**2)
R3=((x3/OR)**2)+((y3/length)**2)+((z3/OR)**2)
R4=((x4/OR)**2)+((y4/length)**2)+((z4/OR)**2)
R5=((x5/OR)**2)+((y5/length)**2)+((z5/OR)**2)
R6=((x6/OR)**2)+((y6/length)**2)+((z6/OR)**2)

```

CC Test points

```

    if (R1.LE.1.0) then
      Do 14 n=0,400
        zeta=Pi*n/(36*400)
        Qi=Bragg*(dcos(2*zeta)-1)
        Qj=Bragg*dcos(cut2)*dsin(2*zeta)
        Qk=Bragg*dsin(cut2)*dsin(2*zeta)
        A(1,n)=A(1,n)+2*dcos(Qi*x1+Qj*y1+Qk*z1)
14      continue
        count=count+1
      endif

    if (R2.LE.1.0) then
      Do 24 n=0,400
        zeta=Pi*n/(36*400)
        Qi=Bragg*(dcos(2*zeta)-1)
        Qj=Bragg*dcos(cut2)*dsin(2*zeta)
        Qk=Bragg*dsin(cut2)*dsin(2*zeta)
        A(1,n)=A(1,n)+2*dcos(Qi*x2+Qj*y2+Qk*z2)
24      continue
        count=count+1
      endif

```

```

if (R3.LE.1.0) then
  Do 34 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x3+Qj*y3+Qk*z3)
34  continue
    count=count+1
  endif

if (R4.LE.1.0) then
  Do 44 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x4+Qj*y4+Qk*z4)
44  continue
    count=count+1
  endif

if (R5.LE.1.0) then
  Do 54 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x5+Qj*y5+Qk*z5)
54  continue
    count=count+1
  endif

if (R6.LE.1.0) then
  Do 64 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x6+Qj*y6+Qk*z6)
64  continue
    count=count+1
  endif

```

10 Continue

```
print*, '-----'  
print*, 'Iterations'  
print*, iter  
print*, '-----'  
print*, 'Number of Points'  
print*, count  
print*, '-----'
```

```
final=A(1,0)
```

```
Do 101 n=0,400
```

```
A(1,n)=dlog10((A(1,n)/final)**2)
```

101 Continue

```
sl=rtc()
```

```
timespent=sl-s
```

```
dev=(dble(iter))/timespent
```

```
print*, 'CPU Time (seconds)'
```

```
print*, timespent
```

```
print*, 'Iterations Per Seconds'
```

```
print*, dev
```

```
Do 102 n=0,400
```

```
write (15,*) A(1,n)
```

102 Continue

```
End
```

C

Random Number Function Goes Here

B.3. Cylinder

CC This file will model a cylinder through random numbers.

```
use portlib
real*8 s, sl, timespent
Integer*4 iter, l, number
Integer*4 m, n, count, seed
Integer ia1, ia2, iat, ib1, ib2, ibt,
+ ic1, ic2, ict, id1, id2, idt, ie1, ie2, iet, if1, if2, ift,
+ ja1, ja2, jat, jb1, jb2, jbt, jc1, jc2, jct, jd1, jd2, jdt,
+ je1, je2, jet, jf1, jf2, jft, ka1, ka2, kat, kb1, kb2, kbt,
+ kc1, kc2, kct, kd1, kd2, kdt, ke1, ke2, ket, kf1, kf2, kft
Real*8 OR, A(7,401), Pi, lamda, Qi, dev, length,
+ Qj, Qk, Bragg, zeta, B(1,118), ran2, cut1, cut2,
+ x1, x2, x3, x4, x5, x6, y1, y2, y3, y4, y5,
+ y6, z1, z2, z3, z4, z5, z6, R1, R2, R3, R4,
+ R5, R6, S1, S2, S3, S4, S5, S6, final
```

CC Initilization of some of the variables and the constants.

```
Do l n=0,400
  A(1,n)=0.0
l Continue
Pi=3.1415927
lamda=.154242
Bragg=2*Pi/lamda
count=0
```

CC Input statements.

```
Print*, 'Cylinder Scattering Simulation Through Random Numbers.'
Print*, 'Enter the radius of the cylinder [nm].'
Read*, OR
Print*, 'Enter half the length of the cylinder [nm].'
Read*, length
Print*, 'Enter the slice desired of the cylinder [nm].'
Print*, '(Pi times this number.)'
Read*, cut1
Print*, 'Enter the number of iterations desired.'
Read*, iter
Print*, 'Enter a seed for the random numbers.'
Read*, seed
cut2=cut1*Pi
s=rtc()
```

CC Iteration Loop

```
Do 10 l=1,iter
```

```

    number=seed+1
CC Draw Random Numbers
    Do 2 m=0,117
      B(1,m)=ran2(number)
2    continue

    ia1=idint(10*B(1,100))
    ia2=idint((100*B(1,100))-(10*ia1))
    iat=10*ia2+ia1

    ib1=idint(10*B(1,101))
    ib2=idint((100*B(1,101))-(10*ib1))
    ibt=10*ib2+ib1

    ic1=idint(10*B(1,102))
    ic2=idint((100*B(1,102))-(10*ic1))
    ict=10*ic2+ic1

    id1=idint(10*B(1,103))
    id2=idint((100*B(1,103))-(10*id1))
    idt=10*id2+id1

    ie1=idint(10*B(1,104))
    ie2=idint((100*B(1,104))-(10*ie1))
    iet=10*ie2+ie1

    if1=idint(10*B(1,105))
    if2=idint((100*B(1,105))-(10*if1))
    ift=10*if2+if1

    ja1=idint(10*B(1,106))
    ja2=idint((100*B(1,106))-(10*ja1))
    jat=10*ja2+ja1

    jb1=idint(10*B(1,107))
    jb2=idint((100*B(1,107))-(10*jb1))
    jbt=10*jb2+jb1

    jc1=idint(10*B(1,108))
    jc2=idint((100*B(1,108))-(10*jc1))
    jct=10*jc2+jc1

    jd1=idint(10*B(1,109))
    jd2=idint((100*B(1,109))-(10*jd1))

```

jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1

jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1

ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1

kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
kct=10*kc2+kc1

kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1

ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1

kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1

x1=2.00002*OR*B(1,iat)-1.00001*OR
x2=2.00002*OR*B(1,ibt)-1.00001*OR
x3=2.00002*OR*B(1,ict)-1.00001*OR
x4=2.00002*OR*B(1,idt)-1.00001*OR
x5=2.00002*OR*B(1,iet)-1.00001*OR
x6=2.00002*OR*B(1,ift)-1.00001*OR
y1=2.00002*length*B(1,jat)-1.00001*length
y2=2.00002*length*B(1,jbt)-1.00001*length
y3=2.00002*length*B(1,jct)-1.00001*length
y4=2.00002*length*B(1,jdt)-1.00001*length

```

y5=2.00002*length*B(1,jet)-1.00001*length
y6=2.00002*length*B(1,jft)-1.00001*length
z1=2.00002*OR*B(1,kat)-1.00001*OR
z2=2.00002*OR*B(1,kbt)-1.00001*OR
z3=2.00002*OR*B(1,kct)-1.00001*OR
z4=2.00002*OR*B(1,kdt)-1.00001*OR
z5=2.00002*OR*B(1,ket)-1.00001*OR
z6=2.00002*OR*B(1,kft)-1.00001*OR

```

```

R1=dsqrt((x1**2)+(z1**2))
R2=dsqrt((x2**2)+(z2**2))
R3=dsqrt((x3**2)+(z3**2))
R4=dsqrt((x4**2)+(z4**2))
R5=dsqrt((x5**2)+(z5**2))
R6=dsqrt((x6**2)+(z6**2))
S1=dabs(y1)
S2=dabs(y2)
S3=dabs(y3)
S4=dabs(y4)
S5=dabs(y5)
S6=dabs(y6)

```

CC Test points

```

    if (S1.LE.length) then
    if (R1.LE.OR) then
    Do 14 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x1+Qj*y1+Qk*z1)
14  continue
    count=count+1
    endif
    endif

    if (S2.LE.length) then
    if (R2.LE.OR) then
    Do 24 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x2+Qj*y2+Qk*z2)

```



```

24  continue
    count=count+1
    endif
    endif

    if (S3.LE.length) then
    if (R3.LE.OR) then
    Do 34 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x3+Qj*y3+Qk*z3)
34  continue
    count=count+1
    endif
    endif

    if (S4.LE.length) then
    if (R4.LE.OR) then
    Do 44 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x4+Qj*y4+Qk*z4)
44  continue
    count=count+1
    endif
    endif

    if (S5.LE.length) then
    if (R5.LE.OR) then
    Do 54 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x5+Qj*y5+Qk*z5)
54  continue
    count=count+1
    endif
    endif

```

```

    if (S6.LE.length) then
    if (R6.LE.OR) then
    Do 64 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x6+Qj*y6+Qk*z6)
64  continue
    count=count+1
    endif
    endif
10  Continue
    print*, '-----'
    print*, 'Iterations'
    print*, iter
    print*, '-----'
    print*, 'Number of Points'
    print*, count
    print*, '-----'
    final=A(1,0)
    Do 101 n=0,400
    A(1,n)=(A(1,n)/final)**2
101  Continue
    sl=rtc()
    timespent=sl-s
    dev=(dble(iter))/timespent
    print*, 'CPU Time (seconds)'
    print*, timespent
    print*, 'Iterations Per Seconds'
    print*, dev

    Do 102 n=0,400
    write (15,*) A(1,n)
102  Continue

    End
C
    Random Number Function Goes Here

```

B.4. Tilted Cylinder

CC This file will model t-cylinder scattering through random numbers.

```
use portlib
real*8 s, sl, timespent
Integer*4 iter, l, number
Integer*4 m, n, count, seed
Integer ia1, ia2, iat, ib1, ib2, ibt, ic1, ic2, ict, id1, id2, idt,
+ ie1, ie2, iet, if1, if2, ift, ja1, ja2, jat, jb1, jb2, jbt,
+ jc1, jc2, jct, jd1, jd2, jdt, je1, je2, jet, jf1, jf2, jft,
+ ka1, ka2, kat, kb1, kb2, kbt, kc1, kc2, ket, kd1, kd2, kdt,
+ ke1, ke2, ket, kf1, kf2, kft
Real*8 OR, A(7,401), Pi, lamda, Qi, dev, length, gamma,
+ Qj, Qk, Bragg, zeta, B(1,118), ran2, cut1, cut2,
+ x1, x2, x3, x4, x5, x6, y1, y2, y3, y4, y5,
+ y6, z1, z2, z3, z4, z5, z6, R1, R2, R3, R4,
+ R5, R6, S1, S2, S3, S4, S5, S6, final, alpha,
+ beta, dist, xprime, yprime, zprime, ease
```

CC Initialization of some of the variables and the constants.

```
Do l n=0,400
A(1,n)=0.0
1 Continue
Pi=3.1415927
lamda=.154242
Bragg=2*Pi/lamda
count=0
```

CC Input statements.

```
Print*, 'Cylinder Scattering Simulation Through Random Numbers.'
Print*, 'Enter the radius of the cylinder [nm].'
Read*, OR
Print*, 'Enter half the length of the cylinder [nm].'
Read*, length
Print*, 'Enter the slice desired of the cylinder [nm].'
Print*, '(Pi times this number.)'
Read*, cut1
Print*, 'Enter the number of iterations desired.'
Read*, iter
Print*, 'Enter a seed for the random numbers.'
Read*, seed
Print*, 'Enter the tilt angle.'
Print*, '(Pi times this number.)'
Read*, gamma
```

```

        beta=gamma*Pi
        cut2=cut1*Pi
        s=rtc()
CC Iteration Loop
        Do 10 l=1,iter
            number=seed+l
CC Draw Random Numbers
            Do 2 m=0,117
                B(l,m)=ran2(number)
2          continue

        ia1=idint(10*B(1,100))
        ia2=idint((100*B(1,100))-(10*ia1))
        iat=10*ia2+ia1

        ib1=idint(10*B(1,101))
        ib2=idint((100*B(1,101))-(10*ib1))
        ibt=10*ib2+ib1

        ic1=idint(10*B(1,102))
        ic2=idint((100*B(1,102))-(10*ic1))
        ict=10*ic2+ic1

        id1=idint(10*B(1,103))
        id2=idint((100*B(1,103))-(10*id1))
        idt=10*id2+id1

        ie1=idint(10*B(1,104))
        ie2=idint((100*B(1,104))-(10*ie1))
        iet=10*ie2+ie1

        if1=idint(10*B(1,105))
        if2=idint((100*B(1,105))-(10*if1))
        ift=10*if2+if1

        ja1=idint(10*B(1,106))
        ja2=idint((100*B(1,106))-(10*ja1))
        jat=10*ja2+ja1

        jb1=idint(10*B(1,107))
        jb2=idint((100*B(1,107))-(10*jb1))
        jbt=10*jb2+jb1

        jc1=idint(10*B(1,108))

```

jc2=idint((100*B(1,108))-(10*jc1))
jct=10*jc2+jc1

jd1=idint(10*B(1,109))
jd2=idint((100*B(1,109))-(10*jd1))
jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1

jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1

ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1

kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
kct=10*kc2+kc1

kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1

ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1

kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1

x1=2.00002*OR*B(1,iat)-1.00001*OR
x2=2.00002*OR*B(1,ibt)-1.00001*OR
x3=2.00002*OR*B(1,ict)-1.00001*OR
x4=2.00002*OR*B(1,idt)-1.00001*OR
x5=2.00002*OR*B(1,iet)-1.00001*OR

```

x6=2.00002*OR*B(1,ift)-1.00001*OR
y1=2.00002*length*B(1,jat)-1.00001*length
y2=2.00002*length*B(1,jbt)-1.00001*length
y3=2.00002*length*B(1,jct)-1.00001*length
y4=2.00002*length*B(1,jdt)-1.00001*length
y5=2.00002*length*B(1,jet)-1.00001*length
y6=2.00002*length*B(1,jft)-1.00001*length
z1=2.00002*OR*B(1,kat)-1.00001*OR
z2=2.00002*OR*B(1,kbt)-1.00001*OR
z3=2.00002*OR*B(1,kct)-1.00001*OR
z4=2.00002*OR*B(1,kdt)-1.00001*OR
z5=2.00002*OR*B(1,ket)-1.00001*OR
z6=2.00002*OR*B(1,kft)-1.00001*OR

```

```

R1=dsqrt((x1**2)+(z1**2))
R2=dsqrt((x2**2)+(z2**2))
R3=dsqrt((x3**2)+(z3**2))
R4=dsqrt((x4**2)+(z4**2))
R5=dsqrt((x5**2)+(z5**2))
R6=dsqrt((x6**2)+(z6**2))
S1=dabs(y1)
S2=dabs(y2)
S3=dabs(y3)
S4=dabs(y4)
S5=dabs(y5)
S6=dabs(y6)

```

CC Test points

```

if (S1.LE.length) then
if (R1.LE.OR) then
if (y1.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x1/y1)
end if
dist=dsqrt((x1**2)+(y1**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z1
Do 14 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)

```

```

Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
14 continue
count=count+1
endif
endif

if (S2.LE.length) then
if (R2.LE.OR) then
if (y2.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x2/y2)
end if
dist=dsqrt((x2**2)+(y2**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z2
Do 24 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
24 continue
count=count+1
endif
endif

if (S3.LE.length) then
if (R3.LE.OR) then
if (y3.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x3/y3)
end if
dist=dsqrt((x3**2)+(y3**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z3
Do 34 n=0,400
zeta=Pi*n/(36*400)

```

```

    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    ease=Qi*xprime+Qj*yprime+Qk*zprime
    A(1,n)=A(1,n)+2*dcos(ease)
34  continue
    count=count+1
    endif
    endif

    if (S4.LE.length) then
    if (R4.LE.OR) then
    if (y4.EQ.0.0) then
    alpha=Pi/2
    else
    alpha=datan(x4/y4)
    end if
    dist=dsqrt((x4**2)+(y4**2))
    xprime=dist*dsin(alpha+beta)
    yprime=dist*dcos(alpha+beta)
    zprime=z4
    Do 44 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    ease=Qi*xprime+Qj*yprime+Qk*zprime
    A(1,n)=A(1,n)+2*dcos(ease)
44  continue
    count=count+1
    endif
    endif

    if (S5.LE.length) then
    if (R5.LE.OR) then
    if (y5.EQ.0.0) then
    alpha=Pi/2
    else
    alpha=datan(x5/y5)
    end if
    dist=dsqrt((x5**2)+(y5**2))
    xprime=dist*dsin(alpha+beta)
    yprime=dist*dcos(alpha+beta)
    zprime=z5

```



```

Do 54 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
54 continue
count=count+1
endif
endif

if (S6.LE.length) then
if (R6.LE.OR) then
if (y6.EQ.0.0) then
alpha=Pi/2
else
alpha=atan(x6/y6)
end if
dist=dsqrt((x6**2)+(y6**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z6
Do 64 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
64 continue
count=count+1
endif
endif
10 Continue
print*, '-----'
print*, 'Iterations'
print*, iter
print*, '-----'
print*, 'Number of Points'
print*, count
print*, '-----'

final=A(1,0)

```

```

    Do 101 n=0,400
    A(1,n)=dlog10((A(1,n)/final)**2)
101  Continue

    sl=rtc()
    timespent=sl-s
    dev=(dble(iter))/timespent
    print*, 'CPU Time (seconds)'
    print*, timespent
    print*, 'Iterations Per Seconds'
    print*, dev

    Do 102 n=0,400
    write (15,*) A(1,n)
102  Continue

    End
C
    Random Number Function Goes Here

```

B.5. Elongated Hexagon

CC This file will model elongated hexagon scattering through random numbers.

```
use portlib
real*8 s, sl, timespent
Integer*4 iter, l, number
Integer*4 m, n, count, seed
Integer ial, ia2, iat, ib1, ib2, ibt, ic1, ic2, ict, id1, id2, idt,
+ iel, ie2, iet, if1, if2, ift, jal, ja2, jat, jbl, jb2, jbt,
+ jcl, jc2, jct, jdl, jd2, jdt, je1, je2, jet, jf1, jf2, jft,
+ ka1, ka2, kat, kb1, kb2, kbt, kc1, kc2, ket, kd1, kd2, kdt,
+ kel, ke2, ket, kfl, kf2, kft
Real*8 OR, A(7,401), Pi, lamda, Qi, dev, length, gamma,
+ Qj, Qk, Bragg, zeta, B(1,118), ran2, cut1, cut2,
+ x1, x2, x3, x4, x5, x6, y1, y2, y3, y4, y5,
+ y6, z1, z2, z3, z4, z5, z6, R1, R2, R3, R4,
+ R5, R6, S1, S2, S3, S4, S5, S6, T1, T2, T3,
+ T4, T5, T6, U1, U2, U3, U4, U5, U6, V1, V2,
+ V3, V4, V5, V6, P1, P2, P3, P4, P5, P6, final,
+ alpha, beta, dist, xprime, zulu, yprime, zprime, ease, ang
```

CC Initialization of some of the variables and the constants.

```
Do l n=0,400
  A(1,n)=0.0
1  Continue
  Pi=3.1415927
  lamda=.154242
  Bragg=2*Pi/lamda
  count=0
  ang=Pi/6
  zulu=dsqrt(dble(3.0))
```

CC Input statements.

```
Print*, 'Elongated hexagon Scattering Simulation [Random].'
Print*, 'Enter the radius of the hexagon [nm].'
Read*, OR
Print*, 'Enter half the length of the hexagon [nm].'
Read*, length
Print*, 'Enter the slice desired of the hexagon [nm].'
Print*, '(Pi/3 times this number.)'
Read*, cut1
Print*, 'Enter the number of iterations desired.'
Read*, iter
Print*, 'Enter a seed for the random numbers.'
```

```

Read*, seed
Print*, 'Enter the tilt angle.'
Print*, '(Pi/3 times this number.)'
Read*, gamma
beta=gamma*Pi/3.0
cut2=cut1*Pi/3.0
s=rtc()
CC Iteration Loop
  Do 10 l=1,iter
    number=seed+l
CC Draw Random Numbers
  Do 2 m=0,117
    B(l,m)=ran2(number)
  2 continue

  ia1=idint(10*B(1,100))
  ia2=idint((100*B(1,100))-(10*ia1))
  iat=10*ia2+ia1

  ib1=idint(10*B(1,101))
  ib2=idint((100*B(1,101))-(10*ib1))
  ibt=10*ib2+ib1

  ic1=idint(10*B(1,102))
  ic2=idint((100*B(1,102))-(10*ic1))
  ict=10*ic2+ic1

  id1=idint(10*B(1,103))
  id2=idint((100*B(1,103))-(10*id1))
  idt=10*id2+id1

  ie1=idint(10*B(1,104))
  ie2=idint((100*B(1,104))-(10*ie1))
  iet=10*ie2+ie1

  if1=idint(10*B(1,105))
  if2=idint((100*B(1,105))-(10*if1))
  ift=10*if2+if1

  ja1=idint(10*B(1,106))
  ja2=idint((100*B(1,106))-(10*ja1))
  jat=10*ja2+ja1

  jbl=idint(10*B(1,107))

```

```

jb2=idint((100*B(1,107))-(10*jb1))
jbt=10*jb2+jb1

jc1=idint(10*B(1,108))
jc2=idint((100*B(1,108))-(10*jc1))
jct=10*jc2+jc1

jd1=idint(10*B(1,109))
jd2=idint((100*B(1,109))-(10*jd1))
jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1

jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1

ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1

kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
kct=10*kc2+kc1

kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1

ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1

kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1

x1=2.00002*OR*B(1,iat)-1.00001*OR

```

```

x2=2.00002*OR*B(1,ibt)-1.00001*OR
x3=2.00002*OR*B(1,ict)-1.00001*OR
x4=2.00002*OR*B(1,idt)-1.00001*OR
x5=2.00002*OR*B(1,iet)-1.00001*OR
x6=2.00002*OR*B(1,ift)-1.00001*OR
y1=2.00002*length*B(1,jat)-1.00001*length
y2=2.00002*length*B(1,jbt)-1.00001*length
y3=2.00002*length*B(1,jct)-1.00001*length
y4=2.00002*length*B(1,jdt)-1.00001*length
y5=2.00002*length*B(1,jet)-1.00001*length
y6=2.00002*length*B(1,jft)-1.00001*length
z1=2.00002*OR*B(1,kat)-1.00001*OR
z2=2.00002*OR*B(1,kbt)-1.00001*OR
z3=2.00002*OR*B(1,kct)-1.00001*OR
z4=2.00002*OR*B(1,kdt)-1.00001*OR
z5=2.00002*OR*B(1,ket)-1.00001*OR
z6=2.00002*OR*B(1,kft)-1.00001*OR

```

```

R1=dabs(x1)
R2=dabs(x2)
R3=dabs(x3)
R4=dabs(x4)
R5=dabs(x5)
R6=dabs(x6)
S1=dabs(y1)
S2=dabs(y2)
S3=dabs(y3)
S4=dabs(y4)
S5=dabs(y5)
S6=dabs(y6)
T1=dabs(z1)
T2=dabs(z2)
T3=dabs(z3)
T4=dabs(z4)
T5=dabs(z5)
T6=dabs(z6)

```

```

if (R1.EQ.0.0) then
U1=Pi/2
else
U1=datan(T1/R1)
end if

```

```

if (R2.EQ.0.0) then
U2=Pi/2
else
U2=datan(T2/R2)
end if

```

```

if (R3.EQ.0.0) then
U3=Pi/2
else
U3=datan(T3/R3)
end if

```

```

if (R4.EQ.0.0) then
U4=Pi/2
else
U4=datan(T4/R4)
end if

```

```

if (R5.EQ.0.0) then
U5=Pi/2
else
U5=datan(T5/R5)
end if

```

```

if (R6.EQ.0.0) then
U6=Pi/2
else
U6=datan(T6/R6)
end if

```

CC Test points

```

if (S1.LE.length) then
if (U1.LE.ang) then
V1=OR*dcos(ang)
if (R1.LE.V1) then
if (y1.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x1/y1)
end if
dist=dsqrt((x1**2)+(y1**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)

```

```

zprime=z1
Do 14 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
14 continue
count=count+1
endif
else
P1=OR-(R1/zulu)
if (T1.LE.P1) then
if (y1.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x1/y1)
end if
dist=dsqrt((x1**2)+(y1**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z1
Do 15 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
15 continue
count=count+1
endif
endif
endif

if (S2.LE.length) then
if (U2.LE.ang) then
V2=OR*dcos(ang)
if (R2.LE.V2) then
if (y2.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x2/y2)

```



```

end if
dist=dsqrt((x2**2)+(y2**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z2
Do 24 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
24 continue
count=count+1
endif
else
P2=OR-(R2/zulu)
if (T2.LE.P2) then
if (y2.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x2/y2)
end if
dist=dsqrt((x2**2)+(y2**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z2
Do 25 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
25 continue
count=count+1
endif
endif
endif

if (S3.LE.length) then
if (U3.LE.ang) then
V3=OR*dcos(ang)
if (R3.LE.V3) then

```

```

    if (y3.EQ.0.0) then
    alpha=Pi/2
    else
    alpha=atan(x3/y3)
    end if
    dist=dsqrt((x3**2)+(y3**2))
    xprime=dist*dsin(alpha+beta)
    yprime=dist*dcos(alpha+beta)
    zprime=z3
    Do 34 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    ease=Qi*xprime+Qj*yprime+Qk*zprime
    A(1,n)=A(1,n)+2*dcos(ease)
34  continue
    count=count+1
    endif
    else
    P3=OR-(R3/zulu)
    if (T3.LE.P3) then
    if (y3.EQ.0.0) then
    alpha=Pi/2
    else
    alpha=atan(x3/y3)
    end if
    dist=dsqrt((x3**2)+(y3**2))
    xprime=dist*dsin(alpha+beta)
    yprime=dist*dcos(alpha+beta)
    zprime=z3
    Do 35 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qj=Bragg*dcos(cut2)*dsin(2*zeta)
    Qk=Bragg*dsin(cut2)*dsin(2*zeta)
    ease=Qi*xprime+Qj*yprime+Qk*zprime
    A(1,n)=A(1,n)+2*dcos(ease)
35  continue
    count=count+1
    endif
    endif
    endif

```

```

if (S4.LE.length) then
if (U4.LE.ang) then
V4=OR*dcos(ang)
if (R4.LE.V4) then
if (y4.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x4/y4)
end if
dist=dsqrt((x4**2)+(y4**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z4
Do 44 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
44 continue
count=count+1
endif
else
P4=OR-(R4/zulu)
if (T4.LE.P4) then
if (y4.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x4/y4)
end if
dist=dsqrt((x4**2)+(y4**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z4
Do 45 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
45 continue
count=count+1

```

```

endif
endif
endif

if (S5.LE.length) then
if (U5.LE.ang) then
V5=OR*dcos(ang)
if (R5.LE.V5) then
if (y5.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x5/y5)
end if
dist=dsqrt((x5**2)+(y5**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z5
Do 54 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
54 continue
count=count+1
endif
else
P5=OR-(R5/zulu)
if (T5.LE.P5) then
if (y5.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x5/y5)
end if
dist=dsqrt((x5**2)+(y5**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z5
Do 55 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)

```

```

ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
55 continue
count=count+1
endif
endif
endif

if (S6.LE.length) then
if (U6.LE.ang) then
V6=OR*dcos(ang)
if (R6.LE.V6) then
if (y6.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x6/y6)
end if
dist=dsqrt((x6**2)+(y6**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z6
Do 64 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
64 continue
count=count+1
endif
else
P6=OR-(R6/zulu)
if (T6.LE.P6) then
if (y6.EQ.0.0) then
alpha=Pi/2
else
alpha=datan(x6/y6)
end if
dist=dsqrt((x6**2)+(y6**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z6
Do 65 n=0,400

```

```

zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease=Qi*xprime+Qj*yprime+Qk*zprime
A(1,n)=A(1,n)+2*dcos(ease)
65  continue
    count=count+1
    endif
    endif
    endif

10  Continue
    print*, '-----'
    print*, 'Iterations'
    print*, iter
    print*, '-----'
    print*, 'Number of Points'
    print*, count
    print*, '-----'
    final=A(1,0)

    Do 101 n=0,400
    A(1,n)=dlog10((A(1,n)/final)**2)
101  Continue

    sl=rtc()
    timespent=sl-s
    dev=(dble(iter))/timespent
    print*, 'CPU Time (seconds)'
    print*, timespent
    print*, 'Iterations Per Seconds'
    print*, dev

    Do 102 n=0,400
    write (15,*) A(1,n)
102  Continue

    End

C
    Random Number Function Goes Here

```

B.6. Bundled Cylinders

CC This file will model fiber scattering through random numbers.

```
use portlib
real*8 s, sl, timespent
Integer*4 iter, l, number
Integer*4 m, n, count, seed
Integer ial, ia2, iat, ib1, ib2, ibt, ic1, ic2, ict, id1, id2, idt,
+ ie1, ie2, iet, if1, if2, ift, ja1, ja2, jat, jb1, jb2, jbt,
+ jc1, jc2, jct, jd1, jd2, jdt, je1, je2, jet, jf1, jf2, jft,
+ ka1, ka2, kat, kb1, kb2, kbt, kc1, kc2, kct, kd1, kd2, kdt,
+ ke1, ke2, ket, kf1, kf2, kft
Real*8 OR, A(7,401), Pi, lamda, Qi, dev, length, gamma,
+ Qj, Qk, Bragg, zeta, B(1,118), ran2, cut1, cut2,
+ x1, x2, x3, x4, x5, x6, y1, y2, y3, y4, y5,
+ y6, z1, z2, z3, z4, z5, z6, R1, R2, R3, R4,
+ R5, R6, S1, S2, S3, S4, S5, S6, final, alpha,
+ beta, dist, xprime, yprime, zprime, ease1, ease2,
+ ease3, ease4, ease5, ease6, ease7, f1, f2, f3
```

CC Initialization of some of the variables and the constants.

```
Do 1 n=0,400
  A(1,n)=0.0
1  Continue
  Pi=3.1415927
  lamda=.154242
  Bragg=2*Pi/lamda
  count=0
```

CC Input statements.

```
Print*, 'Fiber Scattering Simulation Through Random Numbers.'
Print*, 'Enter the radius of the innermost cylinder [nm].'
Read*, OR
Print*, 'Enter half the length of the fiber [nm].'
Read*, length
Print*, 'Enter the slice desired of the fiber [nm].'
Print*, '(Pi/3 times this number.)'
Read*, cut1
Print*, 'Enter the number of iterations desired.'
Read*, iter
Print*, 'Enter a seed for the random numbers.'
Read*, seed
Print*, 'Enter the tilt angle.'
Print*, '(Pi/3 times this number.)'
Read*, gamma
```

```

    beta=gamma*Pi/3.0
    cut2=cut1*Pi/3.0
    s=rtc()
    f1=2*OR*dcos(Pi/6)
    f2=2*OR*dsin(Pi/6)
    f3=2*OR
CC Iteration Loop
    Do 10 l=1,iter
        number=seed+l
CC Draw Random Numbers
    Do 2 m=0,117
        B(l,m)=ran2(number)
2    continue

    ia1=idint(10*B(1,100))
    ia2=idint((100*B(1,100))-(10*ia1))
    iat=10*ia2+ia1

    ib1=idint(10*B(1,101))
    ib2=idint((100*B(1,101))-(10*ib1))
    ibt=10*ib2+ib1

    ic1=idint(10*B(1,102))
    ic2=idint((100*B(1,102))-(10*ic1))
    ict=10*ic2+ic1

    id1=idint(10*B(1,103))
    id2=idint((100*B(1,103))-(10*id1))
    idt=10*id2+id1

    ie1=idint(10*B(1,104))
    ie2=idint((100*B(1,104))-(10*ie1))
    iet=10*ie2+ie1

    if1=idint(10*B(1,105))
    if2=idint((100*B(1,105))-(10*if1))
    ift=10*if2+if1

    ja1=idint(10*B(1,106))
    ja2=idint((100*B(1,106))-(10*ja1))
    jat=10*ja2+ja1

    jb1=idint(10*B(1,107))
    jb2=idint((100*B(1,107))-(10*jb1))

```


jbt=10*jb2+jb1

jc1=idint(10*B(1,108))
jc2=idint((100*B(1,108))-(10*jc1))
jct=10*jc2+jc1

jd1=idint(10*B(1,109))
jd2=idint((100*B(1,109))-(10*jd1))
jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1

jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1

ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1

kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
kct=10*kc2+kc1

kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1

ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1

kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1

x1=2.00002*OR*B(1,iat)-1.00001*OR
x2=2.00002*OR*B(1,ibt)-1.00001*OR

```

x3=2.00002*OR*B(1,ict)-1.00001*OR
x4=2.00002*OR*B(1,idt)-1.00001*OR
x5=2.00002*OR*B(1,iet)-1.00001*OR
x6=2.00002*OR*B(1,ift)-1.00001*OR
y1=2.00002*length*B(1,jat)-1.00001*length
y2=2.00002*length*B(1,jbt)-1.00001*length
y3=2.00002*length*B(1,jct)-1.00001*length
y4=2.00002*length*B(1,jdt)-1.00001*length
y5=2.00002*length*B(1,jet)-1.00001*length
y6=2.00002*length*B(1,jft)-1.00001*length
z1=2.00002*OR*B(1,kat)-1.00001*OR
z2=2.00002*OR*B(1,kbt)-1.00001*OR
z3=2.00002*OR*B(1,kct)-1.00001*OR
z4=2.00002*OR*B(1,kdt)-1.00001*OR
z5=2.00002*OR*B(1,ket)-1.00001*OR
z6=2.00002*OR*B(1,kft)-1.00001*OR

```

```

R1=dsqrt((x1**2)+(z1**2))
R2=dsqrt((x2**2)+(z2**2))
R3=dsqrt((x3**2)+(z3**2))
R4=dsqrt((x4**2)+(z4**2))
R5=dsqrt((x5**2)+(z5**2))
R6=dsqrt((x6**2)+(z6**2))
S1=dabs(y1)
S2=dabs(y2)
S3=dabs(y3)
S4=dabs(y4)
S5=dabs(y5)
S6=dabs(y6)

```

CC Test points

```

if (S1.LE.length) then
  if (R1.LE.OR) then
    if (y1.EQ.0.0) then
      alpha=Pi/2
    else
      alpha=datan(x1/y1)
    end if
    dist=dsqrt((x1**2)+(y1**2))
    xprime=dist*dsin(alpha+beta)
    yprime=dist*dcos(alpha+beta)
    zprime=z1
  Do 14 n=0,400

```

```

zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease1=Qi*xprime+Qj*yprime+Qk*zprime
ease2=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime+f2)
ease3=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime-f2)
ease4=Qi*xprime+Qj*yprime+Qk*(zprime+f3)
ease5=Qi*xprime+Qj*yprime+Qk*(zprime-f3)
ease6=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime+f2)
ease7=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime-f2)
A(1,n)=A(1,n)+2*dcos(ease1)+2*dcos(ease2)+
+ 2*dcos(ease3)+2*dcos(ease4)+2*dcos(ease5)+
+ 2*dcos(ease6)+2*dcos(ease7)
14 continue
count=count+1
endif
endif

if (S2.LE.length) then
if (R2.LE.OR) then
if (y2.EQ.0.0) then
alpha=Pi/2
else
alpha=atan(x2/y2)
end if
dist=dsqrt((x2**2)+(y2**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z2
Do 24 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease1=Qi*xprime+Qj*yprime+Qk*zprime
ease2=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime+f2)
ease3=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime-f2)
ease4=Qi*xprime+Qj*yprime+Qk*(zprime+f3)
ease5=Qi*xprime+Qj*yprime+Qk*(zprime-f3)
ease6=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime+f2)
ease7=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime-f2)
A(1,n)=A(1,n)+2*dcos(ease1)+2*dcos(ease2)+
+ 2*dcos(ease3)+2*dcos(ease4)+2*dcos(ease5)+

```

```

+      2*dcos(ease6)+2*dcos(ease7)
24    continue
      count=count+1
      endif
      endif

      if (S3.LE.length) then
      if (R3.LE.OR) then
      if (y3.EQ.0.0) then
      alpha=Pi/2
      else
      alpha=datan(x3/y3)
      end if
      dist=dsqrt((x3**2)+(y3**2))
      xprime=dist*dsin(alpha+beta)
      yprime=dist*dcos(alpha+beta)
      zprime=z3
      Do 34 n=0,400
      zeta=Pi*n/(36*400)
      Qi=Bragg*(dcos(2*zeta)-1)
      Qj=Bragg*dcos(cut2)*dsin(2*zeta)
      Qk=Bragg*dsin(cut2)*dsin(2*zeta)
      ease1=Qi*xprime+Qj*yprime+Qk*zprime
      ease2=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime+f2)
      ease3=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime-f2)
      ease4=Qi*xprime+Qj*yprime+Qk*(zprime+f3)
      ease5=Qi*xprime+Qj*yprime+Qk*(zprime-f3)
      ease6=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime+f2)
      ease7=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime-f2)
      A(1,n)=A(1,n)+2*dcos(ease1)+2*dcos(ease2)+
+      2*dcos(ease3)+2*dcos(ease4)+2*dcos(ease5)+
+      2*dcos(ease6)+2*dcos(ease7)
34    continue
      count=count+1
      endif
      endif

      if (S4.LE.length) then
      if (R4.LE.OR) then
      if (y4.EQ.0.0) then
      alpha=Pi/2
      else
      alpha=datan(x4/y4)
      end if

```

```

dist=dsqrt((x4**2)+(y4**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z4
Do 44 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease1=Qi*xprime+Qj*yprime+Qk*zprime
ease2=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime+f2)
ease3=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime-f2)
ease4=Qi*xprime+Qj*yprime+Qk*(zprime+f3)
ease5=Qi*xprime+Qj*yprime+Qk*(zprime-f3)
ease6=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime+f2)
ease7=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime-f2)
A(1,n)=A(1,n)+2*dcos(ease1)+2*dcos(ease2)+
+ 2*dcos(ease3)+2*dcos(ease4)+2*dcos(ease5)+
+ 2*dcos(ease6)+2*dcos(ease7)
44 continue
count=count+1
endif
endif

if (S5.LE.length) then
if (R5.LE.OR) then
if (y5.EQ.0.0) then
alpha=Pi/2
else
alpha=atan(x5/y5)
end if
dist=dsqrt((x5**2)+(y5**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z5
Do 54 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease1=Qi*xprime+Qj*yprime+Qk*zprime
ease2=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime+f2)
ease3=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime-f2)
ease4=Qi*xprime+Qj*yprime+Qk*(zprime+f3)

```

```

ease5=Qi*xprime+Qj*yprime+Qk*(zprime-f3)
ease6=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime+f2)
ease7=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime-f2)
A(1,n)=A(1,n)+2*dcos(ease1)+2*dcos(ease2)+
+ 2*dcos(ease3)+2*dcos(ease4)+2*dcos(ease5)+
+ 2*dcos(ease6)+2*dcos(ease7)
54 continue
count=count+1
endif
endif

if (S6.LE.length) then
if (R6.LE.OR) then
if (y6.EQ.0.0) then
alpha=Pi/2
else
alpha=atan(x6/y6)
end if
dist=dsqrt((x6**2)+(y6**2))
xprime=dist*dsin(alpha+beta)
yprime=dist*dcos(alpha+beta)
zprime=z6
Do 64 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
ease1=Qi*xprime+Qj*yprime+Qk*zprime
ease2=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime+f2)
ease3=Qi*(xprime+f1)+Qj*yprime+Qk*(zprime-f2)
ease4=Qi*xprime+Qj*yprime+Qk*(zprime+f3)
ease5=Qi*xprime+Qj*yprime+Qk*(zprime-f3)
ease6=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime+f2)
ease7=Qi*(xprime-f1)+Qj*yprime+Qk*(zprime-f2)
A(1,n)=A(1,n)+2*dcos(ease1)+2*dcos(ease2)+
+ 2*dcos(ease3)+2*dcos(ease4)+2*dcos(ease5)+
+ 2*dcos(ease6)+2*dcos(ease7)
64 continue
count=count+1
endif
endif

10 Continue

```

```

    print*, '-----'
    print*, 'Iterations'
    print*, iter
    print*, '-----'
    print*, 'Number of Points'
    print*, count
    print*, '-----'

    final=A(1,0)

    Do 101 n=0,400
    A(1,n)=dlog10((A(1,n)/final)**2)
101 Continue

    sl=rtc()
    timespent=sl-s
    dev=(dble(iter))/timespent
    print*, 'CPU Time (seconds)'
    print*, timespent
    print*, 'Iterations Per Seconds'
    print*, dev

    Do 102 n=0,400
    write (15,*) A(1,n)
102 Continue

    End
C
    Random Number Function Goes Here

```

APPENDIX C

FORTRAN Programs Used in Chapter 3

C.1. Core/Corona

CC This file will model a core/corona through random numbers.

```
      use portlib
      real*8 s, sl, timespent
      Integer*4 iter, l, number
      Integer*4 m, n
      Integer ia1, ia2, iat, ib1, ib2, ibt,
+      ic1, ic2, ict, id1, id2, idt, ie1, ie2, iet, if1, if2, ift,
+      ja1, ja2, jat, jb1, jb2, jbt, jc1, jc2, jct, jd1, jd2, jdt,
+      je1, je2, jet, jf1, jf2, jft, ka1, ka2, kat, kb1, kb2, kbt,
+      kc1, kc2, kct, kd1, kd2, kdt, ke1, ke2, ket, kf1, kf2, kft
      Real*8 OR1, OR2, A(1,401), Pi, lamda, Qi, dev,
+      Qk, Bragg, zeta, final1, B(1,118), ran2,x1, x2, x3, x4, x5,
+      x6, y1, y2, y3, y4, y5, y6, z1, z2, z3, z4, z5, z6, R1, R2,
+      R3, R4, R5, R6, l1, l2, l3, C(1,401), final2, final3, f1, f2, f3
```

CC Initilization of some of the variables and the constants.

```
      Do 1 n=0,400
      A(1,n)=0.0
      C(1,n)=0.0
1      Continue
      Pi=3.1415927
      lamda=.154242
      Bragg=2*Pi/lamda
```

CC Input statements.

```
      Print*, 'Corona Sphere Scattering'
      Print*, 'Enter the radius of the core [nm].'
      Read*, OR1
      Print*, 'Enter the radius of the corona [nm].'
      Read*, OR2
      Print*, 'Enter the number of iterations desired.'
      Read*, iter
      Print*, 'Enter the ratio of the electron denisty'
      Print*, 'of the outer shell to the core (assume 1.0).'
      Read*, l1
      s=rtc()
```

CC Iteration Loop

```
      Do 10 l=1,iter
      number=l
```


CC Draw Random Numbers

Do 2 m=0,117

B(1,m)=ran2(number)

2 continue

ia1=idint(10*B(1,100))

ia2=idint((100*B(1,100))-(10*ia1))

iat=10*ia2+ia1

ib1=idint(10*B(1,101))

ib2=idint((100*B(1,101))-(10*ib1))

ibt=10*ib2+ib1

ic1=idint(10*B(1,102))

ic2=idint((100*B(1,102))-(10*ic1))

ict=10*ic2+ic1

id1=idint(10*B(1,103))

id2=idint((100*B(1,103))-(10*id1))

idt=10*id2+id1

ie1=idint(10*B(1,104))

ie2=idint((100*B(1,104))-(10*ie1))

iet=10*ie2+ie1

if1=idint(10*B(1,105))

if2=idint((100*B(1,105))-(10*if1))

ift=10*if2+if1

ja1=idint(10*B(1,106))

ja2=idint((100*B(1,106))-(10*ja1))

jat=10*ja2+ja1

jb1=idint(10*B(1,107))

jb2=idint((100*B(1,107))-(10*jb1))

jbt=10*jb2+jb1

jc1=idint(10*B(1,108))

jc2=idint((100*B(1,108))-(10*jc1))

jct=10*jc2+jc1

jd1=idint(10*B(1,109))

jd2=idint((100*B(1,109))-(10*jd1))

jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1

jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1

ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1

kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
kct=10*kc2+kc1

kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1

ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1

kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1

x1=2.00002*OR2*B(1,iat)-1.00001*OR2
x2=2.00002*OR2*B(1,ibt)-1.00001*OR2
x3=2.00002*OR2*B(1,ict)-1.00001*OR2
x4=2.00002*OR2*B(1,idt)-1.00001*OR2
x5=2.00002*OR2*B(1,iet)-1.00001*OR2
x6=2.00002*OR2*B(1,ift)-1.00001*OR2
y1=2.00002*OR2*B(1,jat)-1.00001*OR2
y2=2.00002*OR2*B(1,jbt)-1.00001*OR2
y3=2.00002*OR2*B(1,jct)-1.00001*OR2
y4=2.00002*OR2*B(1,jdt)-1.00001*OR2

```

y5=2.00002*OR2*B(1,jet)-1.00001*OR2
y6=2.00002*OR2*B(1,jft)-1.00001*OR2
z1=2.00002*OR2*B(1,kat)-1.00001*OR2
z2=2.00002*OR2*B(1,kbt)-1.00001*OR2
z3=2.00002*OR2*B(1,kct)-1.00001*OR2
z4=2.00002*OR2*B(1,kdt)-1.00001*OR2
z5=2.00002*OR2*B(1,ket)-1.00001*OR2
z6=2.00002*OR2*B(1,kft)-1.00001*OR2

```

```

R1=dsqrt((x1**2)+(y1**2)+(z1**2))
R2=dsqrt((x2**2)+(y2**2)+(z2**2))
R3=dsqrt((x3**2)+(y3**2)+(z3**2))
R4=dsqrt((x4**2)+(y4**2)+(z4**2))
R5=dsqrt((x5**2)+(y5**2)+(z5**2))
R6=dsqrt((x6**2)+(y6**2)+(z6**2))

```

CC Test points

```

    if (R1.LE.OR2) then
    if (R1.LE.OR1) then
    Do 3 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qk=Bragg*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x1+Qk*z1)
3    continue
    else
    Do 13 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qk=Bragg*dsin(2*zeta)
    C(1,n)=C(1,n)+2*dcos(Qi*x1+Qk*z1)
13   continue
    endif
    endif

    if (R2.LE.OR2) then
    if (R2.LE.OR1) then
    Do 4 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qk=Bragg*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x2+Qk*z2)
4    continue

```

```

else
Do 14 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x2+Qk*z2)
14 continue
endif
endif

if (R3.LE.OR2) then
if (R3.LE.OR1) then
Do 5 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x3+Qk*z3)
5 continue
else
Do 15 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x3+Qk*z3)
15 continue
endif
endif

if (R4.LE.OR2) then
if (R4.LE.OR1) then
Do 6 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x4+Qk*z4)
6 continue
else
Do 16 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x4+Qk*z4)
16 continue
endif
endif

```

```

endif

if (R5.LE.OR2) then
if (R5.LE.OR1) then
Do 7 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x5+Qk*z5)
7 continue
else
Do 17 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x5+Qk*z5)
17 continue
endif
endif

if (R6.LE.OR2) then
if (R6.LE.OR1) then
Do 8 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x6+Qk*z6)
8 continue
else
Do 18 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x6+Qk*z6)
18 continue
endif
endif

10 Continue
print*, '-----'
print*, 'Radius of the Core'
print*, OR1
print*, '-----'
print*, 'Radius of the Corona'

```

```

print*, OR2
print*, '-----'
print*, 'Iterations'
print*, iter
print*, '-----'
final1=A(1,0)
final2=C(1,0)

f1=4*Pi*(OR1**3)/3
f2=4*Pi*(OR2**3)/3
f3=f2-f1
l2=l1*final1*f3/final2
l3=l2/f1
Do 101 n=0,400
C(1,n)=C(1,n)*l3
101 Continue
Do 102 n=0,400
A(1,n)=A(1,n)+C(1,n)
102 Continue
final3=A(1,0)
Do 103 n=0,400
A(1,n)=A(1,n)/final3
103 Continue
Do 104 n=0,400
A(1,n)=(A(1,n))**2
104 Continue
Do 105 n=0,400
A(1,n)=dlog10(A(1,n))
105 Continue
sl=rtc()
timespent=sl-s
dev=(dble(iter))/timespent
print*, 'CPU Time (seconds)'
print*, timespent
print*, 'Iterations Per Seconds'
print*, dev
Do 106 n=0,400
write (15,*) A(1,n)
106 Continue
dev=1
End
C
Random Number Function Goes Here

```

C.2. Core/Shell

CC This file will model a sphere with an outer shell through random numbers.

```
    use portlib
    real*8 s, sl, timespent
    Integer*4 iter, l, number
    Integer*4 m, n
    Integer ial, ia2, iat, ib1, ib2, ibt, ic1, ic2, ict, id1, id2, idt,
+   ie1, ie2, iet, if1, if2, ift, ja1, ja2, jat, jb1, jb2, jbt, jc1, jc2, jct,
+   jd1, jd2, jdt, je1, je2, jet, jf1, jf2, jft, ka1, ka2, kat, kb1, kb2,
+   kbt, kc1, kc2, kct, kd1, kd2, kdt, ke1, ke2, ket, kf1, kf2, kft
    Real*8 OR1, OR2, OR3, A(1,401), Pi, lamda, Qi, dev,
+   Qk, Bragg, zeta, final1, B(1,118), ran2, x1, x2, x3, x4, x5
+   x6, y1, y2, y3, y4, y5, y6, z1, z2, z3, z4, z5, z6, R1, R2, R3
+   R4, R5, R6, l1, l2, l3, C(1,401), final2, final3, f1, f2, f3, f4
```

CC Initilization of some of the variables and the constants.

```
    Do 1 n=0,400
        A(1,n)=0.0
        C(1,n)=0.0
1    Continue
    Pi=3.1415927
    lamda=.154242
    Bragg=2*Pi/lamda
```

CC Input statements.

```
    Print*, 'Corona Sphere Scattering'
    Print*, 'Enter the radius of the core [nm]. '
    Read*, OR1
    Print*, 'Enter the inner radius of the corona [nm]. '
    Read*, OR3
    Print*, 'Enter the outer radius of the corona [nm]. '
    Read*, OR2
    Print*, 'Enter the number of iterations desired.'
    Read*, iter
    Print*, 'Enter the ratio of the electron denisty'
    Print*, 'of the outer shell to the core (assume 1.0).'
    Read*, l1
    s=rtc()
```

CC Iteration Loop

```
    Do 10 l=1,iter
        number=l
```

CC Draw Random Numbers

```
    Do 2 m=0,117
        B(1,m)=ran2(number)
2    continue
```

ia1=idint(10*B(1,100))
ia2=idint((100*B(1,100))-(10*ia1))
iat=10*ia2+ia1

ib1=idint(10*B(1,101))
ib2=idint((100*B(1,101))-(10*ib1))
ibt=10*ib2+ib1

ic1=idint(10*B(1,102))
ic2=idint((100*B(1,102))-(10*ic1))
ict=10*ic2+ic1

id1=idint(10*B(1,103))
id2=idint((100*B(1,103))-(10*id1))
idt=10*id2+id1

ie1=idint(10*B(1,104))
ie2=idint((100*B(1,104))-(10*ie1))
iet=10*ie2+ie1

if1=idint(10*B(1,105))
if2=idint((100*B(1,105))-(10*if1))
ift=10*if2+if1

ja1=idint(10*B(1,106))
ja2=idint((100*B(1,106))-(10*ja1))
jat=10*ja2+ja1

jb1=idint(10*B(1,107))
jb2=idint((100*B(1,107))-(10*jb1))
jbt=10*jb2+jb1

jc1=idint(10*B(1,108))
jc2=idint((100*B(1,108))-(10*jc1))
jct=10*jc2+jc1

jd1=idint(10*B(1,109))
jd2=idint((100*B(1,109))-(10*jd1))
jdt=10*jd2+jd1

je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1


```

jft=iding(10*B(1,111))
jf2=iding((100*B(1,111))-(10*jft))
jft=10*jf2+jft

ka1=iding(10*B(1,112))
ka2=iding((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1

kb1=iding(10*B(1,113))
kb2=iding((100*B(1,113))-(10*kb1))
kbt=10*kb2+kbt

kcl=iding(10*B(1,114))
kc2=iding((100*B(1,114))-(10*kcl))
ket=10*kc2+kcl

kdl=iding(10*B(1,115))
kd2=iding((100*B(1,115))-(10*kdl))
kdt=10*kd2+kdt

kel=iding(10*B(1,116))
ke2=iding((100*B(1,116))-(10*kel))
ket=10*ke2+ket

kft=iding(10*B(1,117))
kf2=iding((100*B(1,117))-(10*kft))
kft=10*kf2+kft

x1=2.00002*OR2*B(1,iat)-1.00001*OR2
x2=2.00002*OR2*B(1,ibt)-1.00001*OR2
x3=2.00002*OR2*B(1,ict)-1.00001*OR2
x4=2.00002*OR2*B(1,idt)-1.00001*OR2
x5=2.00002*OR2*B(1,iet)-1.00001*OR2
x6=2.00002*OR2*B(1,ift)-1.00001*OR2
y1=2.00002*OR2*B(1,jat)-1.00001*OR2
y2=2.00002*OR2*B(1,jbt)-1.00001*OR2
y3=2.00002*OR2*B(1,jct)-1.00001*OR2
y4=2.00002*OR2*B(1,jdt)-1.00001*OR2
y5=2.00002*OR2*B(1,jet)-1.00001*OR2
y6=2.00002*OR2*B(1,jft)-1.00001*OR2
z1=2.00002*OR2*B(1,kat)-1.00001*OR2
z2=2.00002*OR2*B(1,kbt)-1.00001*OR2
z3=2.00002*OR2*B(1,kct)-1.00001*OR2
z4=2.00002*OR2*B(1,kdt)-1.00001*OR2

```

```

z5=2.00002*OR2*B(1,ket)-1.00001*OR2
z6=2.00002*OR2*B(1,kft)-1.00001*OR2

```

```

R1=dsqrt((x1**2)+(y1**2)+(z1**2))
R2=dsqrt((x2**2)+(y2**2)+(z2**2))
R3=dsqrt((x3**2)+(y3**2)+(z3**2))
R4=dsqrt((x4**2)+(y4**2)+(z4**2))
R5=dsqrt((x5**2)+(y5**2)+(z5**2))
R6=dsqrt((x6**2)+(y6**2)+(z6**2))

```

CC Test points

```

if (R1.LE.OR1) then
  Do 3 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qk=Bragg*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x1+Qk*z1)
3  continue
end if

```

```

if (R1.GE.OR3) then
if (R1.LE.OR2) then
  do 13 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qk=Bragg*dsin(2*zeta)
    C(1,n)=C(1,n)+2*dcos(Qi*x1+Qk*z1)
13 continue
endif
endif

```

```

if (R2.LE.OR1) then
  Do 4 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)
    Qk=Bragg*dsin(2*zeta)
    A(1,n)=A(1,n)+2*dcos(Qi*x2+Qk*z2)
4  continue
endif

```

```

if (R2.GE.OR3) then
if (R2.LE.OR2) then
  Do 14 n=0,400
    zeta=Pi*n/(36*400)
    Qi=Bragg*(dcos(2*zeta)-1)

```

```

Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x2+Qk*z2)
14  continue
    endif
endif

```

```

if (R3.LE.OR1) then
Do 5 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x3+Qk*z3)
5  continue
endif

```

```

if (R3.GE.OR3) then
if (R3.LE.OR2) then
Do 15 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x3+Qk*z3)
15  continue
endif
endif
endif

```

```

if (R4.LE.OR1) then
Do 6 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x4+Qk*z4)
6  continue
endif

```

```

if (R4.GE.OR3) then
if (R4.LE.OR2) then
Do 16 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x4+Qk*z4)
16  continue
endif

```

```

endif

if (R5.LE.OR1) then
Do 7 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x5+Qk*z5)
7 continue
endif

if (R5.GE.OR3) then
if (R5.LE.OR2) then
Do 17 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x5+Qk*z5)
17 continue
endif
endif

if (R6.LE.OR1) then
Do 8 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
A(1,n)=A(1,n)+2*dcos(Qi*x6+Qk*z6)
8 continue
endif

if (R6.GE.OR3) then
if (R6.LE.OR2) then
Do 18 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qk=Bragg*dsin(2*zeta)
C(1,n)=C(1,n)+2*dcos(Qi*x6+Qk*z6)
18 continue
endif
endif

10 Continue

```

```

print*, '*-----*'
print*, 'Radius of the Core'
print*, OR1
print*, '*-----*'
print*, 'Inner radius of the Corona'
print*, OR3
print*, '*-----*'
print*, 'Outer radius of the Corona'
print*, OR2
print*, '*-----*'
print*, 'Number of Iterations'
print*, iter
print*, '*-----*'
final1=A(1,0)
final2=C(1,0)

f1=4*Pi*(OR1**3)/3
f2=4*Pi*(OR2**3)/3
f3=4*Pi*(OR3**3)/3
f4=f2-f3
l2=l1*final1*f4/final2
l3=l2/f1

Do 101 n=0,400
C(1,n)=C(1,n)*l3
101 Continue
Do 102 n=0,400
A(1,n)=A(1,n)+C(1,n)
102 Continue
final3=A(1,0)
Do 103 n=0,400
A(1,n)=A(1,n)/final3
103 Continue
Do 104 n=0,400
A(1,n)=(A(1,n))**2
104 Continue
Do 105 n=0,400
A(1,n)=dlog10(A(1,n))
105 Continue
sl=rtc()
timespent=sl-s
dev=(dble(iter))/timespent
print*, 'CPU Time (seconds)'
print*, timespent

```

```

    print*, 'Iterations Per Seconds'
    print*, dev
    print*, final1
    print*, final2
    print*, f1
    print*, f4
    print*, l3

    Do 106 n=0,400
    write (15,*) A(1,n)
106 Continue
    dev=1
    End
C
    Random Number Function Goes Here

```

C.3. Layered Cylinder

CC This file will model a layered cylinder through random numbers.

```
use portlib
real*8 s, sl, timespent
Integer*4 iter, l, m, n
Integer ia1, ia2, iat, ib1, ib2, ibt, ic1, ic2, ict, id1, id2, idt,
+ ie1, ie2, iet, if1, if2, ift, ja1, ja2, jat, jb1, jb2, jbt,
+ jc1, jc2, jct, jd1, jd2, jdt, je1, je2, jet, jf1, jf2, jft,
+ ka1, ka2, kat, kb1, kb2, kbt, kc1, kc2, ket, kd1, kd2, kdt,
+ ke1, ke2, ket, kf1, kf2, kft
Real*8 OR1, OR2, halfL, A(1,401), B(1,118), C(1,401),
+ Pi, lamda, dev, l1, l2, l3, l4, l5, l6, l7, m1, m2, Qi,
+ Qj, Qk, Bragg, zeta, ran2, cut1, cut2, x1, x2, x3, x4,
+ x5, x6, y1, y2, y3, y4, y5, y6, z1, z2, z3, z4, z5, z6,
+ R1, R2, R3, R4, R5, R6, R12, S1, S2, S3, S4, S5, S6, n1,
+ final1, final2, final3, t1, t2, t3, t4, t5, volume1,
+ volume2, ratio1, ratio2, ratio3, check1, check2, check3
```

CC Initilization of some of the variables and the constants.

```
Do 1 n=0,400
  A(1,n)=0.0
  C(1,n)=0.0
1 Continue
Pi=3.1415927
lamda=.154242
Bragg=2*Pi/lamda
```

CC Input statements.

```
Print*, 'Layered Cylinder Scattering Simulation.'
Print*, 'Enter the half length of the fiber [nm].'
Read*, halfL
Print*, 'Enter half the length of the main segment [nm].'
Read*, l1
Print*, 'Enter the radius of the main segment [nm].'
Read*, OR1
Print*, 'Enter half the length of the secondary segment [nm].'
Read*, l2
Print*, 'Enter the radius of the secondary segment [nm].'
Read*, OR2
Print*, 'Assuming the electron density of the main segment to be.'
Print*, 'one. Enter the ratio of the secondary segment.'
Read*, R12
Print*, 'Enter the slice desired of the cylinder [nm].'
```

```

Print*, '(Pi times this number.)'
Read*, cut1
Print*, 'Enter the number of iterations desired.'
Read*, iter
cut2=cut1*Pi
s=rtc()
l3=(2*l1)+(2*l2)
m1=l1/l3
m2=(l1+2.0*l2)/l3

```

CC Iteration Loop

```
Do 10 l=1,iter
```

CC Draw Random Numbers

```
Do 2 m=0,117
```

```
B(1,m)=ran2(l)
```

```
2 continue
```

```

ial=idint(10*B(1,100))
ia2=idint((100*B(1,100))-(10*ial))
iat=10*ia2+ial

```

```

ib1=idint(10*B(1,101))
ib2=idint((100*B(1,101))-(10*ib1))
ibt=10*ib2+ib1

```

```

ic1=idint(10*B(1,102))
ic2=idint((100*B(1,102))-(10*ic1))
ict=10*ic2+ic1

```

```

id1=idint(10*B(1,103))
id2=idint((100*B(1,103))-(10*id1))
idt=10*id2+id1

```

```

ie1=idint(10*B(1,104))
ie2=idint((100*B(1,104))-(10*ie1))
iet=10*ie2+ie1

```

```

if1=idint(10*B(1,105))
if2=idint((100*B(1,105))-(10*if1))
ift=10*if2+if1

```

```

ja1=idint(10*B(1,106))
ja2=idint((100*B(1,106))-(10*ja1))
jat=10*ja2+ja1

```



```
jb1=idint(10*B(1,107))
jb2=idint((100*B(1,107))-(10*jb1))
jbt=10*jb2+jb1
```

```
jc1=idint(10*B(1,108))
jc2=idint((100*B(1,108))-(10*jc1))
jct=10*jc2+jc1
```

```
jd1=idint(10*B(1,109))
jd2=idint((100*B(1,109))-(10*jd1))
jdt=10*jd2+jd1
```

```
je1=idint(10*B(1,110))
je2=idint((100*B(1,110))-(10*je1))
jet=10*je2+je1
```

```
jf1=idint(10*B(1,111))
jf2=idint((100*B(1,111))-(10*jf1))
jft=10*jf2+jf1
```

```
ka1=idint(10*B(1,112))
ka2=idint((100*B(1,112))-(10*ka1))
kat=10*ka2+ka1
```

```
kb1=idint(10*B(1,113))
kb2=idint((100*B(1,113))-(10*kb1))
kbt=10*kb2+kb1
```

```
kc1=idint(10*B(1,114))
kc2=idint((100*B(1,114))-(10*kc1))
kct=10*kc2+kc1
```

```
kd1=idint(10*B(1,115))
kd2=idint((100*B(1,115))-(10*kd1))
kdt=10*kd2+kd1
```

```
ke1=idint(10*B(1,116))
ke2=idint((100*B(1,116))-(10*ke1))
ket=10*ke2+ke1
```

```
kf1=idint(10*B(1,117))
kf2=idint((100*B(1,117))-(10*kf1))
kft=10*kf2+kf1
```

```

x1=2.00002*B(1,iat)-1.00001
x2=2.00002*B(1,ibt)-1.00001
x3=2.00002*B(1,ict)-1.00001
x4=2.00002*B(1,idt)-1.00001
x5=2.00002*B(1,iet)-1.00001
x6=2.00002*B(1,ift)-1.00001
y1=2.00002*B(1,jat)-1.00001
y2=2.00002*B(1,jbt)-1.00001
y3=2.00002*B(1,jct)-1.00001
y4=2.00002*B(1,jdt)-1.00001
y5=2.00002*B(1,jet)-1.00001
y6=2.00002*B(1,jft)-1.00001
z1=2.00002*B(1,kat)-1.00001
z2=2.00002*B(1,kbt)-1.00001
z3=2.00002*B(1,kct)-1.00001
z4=2.00002*B(1,kdt)-1.00001
z5=2.00002*B(1,ket)-1.00001
z6=2.00002*B(1,kft)-1.00001

```

```

R1=dsqrt((x1**2)+(z1**2))
R2=dsqrt((x2**2)+(z2**2))
R3=dsqrt((x3**2)+(z3**2))
R4=dsqrt((x4**2)+(z4**2))
R5=dsqrt((x5**2)+(z5**2))
R6=dsqrt((x6**2)+(z6**2))
S1=dabs(y1)
S2=dabs(y2)
S3=dabs(y3)
S4=dabs(y4)
S5=dabs(y5)
S6=dabs(y6)

```

CC Test points

```

if (S1.LE.1.0) then
  if (R1.LE.1.0) then
    l4=halfL*y1
    l5=l4/l3
    l6=dabs(l5)
    l7=l6-dint(l6)
    if (l7.LE.m1.or.l7.GE.m2) then
      do 14 n=0,400
        zeta=Pi*n/(36*400)
        Qi=Bragg*(dcos(2*zeta)-1)
        Qj=Bragg*dcos(cut2)*dsin(2*zeta)

```

```

Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x1*OR1)+(Qk*z1*OR1)+(Qj*I4)
A(1,n)=A(1,n)+2*dcos(n1)
14 continue
else
do 15 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x1*OR2)+(Qk*z1*OR2)+(Qj*I4)
C(1,n)=C(1,n)+2*dcos(n1)
15 continue
endif
endif
endif

if (S2.LE.1.0) then
if (R2.LE.1.0) then
l4=halfL*y2
l5=l4/l3
l6=dabs(l5)
l7=l6-dint(l6)
if (l7.LE.m1.or.l7.GE.m2) then
do 24 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x2*OR1)+(Qk*z2*OR1)+(Qj*I4)
A(1,n)=A(1,n)+2*dcos(n1)
24 continue
else
do 25 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x2*OR2)+(Qk*z2*OR2)+(Qj*I4)
C(1,n)=C(1,n)+2*dcos(n1)
25 continue
endif
endif
endif

```

```

if (S3.LE.1.0) then
if (R3.LE.1.0) then
l4=halfL*y3
l5=l4/l3
l6=dabs(l5)
l7=l6-dint(l6)
if (l7.LE.m1.or.l7.GE.m2) then
do 34 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x3*OR1)+(Qk*z3*OR1)+(Qj*l4)
A(1,n)=A(1,n)+2*dcos(n1)
34 continue
else
do 35 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x3*OR2)+(Qk*z3*OR2)+(Qj*l4)
C(1,n)=C(1,n)+2*dcos(n1)
35 continue
endif
endif
endif

if (S4.LE.1.0) then
if (R4.LE.1.0) then
l4=halfL*y4
l5=l4/l3
l6=dabs(l5)
l7=l6-dint(l6)
if (l7.LE.m1.or.l7.GE.m2) then
do 44 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x4*OR1)+(Qk*z4*OR1)+(Qj*l4)
A(1,n)=A(1,n)+2*dcos(n1)
44 continue
else

```

```

do 45 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x4*OR2)+(Qk*z4*OR2)+(Qj*I4)
C(1,n)=C(1,n)+2*dcos(n1)
45 continue
endif
endif
endif

if (S5.LE.1.0) then
if (R5.LE.1.0) then
l4=halfL*y5
l5=l4/l3
l6=dabs(l5)
l7=l6-dint(l6)
if (l7.LE.m1.or.l7.GE.m2) then
do 54 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x5*OR1)+(Qk*z5*OR1)+(Qj*I4)
A(1,n)=A(1,n)+2*dcos(n1)
54 continue
else
do 55 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
n1=(Qi*x5*OR2)+(Qk*z5*OR2)+(Qj*I4)
C(1,n)=C(1,n)+2*dcos(n1)
55 continue
endif
endif
endif

if (S6.LE.1.0) then
if (R6.LE.1.0) then
l4=halfL*y6
l5=l4/l3

```

```

l6=dabs(l5)
l7=l6-dint(l6)
if (l7.LE.m1.or.l7.GE.m2) then
do 64 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
nl=(Qi*x6*OR1)+(Qk*z6*OR1)+(Qj*l4)
A(l,n)=A(l,n)+2*dcos(nl)
64 continue
else
do 65 n=0,400
zeta=Pi*n/(36*400)
Qi=Bragg*(dcos(2*zeta)-1)
Qj=Bragg*dcos(cut2)*dsin(2*zeta)
Qk=Bragg*dsin(cut2)*dsin(2*zeta)
nl=(Qi*x6*OR2)+(Qk*z6*OR2)+(Qj*l4)
C(l,n)=C(l,n)+2*dcos(nl)
65 continue
endif
endif
endif

10 Continue
final1=A(l,0)
final2=C(l,0)
t1=halfL/l3
t2=dint(t1)
t3=t1-t2
t4=t3*l3
t5=l1+(2*t2)

if (t4.LE.l1) then
volume1=(4*t2*t1*OR1)+(2*t4*OR1)
volume2=4*t2*t2*OR2
elseif (t4.GT.t5) then
volume1=(4*t2*t1*OR1)+(2*t1*OR1)+2*OR1*(t4-t5)
volume2=(4*t2*t2*OR2)+(4*t2*OR2)
else
volume1=(4*t2*t1*OR1)+(2*t1*OR1)
volume2=(4*t2*t2*OR2)+2*OR2*(t4-l1)
endif

```

```

ratio1=R12*final1*volume2/final2
ratio2=ratio1/volume1
ratio3=ratio2/R12
check1=volume1/OR1
check2=volume2/OR2
check3=(check1+check2)/2.0

do 102 n=0,400
C(1,n)=C(1,n)*ratio2
102 continue
do 103 n=0,400
A(1,n)=A(1,n)+C(1,n)
103 continue
final3=A(1,0)

do 104 n=0,400
A(1,n)=A(1,n)/final3
104 continue
do 105 n=0,400
A(1,n)=((A(1,n))**2)
105 continue
do 106 n=0,400
A(1,n)=dlog10(A(1,n))
106 continue
sl=rtc()
timespent=sl-s
dev=(dble(iter))/timespent
print*, 'CPU Time (seconds)'
print*, timespent
print*, 'Iterations Per Seconds'
print*, dev
print*, '-----'
print*, halfL
print*, check3
print*, '-----'
print*, ratio3
do 107 n=0,400
write (15,*) A(1,n)
107 continue

```

End

C

Random Number Function Goes Here