CHARACTERIZATION OF SMALL PARTICULATE MATTER BY USING AN INDIGENOUSLY DESIGNED LASER BASED PROBE AND DEVELOPMENT OF RELEVANT SOFTWARE

A thesis submitted in partial fulfillment of the requirements for award of the degree of Doctor of Philosophy

Ankur Gogoi

Registration No: 011 of 2008

Department of Physics
School of Science & Technology
Tezpur University
Napaam, Tezpur - 784 028
Assam, India

June, 2011

Characterization of small particulate matter by using an indigenously designed laser based probe and development of relevant software

Abstract

The light scattering behaviour of an isolated spherical or nonspherical particle, or an aggregate of particles, where the particle size ranges from micrometer to nanometer, presents numerous theoretical and experimental challenges. These particles are normally found as aerosols, suspended particles in solutions or as embedded particles especially nanoparticles in an optically transparent media. Optical characterization of such particles is very important for a wide variety of applications, which include optical diagnostics for industrial aerosol processes and combustion, environmental issues (e.g., visibility and haze problems), remote atmospheric sensing (lidar), astrophysical issues such as the exploration and characterization of particulate matter in different planetary atmospheres and most recently nanoscience issues (e.g., characterization of nanoparticles by optical tools) [1, 2, 49, 50, 99, 167, 278 - 282, 325]. The scattering properties of such particles are determined not only by their bulk optical properties and the medium but also by the shape and dispersion of shapes of particles, size and dispersion of sizes of particles, density, quality of particle surfaces etc. The light scattered by such small particulate matter also depends on the scattering angle and the polarization state of the incident and scattered light [1 - 5, 67]. The study of angular scattering dependency of small particulate matter is very important as such results helps to predict the presence of the scattering particle and for a better understanding of radiation transfer through a medium containing the scatterer [32]. In recent years various theoretical calculations using Mie theory, separation of variable method (SVM), finite difference time domain method (FDTDM), Waterman's T-matrix method,

discrete dipole approximation (DDA), geometrical optics approximation (GOA) etc. have been carried out in an attempt to explain experimentally observed light scattering patterns due to particulate matter [4, 5, 140, 143, 369]. Despite the availability of such advanced numerical techniques, the results of laboratory and in situ experiments still do not agree with the theories exactly. This is because of limitations of these numerical methods to model the light scattering properties of natural particles having too complicated structures with sufficient accuracy as well as due to inaccuracies in the instruments used in the experiments. Combination of computational and experimental approaches can create synergy to the benefit of each of them [4, 99]. Hence in this PhD work an attempt was made to both improve on the theory as well as instrumentation associated with light scattering by particulate matter.

We designed and fabricated a laser based laboratory light scattering instrument that used an array of 16 static Si photodetectors and could be operated at three different incident wavelengths (543.5 nm, 594.5 nm and 632.8 nm) [249, 251, 371]. The whole array of 16 detectors could be rotated simultaneously about an axis perpendicular to the plane of the circular disc. The instrument was designed to investigate natural and artificial small particulate matter, in order to find their scattering properties in terms of the first volume scattering function, $\beta(\theta)$ and the linear polarization ratio, $P(\theta) = -\frac{S_{12}}{S_{11}}$ over the scattering angle range from 10^0 to 170^0 in steps of 1^0. The light scattered from the samples passed through a set of collection optics comprising of appropriate analyzers and were sensed by the radial photodetector array, connected to a high gain, low noise amplifier. The amplified signals were interfaced to a dedicated data acquisition system for data recording. The data obtained was then plotted and analyzed. The accuracy and the reliability of the setup was ensured by conducting measurements on the light scattering properties of water droplets and polystyrene spheres suspended in water [249] and comparing the results with theoretical Mie calculations.

Using the above mentioned light scattering setup extensive experiments were performed on different small particles (such as titania, ice analogue crystals, graphite etc.), nanoparticles (such as ZnS semiconductor nanoparticles, bentonite nanoclay etc.) and tropical hydrosols (diatom - a unicellular biological microorganism). Titania (TiO_2), graphite and bentonite nanoclay particles were collected from different chemical agencies. Ice analogue crystals were prepared in the laboratory using the procedure as described by Ulanowski [61]. For light scattering measurements titania particles were suspended in water held in a cylindrical pyrex glass cuvette [249]. Graphite and ice analogue crystals were sprayed above the scattering centre by using an indigenously designed nebulizer. while bentonite caly particles were embedded in a cylindrical transparent polymer matrix made of epoxy resin. On the other hand ZnS semiconductor nanoparticles were prepared in the laboratory by using a simple chemical route [252, 256, 258]. ZnS nanoparticles were embedded in a rectangular transparent polymer matrix made of Polyvinyl Alcohol (PVA) for the light scattering measurements. Tropical fresh water diatoms were grown in the laboratory by using "WC" culture media proposed by Guillard and Lorenzen (1972) with slight modifications [250, 271]. The siliceous frustules of the diatoms were extracted by acid treatment [370]. Both the diatoms and the frustules were suspended in water in a similar manner as titania particles for light scattering experiments [250]. The detailed results and the discussions about the measurements on these scattering samples are presented in this thesis.

As a mathematical and numerical basis for the analysis of observed data from the experiments with aerosols, hydrosols and nanoparticles, Mie theory [100-102] and Waterman's T-matrix approach [158 -160] were used as they are the most powerful tools for solving light scattering problems for homogeneous and composite particles of sizes not too large compared to the wavelength of incident light [1 - 5]. It is worth mentioning that Mie theory is applicable to spherical particles whereas T-matrix approach is capable of simulating the light scattering patterns of nonspherical particles also. The thesis reports the

development of two accurate and reliable computer programs TUMiescat.c [344] and TUTscat.c based on Mie theory and T-matrix approach respectively. The programs were developed in standard C language and an attempt was made to optimize on the speed and accuracy. TUMiescat.c is an improved version of Bohren and Huffmann's [2] fortran code bhmie.f whereas TUTscat.c is a modified version of Mishchenko's 'tmd.new.f' code [5, 238] which is freely available in the NASA website (ftp://ftp.giss.nasa.gov/pub/crmim/tmd.new.f).

Next an analytical software package TUSCAT incorporated with a graphical user interface (GUI) was developed in java platform for modeling electromagnetic scattering from virtual small particles and also to yield characteristic properties of real particles from experimental data [350]. The source codes behind the light scattering calculations for spherical and nonspherical particles are based on TUMiescat.c and TUTscat.c respectively. Its interactive features enable the user to observe the changes in output scattering properties in real time. In addition to its ease of use, it has high computational accuracy, efficiency, reliability and adaptability. The software was used for the extensive analysis of the experimental results.

The thesis is broadly divided into five chapters. The back ground of the work is contained in Chapter - 1 in the form of introduction. Chapter - 2 discusses about Mie theory and T-matrix approach and the computer programs developed on the basis of these theories to explain the experimentally observed light scattering results from different samples. The detailed design and fabrication of the experimental light scattering setup along with the associated data acquisition and analytical software are illustrated in Chapter - 3. Chapter - 4 presents the results of the light scattering measurements from different hydrosols, nanoparticles and aerosols and highlights the relevant discussions. Chapter - 5 is devoted to briefly summarize the whole work followed by future directions. References and publications followed by addenda are given towards the end of the thesis.

Declaration

I hereby declare that the thesis entitled "**Characterization of small particulate matter by using an indigenously designed laser based probe and development of relevant software**", submitted to the School of Science and Technology, Tezpur University in partial fulfillment for the award of the degree of Doctor of Philosophy in Physics, is a record of original research work carried out by me. Any text, figures, theories, results or designs that are not of own devising are appropriately referenced in order to give credit to the original author(s). All sources of assistance have been assigned due acknowledgement. I also declare that neither this work as a whole nor a part of it has been submitted to any other university or institute for any degree, diploma, associateship, fellowship or any other similar title or recognition.

Ankur Gogoi

Date: 21-06-2011

(Ankur Gogoi)
Department of Physics
School of Science & Technology
Tezpur University
Tezpur-784028, Assam, India

Place: Tezpur

Tezpur University

Certificate

This is to certify that the thesis entitled **"Characterization of small particulate matter by using an indigenously designed laser based probe and development of relevant software"** submitted to the School of Science and Technology, Tezpur University in partial fulfillment for the award of the degree of Doctor of Philosophy in Physics is a record of research work carried out by **Mr. Ankur Gogoi** under our joint supervision and guidance.

All help received by him from various sources have been duly acknowledged.

No part of this thesis has been submitted elsewhere for award of any other degree.

21.6.2011

(Prof. A. Choudhury)
Signature of Associate Supervisor
Designation: Professor
School: Science and Technology
Department: Physics
Date: 21.6.2011
Place: Tezpur-784028, Assam, India

(Dr. G. A. Ahmed)
Signature of Principal Supervisor
Designation: Professor
School: Science and Technology
Department: Physics
Date: 21.6.2011
Place: Tezpur-784028, Assam, India

I would like to dedicate this thesis
to
my beloved family

Acknowledgements

It is a matter of immense pleasure and fortune for me to express deep sense of gratitude to my Principal Supervisor Dr. Gazi A. Ahmed, Department of Physics, Tezpur University and Associate Supervisor Prof. Amarjyoti Choudhury, Department of Physics, Tezpur University for their dynamic and scrupulous supervision that I have received throughout the endeavor of my PhD work at Tezpur University. This work would not have been possible without the invaluable suggestions and discussions of Dr. Gazi A. Ahmed and Prof. Amarjyoti Choudhury. By my few words, I find it difficult to thank my Supervisors for the pain they took and their nonstop effort throughout my Ph.D. work. I take this opportunity to express my intense reverence towards them for the extensive scientific discussions and for giving me the freedom in research. Needless to say, I shall be highly obliged to them for all the time.

I appreciate having capitalized on this great opportunity I have been given at Tezpur University. I take the opportunity to thank Prof. Jayanta Kumar Sarma, Prof. A. Kumar, Dr. Dambarudhar Mohanta, Dr. Nidhi Saxena Bhattacharyya, Dr. Nilakshi Das, Dr. Kishore Barua and Dr. Pritam Deb of Department of Physics, Tezpur University for their encouragement, criticism and inspiration to carry out this work. I would like to thank G. A. Stanciu of Center for Microscopy - Microanalysis and Information Processing, University "Politechnica" of Bucharest, Romania; Pritom Rajkhowa, Computer Centre, Tezpur University; Prof Niranjan Karak, Gautam Das, and Lakhyajyoti Borthakur, Department of Chemical Sciences, Tezpur University; Prof. Alak K. Buragohain and Ranjan Dutta Kalita, Department of Molecular Biology and Biotechnology, Tezpur University; Nirmal Mazumder, Institute of Biophotonics, National Yang-Ming University, Taiwan for their fruitful discussions and valuable suggestions during my enrite Ph.D. work. Special thanks are extended to Mr. Ratan Baruah for SEM characterizations. I acknowledge the help

extended by Mr. Ajanta Pathak, Mr. Umesh Patir, Mr. Narayan Sarma, Mr. Prakash Kurmi and Biju in official work.

I would like to thank all my seniors, juniors, colleagues and friends in Tezpur University who have continuously supported me throughout these years. I take this opportunity to thank Smriti ba, Sovan, Debashis, Sanchita ba, Durga da, Jojo, Sayan, Mayuri Devi, Namita, Nayanmoni, Nibedita, Bidyut, Arup, Manjit, Surojit, Samrat, Jyoti for their company, help and goodwill. Special thanks to members of "Reserachhelp Google Group" for their help and support specially in searching research articles for my Ph.D. work. I would like to thank the research scholars of Dept. of Physics, Tezpur University for their help and support throughout my work.

I would like to acknowledge Council of Scientific and Industrial Research, New Delhi for financial support as Senior Research Fellow (Sanction no: 09/796/(0013)/2009/EMR-I) which helped me to carry on my research work and reach this stage.

I am indebted to my parents, who showed me this beautiful world and always supported me in all my ups and downs throughout my life. I sincerely acknowledge my elder brothers, sisters-in-law, Bina Rajkonwari, Mayuri Rajkumari and Prayash (Zypsy) for their constant inspiration, love and moral support.

Ankur Gogoi

Date: 21-06-2011 *(Ankur Gogoi)*

Contents

List of tables

List of figures

List of plates

Chapter I: Introduction

1.1. General introduction to light scattering

The phenomenon of light scattering is applied for probing the characteristics of isolated spherical and nonspherical particles where the particle size ranges from micrometer to nanometer. This technique has a high relevance in diverse fields like atmospheric science, oceanography, astronomy, and engineering sciences with specific applications in remote sensing, light detection and ranging (lidar) systems, environmental monitoring (e.g., visibility and haze problems), measurement of air quality parameters, meteorology, optical diagnostics for industrial aerosol processes and combustion, climate modeling, ocean optics, astrophysical issues such as the exploration and characterization of particulate matter in different planetary atmospheres and most recently nanoscience issues (e.g., characterization of nanoparticles by optical tools) [1 - 7].

Light scattering may be defined as the change in direction of the incident radiation (or part of the radiation) due to the presence of an obstacle or spatial variations in the optical properties of matter (refractive index, permittivity) in the vicinity of incoming radiation. The mechanism by which light is scattered may be considered as reflection and refraction at the air-solid surface, refraction due to transition of material properties (permittivity, refractive index) over distances, and diffraction of the light around the boundaries of the opaque object [8, 9]. Above all, scattering of electromagnetic radiation or light scattering is the consequence of light-matter interaction. All matter is composed of discrete electric charges: electrons and protons. An incident electromagnetic field forces these elementary particles (mainly electrons) within matter, which could be a single electron, an atom or molecule, a solid or liquid particle, to oscillate at the same frequency as the electric field of the incident radiation. The oscillation or perturbation of the electron cloud results in a periodic separation of the charges

within the molecule, which is called an induced dipole moment. The oscillating induced dipole moment acts as a source of secondary electromagnetic radiation in all directions. This secondary radiation is the radiation scattered by the obstacle. The scattering is said to be elastic or coherent if the frequency of the scattered light coincides with the frequency of the incident light (e.g. Rayleigh scattering and Mie scattering) and inelastic if the radiation is scattered with a new frequency (for example Raman scattering, fluorescence, luminescence, etc.). During the scattering process a part of the incident energy may be absorbed by the scattering particle and converted into other forms of energy (such as thermal energy). As a result of absorption and scattering, the energy of the incident beam is reduced by an amount equal to the sum of absorbed and scattered energy. This reduction is called extinction. Thus,

$$\text{Extinction} = \text{Scattering} + \text{Absorption}$$

The light scattering process is usually divided into two parts: single scattering and multiple scattering. In most practical cases the scattering particles are grouped together and when the particles are sufficiently separated (distance between the particles is larger than 3 times the radius of one particle) [3, 6, 10], no interference occurs between the light scattered by different particles or the interference of light scattered by different particles are undetectable (e.g. light scattering in planetary atmosphere). This is the case of single scattering where the total scattered intensity is the sum of the intensities scattered by each particle. In the case of multiple scattering, the particle separation is smaller than the wavelength of the incident radiation and the individual scattered signals are no longer independent and undergoes successive scattering by other neighboring particles (for example light scattering by the suspended fat globules in milk). Again, the entire process of elastic light scattering may be divided into three basic regions according to the size of the scatterer: the particle size is much smaller than the wavelength of incident light (Rayleigh scattering region), the particle size is comparable to the wavelength (Mie scattering region) and the particle size is much larger than the wavelength (geometrical optics region).

The work reported here is confined to single independent elastic scattering by randomly oriented axially symmetric particles with a special reference to absorption that needs to be taken into account in the Mie scattering region.

1.2. Light scattering and absorption by small particles (aerosols and hydrosols) and their importance

The light scattering patterns of small particulate matter contains characteristic information about their physical and optical properties. These particles are normally found as aerosols, suspended particles in solutions (hydrosols) or as embedded particles, especially nanoparticles in optically transparent media. The intensity of light scattered by a particle (or ensemble of particles) is a function of the angle between the incident and scattered radiation. This angular dependence is also a function of size (and dispersion of sizes) of particles, shape (and dispersion of shapes) of particles, optical properties of particles (refractive index, permittivity, absorption), particle orientation, incident wavelength, polarization of the incident wave, density, structure of aggregates (fluffy, fractal, dense, etc.), and quality of particle surfaces (roughness, buffing, etc.) [1 - 6]. It is very important to study the angular scattering dependency of small particulate matter as such results contain information by which the particle may often be classified or even identified and helps for better understanding of radiation transfer through a medium containing the scatterer [32].

Earth's atmospheric particulate constituents, i.e., aerosols, water vapor droplets, raindrops, snowflakes and ice particles, scatter and absorb both the incoming solar radiation and the radiant energy leaving the earth towards space thereby influencing the radiative balance of the atmosphere and the climate to a great extent [2, 4 - 6, 12]. Aerosol, which is defined as a form of colloid, in which microscopic particles of highly polymerized aggregate of liquid or solid substances are lumped together and are dispersed in a gas, normally possess complicated morphologies and are neither much smaller nor much larger than

the wavelength of optical radiation. True aerosol particles range from 10^{-7} to 10^{-4} cm in diameter, though turbulent media can keep particles 100 times larger in dispersion. A large mass fraction of the aerosols in the earth's atmosphere consists of irregular mineral dust particles [13]. Desert regions, such as the Sahara, are the main source of mineral aerosols on earth. Moreover some of the particles enter the atmosphere from space (cosmic aerosol) [14]. The amount of aerosol can greatly increase during strong volcanic eruptions, as occurred after the Mount Pinatubo (Philippines) eruption in June 1991 [15] and the Iceland volcano Eyjafjallajokul in March-April, 2010. Such eruptions inject volcanic (silicate) ash and sulfate aerosols into the Earth's atmosphere and thereby change the composition of the atmosphere significantly. Again the salt particles with sizes from approximately 0.1 to 1 μm formed by the bursting of bubbles at the ocean surface, generally termed as sea-salt aerosol (SSA), are also an inherent part of the total aerosol concentration. Some more aerosol emission sources are aerosol formed in the atmosphere due to gas to particle conversion, fuel combustion (soot, diesel exhaust particles), forest, grass, or other types of fires (black carbon), biological material present in the atmosphere in the form of pollens, fungal spores, bacteria, viruses, insects, fragments of plants and animals etc. [13].

Research on the light scattering properties of aerosols, which can exist in the atmosphere for a long time, is not only essential for climate modeling but also for remote sensing, detecting regional air pollution (visibility, photochemistry, health effects) and developing atmospheric correction algorithms to remove scattered light from space borne data [16]. Moreover research on the properties of terrestrial aerosols are of great astronomical importance as their properties are found to be similar to the properties of planetary regolith, cometary dust etc. [13, 17, 18]. The interstellar extinction curve was found to fit very well with the dust grains comprising of silicate and graphite spheres with power law grain size distribution [19, 20, 326, 327]. Because it is very difficult to collect such extraterrestrial particles, (polarized) light scattering is an indispensable

source of information in many astronomical investigations. *In-situ* investigation on the composition of the cometary dust particles was performed by the dust impact mass spectrometer CIDA onboard the NASA spacecraft STARDUST and provided evidence that the interstellar dust is a mixture of silicate and carbonaceous materials of highly irregular shape and are also highly porous in nature (fluffy) [21-23]. Observations and simulations of the light scattered by dust particles in cometary comae, interplanetary space and planetary regolith (or analogous terrestrial dust aggregates) is necessary to deduce the physical properties of their constituent particles [24] and may lead to a better understanding of the formation of solar system. Also, knowledge of aerosol optical properties assumes significant importance in the wake of studies strongly correlating airborne particulate matter with adverse health effects. Through the mid 1990s considerable attention was paid to study the scattering properties as well as the size, shape and orientation of spherical and nonspherical aerosols having size comparable to the probing wavelength (visible or UV) [33, 92 - 94]. Such experiments and theoretical calculations not only yielded highly structured light scattering patterns and their dependence on the physical properties like particle size, and the index of refraction but gave valuable information to aid the interpretation of satellite remote sensing data as well as cometary observations [32 - 50].

Another important constituent of Earth's atmosphere is water vapor and is generated mainly by evaporation from water and transpiration from plant leaves. Recent measurements of water vapor droplets in controlled environments, revealed characteristic details of water droplet size distribution in air having different levels of O_2 or CO_2 and discussed about their importance in the formation of rain clouds [1, 52, 53]. Clouds form when a volume of air becomes supersaturated with water vapor and the excess water condenses rapidly on aerosol particles to form droplets. The aerosol particles act as cloud condensation nuclei (CCN). The enhancement of the atmospheric concentration of CCN affects the radiative properties of clouds (indirect aerosol forcing) [51]. It

is noteworthy to mention that a significant number of cloud particles in the Earth's atmosphere are ice crystals. Clouds play a vital role in the Earth's atmosphere radiative balance [25 - 31, 54 - 61]. Basic scattering, absorption, and polarization data for cloud particles are therefore required for reliable modeling of their radiative properties for incorporation in climate models, for interpretation of the observed bidirectional reflectances, fluxes, and heating rates from the air, ground and space, and for the development of remote sensing techniques to infer cloud optical depth, temperature, and ice crystal size [4, 28, 97]. Field measurements of scattering from ice in clouds is complicated and for correct interpretation of the data the knowledge of the size and shape of the observed ice crystals is required. This information is imprecise, especially for smaller sizes. It is also difficult to perform laboratory measurements on ice crystals because it requires the particles to be maintained in equilibrium with water vapor at low temperatures. Moreover, control over ice crystal morphology is poor [59, 61, 95, 96]. However reports of the laboratory measurements of ice analogue crystals (having same shape and refractive index) showed promising resemblance between the measured functions and the analytic phase function for ice clouds [58, 60, 61].

Natural water, both fresh and saline, are a diverse mixture of dissolved and particulate matter such as colloids, bacteria, phytoplankton, detritus (organic nonliving particles), minerogenic particles, ions and gas bubbles and in general are termed as hydrosols. Thus water is a complex medium containing varieties of hydrosols that have varying concentration and composition depending on time and location of the water body. As water bodies cover about 70% of the surface of Earth, it significantly affects the planetary albedo and climate [62, 63, 66]. Water molecules themselves exhibit a complex absorption spectrum and a significant amount of scattering as a result of refractive index fluctuations due to the occurrence of small variations in salt concentration [64]. For instance, backscattering by clean ocean water can alone contribute as much as 80% of the total backscattering coefficient in the blue spectral region. However, during

phytoplankton blooms and atmospheric dust deposition, the backscattering coefficient can be dominated by plankton species and high refractive index mineral particles respectively [62]. It is noteworthy to mention that the information about the ecology, particularly the growth and decay of phytoplankton bloom, radiance from water bodies, marine environment, water quality in lakes and rivers are constantly monitored by satellite remote sensing [65]. In this context, it is extremely important to know the scattering properties of the constituent hydrosols as a function of scattering angle and their variation for different water types (e.g. turbid and/or eutrophic inland water) as such results can be used to investigate the nature of the particles and to predict underwater light climate [63,67]. Recent studies on the extinction, absorption and backscattering efficiency of marine and saline water and the hydrosols, provided excellent information which would be very much useful for the processing and the interpretation of satellite imagery and to develop the radiative transfer algorithm for the study of the ecosystems [68 - 78].

In the last few years, a lot of effort has been made in the development of the science and technology at the nanometer scale, covering from growth and characterization of such nanometer sized particles to device processing. In particular, the shape and size of such low-dimensional structures are crucial parameters to determine their physical, optical as well as electronic properties [79-84]. Semiconductor and metal nanoparticles attract strong interest because of their potential technological applications in sub-wavelength photonic and optoelectronic devices, for nonlinear optics, for optical data storage, for surface-enhanced spectroscopy and for biological labelling and sensing [85]. Although near-field optical techniques are the natural choice for studying particle plasmon interactions and field enhancement at sub-wavelength scales, for sufficiently larger inter-particle distance compared to the optical wavelength, far-field optical methods (laser light scattering, photoluminescence, cathode-luminescence, Raman scattering etc.) can provide valuable information about their complex characteristics and interactions [86]. Among the far field methods, light scattering

techniques are the most versatile and useful set of techniques for *in situ* morphological measurement, size distribution measurement and (in some cases) the shapes of nanoparticles suspended in optically inactive solution [81]. Recently, particle sizing on the nanometer scale, far exceeding the diffraction limit of UV radiation, has been published [79]. Experimental and theoretical work on identifying nanospheres using light scattering spectroscopy (LSS) in the UV range (λ~250-390 nm) has been reported [82]. Also, Mie scattering from nanoparticles with different particle sizes and shapes have been studied [85, 86]. Light scattering by metallic nanoparticles and nanowires near plasmon resonance frequencies, their possible biological applications were also studied by many workers [87 - 91].

1.3. Definition of the problem and objectives of the present work

As it was mentioned in the previous section, in recent years, a tremendous amount of experimental work has been done to study the electromagnetic scattering by spherical and nonspherical particles. Various theoretical approaches which involve computational techniques were also performed in the recent past to explain these experimentally observed light scattering patterns. Some of these techniques are Mie theory, Waterman's T-matrix method, finite difference time domain (FDTD) method, discrete dipole approximation (DDA) etc. [4, 5, 6]. Despite the varieties of theoretical simulations and experimental measurements of visible and infrared light scattering, which were performed during the past to determine the phase function and extinction, polarization and depolarization characteristics of natural and artificially generated cirrus cloud crystals, scattering matrix elements of hydrosols, backscattering Mueller matrix for atmospheric aerosols and hydrosols etc., the complete structure, size, composition and optical parameters of thousands of aerosols, hydrosols and other particulate matters are yet to be done. Again, as the shape and structure of the natural particles are too complex, it is also important to collect adequate information about the morphology of such particles in order to calculate their

light scattering properties with sufficient accuracy. Significant discrepancies between the experimental results and the theoretical predictions are generally observed because the real situation in nature can be very complex in terms of particle shapes, size distribution, refractive index etc. and cannot be modeled exactly as the real situation. In order to improve the usefulness, performance and quality of the light scattering theories, it is crucial to have more measured light scattering data with different particle shapes and size ranges. Conversely measurements also need to be compared with simulations to calibrate the equipment [98]. The combination of theoretical and experimental techniques are capable of creating synergy to benefit each of them by giving useful information about the light scattering from many types of natural and artificial particulate matter [99]. In the context of light scattering studies on nanoparticles, it is noteworthy to mention that the standard techniques used to characterize micron and sub-micron size particles do not have the fine resolution needed for nano-sized particle detection for which exploration of nanoparticles by light scattering tools has not yet been realized to their fullest potential.

Therefore to improve on the theory and measurement techniques involved in light scattering studies, we need:

- efficient instrumentation to measure the light scattering properties of different particulate matter including nanoparticles,
- more precise experimental error reduction procedure,
- complete information about the morphology (size and dispersion of sizes, shape and dispersion of sizes, structure of the aggregates) of the scattering particles by using scanning electron microscopy, transmission electron microscopy or optical microscopy,
- information about the optical properties (refractive index, permittivity) of the scattering particles,
- fast, accurate and more powerful algorithms to compute light scattering patterns of particles with complex shapes and extreme refractive indices.

The problem of designing and utilizing such a system and developing the theory to execute the investigation process is to be tackled. The primary objectives of this PhD thesis, as it was titled, focuses on the following parts:

- Design, fabricate, calibrate and use an efficient and reliable laser based system using He-Ne and diode laser, for light scattering studies.
- Performance optimization and scale-up of the setup by calibrating with standard samples.
- Use the setup for experimental investigations of aerosols and other suspended particulate matter.
- Use the system for experimental investigations of nanoparticles (ZnS).
- Develop Mie theory for light scattering computations from monodisperse and polydisperse particle systems of spherical shape.
- Use T-matrix approach for calculating the scattering matrix elements of monodisperse and polydisperse nonspherical particles.
- Development of analytical software using Mie theory and T-matrix approach for comparative analysis with experimental results to measure the efficiency of the theoretical models.

Before giving a detailed description of these objectives, we first review the basic experimental and theoretical techniques and software used in light scattering studies.

1.4. Light scattering theories

In recent years, a tremendous amount of progress has been made in the study of electromagnetic scattering by spherical and nonspherical particles [1 -5, 140, 142, 325 - 328]. Spherical particles, with sizes comparable to the incident wavelength, are objects that are easiest to describe using classical Mie theory [100 - 104]. This approach provides an exact way to calculate scattering patterns, which reflect the geometrical properties of particulate matter. However the

complexity of the theory of scattering, for shapes other than spherical, made similar computations impossible in the mid 1990s as the scattering properties of nonspherical particles can differ quantitatively and even qualitatively from those of volume- or surface- equivalent spheres [4, 5, 105, 106]. Several attempts were made at that time to develop numerical approaches by using semi-empirical approach through, a modification of the Mie solution for spherical particles, ray tracing technique, etc. [107, 108]. However, some remarkable optical phenomena, such as halos, arcs, pillars and zenith-enhanced backscatter observed for ice crystals using lidar, single scattering with lidar and radar depolarization observed for cirrus clouds etc., could not be described by these techniques. Hence, the last two decades have demonstrated a major effort aimed at a much better understanding of the effect of nonsphericity on light scattering, resulting in the development of several new numerical techniques and approximations for computing the scattered electromagnetic radiation [3–6]. Several methods that have found extensive applications for light scattering investigations on particles of sizes comparable to the incident wavelength are outlined below:

- **Mie theory:** The light scattering properties of single spherical particles with both real and complex indices of refraction have been worked out in detail by use of Mie (1908) theory [1 - 3]. It is an exact analytical solution of Maxwell's equations for spherical particles, and the most widely used numerical method for light scattering. In fact it is the first well-known theoretical framework to explain scattering and extinction of light by small spherical particles embedded in homogenous environments. Because of the simplicity, accuracy and efficiency of Mie theory, it is still extensively used by many researchers across the globe to develop more accurate and approximate analytical solutions and calibrate new scattering instruments [1 - 3, 104, 109, 110]. It is worth mentioning that since 1955, Mie's [103, 104] 1908 paper [100] has been cited in almost 4000 journal articles which itself indicates the profound impact of Mie theory in the development of electromagnetic theory as well as computational and

measurement techniques and methodologies [110]. In the recent years, tremendous amount of work has been done to develop fast, easily understandable numerical algorithms and computer programs based on Mie theory [111 - 123]. Computations have also been made for scattering by polydisperse systems of such spherical particles [124]. However limited work has been done in the area of light scattering studies on ensembles of homogenous spherical particles with very large size parameters.

- **Separation of variable method (SVM):** The method solves the electromagnetic scattering problem for a spheroid (prolate or oblate) in the respective spheroidal coordinate system and its solution is given as an expansion in terms of vector spherical harmonics and the expansion coefficients are determined by directly applying the continuity condition of the tangential electromagnetic field components at the boundary surface of the scatterer, thus ensuring the completeness and the uniqueness of the solution from mathematical point of view [4, 125, 126]. SVM for single, homogenous, isotropic spheroids was pioneered by Asano [127], Asano and Yamamoto [128], and further improved by Voshchinnikov and Farafonov [129]. SVM was successfully applied to obtain exact analytical solutions to problems involving scattering of electromagnetic waves by core-mantle spheres and spheroids, concentric multilayered spheres, homogenous isotropic cylinders, homogenous isotropic spheroids, optically active spheroids, coated spheroids, lossy dielectric spheroids etc [4,130–135]. These results are often used as benchmarks for evaluating the accuracy and efficiency of other methods [126]. Although the numerical algorithm of SVM is fast, accurate and powerful for simple geometries (spheres, spheroids, cylinders etc.), SVM is not suitable for natural particles of highly non-spherical geometries.
- **Finite difference Time Domain method (FDTDM):** The FDTDM implements Maxwell's time dependent curl equations to solve the temporal variations of electromagnetic waves within a finite space that

contains the scattering particle [4, 136, 137]. The FDTDM discretizes the finite space containing the scattering particle by a grid mesh and simulates the field in this region using the technique proposed in 1966 by Yee [138, 139]. The variations of the electromagnetic properties as functions of spatial location are specified by defining the permittivity, permeability and conductivity at each grid point. In FDTDM, time is also approximated by discrete time steps and the evolution of the electromagnetic field in space at a given instant of time is calculated by iterative computation of the initial value at some initial time. The FDTDM is inherently a near field method. To determine the far field scattering pattern, the near field time domain data is first transformed to near field frequency domain and then the near field in frequency domain is transformed to corresponding far field based on a rigorous electromagnetic integral method [140 - 143]. In FDTDM the mesh must be truncated with appropriate boundary conditions or absorbing boundary conditions (ABC) [4] which either absorb outgoing waves or simulate free space conditions. Otherwise the waves impinging on the outer boundary of the grid would be reflected back into the computational domain resulting in numerical instability. Recently, extensive works on the calculation of far field light scattering patterns of biological cells [144], hexagonal, column, bullet rosette ice crystals, cirrus clouds, by using FDTDM were reported [4, 145 - 148].

- **Finite Element method (FEM):** In FEM the scatterer is embedded in a finite computational domain that is discretized into many small volume segments called elements with about 10 to 20 elements per wavelength [4]. Unlike FDTDM where the scattering particle is discretized on a cubic grid leading to a staircase approximation of the shape of the scattering particle, in FEM the particle shape can be discretized by using variety of elements of different shapes such as triangles and rectangles for two-dimensional (2D) problems and tetrahedral elements for three-dimensional (3D) problems and enables one to handle complex particle shapes [4, 137, 142].

The basic object of FEM is to calculate the electromagnetic field values at the nodes or alternatively at the edges of these shapes. The governing differential equations are expressed along with the associated boundary conditions as a set of linear equations that can be solved computationally using optimization methods like Gaussian elimination (GE) and conjugate gradient method (CGM). Although FEM is simple to compute and permits arbitrary shaped and inhomogeneous particles, the computations are time consuming and require a lot of memory. Significant work has been done in the field of medicine, microwave and millimeter wave circuits, three-dimensional scattering and radiation problems etc [4, 46, 142, 149-151] using FEM.

- **Point Matching Method (PMM):** In this differential equation technique, the incident field and the internal field are expanded as vector spherical wave functions (VSWFs) regular at the origin, and the scattered field outside the scatterer is expanded as outgoing VSWFs [4]. The expansion coefficients are calculated by applying appropriate boundary conditions at the surface of the scatterer by combining point matching with a least square fitting procedure. It was originally developed to compute microwave scattering by spheroidal raindrops [142]. Although PMM is suitable for rotationally symmetric scatterers and less analytical and programming effort is needed to implement this method, it requires high computation time and is not accurate for particles with large size parameter [152]. Determination of electromagnetic scattering properties of particles using PMM like coated axisymmetric particles, spheroidal hydrometeors etc., has been done by many workers [4, 6, 140, 142, 143, 153].
- **Discrete Dipole Approximation (DDA):** The DDA (also referred to as the coupled dipole method or CDM) was originally proposed by de Voe [154] and by Purcell and Penny Packer [155]. In DDA a scattering particle of arbitrary shape is treated as a three dimensional assembly of dipoles. The

electromagnetic field experienced by such a dipole is the superposition of the external field and the fields scattered by all other dipoles [4, 140, 152]. DDA is a kind of volume integral equation method (VIEM) and has several advantages. It is applicable to arbitrary shaped, inhomogeneous, anisotropic and optically active particles [126, 156, 157]. However a major disadvantage is that, the numerical accuracy is low, especially for the scattering matrix elements, and only improves slowly as the number of cells is increased. Another disadvantage is the need to repeat calculations for each new angle of incidence of the external field for which the orientation averaging becomes time -consuming [4, 126].

- **T-matrix method (TMM):** The TMM is based on expanding the incident field in VSWFs regular at the origin and expanding the scattered field outside a circumscribing sphere of the scatterer in VSWFs regular at infinity. The expansion coefficients of the scattered field are related to the coefficients of the incident field by the T-matrix (transition matrix) [4, 6]. T-matrix was originally introduced by Waterman [158 - 160] as a technique for calculating electromagnetic scattering by single, homogenous, arbitrary shaped particles based on Huygen's principle. This technique is also known as extended boundary condition method (EBCM) [4, 5, 142, 161], null field method (NFM) [163, 164], or the Ewald-Oseen Extinction theorem [162]. A fundamental feature of the T-matrix method is that the elements of the T-matrix are independent of the incident and the scattered fields and depend only on the shape, size parameter, and refractive index of the scattering particle(s) as well as its orientation with respect to the coordinate system. Thus T-matrix needs to be computed only once and then can be used in computations for any direction of light incidence and scattering. Thus, in contrast to methods such as the FDTD or the FEM, numerical computations do not need to be repeated for each new angle of incidence. Conversely, one can also keep the incident field fixed and rotate the particle. A review of the current status of TMM has

been published by Mishchenko, Travis and Mackowski [163]. They concluded that recent improvements have made TMM applicable to particles much larger than the incident wavelength (size parameter, $x \approx 100$) [165]. Therefore TMM can also be used for checking the accuracy of geometrical optics approximations and its modifications at lower frequencies. Like all other methods, TMM also has its intrinsic limitations. It is very difficult to program T-matrix for extreme particle shapes, especially, because the surface of the particle must be piecewise smooth and it must be mathematically possible to describe the surface as a continuous function. Moreover, the scatterer has to be isotropic and must have a linear response to the incident electromagnetic field [142]. However, the use of T-matrix approach to calculate the 4x4 scattering matrix of spherical and non spherical particulate matter has led to successful interpretation of many observations [163 - 176]. T-matrix is applicable to any kind of shape, and the recent practical applications have been centered on spheroidal, cylindrical and chebyshev particles [4, 5, 169, 173]. A more recent and simpler derivation of T-matrix equations [170] avoids the use of Rayleigh's hypothesis. Special procedures have been developed to improve the numerical stability of the T-matrix method for large size parameters with extreme aspect ratios [169, 175, 176] and thus improve the practical applicability of this method to particles without axial symmetry, for example ellipsoids, cubes, and clusters of spheres.

In addition to such numerical methods on light scattering by particles of sizes comparable to the incident wavelength, several groups have developed theoretical approximations for light scattering calculations from particles which are too small or too large compared to the incident wavelength. Two basic approximations which are widely used are summarized below:

- **Rayleigh, Rayleigh-Gans (RG) and Anomolous Diffraction (AD) Approximations:** Rayleigh (1897) derived an approximation for scattering in the small particle limit (size parameter, $x = \frac{2\pi a}{\lambda} << 1$, where a is the diameter for a spherical particle and λ is the wavelength of radiation) by assuming that the incident field inside and near the particle behaves almost as an electrostatic field and the internal field is homogenous. A number of studies on Rayleigh Approximation were performed in the past to improve its range of applicability. It is worth mentioning that many particles in nature are such that their refractive index m, relative to the surrounding medium is close to unity (i.e. $|(m-1) << 1|$). Such particles are termed as optically soft particles. It was observed that for soft particles which are not too large (but may be larger that the Rayleigh particles), it is possible to obtain relatively simple approximate expressions for the scattering matrix elements, an approximation commonly named as Rayleigh-Gans (RG), Rayleigh-Debye or Born approximation.Within limits of approximation, these expressions are valid for particles of arbitrary shape. Thus the conditions of validity of RG approximation [2, 3, 4, 177] are $x|m-1| << 1$ and $|m-1| << 1$. The anomalous diffraction approximation of van de Hulst assumes that the particle under consideration is bigger than the incident wavelength but the refractive index is very small [2, 3], i.e., $|m-1| << 1$, so that rectilinear propagation within the scattering particle occurs, followed by subsequent diffraction according to Huygen's principle. It was observed that, for a spherical particle, the AD approximation (ADA) extinction efficiency as a function of size parameter agrees with the results predicted by Mie theory for $|m| < 1.6$ [179], which established the fact that ADA can provide a useful description of scattering by nonspherical particles at intermediate size parameters and for moderate refractive indices [178 - 180].

- **Geometrical optics approximation (GOA):** The laws of geometrical optics may be used to compute the angular distribution of light which is scattered when a plane electromagnetic wave is incident on a particle much larger than the wavelength of the incident light. This approximation method is popularly known as geometrical optics approximation (GOA) or ray tracing or ray optics approximation [4,145,181-184, 280, 281]. The basic assumption of GOA is that when the size of the scatterer is much larger than the incident wavelength, the incident plane wave can be represented as a bundle of separate parallel rays which encounters with the particle and obeys Fresnel equations and Snell's law. Each ray may be externally reflected by the particle or refracted into the particle. In the later case the ray may be absorbed in the particle or it may emerge out of the particle after suffering some amount of internal reflections referred to as Fresnelian interactions. Moreover particles much larger than the wavelength also scatter light by the phenomenon of diffraction and removes energy from the light wave passing by the particle. In far field, the diffracted component of the scattered light can be approximated by Fraunhofer diffraction theory. Thus the total intensity of the far field scattered light can be computed from the summation of the intensity contributed by each individual ray originating from geometric reflection, refraction and Fraunhofer diffraction of the incident wave when it encounters the scattering particle [4, 181].

As a mathematical and numerical basis for the analysis of observed data from the experiments with water droplets, aerosols, hydrosols and nanoparticles, Mie theory and T-matrix approach has been specifically used in this research work respectively, for solving the light scattering problems for

spherical and non-spherical particles of sizes not too large compared to the wavelength of incident light.

Justification for selecting Mie theory for spherical particles:

Mie theory as described in Gaustav Mie's 1908 paper is complete in itself, and it served as the basis for much of the work which has been done in this field since its publication [185].

- It provides a complete, comprehensive and exact analytical solution of a classical scattering problem, including the case of an absorbing host medium, based on the fundamental principles of Maxwell's equations,
- It is the simplest, accurate and efficient method to be applied for light scattering calculation of homogenous spherical particles,
- It is a very fast algorithm and requires very little memory.
- It can be extended to coated spheres, stratified spheres and clustered spheres.

Justification for selecting T-matrix approach for nonspherical particles:

The reasons behind the selection of T-matrix algorithm for the numerical calculations of light scattering from nonspherical particles are as follows:

- T-matrix is computationally very fast,
- corresponding numerical errors are low as the expansion to normalized scattering matrices in generalized spherical functions is analytically calculated,
- the orientational averaging is very efficient
- it allows multiple scattering modeling,
- it can be successfully applied to nonspherical geometries under consideration in this research work.

1.5. Light scattering software

It is worth mentioning that the solutions of the light scattering theories are very complex and rigorous, involve time consuming and complex calculations.

Therefore it is convenient to have such scattering problems programmed in suitable computer languages so that the calculation process needs less effort, becomes fast, accurate and reliable [186 - 189]. Synthesis of such high performance computer programs with an interactive Graphical User Interface (GUI) makes the interaction between a user and a computer program easier. Such GUIs have some inherent advantages for example they are more user friendly, users can easily edit the input parameters and they help the researchers to observe the results in real time and verify their own techniques. Also, a suitable GUI makes the comparative analysis of both theoretical and experimental data much easier which helps in providing new clues as to the development of new models for light scattering from complex bodies. However, very few works on the development of such software or GUIs has been reported [186 - 189, 350].

1.6. Light scattering instruments

Light scattering instruments and systems are extensively used for accurate measurement and analysis of a variety of particulate matter. The surface morphology, along with the geometrical shape of the scattering particle, is the most important factor in determining the optical properties of such scatterers [356]. The 16 elements of the 4×4 scattering matrix of randomly oriented particulate matter (nanoparticles, aerosols, hydrosols etc.) can be measured in the laboratory by using laser light scattering setups [99]. Various techniques have been developed to obtain the angular variation of intensity of scattered light of a specific polarization. This was done in order to determine the size and/or size distribution of an assembly of spherical particles with known optical constants [2, 4]. In 1950, Kerker and VKL Mer [66] developed a method using either the polarization ratio or the phase angle of the scattered light as a function of the scattering angle to extract size information. Since that time several techniques and different approaches have been applied with greater or lesser success [191 - 194] by various workers to design instruments for the experimental observation of scattered light and determine the scattering matrix elements. The first

measurements of the elements of the scattering matrix were performed by B.S. Pritchard and W.G. Elliot [190] using a simple subtraction method. The measurement accuracy was significantly improved by implementing polarization modulation techniques [191, 195-198]. A sophisticated, fully computerized setup to investigate laser light scattering was built successfully in the 1980s and subsequently improved and extended by P. Stammes, F. Kuik, H. Volten, J. W. Hovenier and Olga Muñoz at the Department of Physics and Astronomy, Free University, Amsterdam [4, 5, 33-38]. One more highly accurate system for light scattering studies is the PROGRA2 (Propriétés Optiques des Grains Astronomiques et Atmosphériques) or Optical Properties of Astronomical and Atmospheric Grains project which is mainly dedicated to polarization measurements of dust particles. PROGRA2 utilizes a very efficient way of particle levitation technique i.e. microgravity obtained during parabolic flights producing a realistic environment experienced by the dust particles in space [24, 39 - 44]. Some more simplified and advanced experimental setups are described by D Daugeron [15], B. Barkey [58, 59], M. T. Valentine [199], P. H. Kaye [33], Zbigniew Ulanowski [60, 61], G. A. Ahmed [1, 52, 53]. Moreover electromagnetic scattering properties of millimeter and centimeter sized particles was also investigated at microwave frequencies by a sophisticated, fully automated microwave scattering setup [200].

The common instruments, with remote sensing capability and the potential for providing three-dimensional measurements in the atmosphere and also incorporating laser sources, are the Nephelometer, the Transmissiometer, the Point visibility meter and the LIDAR (Light Detection and Ranging) systems [1].

- **Nephelometer**

An instrument for angular light scattering measurements is called a nephelometer or more precisely, a polar nephelometer [1, 2, 190]. Figure 1.1 shows the ideal optical system of a polar nephelometer. As shown in the figure,

its essential elements are a collimated light source and an arm that can be rotated about the scattering sample. The light source may be a tungsten-halogen, high-pressure mercury or xenon lamp or alternatively, a laser. In between the light source and the scattering sample, optical elements such as neutral density filter, color filters and polarizers (linear polarizer, quarter wave plate etc.) are used to control the intensity, color and polarization of the incident beam of light respectively.

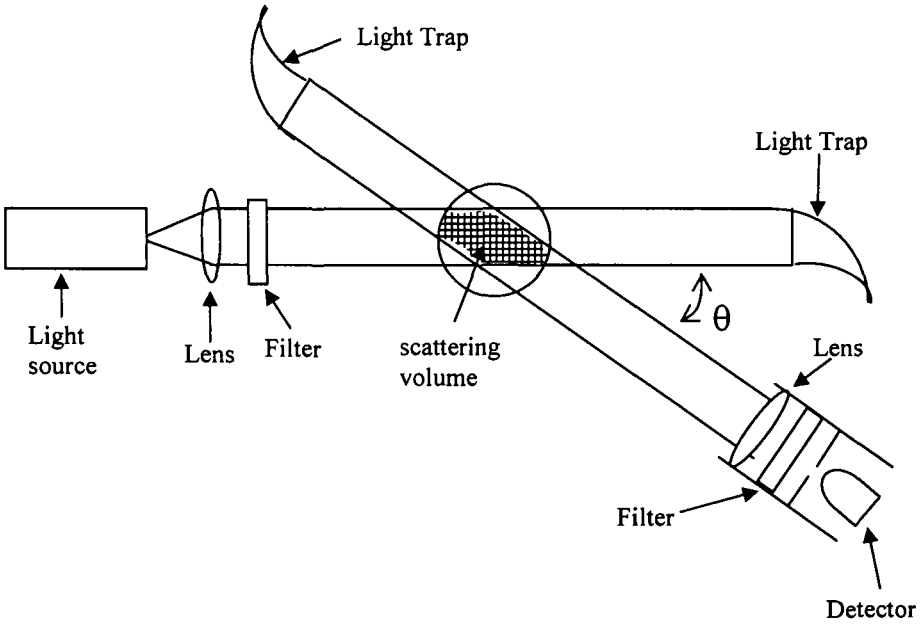

Figure 1.1 A polar nephelometer

The detector system, which includes optical elements to collect light scattered within a small solid angle, is mounted on the arm pivoted at the centre of the scattering plane. It consists of a lens followed by an aperture to limit angular acceptance of the detector, but at the expense of less detection sensitivity. Light traps are used opposite to the direction of the incident beam and the detector to absorb the highly intense projected beam after scattering and provide black background for viewing the scattered radiation. Intersection of the incident beam with the detector field of view determines the scattering volume, which

consequently changes with scattering angle. Therefore it is necessary to employ a correction factor in order to get measurements for constant volume [2]. An inherent drawback of the conventional nephelometers is the difficulty to make measurements at scattering angles near 0^0 and 180^o as the pivoted arm along with the detection system interferes with the incident beam. Angular light scattering measurements are sometimes classified as either absolute or relative. In an absolute measurement the ratio of scattered irradiance I_s to incident irradiance I_i, which is directly related to a factor $dC_{sca}(\theta)/d\Omega$ called the differential scattering cross section, is determined. In a relative measurement the irradiance is referred to some arbitrary scattering angle, say 10^0, so that,

$$\frac{I_s(\theta)}{I_s(22.5°)} = \frac{I_s(\theta)/I_i}{I_s(22.5°)/I_i} = \frac{dC_{sca}(\theta)/d\Omega}{dC_{sca}(22.5°)/d\Omega}$$

Relative measurements are considerably easier to make and are the type most commonly reported [2]. A nephelometer yields information regarding volume scattering coefficients and consequently information of shape and size of the sample under investigation [201 - 205].

- **Transmissiometer**

In the atmosphere, the transmission in the visible portion of the spectrum is significantly affected principally by scattering. The optical transmission of a particular medium is defined as the ratio of the initial intensity to the final intensity of light after it has traversed the medium. Clearly this number will vary with the path length through the medium as well as the characteristic extinction of the medium [206]. An instrument for measuring light transmittance is called a transmissiometer [190, 207]. The essentials of a practical transmissiometer are shown in figure 1.2.

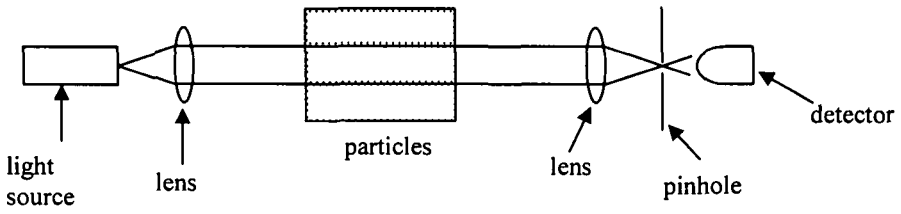

Figure 1.2 An idealized transmissiometer

Light from a point source is collimated, if necessary, by a lens, transmitted through the particulate sample, and focused by a lens onto a detector through a pinhole, if necessary. Thus a transmissiometer detects the light scattered only in the forward direction. Transmittance T of an optical path is the ratio of the final flux P in a light beam to the initial flux P_o.

$$T = \frac{P}{P_o}$$

For a homogeneous path of length D, a transmissiometer gives the extinction coefficient σ as

$$T = e^{-\sigma D}$$

This information is used to measure absorption, visibility and find number densities of the samples under investigation [208 - 213]. However errors in measurement of these parameters may arise if the light scattered in the forward direction is not linearly related to the extinction coefficient.

➢ **Point Visibility Meter (PVM)**

The Point visibility meter (PVM) is a compact instrument that measures the amount of light scattered at a particular forward scattering angle. The extinction coefficient σ is then found from the empirical law relating σ to the scatterer at this angle [214, 215].The block schematic of a point visibility meter is shown in figure 1.3.

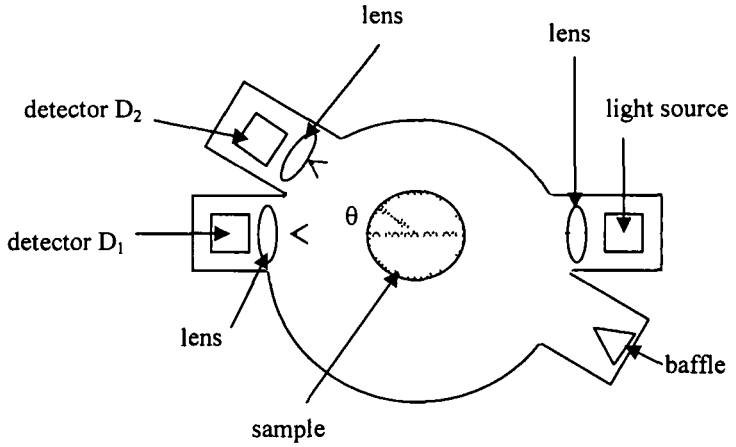

Figure 1. 3 Block schematic of Point visibility meter (PVM).

The radiation emitted by the light source is scattered by the particulate sample. A fraction of the radiation scattered in the forward direction is collected by the scattering detector D_2 at an angle θ. The unscattered radiation is collected by the reference detector D_1. The empirical law derived from the observations [124] of linear dependence of the extinction coefficient σ on intensity of radiation scattered $\beta(\theta)$,

$$\frac{\beta(\theta)}{\sigma} = \text{constant}$$

at an optimum angle between 30° and 40°. This is the basis on which the PVM is operated.

- **Light Detection and Ranging (Lidar)**

Lidar, which is an acronym for light detection and ranging, is used for laser remote sensing of atmospheric properties from a single location [216 - 219]. It consists of a laser and a photo-receiver that measures the backward scattering of light. In lidar, the projection of a short laser pulse is followed by reception of a portion of the radiation reflected from a distant target or from atmospheric constituents such as molecules, aerosols, clouds or dust, as shown in figure 1.4.

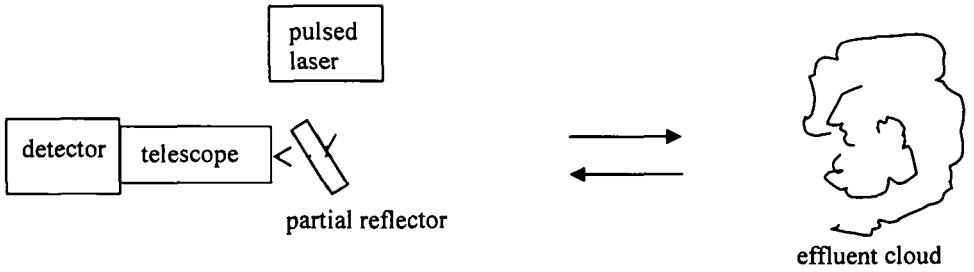

Figure 1. 4 Schematic of a lidar

Generation of a light beam of high power and small angular divergence gives the great advantage to lidars over projector sounding systems that existed before. The possibility of accurate wavelength tuning, as well as spectral return measuring, allows the determination of the chemical composition of the atmosphere and the biochemical composition of the water bodies [333]. Furthermore, as lasers are able to generate powerful pulses of short duration, there is the possibility of measuring time-dependent returns, i.e., measuring not only the integral optical characteristics of a medium, but also their spatial distribution. These features made lidars a powerful tool in the investigation of geophysical media [30].

The basic lidar principles outlined above may be expressed formally by the lidar equation,

$$P_r(R) = P_o\left(\frac{c\tau}{2}\right)\beta(R)A_r R^{-2}\, exp\left[-2\int_0^R \sigma(r)dr\right]$$

where P_r is the instantaneous received power at time t, P_o is the transmitted power at time t_o, c is the velocity of light, τ is the pulse duration, β is the volume backscattering coefficient of the atmosphere, R is the range, A_r is the effective receiver area and σ is the volume extinction coefficient of the atmosphere [1].

Lidar systems are classified as follows: (i) Atmospheric backscatter lidar, where the lidar transmits one laser wavelength and detects changes in the backscatter due to the aerosols or dust in the atmosphere, (ii) Differential-

absorption lidar (DIAL) which measures the concentration of a molecular species in the atmosphere by transmitting two wavelengths, only one of which is absorbed, and detecting the difference in the intensity of the returns at the two wavelengths, (iii) Fluorescence lidar which uses two wavelengths as in the DIAL system and in addition uses spectrometric techniques to separate the wavelength-shifted fluorescence signal from the strong Rayleigh backscatter in the atmosphere, (iv) Raman and Doppler lidar which uses a single-wavelength laser and sophisticated detection techniques to spectrally resolve the wavelength shifted signal from the strong background due to Rayleigh or Mie scattering [1, 30].

Although very expensive and requires highly sophisticated instrumentation, LIDAR systems are the most popular and is being used to explore a wide range of areas [220 - 228] in air quality monitoring.

1.7. Introduction to the presented work

In the context of the present state of light scattering studies mentioned in the above sections, the work in this thesis proposes to design and fabricate a new detector array incorporated He-Ne laser based scattering system where the synthesis of the design aspects of a nephelometer, a transmissiometer, a point visibility meter and features of lidar are attempted. This would contribute firstly, to the development of a fast, reliable, efficient, portable and far more economic light scattering system and secondly, will allow the growing need for investigations on nanoparticles and the wide range of constituents present in the atmosphere. An attempt is also made to develop graphical user interface integrated efficient computer programs using Mie theory and T-matrix approach for light scattering studies on spherical and non-spherical particles.

Chapter I of this thesis gives an introduction to light scattering by aerosols, hydrosols and nanoparticles and their importance, the basic theories and experiments used in light scattering studies.

Chapter II of the thesis reviews the theories of extinction and scattering to bring forward the significance of the parameters monitored by the designed and fabricated light scattering system and explores Mie scattering and T-matrix approach in detail in an effort to computerize the process of calculating scattering matrix elements to aid the light scattering system in determining the physical properties of the scatterers.

Chapter III of the thesis presents the design and instrumentation of the light scattering system and gives the details of every unit of the system. The automated data acquisition system associated with the light scattering system is discussed here. Techniques applied for correction of systematic errors and the data processing software developed for the air quality monitoring system are also presented in this chapter. This chapter also presents the detailed description of an analytical software package developed on java platform in an attempt to analyze the results of the experimental investigations with the theoretical results.

Chapter IV of the thesis presents the results of the experimental investigations carried out in the scattering chamber, on aerosols, hydrosols and nanoparticles. The efforts to yield quantitative measures of size distributions of the scattering particles by the method of comparison of experimental results with the theoretically derived results, discussed in chapter II, are presented and the significant findings from the investigations are laid down. Here the works reported by other authors on the subject of extinction and scattering are also presented for relative assessment with the work presented in this thesis.

Chapter V of the thesis lays down the conclusions drawn from the work presented in the thesis and gives suggestions for future improvements in light scattering studies.

Chapter II: Theory of extinction and scattering

2.1. General introduction

The theoretical basis for describing electromagnetic scattering by particles is formed by classical electrodynamics. In this chapter, we describe two rigorous electromagnetic theories required for the processing the experimental data of the designed and fabricated light scattering setup. We first begin with the Maxwell's equations in Section 2.1.1 in order to present Poynting vector in a form required to for developing the Stokes parameters. In the next section 2.2 the relation between incident and scattered fields and the Stokes parameters are developed. Finally this relation and the Stoke's parameters lead to the formation of the Scattering matrix or Mueller matrix for scattering, the first two elements of which are related to the measurements made by the light scattering setup. The section also describes the related measurements of the scattering matrix elements along with the volume scattering function and the degree of linear polarization. In Section 2.3 the cross-sections of extinction, absorption and scattering are derived and are then used in forming the theory for extinction co-efficient. Since in the experiments with the light scattering setup, both spherical and nonspherical aerosols and hydrosols are investigated, Mie theory and Waterman's T-matrix approach is taken up in Sections 2.4 and 2.5 to explain the observed angular variations of the scattered light. These sections give the numerical algorithms and describe computer programs to calculate theoretical values of scattering matrix elements as well as scattering coefficients over size distributions which may be co-related with experimental results. Thus this chapter incorporates the original efforts in the thesis at computerization of the scattering matrix in a format compatible with the design of the light scattering setup together with the casting of the standard theory of scattering coefficients in a form that is

convenient for computerization. A special reference is made to Ahmed's [1] and Bohren and Huffman's [2] works for developing Mie theory and Mishchenko's [4, 5] work for developing T-matrix method. During the development of Mie theory and T-matrix method, consistency of the notations as used by these authors is maintained as far as possible.

2.1.1. Maxwell's equations, constitutive relations and boundary conditions

The Maxwell's equations [1 - 7, 229 - 231] for the macroscopic electromagnetic field (in SI units) at interior points in the matter are given by,

$$\nabla . D = \rho \qquad \textbf{2.1.1}$$

$$\nabla \times E = -\frac{\partial B}{\partial t} \qquad \textbf{2.1.2}$$

$$\nabla . B = 0 \qquad \textbf{2.1.3}$$

$$\nabla \times B = J + \frac{\partial D}{\partial t} \qquad \textbf{2.1.4}$$

where E is the electric field, B is the magnetic induction, ρ and j are the macroscopic charge density and current density respectively. All quantities entering equations 2.1.1 - 2.1.4 are functions of time, t, and spatial coordinate, r. The electric displacement D and magnetic field H are defined by,

$$D = \varepsilon_0 E + P \qquad \textbf{2.1.5}$$

$$H = \frac{1}{\mu_0} B - M \qquad \textbf{2.1.6}$$

where P is the electric polarization (average electric dipole moment per unit volume), M is the magnetization (average magnetic dipole moment per unit volume), ε_0 and μ_0 are the electric permittivity and magnetic permeability of free space respectively.

Equations 2.1.1 - 2.1.6 are insufficient for the unique determination of the electric and magnetic fields from a given distribution of charges and currents and must be supplemented with so called constitutive relations, which are given by,

$$P = \varepsilon_0 \chi E \qquad \textbf{2.1.7}$$

$$B = \mu H \qquad \textbf{2.1.8}$$

$$J = \sigma E \qquad \textbf{2.1.9}$$

where χ is the electric susceptibility, μ is the magnetic permeability and σ is the conductivity. Equations 2.1.5 and 2.1.7, yield

$$D = \varepsilon E \qquad \textbf{2.1.10}$$

where

$$\varepsilon = \varepsilon_0 [1 + \chi] \qquad \textbf{2.1.11}$$

is the electric permittivity.

The phenomenological constants σ, μ and χ depend on the medium under consideration and also termed as material constants. Throughout this thesis, these material constants will be assumed to be independent of the fields (the medium is linear), independent of position (the medium is homogenous), and independent of direction (the medium is isotropic) [2]. The Maxwell's equations are strictly valid only for points in whose neighborhood the material constants σ, μ and χ vary continuously. At the interface separating one medium from another, these constitutive parameters may vary abruptly which indicates the discontinuous behavior of he field vectors *E*, *D*, *B* and *H* [5, 229]. The boundary conditions at such an interface can be formulated by treating with the integral form of Maxwell's equations and are as follows:

1. $(D_2 - D_1)\hat{n} = \rho_s$ **2.1.12**

   where $\hat{n}$ is unit vector along the local normal to the interface, pointing from medium 1 toward medium 2 and ρ_s be the surface charge

density. Thus there is a discontinuity in the normal component of D if the interface carries a layer of surface charge density.

2. $\hat{n} \times (H_2 - H_1) = J_s$ **2.1.13**

where J_s is the surface current density. Thus there is a discontinuity in the tangential component of H if the interface can carry a surface current. However, media with finite conductivity cannot support surface currents so that

$\hat{n} \times (H_2 - H_1) = 0$ **2.1.14**

3. $\hat{n} \times (E_2 - E_1) = 0$ **2.1.15**

Tangential component of E is continuous across the boundary.

4. $(B_2 - B_1).\hat{n} = 0$ **2.1.16**

Normal component of B is continuous across the boundary.

2.2. Scattering matrix and scattering coefficients

The Maxwell equations for the electromagnetic field in a region where free charge density $\rho_F = 0$ and current density $J_F = 0$ are given as

$\nabla.E = 0$ **2.2.1**

$\nabla \times E = -\mu \frac{\partial H}{\partial t}$ **2.2.2**

$\nabla.H = 0$ **2.2.3**

$\nabla \times H = \varepsilon \frac{\partial E}{\partial t}$ **2.2.4**

where E is the electric field, B is the magnetic field, μ and ε are the magnetic permeability and electric permittivity respectively of the medium. The solutions of Maxwell's equations cannot be arbitrary [1, 2, 126]. Only certain electromagnetic fields, those satisfy Maxwell's equations are physically

realizable. In a infinite linear, homogenous and isotropic medium, the electromagnetic radiation propagates as plane waves given by,

$$E = E_0 e^{i(kr-\omega t)} \qquad \textbf{2.2.5}$$

$$H = H_0 e^{i(kr-\omega t)} \qquad \textbf{2.2.6}$$

where E_0, H_0 and k are the constant complex vectors. As there is no electric and magnetic field component along the direction of propagation of wave i.e. the electromagnetic wave is transverse and the complex field vectors E and H are in phase and mutually perpendicular we can write:

$$k.E_0 = 0 \qquad \textbf{2.2.7}$$

$$k.H_0 = 0 \qquad \textbf{2.2.8}$$

$$k \times E_0 = \omega\mu H_0 \qquad \textbf{2.1.9}$$

$$k \times H_0 = -\omega\varepsilon E_0 \qquad \textbf{2.2.10}$$

Hence it follows that if the refractive index of the medium is given as $n = n' + in''$, then

$$E = E_0 e^{(-k''x)} e^{(ik'x-i\omega t)} \qquad \textbf{2.2.11}$$

$$H = H_0 e^{(-k''x)} e^{(ik'x-i\omega t)} \qquad \textbf{2.2.12}$$

where $k = k' + ik''$ and ω is the angular frequency respectively. $E_0 e^{-k''x}$ and $H_0 e^{-k''x}$ are the amplitudes of the electric and magnetic waves, and $k'x - \omega t$ represent the phase of the waves.

Equations 2.2.6 and 2.2.9 yield,

$$H = \frac{1}{\omega\mu} k \times E \qquad \textbf{2.2.13}$$

The Poynting vector S specifies the magnitude and direction of the rate of transfer of electromagnetic energy at all points of space and has the dimension [energy / (area × time)]. Poynting vector for plane waves is given by,

$$S = \frac{1}{2}\mathrm{Re}(E \times H^*) = \mathrm{Re}\left(\frac{E \times (k^* \times E^*)}{2\omega\mu^*}\right) \qquad \textbf{2.2.14}$$

where $E \times (k^* \times E^*) = k^*(E.E^*) - E^*(k^*.E)$ and if the wave is homogenous, $k.E = 0$ i.e. $k^*.E = 0$ and the Poynting vector for such a wave propagating in the ê direction is given as

$$S = \frac{1}{2}\mathrm{Re}\left\{\left(\frac{\varepsilon}{\mu}\right)^{\frac{1}{2}}(E.E^*)\right\}\hat{e} \qquad \textbf{2.2.15}$$

where E^* is the complex conjugate of E and $\hat{e}$ is the direction of propagation. Assuming the wave with wavelength λ to be moving in the z direction, the above equation becomes

$$S = \frac{k}{2\omega\mu}|E_0|^2 e^{4\pi\frac{n''z}{\lambda}}\hat{e} \qquad \textbf{2.2.16}$$

This gives the modulus of S as

$$|S| = \frac{k}{2\omega\mu}I \qquad \textbf{2.1.17}$$

where I is the irradiance with dimension of energy per unit area and time. As the wave traverses the medium, the irradiance may be given from 2.1.16 and 2.1.17 as,

$$I = I_0 e^{-\alpha' z} \qquad \textbf{2.1.18}$$

where

$$\alpha' = \frac{4\pi n''}{\lambda} \qquad \textbf{2.1.19}$$

is the absorption co-efficient and

$$I_0 = \frac{k}{2\omega\mu}|E_0|^2 \qquad \textbf{2.1.20}$$

is the irradiance at z = 0. Also if there is no absorption in the medium then $n'' = 0$ and 2.1.17 becomes

$$|S| = \frac{k}{2\omega\mu} I_0 \qquad \textbf{2.1.21}$$

The equations 2.1.20 and 2.1.21 are now in a form that can be used for development of the Stokes parameters which in turn is to be used in developing the Mueller matrix for scattering.

Since the relation between the incident and scattered fields is also required in the development of the Mueller matrix for scattering it is derived by taking the case of scattering by a single particle as shown in Fig 2.1. The direction of propagation of the incident light is taken as the *z*-axis. Any point in the particle is chosen as the origin of a rectangular Cartesian co-ordinate system (x,y,z),where x and y axes are orthogonal to the *z*-axis and to each other but are otherwise arbitrary. The orthogonal basis vectors $\hat{e}_x, \hat{e}_y, \hat{e}_z$ are in the directions of the positive x, y and z axes. The direction of scattering may also be given in terms of $\hat{e}_r, \hat{e}_\theta, \hat{e}_\phi$ which are the orthonormal basis vectors associated with the spherical polar co-ordinate system (r,θ,ϕ). Here θ is the scattering angle and $\hat{e}_r$ and $\hat{e}_z$ define the scattering plane. The scattering plane is uniquely determined by the azimuthal angle ϕ except when $\hat{e}_r$ is parallel to the *z*-axis. In these two instances, that is $\hat{e}_r = \pm\hat{e}_z$, any plane containing the *z*-axis is a suitable scattering plane.

It is convenient to resolve the incident field E_i which lies in the x-y plane into the components parallel ($E_\parallel$) and perpendicular ($E_\perp$) to the scattering plane since an unpolarized monochromatic wave can be expressed as the sum of two orthogonally polarized components (i.e. perpendicular and parallel linearly polarized; or left or right circularly polarized [232]). Thus we get the relation,

$$E_i = \left(E_{0\parallel}\hat{e}_{\parallel i} + E_{0\perp}\hat{e}_{\perp i}\right)e^{(ikz-i\omega t)} = E_{\parallel}\hat{e}_{\parallel i} + E_{\perp}\hat{e}_{\perp i} \qquad \textbf{2.2.22}$$

where $k = \frac{2\pi n}{\lambda}$ is the wave number in the medium surrounding the particle, n is the refractive index of the medium and λ is the wavelength of the incident radiation in free space.

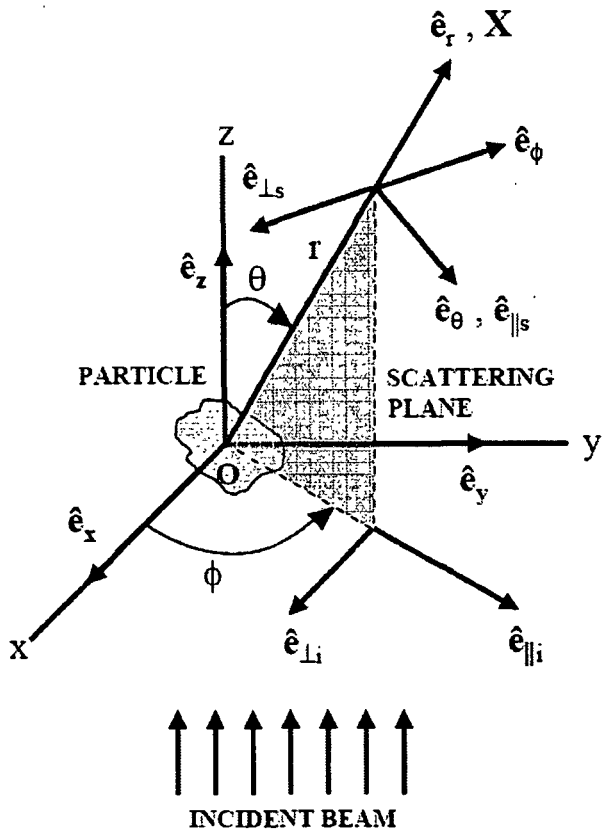

Figure 2.1 Scattering by a particle.

The vectors $e_{\parallel i}$ and $e_{\perp i}$ are defined by,

$$\hat{e}_{\perp i} = \sin\phi\hat{e}_x - \cos\phi\hat{e}_y \qquad \textbf{2.2.23}$$

$$\hat{e}_{\parallel i} = \cos\phi\hat{e}_x + \sin\phi\hat{e}_y \qquad \textbf{2.2.24}$$

where

$$\hat{e}_{\perp i} \times \hat{e}_{\parallel i} = \hat{e}_z \qquad \textbf{2.2.25}$$

also

$$\hat{e}_{\perp i} = -\hat{e}_\phi \qquad \textbf{2.2.26}$$

$$\hat{e}_{\parallel i} = \sin\theta\hat{e}_r + \cos\theta\hat{e}_\theta \qquad \textbf{2.2.27}$$

The x and y components of E_i may be denoted by E_{xi} and E_{yi}. Thus we get

$$E_{\|i} = \cos\phi E_{xi} + \sin\phi E_{yi} \qquad \textbf{2.2.28}$$

$$E_{\perp i} = \sin\phi E_{xi} - \cos\phi E_{yi} \qquad \textbf{2.2.29}$$

In the far field region ($kr >> 1$) i.e. the observer is sufficiently far away from the event being observed, the scattered field E_s is approximately transverse ($\hat{e}_r E_s = 0$) and has the asymptotic form [231]

$$E_s \approx \left(\frac{e^{ikr}}{-ikr}\right) A \qquad kr >> 1 \qquad \textbf{2.2.30}$$

where $\hat{e}_r A = 0$ and A is the ϕ and θ dependent vectorial scattering amplitude. Again E_s in the far field region can be written as

$$E_s = E_{\|}\hat{e}_{\|s} + E_{\perp}\hat{e}_{\perp s} \qquad \textbf{2.2.31}$$

also,

$$\hat{e}_{\|s} = \hat{e}_{\theta} \qquad \textbf{2.2.32}$$

$$\hat{e}_{\perp s} = -\hat{e}_{\phi} \qquad \textbf{2.2.33}$$

$$\hat{e}_{\perp s} \times \hat{e}_{\|s} = \hat{e}_r \qquad \textbf{2.2.34}$$

where the basis vectors $\hat{e}_{\|s}$ and $\hat{e}_{\perp s}$ are parallel and perpendicular to the scattering plane respectively. Thus considering the equations 2.1.22 to 2.1.34 the required relation between incident and scattered fields in 2×2 matrix form may be given as

$$\begin{pmatrix} E_{\|s} \\ E_{\perp s} \end{pmatrix} = \frac{e^{ik(r-z)}}{-ikr} \begin{pmatrix} S_2 & S_3 \\ S_4 & S_1 \end{pmatrix} \begin{pmatrix} E_{\|i} \\ E_{\perp i} \end{pmatrix} \qquad \textbf{2.2.35}$$

where the matrix elements $S_j (j = 1, 2, 3, 4)$ depends on θ and ϕ and the matrix is the so called amplitude scattering matrix which describes the transformation of the θ and ϕ components of the incident plane wave to the θ and ϕ components of the scattered spherical wave. All four elements of the amplitude scattering matrix are dimensionless.

The intensity and state of polarization of a beam of light can be completely described by the Stoke's vector I= *(I, Q, U, V)* where *I* is the total intensity, *Q* is the difference between the polarization intensities at 0^0 and 90^0 to the scattering plane, *U* is the difference between the polarization intensities at $+45^0$ and -45^0 (oblique) to the scattering plane, and *V* is the difference between the left and right circular polarization intensities [1 - 5]. When a beam of light encounters an obstacle (scatterer), it scatters some of the incident radiation in all directions thereby changing the incident intensity and in most general cases changes the polarization state of the incident light. Thus in a scattering process the scatterer modifies the intensity and polarization state of the incident light (i.e. the incident Stokes vectors) and also changes the direction of the incident light by scattering it in any polar (θ) and azimuthal (ϕ) angle [3]. In a light scattering experiment, actually the intensity of the unpolarized or the fraction of a particular polarization state of the scattered light at a particular scattering angle is determined. Most detectors of electromagnetic radiation, especially those in the visible and infrared range, are insensitive to the different polarization states of the scattered beam and can measure only the first element of the Stokes vectors, i.e. the intensity. However with the appropriate use of one or several optical elements (usually polarizers and retarders are used), between the source of light and the detector, we can extract information about the second, third and fourth Stokes parameters of the original beam [2, 229]. In order to derive the Stokes parameters let us now consider a series of hypothetical experiments which can be performed with an arbitrary monochromatic beam, a detector, and various polarizers, as shown in figure 2.2, in a non-absorbing medium. The detector responds to irradiance independently of the polarization state, and the polarizers are assumed to be ideal. The electric field E is referred to orthogonal axes $\hat{e}_{\parallel}$ and $\hat{e}_{\perp}$, which is called horizontal and vertical respectively, and E_0 is given as:

$$E_0 = E_{\parallel}\hat{e}_{\parallel} + E_{\perp}\hat{e}_{\perp} \qquad \textbf{2.2.36}$$

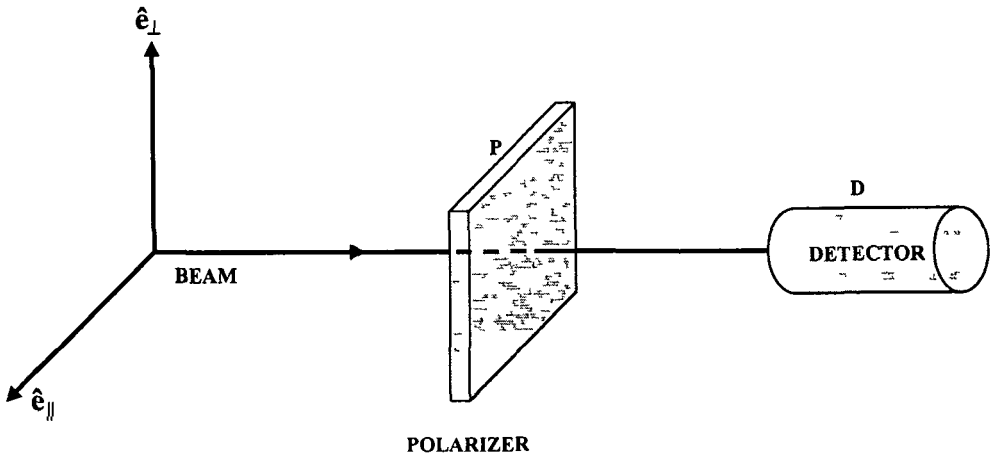

Figure 2.2. Setup for finding Stoke's parameters

Case 1: With no polarizer.

If there is no polarizer in the beam the irradiance I recorded by the detector is given by

$$I = I_{\parallel} + I_{\perp} \qquad \textbf{2.2.37}$$

or by using 2.2.20 and 2.2.36,

$$I = \frac{k}{2\omega\mu_0}\left(E_{\parallel}E_{\parallel}^{*} + E_{\perp}E_{\perp}^{*}\right) \qquad \textbf{2.2.38}$$

Case 2: With horizontal and vertical Polarizers.

Let P, in figure 2.2, be a horizontal polarizer; the amplitude of the transmitted wave is $E_{\parallel}$ and the irradiance $I_{\parallel}$ recorded by the detector is $\left(k/2\omega\mu_0\right)E_{\parallel}E_{\parallel}^{*}$. Next let P be a vertical polarizer; the amplitude of the transmitted wave is $E_{\perp}$ and the irradiance $I_{\perp}$ recorded by the detector is $\left(k/2\omega\mu_0\right)E_{\perp}E_{\perp}^{*}$. The difference between these two measured irradiances is

$$I_{\parallel} - I_{\perp} = \frac{k}{2\omega\mu_0}\left(E_{\parallel}E_{\parallel}^{*} - E_{\perp}E_{\perp}^{*}\right) \qquad \textbf{2.2.39}$$

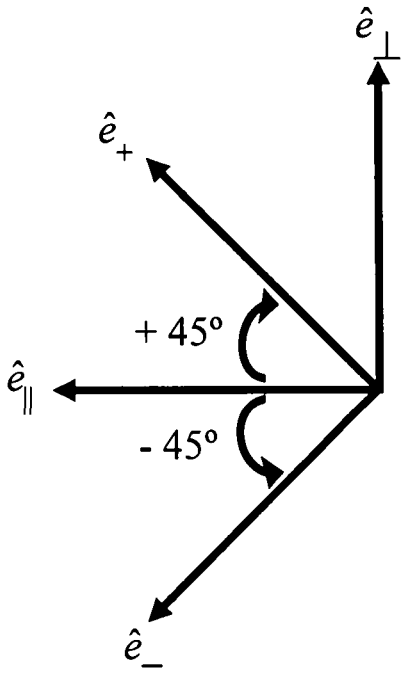

Figure 2.3. Basis vectors $\hat{e}_{+}$ and $\hat{e}$

Case 3: +45° and -45° polarizers.

An orthonormal set of basis vectors $\hat{e}_{+}$ and $\hat{e}_{-}$, which are obtained by rotating $\hat{e}_{\parallel}$ by +45° and -45°, as shown in figure 2.3 may be introduced. Then since,

$$\hat{e}_{+} = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(\hat{e}_{\parallel} + \hat{e}_{\perp}\right) \qquad \textbf{2.2.40}$$

$$\hat{e}_{-} = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(\hat{e}_{\parallel} - \hat{e}_{\perp}\right) \qquad \textbf{2.2.41}$$

Therefore electric field E_0 may be written as

$$E_0 = E_{+}\hat{e}_{+} + E_{-}\hat{e}_{-} \qquad \textbf{2.2.42}$$

where,

$$E_{+} = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(E_{\parallel} + E_{\perp}\right) \qquad \textbf{2.2.43}$$

$$E_{-} = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(E_{\parallel} - E_{\perp}\right) \qquad \textbf{2.2.44}$$

Now if P in figure 2.2 is a +45° polarizer then the irradiance of the transmitted wave is,

$$I_{+} = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\parallel}^{*} + E_{\parallel}E_{\perp}^{*} + E_{\perp}E_{\parallel}^{*} + E_{\perp}E_{\perp}^{*}\right)/2 \qquad \textbf{2.2.45}$$

Again if P is a -45° polarizer then the irradiance of the transmitted wave is

$$I_{-} = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\parallel}^{*} - E_{\parallel}E_{\perp}^{*} - E_{\perp}E_{\parallel}^{*} + E_{\perp}E_{\perp}^{*}\right)/2 \qquad \textbf{2.2.46}$$

The difference between these two irradiances is

$$I_{+} - I_{-} = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\perp}^{*} + E_{\perp}E_{\parallel}^{*}\right) \qquad \textbf{2.2.47}$$

Case 4: With circular polarizers.

Another set of basis vectors $\hat{e}_R$ and $\hat{e}_L$ are defined as

$$\hat{e}_R = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(\hat{e}_{\parallel} + i\hat{e}_{\perp}\right) \qquad \textbf{2.2.48A}$$

$$\hat{e}_L = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(\hat{e}_{\parallel} - i\hat{e}_{\perp}\right) \qquad \textbf{2.2.48B}$$

These basis vectors represent right-circularly and left-circularly polarized waves and are orthonormal in the sense that $\hat{e}_R.\hat{e}_R^{*} = 1$, $\hat{e}_L.\hat{e}_L^{*} = 1$, $\hat{e}_R.\hat{e}_L^{*} = 0$ and $\hat{e}_L.\hat{e}_R^{*} = 0$. Hence, the incident field may be written as

$$E_0 = E_R\hat{e}_R^{*} + E_L\hat{e}_L^{*} \qquad \textbf{2.2.49}$$

where,

$$E_R = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(E_{\parallel} - iE_{\perp}\right) \qquad \textbf{2.2.50}$$

$$E_L = \left(\frac{1}{2}\right)^{\frac{1}{2}}\left(E_{\parallel} + iE_{\perp}\right) \qquad \textbf{2.2.51}$$

Now if P is a right-handed polarizer, then the transmitted irradiance I_R is

$$I_R = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\parallel}^{*} - iE_{\parallel}^{*}E_{\perp} + iE_{\perp}^{*}E_{\parallel} + E_{\perp}E_{\perp}^{*}\right)/2 \qquad \textbf{2.2.52}$$

Again if P is a left-handed polarizer, then the transmitted irradiance I_L is

$$I_L = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\parallel}^{*} + iE_{\perp}E_{\parallel}^{*} - iE_{\parallel}E_{\perp}^{*} + E_{\perp}E_{\perp}^{*}\right)/2 \qquad \textbf{2.2.53}$$

The difference between these two irradiances is

$$I_R - I_L = i\left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\perp}^{*} - E_{\perp}E_{\parallel}^{*}\right) \qquad \textbf{2.2.54}$$

Therefore the required Stokes parameters I, Q, U, V are

$$I = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\parallel}^{*} + E_{\perp}E_{\perp}^{*}\right) \qquad \textbf{2.2.55}$$

$$Q = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\parallel}^{*} - E_{\perp}E_{\perp}^{*}\right) \qquad \textbf{2.2.56}$$

$$U = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\perp}^{*} + E_{\perp}E_{\parallel}^{*}\right) \qquad \textbf{2.2.57}$$

$$V = i\left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel}E_{\perp}^{*} - E_{\perp}E_{\parallel}^{*}\right) \qquad \textbf{2.2.58}$$

The Stoke's parameters of a monochromatic beam of light are not completely independent and related by the equality,

$$I^2 = Q^2 + U^2 + V^2 \qquad \textbf{2.2.59}$$

However for a quasi-monochromatic beam of light the Stoke's parameters are related by the inequality,

$$I^2 \geq Q^2 + U^2 + V^2 \qquad \textbf{2.2.60}$$

Thus for an arbitrary beam of light, the degree of (elliptical) polarization, P_{pol} , degree of linear polarization, P_{lp} and degree of circular polarization, P_{cp} can be defined as [2, 5],

$$P_{pol} = \frac{\left(Q^2 + U^2 + V^2\right)^{\frac{1}{2}}}{I} \leq 1 \qquad \textbf{2.2.61}$$

$$P_{lp} = \frac{\left(Q^2 + U^2\right)^{\frac{1}{2}}}{I} \leq 1 \qquad \textbf{2.2.62}$$

$$P_{cp} = \frac{V}{I} \leq 1 \qquad \textbf{2.2.63}$$

where $\left(Q^2 + U^2 + V^2\right)^{\frac{1}{2}}$, $\left(Q^2 + U^2\right)^{\frac{1}{2}}$ and V represent the intensity of fully polarized, linearly polarized and circularly polarized component.

If $U = 0$ (linearly or elliptically polarized light with the major polarization direction parallel or perpendicular to the reference plane), equation 2.2.62 reduces to [330]:

$$P_{lp} = \frac{|Q|}{I} \leq 1 \qquad \textbf{2.2.64}$$

Additionally for $U = 0$ we will also use another convention [2, 5, 229 - 331] for defining the degree of linear polarization (or the signed degree of linear polarization) and its direction,

$$P = -\frac{Q}{I} = \frac{I_{\perp} - I_{\parallel}}{I} \qquad \textbf{2.2.65}$$

Thus P is positive when the vibrations of the electric vector perpendicular to the plane of reference for the Stokes parameters dominate over the vibrations of the electric vector parallel to this plane [5, 331]. It is worth mentioning here at this point that the present convention of the degree of linear polarization is related to one of the measurements made by the light scattering setup.

Now, in case of scattering of the incident field by a particle, the time averaged Poynting vector S at any point in the medium surrounding the particle can be written as the sum of three terms.

$$S = \left(\frac{1}{2}\right)\mathrm{Re}\left(E_2 \times H_2\right) = \left(\frac{1}{2}\right)\mathrm{Re}\left\{\left(E_i + E_s\right) \times \left(H_i^* + H_s^*\right)\right\} = S_i + S_s + S_{ext} \qquad \textbf{2.2.66}$$

where, E_2 and H_2 are the electric and magnetic fields outside the particle respectively.

Also,

$$S_i = \left(\frac{1}{2}\right)\mathrm{Re}\left(E_i \times H_i^*\right) \qquad \textbf{2.2.67}$$

$$S_s = \left(\frac{1}{2}\right)\mathrm{Re}\left(E_s \times H_s^*\right) \qquad \textbf{2.2.68}$$

$$S_{ext} = \left(\frac{1}{2}\right)\mathrm{Re}\left\{\left(E_i \times H_s^*\right) + \left(E_s \times H_i^*\right)\right\} \qquad \textbf{2.2.69}$$

where S_i is the Poynting vector associated with the incident wave and is independent of position if the medium is nonabsorbing. S_s is the poynting vector of the scattered field and S_{ext} is the poynting vector arising because of interaction between the incident and scattered waves. If a suitably collimated detector of detector area ΔA is placed at a distance D from the particle in the far field region, with ΔA aligned normal to $\hat{e}_r$, and if $\hat{e}_r$ is not too near the forward direction $\hat{e}_z(\theta = 0°)$, the detector will record a signal proportional to

$$S_s \hat{e}_r \Delta A \qquad \textbf{2.2.70}$$

provided ΔA is sufficiently small so that S_s does not vary greatly over the detector. From 2.2.30 and 2.2.67 to 2.2.69 it follows that

$$\mathbf{S}_s.\hat{\mathbf{e}}_r \Delta A = \frac{k}{2\omega\mu_0}\frac{|\mathbf{A}|^2}{k^2}\Delta\Omega \qquad \textbf{2.2.71}$$

where

$$\Delta\Omega = \frac{\Delta A}{D^2} \qquad \textbf{2.2.72}$$

is the solid angle subtended by the detector . Hence it is possible to obtain $|A|^2$ as a function of direction, to within a solid angle $\Delta\Omega$, by recording the detector response at various positions on a hemisphere surrounding the particle. By interposing various polarizers in front of the detector and proceeding in a manner identical to the method used in finding relations 2.2.55 to 2.2.58, the Stokes parameters of the light scattered by a particle is obtained as

$$I_s = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel s}E_{\parallel s}^* + E_{\perp s}E_{\perp s}^*\right) \quad \textbf{2.2.73}$$

$$Q_s = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel s}E_{\parallel s}^* - E_{\perp s}E_{\perp s}^*\right) \quad \textbf{2.2.74}$$

$$U_s = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel s}E_{\perp s}^* + E_{\perp s}E_{\parallel s}^*\right) \quad \textbf{2.2.75}$$

$$V_s = \left(\frac{k}{2\omega\mu_0}\right)\left(E_{\parallel s}E_{\perp s}^* - E_{\perp s}E_{\parallel s}^*\right) \quad \textbf{2.2.76}$$

Thus relations 2.2.35, 2.2.55-2.2.58 and 2.2.73-2.2.76 yields

$$\begin{pmatrix} I_s \\ Q_s \\ U_s \\ V_s \end{pmatrix} = \frac{1}{k^2 D^2}\begin{pmatrix} S_{11} & S_{12} & S_{13} & S_{14} \\ S_{21} & S_{22} & S_{23} & S_{24} \\ S_{31} & S_{32} & S_{33} & S_{34} \\ S_{41} & S_{42} & S_{43} & S_{44} \end{pmatrix}\begin{pmatrix} I_i \\ Q_i \\ U_i \\ V_i \end{pmatrix} \quad \textbf{2.2.77}$$

where

$$S_{11} = \left(\frac{1}{2}\right)\left(|S_1|^2 + |S_2|^2 + |S_3|^2 + |S_4|^2\right) \quad \textbf{2.2.78}$$

$$S_{12} = \left(\frac{1}{2}\right)\left(|S_2|^2 - |S_1|^2 + |S_4|^2 - |S_3|^2\right) \quad \textbf{2.2.79}$$

$$S_{13} = \mathrm{Re}\left(S_2 S_3^* + S_1 S_4^*\right) \quad \textbf{2.2.80}$$

$$S_{14} = \mathrm{Im}\left(S_2 S_3^* - S_1 S_4^*\right) \quad \textbf{2.2.81}$$

$$S_{21} = \left(\frac{1}{2}\right)\left(|S_2|^2 - |S_1|^2 - |S_4|^2 + |S_3|^2\right) \quad \textbf{2.2.82}$$

$$S_{22} = \left(\frac{1}{2}\right)\left(|S_2|^2 + |S_1|^2 - |S_4|^2 - |S_3|^2\right) \quad \textbf{2.2.83}$$

$$S_{23} = \mathrm{Re}\left(S_2 S_3^* - S_1 S_4^*\right) \quad \textbf{2.2.84}$$

$$S_{24} = \mathrm{Im}\left(S_2 S_3^* + S_1 S_4^*\right) \quad \textbf{2.2.85}$$

$$S_{31} = \mathrm{Re}\left(S_2 S_4^* + S_1 S_3^*\right) \quad \textbf{2.2.86}$$

$$S_{32} = \mathrm{Re}\left(S_2 S_4^* - S_1 S_3^*\right) \quad \textbf{2.2.87}$$

$$S_{33} = \mathrm{Re}\left(S_1 S_2^* + S_3 S_4^*\right) \quad \textbf{2.2.88}$$

$$S_{34} = \mathrm{Im}\left(S_2 S_1^* + S_4 S_3^*\right) \qquad \textbf{2.2.89}$$

$$S_{41} = \mathrm{Im}\left(S_2^* S_4 + S_3^* S_1\right) \qquad \textbf{2.2.90}$$

$$S_{42} = \mathrm{Im}\left(S_2^* S_4 - S_3^* S_1\right) \qquad \textbf{2.2.91}$$

$$S_{43} = \mathrm{Im}\left(S_1 S_2^* - S_3 S_4^*\right) \qquad \textbf{2.2.92}$$

$$S_{44} = \mathrm{Re}\left(S_1 S_2^* - S_3 S_4^*\right) \qquad \textbf{2.2.93}$$

Thus the 4×4 matrix in 2.2.77 is the required scattering matrix and is the Mueller matrix for scattering by a single particle. Each of the sixteen elements of the matrix depends on the physical and optical properties of the scatterer and are functions of (θ, ϕ) or simply θ in the case of randomly oriented axially-symmetrical particles. The angle dependence of the various elements of the scattering matrix either provide a direct measure or are indicative of the physical properties of the scatterer as mentioned in Table 2.1 [4, 64, 67, 233]. Among them, the element S_{11} and S_{12} are related to the measurements made by the light scattering setup, a fact that becomes evident when 2.2.77 is used in developing the Mie theory.

Table 2.1. Scattering matrix elements: their measurements and significance

Matrix element	Measurement & Significance
S_{11}	the angular dependence of the scattered light with unpolarized incident radiation, which is proportional to the phase function; gives general size information;
S_{12}	describes the linear cross polarization introduced into the scattered light by particles of an isolated symmetric scattering medium (S_{21} is the converse); depends on size, shape and complex refractive index of the scatterers;
S_{13}	measure of the linear polarization ±45 degree to the scattering plane (S_{31} is the converse); depends on size, nonzero element indicates the presence of orientation effect (nonspherical particles);
S_{14}	measures the degree by which the medium introduces circular polarization into the scattered light (S_{41} is the converse); nonzero elements indicates the presence of optical activity, helical structures, orientation effect;
S_{22}	measurement of the transformation of the linearly polarized incident light (±90 degree) to the linearly polarized scattered light (±90 degree); deviation from S_{11} indicates the presence of nonspherical particles in the scattering volume element;
S_{23}, S_{32}	measurement of the transformation of the linearly polarized incident light (±45 degree) to the linearly polarized scattered light (±90 degree) (S_{32} is the converse); nonzero elements indicates the presence of nonrandom orientation, complex optical properties such as birefringence, circular dichroism, optical rotary dispersion etc.;
S_{24}, S_{42}	measurement of the transformation of the circularly polarized incident light to the linearly polarized scattered light (±90 degree) (S_{42} is the converse); nonzero elements indicates the presence of nonrandom orientation, complex optical properties such as birefringence, circular dichroism, optical rotary dispersion etc.;
S_{33}	measurement of the transformation of the linearly polarized incident light (±45 degree) to the linearly polarized scattered light (±45 degree); deviation from S_{44} indicates the presence of nonspherical symmetry;
S_{34}, S_{43}	measurement of the transformation of the circularly polarized incident light to the linearly polarized scattered light (±45 degree) (S_{43} is the converse); strongly depends on the size and complex refractive index of the scatterers;
S_{44}	measurement of the transformation of the circularly polarized incident light to the circularly polarized scattered light; deviation from S_{33} indicates the presence of nonspherical symmetry.

2.2.1. Measurement of the scattering matrix

For a scattering event where the positions of the particles in a scattering volume are such that each particle scatters independently of all others (single scattering region), the Stokes vector of the incident and the scattered light are connected by the 16 element Mueller matrix provided the frequency of the scattered wave is same as that of the incident wave (elastic scattering). The matrix elements are functions of the direction of observation of the scattered light (scattering angle, θ for randomly oriented axisymmetric particles). Equation 2.2.77 can be rewritten as,

$$\mathrm{I}_s = \frac{1}{k^2 D^2}\mathrm{S}(\theta)\mathrm{I}_i$$

$$\Rightarrow \begin{pmatrix} I_s \\ Q_s \\ U_s \\ V_s \end{pmatrix} = \frac{1}{k^2 D^2} \begin{pmatrix} S_{11}I_i + S_{12}Q_i + S_{13}U_i + S_{14}V_i \\ S_{21}I_i + S_{22}Q_i + S_{23}U_i + S_{24}V_i \\ S_{31}I_i + S_{32}Q_i + S_{33}U_i + S_{34}V_i \\ S_{41}I_i + S_{42}Q_i + S_{43}U_i + S_{44}V_i \end{pmatrix} \qquad \textbf{2.2.94}$$

The subscripts i and s refer to the incident and scattered beams, k is the wave vector and D is the distance from the sample to the detector. As the dimensions of the Stokes vector elements are that of irradiance, i.e. power/length^2, the above equation 2.2.94 indicates that the Mueller matrix elements are dimensionless. Irrespective of the shape of the particles in the scattering volume the wave scattered by a particle spherically propagates in polar (θ) and azimuthal (ϕ) angles [336]. The Mueller matrix represents the transformation performed on the incident Stokes parameters by the particles in the scattering volume as the result of a single scattering event [331]. The matrix elements depend on the physical properties (e.g. size, shape, surface morphology) and optical properties (refractive index) of the scatterer. In other words the elements of the Mueller scattering matrix at an angle carry the signature of the scatterer. Inversely these signatures or these scattering information can be utilized to extract the characteristics of the scatterer. For any kind of scatterer, it is possible to measure each element of the scattering matrix as

a θ-dependent intensity curve by the appropriate adjustment of the delivering and collection optics combinations [99, 193, 233].

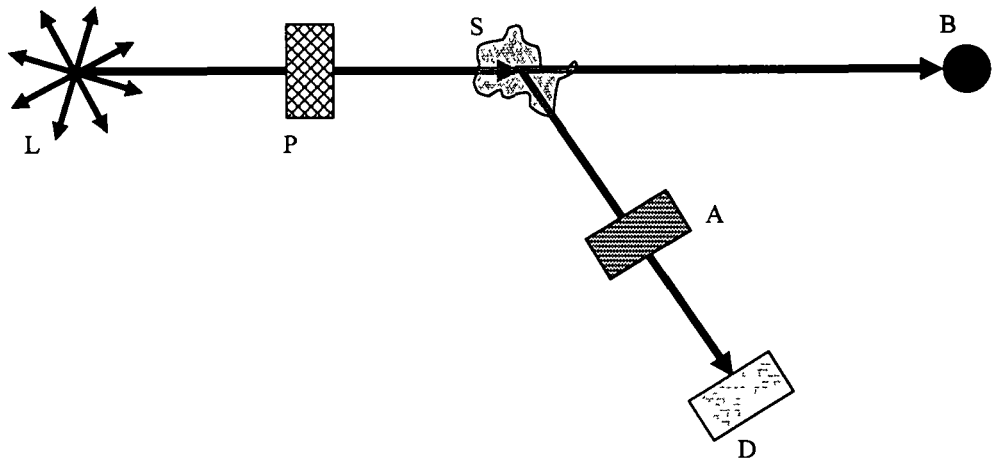

Figure 2.4. Schematic diagram of a simple Mueller matrix measurement arrangement. L: randomly polarized incident laser beam; P: polarizer; S: scattering centre; A: analyzer; D: polarization insensitive photo-detector; B: beam stop.

As shown in figure 2.4, the scattering matrix elements of a particle is measured by using an arrangement of polarizer (to select the state of polarization of the incident light), analyzer (to pass the desired state of polarization of the scattered light for detection) and a photo-detector which should essentially be polarization insensitive. The equivalent mathematical expression for the above arrangement can be given by [33, 64, 334, 341],

$$I_s(\theta) = \left[\frac{1}{kD^2}\right] S_A S(\theta) S_P I_i \qquad \textbf{2.2.95}$$

where I_s and I_i are the Stokes vectors of the scattered and incident light respectively, θ is the scattering angle, S_A and S_P are Mueller matrices of the analyzer and the polarizer. Bickel et al [233], Jonasz [64, 336] and Heilscher et al [335] has be shown that 49 combinations of delivering and collection optics combinations are necessary to obtain all the 16 elements of the Mueller matrix or scattering matrix as given in the table 2.2. The combinations are listed in Table 2.2. In reality each matrix element combination in the table is the first element of the output Stokes vector [233, 336] and is proportional to the intensity measured

by the detector. From table 2.2, the 16 relations (2.2.96 - 2.2.111) can be constructed which are required to measure all the elements of the scattering matrix [64, 233, 335, 336]. For simplicity and following Bickel et al [233] the normalization constants associated with the Mueller matrices of the optical elements and the term $(\frac{1}{kD^2})$ are ignored and the incident intensity was normalized to unity particularly while developing table 2.2 and the following equations.

$$S_{11} = I_{UU} \qquad \textbf{2.2.96}$$

$$S_{12} = \frac{I_{HU} - I_{VU}}{2} \qquad \textbf{2.2.97}$$

$$S_{13} = \frac{I_{PU} - I_{MU}}{2} \qquad \textbf{2.2.98}$$

$$S_{14} = \frac{I_{LU} - I_{RU}}{2} \qquad \textbf{2.2.99}$$

$$S_{21} = \frac{I_{UH} - I_{UV}}{2} \qquad \textbf{2.2.100}$$

$$S_{22} = \frac{1}{4}[(I_{HH} + I_{VV}) - (I_{HV} + I_{VH})] \qquad \textbf{2.2.101}$$

$$S_{23} = \frac{1}{4}[(I_{PH} + I_{MV}) - (I_{PV} + I_{MH})] \qquad \textbf{2.2.102}$$

$$S_{24} = \frac{1}{4}[(I_{LH} + I_{RV}) - (I_{LV} + I_{RH})] \qquad \textbf{2.2.103}$$

$$S_{31} = \frac{I_{UP} - I_{UM}}{2} \qquad \textbf{2.2.104}$$

$$S_{32} = \frac{1}{4}[(I_{HP} + I_{VM}) - (I_{HM} + I_{VP})] \qquad \textbf{2.2.105}$$

$$S_{33} = \frac{1}{4}[(I_{PP} + I_{MM}) - (I_{PM} + I_{MP})] \qquad \textbf{2.2.106}$$

$$S_{34} = \frac{1}{4}[(I_{LP} + I_{RM}) - (I_{LM} + I_{RP})] \qquad \textbf{2.2.107}$$

$$S_{41} = \frac{I_{UL} - I_{UR}}{2} \qquad \textbf{2.2.108}$$

$$S_{42} = \frac{1}{4}[(I_{HL} + I_{VR}) - (I_{HR} + I_{VL})] \qquad \textbf{2.2.109}$$

$$S_{43} = \frac{1}{4}[(I_{PL} + I_{MR}) - (I_{PR} + I_{ML})] \qquad \textbf{2.2.110}$$

$$S_{44} = \frac{1}{4}[(I_{LL} + I_{RR}) - (I_{LR} + I_{RL})] \qquad \textbf{2.2.111}$$

where I is the intensity measured by the photo-detector and the two subscript defines the arrangement of polarizer and analyzer respectively as follows: U - unpolarized light (no polarizer), H - linear polarizer at an angle 0° to the scattering plane, V - linear polarizer at an angle 90°, P - linear polarizer at an angle +45°, M - linear polarizer at an angle - 45° (or +135°), L - left handed circular polarizer and R - right handed circular polarizer.

For convenience all the matrix elements $S_{ij}(\theta)$ where i, j = 1 to 4, except $S_{11}(\theta)$ itself, are normalized to $S_{11}(\theta)$ i.e. the scattering matrix is reformed as [34, 36, 192],

$$S'(\theta) = \begin{pmatrix} S_{11}(\theta) & \frac{S_{12}(\theta)}{S_{11}(\theta)} & \frac{S_{13}(\theta)}{S_{11}(\theta)} & \frac{S_{14}(\theta)}{S_{11}(\theta)} \\ \frac{S_{21}(\theta)}{S_{11}(\theta)} & \frac{S_{22}(\theta)}{S_{11}(\theta)} & \frac{S_{23}(\theta)}{S_{11}(\theta)} & \frac{S_{24}(\theta)}{S_{11}(\theta)} \\ \frac{S_{31}(\theta)}{S_{11}(\theta)} & \frac{S_{32}(\theta)}{S_{11}(\theta)} & \frac{S_{33}(\theta)}{S_{11}(\theta)} & \frac{S_{34}(\theta)}{S_{11}(\theta)} \\ \frac{S_{41}(\theta)}{S_{11}(\theta)} & \frac{S_{42}(\theta)}{S_{11}(\theta)} & \frac{S_{43}(\theta)}{S_{11}(\theta)} & \frac{S_{44}(\theta)}{S_{11}(\theta)} \end{pmatrix} \qquad \textbf{2.2.112}$$

However it can be noted that $\left|\frac{S_{ij}(\theta)}{S_{11}(\theta)}\right| \leq 1$ [34, 36] . Further, the values of $S_{11}(\theta)$ are normalized such that they equal one at $\theta = 10°$. This normalization removes the magnitude effect in the light scattering at a given scattering angle θ and accentuates the polarization effects [336].

Table 2.2. The 49 combinations of delivering and collection optics which are necessary for measuring all the 16 elements of the scattering matrix [233]. ✳, ↔, ↕, ⤢, ⤡, ○ and ○ symbolize the unpolarized (U) light, linearly polarized light at an angle 0° (horizontal polarization, H), linearly polarized light at an angle 90° (vertical polarization, V), linearly polarized light at an angle +45° (P), linearly polarized light at an angle - 45° (M), left handed circularly polarized light (L) and right handed circularly polarized light respectively. The subscripts P and A denotes the polarizer and the analyzer.

✳$_P$✳$_A$	S_{11}	↔$_P$✳$_A$ ↕$_P$✳$_A$	$S_{11}+S_{12}$ $S_{11}-S_{12}$	⤢$_P$✳$_A$ ⤡$_P$✳$_A$	$S_{11}+S_{13}$ $S_{11}-S_{13}$	○$_P$✳$_A$ ○$_P$✳$_A$	$S_{11}+S_{14}$ $S_{11}-S_{14}$
✳$_P$↔$_A$ ✳$_P$↕$_A$	$S_{11}+S_{21}$ $S_{11}-S_{21}$	↔$_P$↔$_A$ ↔$_P$↕$_A$ ↕$_P$↔$_A$ ↕$_P$↕$_A$	$S_{11}+S_{12}+S_{21}+S_{22}$ $S_{11}+S_{12}-S_{21}-S_{22}$ $S_{11}-S_{12}+S_{21}-S_{22}$ $S_{11}-S_{12}-S_{21}+S_{22}$	⤢$_P$↔$_A$ ⤢$_P$↕$_A$ ⤡$_P$↔$_A$ ⤡$_P$↕$_A$	$S_{11}+S_{13}+S_{21}+S_{23}$ $S_{11}+S_{13}-S_{21}-S_{23}$ $S_{11}-S_{13}+S_{21}-S_{23}$ $S_{11}-S_{13}-S_{21}+S_{23}$	○$_P$↔$_A$ ○$_P$↕$_A$ ○$_P$↔$_A$ ○$_P$↕$_A$	$S_{11}+S_{14}+S_{21}+S_{24}$ $S_{11}+S_{14}-S_{21}-S_{24}$ $S_{11}-S_{14}+S_{21}-S_{24}$ $S_{11}-S_{14}-S_{21}+S_{24}$
✳$_P$⤢$_A$ ✳$_P$⤡$_A$	$S_{11}+S_{31}$ $S_{11}-S_{31}$	↔$_P$⤢$_A$ ↔$_P$⤡$_A$ ↕$_P$⤢$_A$ ↕$_P$⤡$_A$	$S_{11}+S_{12}+S_{31}+S_{32}$ $S_{11}+S_{12}-S_{31}-S_{32}$ $S_{11}-S_{12}+S_{31}-S_{32}$ $S_{11}-S_{12}-S_{31}+S_{32}$	⤢$_P$⤢$_A$ ⤢$_P$⤡$_A$ ⤡$_P$⤢$_A$ ⤡$_P$⤡$_A$	$S_{11}+S_{13}+S_{31}+S_{33}$ $S_{11}+S_{13}-S_{31}-S_{33}$ $S_{11}-S_{13}+S_{31}-S_{33}$ $S_{11}-S_{13}-S_{31}+S_{33}$	○$_P$⤢$_A$ ○$_P$⤡$_A$ ○$_P$⤢$_A$ ○$_P$⤡$_A$	$S_{11}+S_{14}+S_{21}+S_{34}$ $S_{11}+S_{14}-S_{21}-S_{34}$ $S_{11}-S_{14}+S_{21}-S_{34}$ $S_{11}-S_{14}-S_{21}+S_{34}$
✳$_P$○$_A$ ✳$_P$○$_A$	$S_{11}+S_{41}$ $S_{11}-S_{41}$	↔$_P$○$_A$ ↔$_P$○$_A$ ↕$_P$○$_A$ ↕$_P$○$_A$	$S_{11}+S_{12}+S_{31}+S_{42}$ $S_{11}+S_{12}-S_{31}-S_{42}$ $S_{11}-S_{12}+S_{31}-S_{42}$ $S_{11}-S_{12}-S_{31}+S_{42}$	⤢$_P$○$_A$ ⤢$_P$○$_A$ ⤡$_P$○$_A$ ⤡$_P$○$_A$	$S_{11}+S_{13}+S_{41}+S_{43}$ $S_{11}+S_{13}-S_{41}-S_{43}$ $S_{11}-S_{13}+S_{41}-S_{43}$ $S_{11}-S_{13}-S_{41}+S_{43}$	○$_P$○$_A$ ○$_P$○$_A$ ○$_P$○$_A$ ○$_P$○$_A$	$S_{11}+S_{14}+S_{41}+S_{44}$ $S_{11}+S_{14}-S_{41}-S_{44}$ $S_{11}-S_{14}+S_{41}-S_{44}$ $S_{11}-S_{14}-S_{41}+S_{44}$

In general the scattering matrix of particles in a volume element has 16 non zero and independent elements (equation 2.2.77). However in almost all practical cases symmetry reduces the number of matrix elements [192]. For a (relatively dense) cloud or aggregate of spherical or non-spherical particles oriented randomly relative to the direction of the incident wave, the Mueller matrix is a function of the scattering angle θ and reduces to eight non zero elements out of which six are independent [3, 5, 192, 261, 337],

$$\begin{pmatrix} I_s \\ Q_s \\ U_s \\ V_s \end{pmatrix} = \frac{1}{k^2 D^2} \begin{pmatrix} S_{11} & S_{12} & 0 & 0 \\ S_{12} & S_{22} & 0 & 0 \\ 0 & 0 & S_{33} & S_{34} \\ 0 & 0 & -S_{34} & S_{44} \end{pmatrix} \begin{pmatrix} I_i \\ Q_i \\ U_i \\ V_i \end{pmatrix} \qquad \textbf{2.2.113}$$

But orientation does not play a role on the light scattering properties of homogeneous spherical particles and in the case of a collection of optically inactive homogenous spherical particles, $S_{11} = S_{22}$ and $S_{33} = S_{44}$. Out of these six non-zero elements, the measurement of the elements S_{11} and $S_{12}(= S_{21})$ requires randomly polarized incident light. Thus these two elements are of utmost importance for the study of light scattering by atmospheric, interplanetary and cometary particulates where the incident light is randomly polarized (e.g. sunlight) [99]. Therefore for most of the light scattering applications including studies on atmospheric, interplanetary and cometary particles, it is sufficient to measure these two elements of the scattering matrix which provides the phase function and degree of linear polarization of the scattered light for unpolarized incident light [39, 99]. The first element S_{11} of the scattering matrix is also called the scattering function and is equivalent to the volume scattering function if the light is scattered by an ensemble of particles in a volume element.

2.2.2. Volume scattering function

The volume scattering function usually denoted as β [(solid angle)$^{-1}$ length^{-1}] is defined as [75 , 340],

$$\beta(\theta,\phi)=\frac{dJ(\theta,\phi)}{H_i dV} \qquad \textbf{2.2.114}$$

where $dJ(\theta,\phi)$ is an element of the radiant intensity [power (solid angle)$^{-1}$] scattered in a certain direction in space which can be defined by the scattering angle θ and azimuthal angle ϕ, dV is the volume element [(length)^3] and H_i is the irradiance [(power) (area)$^{-1}$] of the incident light. Thus, volume scattering function may be defined as the scattered intensity per unit irradiance of the incident light per unit scattering volume. For an ensemble of spatially symmetric axially symmetric particles the volume scattering function, β only depends on the scattering angle and reduces to [64],

$$\beta(\theta)=\frac{dJ(\theta)}{H_i dV} \qquad \textbf{2.2.115}$$

In many applications e.g. the theory of radiative transfer, the volume scattering function is normalized by the scattering coefficient, b to give the phase function, $p(\theta)$ [64]

$$p(\theta)=\frac{\beta(\theta)}{b} \qquad \textbf{2.2.116}$$

where b is the integral of the scattering function over all directions (i.e. full solid angle)

$$b=2\pi\int_0^{\pi}\beta(\theta)\sin\theta d\theta \qquad \textbf{2.2.117}$$

The phase function gives the probability of finding a photon scattered at a scattering angle θ.

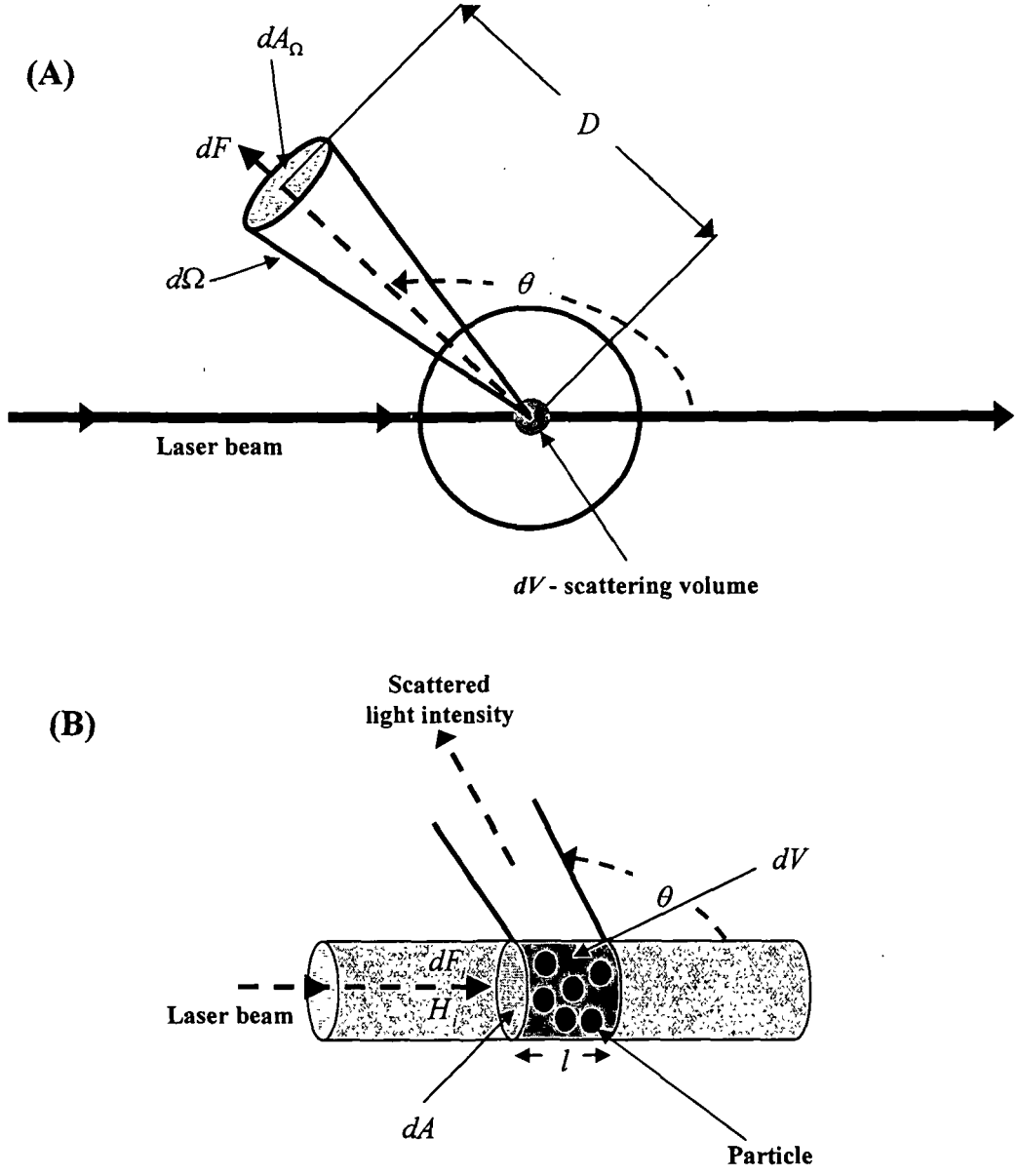

Figure 2.5. (A) Schematic diagram of the light scattering from N number of particles possessing identical optical properties inside a sample volume element dV with a face area dA when the volume is illuminated by a light beam of irradiance H_i (power dF_i). The elementary volume scatters power dF at scattering angle θ within an elementary solid angle $d\Omega$ which is at a distance l subtending an area dA_Ω. (B) Magnified view of the area under the black circle in figure 2.5 (A).

From figure 2.5, for a practical instrument where scattered light from a sample volume element, dV is collected at a scattering angle θ over a solid angle $d\Omega$, equation 2.2.115 can be rewritten as [75],

$$\beta(\theta) = \frac{\frac{dF(\theta)}{d\Omega}}{\frac{dF_i(0)}{dA}.dAl} = \frac{dF(\theta)}{dF_i(0)}.\frac{1}{d\Omega l} \qquad 2.2.118$$

where $F(\theta)$ is the power scattered into the solid angle $\Omega(\theta)$ in the direction θ, $F_i(0)$ is the power incident on the sample volume, and l is the length of the sample volume in the direction of incident radiation. The length, l of the scattering volume and the small solid angle $\Omega(\theta)$ over which the radiant intensity is collected and measured are determined by the optical geometry of the instrument [75, 342].

2.2.3. Relation between scattering matrix and volume scattering matrix

The scattering matrix described in the above section 2.2.1 represents the light scattering properties of a single particle in an ensemble of identical particles. Usually light scattering experiments involve a mixture of particles in a volume element which can be represented by using different distribution functions (such as gamma, normal, lognormal etc.) which will be described in detail in section 2.4.3. In case of light scattering from such dispersion or ensemble of particles, the scattering matrix is equivalent to the matrix form of the volume scattering function [342]. If the particles in the scattering volume are randomly oriented in space and sufficiently far from each other to scatter light in the single scattering regime, the scattering matrix represents the sum of the light scattering properties of all the particles in the scattering volume. Thus for light scattering from a volume element dV containing $dN = ndV$ number of identical particles where n is the number of particles per unit volume, equation 2.2.77 can be rewritten as,

$$\mathrm{I}_s(\theta) = \frac{ndV}{k^2 D^2} S(\theta) \mathrm{I}_i(\theta) \qquad \textbf{2.2.119}$$

Now each element of the incident and scattered Stokes vectors I_i and I_s has the dimension of irradiance. Hence from figure 2.5, the incident Stokes vector can be expressed as,

$$\begin{aligned} \mathrm{I}_i(\theta) &= \frac{\text{total power in the laser beam (in mW)}}{dA} \\ &= \frac{dF(\theta)}{dA_\Omega} \\ &= H_i(\theta) \end{aligned} \qquad \textbf{2.2.120}$$

where $H_i(\theta)$ is the irradiance incident on the scattering volume. Similarly the scattered Stokes vector I_s can be modified as,

$$\begin{aligned} \mathrm{I}_s(\theta) &= \frac{dF(\theta)}{dA_\Omega} \\ &= \frac{dF(\theta)}{d\Omega . D^2} \\ &= \frac{J(\theta)}{D^2} \end{aligned} \qquad \textbf{2.2.121}$$

where, $J(\theta)$ is the radiant intensity scattered at an angle θ and

$$d\Omega = \frac{dA}{D^2} \qquad \textbf{2.2.122}$$

Thus equation 2.2.119 can be rewritten as,

$$\mathrm{J}(\theta) = \left[\frac{n}{k^2} S(\theta)\right] dV \mathrm{H}_i(\theta) \qquad \textbf{2.2.123}$$

where $\mathrm{H}_i(\theta)$ and $\mathrm{J}(\theta)$ are the Stokes vectors of the incident [expressed in irradiance units] and scattered light [expressed in intensity units] respectively. Comparing with equation 2.2.114 we get,

$$\beta(\theta) = \frac{n}{k^2} S(\theta) \qquad \textbf{2.2.124}$$

where $\beta(\theta)$ is a matrix operator with 4×4 elements, i.e., a matrix form of the polarization dependent volume scattering function. For a polydispersion of particles,

$$\beta(\theta) = \frac{1}{k^2}\sum_{i=1,\quad,N} n_i S(\theta) \qquad \textbf{2.2.125}$$

Here index i is the number N of various size and shape of particles. The first element $\beta_{11}(\theta)$ of the $\beta(\theta)$ matrix represents the scattering properties of an ensemble of randomly oriented particles in a scattering volume element and in reality is a counterpart of the element $S_{11}(\theta)$ of the scattering matrix $S(\theta)$ [342]. Following traditional notation [1, 342] we will represent $\beta_{11}(\theta)$ as $\beta(\theta)$ from this point onward. Thus the volume scattering function for a polydispersed system can be defined as,

$$\beta(\theta) = \frac{1}{k^2}\sum_{i=1,\quad,N} n_i S_{11}(\theta) \qquad \textbf{2.2.126}$$

The above equation 2.2.126 is equivalent to the equation 2.4.138 in section 2.4 where the summation is replaced by integral for an ensemble or dispersion of particles having a wide range of particle sizes.

2.2.4. Quantities measured in this work

In this research work, light scattering measurements were performed on a multi-particle system i.e. dispersion of identical particles (e.g. aerosols, hydrosols and nanoparticles) under consideration. Such measurements are advantageous as they can be made *in situ* without a need for collecting samples and unlike single particle scattering measurements the detection system becomes simpler and less expensive due to the relatively higher intensity scattered beam obtained as compared to the light scattered by the a single particle [339]. As already mentioned in the first chapter the measurements were conducted in single elastic scattering regime on randomly oriented particles in a volume element. The measurements reported here are the volume scattering function $\beta(\theta)$ and the degree of linear polarization (or linear polarization ratio), $P(\theta)$. The volume scattering function $\beta(\theta)$ is proportional to the flux of the scattered light and is determined by measuring the scattered light intensity as functions of scattering

angle for unpolarized incident light as shown in equation 2.2.115. $\beta(\theta)$ is normalized to unity at 10° using the equation,

$$\beta_{norm} = \frac{\beta(\theta)}{\beta(10°)} \qquad \textbf{2.2.127}$$

For unpolarized incident light the degree of linear polarization is defined by the ratio, $-S_{12}/S_{11}$ [63, 99, 343] and is given by,

$$P(\theta) = -\frac{S_{12}(\theta)}{S_{11}(\theta)} = -\frac{\beta_{12}(\theta)}{\beta_{11}(\theta)} = \frac{I_{UV} - I_{UH}}{I_{UU}} \qquad \textbf{2.2.128}$$

Equation 2.2.128 is equivalent to the equation 2.2.65.

2.3. Cross sections and efficiencies

Suppose that one or more particles are placed in a beam of electromagnetic radiation and the rate at which electromagnetic energy is received by a detector D placed behind the particles is denoted by U. If the particles are removed, the power received by the detector is U_0, where $U_0 > U$. Thus the presence of the particles results in extinction of the incident beam. If the medium in which the particles are embedded is nonabsorbing, the difference $(U_0 - U)$ is accounted for absorption in the particles and scattering by the particles.

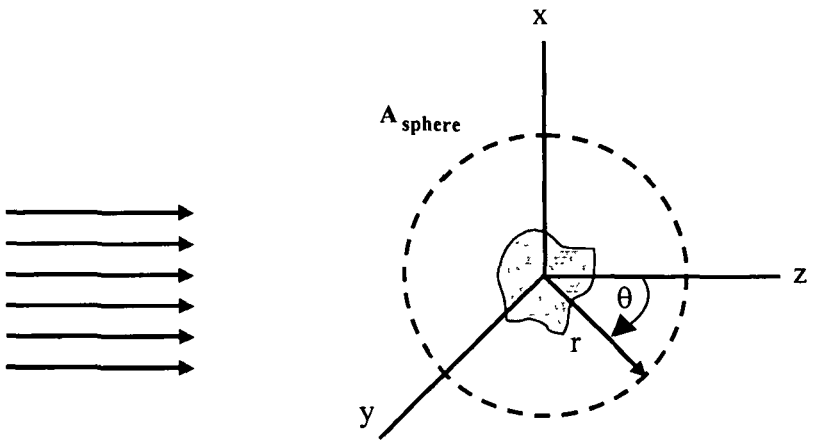

Figure 2.6. Extinction by a single particle

To investigate extinction by a single arbitrary particle embedded in a non-absorbing medium and illuminated by a plane wave an imaginary sphere of radius r is constructed around the particle as in figure 2.6. Then the net rate W_a at which electromagnetic energy crosses the surface A_{sph} of this sphere is

$$W_a = -\int_{A_{sph}} S.\hat{e}_r dA \qquad \textbf{2.3.1}$$

where S is the Poynting vector. If $W_a > 0$, energy is absorbed within the sphere. But the medium is non-absorbing, which implies that W_a is the rate at which energy is absorbed by the particle. Because of the Stoke's parameters 2.2.55 - 2.2.58 and Poynting vectors 2.2.67 - 2.2.69, W_a may be written as the sum of three terms

$$W_i = -\int_{A_{sph}} S_i.\hat{e}_r dA \qquad \textbf{2.3.2}$$

$$W_s = \int_{A_{sph}} S_s.\hat{e}_r dA \qquad \textbf{2.3.3}$$

and $$W_{ext} = -\int_{A_{sph}} S_{ext}.\hat{e}_r dA \qquad \textbf{2.3.4}$$

Now, W_i vanishes identically for a non-absorbing medium and W_s is the rate at which energy is scattered across the surface A_{sph}. Therefore W_{ext} is just the sum of the energy absorption rate and the energy scattering rate. Thus,

$$W_{ext} = W_a + W_s \qquad \textbf{2.3.5}$$

For convenience the incident electric field $E_i = E\hat{e}_x$ is taken to be x-polarized. Because the medium is non-absorbing, W_a is independent of the radius r of the imaginary sphere. Therefore r may be chosen sufficiently large such that it is in the far-field region where

$$\mathbf{E}_s \sim \frac{e^{ik(r-z)}}{-ikr}\mathbf{X}E \qquad \textbf{2.3.6}$$

$$\mathbf{H}_s \sim \frac{k}{\omega\mu}\hat{e}_r \times \mathbf{E}_s \qquad \textbf{2.3.7}$$

and $\hat{e}_r.X = 0$. As a reminder that the incident light is x-polarized the symbol X is used for the vector scattering amplitude. The limiting values of W_{ext} as $kr \to \infty$ is therefore [2]

$$W_{ext} = I_i\left(\frac{4\pi}{k^2}\right)\mathrm{Re}\{(X.\hat{e}_x)_{\theta=0}\} \qquad \textbf{2.3.8}$$

where I_i is the incident irradiance. Now C_{ext}, defined as the cross-section of extinction, with dimensions of area, then becomes

$$C_{ext} = \frac{W_{ext}}{I_i} = \left(\frac{4\pi}{k^2}\right)\mathrm{Re}\{(X.\hat{e}_x)_{\theta=0}\} \qquad \textbf{2.3.9}$$

It follows from 2.3.5 that the C_{ext} may be written as the sum of C_{abs}, the cross-section of absorption, and C_{sca}, the cross-section of scattering. Hence,

$$C_{abs} = C_{ext} - C_{sca} \qquad \textbf{2.3.10}$$

where $C_{abs} = \frac{W_{abs}}{I_i}$ **2.3.11**

and $C_{sca} = \frac{W_s}{I_i}$ **2.3.12**

We may define efficiencies or efficiency factors for extinction, scattering and absorption [2, 3, 5]:

$$Q_{ext} = \frac{C_{ext}}{G} \qquad \textbf{2.3.13}$$

$$Q_{sca} = \frac{C_{sca}}{G} \qquad \textbf{2.3.14}$$

$$Q_{abs} = \frac{C_{abs}}{G} \qquad \textbf{2.3.15}$$

where G is the geometrical cross sectional area of the particle, projected onto a plane perpendicular to the direction of incident light. As the particle may have non-spherical (or rather irregular) shapes, the G value can change abruptly with orientation of the particle in respect to the direction of the incident beam. Therefore, in many electromagnetic theories like DDA, T-matrix approach, the G for a non-spherical particle is replaced by the projection area of the volume or surface area equivalent sphere. If V is the volume of an arbitrary shaped particle, the effective radius of an equal volume sphere is [126],

$$a_{eff} = \sqrt[3]{\frac{3V}{4\pi}} \qquad \textbf{2.3.16}$$

and the projection area of the monodisperse sphere is

$$G_{eff} = \pi a_{eff}^2 \neq G \qquad \textbf{2.3.17}$$

then we have modified efficiency factors in the form [126]

$$Q_{ext}^{eff} = \frac{C_{ext}}{G_{eff}} = \frac{C_{ext}}{\pi a_{eff}^2} \qquad \textbf{2.3.18}$$

$$Q_{sca}^{eff} = \frac{C_{sca}}{G_{eff}} = \frac{C_{sca}}{\pi a_{eff}^2} \qquad \textbf{2.3.19}$$

$$Q_{abs}^{eff} = \frac{C_{abs}}{G_{eff}} = \frac{C_{abs}}{\pi a_{eff}^2} \qquad \textbf{2.3.20}$$

For spherical particles $G \equiv G_{eff}$.

Now, from **2.3.2 - 2.3.4** and **2.3.6 - 2.3.7** we have,

$$C_{sca} = \int_0^{2\pi}\int_0^{\pi} \frac{|\mathbf{X}|^2}{k^2 r^2} r^2 \sin\theta \, d\theta \, d\phi = \int_{4\pi} \frac{|\mathbf{X}|^2}{k^2} d\Omega \qquad \textbf{2.3.21}$$

The quantity $\frac{|\mathbf{X}|^2}{k^2}$ is the differential scattering cross-section and symbolically denoted by $\frac{dC_{sca}}{d\Omega}$ which physically specifies the angular distribution of the

scattered light, that is, the amount of light for unit incident irradiance scattered into a unit solid angle about a given direction. Notably, for an isotropic medium such as a collection of randomly oriented particles, which may themselves be anisotropic, the scattered intensity and hence the differential scattering cross section is independent of the azimuthal angle, φ. For such a system of particles we can also define the differential scattering cross-section, $\frac{dC_{sca}}{d\Omega}$ in terms of dimensionless scattered irradiances i_1 and i_2 for unpolarized incident light as [2, 29],

$$\frac{dC_{sca}}{d\Omega} = \frac{S_{11}}{k^2} = \frac{(i_1 + i_2)}{2k^2} \qquad \textbf{2.3.22}$$

In the modeling of transfer of electromagnetic radiation through a medium, it is advantageous to introduce the terms phase function, single scattering albedo and asymmetry parameter which are important to describe the optical properties of the medium [126]. The phase function is a normalized measurement of how the intensity of scattered light varies with the scattering angle [2, 126, 234] and is defined as

$$p = \frac{4\pi}{C_{sca}} \frac{|X^2|}{k^2} \qquad \textbf{2.3.23}$$

and the normalization condition is,

$$\frac{1}{4\pi} \int_{4\pi} p \, d\Omega = 1 \qquad \textbf{2.3.24}$$

The average cosine of the scattering angle, or the asymmetry parameter g is defined as

$$g = <\cos\theta> = \frac{1}{4\pi} \int_{4\pi} p \cos\theta \, d\Omega \qquad \textbf{2.3.25}$$

Its value vanishes when the particle scatters light isotropically or symmetrically about a scattering angle of 90 degree. If the particle scatters more

in the forward direction ($\theta = 0$), g is positive and if the particle scatters more in the backward direction ($\theta = 180$), g is negative.

The single scattering albedo is defined as the ratio of the scattering and extinction cross sections, i.e.,

$$\varpi = \frac{C_{sca}}{C_{ext}} \qquad \textbf{2.3.26}$$

The range of values is $0 \leq \varpi \leq 1$.

In addition to energy, light carries momentum. Therefore a beam of light that interacts with a particle causes the transfer of momentum from the electromagnetic field to the scattering particle. The resulting force is called the radiation pressure. The radiation pressure cross section is defined as,

$$C_{pr} = C_{ext} - <\cos\theta> C_{sca} \qquad \textbf{2.3.27}$$

By analogy with equations 2.3.18 - 2.3.20, we can define radiation pressure efficiency as

$$Q_{pr}^{eff} = \frac{C_{pr}}{G_{eff}} = \frac{C_{pr}}{\pi a_{eff}^2} \qquad \textbf{2.3.28}$$

2.4. Mie theory - the spherical basis

As Mie theory [100 - 102] has been applied successfully to explain extinction and scattering by spherical particles and as many of the samples (like water droplets and polystyrene particles) used in the experiments conducted by using the light scattering setup described in chapter III falls in this category of spherical particles at the detectable levels, a detailed description of Mie theory is presented in this section. Rayleigh and Raman scattering are left out as the particles being investigated are neither very small nor has the appropriate density to be able to produce detectable levels of Raman scattering signals.

2.4.1. The vector wave equations

In a linear, isotropic, homogenous, charge free medium the vector wave equations developed from Maxwell's equations reduces to,

$$\nabla^2 E(r) + k^2 E(r) = 0 \qquad \textbf{2.4.1}$$

$$\nabla^2 H(r) + k^2 H(r) = 0 \qquad \textbf{2.4.2}$$

$$\nabla . E(r) = 0 \qquad \textbf{2.4.3}$$

$$\nabla . H(r) = 0 \qquad \textbf{2.4.4}$$

where

$$k^2 = \omega^2 \varepsilon \mu \qquad \textbf{2.4.5}$$

$$\nabla \times E(r) = i\omega\mu H(r) \qquad \textbf{2.4.6}$$

$$\nabla \times H(r) = -i\omega\mu E(r) \qquad \textbf{2.4.7}$$

Now, one can construct a vector function

$$M(r) = \nabla \times [c\psi(r)] \qquad \textbf{2.4.8}$$

where c is a constant vector and ψ is any scalar function. Since $\nabla . (\nabla \times v) = 0$ for any vector function v, we have,

$$\nabla . M(r) = 0 \qquad \textbf{2.4.9}$$

Substituting $E(r)$ by $M(r)$, one can write equation 2.4.1 as follows [2] (see Appendix A),

$$\left[\nabla^2 + k^2\right] M(r) = \nabla \times \left\{c\left[\nabla^2 \psi(r) + k^2 \psi(r)\right]\right\} \qquad \textbf{2.4.10}$$

Thus M is a solution to the vector wave equation, if ψ is a solution to the scalar wave equation,

$$\left[\nabla^2 + k^2\right] \psi(r) = 0 \qquad \textbf{2.4.11}$$

We may also write, $M = -c \times \nabla \psi$, which shows M is perpendicular to c. We can define a second vector function,

$$N(r) = \frac{\nabla \times M(r)}{k} \qquad \textbf{2.4.12}$$

with zero divergence, which also satisfies the vector wave equation,

$$\left[\nabla^2 + k^2\right] N(r) = 0 \qquad \textbf{2.4.13}$$

We also have,

$$\nabla \times N(r) = kM(r) \qquad \textbf{2.4.14}$$

As M and N satisfy the vector wave equation, they are divergence free, the curl of M is proportional to N and curl of N is proportional to M, the pair (M,N) can be applied to construct electric E and magnetic H fields. The scalar function ψ is termed as the generating function for the vector harmonics M and N and the arbitrary vector c is called the guiding or the pilot vector. Now changing the arbitrary pilot vector c to the radius vector, r we have solutions to the vector wave equation in spherical polar coordinates. Using the following transformation between Cartesian and spherical coordinates,

$$\begin{aligned} x &= r\sin\theta\cos\phi \\ y &= r\sin\theta\sin\phi \\ z &= r\cos\phi \end{aligned} \qquad \textbf{2.4.15}$$

we get the scalar wave equation in spherical polar coordinate as [2],

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial\psi}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\psi}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2\psi}{\partial\phi^2} + k^2\psi = 0 \qquad \textbf{2.4.16}$$

Equation 2.4.16 is separable when

$$\psi(r,\theta,\phi) = R(r)\Theta(\theta)\Phi(\phi) \qquad \textbf{2.4.17}$$

Substituting equation 2.4.17 into equation 2.4.16 and dividing the entire equation by $\psi(r,\theta,\phi)$, we get [347]

$$\frac{1}{r^2}\frac{1}{R}\frac{\partial}{\partial r}\left(r^2\frac{\partial R}{\partial r}\right)+\frac{1}{r^2\sin\theta}\frac{1}{\Theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Theta}{\partial\theta}\right)+\frac{1}{r^2\sin^2\theta}\frac{1}{\Phi}\frac{\partial^2\Phi}{\partial\phi^2}+k^2=0$$

2.4.18

Again multiplying equation 2.4.18 by $r^2\sin^2\theta$, we get

$$\left[\sin^2\theta\frac{1}{R}\frac{\partial}{\partial r}\left(r^2\frac{\partial R}{\partial r}\right)+\sin\theta\frac{1}{\Theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Theta}{\partial\theta}\right)+k^2r^2\sin^2\theta\right]+\frac{1}{\Phi}\frac{\partial^2\Phi}{\partial\phi^2}=0$$

2.4.19

Equation 2.4.19 is valid only if,

$$\frac{1}{\Phi}\frac{\partial^2\Phi}{\partial\phi^2}=\text{constant}=-\kappa \qquad \textbf{2.4.20}$$

Also,

$$\sin^2\theta\frac{1}{R}\frac{\partial}{\partial r}\left(r^2\frac{\partial R}{\partial r}\right)+\sin\theta\frac{1}{\Theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Theta}{\partial\theta}\right)+k^2r^2\sin^2\theta-\kappa=0$$

2.4.21

Dividing equation 2.4.21 by $\sin^2\theta$, we get

$$\left[\frac{1}{R}\frac{\partial}{\partial r}\left(r^2\frac{\partial R}{\partial r}\right)+k^2r^2\right]+\left[\frac{1}{\sin\theta}\frac{1}{\Theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Theta}{\partial\theta}\right)-\frac{\kappa}{\sin^2\theta}\right]=0 \qquad \textbf{2.4.22}$$

Evidently, equation 2.4.22 will be valid provided,

$$\frac{1}{R}\frac{\partial}{\partial r}\left(r^2\frac{\partial R}{\partial r}\right)+k^2r^2=\text{constant}=\Lambda \qquad \textbf{2.4.23}$$

and $$\frac{1}{\sin\theta}\frac{1}{\Theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Theta}{\partial\theta}\right)-\frac{\kappa}{\sin^2\theta}=\text{constant}=-\Lambda \qquad \textbf{2.4.24}$$

The selection of the constants κ and Λ are for mathematical convenience. Rearranging equations 2.4.20, 2.4.23 and 2.4.24, we obtain,

$$\frac{d^2\Phi}{d\phi^2} + \kappa\Phi = 0 \qquad \textbf{2.4.25}$$

$$\frac{1}{\sin\theta}\frac{d}{d\theta}\left(\sin\theta\frac{d\Theta}{d\theta}\right) + \left(\Lambda - \frac{\kappa}{\sin^2\theta}\right)\Theta = 0 \qquad \textbf{2.4.26}$$

$$\frac{d}{dr}\left(r^2\frac{dR}{dr}\right) + \left[k^2r^2 - \Lambda\right]R = 0 \qquad \textbf{2.4.27}$$

Now, ϕ varies from 0 to 2π and therefore the solution of equation 2.4.25 must show the periodicity, $\Theta(0) = \Theta(2\pi)$. It is noteworthy to mention that the equation 2.4.25 is equivalent to the one dimensional Helmholtz equation when $\kappa = m^2$, where $m \in Z$ is a arbitrary integer number ($m = 0,1,.....$). Thus, the general solution of the equation 2.4.25 is the linear combination of $\cos\left(\sqrt{\kappa}\phi\right) = \cos(m\phi)$ and $\sin\left(\sqrt{\kappa}\phi\right) = \sin(m\phi)$. Therefore, the two non trivial and linearly independent solutions are,

(a) even solution:

$$\Phi_e(\phi) = \cos(m\phi) \qquad \textbf{2.4.28}$$

(b) odd solution:

$$\Phi_o(\phi) = \sin(m\phi) \qquad \textbf{2.4.29}$$

where the subscripts e and o denotes even and odd.

Substituting $\xi = \cos\theta$ in equation 2.4.26, we obtain,

$$\begin{aligned}&\left(1-\xi^2\right)\frac{d^2\Theta}{d\xi^2} - 2\xi\frac{d\Theta}{d\xi} + \left(\Lambda - \frac{\kappa}{1-\xi^2}\right)\Theta = 0 \\ &\Rightarrow \frac{d\Theta}{d\xi}\left[\left(1-\xi^2\right)\frac{d\Theta}{d\xi}\right] + \left(\Lambda - \frac{m^2}{1-\xi^2}\right)\Theta = 0\end{aligned} \qquad \textbf{2.4.30}$$

Equation 2.4.30 represents the general Legendre equation which has a finite solution only when $\Lambda = n(n+1)$ where n is an integer. The solutions to equation 2.4.30 that are finite at $\theta = 0$ and $\theta = \pi$ are the associated Legendre functions of the first kind $P_n^m(\xi)$ of degree n and order m where $n = m, m+1, \ldots\ldots$ related to the corresponding Legendre polynomial, $P_n(\xi)$ by the equation [126, 235, 348],

$$\Theta = P_n^m(\xi) = (1-\xi^2)^{|m|/2} \frac{d^{|m|}}{d\xi^{|m|}} P_n(\xi) \qquad \textbf{2.4.31}$$

where $|m| \le n$ and $\xi = \cos\theta$ and

$$P_n(\xi) = \frac{1}{2^n n!} \frac{d^n}{d^n \xi} (\xi^2 - 1)^n \qquad \textbf{2.4.32}$$

Thus, for $m = 0$ the associated Legendre functions reduces to Legendre polynomials. Notably, the phase factor $(-1)^m$ which is sometimes used in the definition of associated Legendre functions [5] is omitted in the present derivations. Equation 2.4.32 is well known as Rodrigue's formula. The associated Legendre functions satisfy the orthogonality relation ($|m| \le n$ or n'),

$$\int_{-1}^{1} P_n^m(\xi) P_{n'}^m(\xi) d\xi = \frac{2(n+m)!}{(2n+1)(n-m)!} \delta_{n'n} \qquad \textbf{2.4.33}$$

where $\delta_{n'n}$ is the Kronecker delta which is unity if $n = n'$ and zero otherwise.

Substituting $\kappa = m^2$ and $\Lambda = n(n+1)$ in equations 2.4.25, 2.4.26 and 2.4.27, we obtain,

$$\frac{d^2\Phi}{d\phi^2} + m^2\Phi = 0 \qquad \textbf{2.4.34}$$

$$\frac{1}{\sin\theta} \frac{d}{d\theta}\left(\sin\theta \frac{d\Theta}{d\theta}\right) + \left(n(n+1) - \frac{m^2}{\sin^2\theta}\right)\Theta = 0 \qquad \textbf{2.4.35}$$

$$\frac{d}{dr}\left(r^2 \frac{dR}{dr}\right) + \left[k^2 r^2 - n(n+1)\right]R = 0 \qquad \textbf{2.4.36}$$

Setting $kr = \rho$ and $R = \frac{1}{\sqrt{\rho}} Z(\rho)$, in equation 2.4.36, we obtain

$$\rho \frac{d}{d\rho}\left(\rho \frac{dZ(\rho)}{d\rho}\right) + \left[\rho^2 - \left(n+\frac{1}{2}\right)^2\right] Z(\rho) = 0$$

$$\Rightarrow \rho^2 \frac{d^2 Z(\rho)}{d\rho^2} + \rho \frac{dZ(\rho)}{d\rho} + \left[\rho^2 - \left(n+\frac{1}{2}\right)^2\right] Z(\rho) = 0 \qquad \textbf{2.4.37}$$

$$\Rightarrow \rho^2 \frac{d^2 Z(\rho)}{d\rho^2} + \rho \frac{dZ(\rho)}{d\rho} + \left[\rho^2 - \nu^2\right] Z(\rho) = 0$$

where $\nu = n + \frac{1}{2}$. Equation 2.4.37 is the Bessel differential equation. The solutions to equation 2.4.37 are the Bessel functions of the first kind, J_ν and the second kind, Y_ν (occasionally denoted as Neumann function, N_ν) of order $\nu = 1 + \frac{1}{2}$ is half integral. Therefore the linearly independent solutions to equation 2.4.37 are the spherical Bessel functions of the first kind and the second kind [2, 5, 126, 347],

$$j_n(\rho) = \sqrt{\frac{\pi}{2\rho}} J_{n+\frac{1}{2}}(\rho) = \rho^n \left(-\frac{1}{\rho}\frac{d}{d\rho}\right)^n \left(\frac{\sin\rho}{\rho}\right) \qquad \textbf{2.4.38}$$

$$y_n(\rho) = \sqrt{\frac{\pi}{2\rho}} Y_{n+\frac{1}{2}}(\rho) = -\rho^n \left(-\frac{1}{\rho}\frac{d}{d\rho}\right)^n \left(\frac{\cos\rho}{\rho}\right) \qquad \textbf{2.4.39}$$

Now the spherical Bessel function satisfy the recurrence relations

$$z_{n-1}(\rho) + z_{n+1}(\rho) = \frac{2n+1}{\rho} z_n(\rho) \qquad \textbf{2.4.40}$$

and $$(2n+1)\frac{d}{d\rho} z_n(\rho) = n z_{n-1}(\rho) - (n+1) z_{n+1}(\rho) \qquad \textbf{2.4.41}$$

where z_n is either j_n or y_n.

From equations 2.4.38 and 2.4.39 the first two orders of the spherical Bessel functions can be found as,

$$j_0(\rho) = \frac{\sin\rho}{\rho}\ ,\ j_1(\rho) = \frac{\sin\rho}{\rho^2} - \frac{\cos\rho}{\rho} \qquad \textbf{2.4.42}$$

and

$$y_0(\rho)=-\frac{\cos\rho}{\rho},\ y_1(\rho)=-\frac{\cos\rho}{\rho^2}-\frac{\sin\rho}{\rho} \qquad \textbf{2.4.43}$$

The higher order terms can be generated by using the recursion relations. To provide a more general solution the complex valued spherical Bessel functions of third kind are used which are given by,

$$h_n^{(1)}(\rho)=j_n(\rho)+iy_n(\rho) \qquad \textbf{2.4.44}$$

and $$h_n^{(2)}(\rho)=j_n(\rho)-iy_n(\rho) \qquad \textbf{2.4.45}$$

where $h_n^{(1)}(\rho)$ and $h_n^{(2)}(\rho)$ are usually termed as spherical Hankel functions of first kind and second kind respectively.

Thus the complete solution of equation 2.4.16 is given by the even and odd scalar solutions,

$$\psi_{emn}(kr,\theta,\varphi)=\cos(m\varphi)P_n^m(\cos\theta)z_n(kr) \qquad \textbf{2.4.46}$$

$$\psi_{omn}(kr,\theta,\varphi)=\sin(m\varphi)P_n^m(\cos\theta)z_n(kr) \qquad \textbf{2.4.47}$$

where z_n is any of the four spherical Bessel functions j_n, y_n, $h_n^{(1)}$, and $h_n^{(2)}$ and P_n^m is the associated Legendre functions. Inserting generating functions ψ_{emn} and ψ_{omn} into equations 2.4.8 and 2.4.12 we obtain for the vector harmonics M and N, being given by the relations,

$$M_{emn}=\frac{-m}{\sin\theta}\sin m\phi P_n^m(\cos\theta)z_n(\rho)\hat{\mathbf{e}}_\theta-\cos m\phi\frac{dP_n^m(\cos\theta)}{d\theta}z_n(\rho)\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.48}$$

$$M_{omn}=\frac{m}{\sin\theta}\cos m\phi P_n^m(\cos\theta)z_n(\rho)\hat{\mathbf{e}}_\theta-\sin m\phi\frac{dP_n^m(\cos\theta)}{d\theta}z_n(\rho)\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.49}$$

$$N_{emn} = \frac{z_n(\rho)}{\rho}\cos m\phi\, n(n+1)P_n^m(\cos\theta)\hat{\mathbf{e}}_r + \cos m\phi \frac{dP_n^m(\cos\theta)}{d\theta}\frac{1}{\rho}\frac{d}{d\rho}[\rho z_n(\rho)]\hat{\mathbf{e}}_\theta - m\sin m\phi \frac{P_n^m(\cos\theta)}{\sin\theta}\frac{1}{\rho}\frac{d}{d\rho}[\rho z_n(\rho)]\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.50}$$

$$N_{omn} = \frac{z_n(\rho)}{\rho}\sin m\phi\, n(n+1)P_n^m(\cos\theta)\hat{\mathbf{e}}_r + \sin m\phi \frac{dP_n^m(\cos\theta)}{d\theta}\frac{1}{\rho}\frac{d}{d\rho}[\rho z_n(\rho)]\hat{\mathbf{e}}_\theta + m\cos m\phi \frac{P_n^m(\cos\theta)}{\sin\theta}\frac{1}{\rho}\frac{d}{d\rho}[\rho z_n(\rho)]\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.51}$$

where P_n^m are the Associated Legendre Polynomials of degree n and order m,

z_n is a Bessel function,

$\rho = kr$, k being the propagation constant, is a dimensionless parameter.

2.4.2. Expansion of the electric and magnetic fields in vector spherical harmonics

As it is known, Mie theory begins by expanding the incident plane x-polarized wave

$$E_i = E_0 e^{i(kz-\omega t)}\hat{e}_x \qquad \textbf{2.4.52}$$

written in spherical polar co-ordinates as

$$E_i = E_0 e^{ikr\cos\theta}\hat{e}_x \qquad \textbf{2.4.53}$$

where

$$\hat{e}_x = \sin\theta\cos\phi\hat{e}_r + \cos\theta\cos\phi\hat{e}_\theta + \sin\theta\hat{e}_\phi \qquad \textbf{2.4.54}$$

in vector spherical harmonics as,

$$\mathbf{E}_i = \sum_{m=0}^{\infty}\sum_{n=m}^{\infty}(a_{emn}\mathbf{M}_{emn} + a_{omn}\mathbf{M}_{omn} + b_{emn}\mathbf{N}_{emn} + b_{omn}\mathbf{N}_{omn}) \qquad \textbf{2.4.55}$$

Thus for internal and scattered waves,

$$\mathbf{E}_1 = \sum_{m=0}^{\infty}\sum_{n=m}^{\infty}(c_{emn}\mathbf{M}_{emn} + c_{omn}\mathbf{M}_{omn} + d_{emn}\mathbf{N}_{emn} + d_{omn}\mathbf{N}_{omn}) \qquad \textbf{2.4.56}$$

$$\mathbf{E}_s = \sum_{m=0}^{\infty}\sum_{n=m}^{\infty}(p_{emn}\mathbf{M}_{emn} + p_{omn}\mathbf{M}_{omn} + q_{emn}\mathbf{N}_{emn} + q_{omn}\mathbf{N}_{omn}) \qquad \textbf{2.4.57}$$

where a_{emn}, a_{omn}, b_{emn}, b_{omn} are the expansion co-efficients of the incident field; c_{emn}, c_{omn}, d_{emn}, d_{omn} are the expansion co-efficients of the internal field and p_{emn}, p_{omn}, q_{emn}, q_{omn} are the expansion co-efficients of the scattered field.

The orthogonality of all the vector spherical harmonics implies that the expansion co-efficients in 2.4.55 are of the form

$$a_{emn} = \frac{\int_0^{2\pi}\int_0^{\pi}\mathbf{E}_i.\mathbf{M}_{emn}\sin\theta\, d\theta\, d\phi}{\int_0^{2\pi}\int_0^{\pi}|\mathbf{M}_{emn}|^2\sin\theta\, d\theta\, d\phi} \qquad \textbf{2.4.58}$$

with similar expressions for a_{omn}, b_{emn} and b_{omn}. It follows from 2.4.48, 2.4.51 and 2.4.54 together with the orthogonality of sine and cosine that $a_{emn} = b_{omn} = 0$ for all m and n. Moreover, the remaining co-efficients vanish unless $m = 1$ for the same reason. Notably, for all orders n, the functions $j_n(kr)$ is regular in every finite domain of the ρ-plane including the origin, whereas the functions $y_n(kr)$ have singularities at the origin $\rho = 0$ where they become infinite. Hence to represent the incident wave, we use $j_n(kr)$ but not $y_n(kr)$. Hence the superscript (1) will be appended to vector spherical harmonics involving $j_n(kr)$. Thus the expansion for the incident field has the form,

$$E_i = \sum_{n=1}^{\infty}\left(a_{o1n}M_{o1n}^{(1)} + b_{e1n}N_{e1n}^{(1)}\right) \qquad \textbf{2.4.59}$$

The expansion co-efficients that finally remain after evaluation are,

$$a_{oln} = i^n E_0 \frac{2n+1}{n(n+1)} \qquad \textbf{2.4.60}$$

and

$$b_{eln} = -iE_0 i^n \frac{2n+1}{n(n+1)} \qquad \textbf{2.4.61}$$

Thus the expansion of the plane wave in spherical harmonics finally takes the form

$$\begin{aligned} \mathbf{E}_i &= E_0 \sum_{n=1}^{\infty} i^n \frac{2n+1}{n(n+1)} (\mathbf{M}_{oln}^{(1)} - i\mathbf{N}_{eln}^{(1)}) \\ &= \sum_{n=1}^{\infty} E_n (\mathbf{M}_{oln}^{(1)} - i\mathbf{N}_{eln}^{(1)}) \end{aligned} \qquad \textbf{2.4.62}$$

where

$$E_n = i^n E_0 \frac{(2n+1)}{n(n+1)} \qquad \textbf{2.4.63}$$

and the corresponding incident magnetic field is obtained from the curl of 2.4.62 as

$$\begin{aligned} \mathbf{H}_i &= \frac{-k}{\omega\mu} E_0 \sum_{n=1}^{\infty} i^n \frac{2n+1}{n(n+1)} (\mathbf{M}_{eln}^{(1)} + i\mathbf{N}_{oln}^{(1)}) \\ &= -\frac{k}{\omega\mu} \sum_{n=1}^{\infty} E_n (\mathbf{M}_{eln}^{(1)} + i\mathbf{N}_{oln}^{(1)}) \end{aligned} \qquad \textbf{2.4.64}$$

Now the boundary conditions at the boundary between the spherical particle and the medium are given as [1, 2],

$$(E_i + E_s - E_1) \times \hat{e}_r = (H_i + H_s - H_1) \times \hat{e}_r = 0 \qquad \textbf{2.4.65}$$

As was in the case of incident fields, the inner fields inside the sphere are required to be finite (regular) at the origin. Hence the spherical Bessel function $j_n(k_1 r)$ where k_1 is the wave number inside the sphere, is the only acceptable representation of z_n. Thus when the plane x-polarized wave is incident on a

homogeneous, isotropic sphere of radius r, the fields inside the particle are given as

$$\mathbf{E}_1 = \sum_{n=1}^{\infty} E_n (c_n \mathbf{M}_{o1n}^{(1)} - i d_n \mathbf{N}_{e1n}^{(1)}) \qquad \textbf{2.4.66}$$

and $$\mathbf{H}_1 = \frac{-k}{\omega \mu_1} \sum_{n=1}^{\infty} E_n (d_n \mathbf{M}_{e1n}^{(1)} + i c_n \mathbf{N}_{o1n}^{(1)}) \qquad \textbf{2.4.67}$$

where μ_1 is the permeability of the sphere and c_n and d_n are the expansion coefficients.

In the region outside the sphere both j_n and y_n are well behaved for which the expansion for the scattered fields involve both these functions. Therefore it is convinient to use the spherical hankel functions $h_n^{(1)}(kr)$ and $h_n^{(2)}(kr)$ for the expansion of the scattered fields which are given asymptotically by

$$h_n^{(1)}(kr) \underset{kr \to \infty}{\sim} \frac{(-i)^n e^{ikr}}{ikr}, \qquad kr >> n^2 \qquad \textbf{2.4.68}$$

$$h_n^{(2)}(kr) \underset{kr \to \infty}{\sim} -\frac{(i)^n e^{-ikr}}{ikr}, \qquad kr >> n^2 \qquad \textbf{2.4.69}$$

However only the first function, $h_n^{(1)}(kr)$ given by the equation 2.4.68 is required for the expansion for the scattered fields as only this asymptotic expression corresponds to an outgoing spherical wave while the second function, $h_n^{(2)}(kr)$ corresponds to an incoming spherical wave. At large distances the asymptotic expression for the derivative of $h_n^{(1)}(kr)$ is given by,

$$\frac{dh_n^{(1)}(kr)}{d(kr)} \underset{kr \to \infty}{\sim} \frac{(-i)^n e^{ikr}}{kr} \sim i h_n^{(1)}(kr) \qquad \textbf{2.4.70}$$

Thus the scattered field expressions are given by,

$$\mathbf{E}_s = \sum_{n=1}^{\infty} E_n (i q_n \mathbf{N}_{e1n}^{(3)} - p_n \mathbf{M}_{o1n}^{(3)}) \qquad \textbf{2.4.71}$$

and $$\mathbf{H}_s = \frac{k}{\omega\mu}\sum_{n=1}^{\infty} E_n(iq_n\mathbf{N}_{o1n}^{(2)} + p_n\mathbf{M}_{e1n}^{(2)}) \qquad \textbf{2.4.72}$$

where p_n and q_n are the expansion co-efficients and the superscript (3) appended to the vector spherical harmonics indicate that the Bessel function $z_n(kr)$ in these harmonics is the spherical Hankel function, $h_n^{(1)}(kr)$.

For simplifying the vector spherical harmonics it is found to be convenient to define two functions

$$\pi_n = \frac{P_n^1}{\sin\theta} \qquad \textbf{2.4.73}$$

and $$\tau_n = \frac{d}{d\theta}P_n^1 \qquad \textbf{2.4.74}$$

which can be computed by begining with $\pi_0 = 0$ and $\pi_1 = 1$ since they follow the recurrence relations

$$\pi_n = \frac{2n-1}{n-1}\mu\pi_{n-1} - \frac{n}{n-1}\pi_{n-2} \qquad \textbf{2.4.75}$$

$$\tau_n = n\mu\pi_n - (n+1)\pi_{n-1} \qquad \textbf{2.4.76}$$

where

$$\mu = \cos\theta \qquad \textbf{2.4.77}$$

$$\pi_n(-\mu) = (-1)^{n-1}\pi_n(\mu) \qquad \textbf{2.4.78}$$

$$\tau_n(-\mu) = (-1)^{n-1}\tau_n(\mu) \qquad \textbf{2.4.79}$$

and $\pi_n + \tau_n$ and $\pi_n - \tau_n$ are orthogonal **2.4.80**

The vector spherical harmonics 2.4.48- 2.4.51 with $m = 1$ in the expansions of the internal fields 2.4.66 and 2.4.67 and the scattered fields 2.4.71 and 2.4.72 can now be written in a more concise form as

$$\mathbf{M}_{o1n} = \cos\phi\,\pi_n(\cos\theta)z_n(\rho)\hat{\mathbf{e}}_\theta - \sin\phi\,\tau_n(\cos\theta)z_n(\rho)\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.81}$$

$$\mathbf{M}_{eln} = -\sin\phi\,\pi_n(\cos\theta)z_n(\rho)\hat{\mathbf{e}}_\theta - \cos\phi\,\tau_n(\cos\theta)z_n(\rho)\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.82}$$

$$\mathbf{N}_{oln} = \sin\phi\, n(n+1)\sin\theta\,\pi_n(\cos\theta)\frac{z_n(\rho)}{\rho}\hat{\mathbf{e}}_r + \sin\phi\,\tau_n(\cos\theta)\frac{[\rho\, z_n(\rho)]'}{\rho}\hat{\mathbf{e}}_\theta + \cos\phi\,\pi_n(\cos\theta)\frac{[\rho\, z_n(\rho)]'}{\rho}\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.83}$$

$$\mathbf{N}_{eln} = \cos\phi\, n(n+1)\sin\theta\,\pi_n(\cos\theta)\frac{z_n(\rho)}{\rho}\hat{\mathbf{e}}_r + \cos\phi\,\tau_n(\cos\theta)\frac{[\rho\, z_n(\rho)]'}{\rho}\hat{\mathbf{e}}_\theta - \sin\phi\,\pi_n(\cos\theta)\frac{[\rho\, z_n(\rho)]'}{\rho}\hat{\mathbf{e}}_\phi \qquad \textbf{2.4.84}$$

where the prime indicates differentiation, a process that is aided by the identity

$$\frac{d}{d\rho}z_n = \frac{n\,z_{n-1} - (n+1)z_{n+1}}{2n+1} \qquad \textbf{2.4.85}$$

Now the boundary conditions at the boundary between the spherical particle and the medium are given as [1, 2],

$$(E_i + E_s - E_1)\times\hat{e}_r = 0 \qquad \textbf{2.4.86}$$

$$(H_i + H_s - H_1)\times\hat{e}_r = 0 \qquad \textbf{2.4.87}$$

and may be written in component form as

$$E_{i\theta} + E_{s\theta} = E_{1\theta} \qquad \textbf{2.4.88}$$

$$E_{i\phi} + E_{s\phi} = E_{1\phi} \qquad \textbf{2.4.89}$$

$$H_{i\theta} + H_{s\theta} = H_{1\theta} \qquad \textbf{2.4.90}$$

$$H_{i\phi} + H_{s\phi} = H_{1\phi} \qquad \textbf{2.4.91}$$

Hence from the orthogonality of $\sin\phi$ and $\cos\phi$, the orthogonality relations 2.4.80, the boundary conditions 2.4.88 - 2.4.91, the expansions 2.4.62, 2.4.64, 2.4.66, 2.4.67, 2.4.71, 2.4.72 and the expressions 2.4.81 - 2.4.84 for the vector

spherical harmonics, four linear equations are eventually obtained in the expansion coefficients as

$$j_n(mx)c_n + h_n^{(1)}(x)p_n = j_n(x) \qquad \textbf{2.4.92}$$

$$\mu[mxj_n(mx)]'c_n + \mu_1[xh_n^{(1)}(x)]'p_n = \mu_1[xj_n(x)]' \qquad \textbf{2.4.93}$$

$$\mu m j_n(mx)d_n + \mu_1 h_n^{(1)}(x)q_n = \mu_1 j_n(x) \qquad \textbf{2.4.94}$$

$$[mx j_n(mx)]'d_n + m[xh_n^{(1)}(x)]'q_n = m[x j_n(x)]' \qquad \textbf{2.4.95}$$

where the prime indicates differentiation with respect to the argument in parenthesis and the size parameter x and the relative refractive index m are

$$x = kr = 2\pi\frac{N}{\lambda}a \qquad \textbf{2.4.96}$$

$$m = \frac{k_1}{k} = \frac{N_1}{N} \qquad \textbf{2.4.97}$$

and N_1 and N are the refractive indices of particle and medium, respectively. However in most practical cases (such as cosmic dust particles moving in interstellar space, aerosols distributed in planetary atmosphere etc.) $|m-1| << 1$, and therefore the size parameter, $x \approx \frac{2\pi}{\lambda}a$.

Solving the four simultaneous equations 2.4.92 - 2.4.95, the scattering coefficients are obtained as

$$q_n = \frac{\mu m^2 j_n(mx)[xj_n(x)]' - \mu_1 j_n(x)[mxj_n(mx)]'}{\mu m^2 j_n(mx)[xh_n^{(1)}(x)]' - \mu_1 h_n^{(1)}(x)[mxj_n(mx)]'} \qquad \textbf{2.4.98}$$

$$p_n = \frac{\mu_1 j_n(mx)[xj_n(x)]' - \mu j_n(x)[mxj_n(mx)]'}{\mu_1 j_n(mx)[xh_n^{(1)}(x)]' - \mu h_n^{(1)}(x)[mxj_n(mx)]'} \qquad \textbf{2.4.99}$$

These coefficients are further simplified by introducing the Riccati-Bessel functions $\psi_n(\rho)$ and $\xi_n(\rho)$ related to the first kinds of spherical Bessel and Hankel functions [2, 3, 324].

$$\psi_n(\rho) = \rho j_n(\rho) = \sqrt{\frac{\pi\rho}{2}} J_{n+\frac{1}{2}}(\rho) \qquad \textbf{2.4.100}$$

$$\xi_n(\rho) = \rho h_n^{(1)}(\rho) = \sqrt{\frac{\pi\rho}{2}} H^{(1)}_{n+\frac{1}{2}}(\rho) = \psi_n(\rho) - i\chi_n(\rho) \qquad \textbf{2.4.101}$$

If the permeability μ_1 and μ of the particle and the surrounding medium, respectively, is the same, then 2.4.98 and 2.4.99 become

$$q_n = \frac{m\psi_n(mx)\psi_n'(x) - \psi_n(x)\psi_n'(mx)}{m\psi_n(mx)\xi_n'(x) - \xi_n(x)\psi_n'(mx)} \qquad \textbf{2.4.102}$$

$$p_n = \frac{\psi_n(mx)\psi_n'(x) - m\psi_n(x)\psi_n'(mx)}{\psi_n(mx)\xi_n'(x) - m\xi_n(x)\psi_n'(mx)} \qquad \textbf{2.4.103}$$

Notably p_n and q_n vanish as m approaches unity which indicates that a particle ceases to scatter light when it disappears.

Since the series expansion 2.4.71 and 2.4.72 of the scattered field is uniformly convergent, the series can be terminated after n_c terms and the resulting error is arbitrarily small for all kr if n_c is sufficiently large. Now if $kr >> n_c^2$, substituting the asymptotic relation 2.4.68 and its derivative 2.4.70, in the truncated series of 2.4.71 and 2.4.72, yields the transverse components of the scattered electric field as

$$E_{s\theta} \sim E_0 \frac{e^{ikr}}{-ikr} \cos\phi \; S_2(\cos\theta) \qquad \textbf{2.4.104}$$

and $$E_{s\phi} \sim -E_0 \frac{e^{ikr}}{-ikr} \sin\phi \; S_1(\cos\theta) \qquad \textbf{2.4.105}$$

where,

$$S_1 = \sum_n \frac{2n+1}{n(n+1)} (q_n\pi_n + p_n\tau_n) \qquad \textbf{2.4.106}$$

and $$S_2 = \sum_n \frac{2n+1}{n(n+1)}(q_n\tau_n + p_n\pi_n) \qquad \textbf{2.4.107}$$

and the series are terminated after n_c terms. The relation between incident and scattered field amplitudes therefore becomes

$$\begin{pmatrix} E_{\parallel s} \\ E_{\perp s} \end{pmatrix} = \frac{e^{ik(r-z)}}{-ikr}\begin{pmatrix} S_2 & 0 \\ 0 & S_1 \end{pmatrix}\begin{pmatrix} E_{\parallel i} \\ E_{\perp i} \end{pmatrix} \qquad \textbf{2.4.108}$$

From this relation 2.4.108 and using the 4x4 Mueller matrix for scattering from 2.2.77, the relation between incident and scattered Stokes parameters follows as

$$\begin{pmatrix} I_s \\ Q_s \\ U_s \\ V_s \end{pmatrix} = \frac{1}{k^2r^2}\begin{pmatrix} S_{11} & S_{12} & 0 & 0 \\ S_{12} & S_{11} & 0 & 0 \\ 0 & 0 & S_{33} & S_{34} \\ 0 & 0 & -S_{34} & S_{33} \end{pmatrix}\begin{pmatrix} I_i \\ Q_i \\ U_i \\ V_i \end{pmatrix} \qquad \textbf{2.4.109}$$

where,

$$S_{11} = \tfrac{1}{2}(|S_2|^2 + |S_1|^2) \qquad \textbf{2.4.110}$$

$$S_{12} = \tfrac{1}{2}(|S_2|^2 - |S_1|^2) \qquad \textbf{2.4.111}$$

$$S_{33} = \tfrac{1}{2}(S_2^*S_1 + S_2S_1^*) \qquad \textbf{2.4.112}$$

$$S_{34} = \frac{i}{2}(S_1S_2^* - S_2S_1^*) \qquad \textbf{2.4.113}$$

If the incident light is unpolarized, i.e.,

$$Q = U = V = 0 \qquad \textbf{2.4.114}$$

then due to relations 2.2.73 - 2.2.76 the Stokes parameters of the scattered light in 2.4.109 become

$$U_s = V_s = 0 \qquad \textbf{2.4.115}$$

$$Q_s = \left(\frac{1}{k^2r^2}\right)S_{12}I_i \qquad \textbf{2.4.116}$$

$$I_s = \left(\frac{1}{k^2 r^2}\right) S_{11} I_i \qquad \textbf{2.4.117}$$

Thus $S_{11}(\theta)$ characterizes the scattered light intensity as a function of scattering angle for unpolarized light and can also be related to the scattered intensities $I_{\parallel}$ and $I_{\perp}$ polarized parallel and perpendicular to the scattering plane with the following equation [2]:

$$S_{11} = \frac{I_{\parallel} + I_{\perp}}{2} \qquad \textbf{2.4.118}$$

Also following equation 2.2.65, 2.2.128 and 2.4.116 the degree of linear polarization or linear polarization ratio of scattered light for a single particle will now be given by,

$$P = -\frac{Q_s}{I_s} = -\frac{S_{12}}{S_{11}} = \frac{I_{\perp} - I_{\parallel}}{I_{\perp} + I_{\parallel}} \qquad \textbf{2.4.119}$$

The scattering, extinction and backscattering efficiency (Q_{sca}, Q_{ext} and Q_{back}) as well as radiation pressure (Q_{pr}) can be calculated directly from the amplitude coefficients [2, 121, 122],

$$\begin{aligned} Q_{ext} &= \frac{2}{x^2}\sum_{n=1}^{\infty}(2n+1)\mathrm{Re}\{q_n + p_n\} \\ &= \frac{4}{x^2}\mathrm{Re}\{S_1(0°)\} \end{aligned} \qquad \textbf{2.4.120}$$

because [2, 118],

$$S_2(0°) = S_1(0°) = \frac{1}{2}\sum_{n=1}^{\infty}(2n+1)(q_n + p_n) \qquad \textbf{2.4.121}$$

$$Q_{sca} = \frac{2}{x^2}\sum_{n=1}^{\infty}(2n+1)\left(|q_n|^2 + |p_n|^2\right) \qquad \textbf{2.4.122}$$

$$Q_{back} = \frac{2}{x^2}\left|\sum_{n=1}^{\infty}(2n+1)(-1)^n(q_n - p_n)\right|^2 = \frac{4}{x^2}|S_1(180°)|^2 \qquad \textbf{2.4.123}$$

because [2],

$$S_2(180°) = -S_1(180°) = \frac{1}{2}\sum_{n=1}^{\infty}(2n+1)(-1)^n(q_n - p_n) \qquad \textbf{2.4.124}$$

$$Q_{pr} = Q_{ext} - gQ_{sca} = Q_{ext} - \frac{4}{x^2}\left\{\sum_{n=1}^{\infty}\left[\frac{n(n+1)}{n+2}\mathrm{Re}(q_n q_{n+1} + p_n p_{n+1}) + \frac{2n+1}{n(n+1)}\mathrm{Re}(q_n p_n)\right]\right\} \qquad \textbf{2.4.125}$$

From these efficiency factors we can also derive absorption efficiency factor (Q_{abs}), single scattering albedo(Λ) and asymmetry parameter (g) as follows,

$$Q_{abs} = Q_{ext} - Q_{sca} \qquad \textbf{2.4.126}$$

$$\Lambda = \frac{Q_{sca}}{Q_{ext}} \qquad \textbf{2.4.127}$$

$$g = \frac{Q_{ext} - Q_{pr}}{Q_{sca}} \qquad \textbf{2.4.128}$$

From the efficiency factors one can find the corresponding cross sections by using the simple relation, $C = GQ$ where $G = \pi r^2$ is the geometrical cross section of the particle.

2.4.3. Mie Scattering by polydisperse systems of particles

Till now electromagnetic scattering by a "single particle" has been discussed. However in nature the mixture of particle sizes encountered in a unit volume of natural aerosols or hydrosols can usually be modeled quite well with

analytical size distributions such as gamma, normal, lognormal etc. [1, 5, 10, 12]. In the single scattering regime, the Stokes parameters of the light scattered by an ensemble of randomly separated particles are the sum of the Stokes parameters of the light scattered by the individual particles. Therefore the scattering matrix for such a collection is also additive, i.e., merely the sum of the individual particle scattering matrices, provided linear dimension of the volume occupied by the scatterers is small compared with the distance at which the scattered light is observed.

In most cases, the experimentally measured distribution of water cloud and fog particles can be well represented by the gamma distribution [1, 215]. The gamma distribution is given by,

$$n(r) = ar^{\alpha} \exp(-br) \qquad \textbf{2.4.129}$$

where,

$$a = N\frac{b^{\alpha+1}}{\Gamma(\alpha+1)} \qquad \textbf{2.4.130}$$

$$b = \frac{\alpha}{r_m} \qquad \textbf{2.4.131}$$

where r is the radius of the particle, r_m is the modal radius, a is the normalization constant, Γ is the gamma function and the parameter α characterizes the width of the distribution being smaller for wider distributions. Hence, the gamma distribution (equation 2.4.129) is completely specified by the index α and the modal radius r_m.

Again normal and lognormal size distribution is often used to describe the size distribution of hydrosols (biological cells) and aerosols [4, 10, 12, 14, 64]. The normal size distribution is given by [82],

$$n(r) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma_g}\exp\left(-\frac{(r-r_g)^2}{2\sigma_g{}^2}\right) \qquad \textbf{2.4.132}$$

and lognormal size distribution is given by,

$$n(r)=\frac{1}{(2\pi)^{\frac{1}{2}} r \ln(\sigma_g)}\exp\left(-\frac{(\ln r-\ln r_g)^2}{2\ln^2(\sigma_g)}\right) \qquad \textbf{2.4.133}$$

where r_g is the modal radius, σ_g if the standard deviation and measures the width of the distribution.

It is convenient to represent the mean particle size and width of the size distribution functions by two parameters, i.e., effective radius, r_{eff} and effective variance, v_{eff} defined as [5, 10, 238],

$$r_{eff}=\frac{\int_{r_{min}}^{r_{max}} r\pi r^2 n(r)dr}{\int_{r_{min}}^{r_{max}} \pi r^2 n(r)dr}=\frac{1}{G}\int_{r_{min}}^{r_{max}} r\pi r^2 n(r)dr \qquad \textbf{2.4.134}$$

$$v_{eff}=\frac{1}{Gr_{eff}^2}\int_{r_{min}}^{r_{max}}(r-r_{eff})^2\pi r^2 n(r)dr \qquad \textbf{2.4.135}$$

In the theoretical computations of single scattering by a small volume element consisting of such particles, it is necessary to average out the scattering matrices and efficiencies over the representative particle ensemble. The ensemble averaged values of the scattering matrix elements and the scattering cross sections can be derived from their weighted contributions [5, 10],

$$\langle S_{ij}(\theta)\rangle=\int_{r_{min}}^{r_{max}} S_{ij}(\theta,r)n(r)dr \qquad \textbf{2.4.136}$$

$$\langle S_{ij}(\theta)\rangle = \int_{r_{max}}^{r_{min}} S_{ij}(\theta,r)n(r)dr$$

$$\langle C_{sca}\rangle = \int_{r_{max}}^{r_{min}} C_{sca}(r)n(r)dr$$

$$\langle C_{ext}\rangle = \int_{r_{max}}^{r_{min}} C_{ext}(r)n(r)dr \qquad \textbf{2.4.136A}$$

$$\langle C_{abs}\rangle = \int_{r_{max}}^{r_{min}} C_{abs}(r)n(r)dr$$

where i, j = 1 to 4. Here $n(r)dr$ is the fraction of particles having radii between r and $r+dr$ and r_{min} and r_{max} are the minimum and maximum particle radius. The distribution function $n(r)$ is normalized to unity as follows:

$$\int_{r_{min}}^{r_{max}} n(r)dr = 1 \qquad \textbf{2.4.137}$$

The use of size distribution and the averaging process (equation 2.4.136) then leads to the formation of the volume scattering function $\beta(\theta)$ which characterizes the angular pattern of light scattered by a unit volume of scattering particles (e.g. aerosols, hydrosols etc.). In general, the volume scattering function $\beta(\theta)$ depends explicitly on the nature of the incident wave (wavelength, λ and polarization state) and the scattering angle, θ. Implicitly it depends on the composition of the particle ensemble in terms of size, shape and refractive index. For an ensemble of randomly oriented axially symmetric particles illuminated by unpolarized light the volume scattering function $\beta(\theta)$ is given by [1, 29, 64, 190, 329],

$$\beta(\theta) = \frac{1}{k^2}\int_{r_{min}}^{r_{max}} S_{11}(\theta,r)n(r)dr \qquad \textbf{2.4.138}$$

In terms of size parameter equation 2.4.138 can be written as,

$$\beta(\theta) = \frac{1}{k^3}\int_{r_{min}}^{r_{max}} S_{11}(\theta,x)n(x)dx \qquad \textbf{2.4.139}$$

where $\beta(\theta)$ is in units of per steradian per centimeter (sr $^{-1}$cm^{-1}). Notably, finding this function $\beta(\theta)$ is the most crucial part of computing light field in a scattering media [64].

For an ensemble of polydisperse particles the degree of linear polarization, P originally defined by equation 2.2.65 becomes [332],

$$P(\theta) = -\frac{\langle S_{12}(\theta)\rangle}{\langle S_{11}(\theta)\rangle} = \frac{I_{\perp} - I_{\parallel}}{I_{\perp} + I_{\parallel}} \quad \text{such that } |P(\theta)| \le 1 \qquad \textbf{2.4.140}$$

It is evident from equations 2.4.118, 2.4.119, 2.4.138 and 2.4.140 that experimental measurements of I_s and I_t using unpolarized light will yield experimental $\beta(\theta)$ and $P(\theta)$ values, a fact that is very much used in the light scattering setup.

2.4.4. Computational approach: Numerical algorithm and convergence procedures

The basic equations of Mie theory are of infinite sums, but only a finite number of terms can be calculated. Therefore, in practical computer calculations the infinite series of equations 2.4.106 and 2.4.107 has to be truncated after a suitable finite number of terms n_{max}, which depends on the size parameter x, in order to find the correct solution with the required accuracy. However, a too high value of n_{max} may lead to numerical instability [174, 317] in Mie coefficient calculations and also high CPU time and memory usage. The criterion for choosing n_{max} has been discussed by many authors [1, 2, 121 - 123, 263, 318 - 323]. Wang et. al. [317] reported a method for determining n_{max} during calculation when $a_{n_{max}}$ and $b_{n_{max}}$ become smaller than a desired accuracy. However it is advantageous to determine the n_{max} value prior to Mie calculations as it simplifies the computational effort, reduces computational time and is advantageous for vectorized calculations [123, 263, 322]. Many workers usually follow the method proposed by Wiscombe's [123, 263] for calculating n_{max} as

given by the empirical equation 2.4.141 [1, 2, 122]. Some other efficient algorithms use the ratio between the Ricatti-Bessel functions, $-\frac{\chi_n(x)}{\psi_n(x)}$ to determine the value of n_{max} [318, 321 - 323]. In this work, we used the following two methods for determining the value of n_{max}.

Method 1: This method utilizes the criterion proposed by Wiscombe [123] for choosing the value of n_{max}. It determines n_{max} accurately upto 5 - 6 significant digits [123]. According to the criterion the value of n_{max} is given by,

$$n_{max} = \text{integer value of} \left\{ \begin{array}{ll} x + 4x^{\frac{1}{3}} + 1, & \text{when } 0.02 \le x < 8 \\ x + 4.05x^{\frac{1}{3}} + 2, & \text{when } 8 \le x < 4200 \\ x + 4x^{\frac{1}{3}} + 2, & \text{when } 4200 \le x < 20000 \end{array} \right\} \qquad \textbf{2.4.141}$$

Method 2: This method described by Cai et. al. [322] involves the calculation of G_n, the ratio of Riccati-Bessel functions ψ_n and χ_n, as given below,

$$G_n = -\frac{\chi_n}{\psi_n} = \frac{Y_{n+1/2}(x)}{J_{n+1/2}(x)} \qquad \textbf{2.4.142}$$

where J and Y represent Bessel functions of first and second kind respectively. $G_n(x)$ can be calculated by using the following upward recursion relation,

$$\frac{1}{G_n(x)} = \frac{\frac{(2n-1)}{x} - r_{n-1}(x)}{\frac{(2n-1)}{x} G_{n-1}(x) - r_{n-1}(x) G_{n-2}(x)} \qquad \textbf{2.4.143}$$

where,

$$r_n(x) = \frac{J_{n-1/2}(x)}{J_{n+1/2}(x)} \qquad \textbf{2.4.144}$$

For $n < x$, $G_n(x)$is oscillatory and bounded (the oscillatory regime) and increases rapidly when $n > x$ (the exponential regime) [321, 322]. Therefore to

achieve convergence of a_n and b_n, the value of $G_n(x)$ is truncated in the exponential regime for that value of n for which $1/G_n(x)$ becomes smaller than the tolerable accuracy ε. This value at a certain value of n is then set as $n_{\max}$. From equation 2.4.143, before calculating $G_n(x)$, it is required to calculate $r_n(x)$. In this work, the method of continued fraction described by Lentz [319] was used to calculate $r_n(x)$. After the calculation of $r_n(x)$, using the relation given by equation 2.4.143, $G_n(x)$can be determined very easily. Details of the derivation of equations 2.4.142, 2.4.143 and $r_n(x)$ is described in Appendix B. Thus the calculation of $n_{\max}$ using this method can be divided into five steps:

1. setting the cutoff value of $G_n(x) = \frac{1}{\varepsilon}$ as an input,
2. calculation of $r_n(x)$ using the continued fraction method described by Lentz,
3. calculation of $1/G_n(x)$ using the upward recursion relation given by equation 2.4.143,
4. determination of n for which $1/G_n(x)$ becomes smaller than ε,
5. setting the value of n to be $n_{\max}$.

Like Wiscombe's criterion, this method also calculates $n_{\max}$ *a priori* which is helpful for vectorized computations [123]. However, this method enables the calculation of $n_{\max}$ at a desired accuracy of calculations.

Now in computation of scattering coefficients the logarithmic derivative,

$$D_n(\rho) = \frac{d}{d\rho}\log_e \psi_n(\rho) \qquad \textbf{2.4.145}$$

which satisfies the recurrence relation,

$$D_{n-1} = \frac{n}{p} - \frac{1}{D_n + \frac{n}{p}} \qquad \textbf{2.4.146}$$

is introduced [2], as this recasts the equations 2.4.102 and 2.4.103 in a convenient form given as,

$$q_n = \frac{\left[\frac{D_n(mx)}{m} + \frac{n}{x}\right]\psi_n(x) - \psi_{n-1}(x)}{\left[\frac{D_n(mx)}{m} + \frac{n}{x}\right]\xi_n(x) - \xi_{n-1}(x)} \qquad \textbf{2.4.147}$$

$$p_n = \frac{\left[mD_n(mx) + \frac{n}{x}\right]\psi_n(x) - \psi_{n-1}(x)}{\left[mD_n(mx) + \frac{n}{x}\right]\xi_n(x) - \xi_{n-1}(x)} \qquad \textbf{2.4.148}$$

where the recurrence relations

$$\psi_n'(x) = \psi_{n-1}(x) - \frac{n\psi_n(x)}{x} \qquad \textbf{2.4.149}$$

$$\xi_n'(x) = \xi_{n-1}(x) - \frac{n\xi_n(x)}{x} \qquad \textbf{2.4.150}$$

were used to eliminate ψ_n' and ξ_n'. $D_n(mx)$ is computed by downward recurrence, that is lower orders are calculated from higher orders beginning with $D_{n^*_{max}} = 0.0 + 0.0i$ [1, 2]. Here n^*_{max} is greater than both n_{max} and mx and a good choice can be obtained by adding 15 with the greater of the two numbers n_{max} and $|mx|$, i. e., $n^*_{max} = \max(n_{max}, |mx|) + 15$ [1, 2, 121].

Again $\psi_n(x)$ and $\xi_n(x)$, which are connected by the relation $\xi_n = \psi_n - i\chi_n$ and satisfies the recurrence relation,

$$\psi_{n+1}(x) = \frac{2n+1}{x}\psi_n(x) - \psi_{n-1}(x) \qquad \textbf{2.4.151}$$

are calculated by upward recurrence, that is higher orders are calculated from lower orders similarly as was done in BHMIE subroutine using the initial values given by [2],

$$\psi_{-1}(x) = \cos(x) \qquad \textbf{2.4.152}$$

$$\psi_0(x) = \sin(x) \qquad \textbf{2.4.153}$$

$$\chi_{-1}(x) = -\sin(x) \qquad \textbf{2.4.154}$$

$$\chi_0(x) = \cos(x) \qquad \textbf{2.4.155}$$

As already mentioned the angular functions π_n and τ_n depend on $\cos(\theta)$ and are computed by using the upward recursion relations.

For a particular scattering angle θ, providing initial values for r, λ, N_p, N_m (particle radius, incident wavelength, refractive index of particle and refractive index of medium respectively) and then using relations 2.4.73, 2.4.74, 2.4.102 and 2.4.103, the values of π_n, τ_n, a_n and b_n may be derived and put in equations 2.4.106 and 2.4.107 to obtain the values of S_1 and S_2 which when put in equations 2.4.110, 2.4.111, 2.4.112, 2.4.113 gives the scattering matrix elements for the particular θ. Again, using equations 2.4.120, 2.4.122, 2.4.123, 2.4.125 extinction, scattering and radar backscattering efficiency factors and radiation pressure can be calculated respectively and from these calculated values and using equations 2.4.126, 2.4.127 and 2.4.128, absorption efficiency factor, single scattering albedo and asymmetry parameter can also be calculated.

2.4.5. Description of the computer program

The program "TUMiescat.c" is written in standard C programming language. The program is capable of computing scattering matrix elements and efficiencies for a single homogenous spherical particle for a wide range of size parameters as well as for an ensemble of homogenous spherical particles having different size distributions. The program consists of two files, i.e., the main program (TUMiescat.c) and one associated file (cpxarith.c) for calculating complex algebra. TUMiescat.c is composed of five portions- the main program, the improved version of BHMIE subroutine [1] in standard C called 'bhmie', the 'dstn' function which calculates the number of particles for a particular radius of the required size distribution, 'mxlentz' function which calculates the value of $n_{\max}$ and the 'gnn' function which calculates $G_n(x)$ using 'method 2' described in the previous section. In the program, the usual notations used by different workers were adopted with an attempt to make the program easy to understand, simple and easy to modify. The program was tested on a computer running

Fedora Core Linux 9.0 with gcc (version 4.3.0) installed. The computed values of the scattering matrix elements are written in a user defined file whereas the efficiency factors are written in a predefined file named 'crossec.dat'. The processing time of the program was less than a second for calculations on monodisperse particles, 2 - 3 seconds for calculations on polydisperse particles and 20 - 30 seconds for the calculation of scattering properties as functions of size parameter and scattering angle. It is worth mentioning that consistency of units was maintained while developing the program. The program was found to be stable even for very large size parameters. The computer program TUMiescat.c and the associated file cpxarith.c are given in Appendix C.

2.4.5.1. Input parameters

The input parameters are particle radius (rad), real part of particle refractive index (refre), imaginary part of particle refractive index (refim), real part of medium refractive index (refmed.x) and incident wavelength (wavel) or simply size parameter (xk) for calculations for a single spherical particle. The user has to give the value of $\varepsilon = 1/G_n(x)$ as an input (epson), for the calculation of of n_{max} using 'method 2'. For light scattering calculations for an ensemble of spherical particles having different size distribution, the characteristic distribibition function parameters like lowest grain size (radl), highest grain size (radh), modal radius (rc) and α (alpha) for gamma distribution, modal radius r_g (rg) and standard deviation (sigma) for normal and lognormal distribution should be given as input along with the other parameters. Again for calculating light scattering properties as a function of incident wavelength one has to provide the lowest value of incident wavelength (wavell) and the highest value of incident wavelength (wavelh) along with the increament step (stepk). Similarly for the calculation of the light scattering properties as a function of size parameter one has to provide the lowest grain size (radl), highest grain size (radh), incident wavelength (wavel) or lowest size parameter (xkl) and highest

size parameter (xkh). Again, It was also observed that the required optimal size of all the input variables was double precision.

2.4.5.2. Output parameters

The output of the program is written in a user defined data file. The output parameters are scattering angle (ang) and non zero elements of the scattering matrix ($S_{11}, -S_{12}/S_{11}, S_{33}/S_{11}$ and S_{34}/S_{11}). The size paramater (x), scattering efficiency (QSCA), extinction efficiency (QEXT), backscattering efficiency (QBACK), absorption efficiency (QABS), radiation pressure (QPR), single scattering albedo (albedo) and asymmetry parameter (g) are written in a predefined datafile named 'crossec.dat'.

2.5. T-matrix method

In this thesis work, T-matrix method was choosen to analyse the experimental results of carbon black, charcoal dust, diatoms - a group of single celled micro algae etc., measured by using the light scattering setup, as it is a powerful technique for calculating electromagnetic scattering by nonspherical particles and based on numerically solving Maxwell's equations [238]. T-matrix reduces exactly to the Lorentz-Mie theory (Mie theory) when the scatterer is a homogenous or layered sphere composed of isotropic materials [126].

2.5.1. General formulation

Like in Mie theory, in T-matrix method also the incident, scattered, and internal fields are expanded into spherical vector harmonics. Mathematically, these fields are given as series (equations 2.4.55 - 2.4.57) representing a complete set of linear combinations of vector harmonics (equations 2.4.48 - 2.4.51).

As the separated solution 2.4.25, is a differential equation of the second order, there must exist two independent solutions (equations 2.4.28 and 2.4.29 are rewritten below for convenience),

$$\Phi_e = \cos(m\phi) \qquad \textbf{2.5.1}$$

$$\Phi_o = \sin(m\phi) \qquad \textbf{2.5.2}$$

where subscripts e and o denote even and odd, respectively. Here m can be an integer or zero and it can be shown that only positive values of m are sufficient to generate all the linearly independent solutions. A linear combination of both 2.5.1 and 2.5.2 gives the complete solution,

$$\Phi = A_m \cos(m\varphi) + B_m \sin(m\varphi) \qquad \textbf{2.5.3}$$

To simplify any further derivations, equation 2.5.3 can be replaced by the linear combination of the complex functions $e^{im\phi}$ and $e^{-im\phi}$ [126], i.e.,

$$\Phi = c_m e^{im\phi} + c_{-m} e^{-im\phi} \qquad \textbf{2.5.4}$$

where c_m and c_{-m} are in general complex numbers. Thus the even and odd expansions are reduced to only one coefficient. Now equations 2.4.55 - 2.4.57 reduces to,

$$\mathbf{E}_i = \sum_{n=1}^{\infty} \sum_{m=-n}^{n} \left(a_{mn} \mathbf{M}_{mn} + b_{mn} \mathbf{N}_{mn} \right) \qquad \textbf{2.5.5}$$

$$\mathbf{E}_1 = \sum_{n=1}^{\infty} \sum_{m=-n}^{n} \left(c_{mn} \mathbf{M}_{mn} + d_{mn} \mathbf{N}_{mn} \right) \qquad \textbf{2.5.6}$$

$$\mathbf{E}_s = \sum_{n=1}^{\infty} \sum_{m=-n}^{n} \left(p_{mn} \mathbf{M}_{mn} + q_{mn} \mathbf{N}_{mn} \right) \qquad \textbf{2.5.7}$$

Also equations 2.4.46 and 2.4.47 can be combined to one equation,

$$\psi_{mn}(kr, \theta, \varphi) = z_n(kr) P_n^m(\cos\theta) e^{im\varphi} \qquad \textbf{2.5.8}$$

with m having positive and negative values. Inserting formula 2.5.8 into 2.4.8 we can write vector harmonics $\mathbf{M}$, $\mathbf{N}$ in the form [4 - 8, 126, 168, 176, 239],

$$\mathbf{M}_{mn}(kr, \theta, \varphi) = \gamma_{mn} h_n^{(1)}(kr) \mathbf{C}_{mn}(\theta, \varphi) \qquad \textbf{2.5.9}$$

$$\mathbf{N}_{mn}(kr,\theta,\varphi)=\gamma_{mn}\left\{\frac{\mathbf{P}_{mn}(\theta,\varphi)}{kr}h_n^{(1)}(kr)n(n+1)+\frac{\mathbf{B}_{mn}(\theta,\varphi)}{kr}\frac{d\left[krh_n^{(1)}(kr)\right]}{d(kr)}\right\} \qquad \textbf{2.5.10}$$

where $\gamma_{mn}=d_n\sqrt{\frac{(n-m)!}{(n+m)!}}$ **2.5.11**

is a normalization factor originating from

$$P_n^{-m}(\cos\theta)=(-1)^m\frac{(n-m)!}{(n+m)!}P_n^m(\cos\theta) \qquad \textbf{2.5.12}$$

and $d_n=\sqrt{\frac{(2n+1)}{4\pi n(n+1)}}$ **2.5.13**

$$\mathbf{C}_{mn}(\theta,\varphi)=\left[\hat{\theta}\frac{im}{\sin\theta}P_n^m(\cos\theta)-\hat{\varphi}\frac{d}{d\theta}P_n^m(\cos\theta)\right]e^{im\varphi} \qquad \textbf{2.5.14}$$

$$\mathbf{B}_{mn}(\theta,\varphi)=\left[\hat{\theta}\frac{d}{d\theta}P_n^m(\cos\theta)+\hat{\varphi}\frac{im}{\sin\theta}P_n^m(\cos\theta)\right]e^{im\varphi} \qquad \textbf{2.5.15}$$

$$\mathbf{P}_{mn}(\theta,\varphi)=\hat{\mathbf{r}}P_n^m(\cos\theta)e^{im\varphi} \qquad \textbf{2.5.16}$$

here we used the expression of the Wigner d functions in terms of generalized spherical functions as follows [240],

$$d_{lm}^n(\theta)=i^{m-l}P_{lm}^n(\cos\theta) \qquad \textbf{2.5.17}$$

Therefore in analogy with equations 2.4.59 and 2.4.71, the incident and scattered waves are given by,

$$\mathrm{E}_i(r)=\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\left[a_{mn}\mathrm{RgM}_{mn}(kr)+b_{mn}\mathrm{RgN}_{mn}(kr)\right] \qquad \textbf{2.5.18}$$

and $\mathrm{E}_s(r)=\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\left[p_{mn}\mathrm{M}_{mn}(kr)+q_{mn}N_{mn}(kr)\right], r>r_>$ **2.5.19**

The expressions for the functions RgM_{mn} and RgN_{mn} can obtained from equation 2.5.9 and 2.5.10 by replacing spherical Hankel functions $h_n^{(1)}$ by spherical Bessel functions j_n. It should be noted that the functions RgM_{mn} and RgN_{mn} in

equation 2.5.18 are regular at the origin, while the use of the out going functions M_{mn} and N_{mn} in equation2.5.19 ensures that the scattered field satisfies the radiation condition at infinity (i.e., the transverse component of the scattered electric field decays as $1/r$ whereas the radial component decays faster than $1/r$ with $r \to \infty$). The requirement $r > r_>$ in equation 2.5.19 means that the scattered field is considered only outside the smallest circumscribing sphere of the scatterer [4].

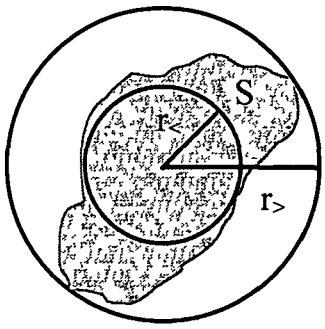

Figure 2.7. Cross sectional view of a scattering object bounded by a closed surface S. $r_>$ and $r_<$ are the radii of circumscribing sphere and the concentric inscribing sphere respectively.

It can be shown that, the expansion coefficients of the plane incident wave are given by the following simple analytical formulas [4]

$$a_{mn} = 4\pi(-1)^m i^n d_n C^*_{mn}(v^{inc}) E_{i0} \exp(-im\phi^{inc}) \qquad \textbf{2.5.20}$$

$$b_{mn} = 4\pi(-1)^m i^{n-1} d_n B^*_{mn}(v^{inc}) E_{i0} \exp(-im\phi^{inc}) \qquad \textbf{2.5.21}$$

where an asterisk indicates complex conjugation.

Since Maxwell's equations and boundary conditions are linear, there must be a linear relationship between the scattered field coefficients p_{mn} and q_{mn} and the incident field coefficients a_{mn} and b_{mn}. In general it can be written as:

$$p_{mn} = \sum_{n'=1}^{\infty} \sum_{m'=-n'}^{n'} \left[T^{11}_{mnm'n'} a_{m'n'} + T^{12}_{mnm'n'} b_{m'n'} \right] \qquad \textbf{2.5.22}$$

$$q_{mn} = \sum_{n'=1}^{\infty} \sum_{m'=-n'}^{n'} \left[T^{21}_{mnm'n'} a_{m'n'} + T^{22}_{mnm'n'} b_{m'n'} \right] \qquad \textbf{2.5.23}$$

or in matrix notation, it reads

$$\begin{bmatrix} p \\ q \end{bmatrix} = T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} T^{11} & T^{12} \\ T^{21} & T^{22} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \qquad \textbf{2.5.24}$$

For asymptotic behavior of a scattered wave in the far field zone we have [5, 126],

$$z_n(kr) = h_n^{(1)}(kr) \underset{r\to\infty}{\sim} \frac{(-i)^{n+1} e^{ikr}}{kr} \qquad \textbf{2.5.25}$$

and therefore equations 2.5.9 and 2.5.10 are transformed to

$$\mathbf{M}_{mn}(kr,\theta,\varphi) \underset{kr\to\infty}{=} \gamma_{mn} \frac{(-i)^{n+1} e^{ik_{out} r}}{kr} \mathbf{C}(\theta,\varphi) \qquad \textbf{2.5.26}$$

$$\mathbf{N}_{mn}(kr,\theta,\varphi) \underset{kr\to\infty}{=} \gamma_{mn} \frac{(-i)^{n} e^{ikr}}{kr} \mathbf{B}_{mn}(\theta,\varphi) \qquad \textbf{2.5.27}$$

Substituting asymptotic formulae 2.5.26 and 2.5.27 into equation 2.5.18 yields

$$\mathbf{E}_s(\mathbf{r}) = \frac{e^{ikr}}{kr} \sum_{n=1}^{\infty} \sum_{m=-n}^{n} i^{-n} \gamma_{mn} \left[-ip_{mn} \mathbf{C}_{mn}(\theta^{sca},\varphi^{sca}) + q_{mn} \mathbf{B}_{mn}(\theta^{sca},\varphi^{sca}) \right] \qquad \textbf{2.5.28}$$

Finally, using 2.5.20 - 2.5.23, 2.5.26 and 2.5.27 we can arrive at the equation,

$$\begin{aligned} S(n^{sca}, n^{inc}) = \frac{4\pi}{k} \sum_{nmn'm'} & i^{n'-n-1} (-1)^{m+m'} d_n d_{n'} \exp\left[i(m\phi^{sca} - m'\phi^{inc}) \right] \\ & \times \left\{ \left[T^{11}_{mnm'n'} C_{mn}(v^{sca}) + T^{21}_{mnm'n'} i B_{mn}(v^{sca}) \right] C^{*}_{m'n'}(v^{inc}) \right. \\ & \left. + \left[T^{12}_{mnm'n'} C_{mn}(v^{sca}) + T^{22}_{mnm'n'} i B_{mn}(v^{sca}) \right] B^{*}_{m'n'}(v^{inc}) / i \right\} \end{aligned} \qquad \textbf{2.5.29}$$

where $S(n^{sca}, n^{inc})$ is another form of amplitude scattering matrix such that,

$$E_s\left(r\hat{\mathbf{n}}^{sca}\right)=\frac{\exp(ikr)}{r}\mathbf{S}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right)E_{i0}$$
$$\Rightarrow\begin{bmatrix}E_{s\theta}\left(r\hat{\mathbf{n}}^{sca}\right)\\E_{s\phi}\left(r\hat{\mathbf{n}}^{sca}\right)\end{bmatrix}=\frac{\exp(ikr)}{r}\begin{pmatrix}S_{11}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right) & S_{12}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right)\\S_{21}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right) & S_{22}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right)\end{pmatrix}\begin{bmatrix}E_{i0\theta}\\E_{i0\phi}\end{bmatrix} \qquad \textbf{2.5.30}$$

The above equation is equivalent with the equation 2.2.35. The matrix elements S_1, S_2, S_3 and S_4 in equation 2.2.35 are dimensionless quantities, while the elements of scattering matrix S_{11}, S_{12}, S_{21}, and S_{22} in equation 2.5.31 have the dimension of length. The component kS_{11} and kS_{22} in S_1 in equation 2.5.31 logically coincides with the component S_2 and S_1 in equation 2.2.35 [126].

From 2.5.29 the elements of the amplitude scattering matrix can be found as,

$$S_{11}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right)=\frac{1}{k}\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\sum_{n'=1}^{\infty}\sum_{m'=-n'}^{n'}\alpha_{mnm'n'}\begin{bmatrix}T^{11}_{mnm'n'}\,\pi_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)+\\+T^{21}_{mnm'n'}\,\pi_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)+\\+T^{12}_{mnm'n'}\,\tau_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)+\\+T^{22}_{mnm'n'}\,\tau_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)\end{bmatrix}\times\exp\left[i\left(m\varphi^{sca}-m'\varphi^{inc}\right)\right] \qquad \textbf{2.5.31}$$

$$S_{12}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right)=\frac{1}{ik}\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\sum_{n'=1}^{\infty}\sum_{m'=-n'}^{n'}\alpha_{mnm'n'}\begin{bmatrix}T^{11}_{mnm'n'}\tau_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)+\\+T^{21}_{mnm'n'}\tau_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)+\\+T^{12}_{mnm'n'}\pi_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)+\\+T^{22}_{mnm'n'}\,\pi_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)\end{bmatrix}\times\exp\left[i\left(m\varphi^{sca}-m'\varphi^{inc}\right)\right] \qquad \textbf{2.5.32}$$

$$S_{21}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right)=\frac{1}{k}\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\sum_{n'=1}^{\infty}\sum_{m'=-n'}^{n'}\alpha_{mnm'n'}\left[\begin{array}{l}T^{11}_{mnm'n'}\,\pi_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)+\\+T^{21}_{mnm'n'}\,\pi_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)+\\+T^{12}_{mnm'n'}\,\tau_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)+\\+T^{22}_{mnm'n'}\,\tau_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)\end{array}\right]\times\exp\left[i\left(m\varphi^{sca}-m'\varphi^{inc}\right)\right]$$

2.5.33

$$S_{22}\left(\hat{\mathbf{n}}^{sca},\hat{\mathbf{n}}^{inc}\right)=\frac{1}{k}\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\sum_{n'=1}^{\infty}\sum_{m'=-n'}^{n'}\alpha_{mnm'n'}\left[\begin{array}{l}T^{11}_{mnm'n'}\,\tau_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)+\\+T^{21}_{mnm'n'}\,\tau_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)+\\+T^{12}_{mnm'n'}\,\pi_{m'n'}\left(\theta^{inc}\right)\tau_{mn}\left(\theta^{sca}\right)+\\+T^{22}_{mnm'n'}\,\pi_{m'n'}\left(\theta^{inc}\right)\pi_{mn}\left(\theta^{sca}\right)\end{array}\right]\times\exp\left[i\left(m\varphi^{sca}-m'\varphi^{inc}\right)\right]$$

2.5.34

where

$$\alpha_{mnm'n'}=4\pi\,i^{n'-n-1}(-1)^{m+m'}d_n d_{n'} \qquad \textbf{2.5.35}$$

and π_{mn} and τ_{mn} are now

$$\pi_{mn}(\theta)=m\frac{d^n_{0m}(\theta)}{\sin\theta} \qquad \textbf{2.5.36}$$

$$\pi_{-mn}(\theta)=(-1)^{m+1}\pi_{mn}(\theta) \qquad \textbf{2.5.37}$$

$$\tau_{mn}(\theta)=m\frac{d\left[d^n_{0m}(\theta)\right]}{d\theta} \qquad \textbf{2.5.38}$$

$$\tau_{-mn}(\theta)=(-1)^m\tau_{mn}(\theta) \qquad \textbf{2.5.39}$$

The function $d^n_{0m}(\theta)$ has the form [240]

$$d^n_{0m}(\theta)=i^m P^n_{0m}(\cos\theta) \qquad \textbf{2.5.40}$$

or we may use an expansion of $d^n_{0m}(\theta)$ into an infinite series

$$d_{0m}^{n}(\theta) = n!\sqrt{(n+m)!(n-m)!}\sum_{k}(-1)^{k}\left[\frac{\sin(\theta/2)}{\cos(\theta/2)}\right]^{2k+m}\frac{\cos^{2n}(\theta/2)}{k!(n-k)!(n-m-k)!(m+k)!} \qquad \textbf{2.5.41}$$

Knowledge of the amplitude scattering matrix will allow one to compute any scattering characteristic discussed earlier. The extinction and scattering cross-sections are given by [5],

$$C_{ext} = -\frac{1}{k^2|E_{i0}|^2}\mathrm{Re}\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\left[a_{mn}(p_{mn})^{*} + b_{mn}(q_{mn})^{*}\right] \qquad \textbf{2.5.42}$$

$$C_{sca} = -\frac{1}{k^2|E_{i0}|^2}\mathrm{Re}\sum_{n=1}^{\infty}\sum_{m=-n}^{n}\left[|p_{mn}|^2 + |q_{mn}|^2\right] \qquad \textbf{2.5.43}$$

In the formulation of the T-matrix method, from equations 2.5.9 - 2.5.24, it can be observed that neither the vector harmonics nor the expansion coefficients (a_{mn} and b_{mn}) depend on geometrical and physical characteristics of the scattering particle. All kind of physical and geometrical characteristics of the scattering particle (refractive index, shape, size, and orientation with respect to the reference frame) are contained in T-matrix. T-matrix is completely independent of the propagation directions and the polarization states of the incident and scattered fields. This means that the T-matrix need to be computed only once and then can be used in calculation for any directions of incidence and scattering and for any polarization state of the incident field.

2.5.2. Calculation of T-matrix: the extended boundary condition method (EBCM)

This section describes the general scheme for computing the T matrix for simple nonspherical particles based on the extended boundary condition method (EBCM), which was originally developed by Waterman [158 - 160] and improved significantly by Mishchenko [4, 5]. The EBCM itself is based on Huygens's principle which states "the field (electric) is the superposition of the spherical

wavelets originating from a surface located between the observational point and the source".

Let us now consider a particle with a finite volume V_{int} positioned in a infinite, homogenous, isotropic, nonmagnetic and nonabsorbing surrounding medium with a volume V_{ext} such that the electric fields in the regions V_{int} and V_{ext} satisfy the vector wave equation (figure 2.7). Now Green's theorem for a regular surface S' bounding a volume V' is given by,

$$\int_{V'} dV' \{\mathbf{a}\cdot(\nabla\times\nabla\times\mathbf{b})-\mathbf{b}\cdot(\nabla\times\nabla\times\mathbf{a})\} = \int_{S'} dS' \hat{\mathbf{n}}\cdot\{\mathbf{b}\times(\nabla\times\mathbf{a})-\mathbf{a}\times(\nabla\times\mathbf{b})\}$$

2.5.44

where $\hat{n}$ is the unit vector along the local outward normal to the surface.

The above equation 2.5.44 can be applied to the exterior region and then using Green's function expressed in terms of vector spherical wave functions (VSWFs), it can be obtained,

$$p_{mn} = -k\,(-1)^m \int_S dS \begin{Bmatrix} \omega\mu[\hat{n}\times H_+(r)]\cdot RgM_{-mn}(k\,r,\theta,\varphi) \\ -ik[\hat{n}\times E_+(r)]\cdot RgN_{-mn}(k\,r,\theta,\varphi) \end{Bmatrix} \qquad \textbf{2.5.45}$$

$$q_{mn} = -k\,(-1)^m \int_S dS \begin{Bmatrix} \omega\mu[\hat{n}\times H_+(r)]\cdot RgN_{-mn}(kr,\theta r\varphi) \\ -ik[\hat{n}\times E_+(r)]\cdot RgM_{-mn}(kr,\theta r\varphi) \end{Bmatrix} \qquad \textbf{2.5.46}$$

Similarly the incident field coefficients are

$$a_{mn} = k(-1)^m \int_S dS \begin{Bmatrix} \omega\mu[\hat{n}\times H_+(r)]\cdot M_{-mn}(kr,\theta r\varphi) \\ -ik[\hat{n}\times E_+(r)]\cdot N_{-mn}(kr,\theta r\varphi) \end{Bmatrix} \qquad \textbf{2.5.47}$$

$$b_{mn} = k(-1)^m \int_S dS \begin{Bmatrix} \omega\mu[\hat{n}\times H_+(r)]\cdot N_{-mn}(kr,\theta r\varphi) \\ -ik\ [\hat{n}\times E_+(r)]\cdot M_{-mn}(kr,\theta r\varphi) \end{Bmatrix} \qquad \textbf{2.5.48}$$

Here E_+ and H_+ denotes the fields outside the particle surface. To find expansion coefficients for the scattered wave from equations 2.5.47 and 2.5.48 we must know electric and magnetic fields in the exterior side of the surface S.

Expressing the electric field E_{-} in the interior of the particle as in equation 2.5.6, we get

$$\mathbf{E}_{inn}(\mathbf{r}) = \sum_{n'=1}^{\infty} \sum_{m'=-n'}^{n'} \left[c_{m'n'} Rg\mathbf{M}_{m'n'}(k_{inn}\mathbf{r}) + d_{m'n'} Rg\mathbf{N}_{m'n'}(k_{inn}\mathbf{r}) \right] \quad ; \quad \mathbf{r} \in V_{int} \tag{2.5.49}$$

where

$$k_{inn}^2 = k^2 n^2 \tag{2.5.50}$$

From equation 2.5.6 we can express the magnetic field H_{-} in the interior of the particle as,

$$\mathbf{H}_{inn}(\mathbf{r}) = \frac{k_{inn}}{i\omega\mu} \sum_{n'=1}^{\infty} \sum_{m'=-n'}^{n'} \left[c_{m'n'} Rg\mathbf{N}_{m'n'}(k_{inn}\mathbf{r}) + d_{m'n'} Rg\mathbf{M}_{m'n'}(k_{inn}\mathbf{r}) \right] \quad ; \quad \mathbf{r} \in V_{int} \tag{2.5.51}$$

According to the boundary conditions, the tangential components of the electric and magnetic fields are continuous. Thus,

$$\left.\begin{aligned} \hat{n} \times E_{+}(r) &= \hat{n} \times E_{-}(r) \\ \hat{n} \times H_{+}(r) &= \hat{n} \times H_{-}(r) \end{aligned}\right\}, \qquad r \in S \tag{2.5.52}$$

Inserting above equations 2.5.49 and 2.5.51 into equations 2.5.47 and 2.5.48 a relation between the expansion coefficients *c* and *d* for internal fields and the expansion coefficients *a* and *b* for the incident fields can be established as given below,

$$\begin{bmatrix} a \\ b \end{bmatrix} = Q \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} Q^{11} & Q^{12} \\ Q^{21} & Q^{22} \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} \tag{2.5.53}$$

where [5, 126]

$$Q^{11}_{mnm'n'} = -ik_{inn}\, k\, J^{21}_{mnm'n'} - ik^2\, J^{12}_{mnm'n'} \tag{2.5.54}$$

$$Q^{12}_{mnm'n'} = -ik_{inn}\, k\, J^{11}_{mnm'n'} - ik^2\, J^{22}_{mnm'n'} \tag{2.5.55}$$

$$Q^{21}_{mnm'n'} = -ik_{inn}\, k\, J^{22}_{mnm'n'} - ik^2\, J^{11}_{mnm'n'} \tag{2.5.56}$$

$$Q^{22}_{mnm'n'} = -i\,k_{inn}\,k\;\; J^{12}_{mnm'n'} - i\,k^2\,J^{21}_{mnm'n'} \qquad \textbf{2.5.57}$$

and

$$J^{11}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}}\cdot\left\{Rg\mathbf{M}_{m'n'}(k_{inn}\,r,\theta,\varphi)\times\mathbf{M}_{-mn}(k\,r,\theta,\varphi)\right\} \qquad \textbf{2.5.58}$$

$$J^{12}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}}\cdot\left\{Rg\mathbf{M}_{m'n'}(k_{inn}\,r,\theta,\varphi)\times\mathbf{N}_{-mn}(k\,r,\theta,\varphi)\right\} \qquad \textbf{2.5.59}$$

$$J^{21}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}}\cdot\left\{Rg\mathbf{N}_{m'n'}(k_{inn}\,r,\theta,\varphi)\times\mathbf{M}_{-mn}(k\,r,\theta,\varphi)\right\} \qquad \textbf{2.5.60}$$

$$J^{22}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}}\cdot\left\{Rg\mathbf{N}_{m'n'}(k_{inn}\,r,\theta,\varphi)\times\mathbf{N}_{-mn}(k\,r,\theta,\varphi)\right\} \qquad \textbf{2.5.61}$$

where $\hat{\mathrm{n}}$ is the surface normal vector.

Similarly substituting equations 2.5.49 and 2.5.51 into equations 2.5.45 and 2.5.46 another relation between the expansion coefficients c and d for internal fields and the expansion coefficients p and q for the incident fields can be found easily,

$$\begin{bmatrix} p \\ q \end{bmatrix} = -\mathrm{Rg}Q\begin{bmatrix} c \\ d \end{bmatrix} = -\begin{bmatrix} \mathrm{Rg}Q^{11} & \mathrm{Rg}Q^{12} \\ \mathrm{Rg}Q^{21} & \mathrm{Rg}Q^{22} \end{bmatrix}\begin{bmatrix} c \\ d \end{bmatrix} \qquad \textbf{2.5.62}$$

where

$$RgQ^{11}_{mnm'n'} = -i\,k_{inn}\,k\,RgJ^{21}_{mnm'n'} - i\,k^2\,RgJ^{12}_{mnm'n'} \qquad \textbf{2.5.63}$$

$$RgQ^{12}_{mnm'n'} = -i\,k_{inn}\,k\;\; RgJ^{11}_{mnm'n'} - i\,k^2\,RgJ^{22}_{mnm'n'} \qquad \textbf{2.5.64}$$

$$RgQ^{21}_{mnm'n'} = -i\,k_{inn}\,k\,RgJ^{22}_{mnm'n'} - i\,k^2\,RgJ^{11}_{mnm'n'} \qquad \textbf{2.5.65}$$

$$RgQ^{22}_{mnm'n'} = -i\,k_{inn}\,k\;\; RgJ^{12}_{mnm'n'} - i\,k^2\,RgJ^{21}_{mnm'n'} \qquad \textbf{2.5.66}$$

and

$$RgJ^{11}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}}\cdot\left\{Rg\mathbf{M}_{m'n'}(k_{inn}\,r,\theta,\varphi)\times Rg\mathbf{M}_{-mn}(k\,r,\theta,\varphi)\right\}$$

2.5.67

$$RgJ^{12}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}} \cdot \left\{ RgM_{m'n'}(k_{inn}\, r,\theta,\varphi) \times RgN_{-mn}(k\, r,\theta,\varphi) \right\}$$

2.5.68

$$Rg\, J^{21}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}} \cdot \left\{ RgN_{m'n'}(k_{inn}\, r,\theta,\varphi) \times RgM_{-mn}(k\, r,\theta,\varphi) \right\}$$

2.5.69

$$RgJ^{22}_{mnm'n'} = (-1)^m \int_S dS\,\hat{\mathbf{n}} \cdot \left\{ RgN_{m'n'}(k_{inn}\, r,\theta,\varphi) \times RgN_{-mn}(k\, r,\theta,\varphi) \right\}$$

2.5.70

where $\hat{\mathrm{n}}$ is the surface normal vector.

Comparing equations 2.5.24, 2.5.53 and 2.5.62, it can be found that the T-matrix and Q-matrix relates as follows

$$\mathbf{T} = -(Rg\mathbf{Q})\,\mathbf{Q}^{-1} \quad . \qquad \text{2.5.71}$$

The surface integrals in equations 2.5.58 - 2.5.61 and 2.5.67 - 2.5.70 are calculated over the particle surface using appropriate coordinate systems and quadrature formulas [5]. It can be observed that Q and RgQ do not depend on the direction of incidence and polarization state of the incident wave. Hence, T-matrix can be calculated only once without the necessity to recalculate Q and RgQ for doing orientational averaging [126].

2.5.3. T-matrix computation for randomly oriented rotationally symmetric particles: numerical algorithm and convergence procedure

To correlate the experimental light scattering results of nonspherical samples such as carbon black, graphite, hexagonal ice analogue crystals, fresh water diatoms etc. measures by using the light scattering setup, T-matrix computations on randomly oriented axially symmetric particles were performed. For such particles T-matrix divides itself into independent submatrices [5], which decreases the computation time significantly.

For a rotationally symmetric particle, equations 2.5.68 - 2.5.61 takes the form [5],

$$J^{11}_{mnm'n'} = -\frac{i}{2}\delta_{mm'}\left[\frac{(2n+1)(2n'+1)}{n(n+1)n'(n'+1)}\right]^{\frac{1}{2}} \times \int_{-1}^{+1} d(\cos\theta) r^2 h_n^{(1)}(kr) j_{n'}(k_{inn} r)\left[\pi_{mn}(\theta)\tau_{mn'}(\theta) + \tau_{mn}(\theta)\pi_{mn'}(\theta)\right] \quad \textbf{2.5.72}$$

$$J^{12}_{mnm'n'} = -\frac{1}{2}\delta_{mm'}\left[\frac{(2n+1)(2n'+1)}{n(n+1)n'(n'+1)}\right]^{\frac{1}{2}} \times \int_{-1}^{+1} d(\cos\theta) r^2 j_{n'}(k_{inn} r) \left\{\frac{1}{kr}\frac{d}{d(kr)}\left[kr h_n^{(1)}(kr)\right] \times \left[\pi_{mn}(\theta)\pi_{mn'}(\theta) + \tau_{mn}(\theta)\tau_{mn'}(\theta)\right] + \frac{r_\theta}{r} n(n+1)\frac{h_n^{(1)}(kr)}{kr} d^n_{0m}(\theta)\tau_{mn'}(\theta)\right\} \quad \textbf{2.5.73}$$

$$J^{21}_{mnm'n'} = -\frac{1}{2}\delta_{mm'}\left[\frac{(2n+1)(2n'+1)}{n(n+1)n'(n'+1)}\right]^{\frac{1}{2}} \times \int_{-1}^{+1} d(\cos\theta) r^2 h_n^{(1)}(kr) \left\{\frac{1}{k_{inn} r}\frac{d}{d(k_{inn} r)}\left[k_{inn} r j_{n'}(k_{inn} r)\right] \times \left[\pi_{mn}(\theta)\pi_{mn'}(\theta) + \tau_{mn}(\theta)\tau_{mn'}(\theta)\right] + \frac{r_\theta}{r} n'(n'+1)\frac{j_{n'}(k_{inn} r)}{k_{inn} r} d^{n'}_{0m}(\theta)\tau_{mn}(\theta)\right\} \quad \textbf{2.5.74}$$

$$J^{22}_{mnm'n'} = -\frac{i}{2}\delta_{mm'}\left[\frac{(2n+1)(2n'+1)}{n(n+1)n'(n'+1)}\right]^{\frac{1}{2}} \times \int_{-1}^{+1} d(\cos\theta) r^2 \left(\frac{1}{kr}\frac{d}{d(kr)}\left[kr h_n^{(1)}(kr)\right] \frac{1}{k_{inn} r}\frac{d}{d(k_{inn} r)}\left[k_{inn} r j_{n'}(k_{inn} r)\right] \times \left[\pi_{mn}(\theta)\tau_{mn'}(\theta) + \tau_{mn}(\theta)\pi_{mn'}(\theta)\right] + \frac{r_\theta}{r}\left\{ n(n+1)\frac{h_n^{(1)}(kr)}{kr}\frac{1}{k_{inn} r}\frac{d}{d(k_{inn} r)}\left[k_{inn} r j_{n'}(k_{inn} r)\right] + n'(n'+1)\frac{j_{n'}(k_{inn} r)}{k_{inn} r}\frac{1}{kr}\frac{d}{d(kr)}\left[kr h_n^{(1)}(kr)\right]\right\}\pi_{mn}(\theta) d^{n'}_{0m}(\theta)\right)$$

2.5.75

where $r_\theta = \frac{\partial r}{\partial \theta}$ **2.5.76**

Similarly, expressions for $RgJ^{kl}_{mnm'n'}$ can be obtained from equations 2.5.72 - 2.5.76 simply by replacing $h_n^{(1)}(kr)$ with $j_n(kr)$ [5]. Equations 2.5.72 - 2.5.76 can be evaluated by means of Gauss quadrature applied to the interval $[-1,+1]$ and given by,

$$\int_{-1}^{+1} f(x)dx \approx \sum_{p=1}^{N_g} w_p f(x_p) \qquad \textbf{2.5.77}$$

where x_p and w_p are quadrature division points and weights respectively.

The functions $r(\theta)$ and $r_\theta(\theta)$ can be calculated analytically from the shape equation of the particle. For rotationally symmetric particles,

$$r(\pi-\theta) = r(\theta) \qquad \textbf{2.5.78}$$

$$r_\theta(\pi-\theta) = -r\theta(\theta) \qquad \textbf{2.5.79}$$

The Wigner d-functions $d^n_{0m}(\theta)$ and its derivative $\frac{d}{d\theta}\left(d^n_{0m}(\theta)\right)$ can be calculated by using the recurrence relations [5, 243],

$$0 = \frac{\sqrt{(n+1)^2-m^2}\sqrt{(n+1)^2-m'^2}}{(n+1)(2n+1)} d^{n+1}_{mm'}(\theta) + \left\{\frac{mm'}{n(n+1)} - \cos(\theta)\right\} d^n_{mm'}(\theta) + \frac{\sqrt{n^2-m^2}\sqrt{n^2-m'^2}}{n(2n+1)} d^{n-1}_{mm'}(\theta)$$

2.5.80

$$\frac{d}{d\theta}\left(d^n_{mm'}\right) = \frac{1}{\sin\theta}\left[-\frac{(n+1)\sqrt{n^2-m^2}\sqrt{n^2-m'^2}}{n(2n+1)} d^{n-1}_{mm'}(\theta) - \frac{mm'}{n(n+1)} d^n_{mm'}(\theta) + \frac{n\sqrt{(n+1)^2-m^2}\sqrt{(n+1)^2-m'^2}}{(n+1)(2n+1)} d^{n+1}_{mm'}(\theta)\right]$$

2.5.81

starting from the initial values given by,

$$d_{mn}^{n_{min}-1}(\theta)=0 \qquad \textbf{2.5.82}$$

$$d_{mm'}^{n_{min}}(\theta)=\xi_{mm'}2^{-n_{min}}\left[\frac{(2n_{min})!}{(|m-m'|)!(|m+m'|)!}\right]^{\frac{1}{2}}(1-\cos\theta)^{|m-n|/2}(1+\cos\theta)^{|m+n|/2}$$

2.5.83

The values of $\pi_{mn}(\theta)$ and $\tau_{mn}(\theta)$ are calculated from equations 2.5.36 - 2.5.39. Again by using the following symmetry relations reduce the computational effort by a factor of 2 [5].

$$\pi_{mn}(\pi-\theta)=(-1)^{m+n}\pi_{mn}(\theta) \qquad \textbf{2.5.84}$$

$$\tau_{mn}(\pi-\theta)=(-1)^{m+n+1}\tau_{mn}(\theta) \qquad \textbf{2.5.85}$$

$$d_{0m}^{n}(\pi-\theta)=(-1)^{m+n}d_{0m}^{n}(\theta) \qquad \textbf{2.5.86}$$

Unlike Mie theory (section 2.4), the Bessel functions are not expressed in terms of Riccati-Bessel functions as given in equations 2.4.100 and 2.4.101 in T-matrix computations. Bessel functions of the first kind, $j_n(kr)$ and second kind, $y_n(kr)$ are directly calculated by using the recurssion relations and the initial values given by equations 2.4.40 - 2.4.41 and 2.4.42 - 2.4.43 respectively. The values of $j_n(kr)$ and $y_n(kr)$ can now be used to compute the Hankel function of the first kind (equation 2.5.91) and their derivatives.

$$h_n^1(x)=j_n(x)+iy_n(x) \qquad \textbf{2.5.87}$$

Standard Gaussian elimination method is generally used for the matrix inversion process encountered in equation 2.5.71.

Moreover, as already mentioned in T-matrix computations both the incident and the scattered electric fields are expanded in VSWFs, and the transformation of the expansion coefficients of the incident fields into that of the

scattered fields is given by the transformation matrix or T-matrix. Theoretically these expansions are of infinite lengths and thus the T-matrix is of infinite size which is truncated after a suitable finite number of terms, $n = n_{\max}$, as in the case of Mie theory presented in section 2.4 so that the solution converges to the desired accuracy with reasonable consumption of CPU memory. This size $n_{\max}$ depends on the required accuracy of computations and is found by increasing the size of $\mathbf{Q}$ and $Rg\mathbf{Q}$ matrices in unit steps until an accuracy criterion is satisfied. However different elements of the $\mathbf{Q}$ matrix can differ by many orders of magnitude, thus making the numerical calculation of the inverse matrix $\mathbf{Q}^{-1}$ an ill-conditioned process and strongly influenced by round-off errors. i.e. small errors in $\mathbf{Q}$ may result in large errors of $\mathbf{Q}^{-1}$ [5, 126, 241].

Following Mishchenko [5, 241], the value of $n_{\max}$ is found by using a convergence criterion where the values of $C_1 (= C_{ext}, m = 0)$ and $C_2 (= C_{sca}, m = 0)$ are calculated until the relative differences between $[C_1(n_{\max}), C_1(n_{\max} - 1)]$ and $[C_2(n_{\max}), C_2(n_{\max} - 1)]$ are less than the required accuracy Δ [242] as given by [241],

$$\max\left[\left|\frac{C_1(n_{\max}) - C_1(n_{\max} - 1)}{C_1(n_{\max})}\right|, \left|\frac{C_2(n_{\max}) - C_2(n_{\max} - 1)}{C_2(n_{\max})}\right|\right] \le 0.1\Delta \qquad \textbf{2.5.88}$$

where,

$$C_1(n_{\max}) = -\frac{2\pi}{k^2}\mathrm{Re}\sum_{n=1}^{n_{\max}}(2n+1)\left(T_{0n0n}^{11} + T_{0n0n}^{22}\right) \qquad \textbf{2.5.89}$$

$$C_2(n_{\max}) = \frac{2\pi}{k^2}\sum_{n=1}^{n_{\max}}(2n+1)\left(\left|T_{0n0n}^{11}\right|^2 + \left|T_{0n0n}^{22}\right|^2\right) \qquad \textbf{2.5.90}$$

The term $n_{\max}$, specifies the size of the matrix Q in calculating the inverse matrix Q^{-1}. Another term $\tilde{n}_{\max} \le n_{\max}$ is used to calculate the optical cross sections and expansion coefficients and can be estimated by using the inequality,

$$\max\left[\left|\frac{C_1(\tilde{n}_{\max})-C_1(n_{\max}-1)}{C_1(n_{\max})}\right|,\left|\frac{C_2(\tilde{n}_{\max})-C_2(n_{\max}-1)}{C_2(n_{\max})}\right|\right]\le 0.1\Delta \qquad \textbf{2.5.91}$$

Another important parameter N_g, i.e., the number of Gaussian quadrature points used in computing the surface integrals (equation 2.5.77), can influence on the accuracy of the T-matrix computations. Initially the value of N_g is chosen as a multiple of $n_{\max}$. After the determination of $n_{\max}$, N_g is increased until $C_1(n_{\max})$ and $C_2(n_{\max})$ converge within the desired accuracy 0.1Δ.

2.5.4. Particle shapes

T-matrix method can be used as a powerful tool for calculating electromagnetic scattering by single, homogenous, arbitrary shaped particles. However, according to the shapes of samples used in the light scattering experiments, the computer program developed in this work calculates the light scattering properties of rotationally symmetric (axisymmetric or revolution) spheroids, finite cylinders and Chebyshev particles with sizes comparable to the wavelength.

The shape of a spheroid in spherical coordinate system can be defined as

$$r(\theta,\phi)=a\left[\sin^2\theta+\frac{a^2}{b^2}\cos^2\theta\right]^{-\frac{1}{2}} \qquad \textbf{2.5.92}$$

where θ is the polar angle, ϕ is the azimuth angle, b is the rotational (vertical) semi-axis and a is the horizontal semi-axis [238, 244]. The shape and size of a spheroid can be completely described by the axial ratio (aspect ratio), $\varepsilon=\frac{a}{b}$ ($\varepsilon=1$ for spheres, $\varepsilon>1$ for oblate spheroids and $\varepsilon<1$ for prolate spheroids) and the radius of the equivalent surface area sphere r_{eqs} (i.e. the radius of the sphere that has the cross-sectional area equal to the averaged projected area of randomly oriented spheroids) given by,

$$r_{eqs} = \frac{1}{2}\left[2a^2 + 2ab\frac{\arcsin(e)}{e}\right]^{\frac{1}{2}} \quad \text{(for prolate spheroids)} \qquad \textbf{2.5.93}$$

and $$r_{eqs} = \frac{1}{2}\left[2a^2 + \frac{b^2}{e}\ln\left(\frac{1+e}{1-e}\right)\right]^{\frac{1}{2}} \quad \text{(for oblate spheroids)} \qquad \textbf{2.5.94}$$

where $$e = \frac{\left(\varepsilon^2 - 1\right)^{\frac{1}{2}}}{\varepsilon} \qquad \textbf{2.5.95}$$

A circular cylinder is a shape which is bounded by side I, $H = \frac{L}{2}$, side II, $H = \frac{L}{2}$ and by side III, $\frac{x^2}{r^2} + \frac{y^2}{r^2} = 1$ [246], where L is the length and r is the radius of the cylinder. The shape of a cylinder can be completely specified by the ratio of diameter, D to length $\varepsilon = \frac{D}{L}$ and and the radius of the equivalent surface area sphere r_{eqs} given by,

$$r_{eqs} = \frac{H}{\left[\frac{2}{3\varepsilon^2}\right]^{\frac{1}{3}}} \qquad \textbf{2.5.96}$$

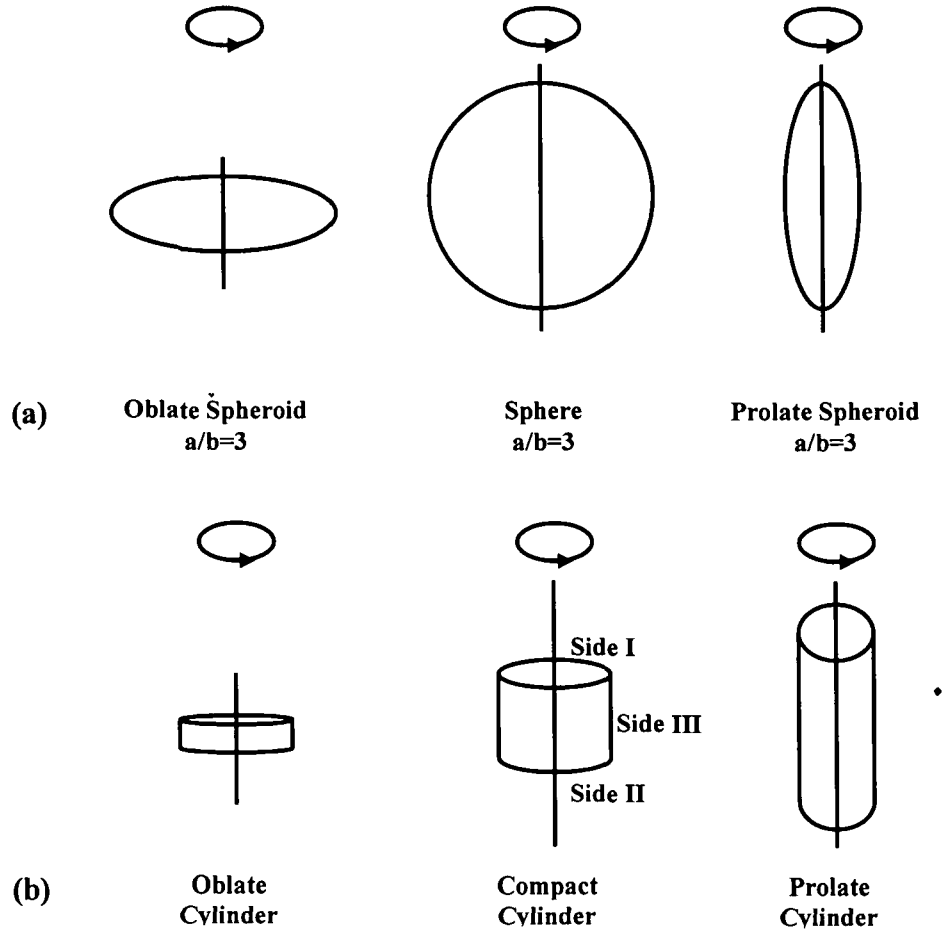

Figure 2.8 (a) Spheroidal and (b) cylindrical particles

It is worth mentioning that for the analysis of the scattering results obtained from the experiments with hexagonal ice analogue crystals, T-matrix data generated by taking cylindrical shapes can be used as it has been reported that pristine ice crystals can be approximated by circular cylinders in scattering calculations [247, 248].

As already mentioned, from these shape equations, the required functions $r(\theta)$ and $r_\theta = \frac{\partial r}{\partial \theta}$ in equations 2.5.76 - 2.5.79 can be calculated.

2.5.5. Orientation and size averaging

In most natural circumstances, scattering particles do not remain in a fixed orientation and instead are distributed over a range of orientations. In such a case, the T-matrix method represents a powerful tool for orientational averaging of the scattering characteristics since the mathematical properties of all the special functions used in T-matrix formulation are well known.

Following Mishchenko [4, 5], we can derive for the general formula of the orientational averaged T-matrix for randomly oriented particles as,

$$< T^{ij}_{mnm'n'} > = \frac{1}{2n+1} \delta_{mm'} \delta_{nn'} \sum_{m1=-n}^{n} T^{ij}_{m_1 n m_1 'n'} \qquad i,j = 1,2 \qquad \textbf{2.5.97}$$

where $\delta_{mm'}$ is the Kronecker delta. The extinction and scattering cross-sections for randomly oriented particles are given by,

$$< C_{ext} > = -\frac{2\pi}{k^2} \mathrm{Re} \sum_{n=1}^{\infty} \sum_{m=-n}^{n} \left[T^{11}_{mnmn} + T^{22}_{mnmn} \right] \qquad \textbf{2.5.98}$$

$$< C_{sca} > = \frac{2\pi}{k^2} \sum_{n=1}^{\infty} \sum_{n'=1}^{\infty} \sum_{m=-n}^{n} \sum_{m'=-n'}^{n'} \sum_{i=1}^{2} \sum_{j=1}^{2} \left| T^{ij}_{mnm'n'} \right|^2 \qquad \textbf{2.5.99}$$

where the sums over n will be truncated at some appropriate value n_{max} calculated as described in section 2.5.3.

The scattering matrices and the coefficients averaged over gamma (equation 2.4.129), normal (equation 2.4.132) and lognormal (equation 2.4.133) can be calculated by using the same procedure as described in section 2.4.3.

2.5.6. Description of the computer program

The required program "TUTscat.c" is written in standard C programming language. The program is capable of computing scattering matrix elements and efficiencies for homogenous nonspherical (spheroids, cylinders and chebyshev) particles having gamma, normal and lognormal size distributions. The program consists of the main program (TUTscat.c) and one associated file (cpxarith.c) for calculating complex algebra. TUTscat.c consists of a main function and 27 associated functiones for convergence evaluation, T-matrix calculation, calculation of Gauss quadrature points, size distribution, Hankel and Bessel functions etc. The program was tested on a computer running Fedora Core Linux 9.0 with gcc (version 4.3.0) installed. The computed values of the scattering matrix elements were written in a user defined file whereas the efficiency factors were written in a predefined file named 'crossec.dat'. The average processing time of the program was 4 - 6 seconds for calculations on monodisperse particles and 20 - 65 seconds for calculations on polydisperse particles. The program is given in Appendix D.

2.5.6.1. *Input parameters*

In the code the size of the particles is specified in terms of the volume-equivalent-sphere radius. The following parameters are request while running the program:

mrr = real part of the particle refractive index

mri = imaginary part of the particle refractive index

lam = wavelength of the incident light (in microns)

ddelt = accuracy of computation

shape = 1 for spheroids

= 2 for circular cylinders

eps = axial ratio for spheroids

= diameter to length ratio for cylinders

ndistr = 0 for monodisperse particles

= 1 for gamma distribution

= 2 for normal distribution

= 3 for normal distribution

axi = particle radius for monodisperse particles

= modal particle radius for polydisperse particles

r1 = minimum particle radius for polydisperse particles

r2 = maximum particle radius for polydisperse particles

b = alpha for gamma distribution

= $(\text{sigma})^2$ for normal distribution

= $[\ln(\text{sigma})]^2$ for lognormal distribution

In addition to the above mentioned parameters the user has to set the following parameters prior to the compilation of the computer program:

npna = number of scattering angles in the range [0, 180]

nkmax= determines the number of Gaussian quadrature points and is such that nkmax+2 is the number of quadrature points in the interval [r1, r2].

ndgs = sets the initial value of Gauss quadrature points (N_g = n_{max}*ndgs)

2.5.6.2. Output parameters

Like "TUMiescat.c" described in section 2.4.5.2 the output parameters such as effective radius, effective variance, scattering angle (ang) and non zero elements of the scattering matrix (S_{11}, $-S_{12}/S_{11}$, S_{22}/S_{11}, S_{33}/S_{11}, S_{34}/S_{11} and S_{44}/S_{11}) are written in a user defined data file. Simillarly, the scattering coefficient (CSCA), extinction coefficient (CEXT), absorption coefficient (CABS), single scattering albedo (albedo) and asymmetry parameter (g) are written in a predefined datafile named 'crossec.dat'.

2.6. Conclusion

This chapter was devoted to the development of two fast and efficient computer programs, TUMiescat.c and TUTscat.c written in standard C to compute the scattering matrix elements and efficiencies for an ensemble of homogenous spherical and nonspherical particles respectively with different size distributions for arbitrary values of incident wavelength, particle radius and index of refraction. The preliminary results of the computer programs were compared with established computer programs and sources in the literature to

optimize its accuracy and reliability which will be described in more detail in section 3.4.2 of this thesis (Chapter III). The programs could directly be used in analysing the measurements obtained with the light scattering setup described in the next chapter.

CHAPTER III: Design and instrumentation of the Light Scattering Setup with the associated development of analytical software

3. 1 General introduction

This chapter presents the design aspects and fabrication of a detector array incorporated laser based light scattering setup (figure 3.1, plate 3.1 - plate 3.3) to characterize the scattering properties of small particulate matter and describes the incorporated instrumentation. The chapter also describes the development of a graphical user interface (GUI) integrated software for the analysis of the experimental results. Section 3.2 gives the detail of the light scattering setup as a whole. Section 3.2.1 to section 3.2.8 deals with the important components of the light scattering setup individually. Section 3.3 points out about the possible errors in the light scattering measurements and gives methods for rectification and correction of the errors in section 3.3.1 to section 3.3.6. Finally section 3.4 explains the software developed for the analysis of the experimental data obtained with the light scattering setup. Here the basic parameters measured are the variations of light intensity due to scattering at different scattering angles and which, after being converted to electrical signal, were obtained in units of voltage. The technique of measurement used in this research work was to observe the scattered intensity from many angles and measure the (i) volume scattering function, $\beta(\theta)$ and (ii) degree of linear polarization, $P(\theta)$.

The design and instrumentation of the light scattering setup [249 - 252] as well as the development of the analytical software constitute a significant part of the original work reported in the thesis.

3.2 TULSS - the light scattering instrument

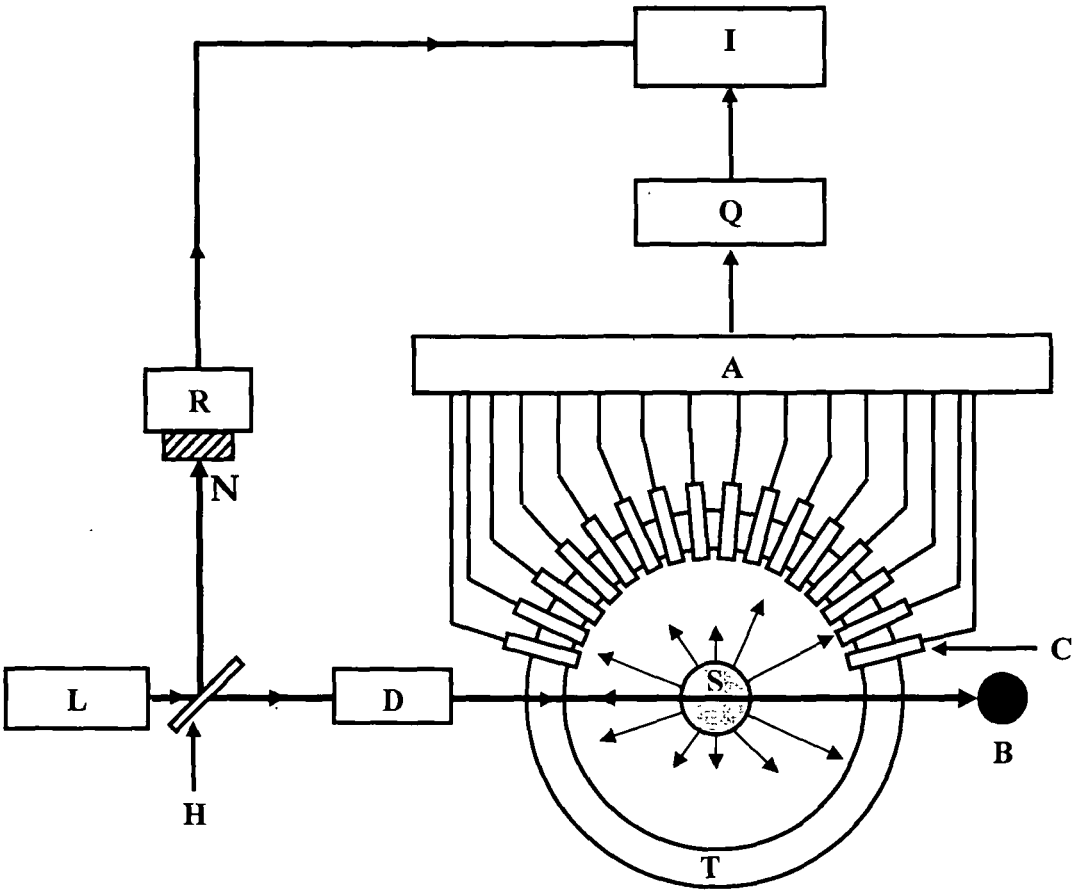

Figure 3.1. Schematic diagram of the light scattering setup. L: laser source; H: beam splitter; D: delivering optics (linear polarizer and quarter wave plates); S: scattering centre; T: turn table; B: beam stop; C: collection optics and photodetectors; A: Signal Amplifier; Q: data acquisition card; I: computer interface with data acquisition software; N: neutral density filter; R: reference detector.

The present light scattering setup was designed and fabricated in the Optoelectronics and Photonics Research Laboratory, Department of Physics, Tezpur University, Assam, India for the measurement of scattering properties of small particles. Figure 3.1 gives the schematic diagram of the setup. The essential components of the setup are an unpolarized laser source, controlled sample holders, photodetector arrangements, data acquisition systems and associated instrumentation. The distance between the laser source L and the scattering

centre S was 250 mm. The beam from the laser source L is split by a 50:50 beam splitter H. The direct beam passes through the delivering optics system D comprising of a linear polarizer and a circular polarizer which is generally used to select the state of polarization of the incident light and placed at a distance of 125 mm from the laser source. The reflected beam after passing through the neutral density filter N enters the reference photodetector R. The reference detector was used to measure the incident light intensity to determine the volume scattering function, $\beta(\theta)$. In the delivering optics unit D, the polarizers were placed in a rotating mount with a vernier dial and were used to change the state of polarization of the incident light as required for the experiments. The polarizer arrangement of the experimental setup was optional and was not specifically used in this experiment as measurement of only the first element of the scattering matrix and the degree of linear polarization was attempted. The unpolarized laser light then passed through the scattering samples which were either sprayed at the scattering centre as a stream of flowing particles (in case of water droplets and aerosols) or were placed in a sample cell made of Pyrex glass at the scattering centre S by a mechanical arrangement (in case of hydrosols) or were embedded in cylindrical polymer matrix placed at the scattering centre S (in case of nanoparticles) and was scattered by the samples. The scattered light intensity, after passing through the collection optics units consisting of appropriate analyzers, was sensed by an array of 16 static Si detectors (BPW34) shown as C in figure 3.1. The detectors had large sensing area (7.5 mm^2) and were mounted on a circular disc. Output signals of the detectors were connected to a high gain, low noise amplifier circuit A and then interfaced to a dedicated data acquisition system (AX5210) shown as Q in figure 3.1, for data recording. The data acquisition card Q has a 12-bit A/D converter with 16 single ended analog input channels and throughput of 30 KHz of memory. With programmable gain of 1, 2, 4, 8, 16 one can define a particular gain value for each input corresponding to the signal level connected to the channel. The whole array of 16 detectors could be rotated simultaneously about an axis

perpendicular to the plane of the circular disc T. Readings were in steps of 1^0 from an angle of 10^0 to 170^0 and each detector was separated from the next one by an angle of 10^0. The combination of L and C represents the nephelometer and point visibility meter. The amplified signals from Q are fed to the computer interfacing unit for data recording and analysis. The whole set up was covered by a black polished metallic enclosure to cutoff electromagnetic noise and beam stops were placed at strategic points to minimize the intensity of stray reflections. Detailed description of the individual components of the light scattering setup is described in the following sections.

Plate 3.1. A photograph of the light scattering setup. L: laser source; S: mechanical jack; N: nebulizer for spraying water droplets; T: turn table; A: detector array; P: aerosol nebulizer; R: external rack.

Plate 3.2. A photograph of the light scattering setup and the associated instrumentation. B: high speed air blower; T: height adjustable stand; I: light scattering setup covered by metallic enclosure to cutoff optical and electromagnetic noise; U: 1KV unaltered power supply (UPS); S: beam stop; C: computer assembly.

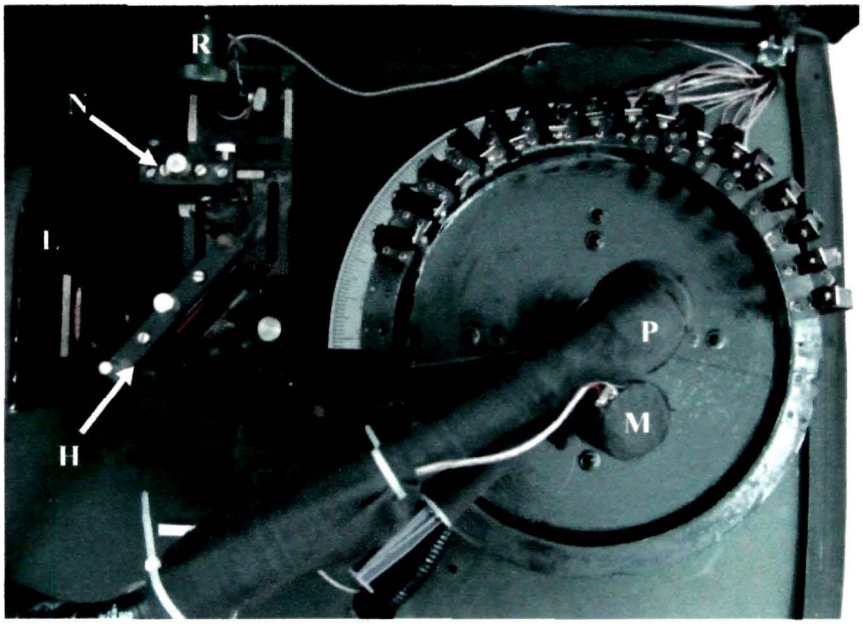

Plate 3.3. Top view of the light scattering setup. L: Laser source; H: beam splitter; N: neutral density filter; R: reference detector; P: aerosol nebulizer; M: electric roller of the aerosol nebulizer.

3.2.1 He-Ne laser

Three He-Ne laser sources [Jain Laser-Tech, Bombay (Mumbai), India] emitting at wavelengths 543.5 nm, 594.5 nm and 632.8 nm with an output power of 5 mW, 5 mW and 2 mW respectively were alternately used as the light sources. Each of the lasers emitted randomly polarized (unpolarized) laser beams with a beam diameter of 1.0 mm and divergence 1.0 mrad. These wavelengths were chosen as it falls in the atmospheric transmission window [216] and can travel long distances without absorption in air.

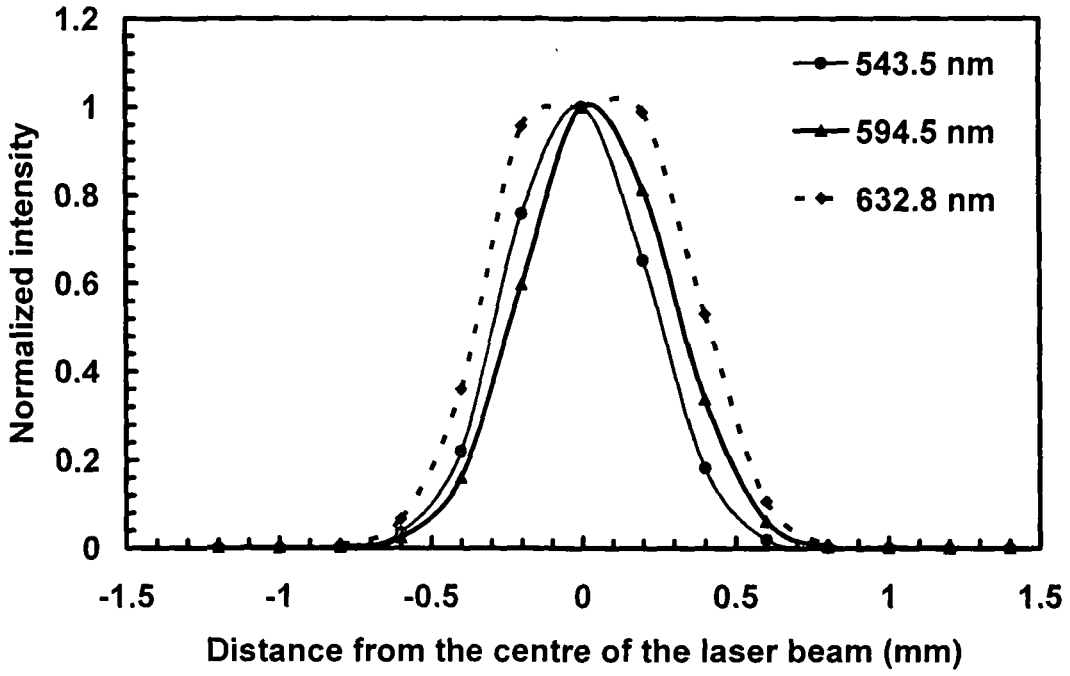

Figure 3.2. Intensity profile of the laser beams measured at a distance 250 mm are shown by solid grey line, solid black line and dotted black line for 543.5 nm, 594.5 nm and 632.8 nm laser wavelengths.

As shown in the plate 3.1 the laser source is mounted on a mechanical jack S whose height can be adjusted for alignment of the beam. The shape of the cross section of the laser beams was measured at the scattering centre. For this, a pinhole was used almost in contact with the detector to minimize the effect of Airy patterns and both the pinhole as well as the detector was moved by using a

micropositioner. It was observed that the diameter of the beam cross-section was approximately 0.00502 cm^2 and the beam intensity was almost flat top in case of 632.8 nm whereas it were Gaussian in case of 594.5 and 543.5 nm as shown in figure 3.2, suggesting that the beam produced fluctuations in the recorded scattered intensities depending on the location of the scattering particles in the scattering volume.

3.2.2 Beam splitter

A broadband (400 nm - 700 nm) non polarizing plate beam splitter (H in figure 3.1 and plate 3.3) was used to separate the incident laser beam into the reflected and transmitted beam (at 90^0 from each other) at a ratio of 50% reflection/50% transmission. The external surfaces of the beam splitter were antireflection coated.

3.2.3 Sample modules

The sample holding arrangement was the most crucial part of the light scattering setup. It is very important to bring the particles into the laser beam at the scattering centre in order to get accurate results. In this work, as measurements were performed on water droplets, aerosols, hydrosols and nanoparticles, the light scattering setup was equipped with four different types of sample holding arrangements which are described in detail in the following sections.

3.2.3.1 Nebulizer to spray liquid particles

To spray liquid particles like water droplets, a commercial neblulizer (Nuneb Pro, MRK Healthcare) [363] was used and was specifically modified for use in the light scattering setup (plate 3.4). It is a powerful piston compressor nebulizer that produces water droplets in the size range 0.5 - 5.0 μm and sprays in front of the laser beam.

Plate 3.4. Nuneb Pro Nebulizer

Some important specifications of the nebulizer are tabulated below (table 3.1).

Table 3.1. Specifications of the NUNEB PRO nebulizer

Description	Value
Type of Compressor	Piston
Pressure	Max 2.5 bar
Air Flow	Min 8 LPM
Noise at 1 meter	55 dBA
Operating times	60 min ON 60 min OFF
Particle sizes	0.5 - 5.0 μm
Dimensions	344mm x 189mm x 120mm
Power Supply	(215-240vAc; 50-60 Hz) (110-130v Ac, 50-60 Hz)
Power Consumption	90 watt maximum
Weight	1.9 Kg (approx.)

After passing through the laser beam the water droplets were removed using a suction assembly consisting of suction pipes and powerful exhaust fans.

3.2.3.2 Design of the aerosol nebulizer

A nebulizer was designed to spray powder aerosol samples as shown in figure 3.3. The nebulizer consisted of an electric blower, a plastic pipe of 15 mm diameter, the sample feeding unit, a projecting spout or nozzle to spray the aerosol grains.

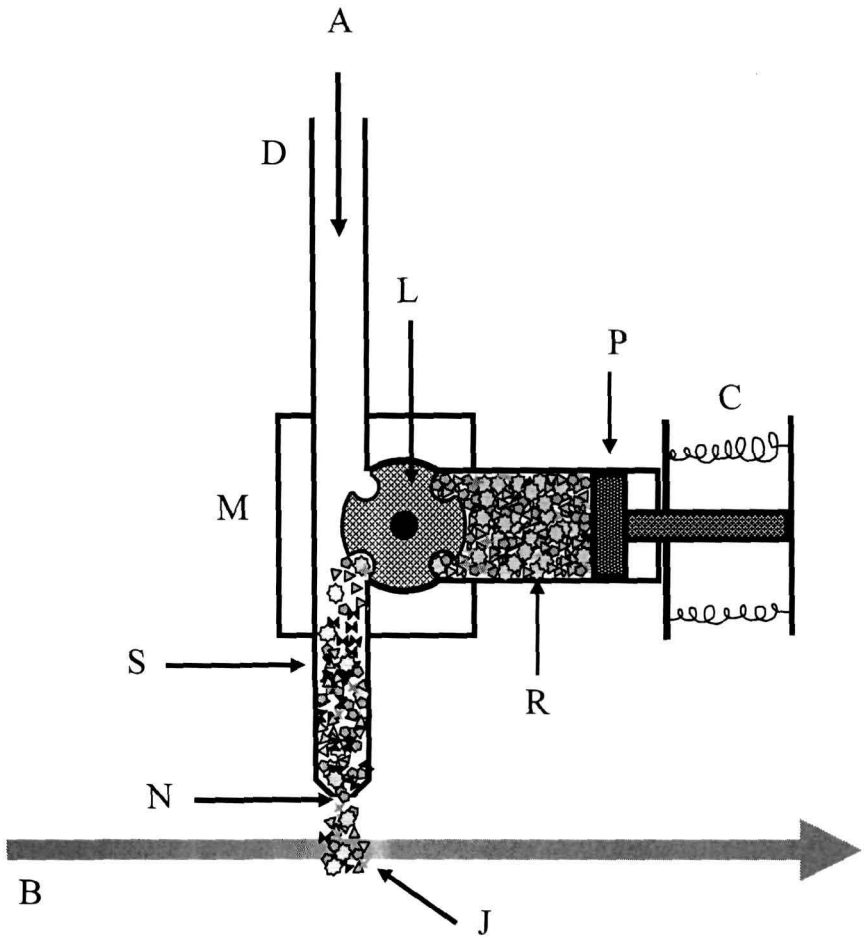

Figure 3.3. Schematic diagram of the aerosol nebulizer. A: high speed air flow from the electric blower; D: plastic nebulizer pipe; L: electric roller with variable speed which brings the aerosols from the reservoir to the air stream; R: aerosol reservoir; M: mechanical holder for the roller and the reservoir; P: piston to push the powders into the holes of the roller; C: high tension coil spring to maintain sufficient pressure by the piston on the aerosols; S: flow of aerosol stream; N: nozzle to spray the aerosol samples; J: aerosol jet; B: laser beam.

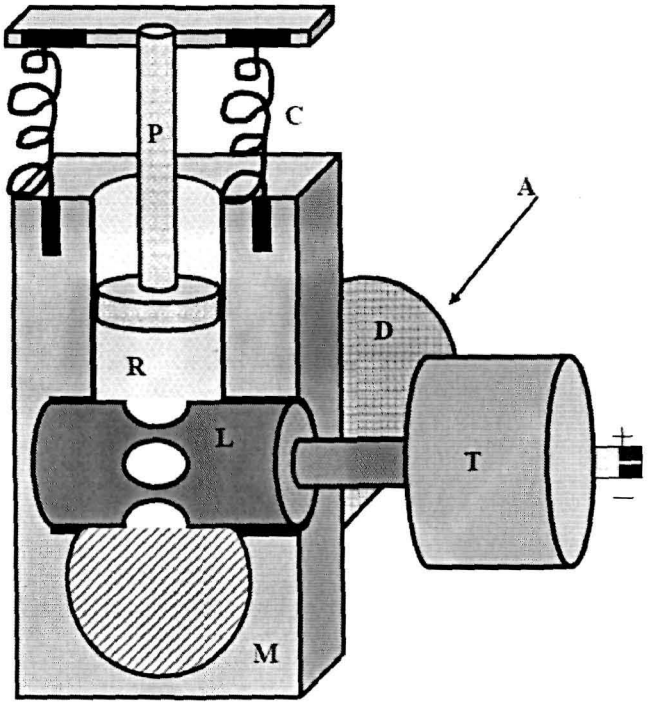

Figure 3.4. Schematic diagram of the sample feeding unit of the aerosol nebulizer. A: high speed air flow from the electric blower; D: plastic nebulizer pipe; R: aerosol reservoir; M: mechanical holder for the roller and the reservoir; P: piston to push the powders into the holes of the roller; C: high tension coil spring the maintain sufficient pressure by the piston on the aerosols; L: electric roller with variable speed which brings the aerosols from the reservoir to the air stream; T: electric motor.

Table 3.2. Important specifications of the electric blower.

Description	Value
Rated voltage	220 volt
Wind volume	2.8 m^3/min
Rated speed	0 - 16000 rpm
Input power	500 W

The variable speed electric blower (Cliff Power Tools, model: EB 500B) was used to generate high speed stream of air A through the nebulizer pipe. The speed of the airflow was controlled by using the rpm (rotation per minute) controller switch. Some important specifications of the blower are given in table 3.2. The airflow A generated by the blower is carried away towards the sample

feeding unit by means of a plastic pipe D of diameter 20 mm. The sample feeding unit consists of a cylindrical feed stock reservoir R, a piston P under constant pressure given with the help of two high tension coil springs C and a 12 V dc motor T coupled with a specially designed roller L to bring the samples into the airflow pipe D. The reservoir R is a glass cylinder of diameter 15 mm having an airtight piston P as shown in figure 3.4. The roller L and the piston P are held together by a mechanical arrangement M. By using a regulated current supply Z to the dc motor the rpm (rotation per minute) of the roller L can be controlled and maintained at a suitable value to ensure single scattering. Depending on the rpm value, the roller removes a well-defined quantity of sample S from the sample reservoir and delivers it into the high speed air stream which is then carried to a nozzle N and sprayed right above the scattering centre as a stream of flowing particles J in front of the laser beam B.

Like the water droplets the sprayed dust particles were carried away by using a suction assembly after passing through the laser beam in order to prevent the contamination of aerosols or dust particles inside the scattering chamber.

3.2.3.3 Sample module for hydrosols

The sample holder unit for hydrosols developed in this work is shown in figures 3.5 and figure 3.6. It consisted of two concentric cylindrical Pyrex glass cuvettes M and N as shown in figure 3.5each having refractive index of 1.5. The outer diameter of the outer cylinder M having wall thickness 2.4 mm is 180 mm and the outer diameter of the inner cylinder N having wall thickness 2.2 mm is 20 mm. The hydrosol samples are kept in the inner cylinder N which is also called the sample cuvette. To minimize the intensity of strong reflection from the sample cuvette that is caused due to the large difference in refractive index between air and sample cuvette [63], it is placed at the centre of the outer cylinder M where the empty space between the two cylinders is filled with glycerin G having almost the same refractive index as that of glass ($\approx$ 1.48). For

such a combination of two cylindrical cuvettes placed coaxially, the specular reflection mentioned above occur at a sufficiently larger distance from the scattering sample. Also, a collimated beam is refracted towards the optical axis when it is incident upon the outer cylinder M. This refracted beam again gets refracted away from the optical axis when it enters the sample cuvette N causing undesirable spreading of the incident beam and also degradation of measurements. It is possible to make correction for this problem by specially designing the outer cylinder M to have flat entrance and exit windows [65]. However there is still a problem with the secondary reflected scattered light from the inner wall of the cylindrical glass cuvette. Details of the correction procedure for this secondary reflection noise will be discussed later in the data reduction section 3.3.

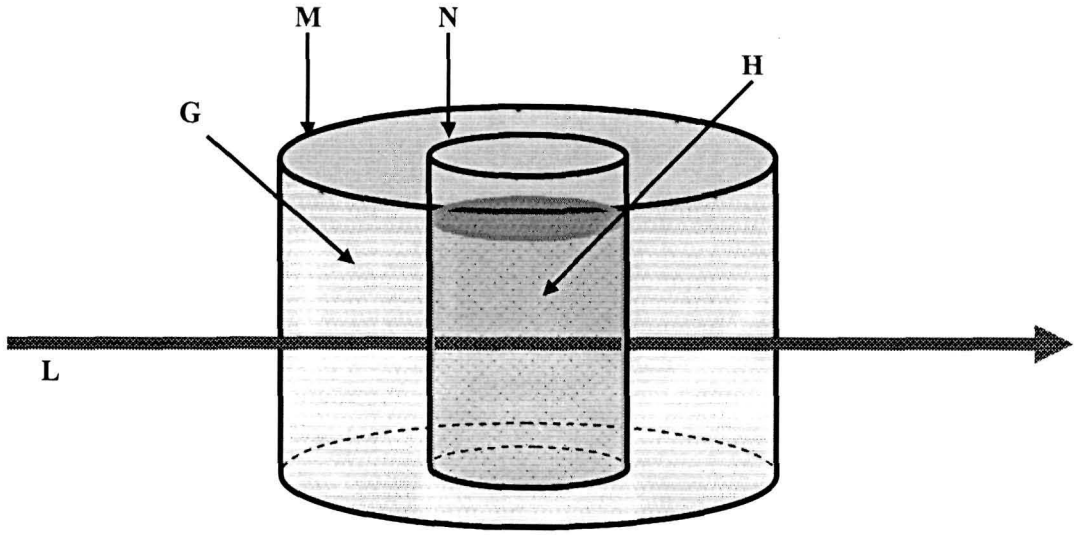

Figure 3.5. Side view of the hydrosol sample holder. L: laser beam; M: index matching basin; N: sample cuvette; G: glycerene; H: hydrosol samples.

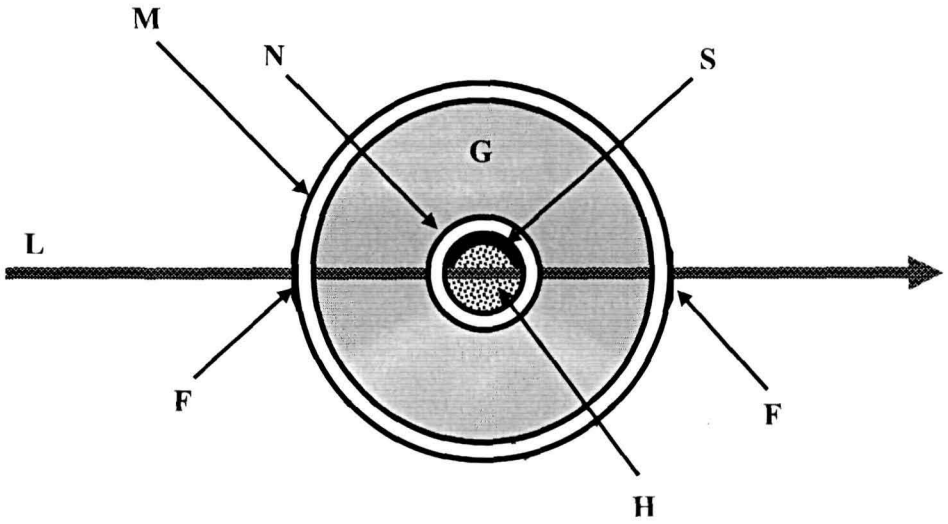

Figure 3.6. Cross sectional view of the hydrosol sample holder. L: laser beam; M: index matching basin; N: sample cuvette; G: glycerene; S: black screen; F: flat entrance and exit windows; H: hydrosol samples.

3.2.3.4 Sample module for nanoparticles and clay particles:

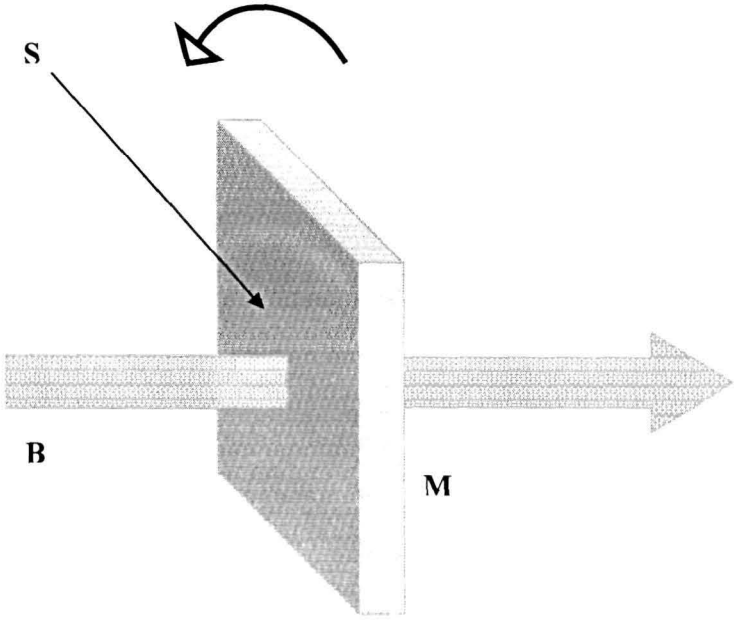

Figure 3.7. Rectangular polymer (PVA) matrix for holding the nanoparticles. B: laser beam; M: rectangular polymer matrix made of PVA; S: uniformly distributed nanoparticles

Unlike micron sized particles, nanoparticles cannot be sprayed and suspended in solutions as they are very reactive and agglomerates to form larger particles as well as degrades very rapidly. In this work, we began with the study of light scattering properties of nanoparticles by embedding them in suitable transparent host polymers with which they do not react. The polymers act as a physical matrix to not only hold the nanoparticles in place, but this matrix also prevents the agglomeration of the nanoparticles with time. Two types of nanoparticle holders were prepared for this purpose - rectangular film type and cylindrical type.

The first type of sample holder (figure 3.7 shows the schematic diagram) were prepared by using polyvinyl alcohol (PVA) matrix to host wide band gap ZnS semiconductor nanoparticles. The reason behind getting the polymer matrix in rectangular shapes was the unique preparation method of such nanoparticles [256 - 260]. PVA is an optically transparent aliphatic polymer having refractive index of 1.55 and dielectric constant of 2.0. It melts at 413K and its density is around 1.08 g/ml. For the preparation of the matrix, a 2.5 wt% PVA solution was prepared in double distilled water, by stirring in a magnetic stirrer with stirring rate at ~200 rpm at a constant temperature of 70^0 C until a transparent solution was formed. As the light scattering experiments were planned to be done in differential mode, PVA samples with and without nanoparticles were prepared in specially prepared sample holders of size 1 sq.cm. The details of the preparation method of the ZnS nanoparticles are given in the next chapter. As the sample holder was rectangular in this case, readings in the angular range 80^o - 90^o were not possible when taking measurements in the angular range 10^o - 170^o. As the nanoparticles under consideration were spherical, scattering measurements did not depend on the rotation of the particles. Therefore, by slightly rotating the sample holder both in clockwise and anticlockwise direction about an axis at the scattering centre and perpendicular to the scattering plane, readings at this angular range (80^o - 90^o) could also be taken separately.

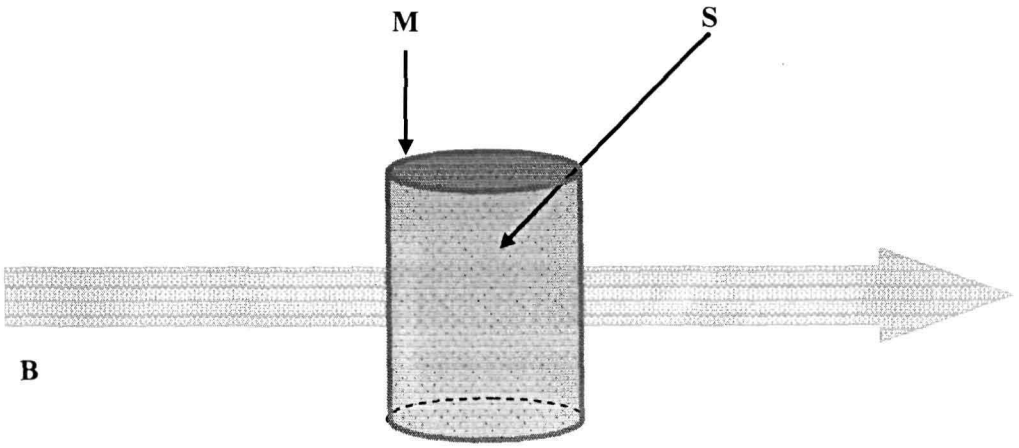

Figure 3.8. Cylindrical polymer matrix for holding the nanoparticles. B: laser beam; M: cylindrical polymer matrix made of epoxy resin; S: uniformly distributed nanoparticle samples. The entrance and exit windows of the laser beam are made flat.

Another type of transparent cylindrical sample holder (figure 3.8 shows the schematic diagram) made of diglycidyl based thermosetting epoxy resin (CY-250, Ciba Geigy , Mumbai) was used to hold bentonite clay particles in front of the laser beam. It is an optically transparent polymer having refractive index of approximately 1.55 at the visible wavelengths. For the preparation of the polymer embedded scattering samples, *ex-situ* technique [349] was adopted wherein bentonite (2% with repect to the epoxy) was first dispersed in Tetrahydrofurane (10 wt%, w/v) and then was added to the epoxy resin (10 wt%, w/v) in the same solvent. The clay was dispersed by using high shear force for 20 min followed by sonication using a single probe sonicator UP200S (Hielscher, Germany) for 10 min. For performing the experiments in differential mode polymer matrix with and without clay particles were used. For this purpose homogeneous mixtures of the poly(amido amine) hardener (HY-850, Ciba Geigy, Mumbai) with pristine resin and polymer embedded bentonite were prepared with 20 phr (parts per 100 g with respect to epoxy resin) separately in a glass beaker at room temperature for 10 min. A high vacuum was applied for about 15 min to remove any trapped volatile gas generated during the mixing.

The mixture was then kept at room temperature in a cylindrical glass tube of diameter 5mm for 36 hours. The glass tube was specially designed so that the cylindrical matrix has flat entrance and exit window for the laser beam. As in the previous case the measurements were performed in differential mode and hence cylindrical matrices with and without the clay particles were prepared in the specially prepared cylindrical glass tubes of diameter 5 mm having flat entrance and exit windows for the laser beam.

A major advantage of the two types of sample holding arrangement described above was the environment it created where the scatterers (nanoparticles or the clay particles) were frozen i.e. not moving with respect to time. This situation helped in taking the average of a large number of scattering measurements on the same set of particles to reduce experimental errors. Notably such measurements gave the scattering properties of the scatterers in the embedding medium only. Therefore such arrangements will be applicable not only for investigating the light scattering properties of polymer embedded nanostructures (such as Au, Cu, ZnO, CdS, TiO2 etc.) but also for measuring scattering properties of other complex shaped nonspherical particles and the results may be used to improve the performance, usefulness and accuracy of the light scattering theories. The results of the laboratory light scattering measurements on such a system of particles under controlled conditions will be presented in Chapter IV. The simplicity and the ease of fabrication of these types of polymer matrices make themselves promising sample holding arrangements for light scattering studies.

3.2.4 Delivering and Collection Optics

As already mentioned, in this research work the volume scattering function $\beta(\theta)$ and the linear polarization ratio $P(\theta)$ were measured, which in turn were related to the first two elements S_{11} and S_{12} of the scattering matrix. The measurement of these two elements essentially requires randomly polarized light. Therefore no polarizers were used (P in figure 3.9 symbolizes no polarizer)

in the delivering optics unit to maintain the random polarization of the beam coming from the unpolarized He-Ne lasers. The volume scattering function, $\beta(\theta)$ and the degree of linear polarization were determined by measuring the quantities in equation 2.2.118 and 2.2.128.

As required for the measurements, the detectors are used to measure the intensity for three states of polarization of the scattered light, that is, unpolarized, perpendicular and parallel polarized light respectively. For this purpose, analyzers optimized for the diode laser wavelength were used in front of the detectors C as shown in the figure 3.9.

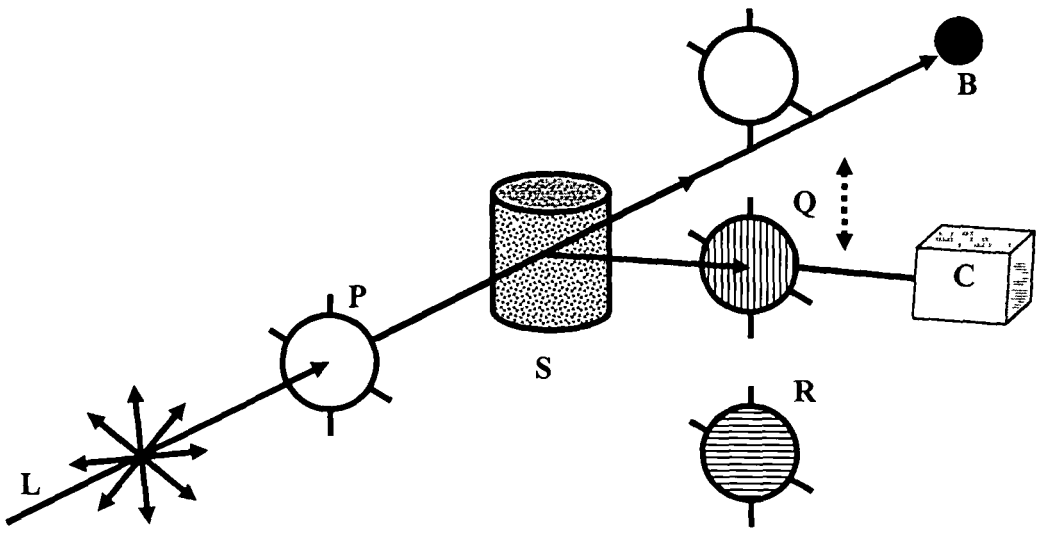

Figure 3.9. Schematic diagram of delivering and collection optics system. L: randomly polarized incident laser beam; S: scattering centre; P: no polarizer; Q: linear polarizer (perpendicular to the scattering plane); R: linear polarizer (parallel to the scattering plane); C: detector; B: beam stop.

3.2.5 Detection system

The detection system of the light scattering system described here consists of two parts: an array of 16 silicon photodiodes (BPW34, SIEMENS) and the OPAMP based amplifier circuit.

3.2.5.1 Part 1: Si - Photo detector Units

Recent rapid growth in the photodiode technology has provided us with high speed, ultrasensitive and energy efficient silicon photodiodes with large sensitive area and small junction capacitance. They can be used to detect the presence or absence of minute light intensities and can be calibrated to measure the intensity of light accurately without cooling or high voltage biasing systems. Therefore such silicon photodiodes are very much suitable for sensing visible and near infrared radiation and preferred as the substitute for photomultiplier tubes when combined with a suitable preamplifier [346]. Some important advantages of silicon photodiodes are:

- they are cheap, compact and light weight,
- they have low noise and consumes less energy,
- their lifetime is comparatively longer,
- they have a broader spectral sensitivity (typically from 400 nm to 1100 nm),
- they can withstand mechanical stress,
- they have excellent linearity of output current as a function of irradiance etc.

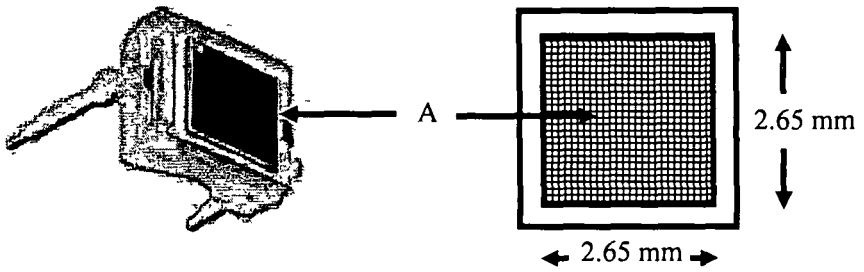

Figure 3.10. BPW34 photodetector.

Table 3.3. Specifications of the photodiode BPW34

Description Value	
Spectral range of sensitivity, λ	400 - 1100 nm
Dimensions of radiant sensitive area, length × breadth	2.65 × 2.6 5 mm²
Dark current (reverse voltage, $V_R = 10$ V)	$2(\leq 30)$ nA
Quantum yield, η (at $\lambda = 850$ nm)	0.90
Open circuit voltage, V_0	$365(\leq 300)$ mV
Rise and fall time of the photocurrent (R_L=50Ω; V_R=5V; λ=850 nm; I_p=800μA)	20 ns
Forward voltage (I_F=100 mA; E=0)	1.3 V
Capacitance (V_R=0 V; f=1 MHz; E=0)	72 pF
Noise equivalent power (V_R=10 V; λ=850 nm)	4.1×10^{-14} W/(Hz)$^{1/2}$
Short circuit current (E_v=1000 lx)	80 μA
Approximate weight	0.1 g

A photodiode is manufactured either in PN or PIN junction configuration. In a PIN configuration the regular PN junction is separated by an intrinsic semiconductor region. As compared to normal PN junction photodiodes the PIN

photodiodes are much faster and more sensitive. The detection system of the present light scattering setup uses silicon PIN photodiodes (BPW34, SIEMENS) which have a large radiant sensitive area of 7.5 mm^2 and short switching time (typical 20 ns). It comes with dual-in-line (DIL) plastic package with high packing density and is especially suitable for applications from 400 nm to 1100 nm [253]. Such 16 photodetectors, fitted with adjustable stands, were mounted on a circular disc to form a photodetector array. The detectors were separated from each other by an angular distance of 10^0 in the scattering plane. Figure 3.10 shows the external dimension of the radiant sensing area of a BPW34 photodetector.Some important specifications of the photodiode BPW34 at 25^o centigrade, given in table 3.3, are appropriate for the light scattering setup as it makes the system portable, compact and efficient.

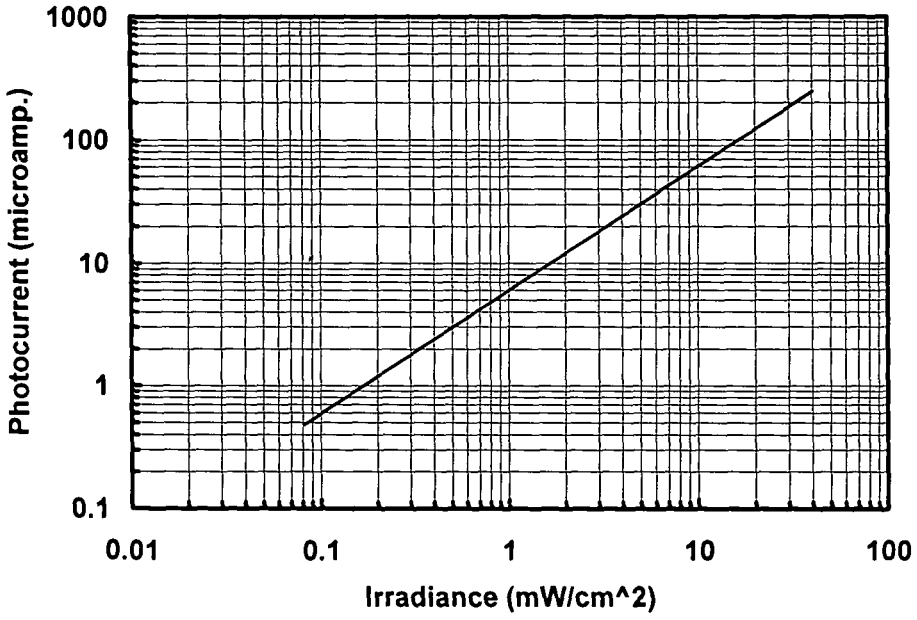

Figure 3.11. Photocurrent versus irradiance graph of BPW34 photodiode at reverse bias voltage, V_R=5 V.

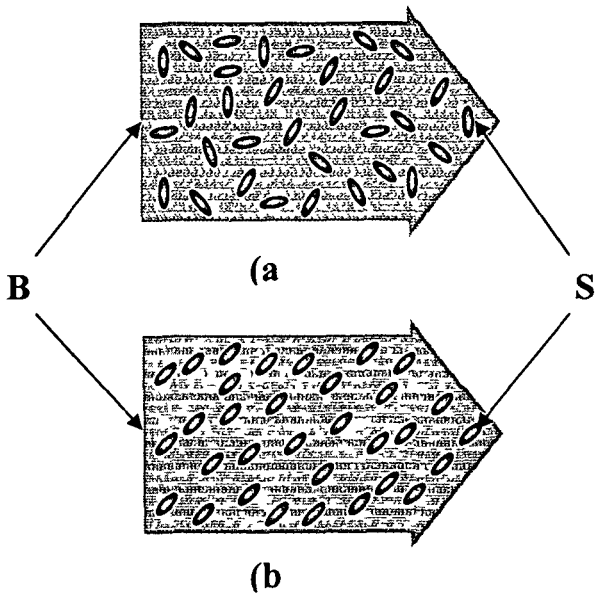

Figure 3.12. (a) Randomly oriented and (b) aligned scattering particles inside a laser beam. B: laser beam; S: scattering particles.

It is worth mentioning that while designing the detection unit of the light scattering setup, detection at azimuthal (φ) angles was not considered. It is because for randomly oriented axially symmetric particles as shown in figure 3.12(a) the scattering matrix elements are functions only of the scattering angle θ [261]. However the scattering matrix elements will also depend on the azimuthal angle φ, when the particles are aligned in a particular orientation as shown in the figure 3.12(b).

Moreover, no pinholes were used to eliminate noise in front of the detectors and the complete sensing area of the detector was utilized for the collection of the scattered light from the scattering volume. However specially designed windows were used in front of the detectors to place the linear polarizers parallel and perpendicular to the scattering plane while measuring the degree of linear polarization. The possible noise collected by the detectors for not using the pinholes in front of them was corrected by using suitable data reduction methods mentioned in section 3.3.

3.2.5.2 Part 2: Amplifier circuit

Output signals of the array of 16 detectors were connected to a high gain, low noise operational amplifier (OPAMP) based amplifier circuit. The eight pin LM308 operational amplifier [254] was chosen as it can be used as a high speed, low drift and low input current amplifier. The following specifications of LM308 (table 3.4) satisfy the requirements of the light scattering setup.

Table 3.4. Important specifications of LM308 operational amplifier.

Description	Value
Supply voltage	±18 volts
Power dissipation	500 mW
Input voltage	±15 volts
Input resistance	70 MΩ
Common mode rejection ratio	110 dB

Moreover pin number 8 of the LM308 was connected to ground via a 100pf capacitor as it improves rejection of ripple present in the dc supply by a factor of 10. It has a linear frequency response within the low frequency range in which the light scattering setup operates as shown in figure 3.13.

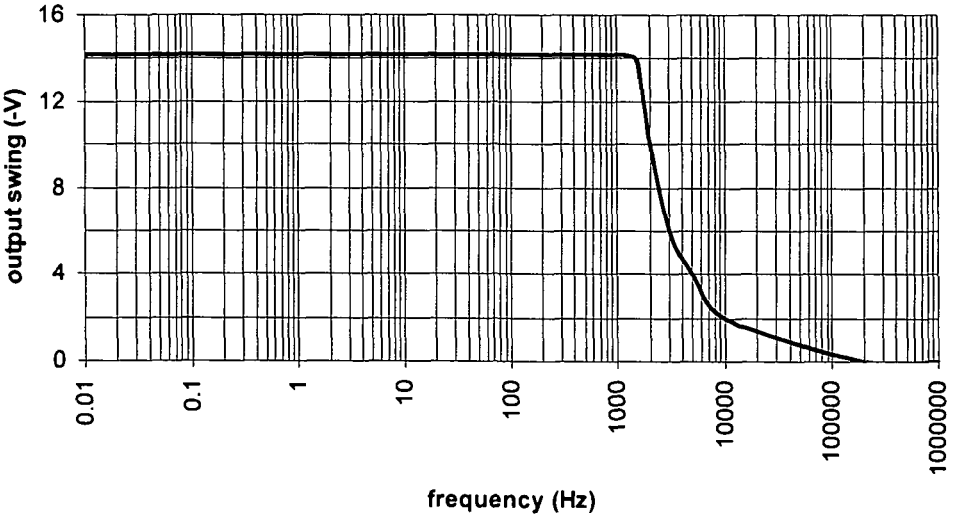

Figure 3.13. Large signal frequency response of the LM308 with pin number 8 connected to ground via a 30pf capacitor.

The equivalent circuit of the photodiode - amplifier unit is shown in figure 3.14. As shown, the photodiode operated in photoconductive mode i.e., it was reverse biased. The light beam entered the unit through the window H and fell on the neutral density filter (only in case of reference detector) and then on the photosensitive area of the photodiode. When the degree of linear polarization was measured, a linear sheet polarizer P (model optosigma) was placed in between the hole and the photodiode D and the scattered light was allowed to pass through it. The photodiode D picked up the light signal from P (optional) and produced photocurrent I_1. A bias voltage of +5 volts was provided to D. I_1 produced a voltage drop V_1 across the load resistance R_1. V_1 was then amplified by the LM308 [254] operational amplifier based stage by A_1 (A_1 = amplification factor) times. This amplification factor A_1 is adjusted, with the laser beam allowed to fall on the detector, to such a value that V_o is less than 5 volts, as it is the maximum value that can be used as the input to the analog-to-digital converter unit.

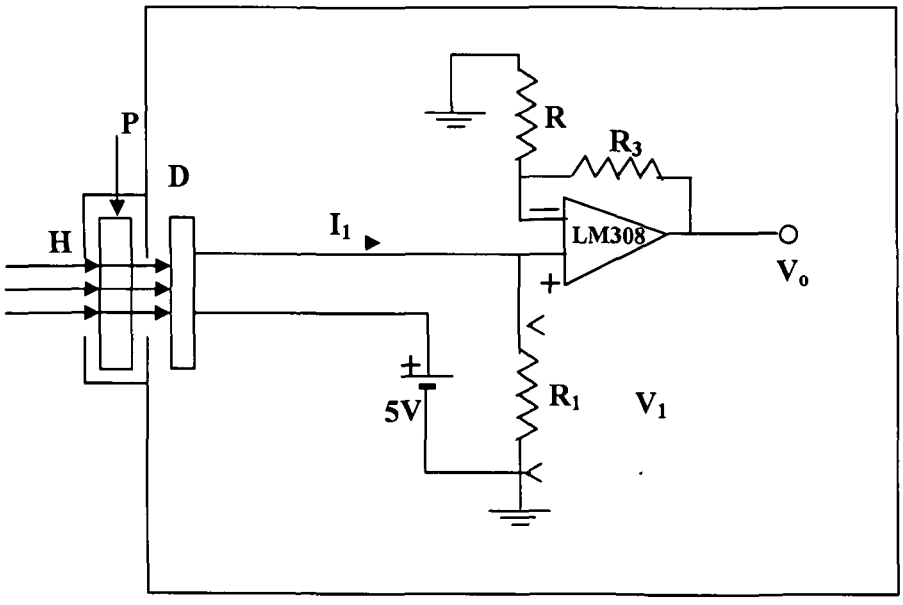

Figure 3.14. Equivalent circuit of the photodetector unit

The graph of phocurrent, I_1 versus illuminance, E_v of BPW34 is linear, as shown in figure 3.11, and fits well with the intensity of the laser source of the light scattering setup. The neutral density filter H (shown in figure 3.1, N in plate 3.3) attenuates the light beam by 933.436 times. N is used in order to keep the intensity of the light incident on R (shown in figure 3.1) at such a level that it does not go into saturation.

The graph in figure 3.11 gives the relationship between I_1 in figure 3.14 and the incident intensity of light E_1 on D as

$$I_1 = 6.0374(E_1)^{1\,0066} \qquad \textbf{3.3.1}$$

As can be seen from figure 3.14, this is equivalent to

$$\frac{V_1}{R_1} = 6.0374(E_1)^{1\,0066} \qquad \textbf{3.3.2}$$

As such the intensity of the radiation entering reference photodetector R is given, using equation 3.3.2 and the value of extinction due to N, that is 933.436, by

$$E_{reference}(in\ mW\ cm^{-2}) = \frac{933.436\left(\frac{1}{6.0374}\right)^{\frac{1}{1\,0066}}\left(\frac{V_1\ in\ mV}{R_1\ in\ ohms}\right)^{\frac{1}{1\,0066}}}{beam\ cross\text{-}section\ in\ sq.cm} \qquad \textbf{3.3.3}$$

where the beam cross-section = 0.00502 sq.cm mentioned in section 3.2.1 is used.

The scattering photodetectors (C in figure 3.1) also have the same configuration as the reference photodetector unit except that the neutral density filter N is absent, and hence, the intensity of the scattered radiation entering the scattering photodetector is

$$E_{scat}(in\ mW\ cm^{-2}) = \left(\frac{1}{6.0374}\right)^{\frac{1}{1\,0066}}\left(\frac{V_1\ in\ mV}{R_1\ in\ ohms}\right)^{\frac{1}{1\,0066}} \qquad \textbf{3.3.4}$$

3.2.6 Data acquisition system

The outputs of the 16 LM308 based amplifiers were interfaced to a dedicated data acquisition system (AX5210) for data recording [255]. The basic function of an analog data acquisition system (DAC) is to convert the analog input signal to the corresponding computer understandable digital format. The DAC system or card, AX5210 consists of four main parts - an analog to digital (A/D) converter having 12 bit resolution, a multiplexer to select exact channel signal to go through the A/D converter, an amplifier to amplify the input signals above the A/D system's minimum resolution, a sample and hold (S/H) circuit for the A/D converter to maintain a constant level of the input signal until the analog to digital conversion is completed. The data acquisition card has 16 single ended analog input channels and throughput of 30 KHz to memory. With programmable gain of 1, 2, 4, 8, 16 one can define a particular gain value for each input channel corresponding to the signal level connected to the channel. This feature gives optimum resolution to each channel's measurement, and minimizes the requirement of front end signal conditioning. The A/D converted data can be collected automatically through software command and will be explained in the later sections. Some important specifications of the data acquisition card are tabulated below (table 3.5).

Table 3.5. Important specifications of the data acquisition card (AX5210).

Description	**Value**
Number of inputs	16 single ended
Resolution	12 bits
Maximum throughput	30 kHz
A/D conversion time	25 μs max.
Channel acquisition time	5 μs max.
System accuracy	± 0.03% FSR
Input ranges	± 5V, ± 2.5V, ± 1.25V, ± 0.625V, ±0.3125 V (all ranges are software selectable)
Input impedence (off channel)	100 MΩ
Input impedence (on channel)	> 10 MΩ
Inherent quantizing error	± 1 LSB
Bias current	± 100 nA
On board clock base frequency	4 MHz
Dimension	(99H × 155W) mm
Weight	250 gm
Operating temperature range	0 to 60 degree centigrade

Function	Pin Number	Pin Number	Function
Channel 0 - analogue input	1	2	Channel 8 - analogue input
Channel 1 - analogue input	3	4	Channel 9 - analogue input
Channel 2 - analogue input	5	6	Channel 10 - analogue input
Channel 3 - analogue input	7	8	Channel 11 - analogue input
Channel 4 - analogue input	9	10	Channel 12 - analogue input
Channel 5 - analogue input	11	12	Channel 13 - analogue input
Channel 6 - analogue input	13	14	Channel 14 - analogue input
Channel 7 - analogue input	15	16	Channel 15 - analogue input
Analogue ground	17	18	N.C.
+ 12 V source	19	20	- 12 V source
N.C.	21	22	N.C.
N.C.	23	24	N.C.
N.C.	25	26	N.C.
Channel 0 - digital output	27	28	Channel 0 - digital input
Channel 1 - digital output	29	30	Channel 1 - digital input
Channel 2 - digital output	31	32	Channel 2 - digital input
Channel 3 - digital output	33	34	Channel 3 - digital input
+ 5 V source	35	36	+ 5 V source
Channel 4 - digital output	37	38	Channel 4 - digital input
Channel 5 - digital output	39	40	Channel 5 - digital input
Channel 6 - digital output	41	42	Channel 6 - digital input
Channel 7 - digital output	43	44	Channel 7 - digital input
+ 5 V source	45	46	+ 12 V source
External trigger input	47	48	N.C.
Digital ground	49	50	Digital ground

Figure 3.15. Pin assignment of the 50 pin CN1 connector on the AX5210 A/D card.

All AX5210 A/D, DIO (digital input output) signals are built in a single 50 pin type CN1 connector. The CN1 pin assignment is shown in the figure 3.15. The pins 18, 21 - 26 and 48 are not used in AX5210. Before using the DAC for data recording it needs to be calibrated for amplifier offset adjustment, A/D offset and full scale measurement by using the AX5210 menu driven calibration program named CAL5210.exe.

Amplifier offset adjustment: It was done to minimize the input amplifier offset error. A sequence of steps was followed as elaborated below.

- Step 1: channel 0 was connected to analog ground.
- Step 2: a digital voltmeter was connected to TP1(+) and TP2(-).
- Step 3: VR3 was trimmed until the voltmeter read out less than 0.5mv.

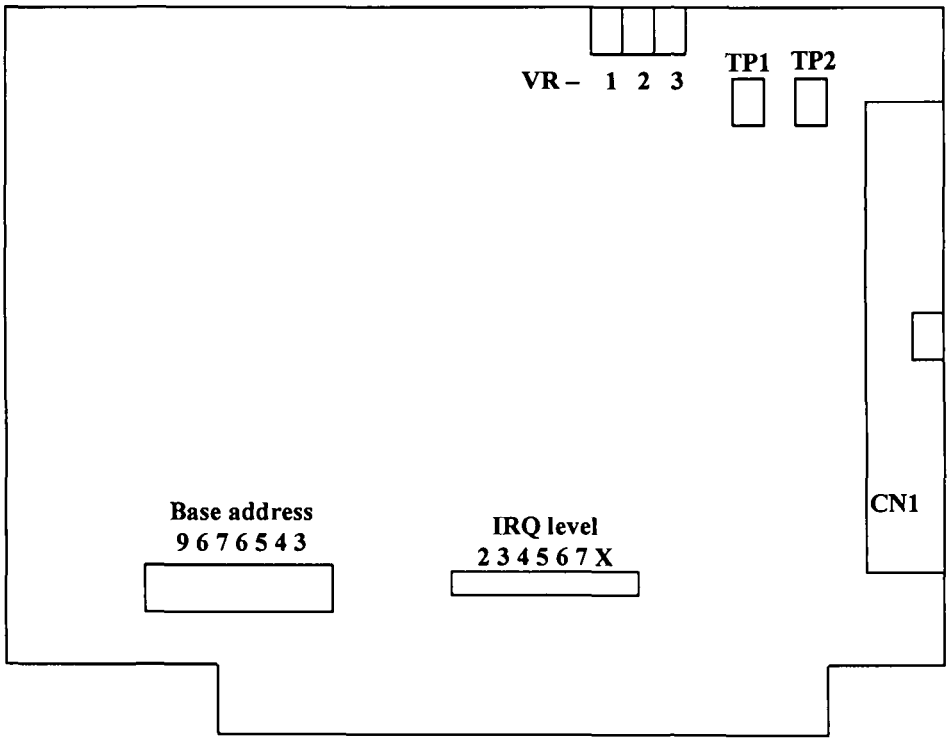

Figure 3.16. Block diagram of AX5210 card that is fitted on the PC.

A/D offset and full scale adjustment: It was performed using the following steps to make sure that the A/D converter was working with the best resolution.

- Step1: a dc voltage 4.9963 V was introduced into channel 1 and channel 0 was connected to analog ground.
- Step 2: VR2 was trimmed for A/D offset until channel 0 read 0.
- Step3: VR1 was trimmed for A/D full scale until channel 1 read between 2046 and 2047.

The calibration for digital input/output check was not performed as it was not required in this work. After calibration, the outputs of the LM308 based preamplifier unit (shown in figure 3.1) were connected to pin numbers 1 to 16 of the AX5210 DAC (shown in figure 3.15). The preamplified signals can be collected, digitized and stored by using the data acquisition program given in the next section. The flowchart of the data acquisition software is given in figure 3.17.

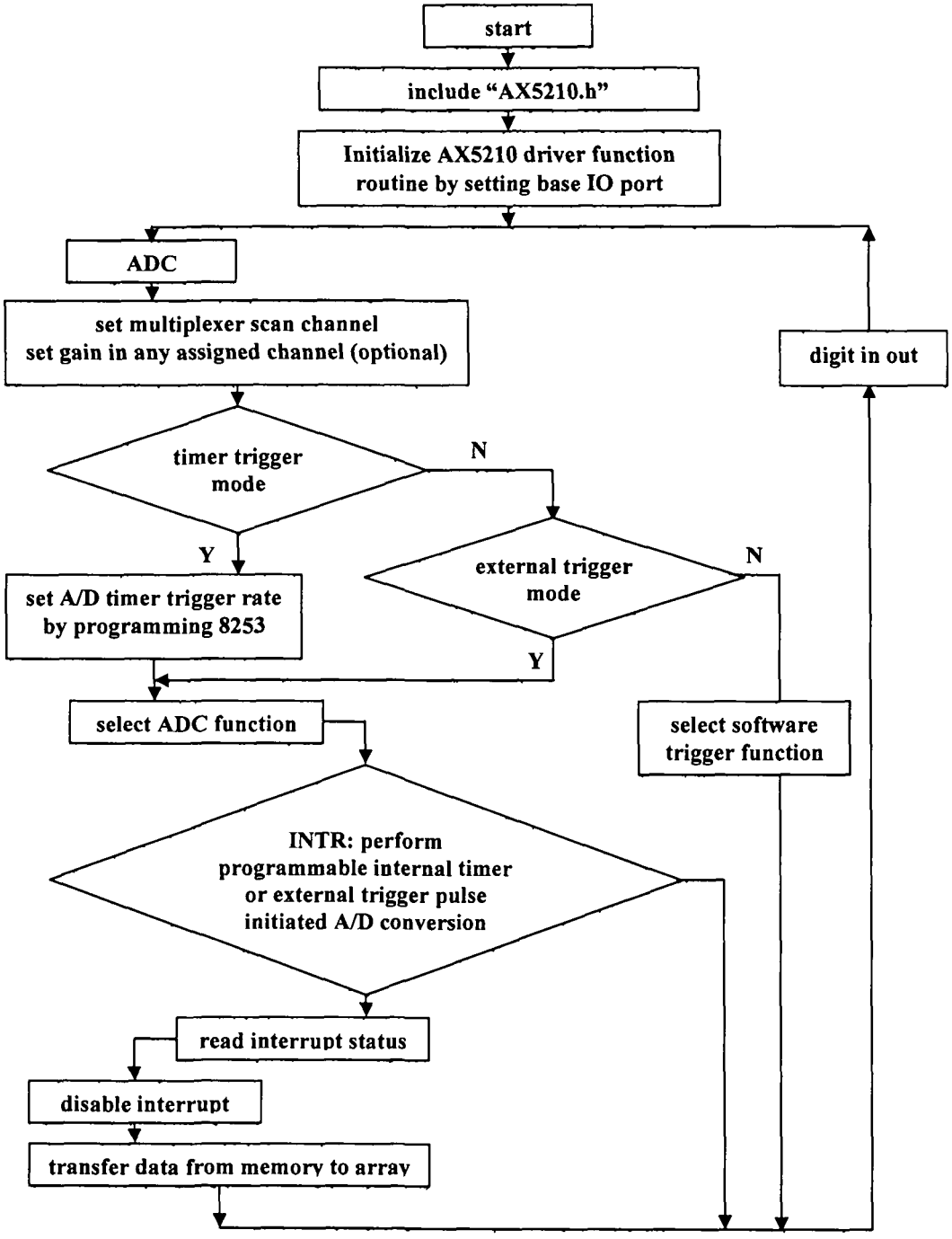

Figure 3.17. flowchart of data acquisition software

The data acquisition program written in standard C language corresponding to the flowchart in figure 3.17 is given below.

Line number Program

```
1     #include<dos.h>
2     #include<stdio.h>
3     #include<conio.h>
4     #include "ax5210.h"
5     #include<alloc.h>
6     #define IOPORT 0x300

      /* AX5210 IO port */
7     int fun,flag;
8     Unsigned int dio[7];
9     int far *ary1;
10    int far *ary2;
11    main ()
12    {
13    unsigned int count=3000;
14    char fl_name[100], coment [150];
15    FILE *fp1;
16    int ch, dat, dat_h, dat_l, i, j;
17    ary1 = farmalloc ((unsigned long) count * size of
      (int));
18    ary2 = farmalloc ((unsigned long) count * size of (int));
19    clrscr();
20    printf("\nEnter data filename :");
21    scanf ("%s", fl_name);
22    printf ("\nEnter comment :");
23    scanf ("%s", coment);
24    fp1=fopen(fl_name,"w");
25    fprintf(fp1,"%s\n",coment);
26    fun=INIT;          /* Initial function */
27    dio[0]=IOPORT;     /* Ioport set */
28    dio[1]=3;          /* IRQ NO. */
29    flag=ax5210(fun,dio,ary1,ary2);
30    if (flag != 0)
31         {
```

```
32          printf("DRIVER  INITIALIZATION  FAILED  !,  flag=%d\n",
             flag);
33          exit(1);
34          }

/* SET GAIN */
35     for (i=0; i<16; i++)
36          {
37          fun=2;    /* Set gain function */
38          dio[0]=i;/* Set channel */
39          dio[1]=16;/* Set gain */
40          flag=ax5210(fun,dio,ary1,ary2);
41          if (flag != 0)
42               {
43               printf("SET   GAIN   FAILED   !,   FLAG   =   %d
                  \n",flag);
44               exit(1);
45               }
46          }

/*SOFTWARE TRIGGER*/
47     outp(IOPORT+2,0);      /*enable software trigger mode*/
48     j=0;
49     do
50          {
51          outp(IOPORT+1,j);     /* set channel */
52          for (i=0; i<10000; i++) ;
53          outp(IOPORT,0);       /* software trigger */
54          while ((i=inp(IOPORT+2) & 0x08) != 0);

/* EOC = 0 ADC ready */
55          dat_l=inp(IOPORT);
56          dat_h=inp(IOPORT+1);
57          ch=(dat_l & 0x0f);
58          dat=(dat_h << 4) + (dat_l >> 4);
59          dat -= 2048;
60          printf("\n CHANNEL %d\n READING %d\n " ,ch,dat);
61          fprintf(fp1,"%d,%d\n",ch,dat);
62          j=j+1;
```

```
63          } while (j < 16);
64      fclose(fp1);
65      return(0);
66      }
```

The program is executed each time an experiment is done with the light scattering setup. The main function is between lines 12 to 66. Several functions predefined with the AX5210 driver are used to develop the data acquisition software. At line 4 the header file 'ax5210.h' is included and line 6 defined the input output port for the DAC. Lines 20 to 33 presents the menu on the screen for entering data file name and any comments such as time of experiment, gains of the photodetectors etc. The DAC AX5210 is initialized at line 26 and line 27 sets the interrupt level. Between lines 35 to 46 is a loop that runs to set the gain for the channels. Between lines 49 and 63 is a loop that runs till the end of the data acquisition program to collect data from the channels and print them in a file whose name was declared at line 21. The data acquisition program described here is the very basic type of program which can be tailored for different requirements. For example, if average value of 100 readings is to be taken, it can be done by running the loop between lines 49 and 63 for 100 times and taking the average of the recorded values.

3.2.7 Beam Stop:

The light beam from the laser source is only partially scattered by the scattering samples. A significant amount of the beam still remains after it crosses the scattering centre and creates undesired noise. To eliminate this portion of laser beam which is not scattered by the scattering samples, an ad hoc beam stop is used. It consists of a hollow beak-shapped tube T fitted with a metallic base B as shown in the figure 3.18. The inside of the beam stop is blackened with a non reflecting paint. Due to the beak like structure the unscattered laser beam L after entering the beam stop tube through the entrance E, suffers multiple reflections,

gets absorbed by the walls of the blackened tube and ultimately the remaining part of the beam leaves the beam stop through the hole S provided in the tube T.

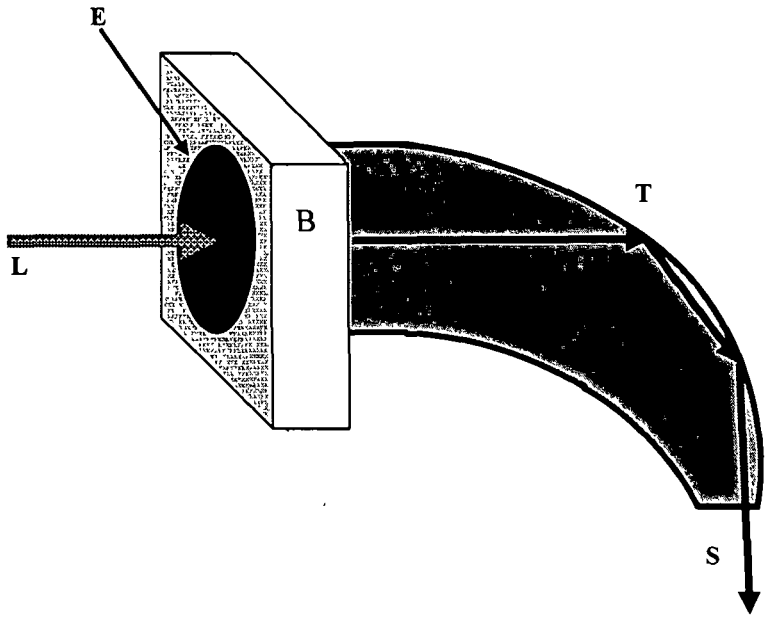

Figure 3.18. Schematic diagram of the beam stop for trapping the unscattered direct light. L: unscattered laser beam; E: entrance of the beam stop; B: mechanical base of the beam stop; T: beam stop tube; S: laser exit window.

3.2.8 Simulation chamber

The laser source, optical elements, detectors, scattering centre etc are installed inside an Alluminium enclosure, whose inside walls were painted black (S in plate 3.2). The dimensions of the enclosure are 1.05 m x 0.85 m with a height of 0.54 m. The black paint used inside the enclosure does not contain scattering pigments. The other components like preamplifier, data acquisition systems are located in an external rack below as shown in the plate 3.1.

3.3 Data reduction methods

There are several crucial error reduction steps that have to be performed during light scattering measurements to eliminate the experimental errors and

ensure the reliability of the results. In this section, the data processing methods which are used to minimize the experimental errors, are discussed.

3.3.1 Alignment of the Light Scattering Setup

The alignment of the light scattering setup ensures that the axis of the incident light beam passes through the rotation axis of the detectors, remains on the detector rotation plane and is aligned to the detector aperture when the later is at the scattering angle of 0°. In this work the whole instrument was mounted on a fabricated stand whose base could be leveled by screws for the alignment purposes. First the laser beam, beam splitter, analyzers (collection optics unit) and the detector which should in the same scattering plane were aligned by using the specially provided adjustable stands.

For the alignment of the laser beam we used the fixture used by Jonasz [64]. The alignment fixture consisted of two screens A and B each having identical grid to mark the beam footprint as it passed through the screen (figure 3.19). The fixture was then mounted on the circular disc or the turn table (T in figure 3.1) where the array of 16 detectors were also mounted. A rough symmetry of the screens about the rotation axis of the detectors or the axis of the circular disc was maintained. The screen grid centers did not need to be aligned with the scattering plane or the desired beam axis. The set of the two screens was then oriented in such a way that the direction from screen A to screen B coincided with the desired direction of the laser beam i.e. this orientation corresponded to the scattering angle 0°. The laser beam was subsequently and arbitrarily positioned so that the center of its footprint got located within the grid area of screen A. This position of the beam footprint, say P_A was noted at screen A. Next the fixture was rotated by 180° so that screen B took the former position of A and vice versa the laser beam. The position of the laser beam footprint, say P_B was noted at screen B. Subsequently the laser beam was properly aligned to make it pass through the same position P_A of screen A at its new position but at the same time keeping the beam footprint at P_B in screen B. On the completion of

the alignment process the laser beam passed through the same position in screen A in its two locations i.e. locations before and after rotation by 180° the laser beam axis passed through and at a right angle to the axis of rotation of the detectors (i.e. axis of the circular disc).

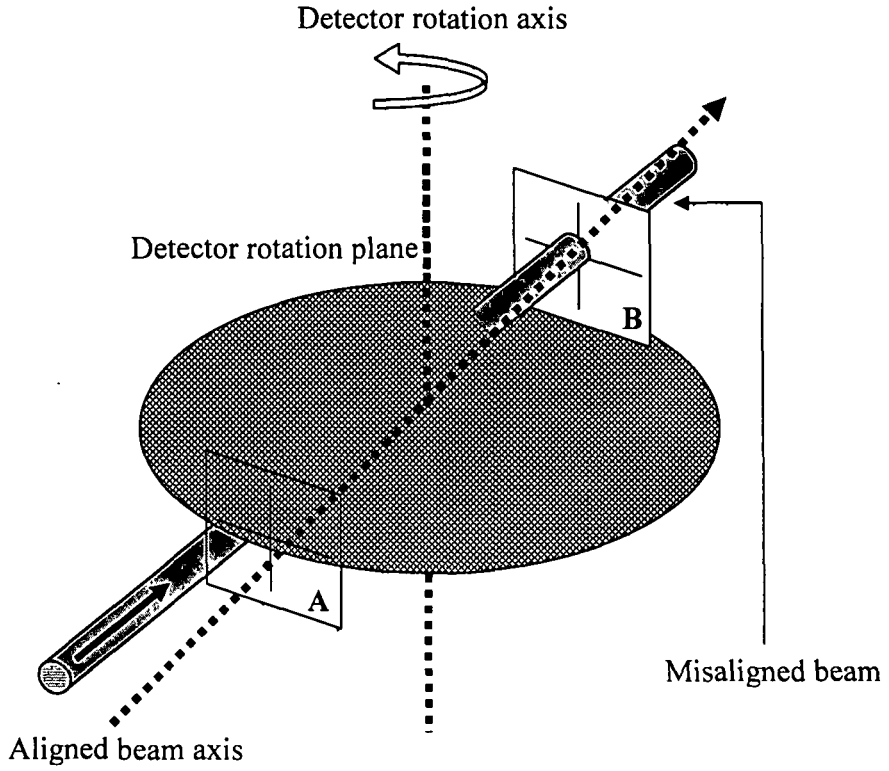

Figure 3.19. The fixture used to align the light scattering setup. The misaligned beam propagates from left to right and from below to above the detector rotation plane

To ensure that the laser beam axis coincide with the detector axis which lied in the detector rotation plane and was aligned to the detector aperture when the later was at the scattering angle of 0°, each of the detectors were placed at that angle one by one and the vertical position and the orientation of the detector axis were adjusted for maximum output signal. To avoid damage, the beam power was appropriately pre-attenuated while performing this alignment step.

3.4.2 Dust removal

The presence of contaminated samples especially airborne particles and dust particles within the scattering chamber is a common problem with the light scattering measurements. Sometimes such particles are larger compared to the particles being measured and therefore scatter the incident radiation strongly, creating some erroneous spikes in the scattering pattern [232]. Such spikes can continue for several consecutive measurements or over several angular measurements and may lead to misinterpretation of the scattering results. Therefore it is necessary to remove dust particles from within the scattering chamber by creating low vacuum or by using strong exhaust fans. In this experiment we used a powerful exhaust fan to remove the contaminating aerosols or dust particles. The process of dust removal was continued until the detectors gave readings at the background noise level which was further corrected by using a special procedure discussed in the next section.

3.3.3 Background noise correction

The background noise is the combination of the dark current signal i.e. the voltage produced by the photodetectors even in the absence of incident light, stray light contribution to the measured signals during the experiment including the undesired light scattered from contaminated aerosol particles in the scattering chamber, electronic noise etc. To minimize the background noise we took our measurements in differential mode, that is, scattered light intensity was initially measured by passing laser light through the scattering centre in the absence of water droplets or aerosol jet and subsequently the measurements were taken by passing laser light through the scattering centre in the presence of water droplets or aerosol jet. Similarly in case of hydrosols the scattered light intensity was initially measured with the cuvette containing distilled water without scattering particles and subsequently the measurements were taken with the cuvette containing scattering particles with distilled water. Again in case of nanoparticles the scattered light intensity was initially measured with the blank

polymer matrix without scattering particles and then measurements were taken with the cylindrical polymer matrix containing nanoparticles under consideration. For all the three cases, the data from the first set of experiments without scattering particles was subtracted from the data of the second set of experiments taken with scattering particles. This took care of the nominal amount of optical and electrical noise that still remained in spite of the use of metallic enclosures and beam stops.

3.3.4 Scattering volume correction

In static light scattering experiments, the scattering volume i.e. the volume that the laser beam occupies in the sample, projected on the detector is determined by the scattering angle, geometry of the scattering volume, distance to the photodetectors and width of the laser beam [63, 232]. This scattering volume changes along with the change of the scattering angle relative to the direction of the illuminating beam as shown in the figure 3.20. The scattering volume is minimum for $\theta = 90^0$ (i.e. the measured intensity will be minimum) and gradually increases on either side. The measured scattering intensity can be corrected for a constant scattering volume by multiplying with the correction factor $\sin\theta$ [2, 63, 64, 232]. However this sine correction is valid upto the scattering angle at which the detection area or the solid angle of the photodetector becomes equal or narrower than the path length of the beam (case I and case II). Outside this area the scattering volume is constant (case III). The standard acceptance range used in this work is between 20^0 and 160^0. Figure 3.21 shows correction factor versus scattering angle graph by which the measured intensity has to be multiplied to correct for the changing scattering volume as seen by the detector.

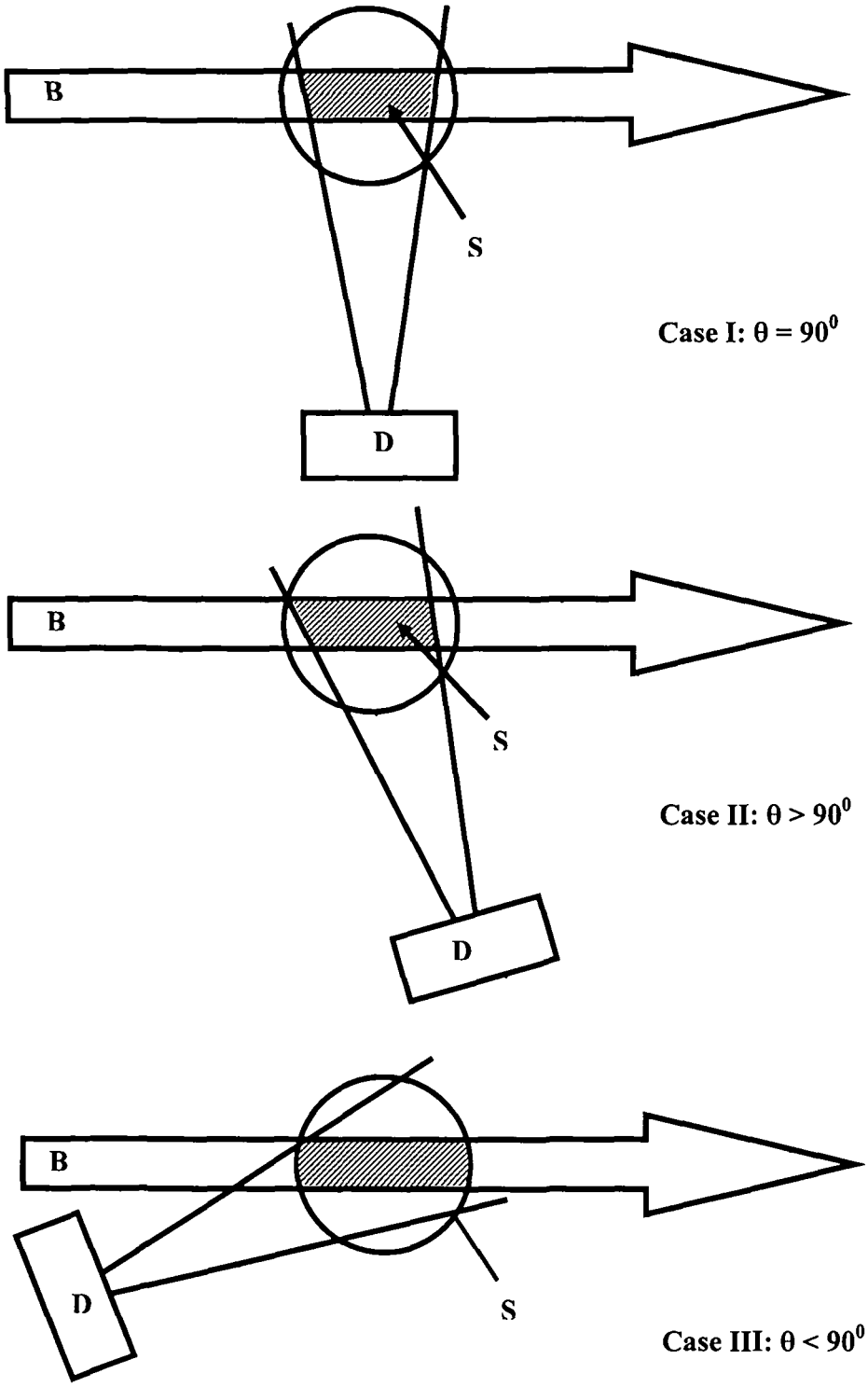

Figure 3.20. Changing scattering volume with scattering angle. Case III shows range of angular acceptance.

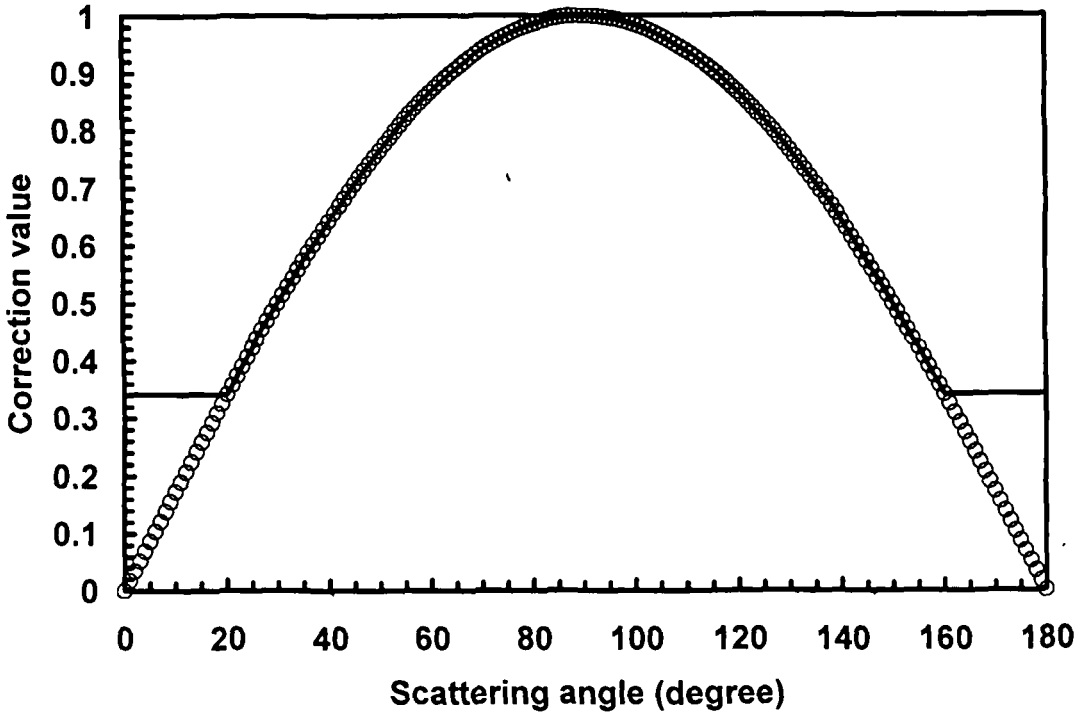

Figure 3.21. Graph for the correction factor versus scattering angle is shown by solid black line which is equal to $sin\theta$ (shown in blank circles) for most of the scattering angle range.

3.3.5 Ensuring single scattering

In this research work, as we are considering our measurements in the single scattering regime, it is very important to optimize the concentration of the particles so that the measurements are not effected by multiple scattering. This can be done by measuring the intensity of scattered light for unpolarized incident light as a function of the concentration of powder samples sprayed in front of the laser beam (powder samples) or particle concentration in the cuvette (for hydrosols) or particle concentration in the cylindrical polymer matrix (for embedded nanoparticles) for a fixed position of the detector [99]. In this work, this was done by measuring the scattered light intensity at the smallest scattering angle used in the experiment i.e. 10^0 for unpolarized incident light. The values of particle concentration upto which the graph between the scattered intensity versus rotor speed and sample concentration is linear indicates the experiment to

be in the single scattering regime. The optimum values of sample concentration are given in the next chapter.

Also, due to the stray light contamination and background light coming mainly from the reflections at the walls of the sample holder, it is usually very difficult to discriminate between the actual scattered signal and noise at lower (less than 20°) and higher (greater than 160°) phase angles and the corrected signal could be lower than the total background contribution at such angles. In such cases, it is very important to test the stability of the polarization measurements because a small error in background correction may lead to abrupt change in the degree of linear polarization. Notably, polarization is independent of the number of scatterers (particle concentration in the scattering volume) in single scattering regime [43]. This polarization stability test was done during the calibration process with polystyrene samples by measuring the polarization as a function of increasing particle concentration within single scattering regime at a fixed phase angle. The reliability of the measurements was ensured by taking 100 readings per angle i.e. each data point was an average of 100 separate measurements.

3.3.6 Reflection correction for hydrosol sample module

While using the experimental data from static light scattering experiments on suspended particles, one must take into account the secondary reflected noise from the inner wall of the cylindrical glass cuvette because neglecting this effect may lead to false interpretation of the size or refractive index of the scatterers [262]. Incident laser light is scattered in all directions by the suspended scatterers and therefore the intensity measured by the detectors is composed of the actual scattered intensity at scattering angle θ along with the secondary scattering intensities from the inner wall of the glass cuvette at scattering angle $\theta' = 180 - \theta$. The contribution of this reflectivity can be eliminated by introducing a semi-cylindrical blackened screen inside the cuvette. However the strong

intensity of the direct incident light gets reflected at the inner wall of the cuvette and follows its trace back and is again scattered along the path as shown in figure 3.22. In order to eliminate this noise we used the method described by Volten et.al. [63] in which the correction function for this error was given by,

$$S_{11}(\theta) = S_{11}^{uncor}(\theta) - S_{11}^{uncor}(180-\theta)R \qquad \textbf{3.4.1}$$

$$S_{12}(\theta) = S_{12}^{uncor}(\theta) - S_{12}^{uncor}(180-\theta)R \qquad \textbf{3.4.2}$$

where $S_{11}^{uncor}(\theta)$ and $S_{12}^{uncor}(\theta)$ are the measurements uncorrected for reflections, $S_{11}(\theta)$ and $S_{12}(\theta)$ are the measurements corrected for reflections and R (=0.017) was the reflection coefficient for Pyrex glass.

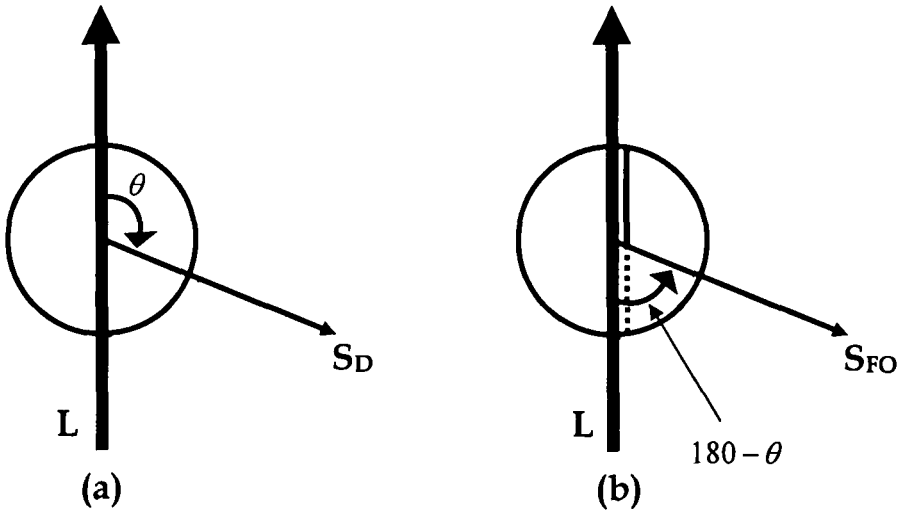

Figure 3.22. Pictorial representation of the (a) direct scattering by the hydrosols and (b) secondary scattering due to the first order reflection of the direct incident light at the inner wall of the cuvette. L: incident laser beam; θ: scattering angle; S_D: direct scattering; S_{FO}: first order reflection-scattering.

The pictorial representation of the data reduction process for the phase function and degree of linear polarization of polystyrene spheres at 543.5 nm are shown in figure 3.23(a) and figure 3.23(b). The same procedure of error correction was applied for the other incident wavelengths.

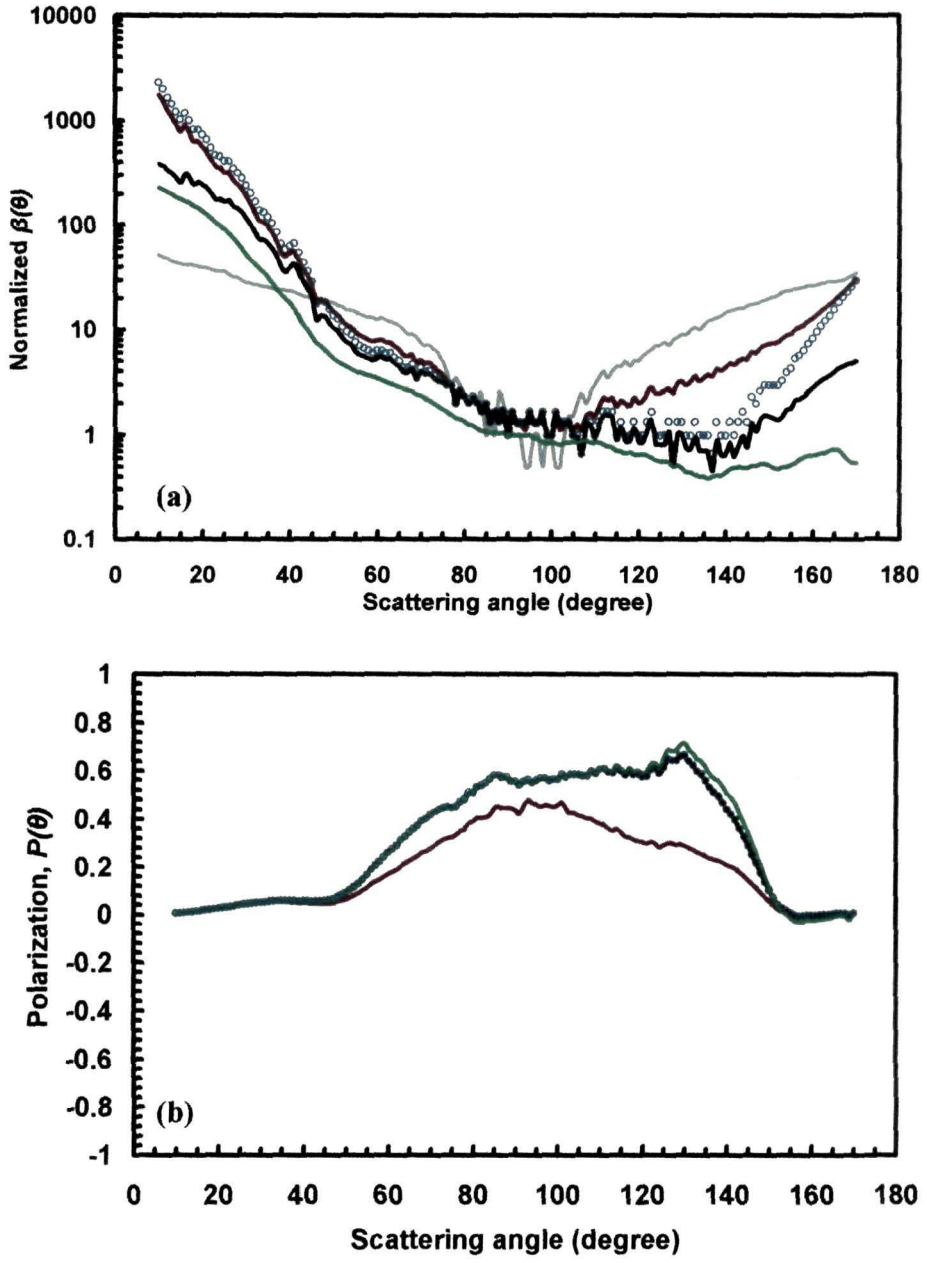

Figure 3.23. The data reduction procedure for volume scattering function, $\beta(\theta)$ and degree of linear polarization, $P(\theta)$ are shown in figure 3.23(a) and figure 3.23(b) respectively for Polystyrene spheres at 543.5 nm incident wavelength. Plot for background scattering is shown by grey line. The plot for raw uncorrected data is shown by brown line. Background correction is shown by blue circles. Plot after the correction for changing scattering volume is denoted by black line whereas the final result after the correction for reflection at the walls of the cuvette is shown by green line.

3.4. Development of the analytical software - Tezpur University Scattering Software (TUSCAT)

This section of the thesis describes the details of the development of an interactive software package TUSCAT (Tezpur University SCATtering Software) integrated with a graphical user interface (GUI) for modeling electromagnetic scattering from virtual small particles and also to yield characteristic properties of real particles from experimental data. Its interactive features enable the user to observe the changes in output scattering properties in real time. In addition to its ease of use, it has high computational accuracy, efficiency, reliability and adaptability. The package uses and involves a user friendly GUI in order to enable the users to enter the required input parameters for light scattering calculations and observe the results more intuitively. The numerical results of the scattering matrix elements and the efficiencies can also be saved in a user defined data file. The computational programs behind TUSCAT are based on Mie theory for spherical particles and T-matrix theory for nonspherical particles (cylindrical and spheroids) described in detail in Chapter II. The software was designed to compare experimental results from some unknown particle with theoretical results so as to provide an analytical tool for light scattering experiments from monodisperse and polydisperse particles conducted by using the designed and fabricated light scattering setup.

3.4.1. Description of TUSCAT

TUSCAT uses a user friendly graphical user interface (GUI) programmed using Java Swing in J2SE1.5 platform [369, 370]. Java Swing is a widget toolkit for Java, part of Sun Microsystems' Java Foundation Classes (JFC) and an application programming interface (API) which offers much more functionality and sophisticated set of GUI components than a collection of standard widgets. The reason for selecting java swing in designing the software is its inherent advantages such as java swing is platform independent both in terms of its

expression and its implementation (non-native universal rendering of widgets), a highly extensible and partitioned architecture and has a rich set of useful GUI components. The swing components are built around the Model-View-Controller (MVC) programming paradigm which conceptually decouples an application's business data logic from its user interface so that they can be evolved independently. Also the unique pluggable look and feel architecture of java swing allows a program to have control over its appearance. In developing the software, these properties of java swing were extensively used [50, 51].

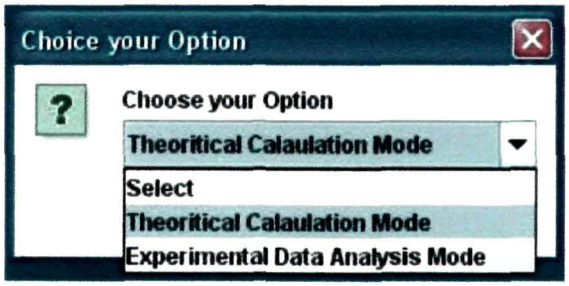

Figure 3.24. Screenshot of the initial selection menu TUSCAT

Description of the graphical user interface (GUI)

The software was designed to operate in two modes for theoretical calculation and experimental data analysis. The mode of operation of the software can be selected by using the initial selection (drop down) menu of TUSCAT as shown in figure 3.24.

Figure 3.25 shows the control panel of the GUI when calculations were done for an ensemble of spherical particles having refractive index 1.65+i*0.001 with normal size distribution (minimum particle radius = 0.5 μm; maximum particle radius = 1.2 μm; modal radius = 0.8 μm and σ_g = 2) at 0.750 μm incident wavelength. The medium refractive index was taken to be 1.0+i*0.0. At the top of the window, a menu bar with two buttons namely 'Info' and 'Help' is placed. The first button (Info) is used to give initial information about the software and

the second one (Help) can be used to open the help file. The user can switch to the Experimental data analysis mode by clicking in the button "Switch to Experimental Data Analysis Mode". As shown in the figure 3.25, below the 'switch button 1', the input box 1 is placed where the user can give the

Figure 3.25. Screenshot of the control panel the 'Theoretical Calculation Mode' of TUSCAT.

real and imaginary part of the refractive index of the particle (default value = 1.00 + i × 0.00). The wavelength of the incident radiation in micrometers (default value = 1.00 μm) can be given in the input box 2. In the particle geometry box (input box 3) the user has to select the shape under consideration in the drop

down menu. In the present version of the software, it is capable of considering three shapes - sphere, cylinder and spheroid. If the user selects sphere for the light scattering calculations, the software automatically selects the computer code based on Mie theory for the light scattering calculations. For the nonspherical shapes cylinder and spheroid the software selects the T-matrix code for further calculations. This T-matrix code is adapted from the version of Mishchenko's 'tmd.new.f' code [5, 238] which is freely available in the NASA website (ftp://ftp.giss.nasa.gov/pub/crmim/tmd. new.f). When the shapes, cylinder or spheroid shapes are selected, the user has to enter the ratio A/B of horizontal (A) to rotational (B) semi-axes (for spheroid) or the diameter (C) to length (L) ratio, C/L (for cylinder) respectively. The default value of both A/B and C/L is 1.00. Also for such nonspherical geometries, one has to specify the desired accuracy of T-matrix computation. One more drop down menu is provided in the input box 4 for selecting the desired size distribution from four options - monodisperse, gamma, normal and log-normal. For monodisperse particles, the user has to give only the radius or equivalent radius of the scatterers whereas for gamma, normal and lognormal distribution, the user has to give lowest and highest particle radius, σ_g (for normal and lognormal distribution) and α (for gamma distribution), and the modal radius of the particles. The theoretical calculations start when the 'Calculate' button is clicked and similarly when 'Show Theoretical Plot' button is clicked the software generates plots for different non zero elements of the scattering matrix and displays in a separate window. The result can be stored in a user defined file when the button 'Save Theoretical Data' is clicked. On the bottom right hand side of the GUI, the calculated values of extinction, scattering, absorption and backscattering efficiencies, single scattering albedo, asymmetry parameter and radiation pressure are displayed in their respective output panels.

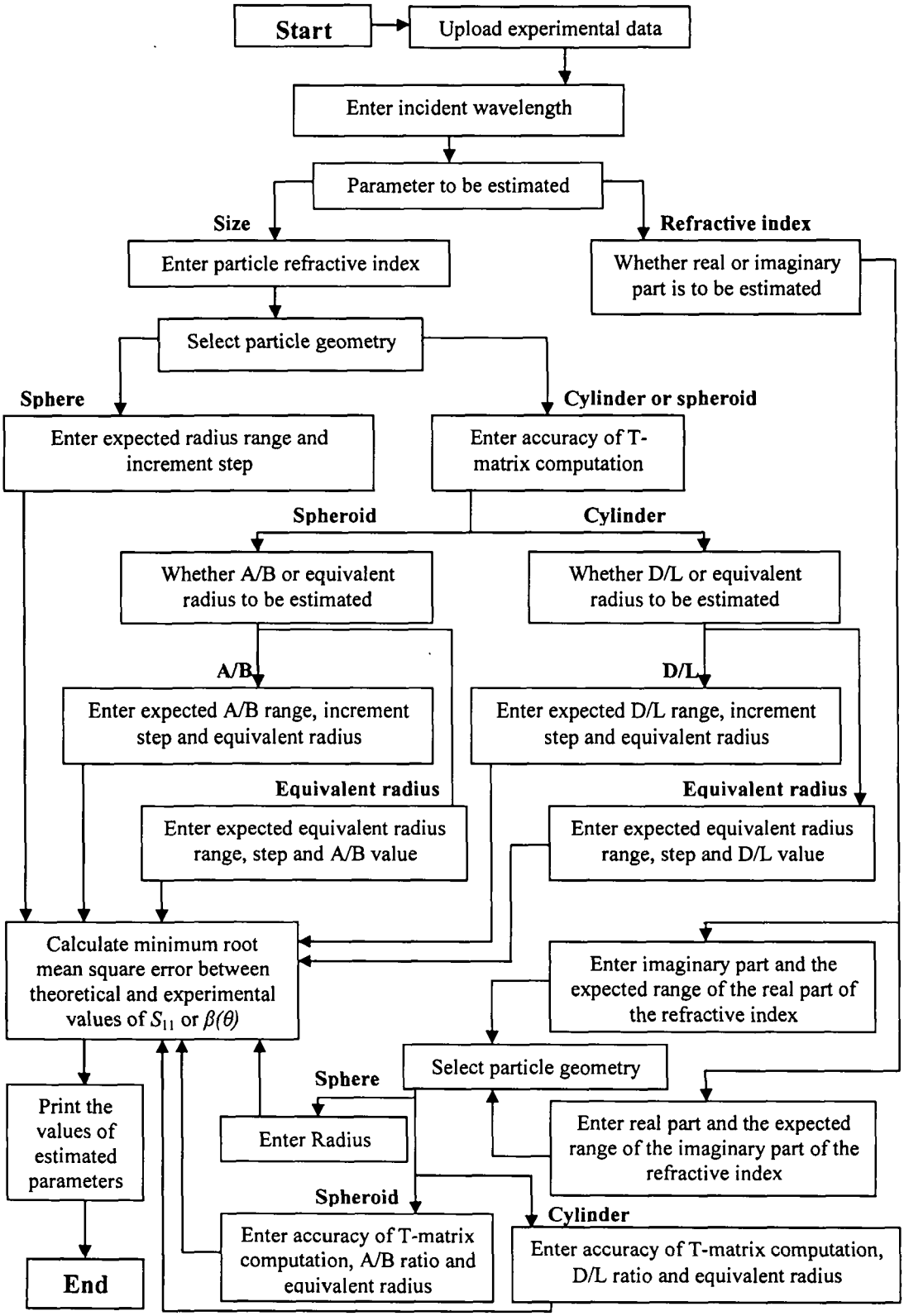

Figure 3.26. Flowchart of "Experimental Data Analysis Mode" of TUSCAT.

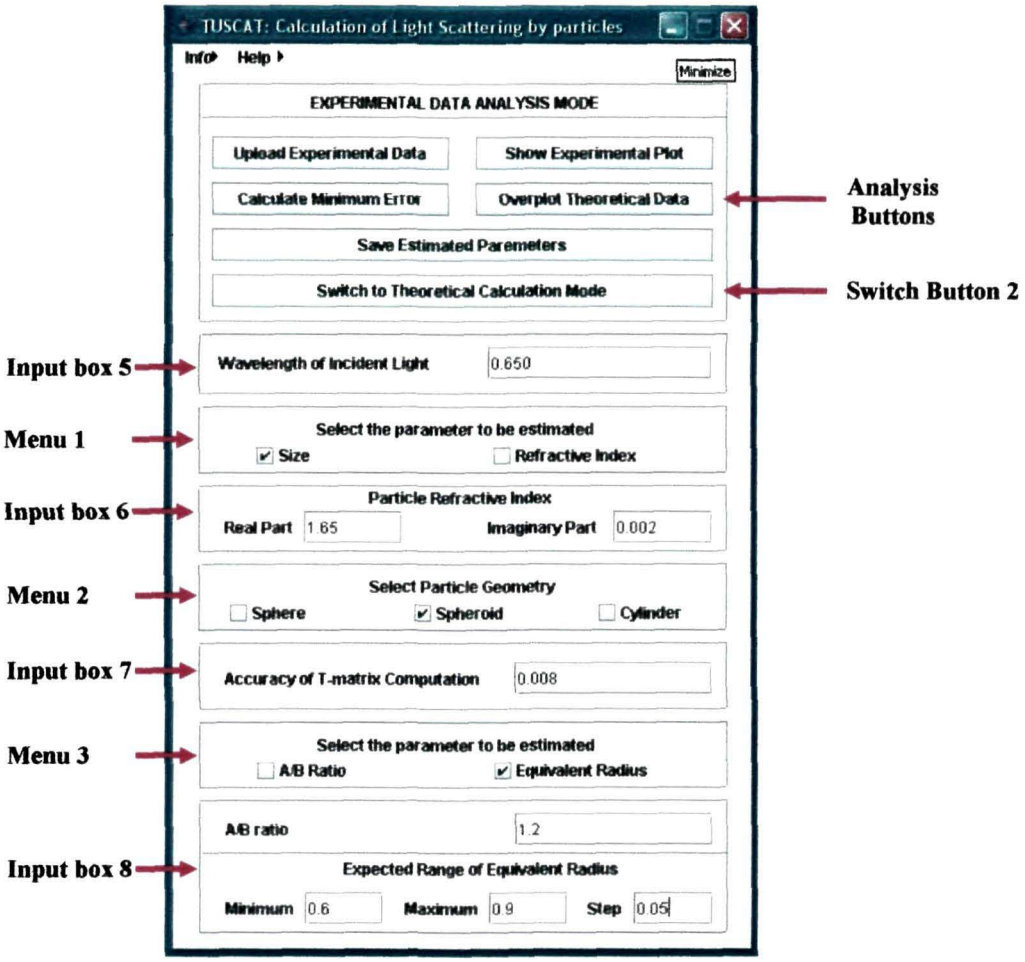

Figure 3.27. Screenshot of the control panel in the 'Experimental Data Analysis Mode' of TUSCAT for size estimation.

Selection of 'Experimental Data Analysis Mode' in the initial drop down menu (figure 3.24), enables the user to upload experimental data, plot graphs for the non zero elements of the scattering matrix and compare the experimental data from an unknown scatterer with the theoretical data generated by varying the input parameters, to estimate the size or refractive index of that scatterer. Figure 3.27 shows the screenshot of the control panel of the GUI for this mode of operation. At the top left of this panel a group of three buttons are placed. The first button 'Upload Experimental Data' can be used to upload the experimental data to be analyzed while the second button 'Show Experimental Plot' enables

the user to observe the experimental plots for S_{11}, $-S_{12}/S_{11}$, S_{33}/S_{11} and S_{34}/S_{11} (both for spherical and non-spherical particles) which are then displayed in a separate plotting window. The third button 'Calculate Minimum Error' is used to estimate the required parameter (size or refractive index) by automated running of the code with the input parameters being successively increased to find the most suitable match in terms of minimum root-mean-square (rms) error of the theoretical and calculated values of S_{11} or $\beta(\theta)$. The fourth button 'Overplot Theoretical Data' can be used to display the superimposed plot of the experimental results and the theoretical results calculated by using the estimated parameters. The estimated parameters can be saved in a user defined data file by clicking the 'Save Estimated Parameters' button. The user can switch to the theoretical calculation mode by clicking the button 'Switch to Theoretical Calculation Mode'. Notably, in all the inverse calculations the environment refractive index was set to 1.0+i*0.0. The parameters that are varied are particle radius (for spherical particles), axial or diameter to length ratio (for spheroids and cylinders), volume equivalent radius (for spheroids and cylinders), real part of the refractive index and imaginary part of the refractive index from the given range of values for the parameter. The values of the other parameters need to be given as input in their respective input boxes. In the first input box (input box 5) of this panel, the user can specify the incident wavelength. Below input box 5, 'menu 1' is placed for selecting the desired parameter to be estimated (radius or refractive index). If radius is selected the user has to give the refractive index of the partcle in the 'input box 6' and then select the particle geometry (sphere, cylinder or spheroid) in menu 2. For spherical particles the user has to give the range of radii of the scatterer within which the most suitable match between the experimental and theoretical values of S_{11} can be expected. For non spherical particles the accuracy of T-matrix computation has to be given in the 'input box 7'. One can further choose to estimate axial ratio, A/B for spheroids, diameter to length ratio, D/L for cylinders or equivalent radius, 'menu 3'. In any of the cases

(whether A/B or D/L or equivalent radius is selected), the user has to give the range of values of the respective parameters within which the most suitable match is expected, in 'input box 8'.

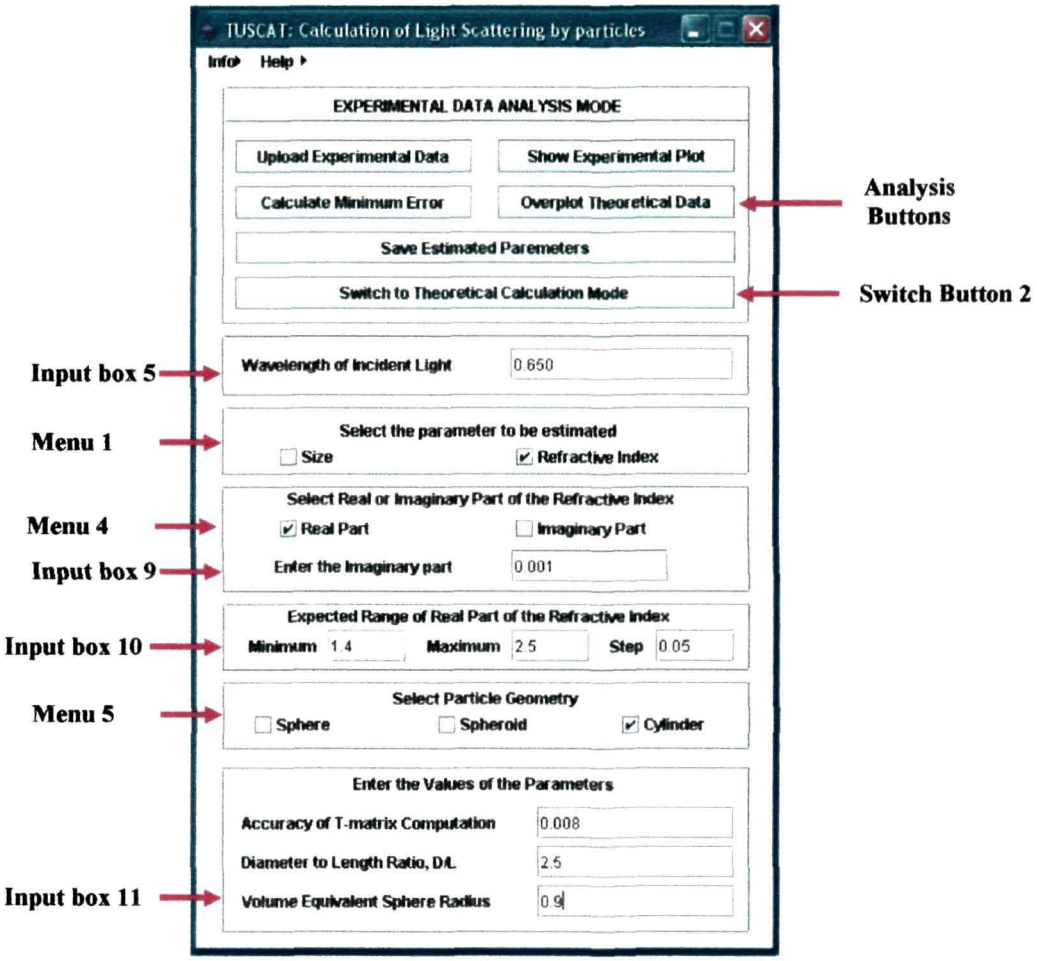

Figure 3.28. Screenshot of the control panel the 'Experimental Data Analysis Mode' of TUSCAT for refractive index estimation.

On the other hand if refractive index is selected in 'menu 1', the user has to choose real part or imaginary part of the refractive index to be estimated in 'menu 4'. The user has to input the respective value of real or imaginary part of the refractive index in the 'input box 9'. In both the cases, the user has to give the suitable range of refractive index values in 'input box 10'. The particle geometry

can be selected in 'menu 5'. The values of the radius, axial ratio, A/B for spheroids, diameter to length ratio, D/L for cylinders and equivalent radius respective to the particle geometry is to be given in 'input box 11'. As already mentioned the inverse calculation to find the minimum rms error between the experimental and theoretical data for the respective input parameters start when the 'Calculate Minimum Error' button is clicked. A message will be displayed with the minimum error and resultant parameters after the successful completion of the calculation. However an error message may come especially in case of spheroids and cylinders if the input parameter is beyond the computational capability of the software.

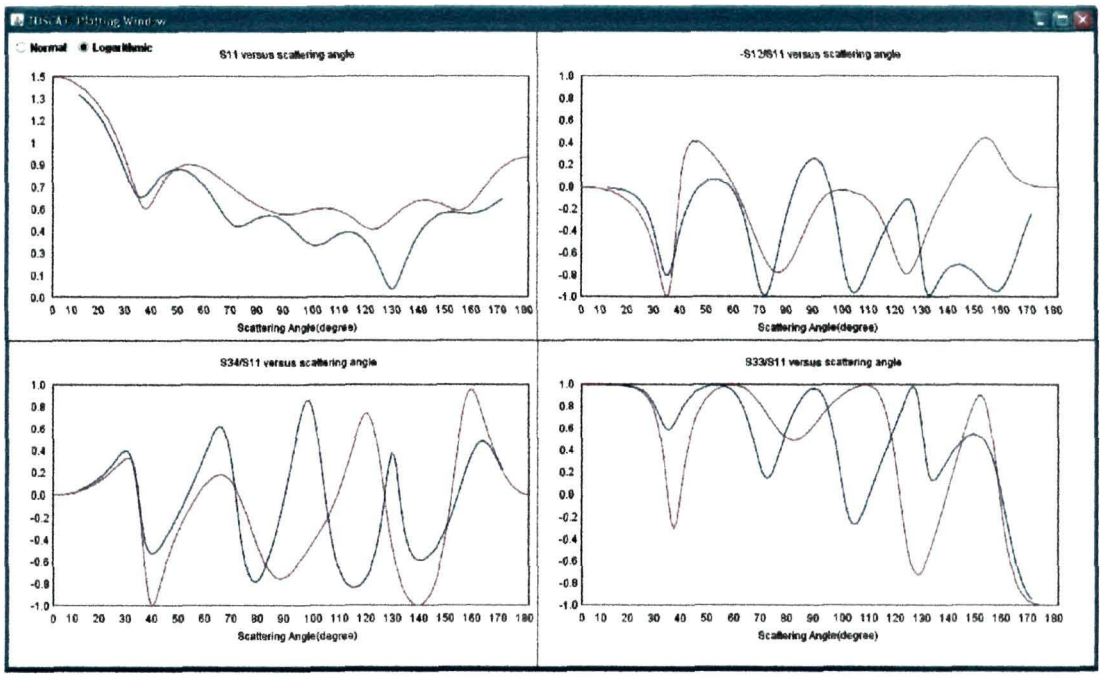

Figure 3.29. Screenshot of the plotting window of TUSCAT showing the superimposed plots of the experimental (blue solid line) and theoretical (red solid line) results.

The plotting window of TUSCAT (figure 3.29) is capable of plotting graphs for S_{11}, $-S_{12}/S_{11}$, S_{33}/S_{11} and S_{34}/S_{11} as functions of scattering angle. Option for plotting logarithmic graphs for S_{11} is also provided at the plotting

Important features

Some of the important features of TUSCAT are that it runs under Windows, Linux and Solaris operation systems and there are no special hardware requirements. It can calculate and display the results for extinction, scattering and absorption efficiencies, single scattering albedo and asymmetry parameter. It can calculate the nonzero elements of the scattering matrix and plot graphs for S_{11}, $-S_{12}/S_{11}$, S_{33}/S_{11} and S_{34}/S_{11} as functions of scattering angle. It has option for saving the theoretical results in a user defined data file. Moreover, it can upload experimental data files in '.txt', '.dat' or '.csv' format, containing the values of scattering angle, S_{11}, $-S_{12}/S_{11}$, S_{33}/S_{11} and S_{34}/S_{11} serially separated by a comma and plot graphs for them as functions of scattering angle. The experimental and theoretical values of S_{11} can be compared by varying the input parameters, to estimate the characteristic properties (size or refractive index) of that particle responsible for the scattering results.

Limitations

For spherical particles, TUSCAT takes processing time of less than a second for monodisperse particles and 2 - 3 seconds for polydisperse particles. The software was found to be stable even for very large spherical particles (size parameters > 1000) with large values of real and imaginary part of the refractive index. In case of spheroids and cylinders the average computation time of TUSCAT was 4 - 6 seconds for monodisperse particles and 20 - 65 seconds for polydisperse particles. However as in the case of Mishchenko's fortran code 'tmd.new.f', the capabilities of the T-matrix code in TUSCAT for computations on spheroids and circular cylinders is limited to only weakly absorbing and rotationally symmetric spheroids and finite cylinders of sizes comparable to the wavelength of the laser beam. These limitations can be overcome by implementing other methods of matrix conversion e.g. Gaussian elimination with back substitution [358] and standard Java language APIs (Application

programming interfaces) for LAPACK (Linear Algebra PACKage) [371] or JLAPACK package [372]. In case of the 'Experimental Data Analysis Mode' the software typically takes 1 - 30 seconds for spherical particles and 10 - 90 seconds for nonspherical particles which in turn depends on the input range of parameters. Moreover the software can be applied to particles dispersed in non-absorbing medium only.

3.4.2. Validation of the software

In addition to the friendliness of the GUI and the ease of use, TUSCAT was tested several times to optimize its computational accuracy and reliability. The results for scattering efficiencies, radiation pressure, single scattering albedo, asymmetry parameter and the scattering matrix elements for different input parameters agree qualitatively well with the available computer programs and sources in the literature [5, 124, 238, 263]. The results of extinction and scattering efficiencies for different combinations of particle refractive index, and size parameter for monodisperse spherical particles were compared with the test results obtained from Wiscombe's fortran program MIEV0 [263] and is shown in Table 3.6. The environment refractive index was taken to be 1.0+i*0.0. It was found from the table that the results for the extinction efficiency, scattering efficiency and asymmetry parameter agree exactly upto six significant digits.

Table 3.6. Comparison of extinction efficiency (Q_{ext}), scattering efficiency (Q_{sca}) and asymmetry parameter as calculated by MIEV0 and TUSCAT at varying refractive index and size parameters for spherical particles.

Refractive index	Size parameter	Q_{ext}		Q_{sca}		Assymatry parameter	
		MIEV0.f	TUSCAT	MIEV0.f	TUSCAT	MIEV0.f	TUSCAT
0.75+0.0*i	10	2.232265	2.232265	2.232265	2.232265	0.896473	0.896473
0.75+0.0*i	1000	1.997908	1.997908	1.997908	1.997908	0.844944	0.844944
1.5+1.0*i	1	2.336321	2.336321	0.663454	0.663454	0.192136	0.192136
1.5+1.0*i	100	2.097502	2.097502	1.283697	1.283697	0.850252	0.850252
1.5+1.0*i	10000	2.004368	2.004368	1.236574	1.236574	0.846310	0.846310
10+10*i	1	2.532993	2.532993	2.049405	2.049405	-0.110664	-0.110664
10+10*i	10000	2.005914	2.005914	1.795393	1.795393	0.548194	0.548194

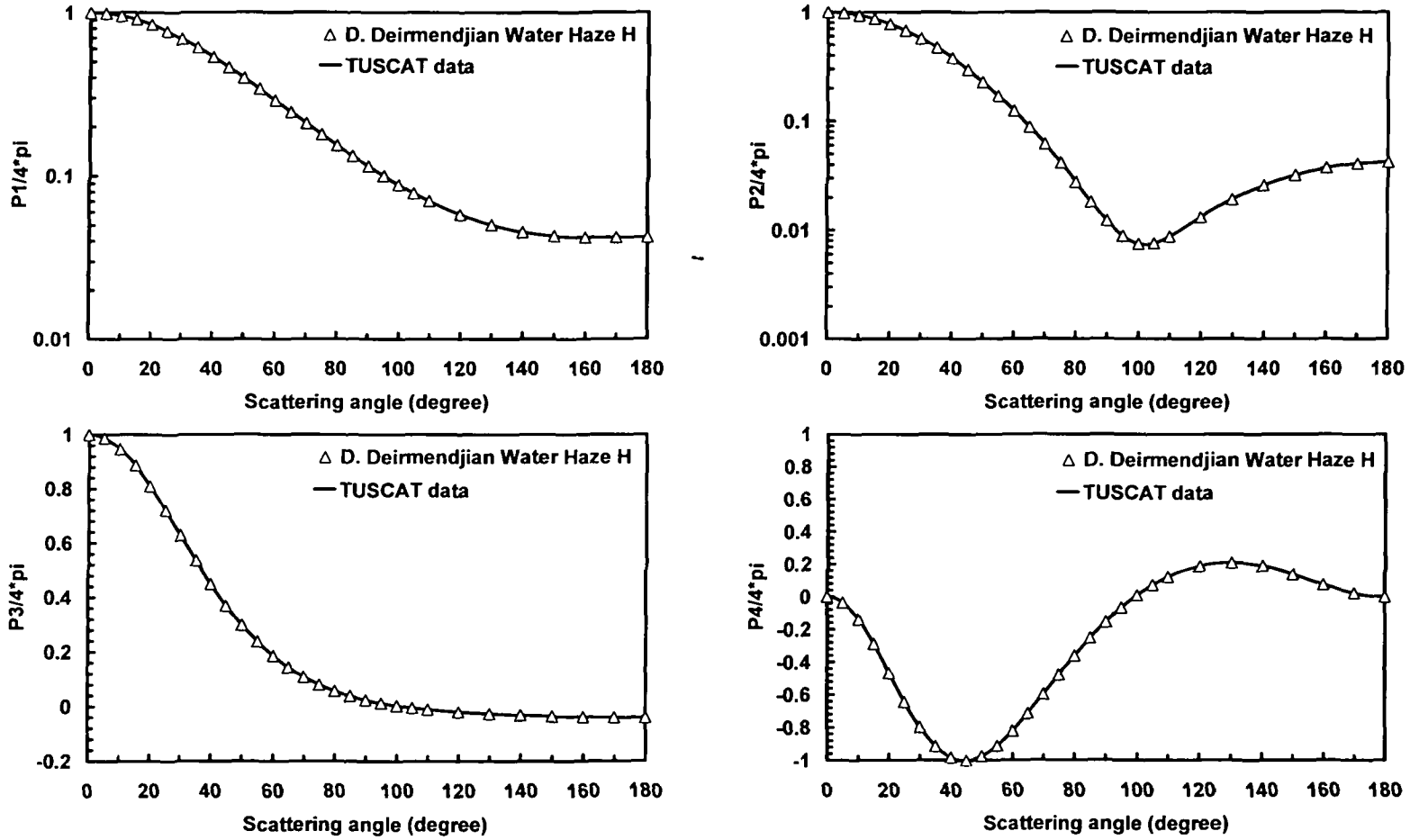

Figure 3.30. Normalized graphs of (a) $P_1/4\pi$, (b) $P_2/4\pi$, (c) $P_3/4\pi$ and (d) $P_4/4\pi$ as a function of scattering angle for Water haze H (as described by D. Deirmendjian [41]) for an ensemble of gamma distributed spherical particles (Water haze H) having refractive index 1.322+i0.00001, total number of particles 100 cm^{-3}, modal radius 0.1 μm, alpha 2, gamma 1 at 1.19 μm incident wavelength are denoted by blank triangles. The graphs generated by using TUSCAT data are shown by solid black line.

For checking the accuracy of the software for polydisperse spherical particles, we used it to give the normalized values of $P_1(\theta)/4\pi$, $P_2(\theta)/4\pi$, $P_3(\theta)/4\pi$ and $P_4(\theta)/4\pi$ and compare the results with the normalized benchmark results for an ensemble of gamma distributed particles (Water haze H) of D. Deirmendjian [124] having refractive index 1.322+i0.00001, total number of particles 100 cm^{-3}, modal radius 0.1 μm, alpha 2, gamma 1 at 1.19 μm incident wavelength as shown in figure 3.30. It was observed that both the results tally within acceptable limits of deviation. These tests also ensured the accuracy and reliability of the C - language computer program TUmiescat.c described in the previous chapter.

Table 3.7. Comparison of extinction coefficient (C_{ext}), scattering coefficient (C_{sca}), single scattering albedo and asymmetry parameter as calculated by tmd.new.f and TUSCAT for monodisperse, gamma and lognormal distributed circular cylinders with volume equivalent minimum radius r_{min} = 0.1 μm and maximum radius r_{max} = 1.1 μm. The common value of alfa (gamma distribution) and σ_g^2 (lognormal distribution) was taken to be 0.2.

Scattering parameters		Monodisperse	Gamma	Lognormal
C_{ext}	tmd.new.f	5.586550	2.33327	4.446570
	TUSCAT	5.586551	2.333269	4.446570
C_{sca}	tmd.new.f	5.552520	2.31275	4.396060
	TUSCAT	5.525195	2.312749	4.396060
Single scattering albedo	tmd.new.f	0.989017	0.991206	0.988640
	TUSCAT	0.989017	0.991206	0.988640
asymmetry parameter	tmd.new.f	0.660263	0.666418	0.657400
	TUSCAT	0.660263	0.666418	0.657400

Again, to validate the accuracy of the T-matrix code running behind TUSCAT for nonspherical particles, the sample results for circular cylinders with monodisperse, gamma and lognormal distribution was compared with the results generated by using the program 'tmd.new.f' developed by M. I. Mishchenko [5, 238] at 0.750 μm incident wavelength. The refractive index and the axial ratio of the cylinders were considered to be 1.65+0.001×i and 2.5 respectively. The volume equivalent radius for monodisperse particles and the volume equivalent modal radius for gamma and lognormal distributed particles were considered to be same i.e. 0.8 μm. In this case also the software generated values of extinction coefficient, scattering coefficient, single scattering albedo and asymmetry parameter agree with those generated by using Mishchenko's code within acceptable limits of accuracy as shown in table 3.7. These preliminary tests ensured the efficiency and reliability of both the C - language computer program TUTscat.c described in the previous chapter and TUSCAT.

The software was developed to analyze the experimental results of the scattering properties of different types of aerosols and hydrosols having different size distributions by comparing with the theoretically generated data. The simplicity and the ease of use of the GUI associated with the software enables the user to visualize the effect of changing input parameters on the resulting scattering patterns in near real time. TUSCAT can be used to not only investigate the theoretical scattering properties of different types of aerosols and hydrosols having different size distribution but also couple these results with experimental results to find some unknown parameters (such as size, refractive index etc.) of the experimental samples. It is expected that the software will be helpful for performance optimization of new light scattering instruments. The source code of TUSCAT is given in APPENDIX E.

Thus this chapter was focused at developing a laboratory light scattering setup for use in the investigations carried out on aerosols (graphite, icelike particles), hydrosols (polystyrene, titania and diatoms - a biological microorganism), nanoparticles (ZnS) and clay particles (bentonite) which are to

be discussed in the next chapter. In the same chapter we described the development of a software TUSCAT for modeling light scattering properties of small spherical and nonspherical (spheroidal and cylindrical) particles and for the analysis of the experimental results from some unknown scatterer. The software was used for the analysis of the experimental results presented in the next chapter.

Chapter IV: Investigations and Results with the Light Scattering Setup

4.1 General Introduction

In this chapter, Section 4.2 contains details about the experimental validation and calibration of the light scattering setup by making rigorous experiments with spherical water droplets and polystyrene particles. Section 4.3 describes the details of the measurements on two types of particles, titania and diatoms which were suspended in water. Investigations carried out on ZnS nanoparticles and bentonite clay particles embedded in both rectangular and cylindrical polymer matrix are presented in section 4.4. Section 4.5 presents the measurement on powder samples (ice analogue crystals and graphite) sprayed by using a nebulizer. Error calculation is presented in section 4.6. Finally section 4.7 discusses the salient features of the light scattering setup. The chapter as a whole describes the details of the measurements and discusses the obtained results in the context of similar work done by other authors in an attempt to make a comparative assessment of the designed and fabricated light scattering setup.

Aggregation (joining together to form larger particles with time) and separation (breaking up into smaller particles) was not observed in case of aerosol, hydrosol and nanoparticle samples that were used in our experiments as verified by scanning electron microscopy (SEM) images of fresh and used samples. For every investigation the dust removal process was conducted as described in the previous chapter to remove the unwanted dust particles inside the simulation chamber. The data was stored in a computer in user defined files and subsequently processed using the data reduction methods as mentioned in the previous chapter.

4.2 Experimental validation of the setup

In order to calibrate and to test the performance and accuracy of the light scattering setup, experiments with perfectly spherical particles was conducted and the experimental data was compared with Mie calculations. In this work, extensive experiments on water droplets and polystyrene spheres suspended in double distilled water were performed with laser beams of wavelengths at 543.5 nm, 594.5 nm and 632.8 nm as incident light and the results obtained were compared with Mie calculations.

4.2.1. Experiment on water droplets

Water droplets are commonly used as standard sample for calibrating light scattering instruments [33, 368]. In this work, water droplets were produced by using a commercial neblulizer (Nuneb Pro, MRK Healthcare) [363] described in section 3.2.3.1. As the exact size distribution of the water droplets was not known, after the light scattering measurements, the experimentally obtained results were analyzed by using TUSCAT to find the modal radius of the size distribution. It is noteworthy to mention that the software TUSCAT generates the theoretical values of $S_{11}(\theta)$ which are equivalent to $\beta(\theta)$ in case of polydisperse particles as discussed in section 2.2.3. For Mie calculations a gamma size distribution was assumed for the water droplets produced by the nebulizer as in many cases the distribution of water droplets can be well represented by gamma distribution [1, 12, 215]. The best fit for the experimental results were obtained for a modal radius value of 2 μm within the size range 0.5 - 5.0 μm. The relevant parameters for the Mie calculations are given in table 4.1.

Figure 4.1(a), figure 4.1(c) and figure 4.1(e) show comparative plots of experimental volume scattering functions of polystyrene spheres with Mie theory whereas figure 4.1(b), figure 4.1(d) and figure 4.1(f) show comparison of degree of linear polarization obtained experimentally with that given by Mie theory. As shown in figure 4.1(a) - figure 4.1(f), the results of the light scattering measurements agree excellently with the Mie calculations over the scattering

angle range 10° - 170° at all the three incident wavelengths. At scattering angles larger than 120°, small deviation from the theoretical curves were observed for the degree of linear polarization especially at 594.5 nm and 632.8 nm laser wavelengths. These minor discrepancies are due to the differences in the estimated and actual size distributions of the droplets and may be due to the slight deviation of some of the larger water droplets from perfectly spherical shape.

Table 4.1. Parameters for Mie calculation of water droplets.

PARAMETERS	VALUE
Size distribution	Gamma
Minimum particle radius (μm)	0.5
Maximum particle radius (μm)	5
Modal radius, r of the water droplets (μm)	2
Alfa, α	2
Particle refractive index at the probing wavelengths	1.33+ i0.00000
Incident light wavelengths in nanometers	543.5, 594.5 and 632.8

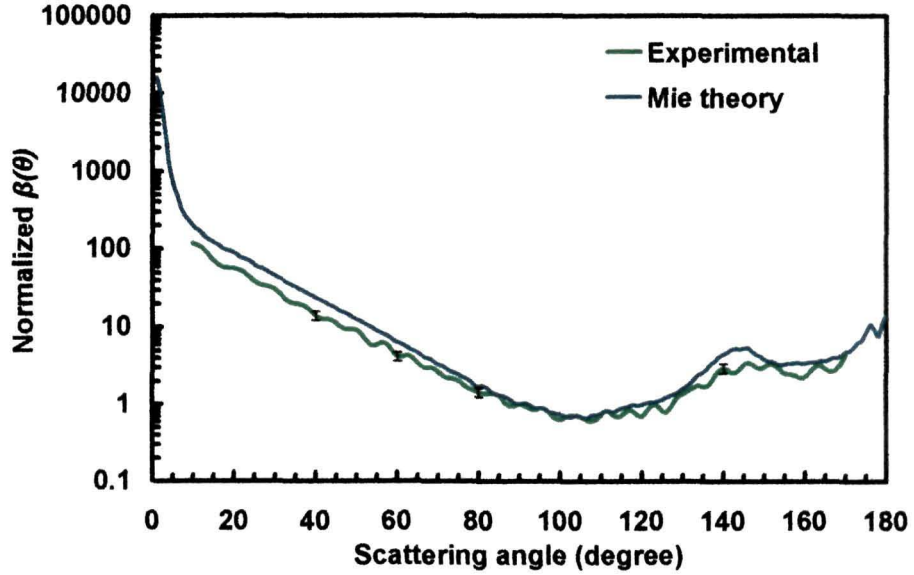

Figure 4.1(a). Measured volume scattering function, $\beta(\theta)$ of water droplets at 543.5 nm incident wavelength is denoted by green lines. The comparative Mie curve is denoted by blue line. Error bars in the plots indicate instrumental error.

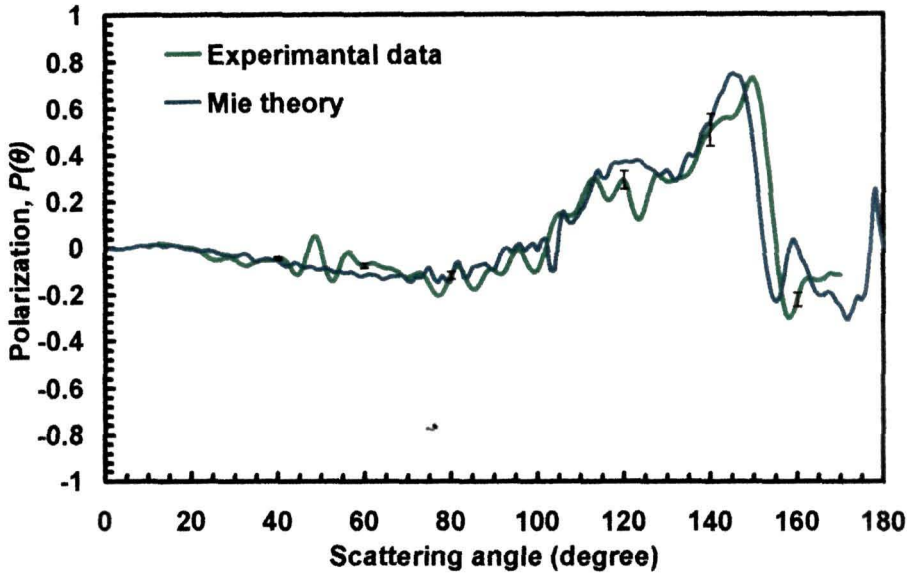

Figure 4.1(b). Measured degree of linear polarization, $P(\theta)$ of water droplets at 543.5 nm incident wavelength is denoted by green lines. The comparative Mie curve is denoted by blue line.

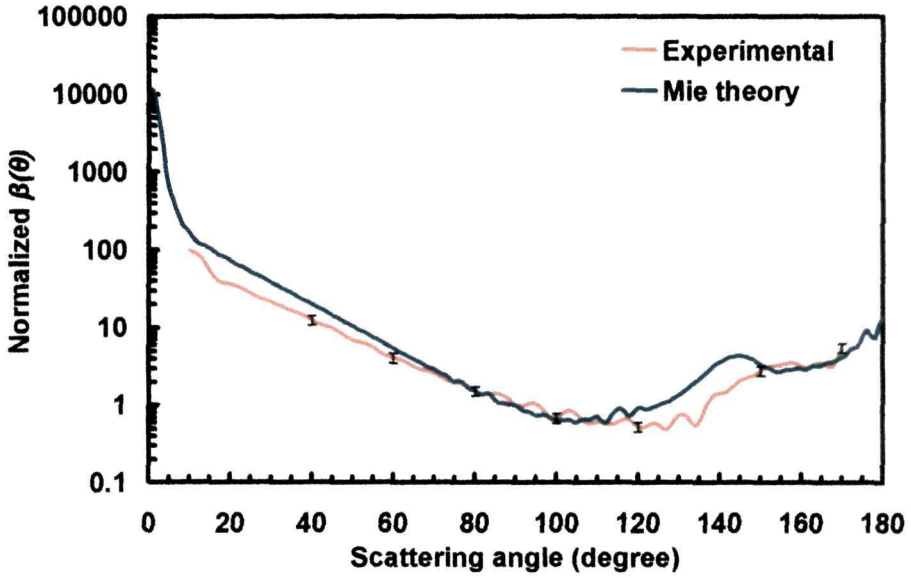

Figure 4.1(c). Measured volume scattering function, $\beta(\theta)$ of water droplets at 594.5 nm incident wavelength is denoted by orange lines. The comparative Mie curve is denoted by blue line.

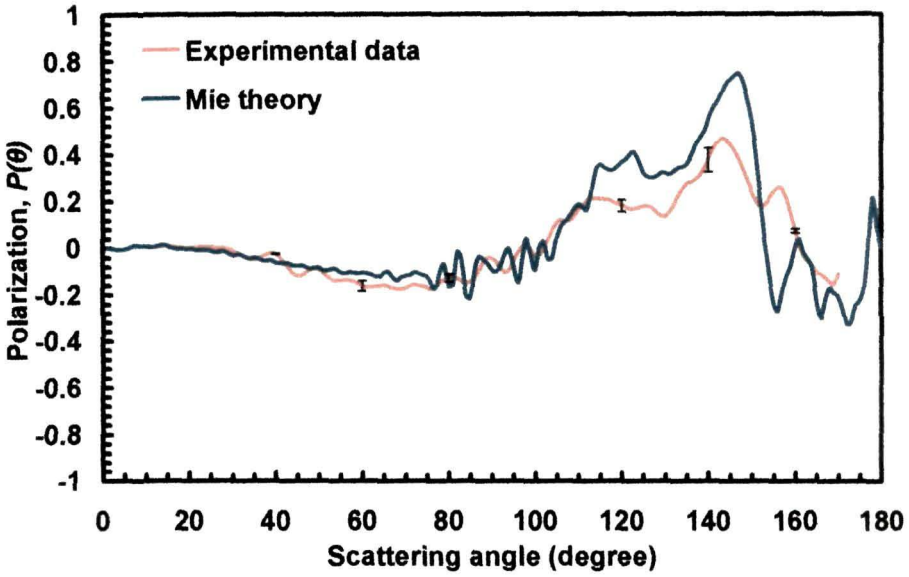

Figure 4.1(d). Measured degree of linear polarization, $P(\theta)$ of water droplets at 594.5 nm incident wavelength is denoted by orange lines. The comparative Mie curve is denoted by blue line.

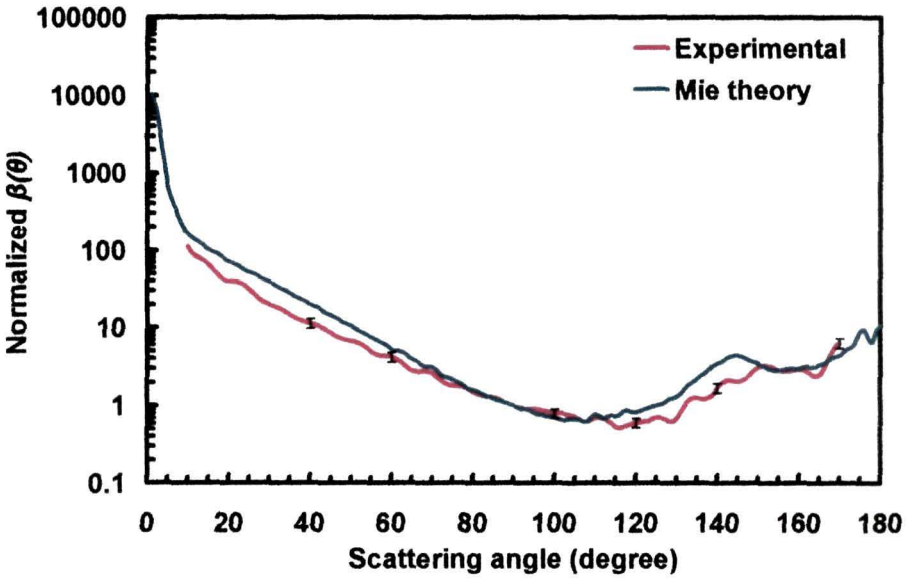

Figure 4.1(e). Measured volume scattering function, $\beta(\theta)$ of water droplets at 632.8 nm incident wavelength is denoted by red lines. The comparative Mie curve is denoted by blue line.

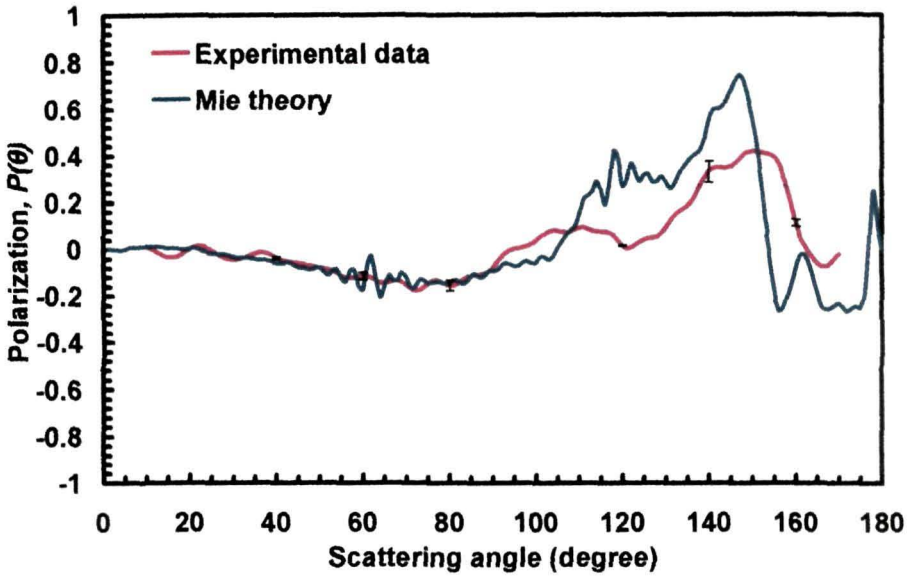

Figure 4.1(f). Measured degree of linear polarization, $P(\theta)$ of titania particles at 632.8 nm incident wavelength is denoted by red lines. The comparative Mie curve is denoted by blue line.

4.2.2. *Experiment on polystyrene samples*

Accuracy of the light scattering setup was also investigated by comparing the experimentally measured light scattering patterns of polystyrene microspheres suspended in double distilled water with Mie calculations. Figure 4.2 shows the scanning electron microscopy (SEM) image of the polystyrene particles used for this purpose. The size distribution of the particles was found to be Gaussian with a mean radius of 0.35 micron as shown in figure 4.3. Although the particles were found to be mostly spherical, a small number of particles were found to be nonspherical and in aggregate form also (one aggregated particle is shown within black circle).

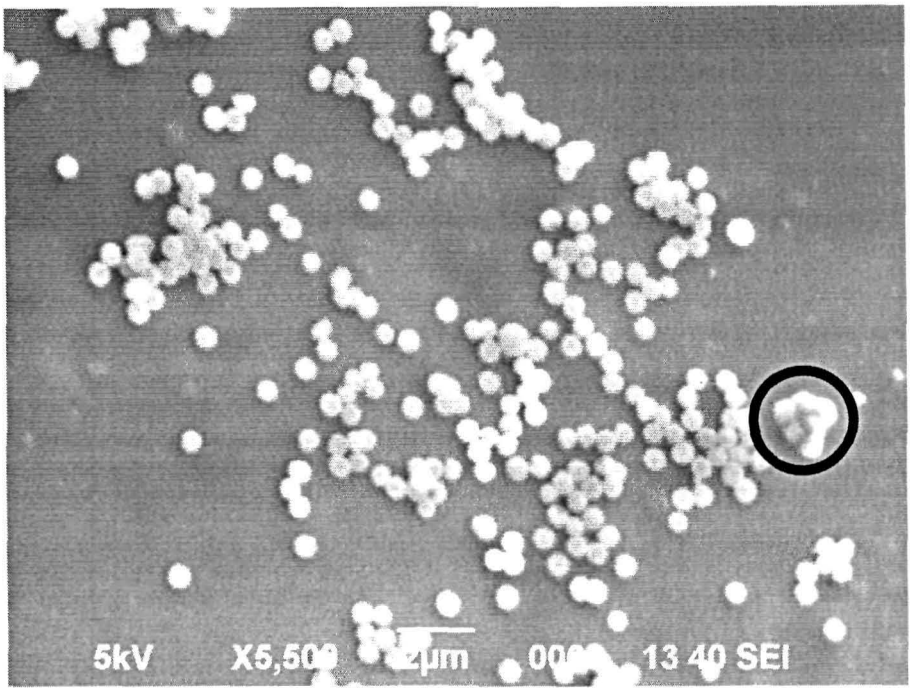

Figure 4.2. SEM image of polystyrene particles at 5500X resolution. An aggregated polystyrene particles is shown within black circle.

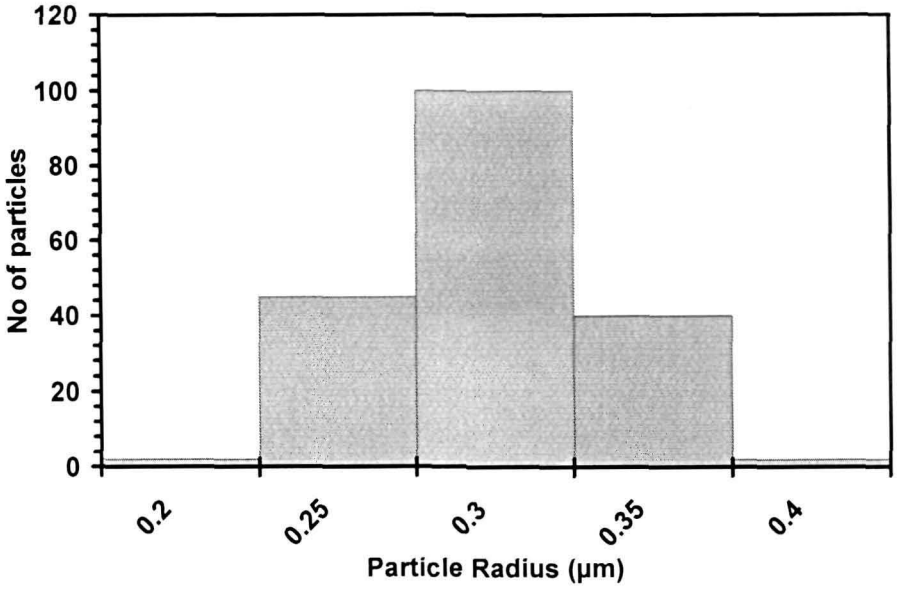

Figure 4.3. Measured size distribution of the polystyrene particles.

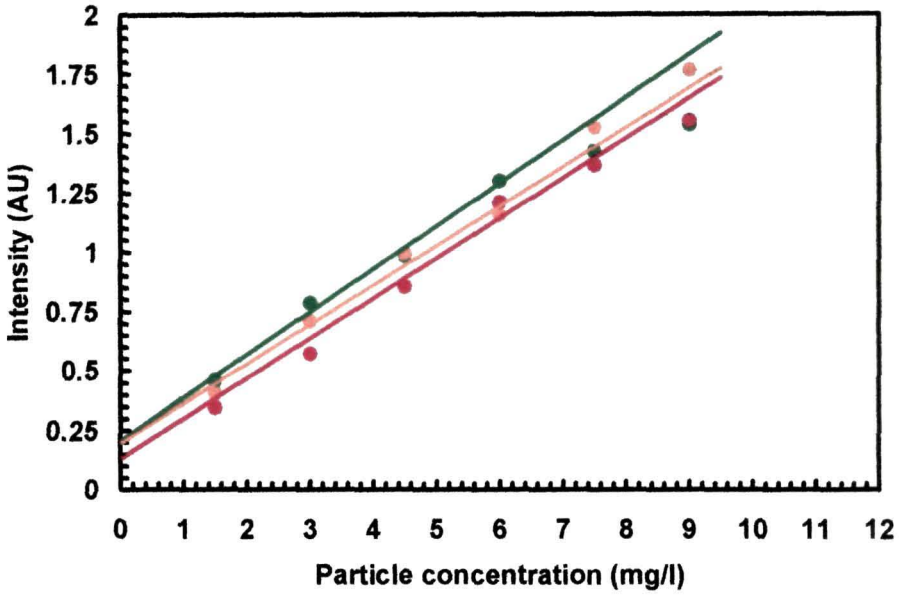

Figure 4.4. Graph for the scattered intensity for unpolarized incident light versus concentration of polystyrene for a fixed position of the detector at 10^0. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

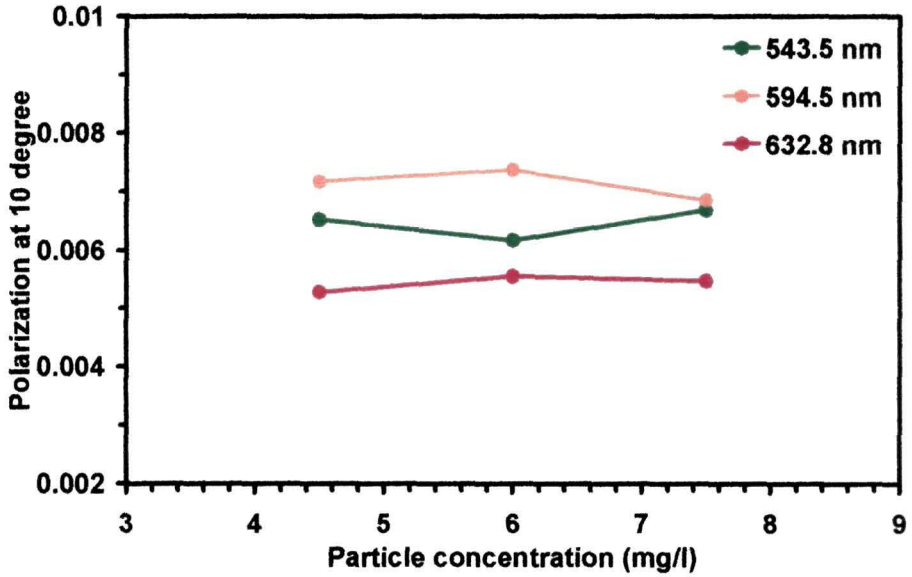

Figure 4.5. Graph for the measurement of the stability of measured polarization of polystyrene at increasing particle concentration within the single scattering regime.

To ensure that the measurements were in the single scattering regime, the intensity of scattered light for unpolarized incident light was measured as a function of increasing particle concentration at the smallest scattering angle used in the experiment i.e. 10^0. The optimum sample concentration for single scattering at the three incident wavelengths was found to be approximately 6 mg/l. Up to this concentration the graph between the scattered intensity versus sample concentration was linear as shown in figure 4.4. Again the stability of the degree of linear polarization at the single scattering regime was checked by using the method described in the section 3.3.5 of the previous chapter. It was found to be fairly stable at 10^0 as shown in figure 4.5.

Figure 4.6(a), figure 4.6(c) and figure 4.6(e) show comparative plots of experimental volume scattering functions of polystyrene spheres with Mie theory. Similarly figure 4.6(b), figure 4.6(d) and figure 4.6(f) show comparison of degree of linear polarization obtained experimentally with that given by Mie theory. For the Mie calculations, the dispersion formula as reported by other researchers [264, 265, 364] was used for determining the refractive index of polystyrene at the three different incident wavelengths. Small amount of oscillations (also known as Mie oscillations) were observed in the experimental scattering patterns as shown in the figures, indicating the presence of a narrow size distribution of the polystyrene particles in the scattering volume. A closer agreement of the measurements with the calculations was found for a normal (Gaussian) distribution of particles as mentioned in table 4.2. The estimated parameters for Mie calculations are given in table 4.2. The measured volume scattering functions, $\beta(\theta)$ and degree of linear polarization, $P(\theta)$ agree qualitatively well with the results drawn from Mie theory within acceptable limits of deviation proving the setup to be quite accurate for light scattering measurements. However, we can still find some deviation from the theory, especially for polarization, $P(\theta)$ which can be attributed to the presence of small aggregates (as shown within black circle in figure 4.2) and nonspherical polystyrene particles in the scattering volume. From the figures it was observed

that the polarization curves are bell shaped having positive values at most of the scattering angles

Table 4.2. Parameters for Mie calculation of Polystyrene spheres.

PARAMETERS	VALUE
Size distribution	Normal (Gaussian)
Variance, σ^2	1
Minimum particle radius (μm)	0.2
Maximum particle radius (μm)	0.45
Modal radius (μm)	0.3
Incident light wavelengths in nanometers	543.5, 594.5 and 632.8
Particle refractive index relative to water at 543.5 nm	1.193+i*0.0003
Particle refractive index relative to water at 594.5 nm	1.191+ i*0.0003
Particle refractive index relative to water at 632.8 nm	1.189+ i*0.0004

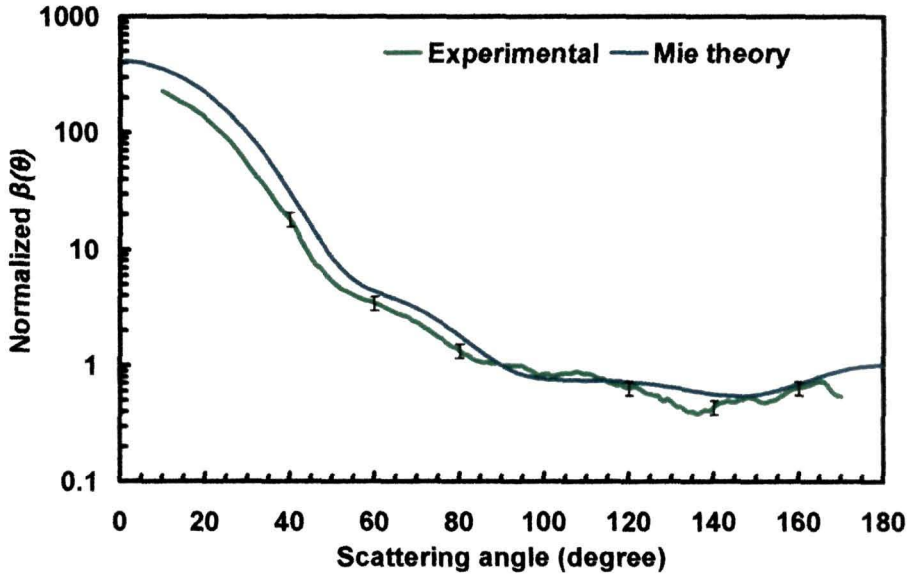

Figure 4.6(a). Measured volume scattering function, $\beta(\theta)$ of polystyrene particles at 543.5 nm incident wavelength is denoted by green lines. The comparative Mie curve is denoted by blue line. Error bars in the plots indicate instrumental error.

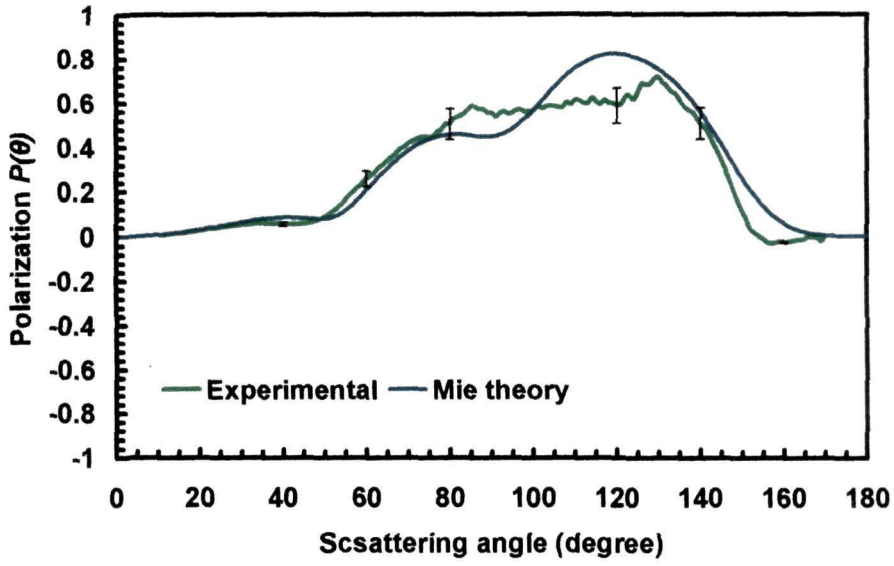

Figure 4.6(b). Measured degree of linear polarization, $P(\theta)$ of polystyrene particles at 543.5 nm incident wavelength is denoted by green lines. The comparative Mie curve is denoted by blue line.

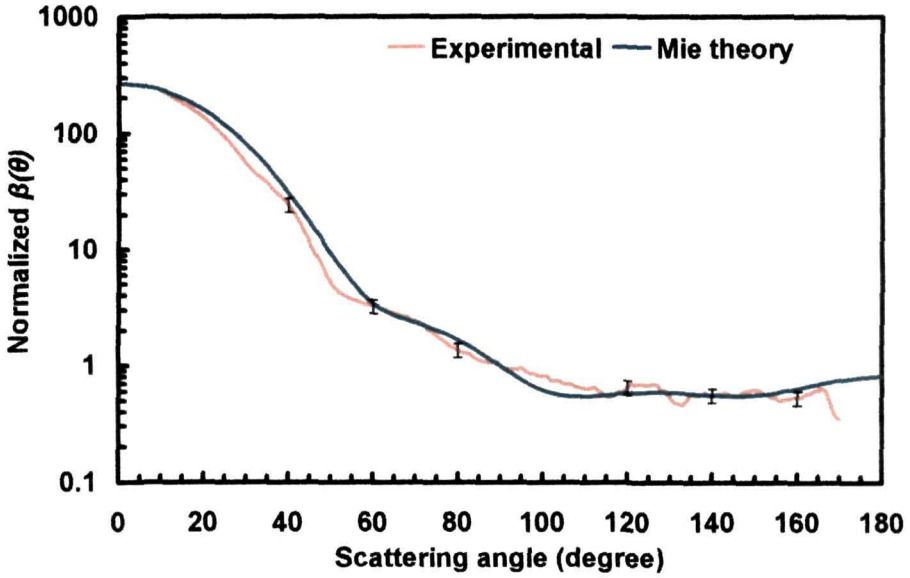

Figure 4.6(c). Measured volume scattering function, $\beta(\theta)$ of polystyrene particles at 594.5 nm incident wavelength is denoted by orange lines. The comparative Mie curve is denoted by blue line.

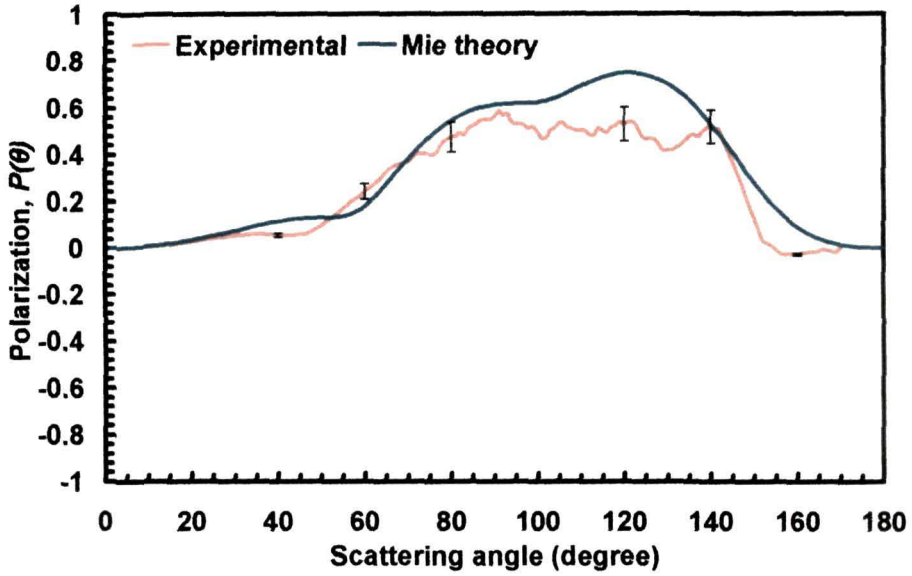

Figure 4.6(d). Measured degree of linear polarization, $P(\theta)$ of polystyrene particles at 594.5 nm incident wavelength is denoted by orange lines. The comparative Mie curve is denoted by blue line.

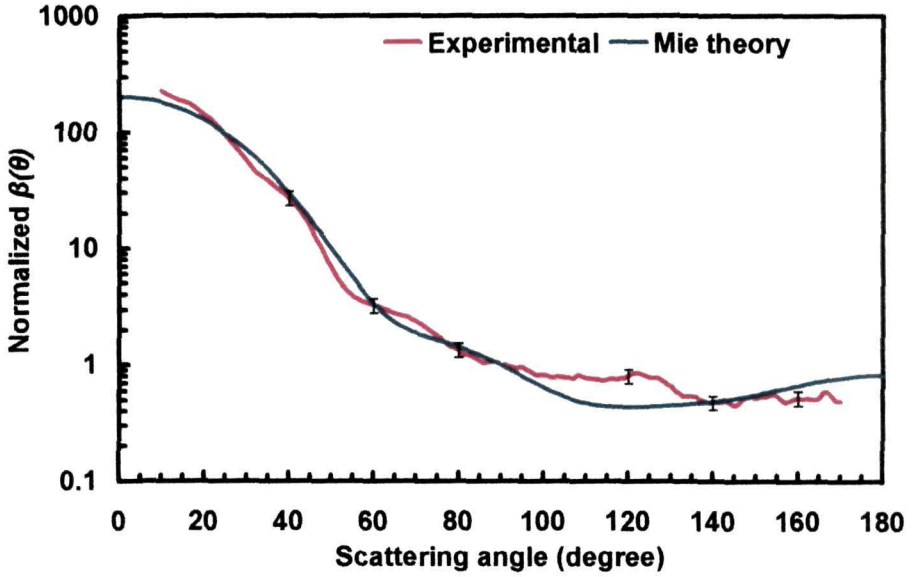

Figure 4.6(e). Measured volume scattering function, $\beta(\theta)$ of polystyrene particles at 632.8 nm incident wavelength is denoted by red lines. The comparative Mie curve is denoted by blue line.

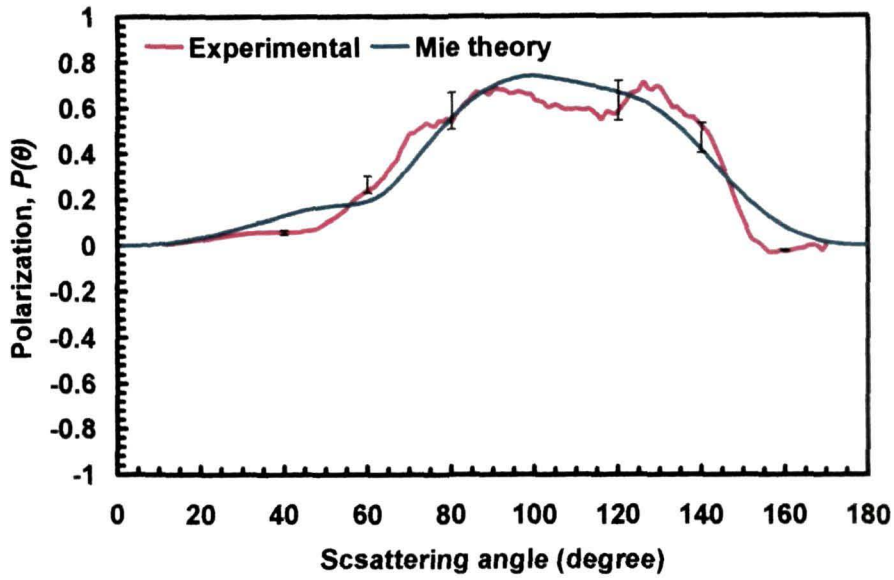

Figure 4.6(f). Measured degree of linear polarization, $P(\theta)$ of polystyrene particles at 632.8 nm incident wavelength is denoted by red lines. The comparative Mie curve is denoted by blue line.

4.3 Case I: Investigations and results on hydrosols

4.3.1 Measurements on titania (TiO_2) particles

In this section we present the results of the measurements of the volume scattering function $\beta(\theta)$ and the degree of linear polarization $P(\theta)$ of spheroidal titania particles suspended in water. Scanning electron microscopy (SEM) was done to correlate the light scattering measurements with the morphology of the titania particles and is shown in figure 4.7. From the SEM image it was estimated that the titania particles were mostly of prolate shape and had a wide dispersion of sizes, having average size of about 0.2 μm. The measured size distribution graph of the titania particles was Gaussian and is shown in figure 4.8.

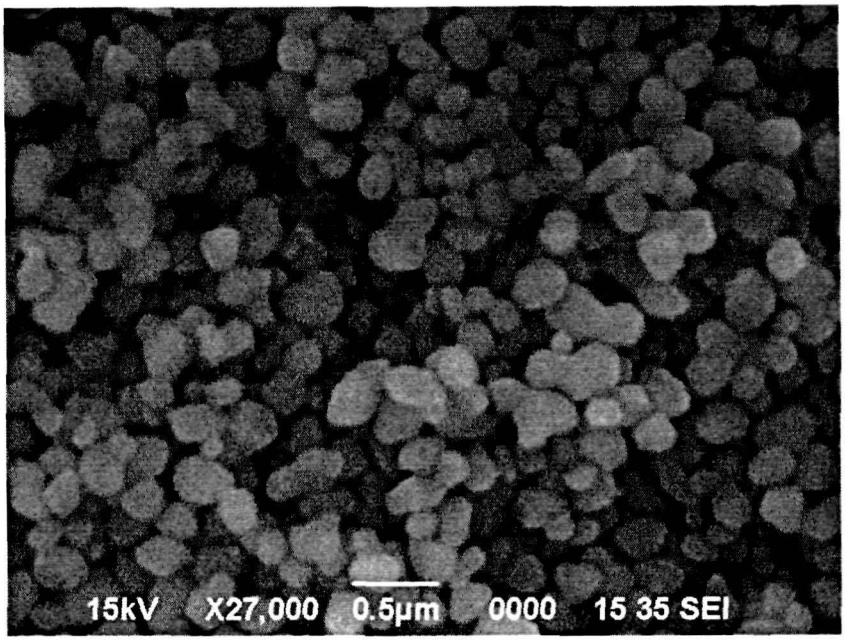

Figure 4.7. SEM image of titania particles at 27000X magnification.

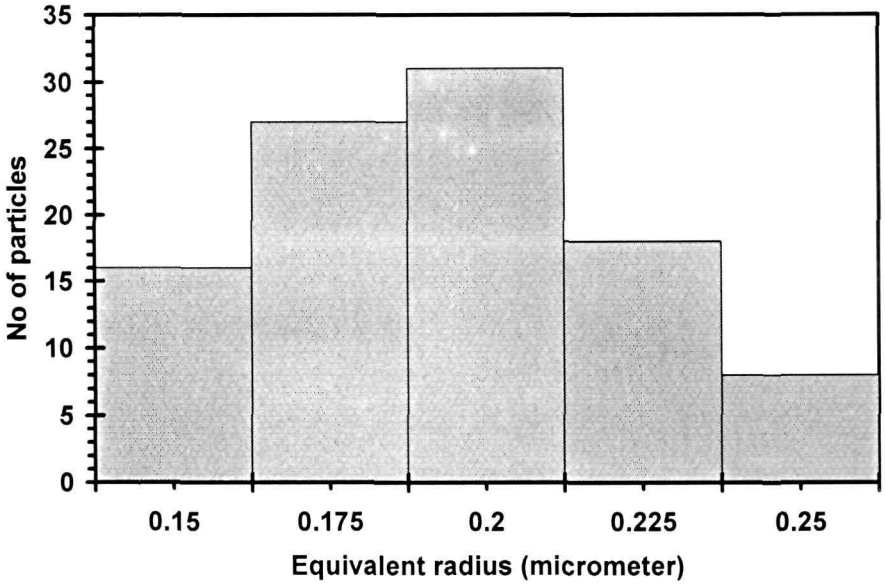

Figure 4.8. Measured size distribution of the titania particles.

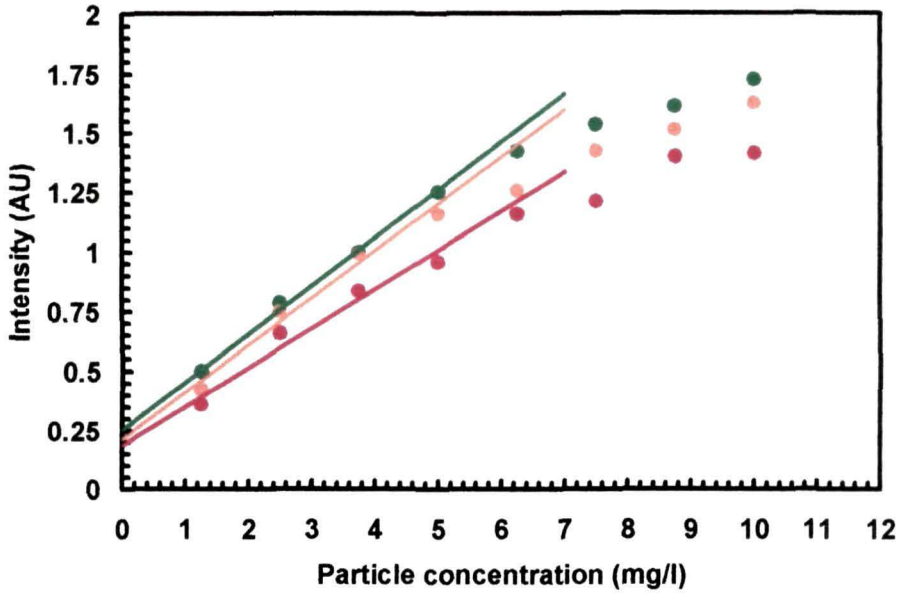

Figure 4.9. Graph for the intensity of scattered light (arbitrary units) for unpolarized light versus concentration of titania for a fixed position of the detector at 10^0. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

The optimum sample concentration for single scattering at the three incident wavelengths was measured using the method described in the section 3.4.5 and found to be approximately 5 mg/l for the titania particles as shown in figure 4.9.

Results of the measurements of the volume scattering function, $\beta(\theta)$ and degree of linear polarization, $P(\theta)$ of titania particles at 543.5 nm, 594.5 nm and 632.8 nm laser wavelengths are shown in figure 4.10(a) - figure 4.10(f). In the same plots we show the comparison of the experimental results with T-matrix calculations. Both the theoretical and experimental graphs for volume scattering functions were presented in logarithmic scale and normalized to 1 at 90 degree. From the measurements it was found that the measured volume scattering functions $\beta(\theta)$ are smooth and have fewer structures indicating a wide size dispersion of the scatterers. Measured scattering functions, $\beta(\theta)$ are sharply peaked in the forward direction. The shapes are found to be similar to those reported for in situ measurements on other aerosol particles of irregular shape

[36 - 38, 312, 313]. From the measurements the scattering efficiency of titania is found to be very strong, especially at side-scattering and backscattering angles for the three incident wavelengths. Again as far as the wavelength dependence of the scattering function is concerned; at 594.5 nm laser wavelength it showed an extra dip around 120°. The measured degree of linear polarization $P(\theta)$ showed an oscillating behavior having positive and negative branches of polarization curves. Also for this ratio, from the figure 4.10, it was observed that the measured $P(\theta)$ curves exhibit similar shapes but with differences in maxima and minima as the laser wavelength vanishes. Significant increment in the maxima of the degree of linear polarization at decreasing wavelengths was observed. The experimental results were found to be similar to those reported by Muñoz *et. al.* [38] for prolate shaped rutile (TiO_2) particles having mean diameter 117 nm. However when compared with the results obtained by PROGRA2 experiment [43, 44], significant difference in the shape of the polarization curves were observed. Previous results of the PROGRA2 experiment show the dominance of positive polarization throughout the complete phase angle range and strong polarization spikes at small phase angles around 15°. This is mainly because of the different size, density and structure of aggregates of titania samples used in the PROGRA2 experiment. In the present study, the light scattering measurements were performed on monodisperse titania particles of average size 150 nm in the single scattering regime, whereas in PROGRA2 experiment, light scattering measurements were performed on fluffy aggregates with a very high porosity (larger than 95%) and made of submicrometer titania grains [43].

It is seen that the computational capability associated with the instrument, which is directly governed by the theory behind it, is established to be quite efficient when investigations were done on polystyrene particles which were spherical in shape. But, in order to check the computational capability when the instrument is used in measurements of extremely aspheric shaped particles, scattering measurements with titania particles were compared with the theoretically generated T-matrix plots using the analytical software TUSCAT

described in Chapter II. The refractive index of titania at the three different incident wavelengths were calculated by using the dispersion formula reported by other researchers [264, 265]. The estimated parameters for T-matrix calculations are given in table 4.3 for titania particles. Figure 4.10(a), figure 4.10(c) and figure 4.10(e) show comparison of volume scattering functions $\beta(\theta)$ of titania particles with T-matrix plots. Similarly figure 4.10(b), figure 4.10(d) and figure 4.10(f) show comparison of degree of linear polarization $P(\theta)$ of titania particles with T-matrix plots.

Table 4.3. Important parameters for Mie calculation for Titania particles.

PARAMETERS	VALUE
Particle shape	Spheroidal
Axial ratio	1.2
Size distribution	Normal (Gaussian)
Minimum particle radius (μm)	0.15
Maximum particle radius (μm)	0.25
Modal radius (μm)	0.2
Deviation, σ	2
Incident light wavelengths in nanometers	543.5, 594.5 and 632.8
Average refractive index the ordinary and the extraordinary ray of titania relative to distilled water at 543.5 nm	1.980 + i0.00
Average refractive index the ordinary and the extraordinary ray of titania relative to distilled water at 594.5 nm	1.976 + i0.00
Average refractive index the ordinary and the extraordinary ray of titania relative to distilled water at 632.8 nm	1.975 + i0.00

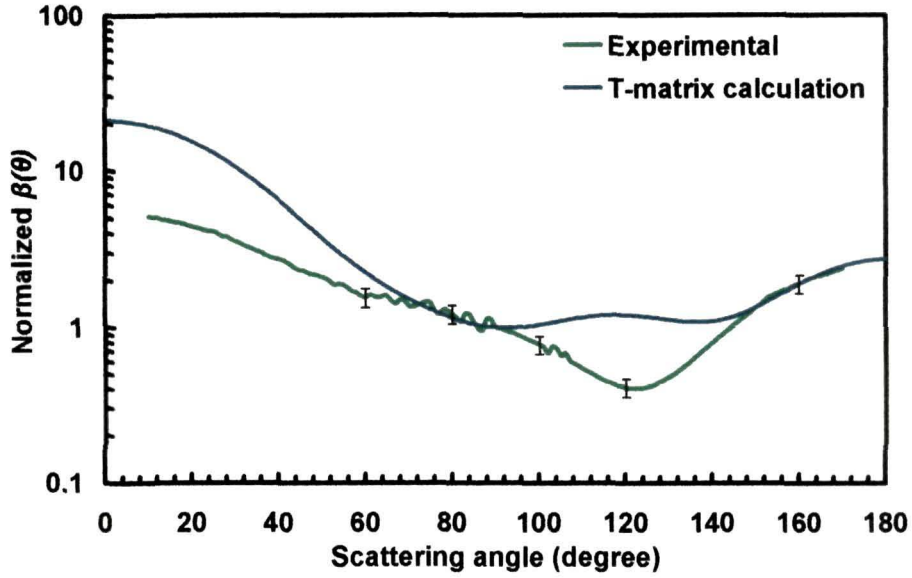

Figure 4.10(a). Measured volume scattering function, $\beta(\theta)$ of titania particles at 543.5 nm incident wavelength is denoted by green lines. The comparative Mie curve is denoted by blue line. Error bars in the plots indicate instrumental error.

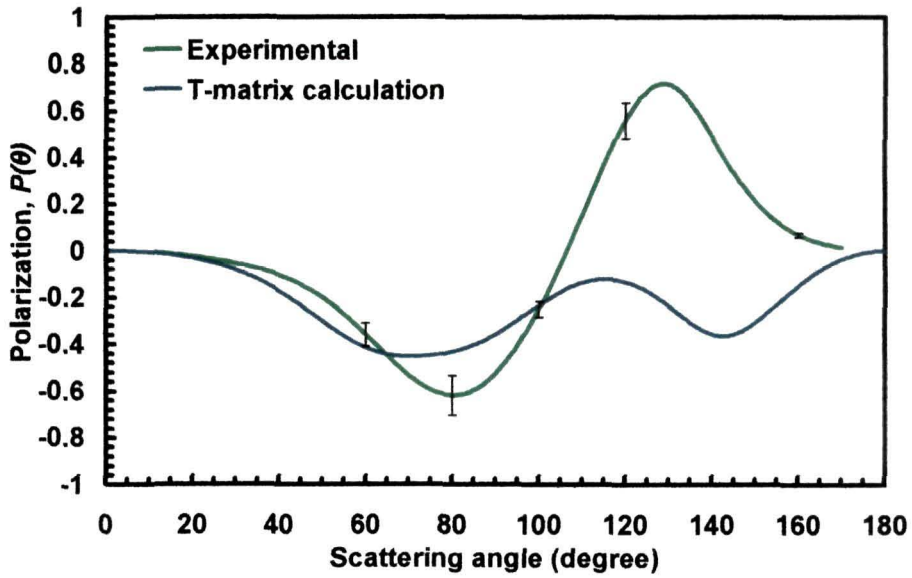

Figure 4.10(b). Measured degree of linear polarization, $P(\theta)$ of titania particles at 543.5 nm incident wavelength is denoted by green lines. The comparative Mie curve is denoted by blue line.

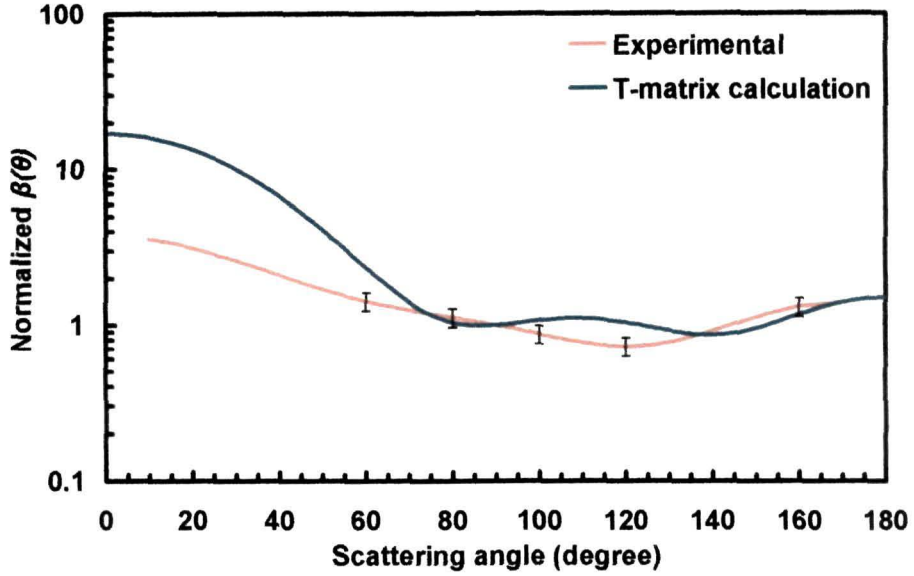

Figure 4.10(c). Measured volume scattering function, $\beta(\theta)$ of titania particles at 594.5 nm incident wavelength is denoted by orange lines. The comparative Mie curve is denoted by blue line.

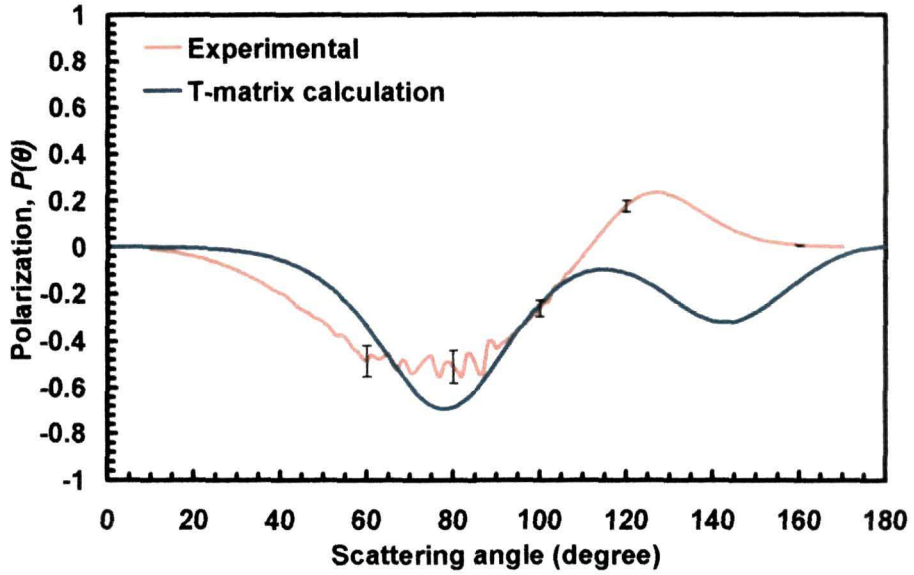

Figure 4.10(d). Measured degree of linear polarization, $P(\theta)$ of titania particles at 594.5 nm incident wavelength is denoted by orange lines. The comparative Mie curve is denoted by blue line.

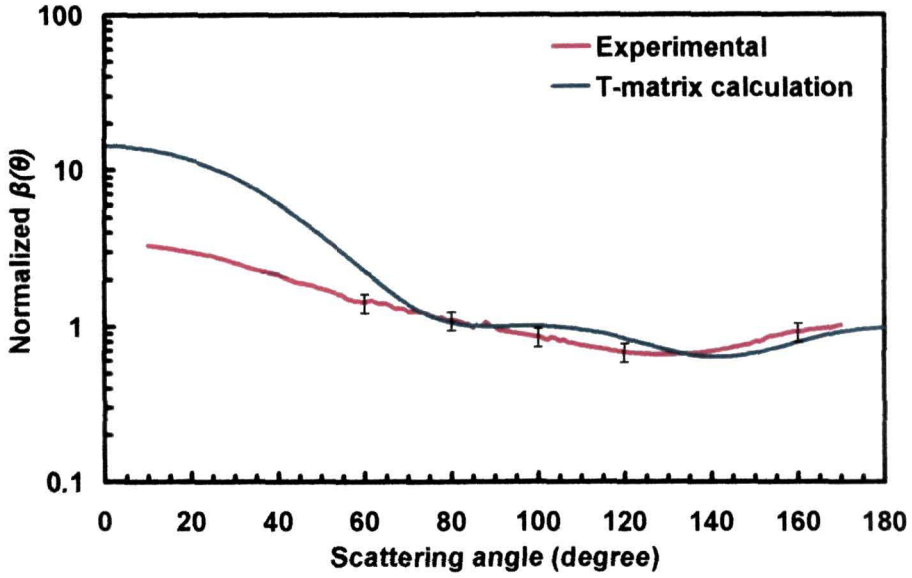

Figure 4.10(e). Measured volume scattering function, $\beta(\theta)$ of titania particles at 632.8 nm incident wavelength is denoted by red lines. The comparative Mie curve is denoted by blue line.

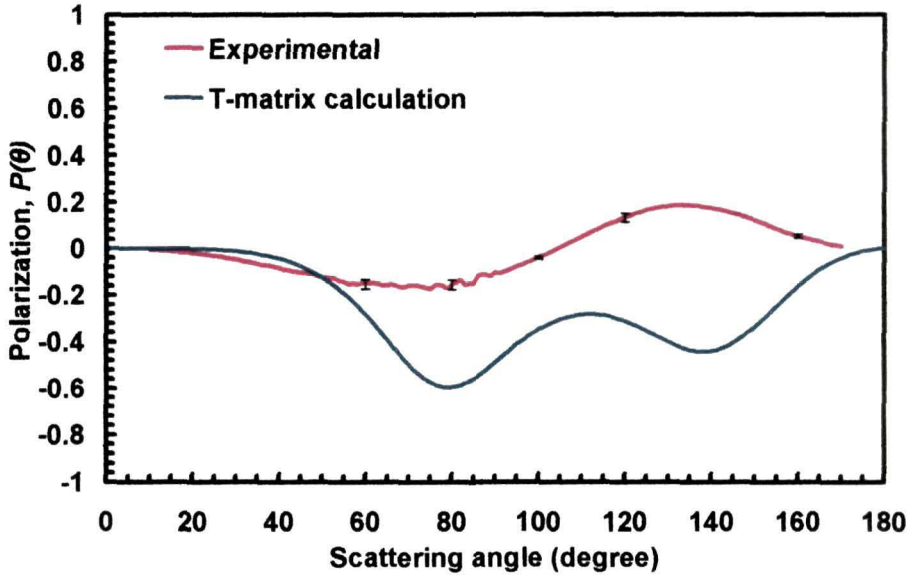

Figure 4.10(f). Measured degree of linear polarization, $P(\theta)$ of titania particles at 632.8 nm incident wavelength is denoted by red lines. The comparative Mie curve is denoted by blue line.

4.3.2 Experiments with fresh water diatoms:

Diatoms are a group of micro algae found in both fresh water and marine environment. There are over 2,00,000 species of these photosynthetic algae with world wide distribution [266]. Diatoms are unicellular structures with the protoplasts enclosed in amorphous silica cell wall (frustule) consisting of two valves joined together by a girdle [267]. The morphogenesis of the peculiar cell wall architecture of diatom involves biomineralization of silica, forming an array of patterns (like the "loculate areolae" and "cribra"). These structures in the diatom frustule range from micrometric to nanometric scales which are different for different species. Such biosynthesized silica nanostructures in diatoms have the following qualities -

- highly ordered: because they are the product of the genetic program of the cell
- stable: because they arise out of the complex cellular processes
- less expensive: as they can be grown in large scale in ambient condition by following some simple procedures
- diverse: as there are more than 200000 species under 250 genera i.e. 200000 different genetic programmes for forming silica structures.

The diversity of such nanostructures extends possibilities for their use in nanofabrications of a multitude of devices having wide range of potential areas of application, such as in optoelectronics, microelectronics, biomedical technology [267 - 269]. Some specific applications under consideration are embedding diatom frustules in metal film membranes, magnetizing frustules for targeted drug delivery, producing silica nanopowders from diatom frustules through large scale culture of diatoms etc. Extensive studies relating to the optical, magnetic, electrical and other biophysical features of different diatom species is crucial for such technology development. The photoluminescence and cathodoluminescence studies on certain fresh water diatoms have been made to explore the possibilities of optoelectronic and photonic applications [269, 270]. Such studies are also necessary to understand the biophysical mechanism of

silica nanoarchitecture formation in diatoms. We selected tropical fresh water diatoms as a candidate for light scattering studies as such measurements were capable of providing information of the finer structures present on its surface, information of underwater light climate, new clues as to the development of new models for light scattering from complex bodies, indication of water quality in water bodies, etc.

Preparation of samples

The living fresh water diatoms were collected from the catchment ponds of a water purification plant in Assam, India during September - November, 2007. The collected diatoms were grown in the Tissue Culture Laboratory, Department of Molecular Biology and Biotechnology of Tezpur University, Assam, India. The sample (20 ml) was centrifuged at 6000 rpm for 15 minutes to allow sedimentation of the heavy diatom particles. The precipitate was suspended in 1 ml of distilled water and washed.

The "WC" media proposed by Guillard and Lorenzen (1972) [271] was used with slight modifications to culture the diatoms using the prepared sample as inoculum. The composition of the media is given in table 4.4. The solid culture plates were incubated at 22^0C under white fluorescent light in a B.O.D. incubator (Narang Scientific Works, New Delhi) for 14 days. For the liquid culture all the growth nutrients were dissolved in 1000 ml of sterile water and the media was autoclaved before inoculation with environmental samples. The sample diatoms for light scattering study were obtained from the same culture and their shapes were similar with a small size variation.

Table 4.4. Major nutrients and micronutrients for modified freshwater "WC" medium.

COMPOSITIONS	AMOUNT (mgL^{-1})
<u>Nutrients</u>	
$CaCl_2.2H_2O$	36.76
$MgSO_4.7H_2O$	39.97
$NaHCO_3$	12.60
K_2HPO_4	8.71
$NaNO_3$	85.01
$Na_2SiO_3.9H_2O$	56.82
Na_2EDTA	4.36
$FeCl_3.6H2O$	3.15
<u>Micronutrients</u>	
$CuSO_4.5H_2O$	0.01
$ZnSO_4.7H_2O$	0.022
$CaCl_2.6H_2O$	0.01
$MnCl_2.4H_2O$	0.18
$Na_2MoO4.2H_2O$	0.006
H_3BO_3	1.0
<u>Vitamins</u>	
Thiamin HCl	0.1
Biotin	0.5
B_{12}	1.0
p^H	6.23

Frustules Isolation

In order to prepare diatom frustules for scanning electron microscopy (SEM), photoluminescence (PL) spectroscopy and light scattering studies, a cleaning procedure was needed that removed the external organic matrix covering the frustules. In this work, this was done by using the following steps:

- culture flask was shaken for 5 minutes to detach all diatoms;
- 10 ml of sample was centrifuged at 5000 rpm for another 10 min;
- the pellet was then washed for several times in double distilled water;
- 37% aqueous HCl was added and centrifuged at 3000 rpm for 10 minutes and was put in water bath for 15 min at 60^0C;
- the acid was pipetted and pellet was washed again in double distilled water 3 times,
- Cleaned frustule valves were then stored in ethanol to avoid contamination and bacteria growth.

Electron Microscopy and EDS analysis

The structural characterizations of cleaned frustules were examined by scanning electron microscope (SEM). The cleaned frustules were partly mounted on brass stubs and coated with platinum for scanning electron microscopy (SEM) and X-ray energy dispersive (EDS) analysis. Figure 4.11 shows a scanning electron micrograph (SEM) and figure 4.12 shows the energy dispersive X-ray spectroscopy (EDS) graph. It is worth mentioning that the distribution and size of the nanopores and nanostructures on the walls of the diatoms (frustules) is a critical factor in determining their light scattering behavior. The microscopy images show that the diatoms have a regular spheroidal shape and are of almost equal size (average length is 7μm and average width is 2 μm). From the SEM pictures the diatom was identified as *Gomphoneis* sp.

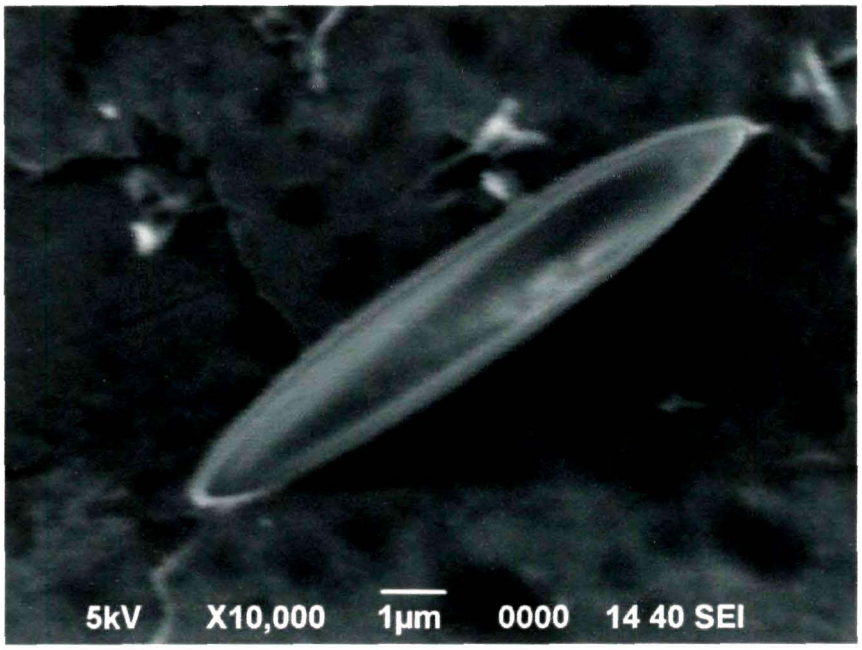

Figure 4.11. Scanning electron micrograph of a freshwater diatom frustule (*Gomphoneis* sp.) at 10000X resolution (scale 1μm).

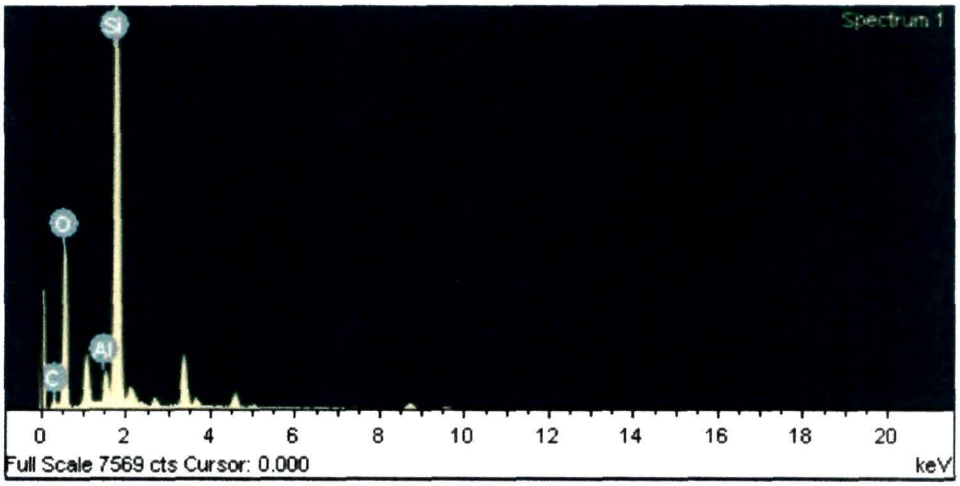

Figure 4.12. Energy Dispersive X-ray Spectroscopy graph for freshwater diatom of *Gomphoneis sp.*

Energy dispersive X -ray spectroscopy (EDS) was done as it is considered as the standard procedure for identifying and quantifying elemental composition of sample area, as small as a few cubic micrometers. From SEM-EDS spot analysis it was confirmed that the frustules isolated from diatoms are composed

mainly of silicon (at 2 KeV in figure 4.12) in the form of amorphous silica (SiO_2). It was observed that the sample also contained a little amount of Al, O and C etc.

Photoluminescence (PL) investigation

As it has been reported [270] that porous semiconductor and insulating materials, especially porous silicon, has great luminescence efficiency at room temperature in the visible region when irradiated at certain wavelengths, photoluminescence of diatoms were measured during light scattering studies with 543.5 nm, 594.5 nm, 632.8 nm and 325 nm excitation wavelengths to investigate if the light scattering data would be effected by the photoluminescence at 543.5 nm, 594.5 nm and 632.8 nm probing wavelengths. The photoluminescence were measured by using Perkin Elmer LS55 which uses Xenon discharge lamp as the excitation source. It was observed from figure 4.13(a) - figure 4.13(f) that diatoms and the frustules did not show any PL activity at 543.5 nm, 594.5 nm and 632.8 nm excitation wavelengths indicating that the light scattering spectra at these wavelengths are not interfered by PL. However, at 325 nm excitation beam both the living diatoms and the frustules emitted a strong blue photoluminescence, that was clearly visible with the naked eye and are shown in figure 4.13(g) and figure 4.13(h) respectively. The PL spectrum of the diatoms and the frustules were collected and was found to have a broad peak at 445 nm (2.79 eV). The Pl peak at 430 nm was found to be more prominent in case of frustules. Although consideration of luminescence in general do not form the main aim of this thesis, it may be pointed out that the relatively large amount of PL obtained at 325 nm excitation wavelength may be used for fabricating efficient luminescence devices in UV range.

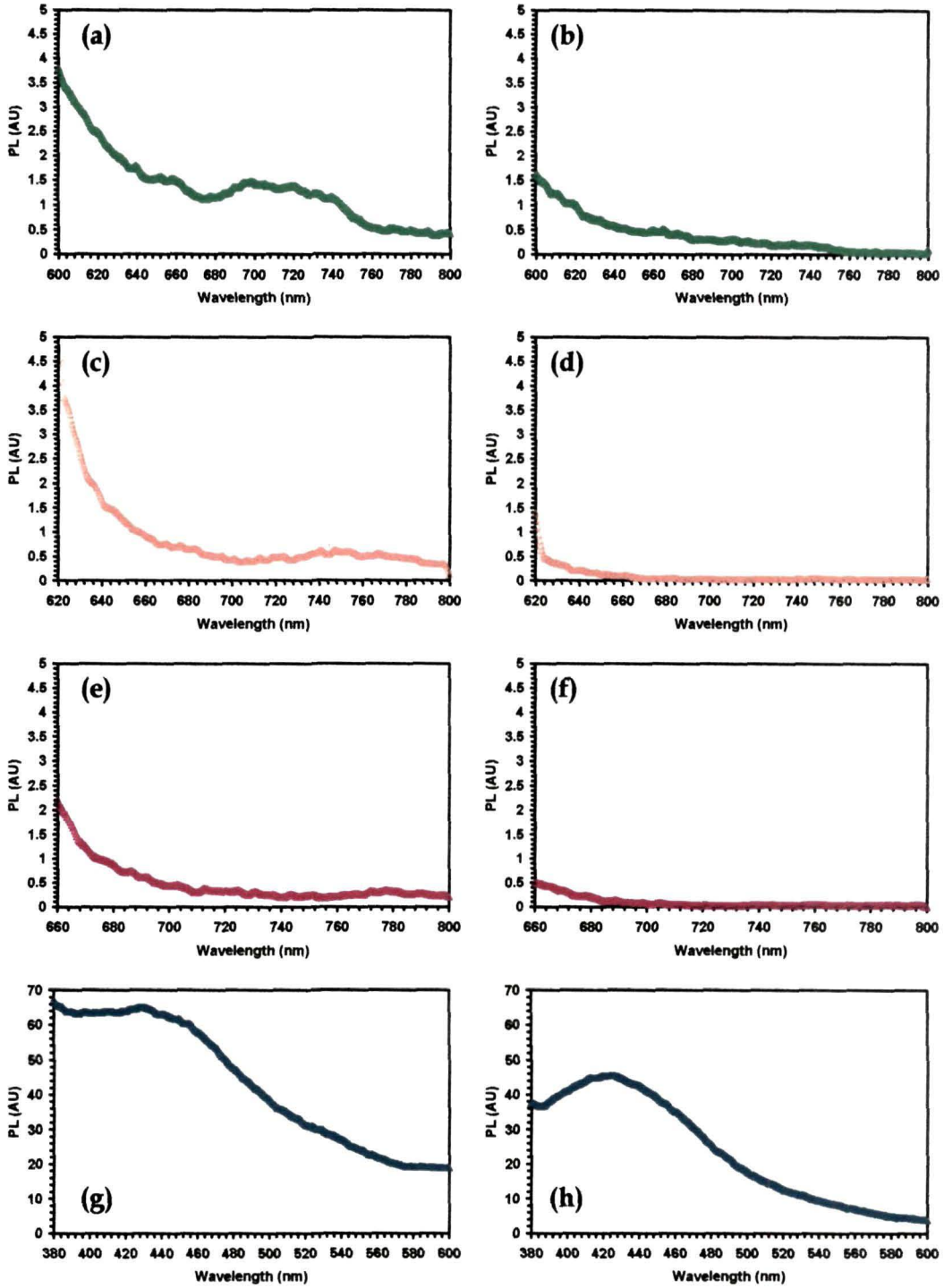

Figure 4.13. Photoluminescence spectra of diatoms and frustules are shown in the left and right panels, respectively for: [(a) and (b)] 543.5 nm; [(c) and (d)] 594.5 nm; [(e) and (f)] 632.8 nm; and [(g) and (h)] 325 nm excitation wavelengths.

Light scattering measurements

The purpose of this work was to measure experimentally the angular scattering behavior of diatoms and their frustules collected from different locations. The results can be used in design of nanophotonic applications and to develop the theoretical models. Volume scattering functions and degree of linear polarization of the diatoms and their frustules were measured as functions of scattering angle at 543.5 nm, 594.5 nm and 632.8 nm laser wavelengths. The results of the measurements are presented in figure 4.16(a) - figure 4.16(f). For the light scattering measurements the optimum sample concentrations for single scattering at the three incident wavelengths were found to be approximately 25×10^4 cells/ml and 29×10^4 cells/ml for diatoms with organic matrix (will be referred as diatoms hereafter) and diatoms without organic matrix (will be referred as frustules hereafter) respectively. The cell counts were determined by using a haemocytometer. Up to these concentrations the graph between the scattered intensity versus sample concentration was linear as shown in figure 4.14 and figure 4.15.

From the results it was seen that, at the three different wavelengths, the volume scattering functions, $\beta(\theta)$ for both diatoms and frustules were sharply peaked in the forward direction. However it was not equally intense for all the three incident wavelengths. The scattering intensities were found to be higher in case of 543.5 nm as compared to 594.5 and 632.8 nm wavelengths. The measured volume scattering functions, $\beta(\theta)$ showed an oscillatory behavior which indicated the narrow size distribution of the samples under study. It was clear from the graphs that the differences in the scattering function behavior for diatoms and frustules do not vary to a large extent. Still we observed that the scattering function for diatoms showed prominence as compared to the case of frustules. This variation may be attributed to the external organic matrix with complex morphological features.

The measured degree of linear polarization, $P(\theta)$ for diatoms with and without organic matrix showed an oscillating behavior having highest amplitude

of +0.54 and lowest amplitude of -0.66 as shown in figure 4.16(b), figure 4.16(d), figure 4.16(f) at 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths respectively. The angular positions of maxima and minima were almost same for all the measurements except for frustules at 110° for 543.5 nm incident wavelength. Considerable difference in the intensities of the maxima and minima were observed for diatoms and frustules.

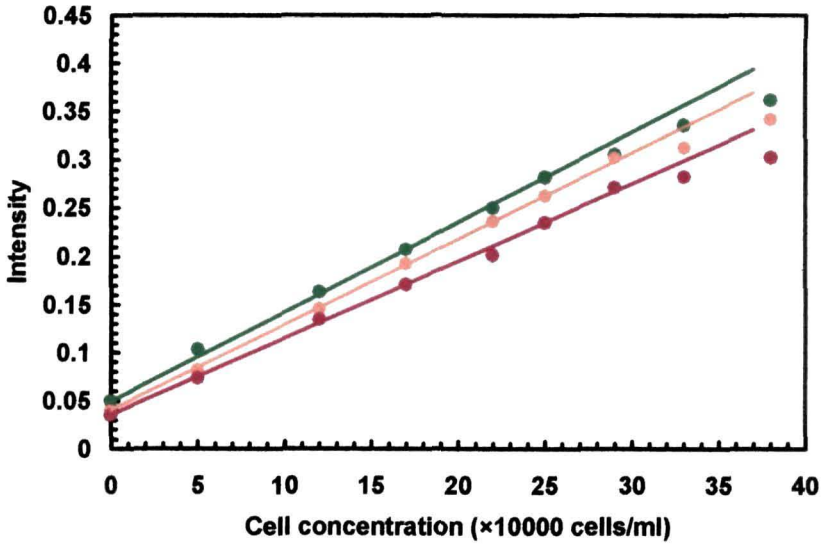

Figure 4.14. Graph for the scattered intensity for unpolarized incident light versus cell concentration of diatoms for a fixed position of the detector at 10°. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

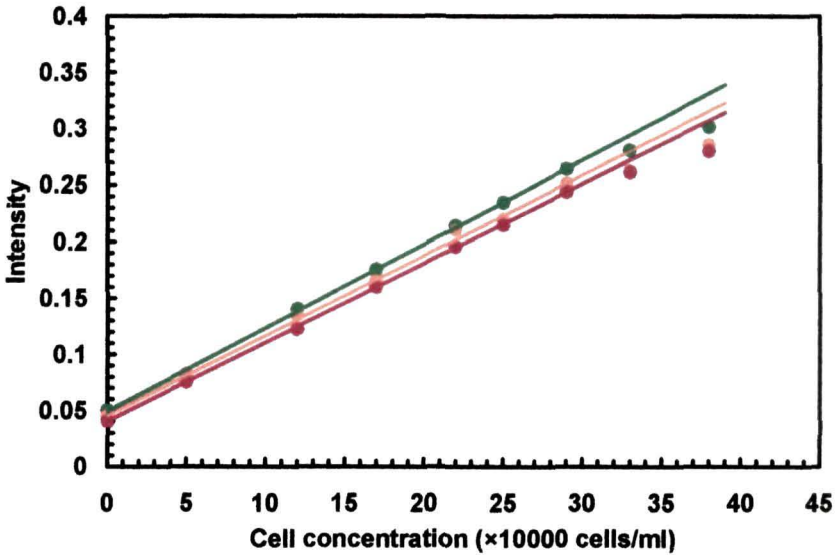

Figure 4.15. Graph for the scattered intensity for unpolarized incident light versus cell concentration of frustules for a fixed position of the detector at 10^0. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

Table 4.5. Important parameters for Mie calculation for diatoms.

PARAMETERS	VALUE
Particle radius in micrometers (average)	4.5
Particle refractive index relative to distilled water	1.0827 + i0.00000
Incident light wavelengths in nanometers	543.5, 594.5 and 632.8

We correlated our measurements for the volume scattering function with Mie calculations [1, 2] using estimated parameters (table 4.5) in order to draw a comparison of scattering measurements with Mie theory predictions. The C language program TUMiescat.c was used to compute theoretical values of $\beta(\theta)$ and $P(\theta)$. The program gave accurate results even for very large size parameters. We found considerable deviation of the theoretical predictions from the measurements for all the three incident wavelengths as shown in figure 4.16(a) -

figure 4.16(f). This was expected as it has been reported that [63] the light scattering behavior of such biological microorganisms cannot be easily predicted from the morphology of the particles. Spherically and cylindrically celled species may, sometimes, produce similar results, whereas two spherical species may yield quite different results. As such it is necessary to understand the morphology of the structure to a greater extent and modify the existing theories to correlate the experimental results.

Comparison with San Diego Harbor scattering function:

The results of the volume scattering function, $\beta(\theta)$ were compared with the scattering function of the San Diego Harbor water measured by Petzold in 1972 [75] in order to get the differences of angular scattering behavior of diatoms and frustules relative to Petzold's standard. This function is taken as a standard to compare measurements of scattering from hydrosol samples as it is frequently used in modeling remotely sensed reflectance of turbid waters [63, 77]. Figure 4.16(a), figure 4.16(c) and figure 4.16(e) show the measured volume scattering functions for diatoms and frustules compared with Petzold's San Diego Harbor scattering data. The Mie scattering pattern is also shown in the same plots. Data obtained from experimental measurements and Mie calculations are scaled to the San Diego Harbor scattering function at 90°. The measured scattering functions were sharply peaked in the forward direction. For all the three probing wavelengths, the scattering functions for diatoms and frustules lay below the San Diego Harbor scattering function at forward angles whereas the scattering functions for diatoms and frustules lay above the San Diego Harbor scattering function at angles greater than 100°. However it was observed that the scattering function of frustules lay closer to the San Diego Harbor scattering function as compared to the scattering function of diatoms. At the lowest measurement angle i.e. 10°, the differences between the measured and San Diego Harbor scattering function were upto a factor 12 lower for diatoms and 16 lower for frustules at 543.5 nm; 12.5 lower for diatoms and 17 lower for frustules at 594.5

frustules at 594.5 nm; 12 lower for diatoms and 16 lower for frustules at 632.8 nm incident wavelengths. Again, for the highest measurement angle (170°), the difference between the experimentally measured and San Diego Harbor scattering function was upto a factor 0.5 higher for diatoms and 0.8 higher for frustules at 543.5 nm; 0.6 higher for diatoms and 1.1 higher for frustules at 594.5 nm; 0.5 higher for diatoms and 0.8 higher for frustules at 632.8 nm incident wavelengths. Thus in the forward direction, the measured scattering functions deviated the most.

The light scattering measurements reported here may provide new clues as to the development of new models for light scattering from complex biological structures.

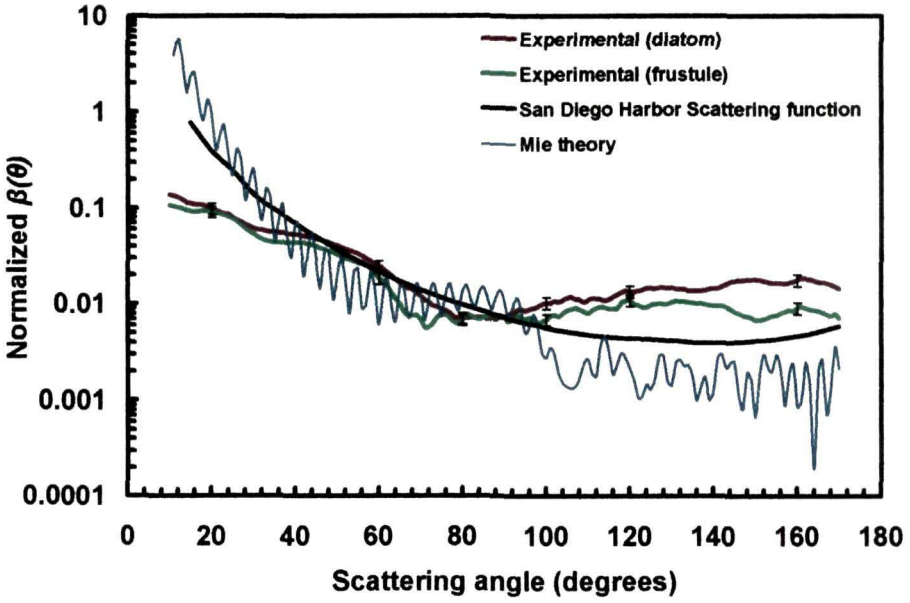

Figure 4.16(a). The comparative volume scattering function, $\beta(\theta)$ at 543.5 nm laser wavelengths. Graph for diatoms (with organic matrix) is shown by brown line and graph for frustules is shown by green line. The function $\beta(\theta)$ is scalled at 90° with the San Diego Harbor Scattering function (black line) and compared with Mie calculations (blue line).

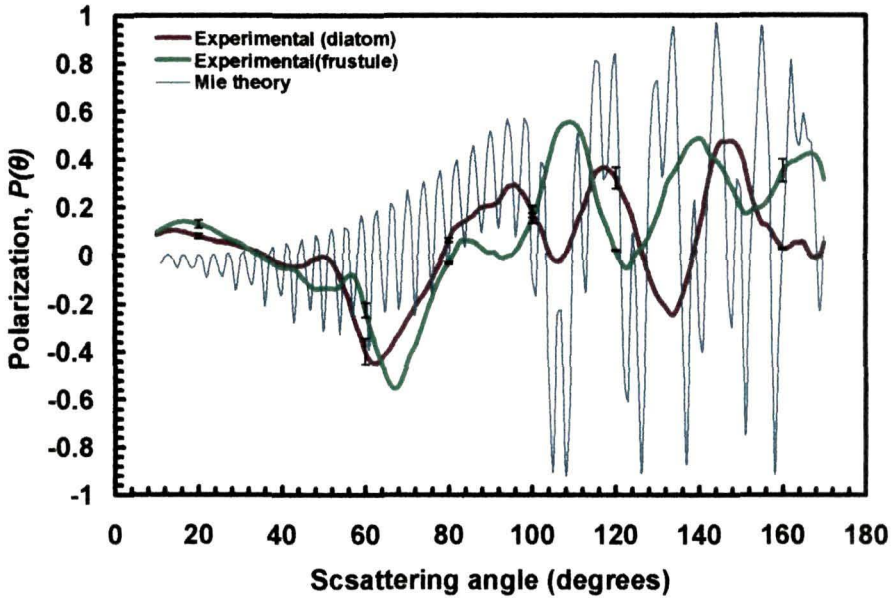

Figure 4.16(b). The degree of linear polarization, $P(\theta)$ at 543.5 nm laser wavelengths. Graph for diatoms (with organic matrix) is shown by brown line and graph for frustules is shown by green line. $P(\theta)$ is also compared with the theoretically generated Mie plot (blue line).

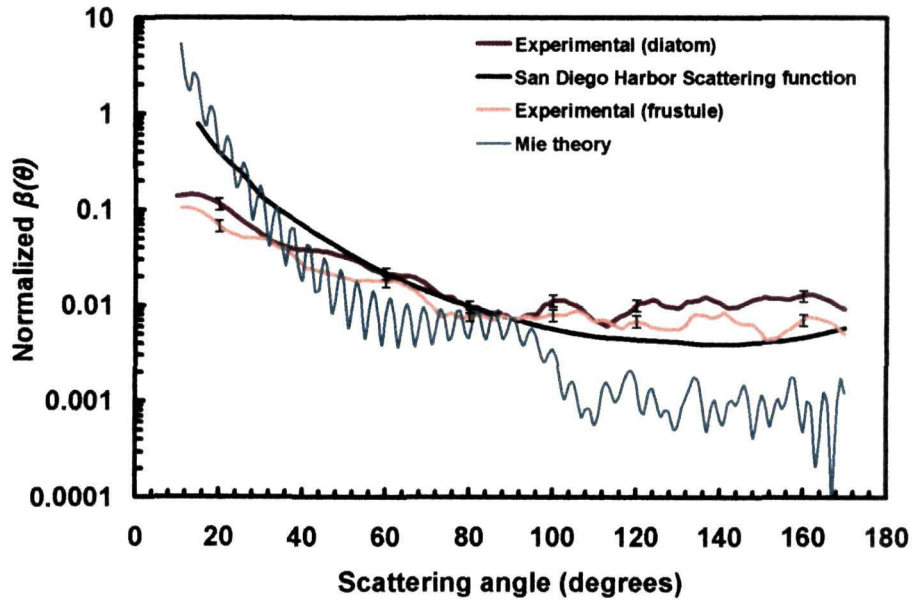

Figure 4.16(c). The comparative volume scattering function, $\beta(\theta)$ at 594.5 nm laser wavelengths. Graph for diatoms (with organic matrix) is shown by brown line and graph for frustules is shown by orange line. The function, $\beta(\theta)$ is scalled at 90° with the San Diego Harbor Scattering function (black line) and compared with Mie calculations (blue line).

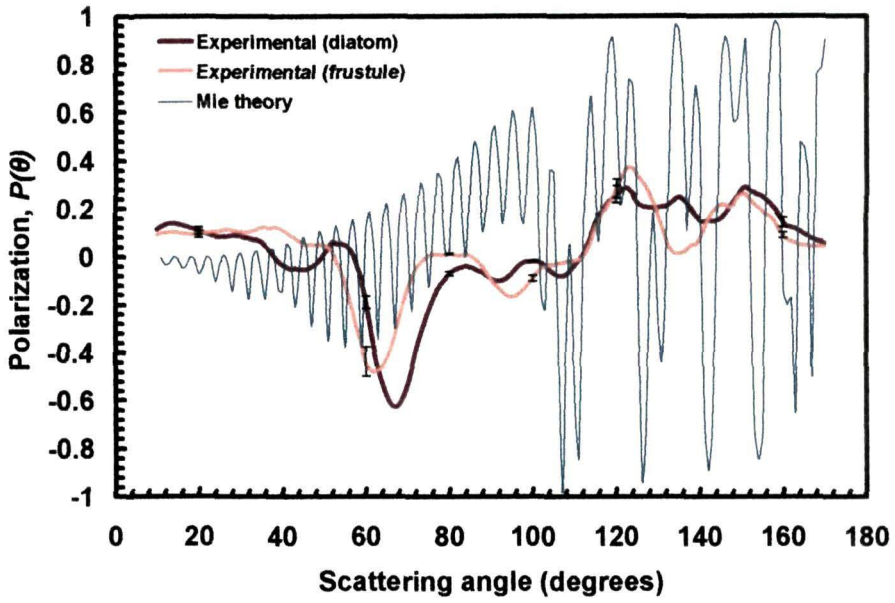

Figure 4.16(d). The degree of linear polarization, $P(\theta)$ at 594.5 nm laser wavelengths. Graph for diatoms (with organic matrix) is shown by brown line and graph for frustules is shown by orange line. $P(\theta)$ is also compared with the theoretically generated Mie plot (blue line).

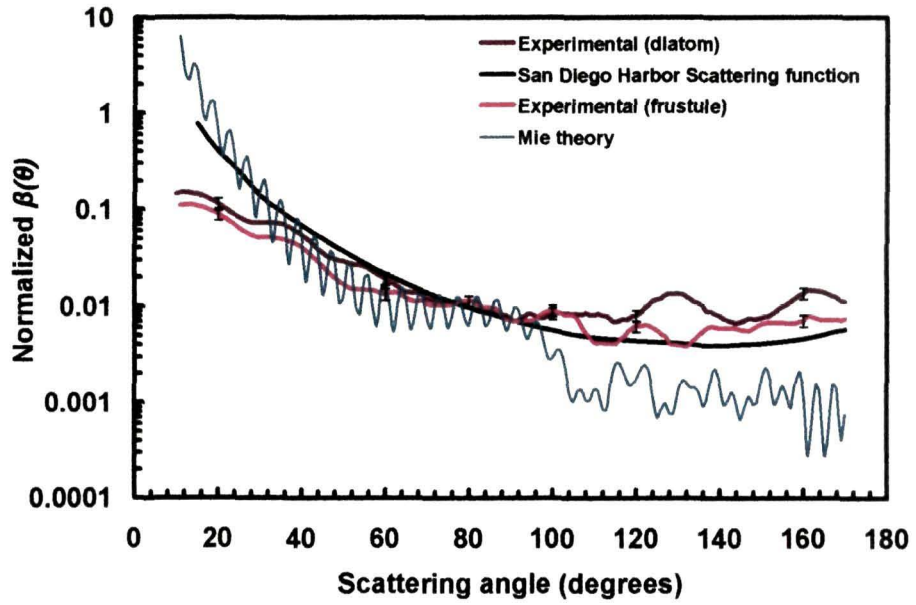

Figure 4.16(e). The comparative volume scattering function, $\beta(\theta)$ at 632.8 nm laser wavelengths. Graph for diatoms (with organic matrix) is shown by brown line and graph for frustules is shown by red line. The function, $\beta(\theta)$ is scalled at 90^0 with the San Diego Harbor Scattering function (black line) and compared with Mie calculations (blue line).

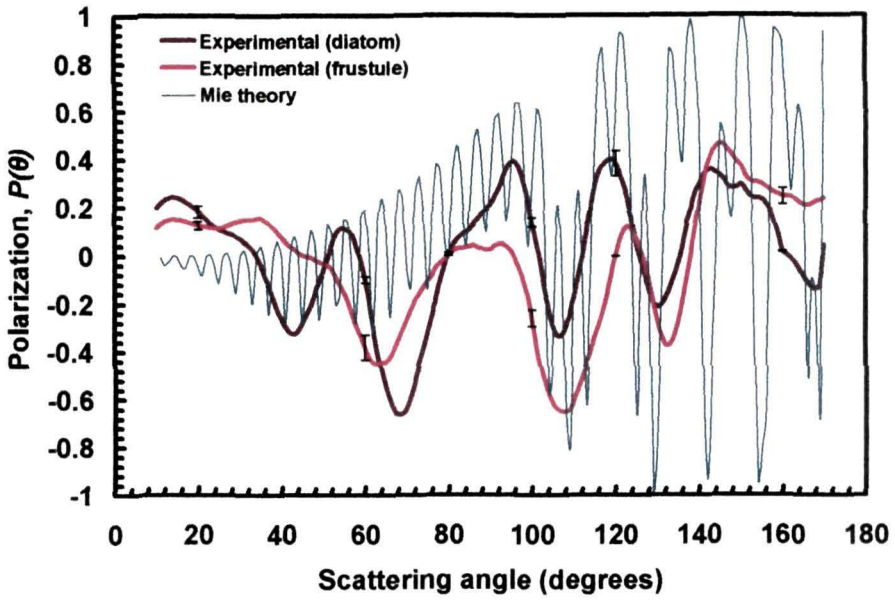

Figure 4.16(f). The degree of linear polarization, $P(\theta)$ at 632.8 nm laser wavelengths. Graph for diatoms (with organic matrix) is shown by brown line and graph for frustules is shown by red line. $P(\theta)$ is also compared with the theoretically generated Mie plot (blue line).

4.4 Case II: Experiments on nanoparticles

4.4.1: Measurements on ZnS nanoparticles with 'Type 1' sample holder

We began with the study of light scattering properties of wide band gap ZnS semiconductor nanoparticles embedded in Polyvinyl Alcohol (PVA) matrix. Polyvinyl Alcohol (Aldrich) was selected as the host polymer matrix as it is an optically transparent aliphatic polymer having refractive index of 1.55 and dielectric constant of 2.0 [275 - 277]. It melts at 413K and its density is around 1.08 g/ml. For the preparation of the matrix, a 2.5 wt.% PVA solution was prepared in double distilled water, by stirring in a magnetic stirrer with stirring rate at ~200 rpm at a constant temperature of 70^{0}C until a transparent solution was formed. Meanwhile, an aqueous solution of 0.15M $ZnCl_2$ was prepared. The aqueous solution was mixed into as prepared 2.5 wt.% optically transparent PVA matrix in a ratio 1:2 and stirred for 6 hours at the same stirring rate as before but this time maintaining the temperature at 65^{0}C. This solution was treated with freshly prepared 0.15M Na_2S solution and left in a cool and dark condition. Colourless or faint milky coloured solution containing ZnS nanoparticles was

obtained after 24 hours. The quality of the nanoparticles in terms of size and shape was controlled by varying the molar concentration of the Na_2S solution from 0.05M keeping the concentration of the $ZnCl_2$ solution fixed. Notably, use of very small amount of Na_2S manifest very small sized nanoparticles (quantum dots) whereas abundant use of Na_2S produces larger particles (sometimes in micrometer range) both of which are not of interest for the present study. The best quality ZnS nanoparticles for light scattering studies were found where the concentration of Na_2S became equal to the concentration of $ZnCl_2$ i.e. 0.15M. Also, upto this combination of molar concentration the graph between the scattered intensity versus Na_2S concentration was linear as shown in figure 4.17. As the experiment was done in differential mode, PVA samples with and without embedded nanoparticles were prepared in specially designed sample holders of size 1 sq.cm. The block diagram of the chemical synthesis process is given in figure 4.18.

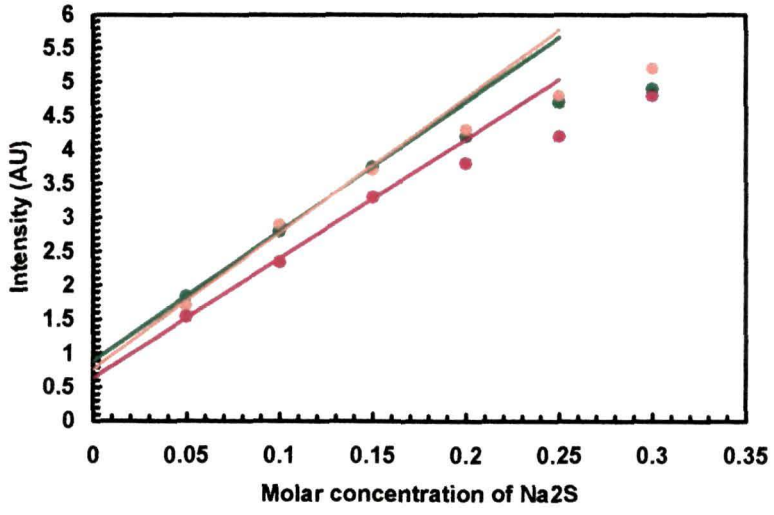

Figure 4.17. Graph for the scattered intensity for unpolarized incident light versus molar concentration of Na_2S for a fixed position of the detector at 10^0. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

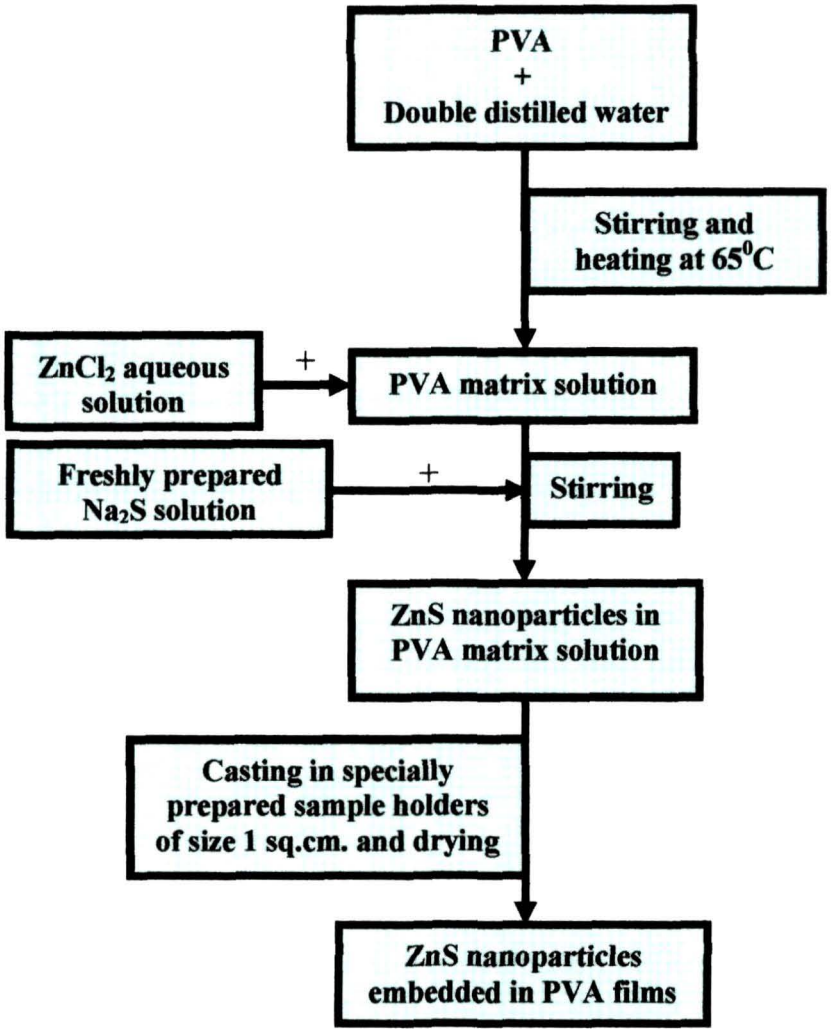

Figure 4.18: Schematic diagram for chemical synthesis of ZnS nanoparticles.

X-Ray Diffraction (XRD) studies of the nanoparticles were carried out with a Rigaku X-Ray Diffractometer (model: MINIFLEX) for particle identification and microstructural characterization. The analysis showed peaks at Bragg's angles around 28° and 47° (fig. 4.19) confirming the presence of ZnS particles as it is known that bulk ZnS show distinct peaks at 27.5° and 46.2°. The broadening of XRD diffraction peak compared to bulk confirms the forrmation of nanoparticles. The XRD gave a rough estimation of average particle size (~70 nm), obtained by measuring full-width-at half maxima (FWHM) and using Scherrer formula ($d = 0.9\frac{\lambda}{w\cos\theta}$).

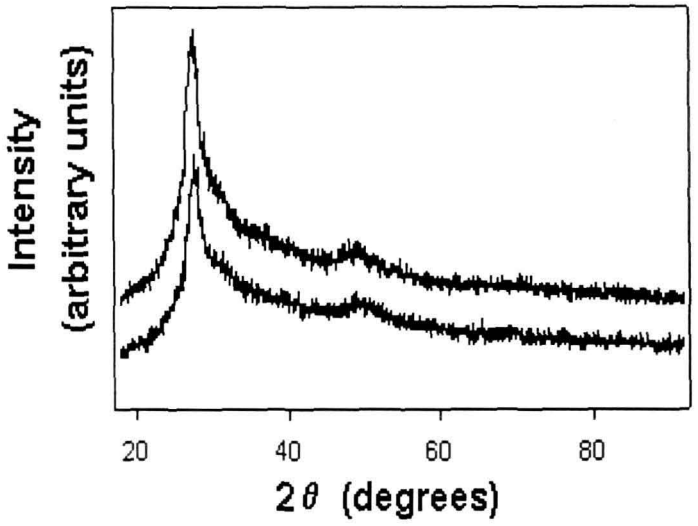

Figure 4.19: XRD pattern of chemicaly synthesized ZnS nanoparticles

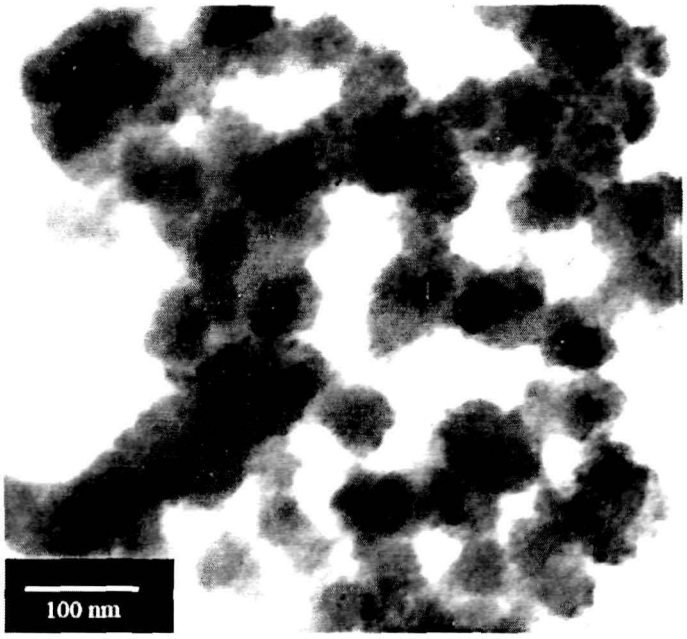

Figure 4.20. Transmission Electron Microscopy image of ZnS nanoparticles dispersed in PVA matrix.

Again to get the surface morphology along with the size of the polymer embedded ZnS nanoparticles, a Digital Transmission Electron Microscope (Model: JEOL JSM 100CX) was used. From the TEM pictures (figure 4.20) the ZnS

nanoparticles were found to be spherical having approximately equal sizes. Some of the particles were found to be in cluster form. The average size of the individual nanoparticles was approximately 70 nm.

The results of the measurements on the angular dependence of the volume scattering function $\beta(\theta)$ and degree of linear polarization $P(\theta)$ (also termed as polarization) of ZnS semiconductor nanoparticles at 543.5 nm, 594.5 nm and 632.8 nm incident laser wavelengths are presented in this section. The results of the measurements were also compared with Mie calculations and are shown in figures 4.21(a) - 4.21(f).

From the results it was observed that, the measured volume scattering functions $\beta(\theta)$ were not equally intense at the three different wavelengths. The scattering intensity, in case of 543.5 nm, was found to be higher as compared to 594.5 and 632.8 nm laser wavelengths. The presence of oscillatory peaks in the patterns of $\beta(\theta)$ indicated that the nanoparticles in the scattering volume were of nearly equal size (monodisperse). At large scattering angles (e.g. around 120° and 170°) the patterns of the volume scattering functions, as can be seen from figure 4.20(a), 4.20(c), 4.20(e), were associated with broad scattering peaks at all the three incident wavelengths. Again, the measured degree of linear polarization, $P(\theta)$ for the nanoparticles was found to be positive at most of the scattering angles at all the three incident wavelengths. A shift of the angular positions of polarization maxima towards higher scattering angles with the increase of incident wavelength was also observed. At 632.8 nm laser wavelength highest positive polarization of +0.75 (equivalent percent of polarization ~75%) was observed. The measured lowest negative polarization was 0.17 (equivalent percent of polarization ~17%) and was obtained at 543.5 nm incident wavelength.

Table 4.6. Estimated parameters for Mie calculations

PARAMETERS	VALUE
Size distribution	Monodisperse
Radius of the particle (in μm)	0.35
Environment refractive index	1.55 + i0.00000
Average particle refractive index at 543.5 nm, 594.5 nm and 632.8 nm	2.367+ i0.00000
Incident light wavelengths in nanometers	543.5, 594.5 and 632.8

We correlated our measurements for the volume scattering function and the degree of linear polarization with Mie calculations using estimated parameters (table 4.6) in order to draw a comparison of scattering measurements with Mie theory predictions. The analytical software TUSCAT [350] was used to compute the theoretical values of volume scattering function and degree of linear polarization. Significant deviation of the theoretical predictions from the measurements for all the three incident wavelengths was observed. The deviation of the experimental plot from the theoretically generated plots in figure 4.21(a) - figure 4.21(f) can be attributed to the superimposition effect of the dominant scattering pattern from individual spherical ZnS nanoparticles with the scattering pattern to a lesser intensity due the non-spherical cluster formation of the ZnS nanoparticles. The measured values of the volume scattering functions at small scattering angles (<60°) is significantly higher than the computed values which indicated the presence of agglomerated particles of larger sizes in the scattering volume. It is noteworthy to mention that the theoretical values of the degree of linear polarization P at 90° scattering angle equals unity, i.e. P(90°) = 1 as $I_{\parallel}$ vanishes at the Rayleigh scattering regime [345]. However this is not the case for a larger particle (beyond Rayleigh scattering regime) where $I_{\parallel} \neq 0$ and hence P(90°) < 1. Thus the presence of measured polarization values less than unity at all the three incident wavelengths also predicted the cluster formation of

the nanoparticles which made the effective sizes of the scatterers larger than the estimated values used for the Mie theary calculations.

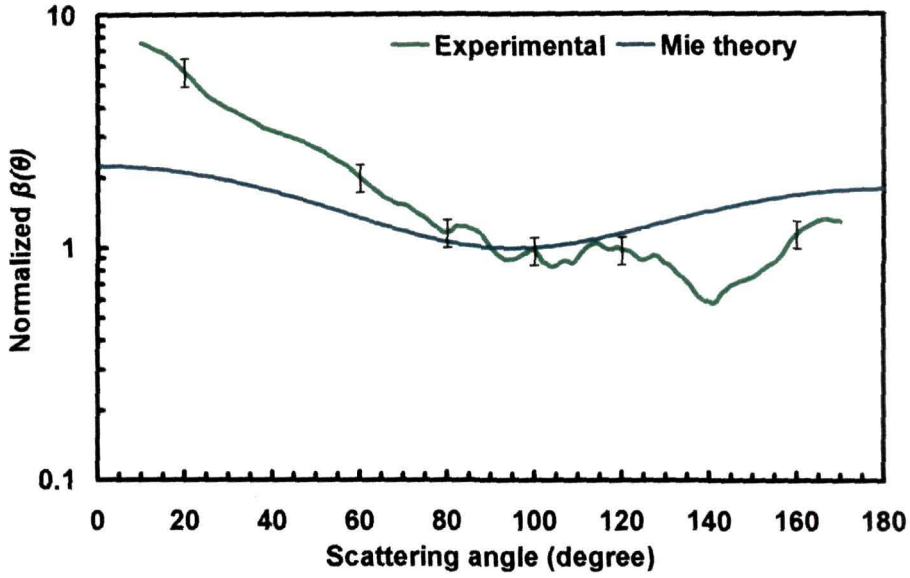

Figure 4.21(a). Measured volume scattering function, $\beta(\theta)$ of ZnS nanoparticles at 543.5 nm incident wavelength is denoted by green line. The comparative Mie curve is denoted by blue line.

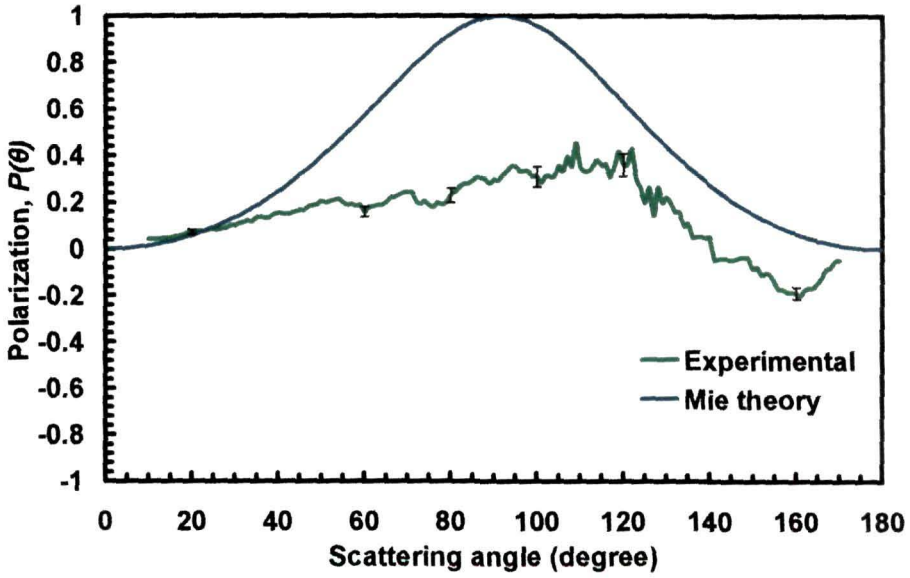

Figure 4.21(b). Measured degree of linear polarization, $P(\theta)$ of ZnS nanoparticles at 543.5 nm incident wavelength is denoted by green line. The comparative Mie curve is denoted by blue line.

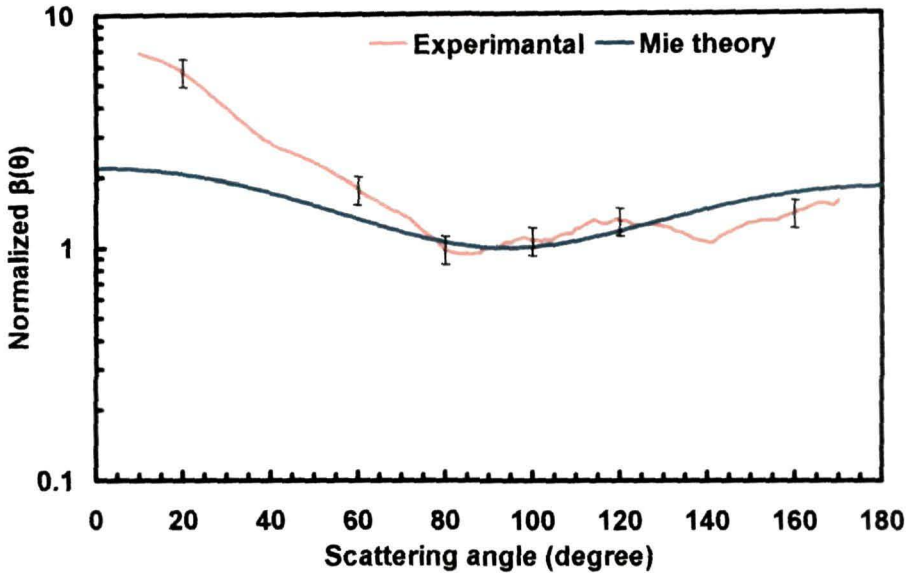

Figure 4.21(c). Measured phase function, $\beta(\theta)$ of ZnS nanoparticles at 594.5 nm incident wavelength is denoted by orange line. The comparative Mie curve is denoted by blue line.

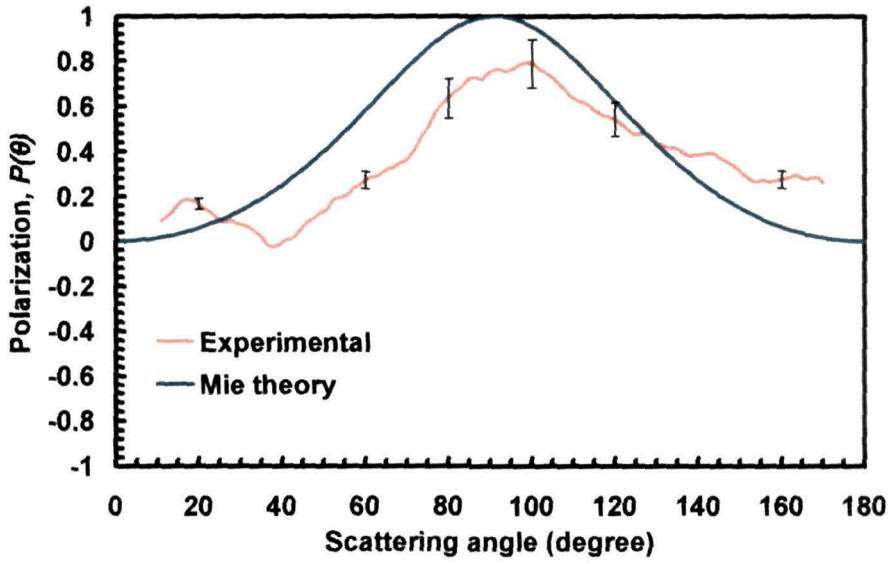

Figure 4.21(d). Measured degree of linear polarization, $P(\theta)$ of ZnS nanoparticles at 594.5 nm incident wavelength is denoted by orange line. The comparative Mie curve is denoted by blue line.

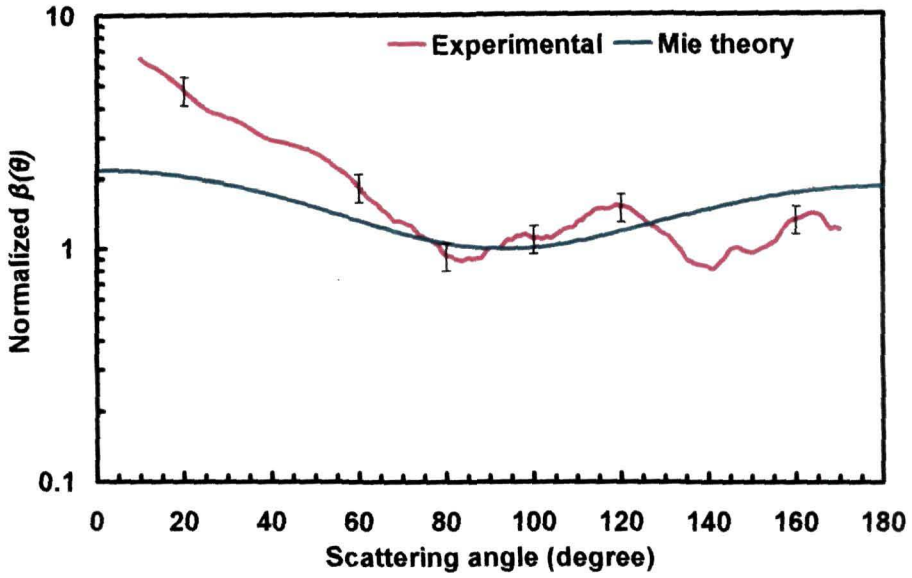

Figure 4.21(e). Measured phase function, $\beta(\theta)$ of ZnS nanoparticles at 632.8 nm incident wavelength is denoted by red line. The comparative Mie curve is denoted by blue line.

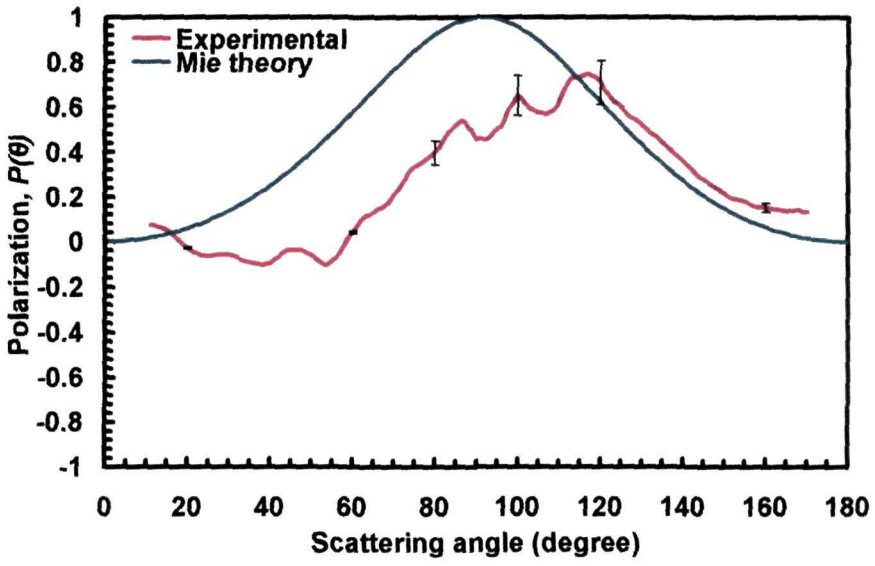

Figure 4.21(f). Measured degree of linear polarization, $P(\theta)$ of ZnS nanoparticles at 632.8 nm incident wavelength is denoted by red line. The comparative Mie curve is denoted by blue line.

4.4.2 Experiments on bentonite clay particles

Bentonite clay particles have their origin as volcanic ash and tuffs and its main meneral constituent is montmorillonite [351, 352]. As bentonite clay particles are classified under 'silicates' [351, 353], light scattering properties of such particles has importance in astrophysics because silicate is one of the major constituents of interplanetary and cometary dust [21, 307, 355]. The aim of the present work was to report an experiment that measures the volume scattering function, $\beta(\theta)$ and degree of linear polarization, $P(\theta)$ of bentonite clay particles embedded in transparent cylindrical thermosetting epoxy matrix.

Scanning electron microscopy (SEM) and transmission electron microscopy (TEM) was employed to observe the physical morphology of the clay particles. Figure 4.22(a) shows the SEM image of bentonite clay particles before they were embedded in the polymer matrix. The TEM image in figure 4.22(b) shows that the shape of the clay particles embedded in the polymer matrix was highly nonspherical. On the other hand figure 4.22(c) shows the typical layered structure [351] of the individual clay particles appearing as dark lines. The measured size distribution of the clay particles are shown in figure 4.23. The optimum concentration of the clay particles for single scattering was found to be 2 wt% in the epoxy matrix (figure 4.24). The details of the techniques for embedding and obtaining uniform dispersion of the clay particles in the cylindrical polymer matrix are given in section 3.2.3.4.

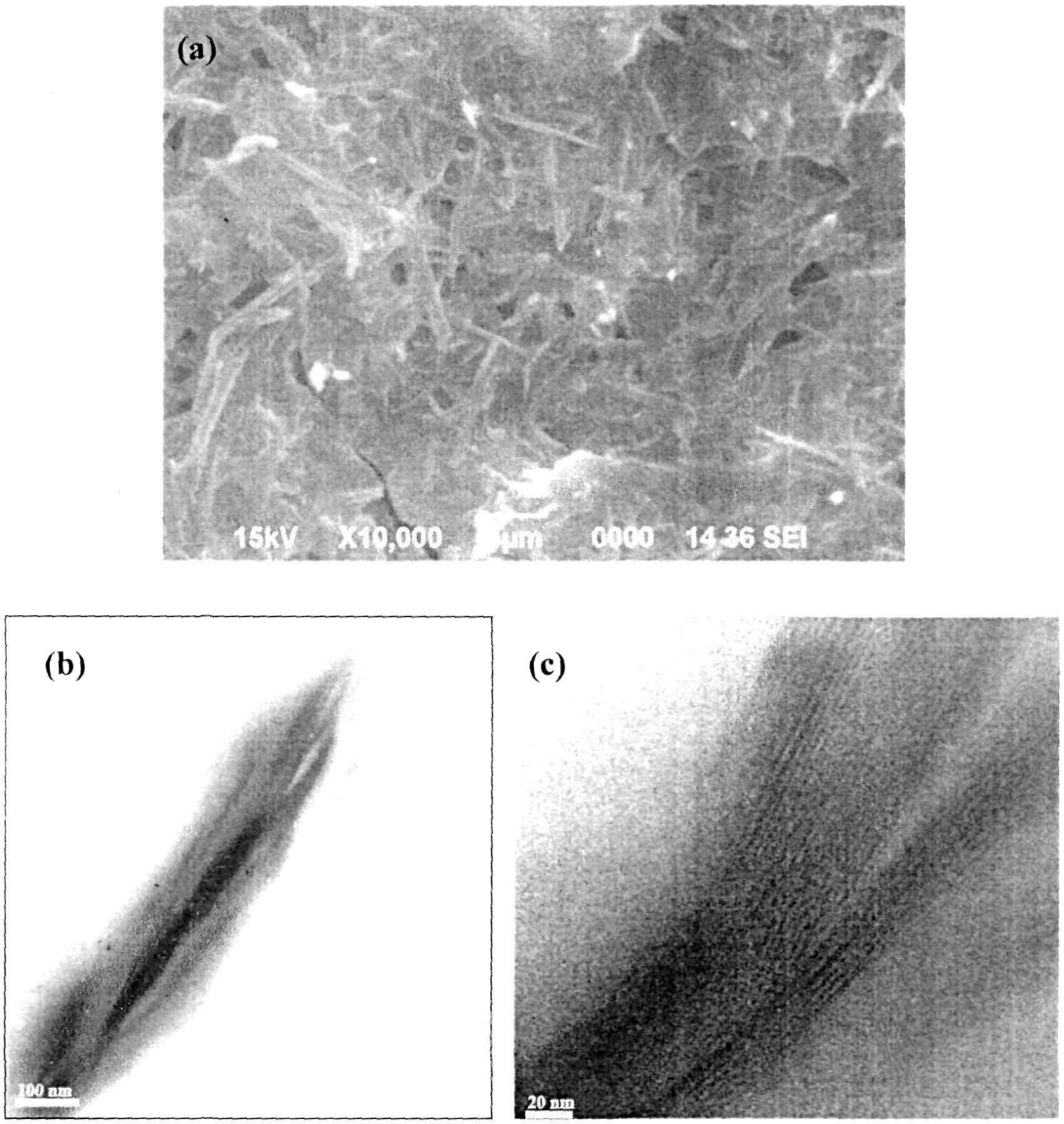

Figure 4.22. (a) SEM image of bentonite clay particles (b) TEM image of a bentonite clay particle dispersed in the polymer matrix, (c) High resolution TEM image showing the typical layered structure (appearing as dark lines) of the bentonite clay particles.

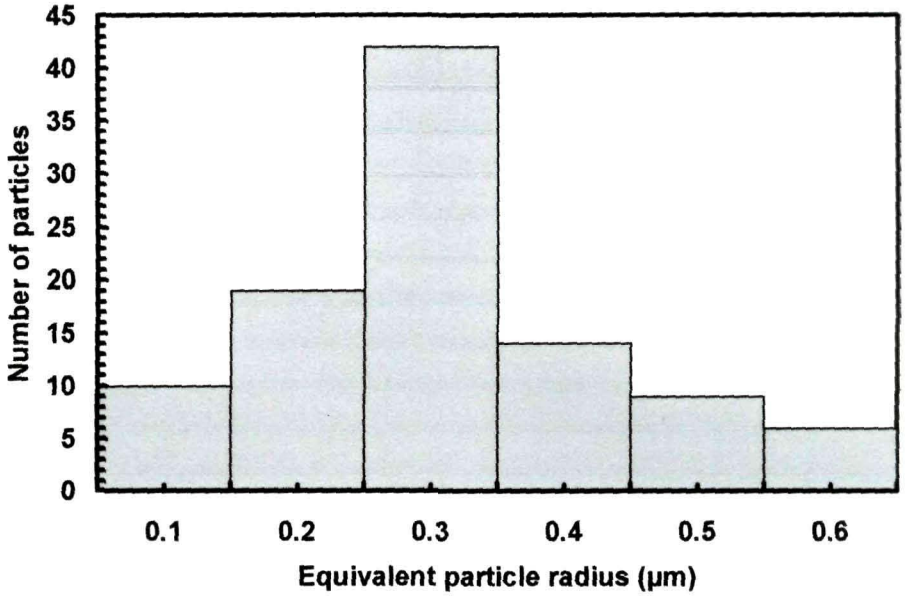

Figure 4.23. Measured size distribution of the clay particles.

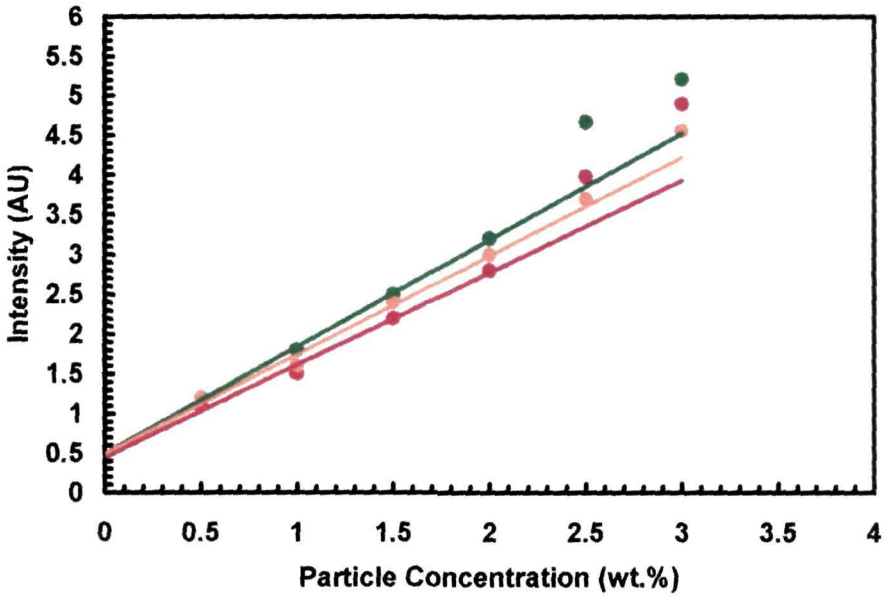

Figure 4.24. Graph for the scattered intensity for unpolarized incident light versus concentration of bentonite clay particles for a fixed position of the detector at 10^0. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

The experimental results of the volume scattering function $\beta(\theta)$ and degree of linear polarization $P(\theta)$ of bentonite clay particles at 543.5 nm, 594.5 nm and 632.8 nm incident laser wavelengths are presented in this section. Figures 4.25(a), 4.25(c) and 4.25(e) present the measured $\beta(\theta)$of the bentonite clay particles at 543.5, 594.5 and 632.8 nm incident wavelengths. The degree of linear polarization, $P(\theta)$ are shown in figures 4.25(b), 4.25(d) and 4.25(f). From the light scattering measurements it was observed that the volume scattering function, $\beta(\theta)$contains sharp forward scattering peaks followed by an almost flat back scattering response at 594.5 nm and 632.8 nm incident wavelengths. However at 543.5 nm incident wavelength the value of $\beta(\theta)$ was found to be increased significantly at large scattering angles (>150°). As the plot for $\beta(\theta)$ was smooth over the entire scattering angle range, it could be deduced that the clay particles in the scattering volume possessed a wide size distribution. The degree of linear polarization $P(\theta)$ was found to be positive over most of the scattering angles for all the incident wavelengths while it went slightly towards negative over 140°. The angular positions of maxima were almost same (at around 55°) for all the incident wavelengths. However considerable difference in the intensities of the maxima and minima were observed. Highest positive polarization of +0.43 (equivalent percent of polarization ~43%) was observed at 543.5 nm incident wavelength whereas the measured lowest negative polarization of -0.21 (equivalent percent of polarization ~21%) was obtained at 632.8 nm incident wavelength.

Table 4.7. Important parameters for T-matrix calculation for bentonite clay particles.

PARAMETERS	VALUE
Size distribution	Log normal
Shape	Cylindrical
Axial ratio, A/B	5
Modal equivalent radius, r of the clay particles (in μm)	0.3
Lowest equivalent particle radius (in μm)	0.2
Highest equivalent particle radius (in μm)	0.5
Variance, σ^2	2
Incident light wavelengths in nanometers	543.5, 594.5 and 632.8
Refractive index of bentonite	1.65 + i0.00
Environment refractive index	1.55 + i0.00
Accuracy of T-matrix calculation	0.008

For comparison purpose the measurements were correlated with T-matrix calculations for cylindrical shapes using the analytical software TUSCAT. The estimated parameters are given in table 4.7. In case of $\beta(\theta)$ both the measured and calculated values were normalized to 1 at 90°. Although the theoretical results calculated by estimating the required input parameters did not mimic the experimental results as can be observed from the figures, the deviation of $\beta(\theta)$ was within acceptable limits as compared to that of $P(\theta)$. Moreover the presence of clay particles of shapes and sizes different from the estimated values in the scattering volume was another reason for the observed discrepancy. Therefore it is important to understand the morphology of the scattering particles to a greater extent for accurate prediction of their scattering properties.

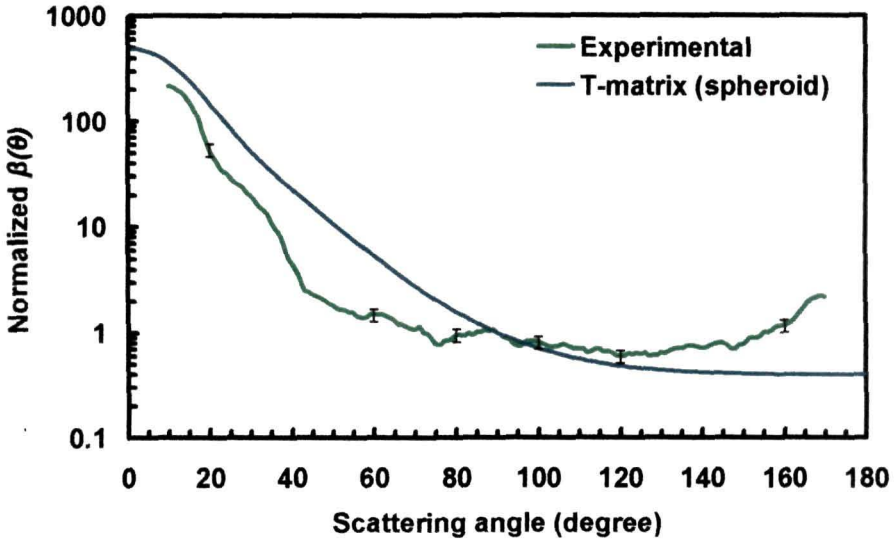

Figure 4.25(a). Measured phase function, $\beta(\theta)$ of bentonite clay particles at 543.5 nm incident wavelength is denoted by green line. The comparative T-matrix curve is denoted by blue line.

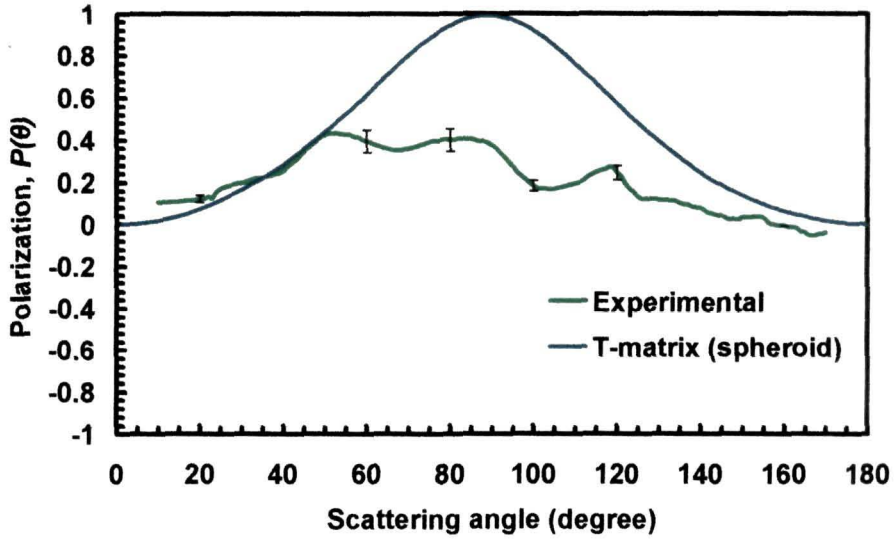

Figure 4.25(b). Measured degree of linear polarization, $P(\theta)$ of bentonite clay particles at 543.5 nm incident wavelength is denoted by green line. The comparative T-matrix curve is denoted by blue line.

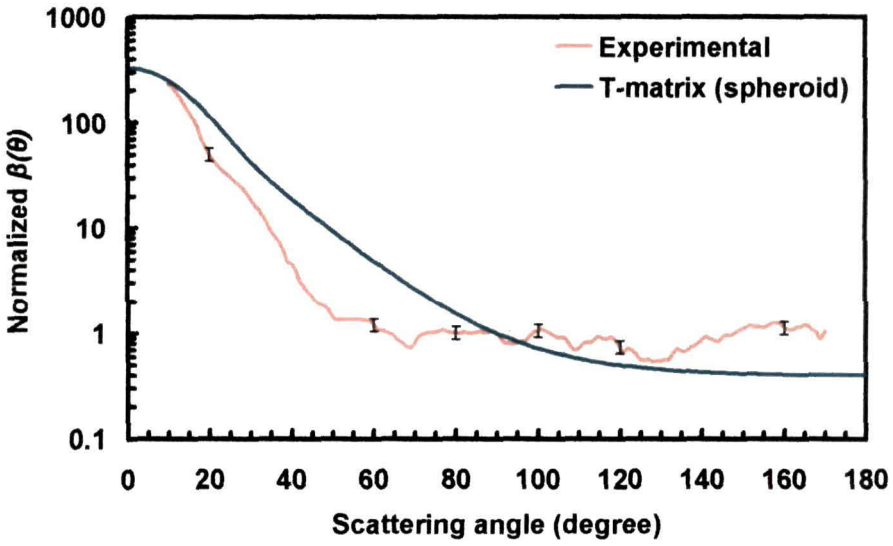

Figure 4.25(c). Measured phase function, $\beta(\theta)$ of bentonite clay particles at 594.5 nm incident wavelength is denoted by orange line. The comparative T-matrix curve is denoted by blue line.

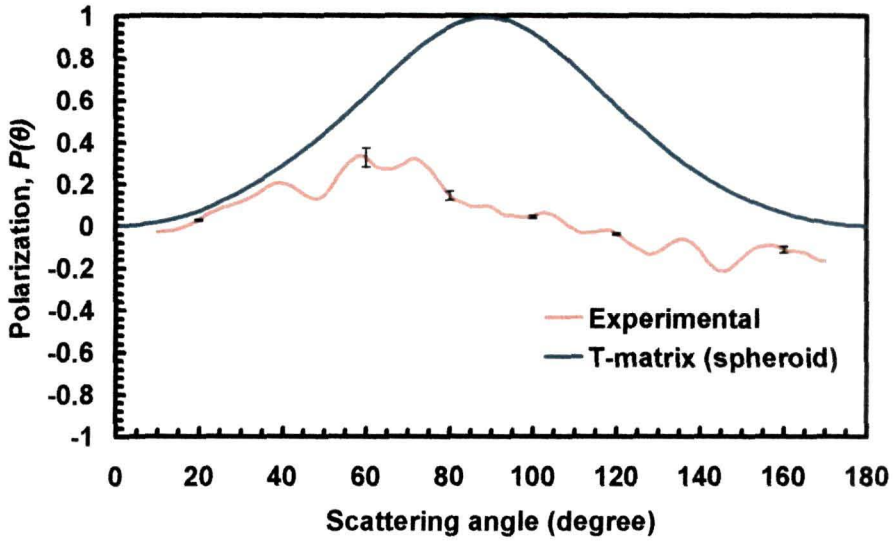

Figure 4.25(d). Measured degree of linear polarization, $P(\theta)$ of bentonite clay particles at 594.5 nm incident wavelength is denoted by orange line. The comparative T-matrix curve is denoted by blue line.

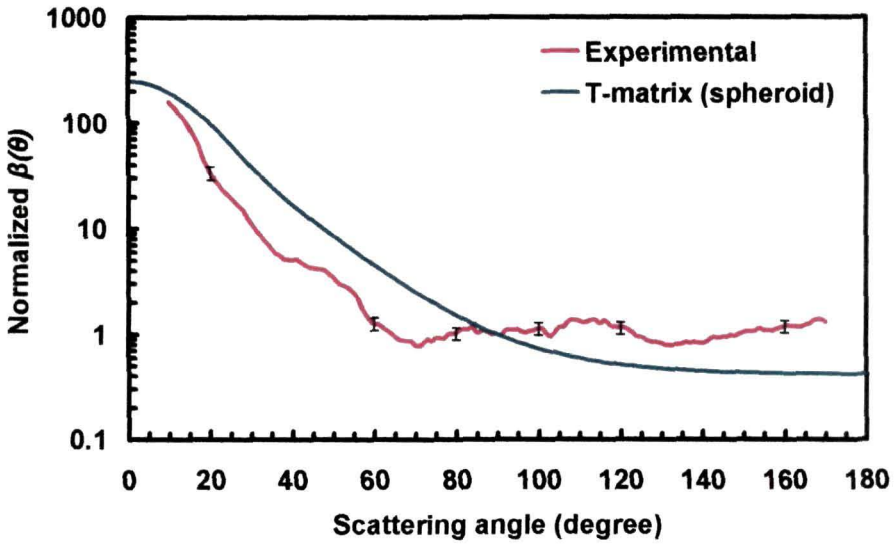

Figure 4.25(e). Measured phase function, $\beta(\theta)$ of bentonite clay particles at 632.8 nm incident wavelength is denoted by red line. The comparative T-matrix curve is denoted by solid blue line.

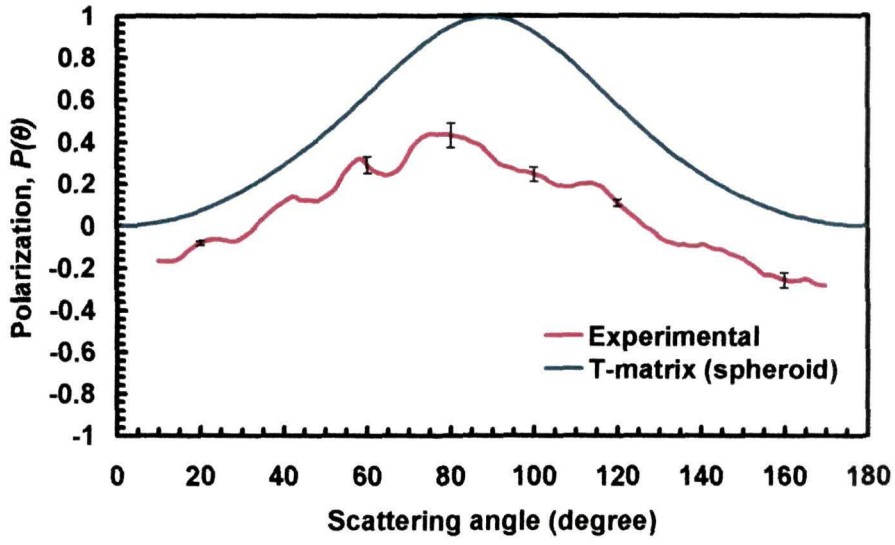

Figure 4.25(f). Measured degree of linear polarization, $P(\theta)$ of bentonite clay particles at 632.8 incident wavelength is denoted by red line. The comparative T-matrix curve is denoted by blue line.

In this section we presented the use of a transparent cylindrical polymer matrix to embed the bentonite clay particles in front of the laser beam. The

simplicity and the ease of fabrication of this type of polymer matrix make itself a promising sample holding arrangement for light scattering studies. It can be specifically applied for the light scattering measurements where the sample density is very low and experimental readings need to be taken for a large number of times.

4.5Case III: Investigations on powder samples by using a nebulizer

4.5.1 Experiments on ice analogue crystals

A number of different experimental works have been done in the past to measure the light scattering properties of ice crystals [58 - 61, 95, 282]. Knowledge about the light scattering properties of cirrus clouds which mainly consists of ice crystals is very important for precise understanding of their impact on Earth's radiation budget and is needed in efficient numerical models of climate prediction and change [283] in the Earth-atmosphere system. Although the ice crystals occurring in the Earth's atmosphere mostly consists of basic hexagonal structures which produces spectacular scattering maxima known as halos [284], several *in situ* observations have confirmed the presence of quasi-spherical and other faceted nonspherical shapes like columns, plates, bullet rosettes, dendrites etc. [284 - 286].

The laboratory measurements on real ice crystals are very complicated as the ice particles have to be created in cloud chambers at very low temperatures where the exact crystal size, morphology and position of the particles cannot be controlled for which the accuracy of the light scattering models developed on the basis of these measurements decreases significantly. Consequently sometimes the ground based and satellite retrieval algorithms may have large uncertainties [287, 288]. To overcome such difficulties associated with the real ice crystals, in this work, we used a type of material which is stable at room temperature having refractive index very close to ice at visible wavelengths (~1.31) and crystallize to shapes and produce halos similar to ice [278, 279].

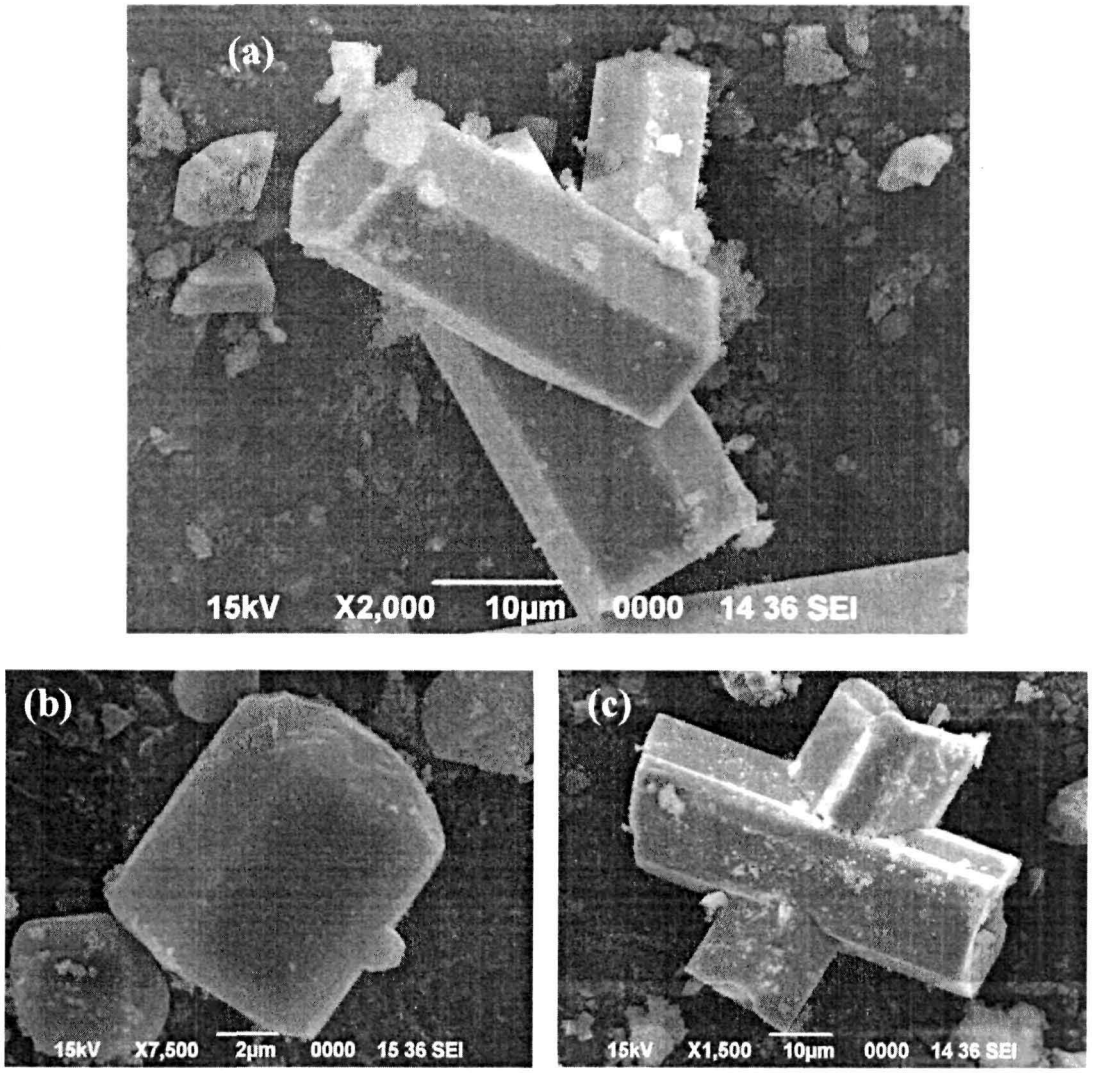

Figure 4.26. Scanning Electron Microscopy images of (a), (b) hexagonal ice analogue crystals. Figure (c) shows a deformed shaped ice analogue crystal.

The icelike crystals were grown in solution by using the method described by Ulanowski et.al. [278, 279, 289]. A small amount of saturated solution of sodium silicofluoride (Na_2SiF_6) or sodium hexafluorosilicate (Loba Chemie Pvt. Ltd., Mumbai, India) was poured over a glass Petri dish and allowed to dry under laminar flow at room temperature (~ 25^0 C). Special caution was taken in the preparation process of the crystals because sodium silicofluoride is toxic and cause severe irritation of eyes and mucous membranes [289, 290]. The dried ice analogue samples in powder form were collected and the surface morphology

was studied under scanning electron microscopy (SEM) both before and after the measurement. Formation of ice analogue crystals of hexagonal shapes were confirmed from SEM images as shown in the figure 4.26. The size distribution of the ice analogue crystals was analyzed from several SEM images and is shown in figure 4.27.

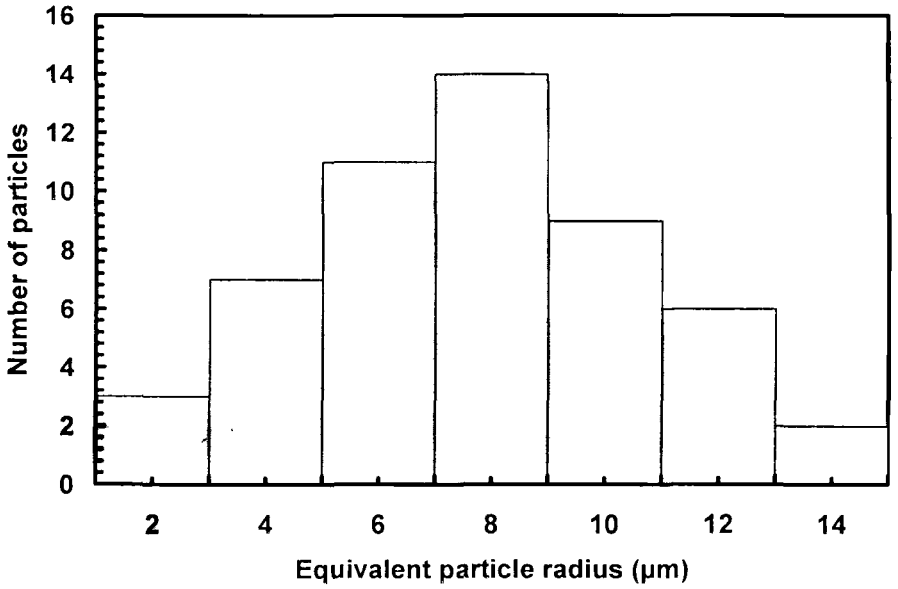

Figure 4.27. Measured size distribution of the ice analogue crystals.

Usually studies of crystal properties inside cirrus clouds implicitly assume a random orientation for the particles [291] which is also more relevant to atmospheric ice [279]. In the present work also light scattering measurements were performed for randomly oriented crystals obtained by using the indigenously designed and fabricated nebulizer that sprayed the icelike particles at the scattering centre as a stream of flowing particles. The optimum sample concentration for single scattering was estimated to be 7.5 mg/L (approximately) at the three incident wavelengths as shown in figure 4.28.

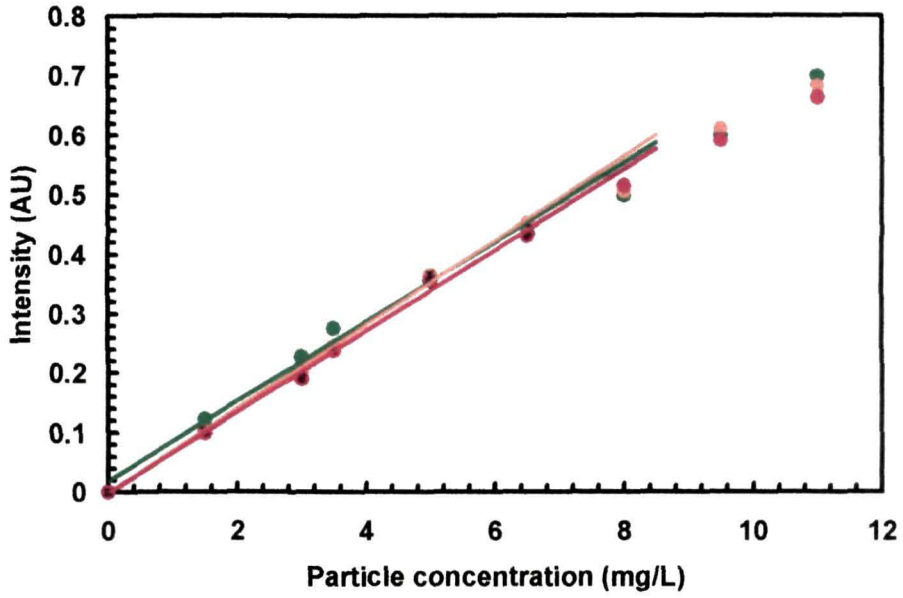

Figure 4.28. Graph for the scattered intensity for unpolarized incident light versus concentration of ice analogue crystals for a fixed position of the detector at 10^0. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

Figure 4.29(a) - figure 4.29(f) show the results of the measurements of volume scattering, $\beta(\theta)$ function and degree of linear polarization, $P(\theta)$ of ice analogue crystal from 10^0 to 170^0 scattering angles. The measured scattering patterns showed several ripple structures which are due to the presence of ice analogues of different nonspherical shapes like hexagonal, cylindrical and aggregates. It has also been observed from the graphs that measured $\beta(\theta)$ is approximately 500 times more intense in the forward direction (near 10^0) as compared to that in the side and backscattered directions. Most importantly, the presence of halo peaks around 22^0 at all the three incident wavelengths as shown in figure 4.29(a), figure 4.29(c) and figure 4.29(e) indicated the presence of hexagonal as well as bullet rosette and column shaped crystals [289]. Another faint halo peak around 46^0 was also observed from the graphs. At the side scattering and back scattering angles, less prominent scattering peaks were observed. It is very interesting to note that previous experiments as well as

calculations of the volume scattering function or phase function of cirrus cloud also reported that polycrystalline or randomly oriented hexagonal shaped ice crystals produce a featureless phase function at larger angles [279, 292]. The measured degree of linear polarization, $P(\theta)$ for ice analogue crystals showed an oscillating behavior having positive polarization at most of the scattering angle range. However for all the three incident laser wavelengths, negative polarization was observed at scattering angles approximately greater than 140^0. At 543.5 nm laser wavelength highest positive polarization of 0.30 (equivalent percent of polarization ~30%) was observed. The measured lowest negative polarization was 0.091 (equivalent percent of polarization ~9.1%) and was obtained at the same incident wavelength. The polarization is very sensitive to shapes of the scattering particles and the comparison of the measured polarization with the available literature reveals the higher presence of icelike crystals of hexagonal plate shapes and column shapes as compared to other shapes in the scattering volume [293].

For validating the procedure the experimental results were compared with the results drawn from Mie theory by approximating the hexagonal icelike particles to a sphere analogue of same volume, V, to a sphere analogue of same surface area, A and a collection of spheres as an analogue of equal volume to surface area ratio, V/A. The third model, as described by Grenfell et al [294 - 296] represents a nonspherical particle by a cloud of spherical particles having the same volume to surface area ratio as that of the nonspherical particle. This model is described in detail in APPENDIX IV. We opted for these assumptions or models because of two reasons. First the inadequacy of Mie theory for nonspherical shapes restricted its direct application for computing the light scattering properties of the hexagonal ice crystals. Second, the T-matrix code described in chapter II, which is the basic program running behind the analytical software TUSCAT for nonspherical particles, gave accurate results for spheroids and cylinders of average size parameter not greater than 20. As such it was not

Table 4.8. Estimated parameters for Mie calculations

Equivalent sphere approximation parameters		**Estimated values**
Equivalent volume, V sphere	Size distribution	Normal (Gaussian)
	Minimum radius, rmin	1.0
	Maximum radius, rmax	12.0
	Modal radius, r in micrometers	7.98
	Deviation, σ	1
Equivalent surface area, A sphere	Size distribution	Normal (Gaussian)
	Minimum radius, rmin	1.0
	Maximum radius, rmax	12.0
	Modal radius, r in micrometers	9.41
	Deviation, σ	1
Equivalent volume to surface area ratio, V / A sphere	Size distribution	Normal (Gaussian)
	Minimum radius, rmin	1.0
	Maximum radius, rmax	12.0
	Modal radius, r in micrometers	6.1168 (ns/n=2.37)
	Deviation, σ	1

used for computations of the light scattering properties of the ice crystals with large size parameter > 100. However geometrical optics approximation (GOA) or anomalous diffraction theory (ADT) could have been applied for the comparison of the experimental results for hexagonal shapes, but it is beyond the scope of our work presented in this thesis. Table 4.8 depicts the estimated parameters for the Mie calculations. Although the theoretical results calculated by taking equivalent spheres could not mimic the measurements at all the three incident wavelengths as can be observed from the figures 4.29(a) - 4.29(f), the deviation of $\beta(\theta)$ is within acceptable limits as compared to that of degree of linear

polarization, $P(\theta)$. The plots for experimental $P(\theta)$ were smooth as compared to the theoretical ones. This significant smoothing may also be due to the variation of shapes of the icelike crystals in the scattering volume (presence of other shapes like bullet rosette, droxtals, dendrites, aggregates etc.) and the presence of air bubbles or inhomogeneity in the crystals [293]. Overall it can be summarized that more detailed understanding of the microphysical properties of the scatterers in the scattering volume is essential for proper prediction of their scattering properties.

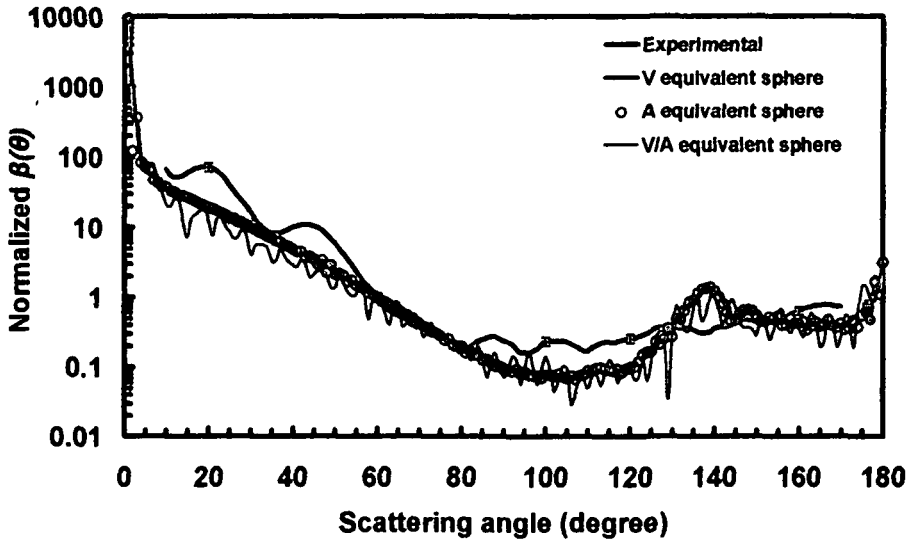

Figure 4.29(a). Measured volume scattering function, $\beta(\theta)$ of icelike particles at 534.5 nm incident wavelength is denoted by green line. The Mie curves for volume, surface area and volume to surface area ratio equivalent sphere radii are denoted by pink line, brown circles and blue line respectively.

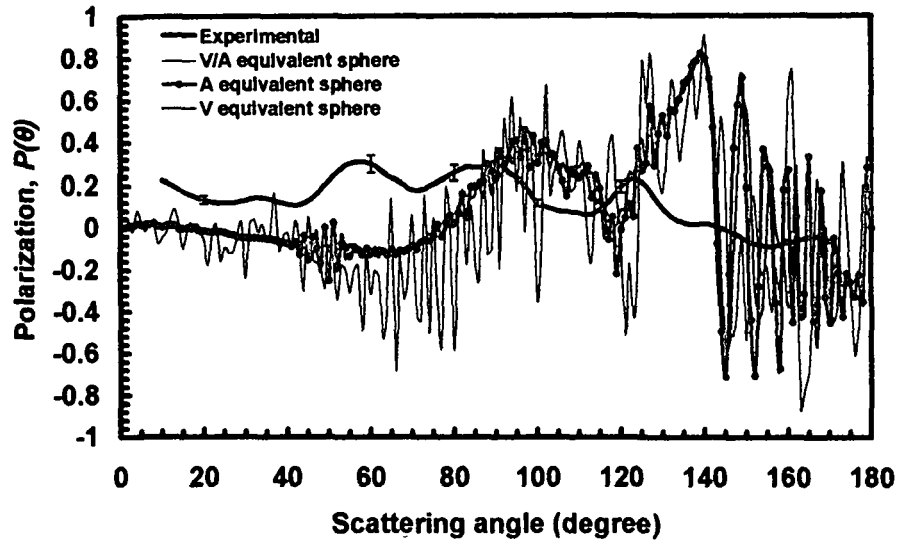

Figure 4.29(b). Measured degree of linear polarization, $P(\theta)$ of icelike particles at 534.5 nm incident wavelength is denoted by green line. The Mie curves for volume, surface area and volume to surface area ratio equivalent sphere radii are denoted by pink line, brown line with circles and blue line respectively.

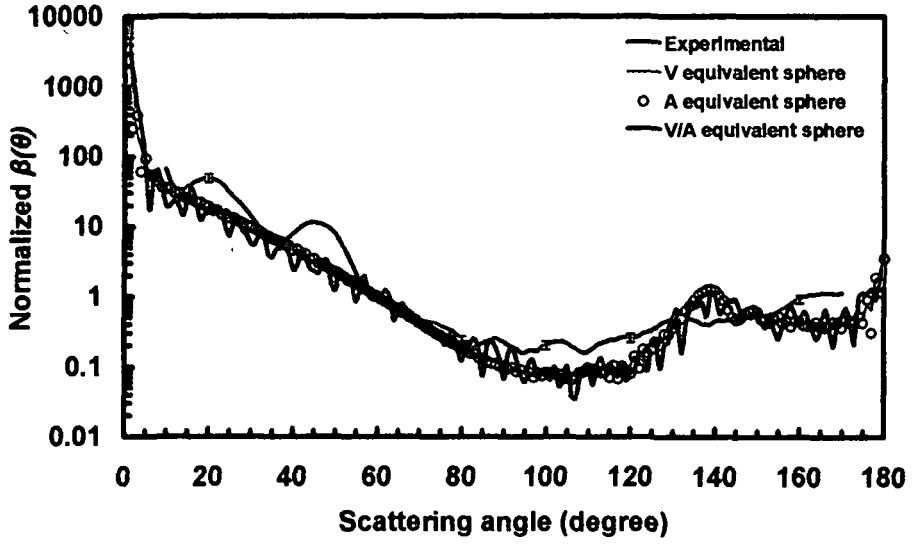

Figure 4.29(c). Measured volume scattering function, $\beta(\theta)$ of icelike particles at 594.5 nm incident wavelength is denoted by orange circles. The Mie curves for volume, surface area and volume to surface area ratio equivalent sphere radii are denoted by pink triangles, solid brown line with filled squares and solid blue line respectively.

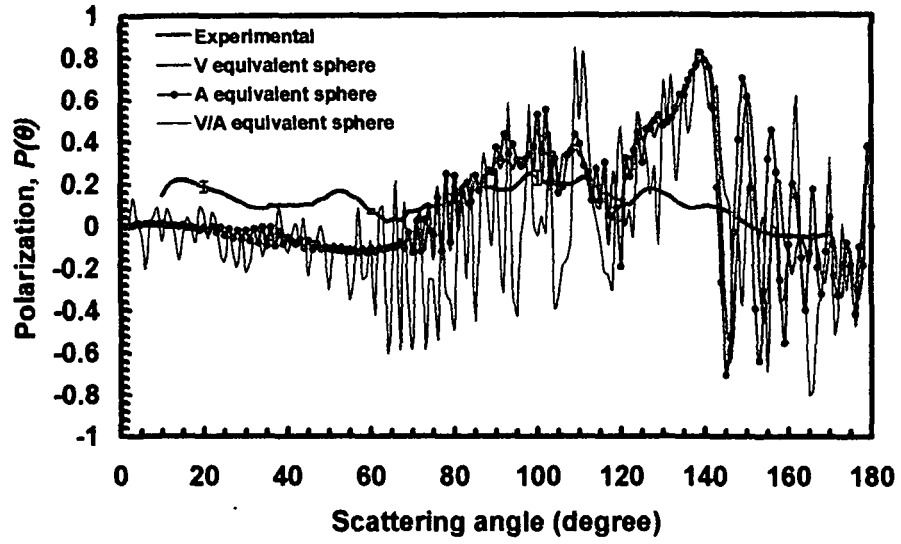

Figure 4.29(d). Measured degree of linear polarization, $P(\theta)$ of icelike particles at 594.5 nm incident wavelength is denoted by orange circles. The Mie curves for volume, surface area and volume to surface area ratio equivalent sphere radii are denoted by pink line, brown line with circles and blue line respectively.

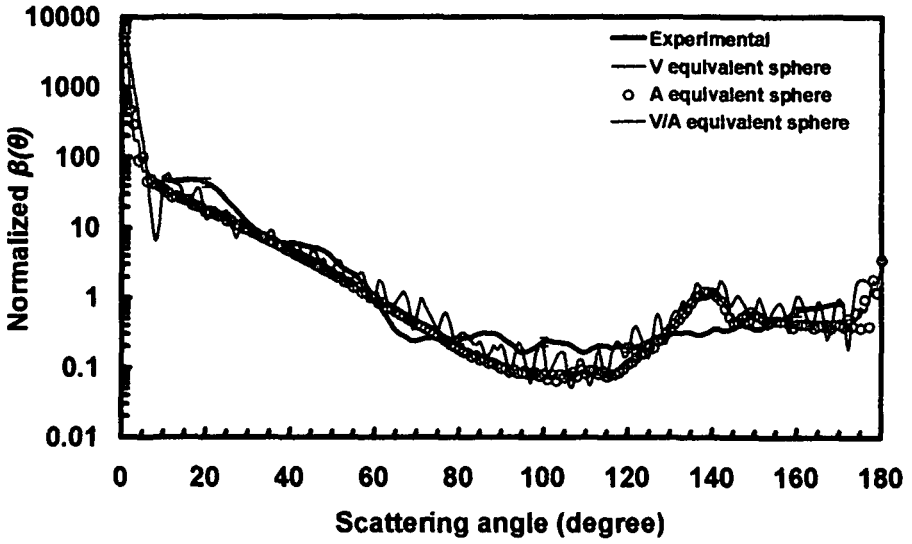

Figure 4.29(e). Measured volume scattering function, $\beta(\theta)$ of icelike particles at 632.8 nm incident wavelength is denoted by red line. The Mie curves for volume, surface area and volume to surface area ratio equivalent sphere radii are denoted by pink line, brown circles and blue line respectively.

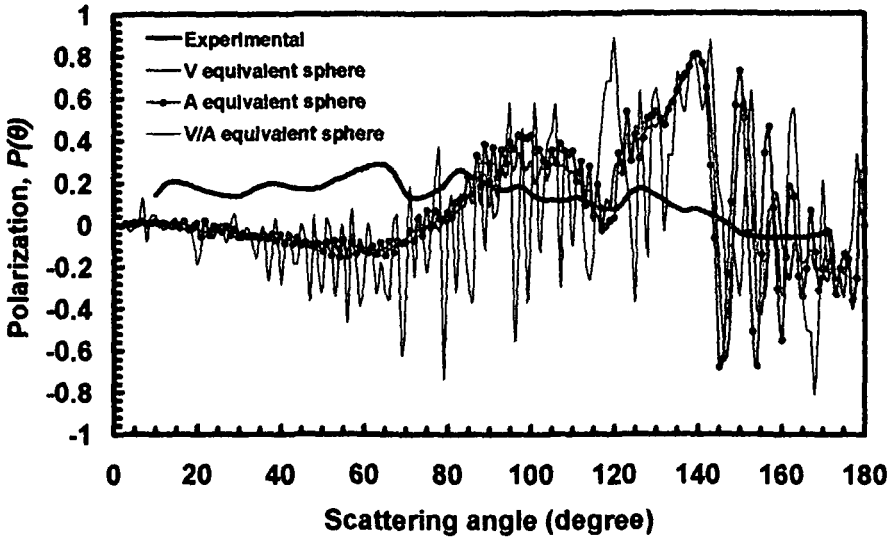

Figure 4.29(f). Measured degree of linear polarization, $P(\theta)$ of icelike particles at 632.8 nm incident wavelength is denoted by red circles. The Mie curves for volume, surface area and volume to surface area ratio equivalent sphere radii are denoted by pink line, brown line with circles and blue line respectively.

4.5.2 Experiments on graphite crystals

Different types of dust particles are present everywhere in the solar system, cometary comae and tail, interstellar dust clouds, circumplanetary dust rings, asteroidal atmospheres and aerosols of other planetary atmospheres. The *in situ* sampling of the cometary dust composition conducted by Cometary and Interstellar Dust Analyzer (CIDA) on the Halley missions revealed that the cometary particulates not only contains rock like materials (Mg, Al, Si, Ca, Fe and O) but also an abundance of S, C and N [297, 298]. Observed interstellar extinction and polarization indicated the presence of amorphous silicate, amorphous carbon, graphite, carbonates, metal oxide grains, amorphous ice particles and nanodiamonds [299 - 303] in the interstellar medium which was further evidenced indirectly by the observations of absorption line with the Hubble space telescope [304]. Such dust particles can be compact (asteroidal origin) or fluffy (cometary origin) aggregates of submicron to micron sized grains which can be sometimes embedded in an absorbing mantle [305 - 307].

These particles act as the heterogeneous media to scatter solar or steller light which is unpolarized in nature [308, 309]. As usual the properties of the scattered light depends on the shape or dispersion of shapes, sizes or dispersion of sizes, porosity, composition and refractive index of the scatterers. Evidently the proper analysis of light scattered by dust particles in a laboratory will help in remote detection and retrieval of information of the physical properties [38, 310]. It is worth mentioning that measurement of the degree of linear polarization can be used to estimate parameters like size, porosity and roughness of the dust particles [311].

A number of theoretical models have been proposed for the modeling of the systems containing circumsteller or interstellar dust by comparing with the observed interstellar extinction curves. However such numerical approaches alone are not sufficient to reproduce the scattering behavior of the irregular shaped dust particles [312]. Combined laboratory measurements and theoretical simulations on the other hand act as an indispensable reference tool for the accurate interpretation of different physical and optical properties of interstellar dust particles. As far as the optical constants are concerned, the materials with astrophysical importance can be broadly divided into broad classes: non absorbing particles and absorbing particles with a large value of real and imaginary part of the refractive index [305, 313]. Here, in the present work, we choose graphite as the scattering sample to simulate the real absorbing astrophysical dust particles. Graphite is one of the most common allotropes of carbon [314] where the C atoms are bound to one another in hexagonal sheets [301]. As a mineral, graphite is one of the softest, a good conductor of electricity and a very good solid lubricant [315].

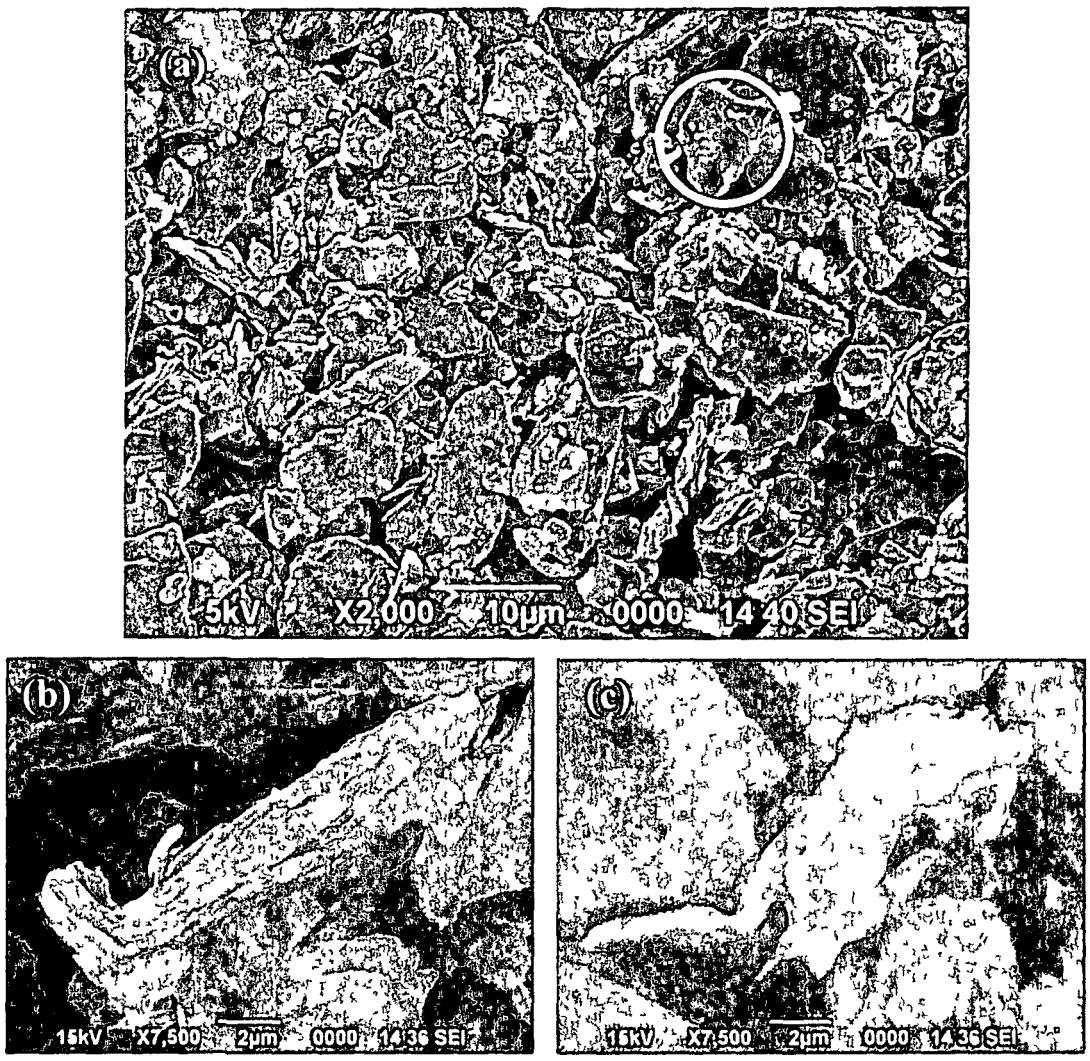

Figure 4 30 (a), (b), (c) Scanning Electron Microscopy (SEM) images of nonspherical graphite particles The white circle in (a) represents an approximated particle having equivalent radius of the circle Side view of a graphite particle is shown in (b) whereas (c) shows another particle of highly irregular shape

Surface morphology of the commercial graphite particles (Sigma Aldrich) was observed using a Variable Pressure Digital Scanning Electron Microscope (SEM) (model JEOLJSM6390LV) with Energy Dispersive X-ray Spectroscopy It was found that the dispersion of shapes and sizes of the graphite particles were very irregular (figure 4 30) The approximated size distribution of the graphite particles are shown in figure 4 31

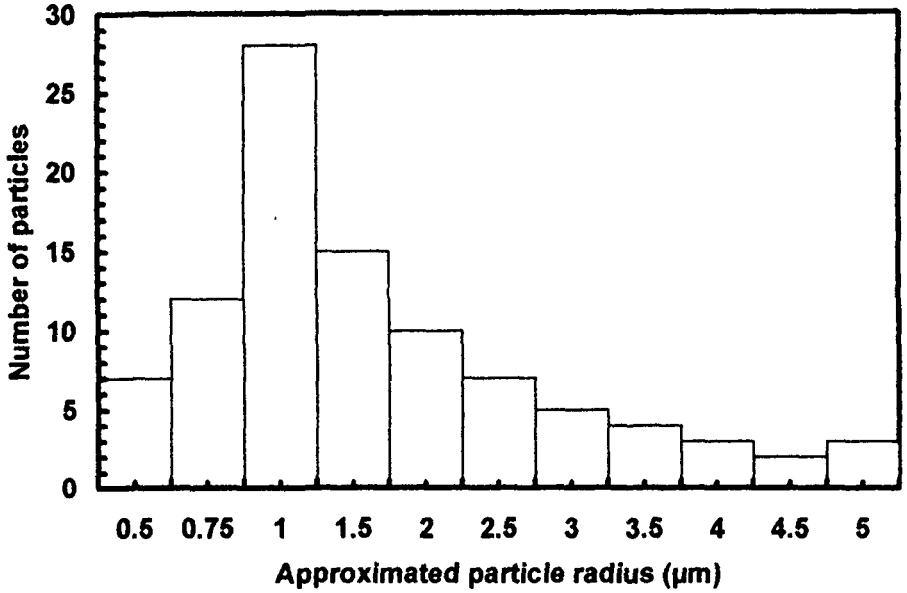

Figure 4.31. Approximated size distribution of the graphite particles.

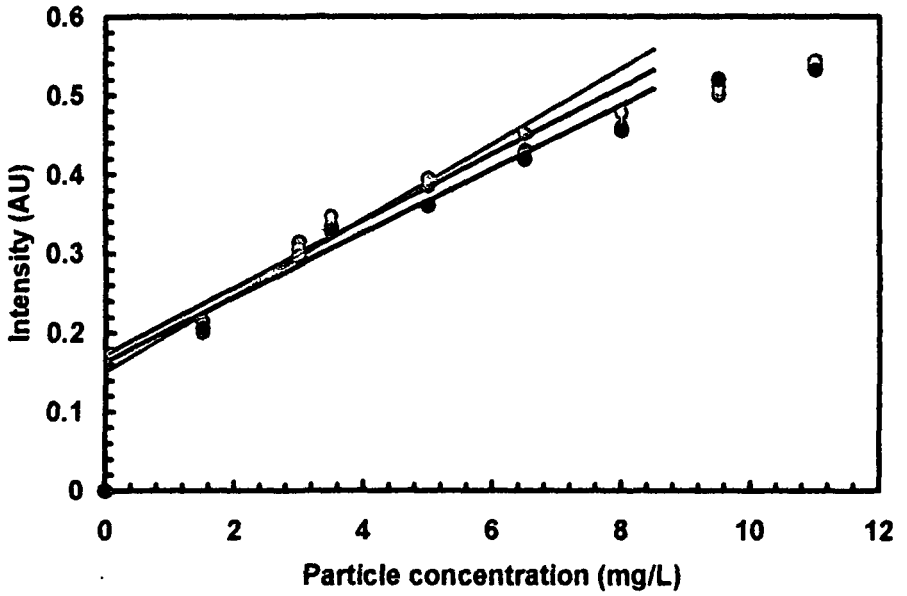

Figure 4.32. Graph for the scattered intensity light for unpolarized light versus concentration of graphite particles for a fixed position of the detector at 10^0. Results for 543.5 nm, 594.5 nm and 632.8 nm incident wavelengths are shown by green, orange and red circles respectively. The straight lines are fits to the initial linear regime.

The dust particles of interstellar space exhibit a random orientation [316, 328] under normal conditions. The present work describes the results of the light scattering measurements performed on randomly oriented graphite grains

obtained by spraying the grains into the scattering volume by using a nebulizer as described in section 3.2.3.2. The optimum sample concentration for single scattering were estimated to be 5 mg/L (approximately) at the three incident wavelengths as shown in figure 4.32. Graphs of the results of the measurements of volume scattering function, $\beta(\theta)$ and degree of linear polarization, $P(\theta)$ of graphite grains from 10° to 170° scattering angles are shown in figure 4.33 (a) - figure 4.33(f). It has been observed that $\beta(\theta)$ is less intense at the small angles (forward direction) at all the three incident wavelengths as compared to the higher angles (backscattering direction). The decrease of scattered intensity in the forward direction may be attributed to absorption of the incident radiation significantly by the graphite grains having very high value of imaginary refractive index. However sharper scattering patterns can be expected at lower angles (less than 10°), the measurement of which is beyond the capability of the present light scattering setup. The graphs at all the three incident wavelengths also show several side scattering peaks in the side scattering regions. A typical increase in $\beta(\theta)$ can be observed in the backscattering regions ($\theta > 140^0$).

The measured degree of linear polarization, $P(\theta)$ for graphite aggregates showed typical bell shaped behavior with the associated negative branch at small phase angles [305, 325 - 328]. For all the three incident laser wavelengths, negative polarization was observed at scattering angles below 25°. The polarization was found to be positive for the remaining phase angles. At 632.8 nm laser wavelength highest positive polarization of 0.54 (equivalent percent of polarization ~54%) was observed. The measured lowest negative polarization was 0.16 (equivalent percent of polarization ~16%) and it was obtained at 543.5 nm incident wavelength. Randomly oriented particles possess a polarization value of 0 (zero) at 180^0 for reasons of symmetry [313]. In our measurements, at the scattering angle of 170^0 we found the values of polarization to be 0.11, 0.07 and 0.04 at 543.5 nm, 594.5 nm and 632.8 nm laser wavelengths respectively. The results of the linear polarization ratio obtained in our measurements are found to

be consistent with other reported measurements on graphite particles [305, 313] and indicates the presence of agglomerated graphite grains with a large size distribution.

Table 4.9. Estimated parameter for Mie calculations for graphite grains

PARAMETERS	VALUE
Size distribution	Log normal
Modal radius (in μm)	1
Variance, σ^2	1
Minimum particle radius (in μm)	0.5
Maximum particle radius (in μm)	5
Incident light wavelengths in nanometers	543.5, 594.5 and 632.8
Environment refractive index	1.00 + i0.00000
Average refractive index of graphite at 543.5 nm	2.57+ i1.595

As the graphite particles were too irregular, correlation of the scattering results with Mie theory was attempted with estimated parameters as shown in table 4.9. Despite the consideration of a wide size distribution of equivalent spherical particles for Mie theory calculations, it was found that the deviation of the theoretical predictions from the experimental results was significantly large. This is due to the highly irregular shapes of the agglomerated graphite grains and the presence of porosity within the particles. As such it is necessary to understand the morphology of the structure to a greater extent and modify the existing theories to correlate the experimental results.

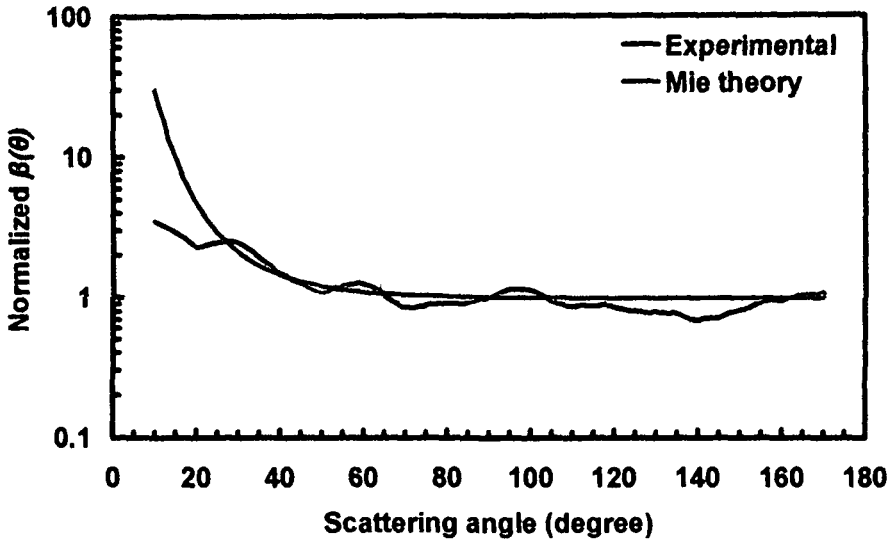

Figure 4.33(a). Measured volume scattering function, $\beta(\theta)$ of graphite particles at 543.5 nm incident wavelength is denoted by green line The comparative Mie curve is denoted by blue line.

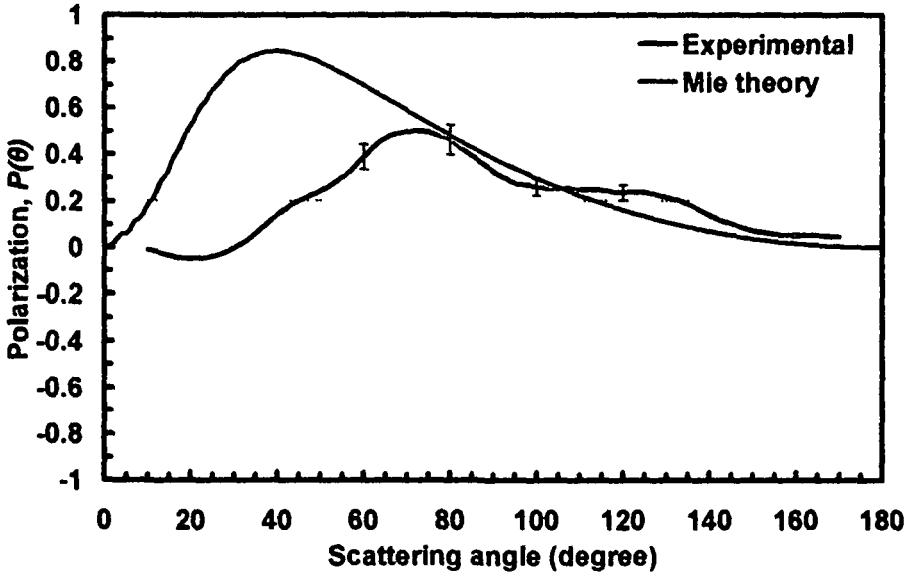

Figure 4.33(b). Measured degree of linear polarization, $P(\theta)$ of graphite particles at 543.5 nm incident wavelength is denoted by green line. The comparative Mie curve is denoted by blue line.

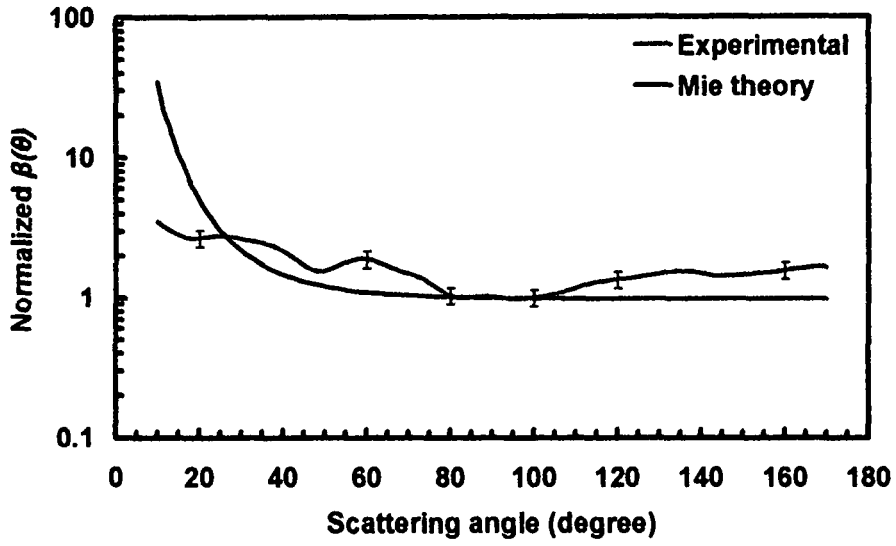

Figure 4.33(c). Measured volume scattering function, $\beta(\theta)$ of graphite particles at 594.5 nm incident wavelength is denoted by orange line. The comparative Mie curve is denoted by blue line.

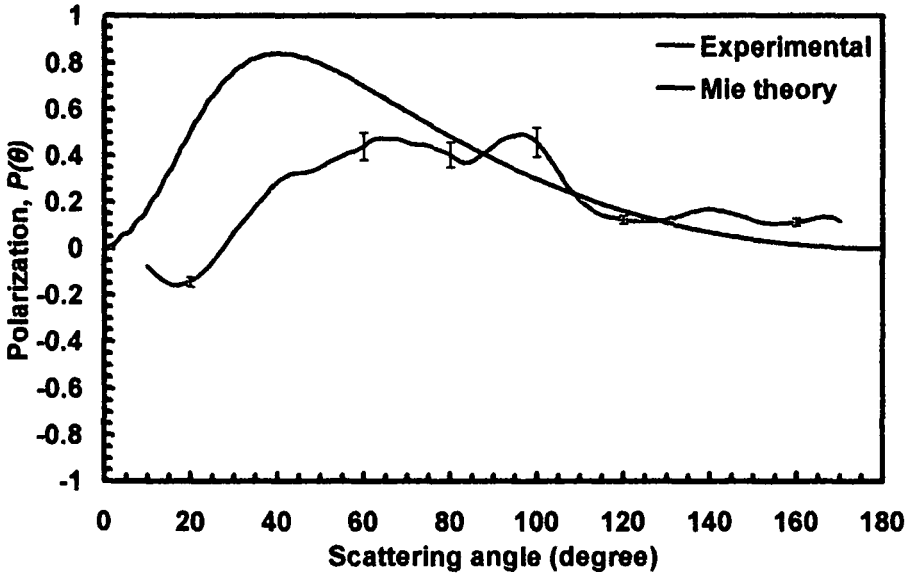

Figure 4.33(d). Measured degree of linear polarization, $P(\theta)$ of graphite particles at 594.5 nm incident wavelength is denoted by orange line. The comparative Mie curve is denoted by blue line.

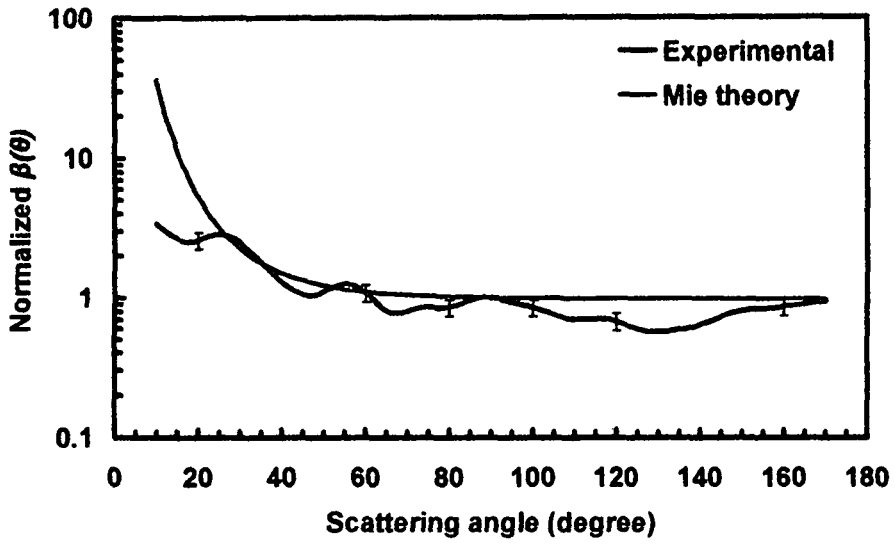

Figure 4.33(e). Measured volume scattering function, $\beta(\theta)$ of graphite particles at 632.8 nm incident wavelength is denoted by red line. The comparative Mie curve is denoted by blue line.

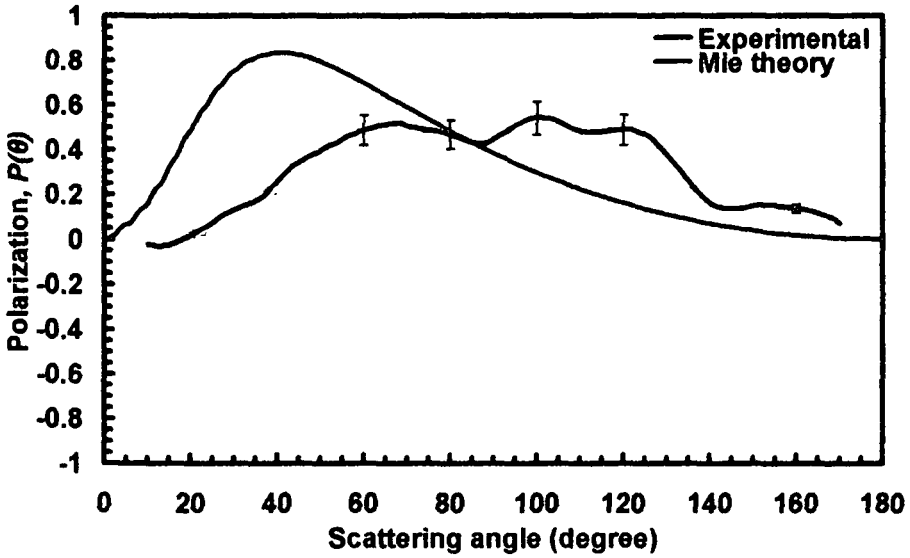

Figure 4.33(d). Measured degree of linear polarization, $P(\theta)$ of graphite particles at 632.8 nm incident wavelength is denoted by red line. The comparative Mie curve is denoted by blue line.

4.6 Error calculation

The instrumental error of the light scattering setup was determined from 100 sets of data obtained by repeating the experiment with a sample of polystyrene spheres of 0.30 μm radius. The instrumental error was found to be ±0.137 which is indicated by error bars in each of the plots presented in this entire thesis.

The root mean square error (RMSE) between the theoretical and experimental readings with the light scattering setup was calculated using the formula (equation 4.1) given below.

$$e_{rms} = \sqrt{\frac{e_1 + e_2 + \ldots\ldots\ldots + e_n}{n}} \qquad \textbf{4.1}$$

Where,

e = Theoretical value - Experimental value

and n = number of data points.

The values of the calculated RMSE are tabulated in table 4.10 for each of the samples used for the measurements. The observed errors as given by e_{rms} are not only due to the systematic error which were introduced in the measurement of $\beta(\theta)$ due to finite angular resolution, detector sensitivity as well as the measurement errors of the light scattering setup mentioned in section 3.4, but may also be on account of the limitations of the theories in terms of modeling the actual size, shape, size distribution and shape distribution of the particles under investigation. However the estimated RMSE values for water droplets and polystyrene microspheres found from our measurements given in table 4.10 are minimal and hence emphasizes the efficiency and reliability of the designed and fabricated light scattering setup.

Table 4.10. Table of root mean square error (RMSE) calculation for the measured volume scattering function, $\beta(\theta)$ and polarization, $P(\theta)$ for all the scattering samples

Parameter	Volume scattering function, $\beta(\theta)$			Polarization, $P(\theta)$		
	543.5 nm	594.5 nm	632.8 nm	543.5 nm	594.5 nm	632.8 nm
Water droplets	0.012	0.027	0.026	0.114	0.131	0.165
Polystyrene	0.020	0.018	0.036	0.088	0.119	0.0781
Titania	0.166	0.201	0.197	0.400	0.210	0.339
Diatom	0.228	0.255	0.264	0.392	0.448	0.444
Frustules	0.253	0.234	0.268	0.433	0.449	0.497
ZnS nanoparticle	0.415	0.392	0.399	0.385	0.214	0.325
Bentonite clay	0.037	0.063	0.075	0.354	0.508	0.417
Ice analogue crystals	0.111	0.067	0.102	0.394	0.340	0.371
Graphite	0.315	0.433	0.319	0.344	0.317	0.308

4.7 Comparison with reported works.

In comparison to the other light scattering instruments across the world, the present light scattering setup reported in this thesis is compact and versatile. The use of silicon photodetectors, low power laser source and minimum power consuming digital data recording unit to measure the volume scattering function, $\beta(\theta)$ and degree of linear polarization, $P(\theta)$ also makes it energy efficient. As described in the section 4.2 the present light scattering setup is accurate and reliable. Although the size, shape, porosity and other relevant parameters of the samples analyzed with our light scattering setup is quite different from the samples analyzed by other workers, the measurements of $\beta(\theta)$ and $P(\theta)$ is found to be in correspondence with reported works, especially for ice analogue

crystals and graphite particles. The design considerations discussed by Mishchenko, Hovenier and Travis [4] for measurement of scattering matrices are also in agreement with the light scattering setup reported here.

Finally in comparison to the reported works, our designed and fabricated light scattering setup reported in this thesis, in which

(a) very cheap and commercially available silicon photodetectors, which also consume very low power, are used in the detectors,

(b) synthesis of the design principles of a nephelometer is made,

(c) provision for measuring rapid changes in the medium under observation by incorporating fast electronic components has been made,

(d) different sample holding and presenting arrangements have been provided,

(e) original software for data acquisition has been developed to record data very quickly and store in user defined data files,

(f) techniques for error estimation and reduction have been incorporated,

(g) original software for data analysis with the capacity to yield some of the physical and optical parameters of the scattering particles has been developed and incorporated,

stands out as a relatively simpler, versatile and far more economic light scattering setup which is also reliable, fast and efficient.

CHAPTER V: Conclusion and Future Directions

5.1 Conclusion and future directions

The design considerations and fabrication details of a detector array incorporated light weight and versatile laboratory light scattering instrument have been described in detail in this thesis. By conducting test measurements with spherical polystyrene samples and comparing the results with Mie calculations, we showed that the setup is quite accurate for light scattering measurements. Also, the experimental investigations on different scattering samples with the designed and fabricated light scattering setup have led to several important conclusions.

1. The use of He-Ne lasers and Silicon photodetectors has made the setup highly compact and energy efficient.
2. The observations has proved the setup to be successful in design and fabrication for *in situ* measurement of the light scattering characteristics of small particles of sizes ranging from nanometer to micrometer.
3. The incorporation of four different types of sample holders (sample cuvette for hydrosols, rectangular and cylindrical sample holder for nanoparticles and nebulizer for aerosols/dust particles) ensured the measurement capability of the setup for different types of aerosols, hydrosols and nanoparticles.
4. Results with spheroidal titania particles revealed that porosity plays an important role in determining the scattering properties of scatterers. It was observed that nonporous titania particles may show quite different polarization properties as compared to the fluffy porous titania particles. From the measurements with diatom particles we found that the light scattering theories for such structures with nanopores need to be developed further. The light

scattering measurements on fresh water diatoms we have reported here may provide new clues as to the development of new models for light scattering from complex bodies. Experiments with nanoparticles revealed the characteristic details of ZnS semiconductor nanoparticles. Such experiments have a direct importance in the understanding of the interaction of light with particles in nanometer dimensions. Again as the ice particles crystals used in our work to represent real ice crystals was able to reproduce the halo peaks, it implies that such icelike crystals can be used for further investigations on atmospheric ice partices. Similarly measurements on graphite grains revealed the characteristic scattering properties of one type of astrophysical dust particles. It is expected that all the results of the measurements will be very much helpful in developing the light scattering properties of particles having very simple to very complex shapes.

The limitation of investigations with only He-Ne laser source having λ = 543.5 nm, 594.5 nm and 632.8 nm can be easily overcome in future by replacing the source with an organic dye laser having a tunable wavelength [22], a 1.06μm Nd:YAG laser [24], a 10.6μm CO_2 laser, a 0.6943μm ruby laser and a 0.4880μm or 0.514μm argon laser, which will enable investigations on a wide range of atmospheric constituents. More sensitive photodetectors can replace the existing photodetectors of the light scattering setup for operating over large distances and to investigate in the Rayleigh and Raman scattering domains where very low intensity scattered light fluxes are to be measured. Measurements by the light scattering setup of the scattered light at very small angles ($<3^0$) and near backscattering angles ($> 177^0$) instead of the scattering angle range 10^0 - 170^0 angles mentioned in this thesis, are to be taken to yield more accurate information about scattering and extinction coefficients. The provision for very small angle measurements will synthesize the design aspects of a Point visibility meter into the existing designed and developed

light scattering setup. Applying a more powerful computer with larger memory as a data recording and processing unit for the light scattering setup, the limitation of theoretical calculation of particle sizes with more complex parameters can be easily overcome. Electro-optic modulation at the source and incorporation of polarizers and analyzers at both the source end and the detector end can be considered, to enable measurements of all the elements of the scattering matrix so as to get better shape and size characteristic details of spherical and non-spherical particles. Measurements of backscattered light from remote locations by the light scattering setup, by incorporating a telescope and making arrangements for producing short laser pulses and with a little modification in the data acquisition system, will get due weightage keeping in view the emerging importance of lidar. Field trials in different environments can be done in the future to broaden the area of applicability of the light scattering setup. In view of the rapid developments in new kinds of lasers and sophisticated instrumentation and computing power, light scattering within earth's atmosphere or from space by means of satellites and space shuttles has a very bright future with the prospect of new scientific findings and practical applications.

The thesis also described two computer programs TUMiescat.c and TUTscat.c based on Mie theory and T-matrix approach for light scattering computations. Again a graphical user interface (GUI) incorporated analytical software TUSCAT based on the computer programs TUMiescat.c and TUTscat.c and written in java platform was developed during this PhD work.

1. This thesis reported the development of a fast, accurate and reliable computer program written in standard C to compute the scattering by monodisperse and polydisperse particles within the framework of Mie theory. Two methods for calculating n_{max} are incorporated in the program in order to study the scattering of electromagnetic radiation from absorbing or non-absorbing particles with wide range of size parameters. Both these methods are very accurate as can be seen from

Table 1. As the 'method 1' is the straightforward implementation of the Wiscombe's criterion for calculating n_{max}, it is relatively faster than 'method 2' which undergoes several steps of calculation as described in section 2.4.4. In 'method 2' the user can specify the accuracy level upto an order of 10^{-7}. In our work the tolerable accuracy level was set at 10^{-7} for the computations. 'Method 2' takes longer CPU (central processing unit) time than the 'method 1'. For example, it took 2.265 seconds longer to calculate the scattering properties of a particle (refractive index = 2.75+i*0.01) of radius 5 μm at accuracy level 10^{-7}. Therefore, for calculations where speed is more important 'method 1' is recommended and where accuracy is more important 'method 2' is recommended. Overall, in comparison to the computer programs developed by other workers, the present program not only gave the scattering properties as a function of size parameter and scattering angle, but was also able to perform Mie calculations for single particles as well as a variety of size distributions. The present program will be very much applicable in atmospheric sciences (remote atmospheric sensing), astrophysics (interplanetary and interstellar dust), ocean optics, biophysics and many other fields. The computer program was found to be very much efficient in calculating the light scattering properties of spherical particles with very large size parameters.

2. Next, the light scattering patterns for nonspherical particles (spheroids and cylinder) were computed using a computer program developed in C language to study variations in the patterns with changes in the size, size distribution, shape and refractive index of small particles in a volume element. As in the case of TUMiescat.c the particle size distributions considered were gamma, normal and lognormal. The program is based on T-matrix approach and stable for computation of the theoretical values of the non-zero elements of the scattering matrix, efficiency factors, single scattering albedo, radiation pressure and asymmetry parameter. After a comparison of the C program with other

reported benchmark results, it has been found that our program is much more accurate and reliable for electromagnetic scattering computations for nonspherical shapes. We acknowledge the fact that our C language T-matrix code in principle relies on Mishchenko's FORTRAN language 'tmd.new.f' code [5, 238] which is freely available in the NASA website (ftp://ftp.giss.nasa.gov/pub/crmim/tmd.new.f).

3. TUSCAT was originally developed to analyze the experimental results for different types of aerosols and hydrosols having different size distribution obtained by using the light scattering setup described in Chapter III. As per the requirement TUSCAT was developed to calculate the scattering properties of spherical, spheroidal and cylindrical particles. The GUI associated with the software enables the user to visualize the effect of changing input parameters on the resulting scattering patterns in near real time. In addition to its ease of use, it has high computational accuracy, efficiency, reliability and adaptability.

In future the improvements on TUMiescat.c and TUTscat.c may be done so as to explain experimentally observed light scattering patterns from other complex shapes. The software can be improved for the calculation of light scattering properties of other nonspherical shapes like chebyshev, star shaped etc. Some other efficient light scattering theories such as DDA, SVM and FDTD can also be incorporated for studying the scattering properties of nonspherical particles. Also the data acquisition and data processing part can be added to the software so as to make it self contained for the complete analysis of the experimental results.

References

1. Ahmed, G. A. *Design considerations of a laser based air quality monitoring system coupled to a microprocessor linked data recording and processing unit,* PhD thesis, Gauhati University, Guwahati, 2001.
2. Bohren, C. F. and Huffman, D. R. *Absorption and Scattering of Light by Small Particles,* John Wiley & Sons Inc, New York, 1983.
3. Van De Hulst, H. C. *Light Scattering by small particles,* John Wiley & Sons, New York, 1957.
4. Mishchenko, M. I., Hovenier, J. W., and Travis, L. D. (editors) *Light Scattering by Nonspherical Particles: Theory, Measurements, and Applications,* Academic, San Diego, California, 2000.
5. Mishchenko, M. I., Travis, L. D., Lacis, A. A. *Scattering absorption and emission of light by small particles,* Electronic Edition, Goddard Institute for Space Studies, New York, 2004 url: http://www.giss.nasa.gov/~crmim/books.html.
6. Lecler, S. *Light scattering by sub-micrometric particles,* PhD thesis, Louis Pasteur University - Strasbourg I, 2005.
7. Hirst, E., Kaye, P. H. Experimental and theoretical light scattering profiles from spherical and nonspherical particles, *J. Geophys. Res.*, **101** (D14), 19231-19235, 1996.
8. Arnold, E. W. *Light Scattering in Fibrous Sheets,* PhD thesis, The Institute of Paper Chemistry, Appleton, Wisconsin, 1960.
9. Schärtl, W. *Light Scattering from Polymer Solutions and Nanoparticle Dispersions,* Springer-Verlag Berlin, Heidelberg, 2007.
10. Hansen, J. E. and Travis, L. D. Light scattering in planetary atmospheres, *Space Sci. Rev.*, **16** (4), 527 - 610, 1974.

11. Ivezic, Z. and Menguc, M. P. An investigation of dependent/ independent scattering regimes using a discrete dipole approximation, *Int. J. of Heat and Mass Transfer*, **39** (4), 811-822, 1996.
12. Hess, M., et al. Optical Properties of Aerosols and Clouds: The Software Package OPAC, *Bull. Am. Meteorol. Soc.* **79** (5), 831-844, 1998.
13. Volten, H., et al. Scattering matrices of mineral aerosol particles at 441.6 nm and 632.8 nm, *J. Geophys. Res.*, **106** (D15), 17375-17401, 2001.
14. Kokhanovsky, A. A. *Aerosol Optics Light Absorption and Scattering by Particles in the Atmosphere,* Praxis Publishing Ltd, Chichester, UK, 2008.
15. Daugeron, D., et al. Laboratory polarization nephelometer for measurements of optical properties of aerosols, *Meas. Sci. Technol.* **18**, 632-638, 2007.
16. Mukai, S., et al. Removal of scattered light in the Earth atmosphere, *Earth Planets Space* **50**, 595-601, 1998.
17. Weiss-Wrana, K. Optical properties of interplanetary dust - Comparison with light scattering by larger meteoritic and terrestrial grains, *Astron. Astrophys.* **126**, 240-250, 1983.
18. Moroz, V. I., et al. Aerosol vertical profile on Mars from the measurements of thermal radiation on the limb, *Planet. Space Sci.* **42** (10), 831-845, 1994.
19. Mathis, J. S., et al. The size distribution of interstellar grains, *Astrophys. J.*, **217**, 425-433, 1977.
20. Gupta, R., et al. Interstellar extinction by spheroidal dust grains, *Astron. Astrophys.* **441**, 555-561, 2005.
21. Gupta, R., et al. Scattering Properties and Composition of Cometary Dust, *Astrophys. Space Sci.*, **301**, 21-31, 2006.
22. Westphal, A. J., et al. Analysis of "Midnight" Tracks in the Stardust Interstellar Dust Collector: Possible Discovery of a Contemporary Interstellar Dust Grain, 41st Lunar and Planetary Science Conference,

March 1-5, 2010, The Woodlands, Texas. LPI Contribution No. 1533, p.2050.

23. Brownlee, D. E. et. al. Stardust: Comet and interstellar dust sample return mission, *J. Geophys. Res.* **108** (E10), 8111(15PP), doi: 10.1029/ 2003JE002087, 2003.
24. Hadamcik, E., et al. Light scattering by fluffy particles with the PROGRA2 experiment: Mixtures of materials, *J. Quant. Spectrosc. Radiat. Transfer* **100**, 143–156, 2006.
25. Stephens, G. L. and Webster, P. J. Clouds and Climate: Sensitivity of Simple Systems, *J. Atmos. Sci.* **38**, 235–247, 1981.
26. Liou, K. N. and Takano, Y. Light scattering by nonspherical particles: remote sensing and climatic applications, *Atmos. Res.* **31**, 271–298,1994.
27. Yang, P. and Liou, K. N. Geometric-optics—integral-equation method for light scattering by nonspherical ice crystals, *Appl. Opt.*, **35** (33), 6568-6584, 1996.
28. Kokhanovsky, A. A. *Cloud Optics*, Springer, The Netherlands, 2006.
29. Kokhanovsky, A. A. (Editor) *Light Scattering Reviews Single and Multiple Light Scattering*, Praxis Publishing Ltd, Chichester, UK, 2006.
30. Kokhanovsky, A. A. (Editor) *Light Scattering Reviews 2 Remote Sensing and Inverse Problems*, Praxis Publishing Ltd, Chichester, UK, 2007.
31. Kokhanovsky, A. A. (Editor) *Light Scattering Reviews 3 Light Scattering and Reflection*, Praxis Publishing Ltd, Chichester, UK, 2008.
32. Kaye, P. H. Spatial light-scattering analysis as a means of characterizing and classifying non spherical particles, *Meas. Sci. Technol.* **9**, 141–149, 1998.
33. Kuik, F., et al. Experimental determination of scattering matrices of water droplets and quartz particles, *Appl. Opt.*, **30** (33), 4872-4881, 1991.
34. Volten, H. et al. WWW scattering matrix database for small mineral particles at 441.6 and 632:8 nm, *J. Quant. Spectrosc. Radiat. Transfer* **90**, 191–206, 2005.

35. Volten, H., et al. An update of the Amsterdam Light Scattering Database, *J. Quant. Spectrosc. Radiat. Transfer* **100**, 437–443, 2006.

36. Volten, H. et al. Scattering matrices of mineral aerosol particles at 441.6 and 632.8 nm, *J. Geophys. Res.* **106** (D15), 17375-17401, 2001.

37. Volten, H. et al. Laboratory Measurements and T-Matrix Calculations of the Scattering Matrix of Rutile Particles in Water, *Appl. Opt.* **38** (24), 5232-5240, 1999.

38. Muñoz, O. et al. Experimental and computational study of light scattering by irregular particles with extreme refractive indices: hematite and rutile, *Astron. Astrophys.* **446**, 525-535, 2006.

39. Worms, J. C. et al. Results of the PROGRA2 Experiment: An Experimental Study in Microgravity of Scattered Polarized Light by Dust Particles with Large Size Parameter, *Icarus* **142**, 281–297,1999.

40. Hadamcik, E., et al. Laboratory light scattering measurements on "natural" particles with the PROGRA2 experiment: an overview, *J. Quant. Spectrosc. Radiat. Transfer* **79–80**, 679–693, 2003.

41. Hadamcik, E. et al. Light scattering by low-density agglomerates of micron-sized grains with the PROGRA2 experiment, *J. Quant. Spectrosc. Radiat. Transfer* **106**, 74–89, 2007.

42. Renard, J. B., et al. Light scattering by dust particles in microgravity: polarization and brightness imaging with the new version of the PROGRA2 instrument, *Appl. Opt.* **41** (4), 609–618, 2002.

43. Hadamcik, E., et al. Polarization of Light Scattered by Fluffy Particles (PROGRA2 Experiment), *Icarus* **155**, 497–508, 2002.

44. Hadamcik, E., et al. Polarimetric study of levitating dust aggregates with the PROGRA2 experiment, *Planet. Space Sci.* **50**, 895–901, 2002.

45. McNeil, L. E. and French, R. H. Multiple scattering from rutile TiO_2 particles, *Acta Matter.* **48**, 4571-4576, 2006.

46. Thiele, E. S. and French, R. H. Light scattering properties of representative, morphological rutile TiO_2 particles studied using a Finite-Element method, *J. Am. Ceram. Soc.* **81** (3), 469-479, 1998.

47. Das, H. S., Sen, A. K., and Kaul, C. L. The polarimetric properties of cometary dust and a possible effect of dust aging by the Sun, *Astron. Astrophys.* **423**, 373-380, 2004.

48. Das, H. S. and Sen, A. K. Polarimetric studies of comet Levy 1990 XX, *Astron. Astrophys.* **459**, 271-273, 2006.

49. Gupta, R., Vaidya, et al. Scattering Properties and Composition of Cometary Dust, *Astrophys. Space Sci.* **301**, 21-31, 2006.

50. Sen, A. K. On the Variation of Cometary Polarisation, in *Proc. of an ESO Workshop 2003*, edited by Käufl, Hans Ulrich et al. (ESO Workshop, Garching, Germany, 18-21 November 2003) 546-549.

51. VanReken, T. M. *et. al.* Toward aerosol/cloud condensation nuclei (CCN) closure during CRYSTAL-FACE, *J. Geophys. Res.* **108** (D20), 4633, doi:10.1029/2003JD003582, 2003.

52. Ahmed, G. A., et al. Investigations by a designed and fabricated laser based Air Quality monitoring system, *J. of Instrument Society of India*, **26** (3), 734-738,1996.

53. Das, D., et al. Investigations on atmospheric humidity profile at varying oxygen levels with a laser based monitoring system, *Asian Journal of Physics* **10** (3), 323-330, 2001.

54. Liou, K. N. Light Scattering by Ice Clouds in the Visible and Infrared: A Theoretical Study, *J. Atmos. Sci.* **29**, 524-536, 1972.

55. Takano, Y. and Jayaweera, K. Scattering phase matrix for hexagonal ice crystals computed from ray optics, *Appl. Opt.* **24** (19), 3254-3263, 1985.

56. Takano, Y. and Liou, K. N. Solar radiative transfer in cirrus clouds. Part I: single scattering and optical properties of hexagonal ice crystals. *J. Atms. Sci.* **46** (1), 3-19,1989.

57. Kokhanovsky, A. A. and Zege, E. P. Scattering optics of snow, *Appl. Opt.* **43** (7), 1589–1602, 2004.

58. Barkey, B. and Liou, K. N. An analog light scattering experiment of hexagonal icelike particles. Part I: Experimental apparatus and test measurements, *J. Atms. Sci.* **56**, 605-612,1999.

59. Barkey, B. and Liou, K. N. Polar nephelometer for light-scattering measurements of ice crystals, *Opt. Lett.* **26** (4), 232-234, 2001.

60. Ulanowski, Z., et al. Scattering of light from atmospheric ice analogues, *J. Quant. Spectrosc. Radiat. Transfer* **79-80 C**, 1091-1102, 2003.

61. Ulanowski, Z., et al. Light scattering by complex ice-analogue crystals, *J. Quant. Spectrosc. Radiat. Transfer* **100** (1-3), 382-392, 2006.

62. Stramski, D., et al. The role of seawater constituents in light backscattering in the ocean, *Prog. Oceanogr.* **61**, 27–56, 2004.

63. Volten, H. et al. Laboratory measurements of angular distributions of light scattered by phytoplankton and silt, *Limnol. Oceanogr.* **43** (6), 1180-1197, 1998.

64. Jonasz, M. and Fournier G. R. *Light Scattering by Particles in Water Theoretical and Experimental Foundations*, Elsevier, Amsterdam, 2007.

65. Lotsberg, J. K., et al. Laboratory measurements of light scattering from marine particles, *Limnol. Oceanogr.: Methods* **5**, 34–40, 2007.

66. Kerker, M. and Mer, V. K. L. Particle Size Distribution in Sulfur Hydrosols by Polarimetric Analysis of Scattered Light, *J. Am. Ceram. Soc.* **72**, 3516–3525, 1950.

67. Quinby-Hunt, M. S., et al. Polarized-Light Scattering Studies of Marine Chlorella, *Limnol. Oceanogr.* **34** (8), 1587-1600, 1989.

68. Morel, A. Optics of marine particles and marine optics, *Particle Analysis in Oceanography* **G27**, 141–188, 1991.

69. Mobley, C. D. et al. Comparison of numerical models for computing underwater light climate, *Appl. Opt.* **32** (36), 7484 - 7504, 1993.

70. Di Toro, D. M. Optics of turbid estuarine waters: approximations and applications, *Water Res.* **12**, 1059–1068, 1978.

71. Morel, A. and Gentili, B. Diffuse reflectance of oceanic waters: its dependence on sun angle as influenced by the molecular scattering contribution, *Appl. Opt.* 30(30), 4427–4438, 1991.

72. Chami, M., et al. Variability of the relationship between the particulate backscattering coefficient and the volume scattering function measured at fixed angles, *J. Geophys. Res.* **111**, C05013, doi:10.1029/2005JC003230, 2006.

73. Bricaud, A. and Morel, A. Light attenuation and scattering by phytoplanktonic cells: a theoretical modeling, *Appl. Opt.* **25**, 571–580,1986.

74. Quirantes, A. and Bernard, S. Light scattering by marine algae: two-layer spherical and nonspherical models, *J. Quant. Spectrosc. Radiat. Transfer* **89**, 311–321, 2004.

75. Petzold, T. T. Volume scattering functions of selected ocean waters (Scripps Institution of Oceanography, Visibility Lab., University of California, San Diego. Ref. 72-78, October 1972).

76. Witkowski, K., et al. A Light-Scattering Matrix for Unicellular Marine Phytoplankton, *Limnol. Oceanogr.* **43** (5), 859-869, 1998.

77. Dekker, A. G., et al. Angular scattering functions of algae and silt: an analysis of backscattering to scattering fraction (In: Steven G. Ackleson. Editor. Proc. SPIE Vol. 2963, Ocean Optics XIII, Halifax, Canada22-25 October, 1996) 392-400.

78. Sokolov, A., et al. Parameterization of volume scattering function of coastal waters based on the statistical approach, *Opt. Express* **18** (5), 4615–4636, 2010.

79. Shu, J., et al. Elastic light scattering from nanoparticles by monochromatic vacuum-ultraviolet radiation, *J. Chem. Phys.*, **124**, 030707, 2006.

80. Pecora, R. Dynamic light scattering measurement of nanometer particles in liquids, *J. Nanopart. Res.* **2**, 123-131, 2000.

81. Chu, B. and Liu, T. Characterization of nanoparticles by scattering techniques, *J. Nanopart. Res.* **2**, 29-41, 2000.

82. Chen, K., et al. Nanoparticle sizing with a resolution beyond the diffraction limit using UV light scattering spectroscopy, *Opt. Commun.* **228**, 1–7, 2003.

83. Jillavenkatesa, A. and Kelly, J. F. Nanopowder characterization: challenges and future directions *J. Nanopart. Res.* **4**, 463-468, 2002.

84. Chen, Z., et al. Backscattering enhancement of light by nanoparticles positioned in localized optical intensity peaks, *Appl. Opt.* **45**, 632-638, 2006.

85. Orendorff, C. J., et al. Light scattering from gold nanorods: tracking material deformation, *Nanotechnology* **16**, 2601-2605, 2005.

86. Lechner, M. D. Influence of Mie scattering on nanoparticles with different particle sizes and shapes: photometry and analytical ultracentrifugation with absorption optics, *J. Serb. Chem. Soc.* **70** (3), 361-369, 2005.

87. van Dijk, M. A. et. al. Absorption and scattering microscopy of single metal nanoparticles, *Phys. Chem. Chem. Phys.*, 8, 3486–3495, 2006.

88. Hellmers, J., et al. Light scattering simulation for the characterization of sintered silver nanoparticles, J. Quant. Spectrosc. Radiat. Transfer 109 (8), 1363-1373, 2008.

89. Zhu, J. Theoretical study of the light scattering from gold nanotubes: Effects of wall thickness, *Mater. Sci. Eng., A*, **454–455**, 685–689, 2007.

90. Tanev, S., et al. Light scattering effects of gold nanoparticles in cells: FDTD modeling *Laser Phys. Lett.* **3** (12), 594–598, 2006.

91. Luk'yanchuk, B. S. et. al. Peculiarities of light scattering by nanoparticles and nanowires near plasmon resonance frequencies in weakly dissipating materials, *J. Opt. A: Pure Appl. Opt.* **9**, S294–S300, 2007.

92. Stam, D. M., et al. Integrating polarized light over a planetary disk applied to starlight reflected by extrasolar planets, *Astron. Astrophys.* **452**, 669-6, 2006.

93. Stammes, P., et al. The polarized internal radiation field of a planetary atmosphere, *Astron. Astrophys.* **225**, 239-259, 1989.

94. Nicolae, D. N. et al. Analytical averaging method in scattering of light by ensembles of nonspherical aerosols, *J. Optoelectron. Adv. Mater.* **6** (3), 831-840, 2004.

95. Bacon, N. J., Swanso, B. D. Laboratory Measurements of Light Scattering by Single Levitated Ice Crystals, *J. Atmos. Sci.* **57**, 2094–2104, 2000.

96. Shcherbakov, V., et al. Light Scattering by Single Natural Ice Crystals, *J. Atmos. Sci.* **63**, 1513–1525, 2006.

97. Yang, P., et al. Single-scattering properties of droxtals, *J. Quant. Spectrosc. Radiat. Transfer* **79–80**, 1159–1169, 2003.

98. Nousiainen, T. *Light scattering by nonspherical atmospheric particles*, Academic dissertation in meteorology, Finnish Meteorological Institute, Helsinki, 2002.

99. Hovenier, J. W., et al. Laboratory studies of scattering matrices for randomly oriented particles: potentials, problems, and perspectives, *J. Quant. Spectrosc. Radiat. Transfer* **79–80**, 741–755, 2003.

100. Mie, G. Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen, *Annalen der Physik* **25** (3), 377-445, 1908.

101. Mie, G. *Contributions on the optics of turbid media, particularly colloidal metal solutions*, Translation: Sandia Laboratories, Albuquerque, New Mexico, SAND78-6018. National Translation Center, Chicago, ILL, Translation 79-21946, 1978.

102. Mie, G. *Contributions to the optics of turbid media, particularly of colloidal metal solutions*, Translation: Royal Aircraft Establishment, Library Translation 1873, RAE-Lit-Trans-1873, 1976.

103. Lilienfeld, P. Gustav Mie: the person, *Appl. Opt.* 30 (33), 4696-4698, 1991.

104. Corry, L. From Mie's Electromagnetic Theory of Matter to Hilbert's Unified Foundation of Physics, *Studies in History and Philosophy of Modern Physics* **30** (2), 159-183, 1999.

105. Huffman, P. J. and Thursby, W. R. Light scattering by ice crystals, *J. Atmos. Sci.* **26**, 1073-1077, 1969.

106. Huffman, P. Polarization of light scattered by ice crystals *J. Atmos. Sci.*, **27**, 1027-1028, 1970.

107. Pollack, J. B. and Cuzzi, J. N. Scattering by nonspherical particles of size comparable to the wavelength: a new semi-emperical theory and its application to tropospheric aerosols, *J. Atmos. Sci.*, **37**, 868-881, 1980.

108. Cai, Q. and Liou, K. N. Polarized light scattering by hexagonal ice crystals: theory, *Appl. Opt.*, **21**, 3569-3580, 1982.

109. Gouesbet, G. Generalized Lorenz–Mie theories, the third decade: A perspective, *J. Quant. Spectrosc. Radiat. Transfer* **110**, 1223–1238, 2009.

110. Mishchenko, M. I. and Travis, L. D. Gaustav Mie and the evolving discipline of electromagnetic scattering by particles, *Bull. Am. Meteorol. Soc.* **89** (12), 1853-1861, 2008.

111. Laven, P. Simulation of rainbows, coronas and glories using Mie theory and the Debye series, *J. Quant. Spectrosc. Radiat. Transfer* **89**, 257-269, 2004.

112. Min, M., et al. Shape effects in scattering and absorption by randomly oriented particles small compared to the wavelength, *Astron. Astrophys.* **404**, 35–46, 2003.

113. Fratalocchi, A., et al. Three-dimensional ab initio investigation of light-matter interaction in Mie lasers, Phys. *Rev. A: At. Mol. Opt. Phys.* **78**, 013806, 2008.

114. Wriedt, T. Mie theory 1908 on the mobile phone, J. Quant. Spectrosc. Radiat. Transfer 109, 1543–1548, 2008.

115. Ren, K. F., et al. Scattering of a Gaussian beam by an infinite cylinder in the framework of generalized Lorenz-Mie theory: formulation and numerical results, *J. Opt. Soc. Am. A* **14** (11), 3014-3025, 1997.

116. Laven, P. Simulation of rainbows, coronas, and glories by use of Mie theory, *Appl. Opt.* **42** (3), 436-444, 2003.

117. Graaff, R. *et al.* Reduced light-scattering properties for mixtures of spherical particles: a simple approximation derived from Mie calculations, *Appl. Opt.* **31** (10), 1370 - 1376, 1992.

118. Shah, G. A. Numerical methods for Mie theory of scattering by a sphere, *Kodalkanal Obs. Bull. Ser. A* **2**, 42-63, 1977.

119. Gouesbet, G. T-matrix formulation and generalized Lorentz-Mie theories in spherical coordinates, *Opt. Commun.* **283**, 517-521, 2010.

120. Ross, W. D. Computation of Bessel Functions in Light Scattering Studies, *Appl. Opt.* **11** (9), 1919- 1923, 1972.

121. Peňa, O. and Pal, U. Scattering of electromagnetic radiation by a multilayered sphere, *Comput. Phys. Commun.* **180**, 2348-2354, 2009.

122. Wolf S. and Voshchinnikov, N. V. Mie scattering by ensembles of particles with very large size parameters, *Comput. Phys. Commun.* **162**, 113-123, 2004.

123. Wiscombe, W. J. Improved Mie scattering algorithms, *Appl. Opt.* **19** (9), 1505-1509, 1980.

124. Deirmendjian, D. *Electromagnetic Scattering on Spherical Polydispersions*, Elsevier, New York, 1969.

125. Voshchinnikov, N.V., et al. Extinction and Polarization of Radiation by Absorbing Spheroids: Shape/Size Effects and Some Benchmarks, *J. Quant. Spectrosc. Radiat. Transfer* **65**, 877-893, 2000.

126. Kocifaj, M. *Lecture Notes: Light scattering by Small Particles Atmospheric Optics and Astrophysical Application Part I-Theory*, Institute of Experimental Physics, University of Vienna and Astronomical Institute, Slovak Academy of Sciences, 2007.

127. Asano, S. Light scattering properties of spheroidal particles, *Appl. Opt.* **18** (5), 712 - 723, 1979.

128. Asano, S. and Yamamoto, G. Light Scattering by a Spheroidal Particle. *Appl. Opt.* **14** (1), 29 - 49, 1975.

129. Voshchinnikov, N. V., and Farafonov, V. G. Optical properties of spheroidal particles, *Astrophys. Space Sci.* **204**, 19–86, 1993.

130. Aden, A. L., Kerker, M., Scattering of electromagnetic waves from two concentric spheres, *J. Appl. Phys.* **22**, 1242–12463, 1951.

131. Bhandari, R. Scattering coefficients for a multilayered sphere: analytic expressions and algorithms, *Appl. Opt.* **24** (13), 1985.

132. Bohren, C. F. Scattering of Electromagnetic Waves by an Optically Active Cylinder, *J. Colloid Interface Sci.* 66 (1), 105-109, 1978.

133. Farafonov, V. G., et al. Light scattering by a core-mantle spheroidal particle, *Appl. Opt.* **35(27)**, 5412-5426 (1996).

134. Voshchinnikov, N. V. Electromagnetic scattering by homogeneous and coated spheroids: Calculations using the separation of variables method, *J. Quant. Spectrosc. Radiat. Transfer.* **55(5)**, 627-636. (1996)

135. Rother T. Generalization of the separation of varialbles method for non-spherical scattering on dielectric objects, *J. Quant. Spectrosc. Radiat. Transfer.* **60** (3), 335-353, 1998.

136. Kunz K. S. and Luebbers R. J. *The Finite Difference Time Domain Method for Electromagnetics,* CRC Press, Florida, 1993.

137. Itoh T. (editor) *Numerical Techniques for Microwave and Millimeter-Wave Passive Structures,* John Wiley & Sons, New York, 1989.

138. Yee, S. K. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antennas Propag.* **14**, 302–307, 1966.

139. Yee, S. K., Chen, J. S., and Chang, A. H. Conformal finite difference time domain (FDTD) with overlapping grids, *IEEE Trans. Antennas Propag.* **40**, 1068–1075, 1992.

140. Kahnert, F. M. Numerical methods in electromagnetic scattering theory. *J. Quant. Spectrosc. Radiat. Transfer.* **79-80,** 775- 824, 2003.

141. Drezek, R., et al. A Pulsed Finite-Difference Time-Domain (FDTD) Method for Calculating Light Scattering from Biological Cells Over Broad Wavelength Ranges, *Opt. Express* **6** (7), 147-157, 2000.

142. Wriedt, T. and Comberg, U. Comparison of computational and scattering methods, *J. Quant. Spectrosc. Radiat. Transfer.* **60** (3), 411-423, 1998.

143. Wriedt, T. A review of elastic light scattering theories, *Part. Part. Syst. Charact.* **15**, 67-74, 1998.

144. Dunn, A., et al. Finite difference time domain simulation of light scattering from single cells, *J. Biomed. Opt.* **2** (3), 262-266, 1997.

145. Yang, P. and Liou, K. N. Light scattering by hexagonal ice crystals: comparison of finite-difference time domain and geometric optics models *J. Opt. Soc. Am. A,* **12** (1), 162-176, 1995.

146. Yang, P. and Liou, K. N. Finite-difference time domain method for light scattering by small ice crystals in three-dimensional space, *J. Opt. Soc. Am. A* **13** (10), 2072-2085, 1996.

147. Yang, P., et al. Efficient Finite-Difference Time-Domain Scheme for Light Scattering by Dielectric Particles: Application to Aerosols, *Appl. Opt.*, **39** (21), 3727-3737, 2000.

148. Sun, W., et al. Examination of Surface Roughness on Light Scattering by Long Ice Columns by Use of a Two-Dimensional Finite-Difference Time-Domain Algorithm. *Appl. Opt.* **43** (9), 1957-1964, 2004.

149. Arridge, S. R., et al. The finite element model for the propagation of light in scattering media: A direct method for domains with nonscattering regions, *Med. Phys.* **27** (1), 252-264, 2000.

150. Arridge, S. R., et al. A finite element approach for modeling photon transport in tissue. *Med. Phys.*, **20** (2 Pt. 1), 299-309, 1993.

151. Volakis, J. L., et al. Review of the finite-element method for three-dimensional electromagnetic scattering, *J. Opt. Soc. Am. A* **11** (4), 1422-1433, 1994.

152. Wiscombe, W. J. and Mungai, A. *Single scattering from non-spherical chebyshev particles: a compendium of calculations*, NASA Ref. Publ. 1157 NASA/GSFC Greenbelt, MD, 1986.

153. Nieminen, T. A., et al. Calculation of the T-matrix: general considerations and application of the point-matching method, *J. Quant. Spectrosc. Radiat. Transfer.* **79-80**, 1019-1029, 2003.

154. DeVoe, H. Optical properties of molecular aggregates. I. Classical model of electronic absorption and refraction, *J. Chem. Phys.* **41**, 393–400, 1964.

155. Purcell, E. M., Pennypacker, C. R. Scattering and absorption of light by nonspherical dielectric grains, Astrophys. J. 186, 705–14 (1973).

156. Kimuran, H., and Mann, I. Radiation pressure cross section for fluffy aggregates, *J. Quant. Spectrosc. Radiat. Transfer.* **60** (3), 425–438, 1998.

157. Liu, C.-L. and Illingworth, A. J. Error analysis of backscatter from discrete dipole approximation for different ice particle shapes, *Atmos. Res.* **44**, 231-241, 1997.

158. Waterman, P. C. and Truell, R. Multiple scattering of waves, *J. Math. Phys.*, **2** (4), 512–537, 1961.

159. Waterman, P.C. New formulation of acoustic scattering, *J. Acoust. Soc. Am.* **45** (6), 1417-1429, 1968.

160. Waterman, P.C. Symmetry, unitarity, and geometry in electromagnetic scattering *Phys. Rev. D: Part. Fields* **3**, 825–839, 1971.

161. Wriedt, T. Using the T-Matrix Method for Light Scattering Computations by Non-axisymmetric Particles: Superellipsoids and Realistically Shaped Particles, *Part. Part. Syst. Charact.* **19**, 256–268, 2002.

162. Schmidt, K., et al. The equivalence of applying the Extended Boundary Condition and the continuity conditions for solving electromagnetic scattering problems, *Opt. Comm.* **150**, 1–4, 1998.

163. Doicu, A., et al. Projection schemes in the null field method, *J. Quant. Spectrosc. Rad. Transfer.* **63**, 175-189, 1999.

164. Doicu, A. and Wriedt, T. Calculation of the T-matrix in the null-field method with discrete sources, *J. Opt. Soc. Am. A* **16**, 2539-2544, 1999.

165. Quirantes, A., et al. Multiple Light Scattering by Spherical Particle Systems and Its Dependence on Concentration: A T-Matrix Study, *J. Colloid Interface Sci.* **240**, 78-82, 2001.

166. Mackowski, D. W. Discrete dipole moment method for calculation of the T-matrix for nonspherical particles, *J. Opt. Soc. Am. A* **19**, 881-893, 2002.

167. Mackowski, D. W. and Mishchenko, M. I. Calculation of the T matrix and the scattering matrix for ensembles of spheres, *J. Opt. Soc. Am. A* **13**, 2266-2278, 1996.

168. Mishchenko, M. I., et al. T-Matrix Computations of Light Scattering By Nonspherical Particles: A Review, *J. Quant. Spectrosc. Radiat. Transfer.* **55** (5), 535-515, 1996.

169. Mishchenko, M. I et al. Scattering of light by polydisperse, randomly oriented, finite circular cylinders, *Appl. Opt.* **35**, 4927-4940, 1996.

170. Wriedt, T. and Doicu, A. Formulations of the extended boundary condition method for three dimensional scattering using the method of discrete sources, *J. Mod. Opt.* **45**, 199-213, 1998.

171. Khlebtsov, N. G. Orientational averaging of light scattering observables in the T-matrix approach, *Appl. Opt.* **31**, 5359-5365, 1992.

172. Laitinen, H., Lumme, K. T-matrix method for general star-shaped particles: first results, *J. Quant. Spectrosc. Radiat. Transfer.* **60** (3), 325-334, 1998.

173. Quirantes, A. and Delgado, A. Experimental determination of spheroidal particles via the T-matrix method, *J. Quant. Spectrosc. Radiat. Transfer.* **60** (3), 463-474, 1998.

174. Quirantes, A. A T-matrix method and computer code for randomly oriented, axially symmetric coated scatterers, *J. Quant. Spectrosc. Radiat. Transfer.*, **92**, 33-381, 2005.

175. Riefler, N., Wriedt, T. *T-matrix simulation of plasmon resonances of particles on or near a surface*, Progress in Electromagnetic Research Symposium, Cambridge, USA, March 26-29, 2006.

176. Petrov D. et al. The T-matrix technique for calculation of scattering properties of ensembles of randomly oriented particles with different size, *J. Quant. Spectrosc. Radiat. Transfer.* **102**, 85-110, 2006.

177. Sharma, S. and Somerford, D. J. Light scattering by optically soft particles Theory and applications (Praxis Publishing, Chichester, UK, 2006).

178. S. K. Sharma and D. J. Somerford. Modified Rayleigh-Gans-Debye approximations applied to sizing transparent homogeneous long fibres of intermediate size. *J. Phys. D: Appl. Phys.* **21**, 403-406, 1988.

179. Chylek, P. and Klett, J. D. Extinction cross sections of nonspherical particles in the anomalous diffraction approximation. *J. Opt. Soc. Am. A* **8** (2), 274–281, 1991.

180. Farone, W. A. and Robinson, M. J. The range of validity of the anomalous diffraction approximation to electromagnetic scattering by a sphere. *Appl. Optics* 7 (4), 643 - 646, 1968.

181. Liou, K. N. and Hansen, J. E. Intensity and polarization for single scattering by polydisperse spheres: a comparison of ray optics and Mie theory, *J. Atmos. Sci.* **28**, 995–1004, 1971.

182. Macke, A., et al. Scattering of light by large nonspherical particles: ray-tracing approximation versus T-matrix method, *Optics Lett.* **20** (19), 1934-1936, 1995.

183. Ravey, J. C. and Mazeron, P. Light scattering in the physical optics approximation; application to large spheroids, *J. Optics (Paris)* **13** (5), 273-282, 1982.

184. Zhang, Z. et al. Geometrical-optics solution to light scattering by droxtal ice crystals, *Appl. Optics* **43** (12), 2490–2499, 2004.

185. Logan, N. A. Survey of some early studies of the scattering of plane waves by a sphere, *Proc. IEEE* **53**, 773–785, 1965.

186. Demir, V., et al. A graphical user interface (GUI) for plane-wave scattering from a conducting dielectric, or chiral sphere, *IEEE Antennas and Propagation Magazine* **46** (5), 94–99, 2004.

187. Sharkawy, M. A., et al. A graphical user interface (GUI) for electromagnetic scattering from two- and three-dimensional canonical and non-canonical objects, *IEEE Antennas and Propagation Magazine* **48** (6), 135–141, 2006.

188. Cheng, G., et al. An interactive visualization environment for an electromagnetic scattering simulation on a high performance computing system.

189. Ko, K. D. and Toussaint Jr., K. C. A simple GUI for modelling the optical properties of single metal nanoparticles, *J. Quant. Spectrosc. Radiat. Transfer.* **110**, 1037–1043, 2009.

190. Pritchard, B. S. and Elliott, W. G. Two instruments for atmospheric optics measurements, *J. Optical Society of America* **50** (3), 191-202, 1960.

191. Hunt, A. J. and Huffman, D. R. A new polarization-modulated light scattering instrument. *Rev. Sci. Instrum.* **44** (12), 1763-1773, 1973.

192. Perry, R. J., et al. Experimental determinations of Mueller scattering matrices for nonspherical particles, *Applied Optics* **17** (17), 2700-2710, 1978.

193. Bickel, W. S., et al. Polarized light scattering from metal surfaces, *J. Appl. Phys.* **61** (12), 5392–5398, 1987.

194. Kratohvil, J. P. and Wallace, T. P. Calibration of light-scattering photometers VII. Calibration by means of colloidal dispersions of Mie scatterers, *J. Phys. D Appl. Phys.* **3**, 221-227, 1970.

195. Donald R. Huffman. Optical properties of particulates. *Astrophysics and Space Science*, Vol. 34, No. 1, 175-184, April 1975.

196. Bickel, W. S. et al. Application of polarization effects in light scattering: a new biophysical tool, *Proc. Nat. Acad. Sci. USA* **73** (2), 486-490, 1976.

197. R. C. Thompson et al. Measurement of polarized light interaction via the Mueller matrix. *Applied Optics*, Vol. 19, 1323-1332, 1980.

198. Anderson, R. Measurement of Mueller matrices, *Applied Optics* **31**, 11-13, 1992.

199. Valentine, M. T., et al. Microscope-based static light-scattering instrument, *Optics Letters* **26** (12), 890-892, 2001.

200. Gustafson, B. A. S. Microwave analog to light scattering measurements: A modern implementation of a proven method to achieve precise control, *J. Quant. Spectrosc. Radiat. Transfer* **55**, 663–672, 1996.

201. Anderson, T. L. et. al. Performance characteristics of a high-sensitivity, three-wavelength, total scatter/backscatter nephelometer, *J. Atms. Sci.* **13**, 967–986, 1996.

202. Doherty, S. J., et al. Measurement of the lidar ratio for atmospheric aerosols with a 180° backscatter nephelometer, *Appl. Optics* **38** (9), 1823–1832, 1999.

203. Quirantes, A., et al. Correction factors for a total scatter/backscatter nephelometer, *J. Quant. Spectrosc. Radiat. Transfer* **109**, 1496–1503, 2008.

204. Hansen, M. Z. and Evans, W. H. Polar nephelometer for atmospheric particulate studies, *Appl. Optics* **19** (19), 3389-3395, 1980.

205. Castagner, J. L. and Bigio, I. J. Particle sizing with a fast polar nephelometer, *Appl. Optics* **46** (4), 527-532, 2007.

206. Lee, P. H., et al. Laser transformer-a description, *The Science of the Total Environment* **23**, 321 – 335, 1982.

207. Stewart, H. S. and Curcio, J. A. The influence of field of view on measurements of atmospheric transmission, *J. Opt. Soc. Am.* **42**, 801-805, 1952.

208. Curcio, J. A., et al. An experimental study of atmospheric transmission, *J. Opt. Soc. Am.* **43**, 97-102, 1953.

209. Eldrige, R. G. and Jhonson, J. C. Diffuse transmission through real atmospheres, *J. Opt. Soc. Am.* **48**, 463-468, 1953.

210. Duntley, S. Q. The visibility of distant objects, *J. Opt. Soc. Am.* **38** (3), 237-249, 1948.

211. Roberts, R. E., et al. Infrared continuum absorption by atmospheric water vapour in the 8-12 micrometer window, *Appl. Opt.* **15**, 2085-2090, 1976.

212. Gibbons, M. G. Transmission and scattering properties of a Nevada Desert atmosphere under cloudy conditions, *J. Opt. Soc. Am.* **51**, 633-640, 1961.

213. Clay, M. R. and Lenham, A. P. Transmission of electromagnetic radiation in fogs in the 0.53 to 10.1 micron wavelength range, *Appl. Opt.* **20**, 3831-3832, 1981.

214. Nefedov, A. P., et al. Application of a forward-angle-scattering transmissiometer for simultaneous measurements of particle size and number density in an optically dense medium, *Appl. Opt.* **37**, 1682-1689, 1998.

215. Winstanley, J. V. and Adams, M. J. Point visibility meter: a forward scatter instrument for the measurement of aerosol extinction coefficient, *Appl. Opt.* **14**, 2151-2157, 1975.

216. Killinger, D. K. and Menyuk, N. Laser remote sensing of the atmosphere, *Science* **235**, 37-45, 1987.

217. Kasparian J. et al. White-light filaments for atmospheric analysis, *Science* **301**, 61–64, 2003.

218. Lefsky, M. A., Cohen, W. B., Parker, G. G. and Harding, D. J. Lidar remote sensing for ecosystem studies, *BioScience* **52** (1), 19-30, 2002.

219. Wehr, A. and Lohr, U. Airborne laser scanning—an introduction and overview, *ISPRS J. Photogrammetry & Remote Sensing* **54**, 68–82, 1999.

220. Remsberg, E. E. and Gordley, L. L. Analysis of differential absorption lidar from the space shuttle, *Appl. Opt.* **17**, 624-630, 1978.

221. Browell, E. V., et al. Water vapour differential-absorption lidar development and evaluation, *Appl. Opt.* **18**, 3474-3483, 1978.

222. Fredriksson, K., et al. Lidar system applied in atmospheric pollution monitoring, *Appl. Opt.* **18**, 2998-3003, 1979.

223. Fredriksson, K., et al. Mobile lidar system for environmental probing, *Appl. Opt.* **20**, 4181-4189, 1981.

224. Menyuk, N., et al. Remote sensing of NO using a differential absorption lidar, *Appl.Opt.* **19**, 3282-3286, 1980.

225. Behrendt, A., et al. Combined Raman lidar for the measurement of atmospheric temperature, water vapor, particle extinction coefficient, and particle backscatter coefficient, *Appl.Opt.* **41** (36), 7657-7666, 2002.

226. Devera, P. C. S., et al. Lidar for environmental monitoring, *J. Instrument Soc. India* **26** (3), 62-67, 1996.

227. Mller, D., et al. Microphysical particle parameters from extinction and backscatter lidar data by inversion with regularization: theory, *Appl. Opt.* **38**, 2346-2357, 1999.

228. Ben-David, A. Backscattering measurements of atmospheric aerosols at CO_2 laser wavelengths: implications of aerosol spectral structure on differential-absorption lidar retrievals of molecular species, *Appl. Opt.* **38**, 2616-2624, 1999.

229. Mishchenko, M. I. and Travis, L. D. *Maxwell's equations, electromagnetic waves, and Stokes parameters.* In: G. Videen, Ya. Yatskiv and M. Mishchenko (editors), Photometry in remote sensing. Kluwer Academic Publishers, Netherlands, 2004.

230. Born, M. and Wolf, E. *Principles of Optics*, Cambridge University Press, Cambridge, 1999.

231. Jackson, J. D. *Classical Electrodynamics*, Wiley Eastern Ltd., New Delhi, 1975.

232. O'Bree, T. A. Investigations of light scattering by Australian natural waters for remote sensing applications. PhD thesis, School of Applied Sciences, Applied Physics, RMIT University, Victoria, Australia, 2007.

233. Bickel, W. S. and Berry, W. M. Stokes vectors, Mueller matrices, and polarized scattered light, *Am. J. Phys.* **53** (5), 468–478, 1985.

234. Smith, A. J. A. The scattering of light by non-spherical particles. First year postdoctoral report, Atmospheric, Oceanic and Planetary Physics, department of Physics, University of Oxford, 2008.

235. Ghatak, A. K., Goyal, I. C. and Chua, S. J. *Mathematical Physics*, Macmillan India Ltd., New Delhi, 1995.

236. Yang, P., Mlynczak, et al. Spectral signature of ice clouds in the far-infrared region: single-scattering calculations and radiative sensitivity study, *J. Geophysical Research* **108** (D18), 4569 (15 pages), 2003.

237. Deirmendjian, D. Scattering and polarization properties of water clouds and hazes in visible and infrared, *Appl. Opt.* **2**, 187-196, 1964.

238. Mishchenko, M. I. and Travis, L. D. Capabilities and limitations of a current fortran implementation of the T-matrix method for randomly oriented, rotationally symmetric scatterers, *J. Quant. Spectrosc. Radiat. Transfer* **60** (3), 309—324, 1998.

239. Mishchenko, M. I. Vector radiative transfer equation for arbitrarily shaped and arbitrarily oriented particles: A microphysical derivation from statistical electromagnetics, *Appl. Opt.* **41**, 7114-7134, 2002.

240. Hovenier, J. W. and van der Mee, C. V. M. Fundamental relationships relevant to the transfer of polarized light in a scattering atmosphere, *Astronomy and Astrophysics* **128** (1), 1-16, 1983.

241. Mishchenko, M. I. Light scattering by size-shape distributions of randomly oriented axially symmetric particles of a size comparable to the wavelength, *Appl. Opt.* **32** (24), 4652– 4666, 1993.

242. Quirantes, A. and Delgado, A. V. Scattering cross sections of randomly oriented coated spheroids, *J. Quant. Spec. Rad. Trans.* **70**, 261–272, 2001.

243. Kostelec, P. J. and Rockmore, D. N. FFTs on the rotation group, *J. Fourier Analysis and Applications* **14** (2), 145-179, 2008.

244. Mishchenko, M. I. and Travis, L. D. Light scattering by polydispersions of randomly oriented spheroids with sizes comparable to wavelengths of observation, *Appl. Opt.* **33** (30), 7206-7225, 1994.

245. Mishchenko, M. I. Extinction of light by randomly-oriented non-spherical grains, *Astrophysics and Space Science* **164**, 1-13, 1990.

246. Xu, M. Light extinction and absorption by arbitrarily oriented finite circular cylinders by use of geometrical path statistics of rays, *Appl. Opt.* **42** (33), 6710-6723, 2003.

247. Lee, Y. K., et al. Use of circular cylinders as surrogates for hexagonal pristine ice crystals in scattering calculations at infrared wavelengths, *Appl. Opt.* **42** (15), 2653-2664, 2003.

248. Chen, G., et al. Scattering phase functions of horizontally oriented hexagonal ice crystals, *J. Quant. Spec. Rad. Trans.* **100**, 91-102, 2006.

249. Gogoi, A., et al. Detector array incorporated optical scattering instrument for nephelometric measurements on small particles *Meas. Sci. Technol.* **20**, 095901 (10pp), 2009.

250. Gogoi, A., et al. Laboratory measurements of light scattering by tropical fresh water diatoms, *J. Quant. Spec. Rad. Trans.***110**, 1566-1578, 2009.

251. Gogoi, A., et al. Construction of a multidetector array incorporated laser based scattering system for ultrafine TiO_2 characterization, *J. Optics* **38** (2), 67-74, 2009.

252. Gogoi, A., et al. Nanoparticle size characterization by laser light scattering, *Indian J. Phys.* **83** (4), 473-477, 2009.

253. Datasheet of BPW34 silicon PIN photodiode (SIEMENS).

254. General purpose linear devices databook, National Semiconductor, 1987.

255. Catalogue for AX5210 data acquisition card (Vinytics).

256. Mohanta, D., et al. Irradiation induced grain growth and surface emission enhancement of chemically tailored ZnS : Mn/PVOH nanoparticles by Cl^{+9} ion impact, *Bull. Mater. Sci.* **26** (3), 289–294, 2003.

257. Mohanta, D., et al. Optical absorption study of 100-MeV chlorine ion-irradiated hydroxyl-free ZnO semiconductor quantum dots, *J. Appl. Phys.* **92** (12), 7149–7152, 2002.

258. Mohanta, D. and Choudhury, A. Laser-induced photocurrent measurement in quasi-arrayed ZnS quantum dots, *Physica E* **27**, 176–182, 2005.

259. Mohanta, D., et al. Spectroscopic Investigations of Carrier Confinement and Surface Phonon Detection in Polymer Embedded CdS Quantum Dot Systems, *Chinese Journal of Physics* **42** (6), 2004.

260. Gogoi, A. and Ahmed, G. A. A T-matrix approach for the morphological characterization of spherical nanoparticles using laser, *Indian J. Phys.* **82** (5), 147-150, 2008.

261. Asano, S. and Sato, M. Light scattering by randomly oriented spheroidal particles, *Appl. Opt.* **19** (6), 962-974, 1980.

262. Schnablegger, H. and Glatter, O. Simultaneous determination of size distribution and refractive index of colloidal particles from static light scattering experiments, *J. Colloid and Interface Science* **158**, 228-242, 1993.

263. Wiscombe, W. J. *Mie scattering calculations: advances in technique and fast, vector-speed computer codes*, NCAR/TN-140+STR, NCAR Tech. Note-National Center for Atmospheric Research, Boulder, Colorado, 1979.

264. Bass, M., Stryland, E. W. V., Williams, D. R. and Wolfe, W. L. (Editors) *Handbook of Optics, Volume II - Devices , Measurements , and Properties*, McGraw-Hill , Inc . New York, 1995.

265. Nikolova, S., et al. Analysis of the dispersion of optical plastic materials, *Optical Materials* **29**, 1481-1490, 2007.

266. Bentley, K., et al. Nature's Batik: a computer evolution model of diatom valve morphogenesis, *J. Nanoscience and Technology* **5**, 25-34, 2005.

267. Wee, K. M., et al. Engineering and medical applications of diatoms, *J. Nanoscience and Technology* **5**, 88-91, 2005.

268. De Stefano, M. and De Stefano, L. Nanostructures in diatom frustules: functional morphology of Valvocopulae in Cocconeidacean Monoraphid Taxa, *J. Nanoscience and Technology* **5**, 15-24, 2005.

269. De Stefano, L., et al. Marine diatoms as optical chemical sensors, *Appl. Phys. Lett.* **87**, 233902, 2005.

270. Butcher, K. S. A., et al. Photoluminescence and cathodoluminescence studies of diatoms- nature's own nano-porous silica structures, *Proc. of the 27th Annual A&NZIP Condensed Matter and Materials Meeting*, Edited by Cashion, J., Finlayson, T., Paganin, D., Smith, A. and Troup, G. 51-53, 2003.

271. Guillard, R. R. L. and Lorenzen, C. J. Yellow-green algae with chlorophyllide c, *J. Phycology* **8**, 10-14, 1972.

272. Mohanta, D., et al. Influence of ion bombardment on the photoluminescence response of embedded CdS nanoparticles, Central European Journal of Physics,CEJP 4(2), 187-195, 2006

273. Chowdhury, S., et al. Third order nonlinear optical response of PbS quantum dots, *Semiconductor Physics, Quantum Electronics & Optoelectronics*, **9** (2), 45-48, 2006.

274. Bayan, S., Mohanta, D. Directed growth characteristics and optoelectronic properties of Eu-doped ZnO nanorods and urchins, *J. Appl. Phys.* **108**, 023512, 2010.

275. Dutta, N., et al. Studies of optical properties and SHI irradiation on PbS sensitized nanoporous TiO2 network, *J. Opt.* **38** (3) : 169-176, 2009.

276. Chowdhury, S., et al. Luminescence study of bare and coated CdS quantum dots: Effect of SHI irradiation and ageing, *Nucl. Instrum. Methods Phys. Res., Sect. B* **240**, 690-696, 2005.

277. Chowdhury, S., et al. Effect of 160 MeV Ni^{12+} ion irradiation on PbS quantum dots, *J. Luminescence* **114**, 95-100, 2005.

278. Das, H. S., et al. Aggregate dust model to study the polarization properties of comet C/1996 B2 Hyakutake, Research in Astron. Astrophys., 10, 4, 355-362, 2010.

279. Das, H. S., et al. Aggregate dust model to describe polarization properties of Comet Hale-Bopp, Mon. Not. R. Astron. Soc., 390, 1195-1199, 2008.

280. *Mikrenska, M. and Koulev, P. Simulation of light scattering by large* particles with randomly distributed spherical or cubic inclusions, J. Quant. Spec. Rad. Trans. 110, 1411-1417, 2009.

281. Mikrenska, M., et al. Direct simulation Monte Carlo ray tracing model of light scattering by a class of real particles and comparison with PROGRA2 experimental results, J. Quant. Spec. Rad. Trans. 100, 256-267, 2006.

282. Loiko, V. A. and Molochko, V. I. Influence of the Director Field Structure on Extinction and Scattering by a Nematic Liquid-Crystal Droplet, Appl. Opt. 38, 2857-2861, 1999.

283. Borovoi, A. G. and Kustova, N. Light scattering by preferentially oriented ice crystals, *PIERS Online* **5** (5), 401-405, 2009.

284. Coleman, R. F. and Liou, K. N. Light scattering by hexagonal ice crystals, *J. Atmos. Sci.* **38**, 1260-1271, 1981.

285. Heymsfield, A. Cirrus uncinus generating cells and the evolution of cirriform clouds. Part I: aircraft observations of the growth of the ice phase, *J. Atmos. Sci.* **32** (4), 799-808, 1975.

286. Vincent, N. and Chepfer, H. Study of ice crystal orientation in cirrus clouds based on satellite polarized radiance measurements, *J. Atmos. Sci.* **61**, 2073-2081, 2004.

287. Um, J. and McFarquhar, G. M. Single-scattering properties of aggregates of plates, *Quarterly J. Royal Meteorological Society* **135** (639), 291-304, 2009.

288. Ou, S. C., et al. Remote sensing of cirrus cloud particle size and optical depth using polarimetric sensor measurements, *J. Atmos. Sci.* **62**, 4371-4383, 2005.

289. Ulanowski, Z. Ice analog halos, *Appl. Opt.* **44**, 5754-5758, 2005.

290. Product catalogue, LOBA CHEMIE PVT. LTD., Mumbai, India. Weblink : http://www. lobachemie .com/downloads/pdf/lobachemie-specmanual.pdf.

291. Zhang, Z. *Computation of the scattering properties of nonspherical ice crystals*, M.Sc Thesis, Texas A&M University, 2004.

292. Baran, A. J., et al. A scattering phase function for ice cloud: tests of applicability using aircraft and satellite multi-angle multi-wavelength radiance measurements of cirrus, *Quarterly J. Royal Meteorological Society* **127** (577), 2395-2416, 2001.

293. Xie, Y., et al. Effect of the inhomogeneity of ice crystals on retrieving ice cloud optical thickness and effective particle size, *J. Geophysical Research* **114** (D11203), 12 pages, 2009.

294. Grenfell, T. C. and Warren, S. G. Representation of a nonspherical ice particle by a collection of independent spheres for scattering and absorption of radiation, *J. Geophysical Research* **104** (D24), 31,697-31,709, 1999.

295. Neshyba, S. P., et al. Representation of a nonspherical ice particle by a collection of independent spheres for scattering and absorption of radiation: 2. Hexagonal columns and plates. *J. Geophysical Research* **108** (4448), 18 pages, 2003.

296. Grenfell, T. C. Representation of a nonspherical ice particle by a collection of independent spheres for scattering and absorption of radiation: 3. Hollow columns and plates *J. Geophysical Research* **110** (D17203), 15 pages, 2005.

297. Gupta, R., et al. Scattering properties and composition of cometary dust, *Astrophysics and Space Science* **301**, 21-31, 2006.

298. Kissel, J., et al. Cometary and interstellar dust analyzer for comet Wild 2, *J. Geophysical Research* **108** (E10), 8114 (8 pages), 2003.

299. Jones, A. P. Interstellar and circumstellar grain formation and survival. *Phil. Trans. R. Soc. Lond. A* **359**, 1961-1972, 2001.

300. Rai, R. K. and Rastogi, S. The scattering and extinction properties of nanodiamonds, *Mon. Not. R. Astron. Soc.* **401**, 2722–2728, 2010.

301. Draine, B. T. Interstellar dust grains, *Annual Review of Astronomy and Astrophysics* **41**, 241-289, 2003.

302. Dai, Z. R., et al. Possible in situ formation of meteoritic nanodiamonds in the early solar system, *Nature* **418**,157-159, 2002.

303. Mann, I. Interstellar dust in the solar System, *Annual Review of Astronomy and Astrophysics* **48**, 173-203, 2010.

304. Savage, B. D. and Sembach, K. R. Interstellar abundances from absorption-line observations with the Hubble space telescope, *Annual Review of Astronomy and Astrophysics* **34**, 279-329, 1996.

305. Hadamcik, E., et al. Light scattering by low-density agglomerates of micron-sized grains with the PROGRA2 experiment, *J. Quant. Spec. Rad. Trans.* **106** (1-3), 74-89, 2007.

306. Gupta, R., et al. Interstellar extinction by spheroidal dust grains, *Astron. Astrophys.* **441**, 555-561, 2005.

307. Vaidya, D. B., et al. Composite interstellar grains, *Monthly Notices of the Royal Astronomical Society* **379** (2), 791–800, 2007.

308. Levasseur-Regourd, A. C., et al. Light scattering by dust under microgravity conditions, *Earth, Moon, and Planets* **80** (1-3), 343-368, 1998.

309. Levasseur-Regourd, A. Polarimetry of dust in the solar system: remote observations, in-situ measurements and experimental simulations. In: photopolarimetry in remote sensing, *NATO Science Series II: Mathematics, Physics and Chemistry* **161** (7), 393-410, 2005.

310. Curtis, D. B., et al. A laboratory investigation of light scattering from representative components of mineral dust aerosol at a wavelength of 550 nm, *J. Geophys. Res.*, **113**, D08210, doi:10.1029/2007JD009387, 2008.

311. Worms, J. C., et al. Light scattering by dust particles with the PROGRA2 instrument—comparative measurements between clouds under microgravity and layers on the ground, *Planetary and Space Science* **48** (5), 493-505, 2000.

312. Volten, H., et al. Experimental scattering matrix elements of martian analog particles. In: *Electromagnetic and Light Scattering - Theory and Applications VII*, Wriedt, T. (Editor), ISBN 3-88722-579-1, 370 - 373, 2003.

313. Wurm, G., et al. Light scattering experiments with micron-sized dust aggregates: results on ensembles of SiO_2 monospheres and of irregularly shaped graphite particles, *J. Quant. Spec. Rad. Trans.* **89** (1-4), 371-384, 2004.

314. Cousins, C. S. G. Elasticity of carbon allotropes. I. Optimization, and subsequent modification, of an anharmonic Keating model for cubic diamond, *Phys. Rev. B* **67**, 024107, 2003.

315. Ramachandra, M. and Radhakrishna, K. Evaluation of mechanical and wear properties of Al-Si(12.2%)-Graphite metal matrix composite. Proceedings of the 7th Asia Pacific Industrial Engineering and Management Systems Conference, 17-20 December, 2006, Bangkok, Thailand, 2396 -2400.

316. Das, H. K., et al. Interstellar extinction and polarization–a spheroidal dust grain approach perspective, *Mon. Not. R. Astron. Soc.* **404**, 265–274, 2010.

317. Wang, R. T. andVan de Hulst, H. C. Rainbows-Mie computations and the airy approximation, *Appl Opt.* **30**, 106–17, 1991.

318. Shen, J. and Cai, X. Algorithm of Numerical Calculation on Lorentz Mie Theory. Progress In Electromagnetics Research Symposium 2005, Hangzhou, China, 691–694.

319. Lentz, W. J. Generating Bessel functions in Mie scattering calculations using continued fractions, *Appl. Opt.* **15**, 668-671, 1976.

320. Du, H. Mie-scattering calculation, *Appl. Opt.* **43** (9), 1951–1956, 2004.

321. Cachorro, V. E. and Salcedo, L. L. New improvements for Mie scattering calculations, *J. of Electromagnetic Waves and Applications.* **5**, 913-926, 1991.

322. Cai, W., et al. Direct recursion of the ratio of Bessel functions with applications to Mie scattering calculations, *J. Quant. Spec. Rad. Trans.* **109**, 2673 - 2678, 2008.

323. Cai, W., et al. Characterization of composite nanoparticles using an improved light scattering program for coated spheres, *Computer Physics Communications* **181**, 978–984, 2010.

324. Arfken, G. B. and Weber, H. J. *Mathematical methods for Physicists,* Academic Press, Elsevier, 2007.

325. Vilaplana, R., et al. A study of the scattering properties of an ensemble of rectangular prisms of different composition, size distribution and aspect ratios: A possible application to cometary dust grains? *Journal of Physics: Conference Series, Light, Dust and Chemical Evolution* **6**, 114–119, 2005.

326. Vilaplana, R., et al. Computations of the single scattering properties of an ensemble of compact and inhomogeneous rectangular prisms: implications for cometary dust, *J. Quant. Spec. Rad. Trans.* **88**, 219–231, 2004.

327. Moreno, F., et al. Comet dust as a size distribution of irregularly shaped, compact particles, *J. Quant. Spec. Rad. Trans.* **106**, 348–359, 2007.

328. Vilaplana, R., et al. Study of the sensitivity of size-averaged scattering matrix elements of nonspherical particles to changes in shape, porosity and refractive index, *J. Quant. Spec. Rad. Trans.* **100**, 415–428, 2006.

329. Measures, R. M. Laser remote sensing, *Krieger Publishing Company,* Florida, 1992.

330. Jonasz, M. 2006, Measures of polarization (www.tpdsci.com/Tpc/StoVecDegPol.php), In: Top. Part. Disp. Sci. (www.tpdsci.com).

331. Hovenier, J. W. Multiple scattering of polarized light in planetary atmosphere, *Astron. Astrophys* **13**, 7-29, 1971.

332. Mishchenko, M. I., Rosenbush, V. K., Kiselev, N. N., Lupishko, D. F., Tishkovets, V. P., Kaydash, V. G., Belskaya, I. N., Efimov, Y. S., and Shakhovskoy, N. M., 2010, Polarimetric Remote Sensing of Solar System Objects, Scientific monograph published by the academic publisher "Akademperiodyka", Kyiv (arXiv link: http://arxiv.org/abs/1010.1171v1).

333. Malinka, A. Raman lidar remote sensing of geophysical media, in light scattering reviews 2: remote sensing and inverse problems, Kokhanovsky, A. ed., Springer, Berlin, 2007.

334. Manhas, S., et al. Mueller matrix approach for determination of optical rotation in chiral turbid media in backscattering geometry, *Optics Express* **14** (1), 190- 202, 2006.

335. Hielscher, A. H et al. Diffuse backscattering Mueller matrices of highly scattering media, *Optics Express* **1** (13), 441- 453, 1997.

336. Jonasz, M. Scattering matrix, 2006 (www.tpdsci.com/Tpc/ScaMtx.php), In: Top. Part. Disp. Sci. (www.tpdsci.com).

337. Braak, C. J., et al. Parameterized scattering matrices for small particles in planetary atmospheres, *J. Quant. Spec. Rad. Trans.* **69** (5), 585-604, 2001.

338. Hecht, E. *Optics 4th Ed*, Pearson Education, India, 2003.

339. http://www.mjcopticaltech.com/Publications/ParticleCharactQuestions.htm

340. Jonasz, M. Size, shape, composition, and structure of microparticles from light scattering. In: principles, methods, and application of particle size analysis, Editor: Syvtsky, J. P. M., Cambridge University Press, Cambridge, 1991.

341. Jonasz, M. Scattering matrix measurement, 2006 (www.tpdsci.com/Tpc/ScaMtxMs.php). In: Top. Part. Disp. Sci. (www.tpdsci.com).

342. Jonasz, M. 2006. Relationship between the scattering matrix and function (www.tpdsci.com/Tpc/ScaMtxFnCnv.php). In: Top. Part. Disp. Sci. (www.tpdsci.com).

343. Munoz, O. and Hovenier, J.W. Laboratory measurements of single light scattering by ensembles of randomly oriented small irregular particles in air. A review, *J. Quant. Spec. Rad. Trans.* Article in Press, Corrected Proof.

344. Gogoi, A., et al. Mie scattering computation of spherical particles with very large size parameters using an improved program with variable speed and accuracy, *J. Modern Optics* **57** (21), 2192—2202, 2010.

345. Setien, B., et al. Spectral behavior of the linear polarization degree at right-angle scattering configuration for nanoparticle systems, *New Journal of Physics* **12**, 103031 (14pp), 2010.

346. Wnnow, D. F. and Veszelei, E. Design review of an instrument for spectroscopic total integrated light scattering measurements in the visible wavelength region, *Rev. Sci. Instrum.* **65** (2), 327-334, 1994.

347. Liou, K. N. A complementary theory of light scattering by homogeneous spheres, *Applied Mathematics and Computation* **3** (4), 331-358, 1977.

348. Riley, K. F., et al. *Mathematical Methods for Physics and Engineering. 3rd edittion,* Cambridge University Press, New York, 2006.

349. Das, G. and Karak, N. Vegetable oil-based flame retardant epoxy/clay nanocomposites, *Polym. Degrad. Stabil.* **94**, 1948-1954, 2009.

350. Gogoi, A., Rajkhowa, P., Choudhury, A. and Ahmed, G. A. Development of a software package for the analysis of electromagnetic scattering from small particles. 58–61; Editors: Muinonen, K., Penttilä, A., Lindqvist, H., Nousiainen, T. and Videen, G. Proc. of 12th Electromagnetic and Light Scattering Conference, June 28–July 2, 2010, University of Helsinki, Finland.

351. Bergaya, F. and Lagaly, G. *General introduction: clays, clay minerals, and clay science, in Handbook of Clay Science,* Bergaya, F., Theng, B. K. G. and Lagaly, G. (editors), Elsevier Ltd., United Kingdom, 2006.

352. Grim, R. E. and Guven, N. *Bentonites: geology, mineralogy and uses*, Elsevier, Amsterdam, 1978.

353. Odom, I. E. Smectite clay minerals: properties and uses, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **311**, 391-409, 1984.

354. Munoz, O., et al. Scattering matrices of volcanic ash particles of Mount St. Helens, Redoubt, and Mount Spurr Volcanoes, *J. Geophysical Research* **109**, D16201 (16 pages), 2004.

355. Volten, H., et al. Experimental light scattering by fluffy aggregates of magnesiosilica, ferrosilica, and alumina cosmic dust analogs, *Astronom. Astrophysics* manuscript no. 6744Volt, May 11, 2007.

356. Li, C., et al. Effects of surface roughness on light scattering by small particles, *J. Quant. Spec. Rad. Trans.* **89**, 123–131, 2004.

357. Barber, P. W. and Hill, S. C. Light Scattering by Particles: Computational Methods, World Scientific, Singapore 1990.

358. Moroz, A. Improvement of Mishchenko's T-matrix code for absorbing particles, Appl. Opt. 44 (17), 3604 - 3609, 2005.

359. Fu, Q., Yang, P., Sun, W. B. An accurate parameterization of the infrared radiative properties of cirrus clouds for climate models, J. Climate 11, 2223 - 2237, 1998.

360. Fu, Q., et al. Modelling of scattering and absorption by nonspherical chirrus ice particles at thermal infrared wavelengths, J. Atmospheric Sciences 56, 2937–2947, 1999.

361. Yang, P., et al. Scattering and absorption property database for nonspherical ice particles in the near-through far-infrared spectral region, *Appl. Optics* **44** (26), 5512– 5523, 2005.

362. Li, H., et al. Numerical accuracy of equivalent spherical approximations for computing ensemble-averaged scattering properties of fractal soot aggregates, *J. Quant. Spec. Rad. Trans.* **111**, 2127–2132, 2010.

363. Nuneb Nebulizer datasheet, http://www.nulife.co.in/catalog-pdf/nuneb-nebulizer.pdf.

364. Ma, X. et al. Determination of complex refractive index of polystyrene microspheres from 370 to 1610 nm, *Phys. Med. Biol.* **48**, 4165–4172, 2003.

365. Mazumder, N., et al. Luminescence studies of fresh water diatom frustules, *Indian J. Phys.* **84** (6), 665–669, 2010.

366. Gogoi, A., Choudhury, A. and Ahmed, G. A. Design considerations of a detector array incorporated laser based scattering system for particle characterization, in *Proc. of XXXIII Optical Society of India (OSI) Symposium 2007*, edited by: P P Sahu and P Deb (Tezpur University, Tezpur, Assam, India, December 18-20, 2007), 382 - 384.

367. http://www.math.ksu.edu/~bennett/jomacg/c.html, Date accessed: 21st June, 2011.

368. Muñoz, O., et al. Experimental determination of scattering matrices of dust particles at visible wavelengths: the IAA light scattering apparatus, J. Quant. Spec. Rad. Trans. 111, 187 - 196, 2010.

369. http://download.oracle.com/Javase/tutorial/ui/features/index.html

370. http://openldkJava.net/groups/swing

371. http://math.nist.gov/javanumerics/

372. http://www.netlib.org/java/f2j/

Appendices

Appendix A

Derivation of scalar wave equation

Let us express left hand side of the equation 2.4.10 as [126],

$$\left[\nabla^2 + k^2\right]\mathbf{M} = \nabla(\nabla\cdot\mathbf{M}) - \nabla\times(\nabla\times\mathbf{M}) + k^2\mathbf{M}, \tag{A.1}$$

Since $\mathbf{M}(\mathbf{r}) = \nabla\times[\mathbf{v}\psi(\mathbf{r})]$ and the divergence of the curl of any vector equals zero we get,

$$\nabla\cdot(\nabla\times\text{Any vector function}) = \nabla\cdot\mathrm{M} = 0 \tag{A.2}$$

Now $\mathbf{v}$ is a constant vector. Hence

$$\nabla\times\mathbf{v} = 0 \tag{A.3}$$

and

$$\nabla\cdot\mathbf{v} = 0 \tag{A.4}$$

Thus the equation (A.1) reads

$$\begin{aligned}
&\left[\nabla^2 + k^2\right]\mathbf{M} \\
&= -\nabla\times(\nabla\times\mathbf{M}) + k^2\mathbf{M} \\
&= -\nabla\times(\nabla\times\mathbf{M}) + k^2\mathbf{M} = -\nabla\times\{\nabla\times[\nabla\times(\mathbf{v}\psi)]\} + k^2\mathbf{M} \\
&= -\nabla\times\{\nabla\times[(\nabla\times\mathbf{v})\psi + (\nabla\psi)\times\mathbf{v}]\} + k^2\nabla\times(\mathbf{v}\psi) \qquad \text{(A.5)} \\
&= -\nabla\times\{[\nabla\times(\nabla\times\mathbf{v})\psi + (\nabla\psi)\times(\nabla\times\mathbf{v})] + \nabla\times[(\nabla\psi)\times\mathbf{v}]\} + k^2\nabla\times(\mathbf{v}\psi) \\
&= -\nabla\times\{(\nabla\psi)(\nabla\cdot\mathbf{v}) - \mathbf{v}(\nabla\cdot\nabla\psi) + (\mathbf{v}.\nabla)(\nabla\psi) - [(\nabla\psi)\cdot\nabla]\mathbf{v}\} + k^2\nabla\times(\mathbf{v}\psi) \\
&= \nabla\times\{\mathbf{v}(\nabla\cdot\nabla\psi) - (\mathbf{v}.\nabla)(\nabla\psi) + [(\nabla\psi)\cdot\nabla]\mathbf{v}\} + k^2\nabla\times(\mathbf{v}\psi)
\end{aligned}$$

Also for constant $\mathbf{v}$,

$$[(\nabla\psi)\cdot\nabla]\mathbf{v} = 0 \tag{A.6}$$

And

$$\nabla\times[(\mathbf{v}.\nabla)(\nabla\psi)] = (\mathbf{v}.\nabla)[\nabla\times(\nabla\psi)] = 0 \tag{A.7}$$

because $curl[grad(\psi)] = 0$.

Thus equation (A.5) becomes,

$$\begin{aligned}\left[\nabla^2 + k^2\right]\mathbf{M} &= \nabla \times \left\{\mathbf{v}(\nabla \cdot \nabla\psi) - (\mathbf{v}.\nabla)(\nabla\psi)\right\} + k^2 \nabla \times (\mathbf{v}\psi) \\ &= \nabla \times \left[\mathbf{v}\nabla^2\psi\right] + \nabla \times \left[\mathbf{v}\, k^2\psi\right]\end{aligned} \tag{A.8}$$

or

$$\left[\nabla^2 + k^2\right]\mathbf{M} = \nabla \times \left[\mathbf{v}\left(\nabla^2\psi + k^2\psi\right)\right] \tag{A.9}$$

Appendix B

Calculation of G_n, the ratio of Riccati-Bessel functions

The real valued spherical Bessel functions of first kind, second kind and Hankel functions (the complex valued combinations of Bessel functions) of first kind and second kind are defined respectively as [324],

$$j_n(x)=\sqrt{\frac{\pi}{2x}}J_{n+1/2}(x) \tag{B1}$$

$$y_n(x)=\sqrt{\frac{\pi}{2x}}Y_{n+1/2}(x)=(-1)^{n+1}\sqrt{\frac{\pi}{2x}}J_{-n-1/2}(x) \tag{B2}$$

$$h_n^{(1)}(x)=\sqrt{\frac{\pi}{2x}}H_{n+1/2}^{(1)}(x)=j_n(x)+iy_n(x) \tag{B3}$$

$$h_n^{(2)}(x)=\sqrt{\frac{\pi}{2x}}H_{n+1/2}^{(2)}(x)=j_n(x)-iy_n(x) \tag{B4}$$

Similarly Riccati-Bessel functions are given by the following equations,

$$\psi_n(x)=xj_n(x) \tag{B5}$$

$$\chi_n(x)=-xy_n(x) \tag{B6}$$

$$\xi_n(x)=xh_n^{(1)}(x)=\psi_n(x)-i\chi_n(x) \tag{B7}$$

$$\zeta_n(x)=xh_n^{(2)}(x)=\psi_n(x)+i\chi_n(x) \tag{B8}$$

Using relations B1, B2, B5 and B6, we can arrive at equation 2.4.114 very easily,

$$-\frac{\chi_n(x)}{\psi_n(x)}=\frac{y_n(x)}{j_n(x)}=\frac{Y_{n+1/2}(x)}{J_{n+1/2}(x)} \tag{B9}$$

Again, the Bessel function obey the following recursion relations,

$$J_{n+1/2}(x)=\frac{2n-1}{x}J_{n-1/2}(x)-J_{n-3/2}(x) \tag{B10}$$

$$Y_{n+1/2}(x)=\frac{2n-1}{x}Y_{n-1/2}(x)-Y_{n-3/2}(x) \tag{B11}$$

Dividing equation (B10) by equation (B11) we can find the recursion relation for $G_n(x)$ [322, 323] given by equation 2.4.115,

$$\frac{J_{n+1/2}(x)}{Y_{n+1/2}(x)} = \frac{\frac{2n-1}{x}J_{n-1/2}(x) - J_{n-3/2}(x)}{\frac{2n-1}{x}Y_{n-1/2}(x) - Y_{n-3/2}(x)}$$

$$= \frac{\frac{2n-1}{x} - \frac{J_{n-3/2}(x)}{J_{n-1/2}(x)}}{\frac{2n-1}{x}\frac{Y_{n-1/2}(x)}{J_{n-1/2}(x)} - \frac{Y_{n-3/2}(x)}{J_{n-3/2}(x)}\frac{J_{n-3/2}(x)}{J_{n-1/2}(x)}} \qquad \text{(B12)}$$

$$\Rightarrow \frac{1}{G_n(x)} = \frac{\frac{2n-1}{x} - r_{n-1}(x)}{\frac{2n-1}{x}G_{n-1}(x) - r_{n-1}(x)G_{n-2}(x)}$$

Now according to Lentz [319], the ratio of the consecutive Bessel functions is expressed as,

$$\frac{J_{n-1/2}(x)}{J_{n+1/2}(x)} = \frac{J_{\nu-1}(x)}{J_{\nu}(x)} = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \ldots\ldots}}} \qquad \text{(B13)}$$

where,

$$\nu = n + \frac{1}{2} \qquad \text{(B14)}$$

$$a_m = (-1)^{m+1}2(\nu + m - 1)x^{-1} \text{ and } m = 1,2,3,\ldots\ldots \qquad \text{(B15)}$$

For simplification and following Lentz a notation for continued fraction can be defined as,

$$r = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \ldots\ldots}}} = [a_1, a_2, a_3, a_4, \ldots\ldots] \qquad \text{(B16)}$$

Similarly, the nth partial convergent is given by the following equation [319],

$$r_n = \frac{[a_1]\ldots\ldots[a_{n-1}, \ldots\ldots, a_1][a_n, \ldots\ldots, a_1]}{[a_2]\ldots\ldots[a_{n-1}, \ldots\ldots, a_1][a_n, \ldots\ldots, a_1]} \qquad \text{(B17)}$$

Using equation B17, the calculations can be initiated from a_1 and then each numerators and the denominator can be generated from the preceding one by adding it's reciprocal to the next one. The whole process can be terminated when the relative difference between the particular n^{th} denominator and numerator is less than the desired accuracy [321, 323].

Appendix C

The computer program TUMiescat.c

```
//*****************************************************************//
//TUMiescat.c: Mie scattering code for particles with arbitrary size//
//*****************************************************************//
//************************** Description ***************************//
//*****************************************************************//
// Input Parameters:                                                //
// refre    = real part of particle refractive index                //
// refim    = imaginary part of particle refractive index           //
// refmed.x = real part of medium refractive index                  //
// mthd     = 1 - use Wiscombe's rule for calculating nmax          //
//          = 2 - use Lentz's rule for calculating nmax             //
//                epson = epsilon (accuracy level)                  //
// distk    = 1 - for monodisperse particle calculations            //
//          sel = 1 - calculation using particle radius             //
//                wavel = incident wavelength                       //
//                radl  = particle radius                           //
//          sel = 2 - calculation using size parameter              //
//                xk    = size parameter                            //
// distk    = 2 - for polydisperse particle calculations            //
//          radlk = lowest grain radius                             //
//          radlh = highest grain radius                            //
//          stepk = increament step                                 //
//          ntotk = number of particles                             //
//          dstnk = 1 - gamma distribution                          //
//                parameters: rck (modal grain radius), sigmak      //
//          dstnk = 2 - normal distribution                         //
//                parameters: rgk (modal radius), sigmak (standard  //
//                deviation)                                        //
//          dstnk = 3 - lognormal distribution                      //
//                parameters: rgk (modal radius), sigmak (standard  //
//                deviation)                                        //
// distk    = 3 - for calculation of scattering properties vs       //
//          incident wavelength                                     //
//          radk   = particle size                                  //
//          wavell = lowest value of incident wavelength            //
//          wavelh = highest value of incident wavelength           //
//          stepk = increament step                                 //
// distk    = 4 - for calculation of scattering properties vs       //
//          scattering angle vs size parameter                      //
//          sel    = 1 - calculation using particle radius          //
//                wavel = incident wavelength                       //
//                xkl   = lowest value of particle size             //
//                xkh   = highest value of particle size            //
//                stepk = increament step                           //
//          sel    = 2 - calculation using size parameter           //
//                xkl   = lowest value of size parameter            //
//                xkh   = highest value of size parameter           //
//                stepk = increament step                           //
//*****************************************************************//
//*****************************************************************//
//*****************************************************************//
```

```
// Output Parameters:                                                    //
// size parameter (x), scattering angle (ang) and scattering matrix      //
// elements (S11, -S12/S11, S33/S11 and S34/S11). The scattering         //
// efficiency (QSCA), extinction efficiency(QEXT), backscattering        //
// efficiency (QBACK), absorption efficiency (QABS), radiation           //
// pressure (QPR), single scattering albedo (albedo) and                 //
// asymmetry parameter (g).                                              //
//***********************************************************************//
//***********************************************************************//

//***********************************************************************//
//************************** Main program ***************************//
//***********************************************************************//

#include<stdio.h>
#include<math.h>
#include<time.h>
#include<dos.h>
#include"cpxarith.c"
#define PI 3.14159265e+0

int distk,mthd,nmaxn;

main()
{
char filename[10];

double nr, s11tot[200], s12tot[200], s33tot[200], s34tot[200],
s11ang[200], qext, qsca, qback, gsca, sca, ext, back, gr, gsc, g, qpr,\
qabs, albedo;

struct complex refrelk,s1k[200],s2k[200],refmed;

double refre, refim, s11, s12, s33, s34, radk, radlk, radhk, wavel,\
wavell, wavelh, xk, xkl, xkh, ang, dang, s11nor, stepk, rck, alfak,\
sigmak,rgk,pol,r;

long double epson;

int nangk=91,nan,j,aj,dstnk,ntotk,sel;

void bhmie();
double dstn();

time_t first,second;

FILE *fp,*fpr,*fpp;

/* Filename to store the scattering matrix elements */

printf("\n filename =");
scanf("%s",filename);
fp = fopen(filename,"w");
fpr = fopen("crssec.dat","w");
fclose(fpr);
```

```
/* Initial input parameters */

printf("\n Enter real part of particle refractive index =");
scanf("%lf",&refre);
printf("\n Enter imaginary part of particle refractive index =");
scanf("%lf",&refim);
printf("\n Enter real part of medium refractive index =");
scanf("%lf",&refmed.x);

refrelk.x = refre;
refrelk.y = refim;
refrelk   = cpxdiv(refrelk,refmed);

/* Selection of method 1 (Wiscombe's criterion [123]) or method 2  */
/* (based on the ratio of Riccati-Bessel functions [322])          */

printf("\nPress 1 to use Wiscombe's formula\n Press 2 to use Lentz's\
rule\n");
scanf("%d",&mthd);

if(mthd==2)
      {
      printf("\n Enter epsilon as exponential (eg. 1e-7) =");
      scanf("%le",&epson);
      }

/* Option for different types of scattering calculations */

printf("\n Press 1 to calculate for monodisperse particles\n Press  \
2 to calculate for polydisperse particles\n Press 3 to calculate    \
scattering properties vs incident wavelength\n Press 4 to calculate \
scattering properties vs scattering angle vs size parameter");
scanf("%d",&distk);

if(distk==1)
      {
      /* Filename to store the scattering efficiencies */
      fpp = fopen("crssec.dat","w");
      printf("\n Press 1 to enter particle size or 2 to enter size\
      parameter");

      scanf("%d",&sel);
      if(sel==1)
            {
            printf("\n Enter particle radius =");
            scanf("%lf",&radk);
            printf("\n Enter incident wavelength =");
            scanf("%lf",&wavel);
            xk=2.0e+0 *PI*radk*(refmed.x)/wavel;
            }
      else
            {
            printf("\n Enter size parameter =");
            scanf("%lf",&xk);
            }
      first=time(NULL);
      dang=(PI/2.0e+0)/(double)(nangk-1);
      bhmie(&xk,&refrelk,&nangk,s1k,s2k,&qext,&qsca,&qback,&gsca,\
      &epson);
```

```
        g=gsca/qsca;
        qpr=qext-gsca;
        qabs=qext-qsca;
        albedo=qsca/qext;

        s11nor=(0.5e+0)*(pow(cabs(s2k[1]),2.0)+pow(cabs(s1k[1]),2.0));
        nan=2*nangk-1;

        /* Calculation of the non-zero scattering matrix elements */

        for(j=1;j<=nan;j++)
              {
              s11=(0.5e+0)*(pow(cabs(s2k[j]),2.0e+0)+ pow(cabs(s1k[j]),\
              2.0e+0));
              s12=(0.5e+0)*(pow(cabs(s2k[j]),2.0e+0)-pow(cabs(s1k[j]), \
              2.0e+0));
              pol=-s12/s11;
              s33=(cpxmult(s2k[j],conjg(s1k[j]))).x;
              s33=s33/s11;
              s34=(cpxmult(s2k[j],conjg(s1k[j]))).y;
              s34=s34/s11;
              s11=s11/s11nor;
              ang=dang*(j-1.0e+0)*(180.0e+0/PI);
              fprintf(fp,"%e,%e,%e,%e,%e\n",ang,s11,pol,s33,s34);
              printf("%e,%e,%e,%e,%e\n",ang,s11,pol,s33,s34);
              }
        second=time(NULL);
        printf("time difference = %Lf \n", difftime(second,first));
        printf("Extinction efficiency, QEXT             = %f \n",qext);
        printf("Scattering efficiency, QSCA             = %f \n",qsca);
        printf("Absorption efficiency, QABS             = %f \n",qabs);
        printf("Single scattering albedo                = %f \n",albedo);
        printf("Asymmetry parameter                     = %f \n",g);
        printf("QPR                                     = %f \n",qpr);
        printf("Backscattering efficiency, QBACK        = %f \n",qback);
        fprintf(fpp,"%f,%f,%f,%f,%f,%f,%f,%f\n",xk,qext,qsca,qabs,\
        albedo,g,qpr,qback);
        }

else if(distk==2)
        {
        /* Filename to store the scattering efficiencies */
        fpp = fopen("crssec.dat","w");
        printf("\n Press 1 for gamma distribution\n Press 2 for normal\
        distribution\nPress 3 for lognormal distribution\n");
        scanf("%d",&dstnk);

        printf("\n Enter lowest grain radius =");
        scanf("%lf",&radlk);
        printf("\n Enter highest grain radius =");
        scanf("%lf",&radhk);
        printf("\n Enter increament step=");
        scanf("%lf",&stepk);
        printf("\n Enter number of particles=");
        scanf("%d",&ntotk);
        printf("\n Enter incident wavelength =");
        scanf("%lf",&wavel);
```

```
if(dstnk==1)
      {
      printf("\n Enter modal grain radius =");
      scanf("%lf",&rck);
      printf("\n Enter alfa=");
      scanf("%lf",&alfak);
      }
else if(dstnk==2)
      {
      printf("\n Enter sigma=");
      scanf("%lf",&sigmak);
      printf("\n Enter rg=");
      scanf("%lf",&rgk);
      }
else
      {
      printf("\n Enter sigma=");
      scanf("%lf",&sigmak);
      printf("\n Enter rg=");
      scanf("%lf",&rgk);
      }

for(j=1;j<=200;j++)
      {
      s11tot[j]=0.0;
      s11ang[j]=0.0;
      s12tot[j]=0.0;
      s33tot[j]=0.0;
      s34tot[j]=0.0;
      }
ext=0.0;
sca=0.0;
back=0.0;
gsc=0.0;
gr=0.0;

for(radk=radlk;radk<=radhk;radk=radk+stepk)
      {
      nr=dstn(ntotk,radlk,radhk,radk,stepk,dstnk,rck,alfak, \
      sigmak, rgk);
      nr=stepk*nr;
      xk=2.0e+0 *PI*radk*(refmed.x)/wavel;
      dang=(PI/2.0e+0)/(double)(nangk-1);

      bhmie(&xk,&refrelk,&nangk,s1k,s2k,&qext,&qsca,&qback,  \
      &gsca,&epson);

      gr=gr+PI*pow(radk,2)*nr;
      sca=sca+PI*pow(radk,2)*nr*qsca;
      ext=ext+PI*pow(radk,2)*nr*qext;
      back=back+PI*pow(radk,2)*nr*qback;
      gsc=gsc+PI*pow(radk,2)*nr*gsca;
      qsca=sca/gr;
      qext=ext/gr;
      qback=back/gr;
      gsca=gsc/gr;
      g=gsca/qsca;
      qpr=qext-gsca;
      qabs=qext-qsca;
```

```
            albedo=qsca/qext;
            nan=2*nangk-1;

/* Calculation of the non-zero scattering matrix elements */

            for(j=1;j<=nan;j++)
                  {
                  aj=j;
                  s11=0.5e+0*(pow(cabs(s2k[j]),2.0e+0)+ \
                  pow(cabs(s1k[j]),2.0e+0));
                  s11tot[j]=s11tot[j]+s11*nr;
                  s12=(0.5e+0)*(pow(cabs(s2k[j]),2.0e+0)- \
                  pow(cabs(s1k[j]),2.0e+0));
                  s12tot[j]=s12tot[j]+s12*nr;
                  s33=(cpxmult(s2k[j],conjg(s1k[j]))).x;
                  s33tot[j]=s33tot[j]+s33*nr;
                  s34=(cpxmult(s2k[j],conjg(s1k[j]))).y;
                  s34tot[j]=s34tot[j]+s34*nr;
                  }
            }

      for(j=1;j<=nan;j++)
            {
            ang=dang*((double)j-1.0e+0)*(180.0e+0/PI);
            printf("%e,%e,%e,%e,%e\n",ang,s11tot[j]/s11tot[1], \
            -s12tot[j]/s11tot[j],s33tot[j]/s11tot[j],\
            s34tot[j]/s11tot[j]);
            fprintf(fp,"%e,%e,%e,%e,%e\n",ang,s11tot[j]/s11tot[1],\
            -s12tot[j]/s11tot[j],s33tot[j]/s11tot[j],\
            s34tot[j]/s11tot[j]);
            }
      printf("Extinction efficiency, QEXT             = %f \n",qext);
      printf("Scattering efficiency, QSCA             = %f \n",qsca);
      printf("Absorption efficiency, QABS             = %f \n",qabs);
      printf("Single scattering albedo                = %f \n",albedo);
      printf("Asymmetry parameter                     = %f \n",g);
      printf("QPR                                     = %f \n",qpr);
      printf("Backscattering efficiency, QBACK  = %f \n",qback);
      fprintf(fpp,"%f,%f,%f,%f,%f,%f,%f\n",qext,qsca,qabs,albedo,g, \
      qpr,qback);
      }

else if(distk==3)
      {
      /* Filename to store the scattering efficiencies */
      fpp = fopen("crssec.dat","a");
      printf("\n Enter particle size =");
      scanf("%lf",&radk);
      printf("\n Enter lowest value of incident wavelength =");
      scanf("%lf",&wavell);
      printf("\n Enter highest value of incident wavelength =");
      scanf("%lf",&wavelh);
      printf("\n Enter increament step =");
      scanf("%lf",&stepk);

      first=time(NULL);
      for(wavel=wavell;wavel<=wavelh;wavel=wavel+stepk)
            {
```

```
            xk=2.0e+0 *PI*radk*(refmed.x)/wavel;
            dang=(PI/2.0e+0)/(double)(nangk-1);
            bhmie(&xk,&refrelk,&nangk,s1k,s2k,&qext,&qsca,&qback, \
            &gsca,&epson);

            g=gsca/qsca;
            qpr=qext-gsca;
            qabs=qext-qsca;
            albedo=qsca/qext;

            s11nor=(0.5e+0)*(pow(cabs(s2k[1]),2.0)+pow(cabs \
            (s1k[1]),2.0));
            nan=2*nangk-1;

/* Calculation of the non-zero scattering matrix elements */

            for(j=1;j<=nan;j++)
                  {
                  s11=(0.5e+0)*(pow(cabs(s2k[j]),2.0e+0)+pow(cabs \
                  (s1k[j]),2.0e+0));
                  s12=(0.5e+0)*(pow(cabs(s2k[j]),2.0e+0)-pow(cabs \
                  (s1k[j]),2.0e+0));
                  pol=-s12/s11;
                  s33=(cpxmult(s2k[j],conjg(s1k[j]))).x;
                  s33=s33/s11;
                  s34=(cpxmult(s2k[j],conjg(s1k[j]))).y;
                  s34=s34/s11;
                  s11=s11/s11nor;
                  ang=dang*(j-1.0e+0)*(180.0e+0/PI);
                  fprintf(fp,"%e,%e,%e,%e,%e,%e\n",wavel,ang,s11, \
                  pol,s33,s34);
                  printf("%e,%e,%e,%e,%e,%e\n",xk,ang,s11,pol,s33,s34);
                  }
            printf("Extinction efficiency, QEXT = %f \n",qext);
            printf("Scattering efficiency, QSCA = %f \n",qsca);
            printf("Absorption efficiency, QABS = %f \n",qabs);
            printf("Single scattering albedo    = %f \n",albedo);
            printf("Asymmetry parameter         = %f \n",g);
            printf("QPR                         = %f \n",qpr);
            printf("Backscattering efficiency, QBACK  = %f \n",qback);
            fprintf(fpp,"%f,%f,%f,%f,%f,%f,%f,%f\n",wavel,qext, \
            qsca,qabs,albedo,g,qpr,qback);
            }
      second=time(NULL);
      printf("time difference = %lf \n",difftime(second,first));
      }



else
      {
      /* Filename to store the scattering efficiencies */
      fpp = fopen("crssec.dat","a");
      printf("\n Press 1 to enter particle size or 2 to enter size \
      parameter =");
      scanf("%d",&sel);

      if(sel==1)
            {
```

```
            printf("\n Enter lowest value of particle size =");
            scanf("%lf",&xkl);
            printf("\n Enter highest value of particle size =");
            scanf("%lf",&xkh);
            printf("\n Enter incident wavelength =");
            scanf("%lf",&wavel);
            printf("\n Enter increament step =");
            scanf("%lf",&stepk);
            }
      else
            {
            printf("\n Enter lowest value of size parameter =");
            scanf("%lf",&xkl);
            printf("\n Enter highest value of size parameter =");
            scanf("%lf",&xkh);
            printf("\n Enter increament step =");
            scanf("%lf",&stepk);
            }
      first=time(NULL);
      for(r=xkl;r<=xkh;r=r+stepk)
            {
            if(sel==1)
                  {
                  radk=r;
                  xk=2.0e+0 *PI*radk*(refmed.x)/wavel;
                  }
            else
                  {
                  xk=r;
                  }
            dang=(PI/2.0e+0)/(double)(nangk-1);
            bhmie(&xk,&refrelk,&nangk,s1k,s2k,&qext,&qsca,&qback, \
            &gsca, &epson);

            g=gsca/qsca;
            qpr=qext-gsca;
            qabs=qext-qsca;
            albedo=qsca/qext;

            s11nor=(0.5e+0)*(pow(cabs(s2k[1]),2.0)+pow(cabs \
            (s1k[1]),2.0));
            nan=2*nangk-1;

/* Calculation of the non-zero scattering matrix elements */

            for(j=1;j<=nan;j++)
                  {
                  s11=(0.5e+0)*(pow(cabs(s2k[j]),2.0e+0)+pow(cabs \
                  (s1k[j]),2.0e+0));
                  s12=(0.5e+0)*(pow(cabs(s2k[j]),2.0e+0)-pow(cabs \
                  (s1k[j]),2.0e+0));
                  pol=-s12/s11;
                  s33=(cpxmult(s2k[j],conjg(s1k[j]))).x;
                  s33=s33/s11;
                  s34=(cpxmult(s2k[j],conjg(s1k[j]))).y;
                  s34=s34/s11;
                  s11=s11/s11nor;
                  ang=dang*(j-1.0e+0)*(180.0e+0/PI);
```

```
                fprintf(fp,"%e,%e,%e,%e,%e,%e\n",xk,ang,s11, \
                pol,s33,s34);
                printf("%e,%e,%e,%e,%e,%e\n",xk,ang,s11,pol,s33,s34);
                }
          printf("Extinction efficiency, QEXT = %f \n",qext);
          printf("Scattering efficiency, QSCA = %f \n",qsca);
          printf("Absorption efficiency, QABS = %f \n",qabs);
          printf("Single scattering albedo    = %f \n",albedo);
          printf("Asymmetry parameter         = %f \n",g);
          printf("QPR                         = %f \n",qpr);
          printf("Backscattering efficiency, QBACK  = %f \n",qback);
          fprintf(fpp,"%f,%f,%f,%f,%f,%f,%f,%f\n",xk,qext,qsca, \
          qabs,albedo,g,qpr,qback);
          }
     second=time(NULL);
     printf("time difference = %Lf \n",difftime(second,first));
     }

fclose(fp);
fclose(fpp);
printf("\n***** Eureka! Calculations successfully completed *****\n");
return(0);
}

//**********************************************************************//
//************************** BHMIE function ****************************//
//**********************************************************************//
// Calculates the Mie coefficients, amplitude scattering matrix      //
// elements and efficiency factors for a given size parameter and    //
// relative refractive index.                                        //
//**********************************************************************//

void bhmie(double *x,struct complex *refrel,int *nang,struct complex \
          s1[],struct complex s2[],double *qext, double *qsca, \
          double *qback, double *gsca, long double *epsilon)
{
double amu[100],theta[100],pi[100],tau[100],pi0[100],pi1[100];
struct complex *d, y, xi, xi1, an, ann, bn, bnn, an1, an2, comp1,\
          comp2, comp3, comp4, comp5;
double psi0, psi1, psi, dn, dx, xstop, nmax, ymod, dang, rn, chi,\
          chi0, chi1, apsi1, apsi, fn, p, t, eps;
int nn, n, nnmax, j, jj;
double mxlentz();

dx=*x;
eps=*epsilon;
an1.x=*x;
an1.y=0.0e+0;
y=cpxmult(an1,*refrel);
nmax=0;

/* Method 1 (Wiscombe's criterion [123]) */

if(mthd==1)
     {
     if(*x<8)
          {
          xstop=*x+4.0*pow(*x,0.333)+1.0;
          }
```

```
      else if(*x<=4200)
            {
            xstop=*x+4.05*pow(*x,0.333)+2.0;
            }
      else
            {
            xstop=*x+4.0*pow(*x,0.333)+2.0;
            }
      }

/* Method 2 (based on the ratio of Riccati-Bessel functions [322]) */

else
      {
      mxlentz(&dx,&eps);
      xstop=nmaxn;
      }

nmax=xstop;
ymod=cabs(y);
nnmax=(int)(max(xstop,ymod))+15;
dang=(PI/2.0e+0)/(double)(*nang-1);
for(j=1;j<=*nang;j++)
      {
      theta[j]=((double)(j)-1.0e+0)*dang;
      amu[j]=cos(theta[j]);
      }

d = (struct complex *) calloc(nnmax , sizeof(struct complex));

/* d(j) calculated by using downward recursion beginning with       */
/* initial value 0.0+i*0.0 at j = nmax                              */

d[nnmax].x=0.0e+0;
d[nnmax].y=0.0e+0;

nn=nnmax-1;
for(n=1;n<=nn;n=n+1)
      {
      rn=(double)(nnmax-n+1);
      an1.x=rn;
      an1.y=0.0;
      an2.x=1.0e+0;
      an2.y=0.0e+0;
      d[nnmax-n]=cpxsub(cpxdiv(an1,y),cpxdiv(an2,cpxadd(d[nnmax- \
                  n+1],cpxdiv(an1,y))));
      }
for(j=1;j<=*nang;j++)
      {
      pi0[j]=0.0;
      pi1[j]=1.0;
      }
nn=2**nang-1;
for(j=1;j<=nn;j++)
      {
      s1[j].x=0.0;
      s1[j].y=0.0;
      s2[j].x=0.0;
      s2[j].y=0.0;
```

```
      }
/* Calculation of Ricatti-Bessel functions with real argument x      */
/* using upward recursion.                                           */

psi0  =  cos(dx);
psi1  =  sin(dx);
chi0  = -sin(*x);
chi1  =  cos(*x);
apsi1 =  psi1;
xi1.x =  apsi1;
xi1.y = -chi1;

*qsca=0.0;
*qext=0.0;
*qback=0.0;
*gsca=0.0;

n=1;

lab200:

dn=(double)(n);
rn=(double)(n);
fn=(2.0*rn+1.0)/(rn*(rn+1.0));
psi=(2.0*dn-1.0)*psi1/dx-psi0;
apsi=psi;
chi=(2.0*rn-1.0)*chi1/(*x)-chi0;
xi.x=apsi;
xi.y=-chi;

if(n>1)
      {
      ann=an;
      bnn=bn;
      }

an1=cpxdiv(d[n],*refrel);
comp1.x=rn/(*x);
comp1.y=0.0;
comp2.x=apsi;
comp2.y=0.0;
comp3.x=apsi1;
comp3.y=0.0;
an1=cpxadd(an1,comp1);
an1=cpxmult(an1,comp2);
an1=cpxsub(an1,comp3);
an2=cpxdiv(d[n],*refrel);
an2=cpxadd(an2,comp1);
an2=cpxmult(an2,xi);
an2=cpxsub(an2,xi1);
an=cpxdiv(an1,an2);
an1=cpxmult(*refrel,d[n]);
an1=cpxadd(an1,comp1);
an1=cpxmult(an1,comp2);
an1=cpxsub(an1,comp3);
an2=cpxmult(*refrel,d[n]);
an2=cpxadd(an2,comp1);
an2=cpxmult(an2,xi);
an2=cpxsub(an2,xi1);
```

```
bn=cpxdiv(an1,an2);

*qsca=*qsca+(2.0*rn+1.0)*(pow(cabs(an),2)+pow(cabs(bn),2));
*gsca=*gsca+(2.0*rn+1.0)*(cpxmult(an,conjg(bn)).x)/((rn+1.0)*rn);
if(n>1)
      {
      *gsca=*gsca+((rn-1.0)* (rn+1.0)* (cpxadd(cpxmult(ann,conjg(an)),\
            cpxmult(bnn,conjg(bn))).x)/(rn));
      }

for(j=1;j<=*nang;j++)
      {
      jj=2**nang-j;
      pi[j]=pi1[j];
      tau[j]=rn*amu[j]*pi[j]-(rn+1.0e+0)*pi0[j];
      p=pow(-1.0e+0,(double)(n-1));
      comp1.x=pi[j];
      comp1.y=0.0e+0;
      comp2.x=tau[j];
      comp2.y=0.0e+0;
      comp3.x=fn;
      comp3.y=0.0e+0;
      an1=cpxmult(an,comp1);
      an2=cpxmult(bn,comp2);
      an1=cpxmult(comp3,cpxadd(an1,an2));
      s1[j]=cpxadd(s1[j],an1);
      t=pow(-1.0e+0,(double)(n));
      an1=cpxmult(an,comp2);
      an2=cpxmult(bn,comp1);
      an1=cpxmult(comp3,cpxadd(an1,an2));
      s2[j]=cpxadd(s2[j],an1);
      if(j!=jj)
            {
            comp1.x=pi[j];
            comp1.y=0.0e+0;
            comp2.x=tau[j];
            comp2.y=0.0e+0;
            comp3.x=fn;
            comp3.y=0.0e+0;
            comp4.x=p;
            comp4.y=0.0;
            comp5.x=t;
            comp5.y=0.0;
            s1[jj]=cpxadd(s1[jj],cpxmult(comp3,cpxadd(cpxmult(an, \
                  cpxmult(comp1,comp4)),cpxmult(bn,cpxmult(comp2,\
                  comp5)))));
            s2[jj]=cpxadd(s2[jj],cpxmult(comp3,cpxadd(cpxmult \
                  (an,cpxmult(comp2,comp5)),cpxmult(bn,cpxmult(comp1,\
                  comp4)))));
            }
      }
psi0=psi1;
psi1=psi;
apsi1=psi1;
chi0=chi1;
chi1=chi;
xi1.x=apsi1;
xi1.y=-chi1;
n=n+1;
```

```
rn=(double)(n);
for(j=1;j<=*nang;j++)
      {
      pi1[j]=((2.0*rn-1.0)/(rn-1.0))*amu[j]*pi[j];
      pi1[j]=pi1[j]-rn*pi0[j]/(rn-1.0);
      pi0[j]=pi[j];
      }
if((n-1-nmax)<0)
      goto lab200;

if((n-1-nmax)>=0)
      {
      *qsca=(2.0/pow(*x,2))*(*qsca);
      *qext=(4.0/pow(*x,2))*s1[1].x;
      *qback=(4.0/pow(*x,2))*pow(cabs(s1[2*(*nang)-1]),2);
      *gsca=(4.0/pow(*x,2))*(*gsca);
      }
return;
}

//*********************************************************************//
//************************** dstn function ***************************//
// Returns the number of particles for a particular radius of the    //
// required size distribution                                        //
//*********************************************************************//

double dstn(int ntot, double radl, double radh, double rad, double \
       step,int dstn,double rc, double alfa, double sigma, double rg)
{
double l,a,b,gama=1.0,nr,rad1,rad2,rad3,rad4,nrn;

/* Gamma distribution: n(r) = ar^a exp(-br)    */

if(dstn==1)
      {
      b=alfa/rc;
      for(l=alfa;l>=1.0e+0;l=l-1.0e+0)
                  gama= gama*l;
      a=ntot/(pow(b,-(alfa+1.0))*gama);
      nr=a*pow(rad,alfa)*exp(-b*rad);
      }

/* Normal distribution: n(r) = 1/((2π)^(1/2) σ_g) exp(-(r-r_g)^2/(2σ_g^2))    */

else if(dstn==2)
      {
      rad1=rad-rg;
      rad2=pow(rad1,2.0);
      rad3=rad2/(2*pow(sigma,2));
      rad4=1/exp(rad3);
      nrn=rad4/(pow(2.0*PI,0.5)*sigma);
      nr=ntot*nrn;
      printf("nr = %lf",nr);
      }
```

```
/* Lognormal distribution: n(r) = 1/((2π)^½ r ln(σ_g)) exp(-(ln r - ln r_g)^2 / (2 ln^2(σ_g)))   */

else
      {
      rad1=log(rad/rg);
      rad2=pow(rad1,2.0);
      rad3=rad2/(2*pow(sigma,2));
      rad4=1/exp(rad3);
      nrn=rad4/(rad*(pow(2.0*PI,0.5)*sigma));
      nr=ntot*nrn;
      }
return(nr);
}


//*********************************************************************//
//************************ mxlentz function ***********************//
//****************** Cqlculates the value of n_max ******************//
//*********************************************************************//

double mxlentz(double *x, long double *epsn)
{
double z2,*gn;

int n;

void gnn();

z2=*x;
n=(int)max(z2,3);

gn = (double *) calloc(n+2 , sizeof(double));

do
      {
      gnn(&n,&z2,gn);
      if((abs((int)(gn[n]))>(1.0/(*epsn))))
            {
            break;
            }
      else
            {
            n=n+1;
            }
      }while(abs((int)(gn[n]))<(1.0/(*epsn)));

nmaxn=n;

return(nmaxn);
}

//*********************************************************************//
//************************ dstn function ***********************//
```

```
//*** Calculates G_n(x) using method 2 described in section 2.4.4 ***//
//*****************************************************************//

void gnn(int *nn,double *z2,double gn[])
{
int nmaxcf=500000,m,i;
double machinezero=1.0e-100,eps=1.0e-16,nu,*rn;
struct complex z,zz,zz1,zz2,*eta1,*a,cftop,cftop2,cfbottom,fn,i1;
struct complex zeta0,zeta1,zeta2,zeta3,zetax2,zetax3;
struct complex xi0,xi1;
double apsi0,apsi1,psi0,psi1,chi0,chi1;

eta1 = (struct complex *) calloc(*nn , sizeof(struct complex));
a = (struct complex *) calloc(nmaxcf , sizeof(struct complex));
rn = (double *) calloc(*nn , sizeof(double));


nu=(double)*nn+0.5;

z.x=*z2;
z.y=0.0;
zz.x=1.0;
zz.y=0.0;

for(m=1;m<=2;m++)
      {
      a[m].x=pow((-1),(m+1))*2.0*(nu+m-1)*cpxdiv(zz,z).x;
      a[m].y=pow((-1),(m+1))*2.0*(nu+m-1)*cpxdiv(zz,z).y;;
      }

cftop=a[1];
cftop2.x=0.0;
cftop2.y=0.0;
cfbottom=a[2];
fn=cpxdiv(a[1],a[2]);

/* The method of continued fraction starts */

m=1;

do
      {
      a[m+2].x=pow((-1),(m+3))*2.0*(nu+m+1.0)*cpxdiv(zz,z).x;
      a[m+2].y=pow((-1),(m+3))*2.0*(nu+m+1.0)*cpxdiv(zz,z).y;
      cftop=cpxadd(a[m+1],cpxdiv(zz,cftop));
      cftop2=cpxadd(a[m+2],cpxdiv(zz,cftop));
      cfbottom=cpxadd(a[m+2],cpxdiv(zz,cfbottom));
      fn=cpxdiv((cpxmult(fn,cftop)),cfbottom);
      m=m+1;

      if (m==nmaxcf-1)
            {
            printf("Max number of terms reached in the calculation \
            of Eta1\n");
            printf("time to increase nmaxcf\n");
            }
      }while((cabs(cpxsub(cftop2,cfbottom)))>eps);
```

```
fn=cpxmult(fn,cftop2);

/* The NN'th term of the Eta1 */
zz1.x=*nn*zz.x;
zz1.y=*nn*zz.y;
eta1[*nn]=cpxsub(fn,cpxdiv(zz1,z));

/* backward recursion to calculate the rest of the Eta1 terms */
for(i=*nn-1;i>=1;i=i-1)
      {
      zz2.x=(1.0+i)*zz.x;
      zz2.y=(1.0+i)*zz.y;
      eta1[i]=cpxsub(cpxdiv(zz2,z),cpxdiv(zz,(cpxadd(cpxdiv(zz2,z), \
            eta1[i+1]))));
      }



/* Calculations of rn starts */

for(i=1;i<=*nn;i++)
      {
      i1.x=i*zz.x;
      i1.y=i*zz.y;
      rn[i]=cpxadd(eta1[i],cpxdiv(i1,z)).x;
      }

/* Calculation of Ricatti-Bessel functions */

*z2=z.x;

/* Initial values of Riccati-Bessel function */

psi0   =  cos(*z2);
psi1   =  sin(*z2);
chi0   = -sin(*z2);
chi1   =  cos(*z2);
zeta0.x=psi0;
zeta0.y=chi0;
zeta1.x=psi1;
zeta1.y=chi1;
apsi0=psi0;
apsi1=psi1;
xi0.x =  apsi0;
xi0.y = -chi0;
xi1.x = apsi1;
xi1.y = -chi1;

gn[-1]=zeta0.x/zeta0.y;
gn[0]=zeta1.x/zeta1.y;
zetax2.x=(1/(*z2))*zeta1.x;
zetax2.y=(1/(*z2))*zeta1.y;
zeta2=cpxsub(zetax2,zeta0);

gn[1]=zeta2.x/zeta2.y;
zetax3.x=(3/(*z2))*zeta2.x;
zetax3.y=(3/(*z2))*zeta2.y;
zeta3=cpxsub(zetax3,zeta1);
gn[2]=zeta3.x/zeta3.y;
```

```
for(i=3;i<=*nn;i++)
      {
      gn[i]=(((2*i-1)/(*z2))-rn[i-1])/((((2*i-1)/(*z2))/gn[i-1]) \
            -(rn[i-1]/gn[i-2]));
      }

for(i=-1;i<=*nn;i++)
      {
      gn[i]=1/gn[i];
      }
}
```

Appendix C

The associated file cpxarith.c

```
#include<math.h>
#include<stdlib.h>
#include<stdio.h>
struct complex cpxadd(struct complex z, struct complex w)
{
      struct complex Z;
      Z.x=z.x+w.x;
      Z.y=z.y+w.y;
      return Z;
}
struct complex cpxsub(struct complex z, struct complex w)
{
      struct complex Z;
      Z.x=z.x-w.x;
      Z.y=z.y-w.y;
      return Z;
}
struct complex cpxmult(struct complex z, struct complex w)
{
      struct complex Z;
      Z.x=(z.x)*(w.x)-(z.y)*(w.y);
      Z.y=(z.y)*(w.x)+(z.x)*(w.y);
      return Z;
}
struct complex cpxdiv(struct complex n, struct complex d)
{
      struct complex Z;
      double temp1,temp2;

      if(cabs(d)<(5.0e-40)*cabs(n)){
            fprintf(stderr,"division by zero in cpxdiv");
            exit(1);
            }
      if(fabs(d.x)<=fabs(d.y)){
            temp1=d.x/d.y;
            temp2=d.y+temp1*d.x;
            Z.x=(temp1*n.x+n.y)/temp2;
            Z.y=(temp1*n.y-n.x)/temp2;
            }
      else{
            temp1=d.y/d.x;
            temp2=d.x+temp1*d.y;
            Z.x=(n.x+temp1*n.y)/temp2;
            Z.y=(n.y-temp1*n.x)/temp2;
            }
      return Z;
}
struct complex conjg(struct complex z)
{
      struct complex Z;
      Z.x=z.x;
      Z.y=-z.y;
      return Z;
}
```

Appendix D

The computer program TUTscat.c

```
//*********************************************************************//
//TUTiescat.c: T-matrix code for randomly oriented axially symmetric//
//******************** particles with arbitrary size ****************//
//*********************************************************************//
//**************************  Description ***************************//
//*********************************************************************//
// Input Parameters:                                                   //
// mrr      = real part of particle refractive index                   //
// mri      = imaginary part of particle refractive index              //
// lam      = wavelength of the incident light (in microns)            //
// ddelt    = accuracy of computation                                  //
// shape    = 1 for spheroids                                          //
//          = 2 for circular cylinders                                 //
// eps      = axial ratio for spheroids                                //
//          = diameter to length ratio for cylinders                   //
// ndistr   = 0 for monodisperse particles                             //
//          = 1 for gamma distribution                                 //
//          = 2 for normal distribution                                //
//          = 3 for log normal distribution                            //
// axi      = radius for monodisperse particles                        //
//          = equivalent  radius for polydisperse particles            //
// r1       = minimum radius for polydisperse particles                //
// r2       = maximum radius for polydisperse particles                //
// b        = alpha for gamma distribution                             //
//          = (sigma)^2 for normal distribution                        //
//          = [ln(sigma)]^2 for lognormal distribution                 //
//          = [ln(sigma)]^2 for lognormal distribution                 //
//*********************************************************************//
//     In addition to the above mentioned parameters the user has      //
// to set the following parameters prior to the compilation of the     //
// computer program:                                                   //
// npna     = number of scattering angles in the range [0, 180]        //
// nkmax    = determines the number of Gaussian quadrature points      //
//            and is such that nkmax+2 is the number of quadrature     //
//            points in the interval [r1, r2].                         //
// ndgs     = sets the initial value of Gauss quadrature points        //
//            (Ng = nmax*ndgs)                                         //
//*********************************************************************//
// Output Parameters:                                                  //
// size parameter (x), scattering angle (ang) and scattering matrix    //
// elements (S11, -S12/S11, S33/S11 and S34/S11). The scattering       //
// coefficient (CSCA), extinction coefficient (CEXT), backscattering   //
// coefficient (QBACK), absorption coefficient (QABS), single          //
// scattering albedo (albedo) and asymmetry parameter (g).             //
//*********************************************************************//
//*********************************************************************//
```

```
//*********************************************************************//
//************************* Main program ******************************//
//*********************************************************************//

#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include"cpxarith.c"

#define PI 3.14159265
#define npn1 100
#define npng1 300
#define npng2 2*npng1
#define npn2 2*npn1
#define npl npn2+1
#define npn3 npn1+1
#define npn4 80
#define npn5 2*npn4
#define npn6 npn4+1
#define npl1 npn5+1

// DECLARATION OF GLOBAL VARIABLES                                  //

double ***rt11,***rt12,***rt21,***rt22,***it11,***it12,***it21,\
       ***it22;

int ichoice;

double **tr1,**ti1;

double **d1, **d2, **trgq1, **trgqr, **tqr, **tq1, **r11, **r12, \
       **r21, **r22, **i11, **i12, **i21, **i22, **rg11, **rg12, \
       **rg21, **rg22, **ig11, **ig12, **ig21, **ig22;

double **j,**y,**jr,**j1,**dj,**dy,**djr,**dj1;

double **qr,**q1,**rgqr,**rgq1;

double ssign[900];

double aa, bb, a, b;


// DEFINING THE FUNCTIONS USED IN THE MAIN PROGRAM                  //

int constt (int ngauss, int nmax, int mmax,double p, double *x, \
            double *w, double *an,double **ann, double *s,\
            double *ss, int np, double eps);

int vary (double lam, double mrr, double mri, double a, double eps,\
            int np, int ngauss, double *x, double p, double *ppi, \
            double *pir, double *pii, double *r, double *dr, double\
            *ddr, double *drr, double *dri, int nmax);

int rsp1 (double *x, int ng, int ngauss, double rev, double eps, \
            int np, double *r, double *dr);
```

```
int rsp3 (double *x, int ng, int ngauss, double rev, double eps, \
            double *r, double *dr);

int bess (double *x, double *xr, double *x1, int ng, int nmax, int \
            nnmax1, int nnmax2);

int rjb (double x, double *y, double *u,int nmax, int nnmax),

int ryb (double x, double *y, double *v, int nmax);

int cjb (double xr, double x1, double *yr, double *y1, double *ur, \
            double *u1, int nmax, int nnmax);

int tmatr0 (int ngauss, double *x, double *w,double *an, double \
            **ann, double *s, double *ss,double pp1, double p1r, \
            double p11, double *r,double *dr, double *ddr, double \
            *drr, double *dr1, int nmax, int ncheck);

int tmatr (int m, int ngauss, double *x,double *w, double *an, \
            double **ann, double *s, double *ss, double pp1, \
            double p1r, double p11,double *r, double *dr, double \
            *ddr, double *drr, double *dr1, int nmax, int ncheck);

int v1g (double *x, int nmax, int m,double *dv1, double *dv2);

int tt (int nmax, int ncheck);

int prod (double **a, double **b, double **c,int ndim, int n);

int inv1 (int nmax, double **f, double **a);

int invert (int ndim, int n, double **a,double **xa, double *cond,\
            int *ipvt, double *work, double *b);

int solve (int ndim, int n, double **a,double *b, int *ipvt);

int gsp (int nmax, double csca, double lam, double *alf1, double \
            *alf2, double *alf3, double *alf4, double *bet1, \
            double *bet2, int *lmax);

int signum (void);

int ccg (int n, int n1, int nmax, int k1, int k2, double **gg);

int direct (int n, int m, int n1, int m1, int nn, int mm, double *c);

int ccgin (int n, int n1, int m, int mm, double *g);

int sarea (double d, double *rat);

int sareac (double eps, double *rat);

int gauss (int n, int ind1, int ind2,double *z, double *w);

int distrb (int nnk, double *yy, double *wy, int ndistr, double \
            aa, double bb, double gam, double r1, double r2, \
```

```
            double *reff, double *veff, double pi);

int hovenr (int ll, double *a1, double *a2, double *a3, double \
            *a4, double *b1, double *b2);

int matr (double *a1, double *a2, double *a3,double *a4, double \
            *b1, double *b2, int lmax, int npna);



main()
{
int i, l, m, n, ll, ml, nl, n2, nll, ii, n22, nk, nm, np, llm, nml, \
      nnl, nn2, nma, iax, ink, nnm, inml, nggg, ndgs, npna, nmin, \
      mmax, nmax, lmax, ixxx, llmax, nmaxl, itime, ngaus, nkmax, \
      ncheck, nnnggg, ngauss, ndistr, shape;

double p, rl, r2, zl, z2, z3, zzl, zz2, zz3, zz4, zz5, zz6, zz7, zz8, \
      gam, lam, dax, axi, rat, mri, eps, mrr, ppi, xev, pir, pii, qsc,\
      wgi, qxt, dsca, dnl, csca, reff, walb, veff, qsca, time, wgii, \
      dext, cext, qext, qscal, tilnn, qextl, trlnn, ddelt, cscat, \
      dqsca, tilnnl, axmax, trlnnl, dqext, asymm, coeffl, cextin, \
      cabsin;

double wg[1000],xg[1000],wgl[2000],xgl[2000],

double *r, *s, *w, *x, *an, *dr, *ss, *bel, *be2, *all, *al2, *al3, \
      *al4, *ddr, *dri, *drr, *betl, *bet2, *alphl, *alph2, *alph3, \
      *alph4;

double **ann;



/* OPEN FILES */

FILE *fp,*fpp;
fp = fopen("test","w");
fpp = fopen("tmatr.write","w");

/* MEMORY ALLOCATION */

r = (double *) calloc(npng2 , sizeof(double));
s = (double *) calloc(npng2 , sizeof(double));
w = (double *) calloc(npng2 , sizeof(double));
x = (double *) calloc(npng2 , sizeof(double));
an = (double *) calloc(npnl , sizeof(double));
dr = (double *) calloc(npng2 , sizeof(double));
ss = (double *) calloc(npng2 , sizeof(double));
bel = (double *) calloc(npl , sizeof(double));
be2 = (double *) calloc(npl , sizeof(double));
all = (double *) calloc(npl , sizeof(double));
al2 = (double *) calloc(npl , sizeof(double));
al3 = (double *) calloc(npl , sizeof(double));
al4 = (double *) calloc(npl , sizeof(double));
ddr = (double *) calloc(npng2 , sizeof(double));
dri = (double *) calloc(npng2 , sizeof(double));
```

```
drr = (double *) calloc(npng2 , sizeof(double));
bet1 = (double *) calloc(npl , sizeof(double));
bet2 = (double *) calloc(npl , sizeof(double));
alph1 = (double *) calloc(npl , sizeof(double));
alph2 = (double *) calloc(npl , sizeof(double));
alph3 = (double *) calloc(npl , sizeof(double));
alph4 = (double *) calloc(npl , sizeof(double));


ann = (double **) calloc(npn1 , sizeof(double *));
if(NULL == ann){free(ann); printf("Memory allocation failed while \
allocating for ann[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      ann[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == ann[l]){free(ann[l]); printf("Memory allocation \
      failed while allocating for ann[l][].\n"); exit(-1);}
      }

rt11 = (double ***)calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      rt11[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            rt11[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

rt12 = (double ***)calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      rt12[m] = (double **) calloc(npn4,sizeof(double *));
      for ( n= 0; n < npn4; n++)
            {
            rt12[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

rt21 = (double ***) calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      rt21[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            rt21[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

rt22 = (double ***)calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      rt22[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            rt22[m][n] = (double *)calloc(npn4,sizeof(double));
            }
```

```
      }

it11 = (double ***)calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      it11[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            it11[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

it12 = (double ***)calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      it12[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            it12[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

it21 = (double ***)calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      it21[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            it21[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

it22 = (double ***)calloc(npn6,sizeof(double**));
for (m = 0; m< npn6; m++)
      {
      it22[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            it22[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }


/* Input parameters */

printf("\n Enter real part of particle refractive index =");
scanf("%lf",&mrr);
printf("\n Enter imaginary part of particle refractive index =");
scanf("%lf",&mri);

printf("\nPress 0 to calculate for monodisperse particles\nPress 1 \
      to cselect gamma size distribution\nPress 2 to cselect normal\
      size distribution\nPress 3 to cselect log normal size \
      distribution\n");
scanf("%d",&ndistr);

nkmax = 5;
```

```
if(ndistr==0)
      {
      printf("\n Enter grain radius =");
      scanf("%lf",&ax1);
      ndistr=1;
      nkmax=-1,
      r1=0.9999999*ax1;
      r2=1.0000001*ax1,
      b=7;
      }

else if(ndistr==1)
      {
      printf("\n Enter minimum grain radius =");
      scanf("%lf",&r1);
      printf("\n Enter maximum grain radius =");
      scanf("%lf",&r2);
      printf("\n Enter modal grain radius=");
      scanf("%lf",&ax1);
      printf("\n Enter alfa =");
      scanf("%lf",&b);
      }

else if(ndistr==2)
      {
      printf("\n Enter minimum grain radius =");
      scanf("%lf",&r1);
      printf("\n Enter maximum grain radius =");
      scanf("%lf",&r2);
      printf("\n Enter modal grain radius=");
      scanf("%lf",&ax1);
      printf("\n Enter (sigma_g)**2=");
      scanf("%lf",&b);
      }

else
      {
      printf("\n Enter minimum grain radius =");
      scanf("%lf",&r1);
      printf("\n Enter maximum grain radius =");
      scanf("%lf",&r2);
      printf("\n Enter modal grain radius=");
      scanf("%lf",&ax1);
      printf("\n Enter [ln(sigma_g)]**2=");
      scanf("%lf",&b);
      }

printf("\n Enter incident wavelength =");
scanf("%lf",&lam);
printf("\nPress '1' for spheroids\nPress '2' for cylinder\n");
scanf("%ld",&shape);
if(shape==1)
      {
      printf("Enter ratio of horizontal to rotational axis, A/B = ");
      scanf("%lf",&eps);
      np=-1;
```

```
      }
else
      {
      printf("Enter ratio of diameter to length, D/L = ");
      scanf("%lf",&eps);
      np=-2;
      }

printf("\n Enter accuracy of computation =");
scanf("%lf",&ddelt);


p=acos(-1);

/* Input parameters that need to be set as per requirement */

/* RAT = 1 - particle size is specified in terms of equivalent -  */
/*          volume - sphere radius                                */
/*     ≠ 1 - particle size is specified in terms of equivalent -  */
/*          surface - area - sphere radius                        */
rat=1;

/* NPNA = number of equidistant scattering angles between 0° \    */
/*          and 180°                                              */
npna = 181;
ndgs = 2;

ncheck = 0;
if (np == -1 || np == -2)
      {
      ncheck = 1;
      }
if (np > 0 && pow((-1),np) == 1)
      {
      ncheck = 1;
      }
printf("ICHOICE, NCHECK = %d,%d \n",ichoice,ncheck);

if ((fabs(rat - 1.0)) > 0.00000001 && np == -1)
      {
      sarea(eps,&rat);
      }
if ((fabs(rat - 1.0)) > 0.00000001 && np == -2)
      {
      sareac(eps,&rat);
      }

if (np == -1 && eps >= 1.0)
      {
      printf("RANDOMLY ORIENTED OBLATE SPHEROIDS, A/B = %f \n",eps);
      }
if (np == -1 && eps < 1.)
      {
      printf("RANDOMLY ORIENTED PROLATE SPHEROIDS, A/B = %f \n",eps);
      }
if (np == -2 && eps >= 1.)
      {
```

```
      printf("RANDOMLY ORIENTED OBLATE CYLINDERS, D/L = %f \n",eps);
      }
if (np == -2 && eps < 1.)
      {
      printf("RANDOMLY ORIENTED PROLATE CYLINDERS, D/L = %f \n",eps);
      }
ddelt = 0.1*ddelt;

nk = (int) (nkmax + 2);
if (nk > 1000)
      {
      printf("NK = %d, I.E. IS GREATER THAN 1000 : EXECUTION \
      TERMINATED");
      }
if (nk > 1000)
      {
      exit(0);
      }

gauss(nk,0,0, xg, wg);

z1 = (r2 - r1) * 0.5;
z2 = (r1 + r2) * 0.5;
z3 = r1 * 0.5;

for (i= 1; i<=nk; i++)
      {
      xg1[i-1] = z1 * xg[i-1] + z2;
      wg1[i-1] = wg[i-1] * z1;
      }

distrb(nk, xg1, wg1, ndistr, axi, b, gam, r1, r2, &reff, &veff, p),

if (fabs(rat-1)<=1e-6)
      {
      printf("EQUAL VOLUME SPHERE REFF, VEFF = %f,%f \n",reff,veff);
      }
if (fabs(rat-1)>=1e-6)
      {
      printf ("EQUAL SURFACE AREA SPHERE REFF, VEFF = %f,%f \n", reff,\
      veff);
      }
printf("NUMBER OF GAUSSIAN QUADRATURE POINTS IN SIZE AVERAGING= %d \
      \n",nk);
for (i=1;i<=npl;i++)
      {
      alph1[i - 1] = 0.;
      alph2[i - 1] = 0.;
      alph3[i - 1] = 0.;
      alph4[i - 1] = 0.;
      bet1[i - 1] = 0.;
      bet2[i - 1] = 0.;
      }
cscat = 0.;
cextin = 0.;
llmax = 0;
for (ink = 1, ink <= nk; ink++)
```

```
  {
  i = nk - ink + 1;
  a = rat * xg1[i - 1];
  xev = p * 2. * a / lam;
  ixxx =(int)(xev + pow(xev,0.333333)*4.05);
  inm1 = max(4,ixxx);
  if (inm1 >= npn1)
        {
        printf("CONVERGENCE IS NOT OBTAINED FOR NPN1 = %d :\
                    EXECUTION TERMINATED \n",npn1);
        }
if (inm1 >= npn1)
        {
        exit(0);
        }
qext1 = 0.0;
qsca1 = 0.0;
for (nma = inm1; nma <= npn1; nma++)
        {
        nmax = nma;
        mmax = 1;
        ngauss = nmax * ndgs;
        if (ngauss > npng1)
              {
              printf("NGAUSS = %d I.E. IS GREATER THAN NPNG1 :\
                          EXECUTION TERMINATED");
              }
        if (ngauss > 300)
              {
              exit(0);
              }

        constt (ngauss, nmax, mmax, p, x, w, an, ann, s, ss, \
                    np, eps);
        vary (lam, mrr, mri, a, eps, np, ngauss, x, p, &ppi, \
                    &pir, &pii, r, dr, ddr, drr, dri, nmax);
        tmatr0 (ngauss, x, w, an, ann, s, ss, ppi, pir, pii, \
                    r, dr, ddr, drr, dri, nmax, ncheck);
        qext = 0.0;
        qsca = 0.0;
        for (n = 1; n <= nmax; n++)
              {
              n1 = n + nmax;
              trlnn = trl[n-1][n-1];
              tilnn = til[n-1][n-1];
              trlnn1 = trl[n1-1][n1-1];
              tilnn1 = til[n1-1][n1-1];
              dn1 = (double) (2*n + 1);
              qsca = qsca + dn1 * (trlnn * trlnn + tilnn * tilnn \
                    + trlnn1 * trlnn1 + tilnn1 * tilnn1);
              qext = qext + (trlnn + trlnn1) * dn1;
              }
        dsca = (fabs((qsca1 - qsca) / qsca));
        dext = (fabs((qext1 - qext) / qext));

        qext1 = qext;
        qsca1 = qsca;
```

```
            nmin = (int) ((double) nmax / 2.0 + 1.);
            for (n = nmin; n <= nmax; ++n)
                  {
                  n1 = n + nmax;
                  tr1nn = tr1[n-1][n-1];
                  ti1nn = ti1[n-1][n-1];
                  tr1nn1 = tr1[n1-1][n1-1];
                  ti1nn1 = ti1[n1-1][n1-1];
                  dn1 = (double) (2*n + 1);
                  dqsca = dn1 * (tr1nn * tr1nn + ti1nn * ti1nn + \
                        tr1nn1 * tr1nn1 + ti1nn1 * ti1nn1);
                  dqext = (tr1nn + tr1nn1) * dn1;
                  dqsca = (fabs(dqsca / qsca));
                  dqext = (fabs(dqext / qext));
                  nmax1 = n;
                  if (dqsca <= ddelt && dqext <= ddelt)
                        {
                        goto L12;
                        }
                  }
L12:
            if (dsca <= ddelt && dext <= ddelt)
                  {
                  goto L55;
                  }
            if (nma == 100)
                  {
                  printf("CONVERGENCE IS NOT OBTAINED FOR NPN1 = \
                        %d : EXECUTION TERMINATED \n",npn1);
                  }
            if (nma == 100)
                  {
                  exit(0);
                  }
            }

L55:
    nnnggg = ngauss + 1;
    if (ngauss == npng1)
            {
            printf("WARNING:NGAUSS=NPNG1");
            }
    mmax = nmax1;
    for (ngaus = nnnggg; ngaus <= npng1; ngaus++)
            {
            ngauss = ngaus;
            nggg = 2*ngauss;

            constt (ngauss, nmax, mmax, p, x, w, an, ann, s, ss, \
                  np, eps);
            vary (lam, mrr, mri, a, eps, np, ngauss, x, p, &ppi, \
                  &pir, &pii, r, dr, ddr, drr, dri, nmax);
            tmatr0 (ngauss, x, w, an, ann, s, ss, ppi, pir, pii, \
                  r,dr, ddr, drr, dri, nmax, ncheck);
            qext = 0.0;
            qsca = 0.0;
            for (n = 1; n <= nmax; ++n)
```

```
                {
                n1 = n + nmax;
                tr1nn = tr1[n-1][n-1];
                ti1nn = ti1[n-1][n-1];
                tr1nn1 = tr1[n1-1][n1-1];
                ti1nn1 = ti1[n1-1][n1-1];
                dn1 = (double) (2*n + 1);
                qsca = qsca + dn1 * (tr1nn * tr1nn + ti1nn * \
                      ti1nn + tr1nn1 * tr1nn1 + ti1nn1 * ti1nn1);
                qext = qext + (tr1nn + tr1nn1) * dn1;
                }
          dsca = (fabs((qsca1 - qsca) / qsca));
          dext = (fabs((qext1 - qext) / qext));

          qext1 = qext;
          qsca1 = qsca;
          if (dsca <= ddelt && dext <= ddelt)
                {
                goto L155;
                }
          if (ngaus == npng1)
                {
                printf("WARNING:NGAUSS=NPNG1");
                }
          }

L155:
     qsca = 0.0;
     qext = 0.0;
     nnm = 2*nmax;
     for (n = 1; n <=nnm; ++n)
          {
          qext = qext + tr1[n-1][n-1];
          }
     if (nmax1 > npn4)
          {
          printf("NMAX = %d , I.E. GREATER THAN NPN4: EXECUTION\
                      TERMINATED");
          }
     if (nmax1 > npn4)
          {
          exit(0);
          }
     for (n2 = 1; n2 <= nmax1; n2++)
          {
          nn2 = n2 + nmax;
          for (n1 = 1; n1 <= nmax1; n1++)
                {
                nn1 = n1 + nmax;
                zz1 = tr1[n1-1][n2-1];
                rt11[0][n1-1][n2-1] = zz1;
                zz2 = ti1[n1-1][n2-1];
                it11[0][n1-1][n2-1]= zz2;
                zz3 = tr1[n1-1][nn2-1];
                rt12[0][n1-1][n2-1] = zz3;
                zz4 = ti1[n1-1][nn2-1];
                it12[0][n1-1][n2-1] = zz4;
```

```
            zz5 = tr1[nn1-1][n2-1];
            rt21[0][n1-1][n2-1] = zz5;
            zz6 = ti1[nn1-1][n2-1];
            it21[0][n1-1][n2-1] = zz6;
            zz7 = tr1[nn1-1][nn2-1];
            rt22[0][n1-1][n2-1] = zz7;
            zz8 = ti1[nn1-1][nn2-1];
            it22[0][n1-1][n2-1] = zz8;
            qsca = qsca + zz1 * zz1 + zz2 * zz2 + zz3 * zz3 \
                  + zz4 * zz4 + zz5 * zz5 + zz6 * zz6 + zz7 \
                  * zz7 + zz8 * zz8;
            }
      }

for (m = 1; m <= nmax1; ++m)
      {
      tmatr (m, ngauss, x, w, an, ann, s, ss, ppi, pir, pii,\
                  r, dr, ddr, drr, dri, nmax, ncheck);
      nm = nmax - m + 1;
      nm1 = nmax1 - m + 1;
      m1 = m + 1;
      qsc = 0.0;
      for (n2 = 1; n2 <= nm1; n2++)
            {
            nn2 = n2 + m - 1;
            n22 = n2 + nm;
            for (n1 = 1; n1 <= nm1; ++n1)
                  {
                  nn1 = n1 + m - 1;
                  n11 = n1 + nm;
                  zz1 = tr1[n1-1][n2-1];
                  rt11[m1-1][nn1-1][nn2-1] = zz1;
                  zz2 = ti1[n1-1][n2-1];
                  it11[m1-1][nn1-1][nn2-1] = zz2;
                  zz3 = tr1[n1-1][n22-1];
                  rt12[m1-1][nn1-1][nn2-1] = zz3;
                  zz4 = ti1[n1-1][n22-1];
                  it12[m1-1][nn1-1][nn2-1] = zz4;
                  zz5 = tr1[n11-1][n2-1];
                  rt21[m1-1][nn1-1][nn2-1] = zz5;
                  zz6 = ti1[n11-1][n2-1];
                  it21[m1-1][nn1-1][nn2-1] = zz6;
                  zz7 = tr1[n11-1][n22-1];
                  rt22[m1-1][nn1-1][nn2-1] = zz7;
                  zz8 = ti1[n11-1][n22-1];
                  it22[m1-1][nn1-1][nn2-1] = zz8;
                  qsc = qsc+ (zz1 * zz1 + zz2 * zz2 + zz3 * zz3 \
                        + zz4 * zz4 + zz5 * zz5 + zz6 * zz6 + \
                        zz7 * zz7 + zz8 * zz8) * 2.;
                  }
            }
      nnm = 2*nm;
      qxt = 0.0;
      for (n = 1; n <= nnm; n++)
            {
            qxt = qxt+tr1[n-1][n-1] * 2.0;
            }
```

```
            qsca = qsca+qsc;
            qext = qext+qxt;
            }
      coeff1 = lam * lam * 0.5 / p;
      csca = qsca * coeff1;
      cext = -qext * coeff1;

      gsp(nmax1, csca, lam, al1, al2, al3, al4, be1, be2, &lmax);

      l1m = lmax + 1;

      l1max = max(l1max,l1m);
      wgii = wg1[i - 1];

      wgi = wgii * csca;
      for (l1 = 1; l1 <= l1m; ++l1)
            {
            alph1[l1 - 1] = alph1[l1 - 1]+al1[l1 - 1] * wgi;
            alph2[l1 - 1] = alph2[l1 - 1]+al2[l1 - 1] * wgi;
            alph3[l1 - 1] = alph3[l1 - 1]+al3[l1 - 1] * wgi;
            alph4[l1 - 1] = alph4[l1 - 1]+al4[l1 - 1] * wgi;
            bet1[l1 - 1] = bet1[l1 - 1]+be1[l1 - 1] * wgi;
            bet2[l1 - 1] = bet2[l1 - 1]+be2[l1 - 1] * wgi;
            }
      cscat = cscat+wgi;
      cextin = cextin+cext * wgii;
      }
for (l1 = 1; l1 <= l1max; ++l1)
      {
      alph1[l1 - 1] = alph1[l1 - 1]/cscat;
      alph2[l1 - 1] = alph2[l1 - 1]/cscat;
      alph3[l1 - 1] = alph3[l1 - 1]/cscat;
      alph4[l1 - 1] = alph4[l1 - 1]/cscat;
      bet1[l1 - 1] = bet1[l1 - 1]/cscat;
      bet2[l1 - 1] = bet2[l1 - 1]/cscat;
      }
walb = cscat / cextin;
cabsin=cextin-cscat;
hovenr(l1max, alph1, alph2, alph3, alph4, bet1, bet2);
asymm = alph1[1] / 3.0;
printf("CEXT,CSCA,CABS,W,<COS> = %f,%f,%f,%f,%f \n", cextin, cscat,\
      cabsin, walb, asymm);
if (walb > 1.)
      {
      printf("WARNING: W IS GREATER THAN 1");
      }

lmax = l1max - 1;
matr(alph1, alph2, alph3, alph4, bet1, bet2, lmax, npna);

return 0;
}
```

```
//*********************************************************************//
//************************ CONSTT function ****************************//
//*********************************************************************//
// Input parameters :                                                  //
// ng = 2*ngauss - number Gauss quadrature points in the interval      //
//                 (-1, 1)                                             //
// nmax, mmax = maximum dimension of the arrays                        //
// p = π = arcos(-1)                                                   //
// Output parameters:                                                  //
// x = Gauss quadrature points = cos(θ)                                //
// w = weight of Gauss quadrature formula                              //
// an(n) = n*(n+1)                                                     //
// ann(n1,n2) = 1/2 sqrt( {2(n1)+1}{2(n2)+1} / (n1){(n1)+1}(n2){2(n2)+1} )   //
// s(i) = 1/sin[arccos{x(i)}] = 1/sin(θ)                               //
// ss(i) = [s(i)]^2 = 1/sin^2(θ)                                       //
//*********************************************************************//

int constt (int ngauss, int nmax, int mmax,double p, double *x, \
            double *w, double *an,double **ann, double *s, double \
            *ss, int np, double eps)
{
/* Local variables */

int i, n, n1, ng, nn, ng1, ng2;
double d, y, dd[100], xx, ddd;
double *w1, *x1, *x2, *w2;

/* Memory Allocation */

w1 = (double *) calloc(npng2 , sizeof(double));
w2 = (double *) calloc(npng2 , sizeof(double));
x1 = (double *) calloc(npng2 , sizeof(double));
x2 = (double *) calloc(npng2 , sizeof(double));

/* Function Body */

for (n = 1; n <= nmax; ++n)
      {
      nn = n * (n + 1);
      an[n-1] = (double) nn;
      d = sqrt ((double) (2*n + 1) / (double) nn);
      dd[n - 1] = d;
      for (n1 = 1; n1 <= n; ++n1)
            {

            ddd = d * dd[n1 - 1] * 0.5;
            ann[n-1][n1-1] = ddd;
            ann[n1-1][n-1] = ddd;
            }
      }
```

```
ng = 2*(ngauss);
if (np == -2)
      {
      goto L11;
      }
gauss (ng, 0, 0, x, w);
goto L19;
L11:
ng1 = (int) ((double) (ngauss) / 2.0);
ng2 = ngauss - ng1;
xx = -cos (atan(eps));
gauss (ng1, 0, 0, x1, w1);
gauss (ng2, 0, 0, x2, w2);
for (i = 1; i <= ng1; ++i)
      {
      w[i-1] = (xx + 1.0) * 0.5 * w1[i - 1];
      x[i-1] = (xx + 1.0) * 0.5 * x1[i - 1] + (xx - 1.0) * 0.5;
      }
for (i= 1; i<=ng2; ++i)
      {
      w[i+ ng1-1] = xx * (-0.5) * w2[i - 1];
      x[i+ ng1-1] = xx * (-0.5) * x2[i - 1] + xx * 0.5;
      }
for (i = 1; i <= ngauss; ++i)
      {
      w[ng - i] = w[i-1];
      x[ng - i] = -x[i-1];
      }
L19:
for (i = 1; i <= ngauss; ++i)
      {
      y = x[i-1];
      y = 1.0 / (1.0 - y * y);
      ss[i-1] = y;
      ss[ng - i] = y;
      y = sqrt(y);
      s[i-1] = y;
      s[ng - i] = y;
/* L20: */
      }
free(w1);
free(x1);
free(w2);
free(x2);
return 0;
}

//*********************************************************************//
//************************* VARY function *****************************//
//*********************************************************************//
// Input parameters: lam, mrr, mri, a, eps, np, ngauss, x, p = π,   //
//                   nmax                                              //
// Output parameters:                                                  //
//          ppi = pi * pi (2π/λ)^2                                     //
```

```
//          pir = ppi * mrr                                              //
//          pii = ppi * mri                                              //
//          r   = [r(θ)]² and dr = dr(θ)/dθ / r(θ)      (see functions RSP1 //
//                and RSP3)                                              //
//          ddr = λ / 2π{r(θ)}^(1/2)                                     //
//          drr = [ mrr / (mrr)² + (mri)² ] × ddr                        //
//          dri = [ -mri / (mrr)² + (mri)² ] × ddr                       //
//          nmax = dimension of T(m) matrix                              //
//***************************************************************//

int vary (double lam, double mrr, double mri, double a, double eps, \
          int np, int ngauss, double *x, double p, double *ppi, \
          double *pir, double *pii, double *r, double *dr, double\
          *ddr, double *drr, double *dri, int nmax)
{
/* Local variables */

int i,ng,nnmax1, nnmax2;
double v, v1, v2,ta, tb, pi, vv, pri, prr;
double *z,*zi,*zr;

/* Memory allocation */
z = (double *) calloc(npng2 , sizeof(double));
zi = (double *) calloc(npng2 , sizeof(double));
zr = (double *) calloc(npng2 , sizeof(double));

/* Function Body */

ng = 2*(ngauss);
if (np == -1)
      {
      rsp1(x, ng, ngauss, a, eps, np, r, dr);
      }
if (np == -2)
      {
      rsp3(x, ng, ngauss, a, eps, r, dr);
      }
pi = p * 2. / lam;
*ppi = pi * pi;
*pir = *ppi * mrr;
*pii = *ppi * mri;
v = 1.0 / (mrr * mrr + mri * mri);
prr = mrr * v;
pri = -(mri) * v;

ta = 0.;
for (i = 1; i <= ng; ++i)
```

```
      {
      vv = sqrt(r[i-1]);
      v = vv * pi;
      ta = max(ta,v);
      vv = 1.0 / v;
      ddr[i-1] = vv;
      drr[i-1] = prr * vv;
      dri[i-1] = pri * vv;
      v1 = v * mrr;
      v2 = v * mri;
      z[i - 1] = v;
      zr[i - 1] = v1;
      zi[i - 1] = v2;
      }

if (nmax > 100)
      {
      printf("NMAX = %d, I.E. GREATER THAN NPN1 = %d \n",nmax,npn1);
      }
if (nmax > 100)
      {
      exit(0);
      }
tb = ta * sqrt(mrr * mrr + mri * mri);

tb = max(tb,(double) (nmax));


nnmax1 = (int) (sqrt((max(ta,(double) (nmax)))) * 1.2 + 3.0);
nnmax2 = (int) (tb + pow(tb, 0.33333) * 4.0 + sqrt(tb) * 1.2);
nnmax2 = nnmax2 - nmax + 5;

bess(z, zr, zi, ng, nmax, nnmax1, nnmax2);

free(z);
free(zi);
free(zr);

return 0;
}

//*********************************************************************//
//************************ RSP1 function ************************//
//*********************************************************************//
// This function is activated for np = -1 (spheroids)                  //
// Shape of a spheroid is given by:                                    //
```

// $r(\theta,\phi) = a\left[\sin^2\theta + \frac{a^2}{b^2}\cos^2\theta\right]^{-\frac{1}{2}}$ //

```
// This function calculates                 ,                          //
```

// $r(i) = [r(y)]^2$ //

// and $dr(i) = \dfrac{dr(y)}{dy} \Big/ r(y)$ //

```
//                  here   y = arccos(x) = θ                              //
// Input parameters: x, ng, ngauss, rev, eps,                             //
//                                                                        //
// Output parameters: r, dr                                               //
//************************************************************************//

int rsp1(double *x, int ng, int ngauss,double rev, double eps, \
      int np, double *r,double *dr)
{
/* Local variables */

int i;
double a,c,s,aa,cc,ee,rr,ss,ee1;

/* Function Body */

a = rev * pow(eps, 1.0/3.0);
aa = a * a;
ee = eps * eps;
ee1 = ee - 1.0;
for (i = 1; i <= ngauss; ++i)
      {
      c = x[i-1];
      cc = c * c;
      ss = 1.0 - cc;
      s = sqrt(ss);
      rr = 1.0 / (ss + ee * cc);
      r[i-1] = aa * rr;
      r[ng - i] = r[i-1];
      dr[i-1] = rr * c * s * ee1;
      dr[ng - i] = -dr[i-1];
    }
return 0;
}

//************************************************************************//
//************************** RSP3 function ********************************//
//************************************************************************//
// This function is activated for np = -2 (cylinder)                      //
// If h is the half length of the cylinder, then  the radius of the       //
// cylinder is given by,                                                  //
//                          a = h × eps                                   //
// Thus,                                                                  //
//                 4π × (rev)^3 / 3 = 2h × πa^2 = 2π × h^3 × (eps)^2      //
//                                                                        //
//                 ⇒ h = rev × [ 2 / (3 × (eps)^2) ]^(1/3)                //
// This function calculates                                               //
//                          r(i) = [r(y)]^2                               //
```

```
//                and   dr(i) = (dr(y)/dy) / r(y)                          //
//                here  y = arccos(x)                                  //
// Input parameters: x, ng, ngauss, rev, eps,                          //
//                                                                     //
// Output parameters: r, dr                                            //
//*********************************************************************//

int rsp3(double *x, int ng, int ngauss,double rev, double eps, double
*r, double *dr)
{
/* Local variables */

int i;
double a, h, co, si, rad, rthet;

/* Function Body */

h = rev * pow((2.0 / (eps * 3.0 * eps)), 1.0/3.0);
a = h * eps;
for (i = 1; i <= ngauss; ++i)
      {
      co = -x[i-1];
      si = sqrt(1.0 - co * co);
      if (si / co > a / h)
            {
          goto L20;
            }
      rad = h / co;
      rthet = h * si / (co * co);
      goto L30;
L20:
      rad = a / si;
      rthet = -a * co / (si * si);
L30:
      r[i-1] = rad * rad;
      r[ng - i] = r[i-1];
      dr[i-1] = -rthet / rad;
      dr[ng - i] = -dr[i-1];
    }
return 0;
}


//*********************************************************************//
//************************** BESS function ****************************//
//*********************************************************************//
// Calaulates spherical Bessel functions of the first kind             //
// j(i,n) = j_n(x) and second kind y(i,n) = y_n(x) of real valued argument  //
// x(i) and first kind jr(i,n) + i*ji(i,n) = j_n(z) of complex argument     //
// z(i) = xr(i) + i*xi(i). The function also calculates:                //
```

```
//                    dj(i,n) = (1/x) d/dx [x × j_n(x)]                    //
//                    dy(i,n) = (1/x) d/dx [x × y_n(x)]                    //
//                    djr(i,n) = Re[ (1/z) d/dz {z × j_n(x)} ]             //
//                    dji(i,n) = Im[ (1/z) d/dz {z × j_n(x)} ]             //
// Input parameters: x, xr, x1, ng, nmax, nnmax1, nnmax2                   //
// Output parameters: j, y, jr, j1, dj, dy, djr, dj1                       //
//*************************************************************************//

int bess(double *x, double *xr, double *x1, int ng, int nmax, int \
      nnmax1, int nnmax2)
{
/* Local variables */

int i, l, n;
double *aj, *ay, y1, yr, xx, *adj, *aj1, *ajr, *ady, *adj1, *adjr;

/* Memory Allocation */
aj = (double *) calloc(npn1 , sizeof(double));
ay = (double *) calloc(npn1 , sizeof(double));
adj = (double *) calloc(npn1 , sizeof(double));
aj1 = (double *) calloc(npn1 , sizeof(double));
ajr = (double *) calloc(npn1 , sizeof(double));
ady = (double *) calloc(npn1 , sizeof(double));
adj1 = (double *) calloc(npn1 , sizeof(double));
adjr = (double *) calloc(npn1 , sizeof(double));

j = (double **) calloc(npng2 , sizeof(double *));
if(NULL == j){free(j); printf("Memory allocation failed while \
      allocating for j[].\n"); exit(-1);}
for(l = 0; l < npng2; l++)
      {
      j[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == j[l]){free(j[l]); printf("Memory allocation failed\
      while allocating for j[l][].\n"); exit(-1);}
      }

y = (double **) calloc(npng2, sizeof(double *));
if(NULL == y){free(y); printf("Memory allocation failed while \
      allocating for y[].\n"); exit(-1);}
for(l = 0; l < npng2; l++)
      {
      y[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == y[l]){free(y[l]); printf("Memory allocation failed\
      while allocating for y[l][].\n"); exit(-1);}
      }

jr = (double **) calloc(npng2 , sizeof(double *));
if(NULL == jr){free(jr); printf("Memory allocation failed while \
      allocating for jr[].\n"); exit(-1);}
```

```
for(l = 0; l < npng2; l++)
      {
      jr[l] = (double *) calloc(npnl , sizeof(double));
      if(NULL == jr[l]){free(jr[l]); printf("Memory allocation failed\
      while allocating for jr[l][].\n"); exit(-1);}
      }

j1 = (double **) calloc(npng2 , sizeof(double *));
if(NULL == j1){free(j1); printf("Memorj1 allocation failed while \
      allocating for j1[].\n"); exit(-1);}
for(l = 0; l < npng2; l++)
      {
      j1[l] = (double *) calloc(npnl , sizeof(double));
      if(NULL == j1[l]){free(j1[l]); printf("Memory allocation failed\
      while allocating for j1[l][].\n"); exit(-1);}
      }

dj = (double **) calloc(npng2 , sizeof(double *));
if(NULL == dj){free(dj); printf("Memory allocation failed while \
      allocating for dj[].\n"); exit(-1);}
for(l = 0; l < npng2; l++)
      {
      dj[l] = (double *) calloc(npnl , sizeof(double));
      if(NULL == dj[l]){free(dj[l]); printf("Memory allocation failed\
      while allocating for dj[l][].\n"); exit(-1);}
      }

dy = (double **) calloc(npng2, sizeof(double *));
if(NULL == dy){free(dy); printf("Memory allocation failed while \
      allocating for dy[].\n"); exit(-1);}
for(l = 0; l < npng2; l++)
      {
      dy[l] = (double *) calloc(npnl , sizeof(double));
      if(NULL == dy[l]){free(dy[l]); printf("Memory allocation failed\
      while allocating for dy[l][].\n"); exit(-1);}
      }

djr = (double **) calloc(npng2 , sizeof(double *)),
if(NULL == djr){free(djr); printf("Memory allocation failed while \
      allocating for djr[].\n"), exit(-1);}
for(l = 0, l < npng2; l++)
      {
      djr[l] = (double *) calloc(npnl , sizeof(double));
      if(NULL == djr[l]){free(djr[l]);printf("Memory allocation failed\
      while allocating for djr[l][].\n"); exit(-1);}
      }

dj1 = (double **) calloc(npng2 , sizeof(double *));
if(NULL == dj1){free(dj1); printf("Memordj1 allocation failed while \
      allocating for dj1[].\n"); exit(-1);}
for(l = 0, l < npng2; l++)
      {
      dj1[l] = (double *) calloc(npnl , sizeof(double));
      if(NULL == dj1[l]){free(dj1[l]);printf("Memory allocation failed\
      while allocating for dj1[l][].\n"); exit(-1);}
      }
```

```
/* Function Body */

for (i = 1; i <= ng; ++i)
      {
      xx = x[i-1];
      rjb(xx, aj, adj, nmax, nnmax1);
      ryb(xx, ay, ady, nmax);
      yr = xr[i-1];
      yi = xi[i-1];
      cjb(yr, yi, ajr, aji, adjr, adji, nmax, nnmax2);
      for (n = 1; n <= nmax; ++n)
            {
            j[i-1][n-1] = aj[n - 1];
            y[i-1][n-1] = ay[n - 1];
            jr[i-1][n-1] = ajr[n - 1];
            ji[i-1][n-1] = aji[n - 1];

            dj[i-1][n-1] = adj[n - 1];
            dy[i-1][n-1] = ady[n - 1],
            djr[i-1][n-1] = adjr[n - 1],
            dji[i-1][n-1] = adji[n - 1];
            }
      }
free(aj);
free(ay);
free(adj);
free(ady);
free(ajr);
free(aji);
free(adjr);
free(adji);

return 0;
}

//*****************************************************************//
//************************** RJB function *************************//
//*****************************************************************//
// Calculates spherical Bessel functions of the first kind J_n(x) of //
// real valued argument x of orders from 1 to nmax by using backward//
// recursion where nmax determines the numerical accuracy.          //
//*****************************************************************//

int rjb(double x, double *y, double *u,int nmax, int nnmax)
{
/* Local variables */

int i, l, i1, l1;
double z[800],y0, z0, y1, yi, xx, yi1;

/* Function Body */

l = nmax + nnmax;
xx = 1.0 / x;
z[l - 1] = 1.0 / ((double) (2*l + 1) * xx);
l1 = l - 1;
```

```
for (i = 1; i<= l1; ++i)
      {
      i1 = l - i;
      z[i1 - 1] = 1.0 / ((double) (2*i1 + 1) * xx - z[i1]);
      }
z0 = 1.0 / (xx - z[0]);
y0 = z0 * cos(x) * xx;
y1 = y0 * z[0];
u[0] = y0 - y1 * xx;
y[0] = y1;
for (i = 2; i <= nmax; ++i)
      {
      yi1 = y[i - 2];
      yi = yi1 * z[i - 1];
      u[i-1] = yi1 - (double) i * yi * xx;
      y[i-1] = yi;
      }
return 0;
}


//*********************************************************************//
//************************** RYB function *****************************//
//*********************************************************************//
// Calculates spherical Bessel functions of the second kind y_n(x) of//
// real valued argument x of orders from 1 to nmax by using upward  //
// recursion where nmax determines the numerical accuracy.           //
//*********************************************************************//

int ryb(double x, double *y, double *v, int nmax)
{
/* Local variables */

int i,nmax1;
double c,s, x1, x2, x3, y1;

/* Function Body */

c = cos(x);
s = sin(x);
x1 = 1.0 / x;
x2 = x1 * x1;
x3 = x2 * x1;
y1 = -c * x2 - s * x1;
y[0] = y1,
y[1] = (x3 * -3.0 + x1) * c - x2 * 3.0 * s;
nmax1 = nmax - 1;
for (i= 2; i<= nmax1; ++i)
      {
/* L5: */
      y[i] = (double) (2*i + 1) * x1 * y[i-1] - y[i - 2];
      }
v[0] = -x1 * (c + y1);
for (i= 2; i <= nmax; ++i)
      {
/* L10: */
```

```
      v[i-1] = y[i - 2] - (double) i * x1 * y[i-1];
    }
return 0;
}


//*****************************************************************//
//************************** CJB function *************************//
//*****************************************************************//
// Calculates spherical Bessel functions of the first kind J_n(z) of //
// complex argument z of orders from 1 to nmax by using backward    //
// recursion where nmax determines the numerical accuracy.          //
//*****************************************************************//

int cjb(double xr, double xi, double *yr, double *yi, double *ur, \
      double *ui, int nmax, int nnmax)
{
/* Local variables */
int i, l, i1, l1;
double ai, ci, ar, cr, qf, qi, ari, *cui, *cyi, czi[1200], *cur, \
      *cyr, czr[1200], cu1i, cy0i, cz0i, cy1i, cu1r, cz0r, cy0r, \
      cy1r, cuii, cyii, cuir, cyir, cxxi, cxxr, xrxi,cyi1i, cyi1r;


/* Memory Allocation */
cui = (double *) calloc(npn1 , sizeof(double));
cyi = (double *) calloc(npn1 , sizeof(double));
cur = (double *) calloc(npn1 , sizeof(double));
cyr = (double *) calloc(npn1 , sizeof(double));


l = nmax + nnmax;
xrxi = 1.0 / (xr * xr + xi * xi);
cxxr = xr * xrxi;
cxxi = -(xi) * xrxi;
qf = 1.0 / (double) (2*l + 1);
czr[l - 1] = xr * qf;
czi[l - 1] = xi * qf;
l1 = l - 1;
for (i = 1; i <=l1; ++i)
      {
      i1 = l - i;
      qf = (double) (2*i1 + 1);
      ar = qf * cxxr - czr[i1];
      ai = qf * cxxi - czi[i1];
      ari = 1.0 / (ar * ar + ai * ai);
      czr[i1 - 1] = ar * ari;
      czi[i1 - 1] = -ai * ari;
      }
ar = cxxr - czr[0];
ai = cxxi - czi[0];
ari = 1.0 / (ar * ar + ai * ai);
cz0r = ar * ari;
cz0i = -ai * ari;
cr = cos(xr) * cosh(xi);
ci = -sin(xr) * sinh(xi);
```

```
ar = cz0r * cr - cz0i * ci;
ai = cz0i * cr + cz0r * ci;
cy0r = ar * cxxr - ai * cxxi;
cy0i = ai * cxxr + ar * cxxi;
cy1r = cy0r * czr[0] - cy0i * czi[0];
cy1i = cy0i * czr[0] + cy0r * czi[0];
ar = cy1r * cxxr - cy1i * cxxi;
ai = cy1i * cxxr + cy1r * cxxi;
cu1r = cy0r - ar;
cu1i = cy0i - ai;
cyr[0] = cy1r;
cyi[0] = cy1i;
cur[0] = cu1r;
cui[0] = cu1i;
yr[0] = cy1r;
yi[0] = cy1i;
ur[0] = cu1r,
ui[0] = cu1i;
for (i = 2, i <= nmax; ++i)
      {
      qi = (double) i;
      cyi1r = cyr[i - 2];
      cyi1i = cyi[i - 2];
      cyir = cyi1r * czr[i - 1] - cyi1i * czi[i - 1];
      cyii = cyi1i * czr[i - 1] + cyi1r * czi[i - 1];
      ar = cyir * cxxr - cyii * cxxi;
      ai = cyii * cxxr + cyir * cxxi,
      cuir = cyi1r - qi * ar;
      cuii = cyi1i - qi * ai;
      cyr[i - 1] = cyir;
      cyi[i - 1] = cyii;
      cur[i - 1] = cuir;
      cui[i - 1] = cuii;
      yr[i-1] = cyir;
      yi[i-1] = cyii;
      ur[i-1] = cuir;
      ui[i-1] = cuii;
      }
return 0;
}

//*****************************************************************//
//************************* TMATR0 function ***********************//
//*****************************************************************//
// Determines the T-matrix for an axially symmetric scatterer for  //
// M = 0.                                                          //
// ngauss, x, w are related to the Gauss quadrature formula.       //
// In the program the refractive index outside the particle (medium)//
// is assumed to be real whereas inside the scatterer the refractive//
// index is complex. Thus the Bessel functions of complex argument z//
// will be for the interior region of the particle.                //
//*****************************************************************//

int tmatr0(int ngauss, double *x, double *w,double *an, double \
           **ann, double *s, double *ss,double ppi, double pir, \
           double pii, double *r,double *dr, double *ddr, double\
           *drr, double *dri, int nmax, int ncheck)
```

```
{
/* Local variables */

int i, l, n,i1, i2, k1, k2, n1, n2,ng,nm,mm1,kk1, kk2,ngss,nnmax;
double f1, f2,a12, a21, a22,si,*rr,aa1, dd1, dd2, b1i, c1i, c2i,\
      b2i, an1, an2, c3i, b3i,c4i, b1r, c1r, c2r, b2r, c3r, b3r,\
      *dv1, *dv2,qj1, c4r, b4r, b4i, c5r, c5i, b5r, b5i,qy1, \
      ai12, ai21, ar12, ar21, gi12, *sig, gr12, gr21, gi21, d1n1, \
      d2n1, d1n2, d2n2, uri, rri, an12, qdj1, qji2, qjr2, qdy1, \
      tai12, tai21, ddri, tgi12, drii, tgi21, tar12, tar21, tgr12,\
      drri, tgr21, tpii,tppi, tpir, qdji2, qdjr2, factor;

/* Memory Allocation */

rr = (double *) calloc(npng2 , sizeof(double));
dv1 = (double *) calloc(npn1 , sizeof(double));
dv2 = (double *) calloc(npn2 , sizeof(double)),
sig = (double *) calloc(npn2 , sizeof(double)),

d1 = (double **) calloc(npng2 , sizeof(double *));
if(NULL == d1){free(d1); printf("Memory allocation failed while\
      allocating for d1[].\n"); exit(-1);}
for(l = 0; l < npng2; l++)
      {
      d1[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == d1[l]){free(d1[l]); printf("Memory allocation failed\
      while allocating for d1[l][].\n"); exit(-1);}
      }

d2 = (double **) calloc(npng2 , sizeof(double *)),
if(NULL == d2){free(d2); printf("Memory allocation failed while \
      allocating for d2[].\n"); exit(-1);}
for(l = 0; l < npng2, l++)
      {
      d2[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == d2[l]){free(d2[l]); printf("Memory allocation failed\
      while allocating for d2[l][].\n"); exit(-1);}
      }

tqi = (double **) calloc(npn2 , sizeof(double *));
if(NULL == tqi){free(tqi); printf("Memory allocation failed while \
      allocating for tqi[].\n"); exit(-1);}
for(l = 0, l < npn2; l++)
      {
      tqi[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == tqi[l]){free(tqi[l]);printf("Memory allocation failed\
      while allocating for tqi[l][].\n"); exit(-1);}
      }

tqr = (double **) calloc(npn2 , sizeof(double *));
if(NULL == tqr){free(tqr);printf("Memory allocation failed while \
      allocating for tqr[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      tqr[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == tqr[l]){free(tqr[l]);printf("Memory allocation failed\
      while allocating for tqr[l][].\n"); exit(-1);}
```

```
trgq1 = (double **) calloc(npn2 , sizeof(double *));
if(NULL == trgq1){free(trgq1); printf("Memory allocation failed while \
      allocating for trgq1[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      trgq1[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == trgq1[l]){free(trgq1[l]);printf("Memory allocation\
      failed while allocating for trgq1[l][].\n"); exit(-1);}
      }

trgqr = (double **) calloc(npn2 , sizeof(double *));
if(NULL == trgqr){free(trgqr); printf("Memory allocation failed while \
      allocating for trgqr[].\n"); exit(-1);}
for(l = 0, l < npn2; l++)
      {
      trgqr[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == trgqr[l]){free(trgqr[l]); printf("Memory allocation\
      failed while allocating for trgqr[l][].\n"); exit(-1);}
      }

r11 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == r11){free(r11); printf("Memory allocation failed while \
      allocating for r11[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      r11[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == r11[l]){free(r11[l]);printf("Memory allocation failed\
      while allocating for r11[l][].\n"); exit(-1),}
      }

r12 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == r12){free(r12); printf("Memory allocation failed while \
      allocating for r12[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      r12[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == r12[l]){free(r12[l]);printf("Memory allocation failed\
      while allocating for r12[l][].\n"); exit(-1);}
      }

r21 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == r21){free(r21); printf("Memory allocation failed while \
      allocating for r21[].\n"), exit(-1);}
for(l = 0; l < npn1; l++)
      {
      r21[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == r21[l]){free(r21[l]);printf("Memory allocation failed\
      while allocating for r21[l][].\n"), exit(-1),}
      }

r22 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == r22){free(r22), printf("Memory allocation failed while \
      allocating for r22[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
```

```
      r22[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == r22[l]){free(r22[l]);printf("Memory allocation failed\
      while allocating for r22[l][].\n"); exit(-1);}
      }

i11 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == i11){free(i11); printf("Memory allocation failed while \
      allocating for i11[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      i11[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == i11[l]){free(i11[l]);printf("Memory allocation failed\
      while allocating for i11[l][].\n"); exit(-1);}
      }

i12 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == i12){free(i12); printf("Memory allocation failed while \
      allocating for i12[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      i12[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == i12[l]){free(i12[l]);printf("Memory allocation failed\
      while allocating for i12[l][].\n"); exit(-1);}
      }

i21 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == i21){free(i21); printf("Memory allocation failed while \
      allocating for i21[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      i21[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == i21[l]){free(i21[l]);printf("Memory allocation failed\
      while allocating for i21[l][].\n"); exit(-1);}
      }

i22 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == i22){free(i22); printf("Memory allocation failed while \
      allocating for i22[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      i22[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == i22[l]){free(i22[l]);printf("Memory allocation failed\
      while allocating for i22[l][].\n"); exit(-1),}
      }

rg11 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == rg11){free(rg11); printf("Memory allocation failed while \
      allocating for rg11[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      rg11[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == rg11[l]){free(rg11[l]); printf("Memory allocation \
      failed while allocating for rg11[l][].\n"); exit(-1);}
      }

rg12 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == rg12){free(rg12); printf("Memory allocation failed while \
```

```
      allocating for rg12[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      rg12[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == rg12[l]){free(rg12[l]); printf("Memory allocation \
      failed while allocating for rg12[l][].\n"); exit(-1);}
      }

rg21 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == rg21){free(rg21); printf("Memory allocation failed while \
      allocating for rg21[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      rg21[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == rg21[l]){free(rg21[l]); printf("Memory allocation \
      failed while allocating for rg21[l][].\n"); exit(-1);}
      }

rg22 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == rg22){free(rg22); printf("Memory allocation failed while \
      allocating for rg22[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      rg22[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == rg22[l]){free(rg22[l]); printf("Memory allocation \
      failed while allocating for rg22[l][].\n"); exit(-1);}
      }

ig11 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == ig11){free(ig11); printf("Memory allocation failed while \
      allocating for ig11[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      ig11[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == ig11[l]){free(ig11[l]); printf("Memory allocation \
      failed while allocating for ig11[l][].\n"); exit(-1);}
      }

ig12 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == ig12){free(ig12); printf("Memory allocation failed while \
      allocating for ig12[].\n"); exit(-1);}
for(l = 0, l < npn1; l++)
      {
      ig12[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == ig12[l]){free(ig12[l]); printf("Memory allocation \
      failed while allocating for ig12[l][].\n"); exit(-1);}
      }

ig21 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == ig21){free(ig21); printf("Memory allocation failed while \
      allocating for ig21[].\n"); exit(-1),}
for(l = 0; l < npn1; l++)
      {
      ig21[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == ig21[l]){free(ig21[l]); printf("Memory allocation \
      failed while allocating for ig21[l][].\n"); exit(-1);}
      }
```

```
ig22 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == ig22){free(ig22); printf("Memory allocation failed while \
      allocating for ig22[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      ig22[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == ig22[l]){free(ig22[l]); printf("Memory allocation \
      failed while allocating for ig22[l][].\n"); exit(-1);}
      }

qr = (double **) calloc(npn2 , sizeof(double *));
if(NULL == qr){free(qr); printf("Memory allocation failed while \
      allocating for qr[] \n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      qr[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == qr[l]){free(qr[l]); printf("Memory allocation failed \
      while allocating for qr[l][].\n"); exit(-1);}
      }

qi = (double **) calloc(npn2 , sizeof(double *));
if(NULL == qi){free(qi); printf("Memory allocation failed while \
      allocating for qi[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      qi[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == qi[l]){free(qi[l]); printf("Memory allocation failed \
      while allocating for qi[l][] \n"); exit(-1);}
      }

rgqi = (double **) calloc(npn2 , sizeof(double *));
if(NULL == rgqi){free(rgqi); printf("Memory allocation failed while \
      allocating for rgqi[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      rgqi[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == rgqi[l]){free(rgqi[l]); printf("Memory allocation \
      failed while allocating for rgqi[l][].\n"); exit(-1);}
      }

rgqr = (double **) calloc(npn2 , sizeof(double *));
if(NULL == rgqr){free(rgqr); printf("Memory allocation failed while \
      allocating for rgqr[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      rgqr[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == rgqr[l]){free(rgqr[l]); printf("Memory allocation \
      failed while allocating for rgqr[l][].\n"); exit(-1);}
      }

trgqi = (double **) calloc(npn2 , sizeof(double *));
if(NULL == trgqi){free(trgqi); printf("Memory allocation failed while \
      allocating for trgqi[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      trgqi[l] = (double *) calloc(npn2 , sizeof(double));
```

```
      if(NULL == trgqi[l]){free(trgqi[l]); printf("Memory allocation \
      failed while allocating for trgqi[l][].\n"); exit(-1);}
      }

trgqr = (double **) calloc(npn2 , sizeof(double *));
if(NULL == trgqr){free(trgqr); printf("Memory allocation failed while \
      allocating for trgqr[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      trgqr[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == trgqr[l]){free(trgqr[l]); printf("Memory allocation \
      failed while allocating for trgqr[l][].\n"); exit(-1);}
      }



/* Function Body */

mm1 = 1;
nnmax = nmax + nmax;
ng = 2*(ngauss);
ngss = ng;
factor = 1.0;
if (ncheck == 1)
      {
      ngss = ngauss;
      factor = 2.0;
      }
si = 1.0;

/* determination of (-1)^n */

for (n = 1; n <= nnmax; ++n)
      {
      si = -si;
      sig[n - 1] = si;
      }
for (i = 1; i<= ngauss; ++i)
      {
      i1 = ngauss + i;
      i2 = ngauss - i + 1;

      vig(&x[i1-1], nmax, 0, dv1, dv2);
      for (n = 1; n <=nmax; ++n)
            {
            si = sig[n - 1];
            dd1 = dv1[n - 1];
            dd2 = dv2[n - 1];
            d1[i1-1][n-1] = dd1;
            d2[i1-1][n-1] = dd2;
            d1[i2-1][n-1] = dd1 * si;
            d2[i2-1][n-1] = -dd2 * si;
            }
      }

for (i = 1; i <= ngss; ++i)
```

```
      {
      rr[i - 1] = w[i-1] * r[i-1];
      }
for (n1 = mm1; n1 <=nmax; ++n1)
      {
      an1 = an[n1-1];

      for (n2 = mm1; n2 <= nmax; ++n2)
            {
            an2 = an[n2-1];
            ar12 = 0.0;
            ar21 = 0.0;
            ai12 = 0.0;
            ai21 = 0.0;
            gr12 = 0.0;
            gr21 = 0.0;
            gi12 = 0.0,
            gi21 = 0.0;
            if (ncheck == 1 && sig[n1 + n2 - 1] < 0.0)
                  {
                  an12 = ann[n1-1][n2-1] * factor;
                  r12[n1-1][n2-1] = ar12 * an12;
                  r21[n1-1][n2-1] = ar21 * an12;
                  i12[n1-1][n2-1] = ai12 * an12;
                  i21[n1-1][n2-1] = ai21 * an12;
                  rg12[n1-1][n2-1] = gr12 * an12;
                  rg21[n1-1][n2-1] = gr21 * an12;
                  ig12[n1-1][n2-1] = gi12 * an12;
                  ig21[n1-1][n2-1] = gi21 * an12;
                  }
            else
                  {
                  for (i = 1; i <= ngss; ++i)
                        {
                        d1n1 = d1[i-1][n1-1];
                        d2n1 = d2[i-1][n1-1];
                        d1n2 = d1[i-1][n2-1];
                        d2n2 = d2[i-1][n2-1];

                        a12 = d1n1 * d2n2;
                        a21 = d2n1 * d1n2;
                        a22 = d2n1 * d2n2;
                        aa1 = a12 + a21,

                        qj1 = j[i-1][n1-1];
                        qy1 = y[i-1][n1-1];
                        qjr2 = jr[i-1][n2-1];
                        qji2 = ji[i-1][n2-1],
                        qdjr2 = djr[i-1][n2-1];
                        qdji2 = dji[i-1][n2-1];
                        qdj1 = dj[i-1][n1-1];
                        qdy1 = dy[i-1][n1-1];

                        c1r = qjr2 * qj1;
                        c1i = qji2 * qj1;
                        b1r = c1r - qji2 * qy1;
                        b1i = c1i + qjr2 * qy1;
```

```
                    c2r = qjr2 * qdj1;
                    c2i = qji2 * qdj1;
                    b2r = c2r - qji2 * qdy1;
                    b2i = c2i + qjr2 * qdy1;

                    ddri = ddr[i-1];
                    c3r = ddri * c1r;
                    c3i = ddri * c1i;
                    b3r = ddri * b1r;
                    b3i = ddri * b1i;

                    c4r = qdjr2 * qj1;
                    c4i = qdji2 * qj1;
                    b4r = c4r - qdji2 * qy1;
                    b4i = c4i + qdjr2 * qy1;

                    drri = drr[i-1],
                    drii = dri[i-1],
                    c5r = c1r * drri - c1i * drii;
                    c5i = c1i * drri + c1r * drii;
                    b5r = b1r * drri - b1i * drii;
                    b5i = b1i * drri + b1r * drii;

                    uri = dr[i-1];
                    rri = rr[i - 1];

                    f1 = rri * a22;
                    f2 = rri * uri * an1 * a12;
                    ar12 = ar12 + f1 * b2r + f2 * b3r;
                    ai12 = ai12 + f1 * b2i + f2 * b3i;
                    gr12 = gr12 + f1 * c2r + f2 * c3r;
                    gi12 = gi12 + f1 * c2i + f2 * c3i;

                    f2 = rri * uri * an2 * a21;
                    ar21 = ar21 + f1 * b4r + f2 * b5r;
                    ai21 = ai21 + f1 * b4i + f2 * b5i;
                    gr21 = gr21 + f1 * c4r + f2 * c5r;
                    gi21 = gi21 + f1 * c4i + f2 * c5i;
                    }
                an12 = ann[n1-1][n2-1] * factor;
                r12[n1-1][n2-1] = ar12 * an12;
                r21[n1-1][n2-1] = ar21 * an12;
                i12[n1-1][n2-1] = ai12 * an12;
                i21[n1-1][n2-1] = ai21 * an12;
                rg12[n1-1][n2-1] = gr12 * an12;
                rg21[n1-1][n2-1] = gr21 * an12;
                ig12[n1-1][n2-1] = gi12 * an12;
                ig21[n1-1][n2-1] = gi21 * an12,
                }
            }
        }

/* Forming Q and RgQ matrices */

tpir = pir;
tpii = pii;
```

```
tppi = ppi;

nm = nmax;
for (n1 = mm1; n1 <= nmax; ++n1)
      {
      k1 = n1 - mm1+1;
      kk1 = k1 + nm;
      for (n2 = mm1; n2 <= nmax, ++n2)
            {
            k2 = n2 - mm1+1;
            kk2 = k2 + nm;
            tar12 = ai12[n1-1][n2-1];
            tai12 = -r12[n1-1][n2-1];
            tgr12 = aig12[n1-1][n2-1],
            tgi12 = -rg12[n1-1][n2-1];

            tar21 = -ai21[n1-1][n2-1];
            tai21 = r21[n1-1][n2-1];
            tgr21 = -aig21[n1-1][n2-1];
            tgi21 = rg21[n1-1][n2-1];

            tqr[k1-1][k2-1] = tpir * tar21 - tpii * tai21 + \
                              tppi * tar12;
            tqi[k1-1][k2-1] = tpir * tai21 + tpii * tar21 + \
                              tppi * tai12;
            trgqr[k1-1][k2-1] = tpir * tgr21 - tpii * tgi21 + \
                              tppi * tgr12;
            trgqi[k1-1][k2-1] = tpir * tgi21 + tpii * tgr21 + \
                              tppi * tgi12,

            tqr[k1-1][kk2-1]= 0.0;
            tqi[k1-1][kk2-1]= 0.0;
            trgqr[k1-1][kk2-1]= 0.0;
            trgqi[k1-1][kk2-1]= 0.0;

            tqr[kk1-1][k2-1] = 0.0;
            tqi[kk1-1][k2-1] = 0.0;
            trgqr[kk1-1][k2-1] = 0.0;
            trgqi[kk1-1][k2-1] = 0.0;

            tqr[kk1-1][kk2-1] = tpir * tar12 - tpii * tai12 + \
                              tppi * tar21;
            tqi[kk1-1][kk2-1] = tpir * tai12 + tpii * tar12 + \
                              tppi * tai21;
            trgqr[kk1-1][kk2-1] = tpir * tgr12 - tpii * tgi12 + \
                              tppi * tgr21;
            trgqi[kk1-1][kk2-1] = tpir * tgi12 + tpii * tgr12 + \
                              tppi * tgi21;
            }
    }

nnmax = 2*nm;
for (n1 = 1; n1 <= nnmax; ++n1)
      {
      for (n2 = 1; n2 <= nnmax; ++n2)
            {
            qr[n1-1][n2-1] = tqr[n1-1][n2-1];
```

```
            q1[n1-1][n2-1] = tq1[n1-1][n2-1];
            rgqr[n1-1][n2-1] = trgqr[n1-1][n2-1];
            rgq1[n1-1][n2-1] = trgq1[n1-1][n2-1],
            }
    }

/* Forming T = -Q^-1 x RgQ matrix */

tt(nmax, ncheck);

return 0;
}


//*****************************************************************//
//************************* TMATR function ************************//
//*****************************************************************//
// Determines the T-matrix for an axially symmetric scatterer for   //
// M > 0                                                            //
// ngauss, x, w are related to the Gauss quadrature formula         //
// If ncheck = 0, then ngss = 2*ngauss and factor = 1.0             //
// If ncheck = 1, then ngss = ngauss and factor = 2.0               //
// In the program the refractive index outside the particle (medium)//
// is assumed to be real whereas inside the scatterer the refractive//
// index is complex. Thus the Bessel functions of complex argument z//
// will be for the interior region of the particle.                 //
//*****************************************************************//

int tmatr(int m, int ngauss, double *x,double *w, double *an, \
      double **ann, double *s, double *ss, double pp1, double \
      p1r, double p11,double *r, double *dr, double *ddr, double\
      *drr, double *dr1, int nmax, int ncheck)
{
/* Local variables */

int i, n,i1, i2,k1, n1, n2, k2,ng,nm,mm1,kk1, kk2,ngss,nnmax;
double e1, f1, f2,e2, e3,a11, a12, a21, a22, *ds,qm, s1, *rr, wr, \
      aa1, aa2, dd1, dd2,b11, c11, c21, b21, an1, an2, c31, b31, \
      c41, b1r, c1r, c2r, b2r, c3r, b3r, *dv1, *dv2, qj1, c4r, b4r,\
      b41, c5r, c51, b5r, b51, c6r, c61, b6r, qy1, b61, c7r, c71, \
      b7r, b71, c8r, c81, b8r, b81,a111, a112, a121, a122, ar11, \
      ar12, ar21,*sig,ar22, gr11,gr12, *dss, qmm, gr21, gr22, gi11,\
      gi12, gi21, gi22, d1n1, d2n1, d1n2, d2n2, ur1, ds1, rr1, an12,\
      qdj1, qj12, qjr2, qdy1, ta111, ta112, ta121, ddr1, ta122, \
      tgi11, tgi12, dri1, tar11, tar12, tar21, tgi21, tar22, tgi22,\
      tgr11, tgr12, drr1, tgr21, dss1, tpi1, tgr22,tpp1, tp1r, \
      qdj12, qdjr2,factor;

/ *Memory Allocation */

rr = (double *) calloc(npng2 , sizeof(double));
ds = (double *) calloc(npng2 , sizeof(double));
dv1 = (double *) calloc(npn1 , sizeof(double));
dv2 = (double *) calloc(npn1 , sizeof(double));
sig = (double *) calloc(npn2 , sizeof(double));
dss = (double *) calloc(npng2 , sizeof(double));
```

```
/* Function Body */

mm1 = m;
qm = (double) (m);
qmm = qm * qm;
ng = 2*(ngauss);
ngss = ng;
factor = 1.;
if (ncheck == 1)
      {
      ngss = ngauss;
      factor = 2.;
      }
si = 1.;
nm = nmax + nmax;

/* determination of (-1)^n */

for (n = 1; n <= nm; ++n)
      {
      si = -si;
      sig[n - 1] = si;
      }
for (i = 1; i <= ngauss; ++i)
      {
      i1 = ngauss + i;
      i2 = ngauss - i + 1;
      vig(&x[i1-1], nmax, m, dv1, dv2);
      for (n = 1; n <= nmax; ++n)
            {
            si = sig[n - 1];
            dd1 = dv1[n - 1];
            dd2 = dv2[n - 1];
            d1[i1-1][n-1] = dd1;
            d2[i1-1][n-1] = dd2;
            d1[i2-1][n-1] = dd1 * si;
            d2[i2-1][n-1] = -dd2 * si;
            }
      }
for (i = 1; i <= ngss; ++i)
      {
      wr = w[i-1] * r[i-1];
      ds[i - 1] = s[i-1] * qm * wr;
      dss[i - 1] = ss[i-1] * qmm;
      rr[i - 1] = wr;
    }
for (n1 = mm1; n1 <= nmax; ++n1)
      {
      an1 = an[n1-1];
      for (n2 = mm1; n2 <= nmax; ++n2)
            {
            an2 = an[n2-1];
            ar11 = 0.0;
            ar12 = 0.0;
            ar21 = 0.0;
            ar22 = 0.0;
```

```
aı11 = 0.0,
aı12 = 0.0;
aı21 = 0.0;
aı22 = 0.0;
gr11 = 0.0;
gr12 = 0.0;
gr21 = 0.0;
gr22 = 0.0;
gi11 = 0.0;
gı12 = 0.0;
gı21 = 0.0;
gı22 = 0.0;
sı = sıg[n1 + n2 - 1];

for (ı = 1; ı<= ngss; ++ı)
      {
      d1n1 = d1[ı-1][n1-1];
      d2n1 = d2[ı-1][n1-1];
      d1n2 = d1[ı-1][n2-1];
      d2n2 = d2[ı-1][n2-1];
      a11 = d1n1 * d1n2,
      a12 = d1n1 * d2n2;
      a21 = d2n1 * d1n2;
      a22 = d2n1 * d2n2;
      aa1 = a12 + a21;
      aa2 = a11 * dss[ı - 1] + a22;
      qȷ1 = ȷ[ı-1][n1-1];
      qy1 = y[ı-1][n1-1];
      qȷr2 = ȷr[ı-1][n2-1];
      qȷı2 = ȷı[ı-1][n2-1];
      qdȷr2 = dȷr[ı-1][n2-1];
      qdȷı2 = dȷı[ı-1][n2-1];
      qdȷ1 = dȷ[ı-1][n1-1];
      qdy1 = dy[ı-1][n1-1];

      c1r = qȷr2 * qȷ1;
      c1ı = qȷı2 * qȷ1;
      b1r = c1r - qȷı2 * qy1;
      b1ı = c1ı + qȷr2 * qy1;

      c2r = qȷr2 * qdȷ1;
      c2ı = qȷı2 * qdȷ1;
      b2r = c2r - qȷı2 * qdy1;
      b2ı = c2ı + qȷr2 * qdy1;

      ddrı = ddr[ı-1];
      c3r = ddrı * c1r;
      c3ı = ddrı * c1ı;
      b3r = ddrı * b1r;
      b3ı = ddrı * b1ı;

      c4r = qdȷr2 * qȷ1,
      c4ı = qdȷı2 * qȷ1;
      b4r = c4r - qdȷı2 * qy1;
      b4ı = c4ı + qdȷr2 * qy1;

      drrı = drr[ı-1];
```

```
dri1 = dri[i-1];
c5r = c1r * drri - c1i * dri1;
c5i = c1i * drri + c1r * dri1;
b5r = b1r * drri - b1i * dri1;
b5i = b1i * drri + b1r * dri1;

c6r = qdjr2 * qdj1;
c6i = qdji2 * qdj1;
b6r = c6r - qdji2 * qdy1;
b6i = c6i + qdjr2 * qdy1;

c7r = c4r * ddri;
c7i = c4i * ddri;
b7r = b4r * ddri;
b7i = b4i * ddri;

c8r = c2r * drri - c2i * dri1;
c8i = c2i * drri + c2r * dri1;
b8r = b2r * drri - b2i * dri1;
b8i = b2i * drri + b2r * dri1;

uri = dr[i-1];
dsi = ds[i - 1];
dssi = dss[i - 1];
rri = rr[i - 1];

if (ncheck == 1)
      {
      if(si > 0.0)
            {
            f1 = rri * aa2;
            f2 = rri * uri * an1 * a12;
            ar12 = ar12 + f1 * b2r + f2 * b3r;
            ai12 = ai12 + f1 * b2i + f2 * b3i;
            gr12 = gr12 + f1 * c2r + f2 * c3r;
            gi12 = gi12 + f1 * c2i + f2 * c3i;

            f2 = rri * uri * an2 * a21;
            ar21 = ar21 + f1 * b4r + f2 * b5r;
            ai21 = ai21 + f1 * b4i + f2 * b5i;
            gr21 = gr21 + f1 * c4r + f2 * c5r;
            gi21 = gi21 + f1 * c4i + f2 * c5i;
            }
      else
            {
            e1 = dsi * aa1;
            ar11 = ar11+e1 * b1r;
            ai11 = ai11+e1 * b1i;
            gr11 = gr11+e1 * c1r;
            gi11 = gi11+e1 * c1i;

            e2 = dsi * uri * a11;
            e3 = e2 * an2;
            e2 = e2 * an1;
            ar22 = ar22 + e1 * b6r + e2 * b7r + \
                  e3 * b8r;
            ai22 = ai22 + e1 * b6i + e2 * b7i + \
```

```
                        e3 * b8i;
                  gr22 = gr22 + e1 * c6r + e2 * c7r + \
                        e3 * c8r;
                  gi22 = gi22 + e1 * c6i + e2 * c7i + \
                        e3 * c8i;
                  }
            }
      else
            {
            e1 = dsi * aa1;
            ar11 = ar11+e1 * b1r;
            ai11 = ai11+e1 * b1i;
            gr11 = gr11+e1 * c1r;
            gi11 = gi11+e1 * c1i;

            f1 = rri * aa2;
            f2 = rri * uri * an1 * a12;
            ar12 = ar12 + f1 * b2r + f2 * b3r;
            ai12 = ai12 + f1 * b2i + f2 * b3i;
            gr12 = gr12 + f1 * c2r + f2 * c3r;
            gi12 = gi12 + f1 * c2i + f2 * c3i;

            f2 = rri * uri * an2 * a21;
            ar21 = ar21 + f1 * b4r + f2 * b5r;
            ai21 = ai21 + f1 * b4i + f2 * b5i;
            gr21 = gr21 + f1 * c4r + f2 * c5r;
            gi21 = gi21 + f1 * c4i + f2 * c5i;

            e2 = dsi * uri * a11;
            e3 = e2 * an2;
            e2 = e2 * an1;
            ar22 = ar22 + e1 * b6r + e2 * b7r + \
                  e3 * b8r;
            ai22 = ai22 + e1 * b6i + e2 * b7i + \
                  e3 * b8i;
            gr22 = gr22 + e1 * c6r + e2 * c7r + \
                  e3 * c8r;
            gi22 = gi22 + e1 * c6i + e2 * c7i + \
                  e3 * c8i;
            }
      }

an12 = ann[n1-1][n2-1] * factor;
r11[n1-1][n2-1] = ar11 * an12;
r12[n1-1][n2-1] = ar12 * an12;
r21[n1-1][n2-1] = ar21 * an12;
r22[n1-1][n2-1] = ar22 * an12;
i11[n1-1][n2-1] = ai11 * an12,
i12[n1-1][n2-1] = ai12 * an12;
i21[n1-1][n2-1] = ai21 * an12;
i22[n1-1][n2-1] = ai22 * an12;
rg11[n1-1][n2-1] = gr11 * an12;
rg12[n1-1][n2-1] = gr12 * an12;
rg21[n1-1][n2-1] = gr21 * an12;
rg22[n1-1][n2-1] = gr22 * an12;
ig11[n1-1][n2-1] = gi11 * an12;
ig12[n1-1][n2-1] = gi12 * an12;
```

```
            ig21[n1-1][n2-1] = gi21 * an12;
            ig22[n1-1][n2-1] = gi22 * an12;
            }
      }
/* Forming Q and RgQ matrices */

tpir = pir;
tpii = pii;
tppi = ppi;
nm = nmax - mm1 + 1,
for (n1 = mm1, n1 <=nmax, ++n1)
      {
      k1 = n1 - mm1 + 1,
      kk1 = k1 + nm;
      for (n2 = mm1; n2 <= nmax; ++n2)
            {
            k2 = n2 - mm1 + 1;
            kk2 = k2 + nm;

            tarl1 = -rl1[n1-1][n2-1];
            tail1 = -il1[n1-1][n2-1];
            tgrl1 = -rgl1[n1-1][n2-1];
            tgil1 = -igl1[n1-1][n2-1];

            tarl2 = il2[n1-1][n2-1],
            tail2 = -rl2[n1-1][n2-1];
            tgrl2 = igl2[n1-1][n2-1];
            tgil2 = -rgl2[n1-1][n2-1];

            tar21 = -i21[n1-1][n2-1];
            tai21 = r21[n1-1][n2-1];
            tgr21 = -ig21[n1-1][n2-1];
            tgi21 = rg21[n1-1][n2-1];

            tar22 = -r22[n1-1][n2-1];
            tai22 = -i22[n1-1][n2-1],
            tgr22 = -rg22[n1-1][n2-1];
            tgi22 = -ig22[n1-1][n2-1];

            tqr[k1-1][k2-1] = tpir * tar21 - tpii * tai21 + \
                              tppi * tarl2;
            tqi[k1-1][k2-1] = tpir * tai21 + tpii * tar21 + \
                              tppi * tail2;
            trgqr[k1-1][k2-1] = tpir * tgr21 - tpii * tgi21 + \
                              tppi * tgrl2;
            trgqi[k1-1][k2-1] = tpir * tgi21 + tpii * tgr21 + \
                              tppi * tgil2;

            tqr[k1-1][kk2-1] = tpir * tarll - tpii * tail1 + \
                              tppi * tar22;
            tqi[k1-1][kk2-1] = tpir * tail1 + tpii * tarl1 + \
                              tppi * tai22;
            trgqr[k1-1][kk2-1] = tpir * tgrl1 - tpii * tgil1 + \
                              tppi * tgr22;
            trgqi[k1-1][kk2-1] = tpir * tgil1 + tpii * tgrl1 + \
                              tppi * tgi22;
```

```
            tqr[kk1-1][k2-1] = tpir * tar22 - tpii * tai22 + \
                               tppi * tar11;
            tqi[kk1-1][k2-1] = tpir * tai22 + tpii * tar22 + \
                               tppi * tai11;
            trgqr[kk1-1][k2-1] = tpir * tgr22 - tpii * tgi22 + \
                               tppi * tgr11;
            trgqi[kk1-1][k2-1] = tpir * tgi22 + tpii * tgr22 + \
                               tppi * tgi11;

            tqr[kk1-1][kk2-1] = tpir * tar12 - tpii * tai12 + \
                               tppi * tar21;
            tqi[kk1-1][kk2-1] = tpir * tai12 + tpii * tar12 + \
                               tppi * tai21;
            trgqr[kk1-1][kk2-1] = tpir * tgr12 - tpii * tgi12 + \
                               tppi * tgr21;
            trgqi[kk1-1][kk2-1] = tpir * tgi12 + tpii * tgr12 + \
                               tppi * tgi21;
            }
      }
nnmax = 2*nm;
for (n1 = 1; n1 <= nnmax; ++n1)
      {
      for (n2 = 1; n2 <= nnmax; ++n2)
            {
            qr[n1-1][n2-1] = tqr[n1-1][n2-1],
            qi[n1-1][n2-1] = tqi[n1-1][n2-1];
            rgqr[n1-1][n2-1] = trgqr[n1-1][n2-1];
            rgqi[n1-1][n2-1] = trgqi[n1-1][n2-1];
            }
      }

/* Forming T = -Q^-1 x RgQ matrix */

tt(nm, ncheck);

return 0;
}


//*****************************************************************//
//************************** VIG function *************************//
//*****************************************************************//
// Calaulates the Wigner d-functions.                             //
// Input parameters: x, nmax, m                                   //
// Output parameters: dv1, dv2                                    //
//*****************************************************************//

int vig(double *x, int nmax, int m,double *dv1, double *dv2)
{
/* Local variables */

int i, n, i2;
double a, d1, d2, d3, qn, qs, qn1, qn2, qs1, der, qmm, qnm, qnm1;

/* Function Body */
```

```
a = 1.0;
qs = sqrt(1.0 - *x * *x);
qs1 = 1.0 / qs;
for (n = 1; n <= nmax; ++n)
      {
      dv1[n-1] = 0.0;
      dv2[n-1] = 0.0;
      }
if (m == 0)
      {
      d1 = 1.0;
      d2 = *x;
      for (n = 1; n <= nmax; ++n)
            {
            qn = (double) n;
            qn1 = (double) (n + 1);
            qn2 = (double) (2*n + 1);
            d3 = (qn2 * *x * d2 - qn * d1) / qn1;
            der = qs1 * (qn1 * qn / qn2) * (-d1 + d3);
            dv1[n-1] = d2;
            dv2[n-1] = der;
            d1 = d2;
            d2 = d3;
            }
      return 0;
      }

qmm = (double) (m * m);
for (i = 1; i <= m; ++i)
      {
      i2 = 2*i;
      a = a * sqrt((double) (i2 - 1) / (double) i2) * qs;
      }
d1 = 0.0;
d2 = a;
for (n = m; n <= nmax; ++n)
      {
      qn = (double) n;
      qn2 = (double) (2*n + 1);
      qn1 = (double) (n + 1);
      qnm = sqrt(qn * qn - qmm);
      qnm1 = sqrt(qn1 * qn1 - qmm);
      d3 = (qn2 * *x * d2 - qnm * d1) / qnm1;
      der = qs1 * (-qn1 * qnm * d1 + qn * qnm1 * d3) / qn2;
      dv1[n-1] = d2;
      dv2[n-1] = der;
      d1 = d2;
      d2 = d3;
      }

return 0;
}

//*****************************************************************//
//************************** TT function **************************//
//*****************************************************************//
```

```
// Calculation of T = -Q^-1 x RgQ matrix.                    //
//*************************************************************//

int tt(int nmax, int ncheck)
{
/* Local variables */

int i,l,n1, n2, ndim, *ipvt, nnmax;
double *b, cond,*work;
double **a,**f,**c,**d,**e;

/* Memory Allocation */

ipvt = (int *) calloc(npn2 , sizeof(int));
b = (double *) calloc(npn2 , sizeof(double));
work = (double *) calloc(npn2 , sizeof(double));

a = (double **) calloc(npn2 , sizeof(double *));
if(NULL == a){free(a); printf("Memory allocation failed while \
      allocating for a[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      a[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == a[l]){free(a[l]); printf("Memory allocation failed\
      while allocating for a[l][].\n"); exit(-1);}
      }

f = (double **) calloc(npn2 , sizeof(double *)),
if(NULL == f){free(f); printf("Memory allocation failed while \
      allocating for f[].\n"); exit(-1),}
for(l = 0; l < npn2; l++)
      {
      f[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == f[l]){free(f[l]); printf("Memory allocation failed\
      while allocating for f[l][].\n"); exit(-1);}
      }

c = (double **) calloc(npn2 , sizeof(double *));
if(NULL == c){free(c); printf("Memory allocation failed while \
      allocating for c[].\n"); exit(-1);}
for(l = 0; l < npn2, l++)
      {
      c[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == c[l]){free(c[l]); printf("Memory allocation failed\
      while allocating for c[l][].\n"); exit(-1);}
      }

d = (double **) calloc(npn2 , sizeof(double *));
if(NULL == d){free(d), printf("Memory allocation failed while \
      allocating for d[].\n"); exit(-1);}
for(l = 0; l < npn2; l++)
      {
      d[l] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == d[l]){free(d[l]); printf("Memory allocation failed\
      while allocating for d[l][].\n"); exit(-1);}
      }
```

```
e = (double **) calloc(npn2 , sizeof(double *));
if(NULL == e){free(e); printf("Memory allocation failed while \
      allocating for e[].\n"); exit(-1);}
for(i = 0; i < npn2; i++)
      {
      e[i] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == e[i]){free(e[i]), printf("Memory allocation failed\
      while allocating for e[i][].\n"), exit(-1);}
      }

tr1 = (double **) calloc(npn2 , sizeof(double *));
if(NULL == tr1){free(tr1); printf("Memory allocation failed while \
      allocating for tr1[].\n"); exit(-1);}
for(i = 0; i < npn2; i++)
      {
      tr1[i] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == tr1[i]){free(tr1[i]);printf("Memory allocation failed\
      while allocating for tr1[i][].\n"); exit(-1),}
      }

ti1 = (double **) calloc(npn2 , sizeof(double *));
if(NULL == ti1){free(ti1); printf("Memory allocatilon failed while \
      allocating for ti1[].\n"); exit(-1);}
for(i = 0; i < npn2; i++)
      {
      ti1[i] = (double *) calloc(npn2 , sizeof(double));
      if(NULL == ti1[i]){free(ti1[i]); printf("Memory allocatilon\
      failed while allocating for ti1[i][].\n"); exit(-1);}
      }

/* Function Body */

ndim = 200,
nnmax = 2*(nmax);

/* Gaussian elimination */

for (n1 = 1; n1 <= nnmax; ++n1)
      {
      for (n2 = 1; n2 <= nnmax; ++n2)
            {
            f[n1-1][n2-1] = qi[n1-1][n2-1];
            }
    }
if (ncheck == 1)
      {
      inv1(nmax, f, a);
      }
else
      {
      invert(ndim, nnmax, f, a, &cond, ipvt, work, b);
      }
prod(qr, a, c, ndim, nnmax);
prod(c, qr, d, ndim, nnmax);
for (n1 = 1; n1 <= nnmax; ++n1)
      {
```

```
      for (n2 = 1; n2 <= nnmax; ++n2)
            {
            c[n1-1][n2-1] = d[n1-1][n2-1] + qi[n1-1][n2-1];
            }
      }
if (ncheck == 1)
      {
      invl(nmax, c, qi);
      }
else
      {
      invert(ndim, nnmax, c, qi, &cond, ipvt, work, b);
      }
prod(a, qr, d, ndim, nnmax);
prod(d, qi, qr, ndim, nnmax);
prod(rgqr, qr, a, ndim, nnmax);
prod(rgqi, qi, c, ndim, nnmax);
prod(rgqr, qi, d, ndim, nnmax);
prod(rgqi, qr, e, ndim, nnmax);
for (n1 = 1; n1 <= nnmax; ++n1)
      {
      for (n2 = 1; n2 <= nnmax; ++n2)
            {
            tr1[n1-1][n2-1] = -a[n1-1][n2-1] - c[n1-1][n2-1];
            ti1[n1-1][n2-1] = d[n1-1][n2-1] - e[n1-1][n2-1];
            }
      }

for(i=0;i<npn2;i++)
      {
      free(a[i]);
      }
free(a);
for(i=0;i<npn2;i++)
      {
      free(f[i]);
      }
free(f);
for(i=0;i<npn2;i++)
      {
      free(c[i]);
      }
free(c);
for(i=0;i<npn2;i++)
      {
      free(d[i]);
      }
free(d);
for(i=0;i<npn2;i++)
      {
      free(e[i]);
      }
free(e);

return 0;
}
```

```
//*********************************************************************//
//*************************** PROD function ***************************//
//*********************************************************************//
// Calculation of the n×n matrix c = a * b                            //
//*********************************************************************//

int prod(double **ax, double **bx, double **c,int ndim, int n)
{
/* Local variables */

int i, j, k, l;
double cij;

/* Function Body */
for (i = 1; i <= n; ++i)
      {
      for (j = 1; j <= n; ++j)
            {
            cij = 0.;
            for (k = 1; k <= n; ++k)
                  {
                  cij = cij + ax[i-1][k-1] * bx[k-1][j-1];
                  }
            c[i-1][j-1] = cij;
            }
      }

return 0;
}



//*********************************************************************//
//*************************** INV1 function ***************************//
//*********************************************************************//

int inv1(int nmax, double **f, double **a)
{
/* Local variables */

int i, l,j, i1, i2, j1, j2, nn1, nn2, *ind1, *ind2, ndim, *ipvt, nnmax;
double *b,cond,*work;
double **p1, **q1, **q2, **p2;

/* Memory Allocation */

ipvt = (int *) calloc(npn1 , sizeof(int));
ind1 = (int *) calloc(npn1 , sizeof(int));
ind2 = (int *) calloc(npn1 , sizeof(int));
b = (double *) calloc(npn1 , sizeof(double));
work = (double *) calloc(npn1 , sizeof(double));

p1 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == p1){free(p1); printf("Memory allocation failed while \
      allocating for p1[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
```

```
      p1[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == p1[l]){free(p1[l]); printf("Memory allocation failed\
      while allocating for p1[l][].\n"); exit(-1);}
      }

q1 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == q1){free(q1); printf("Memory allocation failed while \
      allocating for q1[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      q1[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == q1[l]){free(q1[l]); printf("Memory allocation failed\
      while allocating for q1[l][].\n"); exit(-1);}
      }

q2 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == q2){free(q2); printf("Memory allocation failed while \
      allocating for q2[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      q2[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == q2[l]){free(q2[l]); printf("Memory allocation failed\
      while allocating for p1[l][].\n"); exit(-1);}
      }

p2 = (double **) calloc(npn1 , sizeof(double *));
if(NULL == p2){free(p2); printf("Memory allocation failed while \
      allocating for p2[].\n"); exit(-1);}
for(l = 0; l < npn1; l++)
      {
      p2[l] = (double *) calloc(npn1 , sizeof(double));
      if(NULL == p2[l]){free(p2[l]); printf("Memory allocation failed\
      while allocating for q1[l][].\n"), exit(-1);}
      }

/* Function Body */

ndim = npn1;
nn1 = (int) (((double) (nmax) - 0.1) * 0.5 + 1.0);
nn2 = nmax - nn1;

for (i = 1; i <= nmax; ++i)
      {
      ind1[i - 1] = 2*i - 1;
      if (i > nn1)
            {
            ind1[i - 1] = nmax + 2*(i - nn1);
            }
      ind2[i - 1] = 2*i;
      if (i > nn2)
            {
            ind2[i - 1] = nmax + 2*(i - nn2)-1;
            }
      }
nnmax = 2*(nmax),
for (i = 1; i <= nmax, ++i)
      {
```

```
      i1 = ind1[i - 1];
      i2 = ind2[i - 1];

      for (j = 1; j <= nmax; ++j)
            {
            j1 = ind1[j - 1];
            j2 = ind2[j - 1];
            q1[j-1][i-1] = f[j1-1][i1-1];
            q2[j-1][i-1] = f[j2-1][i2-1];
            }
      }
invert(ndim, nmax, q1, p1, &cond, ipvt, work, b);
invert(ndim, nmax, q2, p2, &cond, ipvt, work, b);

for (i = 1; i <= nnmax; ++i)
      {
      for (j = 1; j <= nnmax; ++j)
            {
            a[j-1][i-1] = 0.0;
            }
      }
for (i = 1; i <= nmax; ++i)
      {
      i1 = ind1[i - 1];
      i2 = ind2[i - 1];
      for (j = 1; j <= nmax; ++j)
            {
            j1 = ind1[j - 1];
            j2 = ind2[j - 1];
            a[j1-1][i1-1] = p1[j-1][i-1];
            a[j2-1][i2-1] = p2[j-1][i-1],
            }
      }

for(i=0;i<npn1;i++)
      {
      free(p1[i]);
      }
free(p1);
for(i=0;i<npn1;i++)
      {
      free(p2[i]);
      }
free(p2);
for(i=0;i<npn1;i++)
      {
      free(q1[i]);
      }
free(q1);
for(i=0;i<npn1;i++)
      {
      free(q2[i]);
      }
free(q2);

return 0;
}
```

```
//*********************************************************************//
//************************** INVERT function ***************************//
//*********************************************************************//
// This function performs the inversion of a square matrix.           //
// Input parameters:                                                  //
//                a = n×n square matrix                               //
//                ndim = declared line dimension of the matrix a      //
// Output unfoamation:                                                //
//                x = inverse of the n×n square matrix a              //
//                cond = estimate of the ill-conditioning of the      //
//                      matrix a                                      //
//*********************************************************************//

int invert(int ndim, int n, double **a,double **xa, double *cond, \
      int *ipvt, double *work, double *b)
{

/* Local Variable */

int i, j, k, l,m,jo, kb, km1, nm1, kp1;

double t, ek, anorm, ynorm, znorm;

/* Function Body */

ipvt[n-1] = 1;
if (n != 1)
      {
      nm1 = n - 1;
      anorm = 0.0;
      for (j = 1; j <= n; ++j)
            {
            t = 0.0;
            for (i = 1; i <= n; ++i)
                  {
                  t = t+ fabs(a[i-1][j-1]);
                  }
            if (t > anorm)
                  {
                  anorm = t;
                  }
            }

      for (k = 1; k <= nm1; ++k)
            {
            kp1 = k + 1;
            m = k;
            jo=k;
            for (i = kp1; ı <= n; ++i)
                  {
                  if (fabs(a[i-1][k-1]) > fabs(a[m-1][k-1]))
                        {
                        m = i;
                        }
                  if(fabs(a[i-1][k-1]) < fabs(a[m-1][k-1]))
                        {
                        m=m;
```

```
                }
            }
        ipvt[k-1] = m;
        if (m != k)
            {
            ipvt[n-1] = -ipvt[n-1];
            }
        t = a[m-1][k-1];
        a[m-1][k-1] = a[k-1][k-1];
        a[k-1][k-1] = t;

        if (t != 0.)
            {
            for (i = kp1; i <= n; ++i)
                {
                a[i-1][k-1] = -a[i-1][k-1]/ t;
                }
            for (j = kp1; j <= n; ++j)
                {
                t = a[m-1][j-1];
                a[m-1][j-1] = a[k-1][j-1];
                a[k-1][j-1] = t;
                if (t != 0.0)
                    {
                    for (i= kp1; i <=n; ++i)
                        {
                        a[i-1][j-1] = a[i-1][j-1]+ \
                                      a[i-1][k-1] * t;
                        }
                    }
                }
            }
        }
for (k = 1; k <= n; ++k)
        {
        t = 0.0;
        if (k != 1)
            {
            km1 = k - 1;
            for (i = 1; i <= km1; ++i)
                {
                t = t + a[i-1][k-1] * work[i-1];
                }
            }
        ek = 1.0;
        if (t < 0.0)
            {
            ek = -1.0;
            }
        if (a[k-1][k-1]== 0.0)
            {
            *cond = 1e+52;
            if (*cond + 1.0 == *cond)
                {
                printf("THE MATRIX IS SINGULAR FOR THE \
                    GIVEN NUMERICAL ACCURACY COND = %e \
                    \n",*cond);
```

```
                }
            for (i= 1; i<= n; ++i)
                {
                for (j = 1; j <= n; ++j)
                    {
                    b[j-1] = 0.0;
                    if (j == i)
                        {
                        b[j-1] = 1.0;
                        }
                    }
                solve(ndim, n, a, b, ipvt);

                for (j = 1; j <= n; ++j)
                    {
                    xa[j-1][i-1] = b[j-1];
                    }
                }
            return 0;
            }
        work[k-1] = -(ek + t) / a[k-1][k-1];
        }
    for (kb = 1; kb <= nm1; ++kb)
        {
        k = n - kb;
        t = 0.0;
        kp1 = k + 1;
        for (i = kp1; i <= n; ++i)
            {
            t = t + a[i-1][k-1]* work[k-1];
            }
        work[k-1] = t;
        m = ipvt[k-1];
        if (m != k)
            {
            t = work[m-1];
            work[m-1] = work[k-1];
            work[k-1] = t;
            }
        }
    ynorm = 0.0;
    for (i = 1; i <= n; ++i)
        {
        ynorm = ynorm+(fabs(work[i-1]));
        }

    solve(ndim, n, a, work, ipvt);

    znorm = 0.0;
    for (i = 1; i <= n; ++i)
        {
        znorm = znorm+(fabs(work[i-1]));
        }

    *cond = anorm * znorm / ynorm;

    if (*cond < 1.0)
```

```
            {
            *cond = 1.0;
            }
      }

goto L110;

*cond = 1.0;
if (a[0][0] == 0.0)
      {
      *cond = 1e+52;
      }

L110:
if (*cond + 1.0 == *cond)
      {
      printf("THE MATRIX IS SINGULAR FOR THE GIVEN NUMERICAL \
            ACCURACY COND = %e \n",*cond);
      }
/*     IF (COND+1D0.EQ.COND) STOP */

for (i= 1; i<= n; ++i)
      {
      for (j = 1; j <= n; ++j)
            {
            b[j-1] = 0.0;
            if (j == i)
                  {
                  b[j-1] = 1.0;
                  }
            }
      solve(ndim, n, a, b, ipvt);

      for (j = 1; j <= n; ++j)
            {
            xa[j-1][i-1] = b[j-1];
            }
      }
return 0;
}

//*****************************************************************//
//************************* SOLVE function ************************//
//*****************************************************************//

int solve(int ndim, int n, double **a,double *b, int *ipvt)
{

/* Local variables */

int i, k, m, kb, km1, nm1, kp1;
double t;

/* Function Body */
if (n != 1)
      {
      nm1 = n - 1;
```

```
      for (k = 1; k <= nm1; ++k)
            {
            kp1 = k + 1;
            m = ipvt[k-1];
            t = b[m-1];
            b[m-1] = b[k-1];
            b[k-1] = t;
            for (i = kp1; i <= n; ++i)
                  {
                  b[i-1] = b[i-1] + a[i-1][k-1] * t;
                  }
            }
      for (kb = 1; kb <= nm1; ++kb)
            {
            km1 = n - kb;
            k = km1 + 1;
            b[k-1] = b[k-1] / a[k-1][k-1];
            t = -b[k-1];
            for (i = 1; i <= km1; ++i)
                  {
                  b[i-1] = b[i-1] + a[i-1][k-1] * t;
                  }
            }
      }
b[0] = b[0] / a[0][0];

return 0;
}

//*********************************************************************//
//************************** GSP function *****************************//
//*********************************************************************//
// Calculates the expansion coefficients for the (I, Q, U, V)        //
// representation.                                                   //
// Input parameters:                                                 //
//                 lam = wavelength of light                         //
//                 csca = scattering cross section                   //
//                 tr and ti elements of the T-matrix                //
//                 nmax = dimension of T-matrix
// Output information:                                               //
//                 the expansion coefficients - alf1, alf2, alf3,    //
//                                              alf4, bet1, bet2     //
//                 lmax = number of coefficients minus 1             //
//*********************************************************************//

int gsp(int nmax, double csca, double lam, double *alf1, double \
      *alf2, double *alf3, double *alf4, double *bet1, double *bet2,\
      int *lmax)
{
/* Local variables */

int i, j, l, m, n,i1, k1, k2, k3, k4, k5, k6, m1, l1, n1, m2, kn,\
      nl, nn, nn1, nnn, mmin, mmax, m1min, l1max, nmax1, m1max,\
      nnmin, nnmax, nn1min, nn1max;

double  t1, t2, t3, t4, x1, x2, x3, x4, x5, x6, x7, x8, **ff,si, \
      sj,sl, xi, xr, xx, dd1, **tr1,**tr2,dd2, *ai1, *ai2, dd3, \
```

```
      dd4,***d5i,dm1, *ar1, *ar2, g1l, g2l, g3l,**ti1, **ti2, ri2,\
      g4l, dm2, dm3, dm4, rr1, rr2, tt1, tt2,ffn, sig,tt3, tt4, \
      tt5, tt6, tt7, tt8,*ssi, *ssj, sss, aai1, aai2, bbi1, bbi2, \
      aar1, aar2, bbr1, bbr2, dd5i, dd5r, dm5i, g5li, dm5r, g5lr,\
      sss1,**fi, dk, **fr, ri1,***d1,***d2,***d3,***d4,***d5r;

struct complex ci,cci, ccj, cim[100],c1;

double **g1,**g2,***m1r,***m1i,***m2r,***m2i;

/* Memory Allocation */

ai1 = (double *) calloc(npn4 , sizeof(double));
ai2 = (double *) calloc(npn4 , sizeof(double));
ar1 = (double *) calloc(npn4 , sizeof(double));
ar2 = (double *) calloc(npn4 , sizeof(double));
ssi = (double *) calloc(npl , sizeof(double));
ssj = (double *) calloc(npn1 , sizeof(double));


ff = (double **) calloc(npn4 , sizeof(double *));
if(NULL == ff){free(ff); printf("Memory allocation failed while \
      allocating for ff[].\n"); exit(-1);}
for(l = 0; l < npn4; l++)
      {
      ff[l] = (double *) calloc(npn4 , sizeof(double));
      if(NULL == ff[l]){free(ff[l]); printf("Memory allocation failed\
      while allocating for ff[l][].\n"); exit(-1);}
      }

fr = (double **) calloc(npn4 , sizeof(double *));
if(NULL == fr){free(fr); printf("Memory allocation failed while \
      allocating for fr[].\n"); exit(-1);}
for(l = 0; l < npn4; l++)
      {
      fr[l] = (double *) calloc(npn4 , sizeof(double));
      if(NULL == fr[l]){free(fr[l]); printf("Memory allocation failed\
      while allocating for fr[l][].\n"); exit(-1);}
      }

fi = (double **) calloc(npn4 , sizeof(double *));
if(NULL == fi){free(fi); printf("Memory allocation failed while \
      allocating for fi[].\n"); exit(-1);}
for(l = 0; l < npn4; l++)
      {
      fi[l] = (double *) calloc(npn4 , sizeof(double));
      if(NULL == fi[l]){free(fi[l]); printf("Memory allocation failed\
      while allocating for fi[l][].\n"); exit(-1);}
      }

ti1 = (double **) calloc(npl1 , sizeof(double *));
if(NULL == ti1){free(ti1); printf("Memory allocation failed while \
      allocating for ti1[].\n"), exit(-1);}
for(l = 0; l < npl1; l++)
      {
      ti1[l] = (double *) calloc(npn4 , sizeof(double));
      if(NULL == ti1[l]){free(ti1[l]); printf("Memory allocation \
```

```
      failed while allocating for ti1[l][].\n"); exit(-1);}
      }

ti2 = (double **) calloc(npl1 , sizeof(double *));
if(NULL == ti2){free(ti2); printf("Memory allocation failed while \
      allocating for ti2[].\n"); exit(-1);}
for(l = 0; l < npl1; l++)
      {
      ti2[l] = (double *) calloc(npn4 , sizeof(double));
      if(NULL == ti2[l]){free(ti1[l]); printf("Memory allocation \
      failed while allocating for ti2[l][].\n"); exit(-1);}
      }

tr1 = (double **) calloc(npl1 , sizeof(double *));
if(NULL == tr1){free(tr1); printf("Memory allocation failed while \
      allocating for tr1[].\n"); exit(-1);}
for(l = 0; l < npl1; l++)
      {
      tr1[l] = (double *) calloc(npn4 , sizeof(double));
      if(NULL == tr1[l]){free(tr1[l]); printf("Memory allocation \
      failed while allocating for tr1[l][].\n"); exit(-1);}
      }

tr2 = (double **) calloc(npl1 , sizeof(double *));
if(NULL == tr2){free(tr2); printf("Memory allocation failed while \
      allocating for tr2[].\n"); exit(-1);}
for(l = 0; l < npl1; l++)
      {
      tr2[l] = (double *) calloc(npn4 , sizeof(double));
      if(NULL == tr2[l]){free(tr2[l]); printf("Memory allocation \
      failed while allocating for tr2[l][].\n"); exit(-1);}
      }

d1 = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      d1[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            d1[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

d2 = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      d2[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            d2[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

d3 = (double ***)calloc(npl1,sizeof(double**));
for (m = 0, m< npl1; m++)
      {
      d3[m] = (double **) calloc(npn4,sizeof(double *));
```

```
      for (n = 0; n < npn4; n++)
            {
            d3[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

d4 = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      d4[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            d4[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }
d5i = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      d5i[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            d5i[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

d5r = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      d5r[m] = (double **) calloc(npn4,sizeof(double *));
      for (n = 0; n < npn4; n++)
            {
            d5r[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

g1 = (double **) calloc(npl1 , sizeof(double *));
if(NULL == g1){free(g1); printf("Memory allocation failed while \
      allocating for g1[].\n"); exit(-1);}
for(l = 0; l < npl1; l++)
      {
      g1[l] = (double *) calloc(npn6 , sizeof(double));
      if(NULL == g1[l]){free(g1[l]); printf("Memory allocation \
      failed while allocating for g1[l][].\n"); exit(-1);}
      }

g2 = (double **) calloc(npl1 , sizeof(double *));
if(NULL == g2){free(g2); printf("Memory allocation failed while \
      allocating for g2[].\n"); exit(-1);}
for(l = 0; l < npl1; l++)
      {
      g2[l] = (double *) calloc(npn6 , sizeof(double));
      if(NULL == g2[l]){free(g2[l]); printf("Memory allocation \
      failed while allocating for g2[l][].\n"); exit(-1);}
      }


m1r = (double ***)calloc(npl1,sizeof(double**));
```

```
for (m = 0; m< npl1; m++)
      {
      m1r[m] = (double **) calloc(npl1,sizeof(double *));
      for (n = 0; n < npl1; n++)
            {
            m1r[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

m2r = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      m2r[m] = (double **) calloc(npl1,sizeof(double *));
      for ( n= 0; n < npl1; n++)
            {
            m2r[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

m1i = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      m1i[m] = (double **) calloc(npl1,sizeof(double *));
      for (n = 0; n < npl1; n++)
            {
            m1i[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

m2i = (double ***)calloc(npl1,sizeof(double**));
for (m = 0; m< npl1; m++)
      {
      m2i[m] = (double **) calloc(npl1,sizeof(double *));
      for (n = 0; n < npl1; n++)
            {
            m2i[m][n] = (double *)calloc(npn4,sizeof(double));
            }
      }

/* Function Body */
signum();
*lmax = 2*(nmax);
llmax = *lmax + 1;
ci.x = 0.0, ci.y = 1.0;
cim[0].x = ci.x;
cim[0].y = ci.y;
for (i = 2; i <= nmax; ++i)
      {
      cim[i-1] = cpxmult(cim[i-2],ci);
      }
ssi[0] = 1.0;
for (i = 1; i<= *lmax; ++i)
      {
      i1 = i + 1;
      si = (double) (2*i + 1);
      ssi[il - 1] = si;
      if (i <= nmax)
```

```
            {
            ssj[i - 1] = pow(si,0.5);
            }
      }

ci.x=-ci.x;
ci.y=-ci.y;
for (i = 1; i<= nmax; ++i)
      {
      si = ssj[i - 1];
      cci.x = cim[i-1].x;
      cci.y = cim[i-1].y;
      for (j = 1; j <= nmax; ++j)
            {
            sj = 1.0 / ssj[j - 1];
            c1.x=sj*cim[j-1].x;
            c1.y=sj*cim[j-1].y;
            ccj=cpxdiv(c1,cci);
            fr[j-1][i-1]=ccj.x;
            fi[j-1][i-1]=cpxmult(ccj,ci).x;
            ff[j-1][i-1] = si * sj;
            }
      }

nmax1 = nmax + 1;

/******  Calculation of the arrays b1 and b2  ******/

k1 = 1;
k2 = 0;
k3 = 0;
k4 = 1;
k5 = 1;
k6 = 2;

for (n = 1; n <= nmax; ++n)
      {
      /* Calculation of the arrays t1 and t2  */
      for (nn = 1; nn <= nmax; ++nn)
            {
            m1max = min(n,nn) + 1;
            for (m1 = 1; m1 <= m1max; ++m1)
                  {
                  m = m1 - 1;
                  l1 = npn6+m;
                  tt1 = rt11[m1-1][n-1][nn-1];
                  tt2 = rt12[m1-1][n-1][nn-1];
                  tt3 = rt21[m1-1][n-1][nn-1];
                  tt4 = rt22[m1-1][n-1][nn-1];
                  tt5 = it11[m1-1][n-1][nn-1];
                  tt6 = it12[m1-1][n-1][nn-1];
                  tt7 = it21[m1-1][n-1][nn-1];
                  tt8 = it22[m1-1][n-1][nn-1];
                  t1 = tt1 + tt2;
                  t2 = tt3 + tt4;
                  t3 = tt5 + tt6;
                  t4 = tt7 + tt8;
```

```
            tr1[l1-1][nn-1] = t1 + t2;
            tr2[l1-1][nn-1] = t1 - t2;
            ti1[l1-1][nn-1] = t3 + t4;
            ti2[l1-1][nn-1] = t3 - t4;
            if (m != 0)
                  {
                  l1 = 81 - m;
                  t1 = tt1 - tt2;
                  t2 = tt3 - tt4;
                  t3 = tt5 - tt6;
                  t4 = tt7 - tt8;
                  tr1[l1-1][nn-1] = t1 - t2;
                  tr2[l1-1][nn-1] = t1 + t2;
                  ti1[l1-1][nn-1] = t3 - t4;
                  ti2[l1-1][nn-1] = t3 + t4;
                  }


            }
      }

/* End of calculation of the arrays t1 and t2  */
nn1max = nmax1 + n;
for (nn1 = 1; nn1 <= nn1max; ++nn1)
      {
      n1 = nn1 - 1;

      /* Calculation of the arrays a1 and a2  */

      ccg(n, n1, nmax, k1, k2, g1);
      nnmax = min(nmax,n1 + n);
      nnmin = max(1,abs(n - n1));
      kn = n + nn1;

      for (nn = nnmin; nn <= nnmax; ++nn)
            {
            nnn = nn + 1;
            sig = ssign[kn + nn - 1];
            m1max = npn6+min(n,nn);
            aar1 = 0.0;
            aar2 = 0.0;
            aai1 = 0.0;
            aai2 = 0.0;


            for (m1 = npn6; m1 <= m1max; ++m1)
                  {
                  m = -(npn6-m1);
                  sss = g1[m1-1][nnn-1];
                  rr1 = tr1[m1-1][nn-1];
                  ri1 = ti1[m1-1][nn-1];
                  rr2 = tr2[m1-1][nn-1];
                  ri2 = ti2[m1-1][nn-1];
                  if (m != 0)
                        {
                        m2 = npn6 - m;
                        rr1 = rr1 + tr1[m2-1][nn-1] * sig;
```

```
                        ri1 = ri1 + ti1[m2-1][nn-1] * sig;
                        rr2 = rr2 + tr2[m2-1][nn-1] * sig;
                        ri2 = ri2 + ti2[m2-1][nn-1] * sig;
                        }

                    aar1 += sss * rr1;
                    aai1 += sss * ri1;
                    aar2 += sss * rr2;
                    aai2 += sss * ri2;
                    }
                xr = fr[nn-1][n-1];
                xi = fi[nn-1][n-1];
                ar1[nn - 1] = aar1 * xr - aai1 * xi;
                ai1[nn - 1] = aar1 * xi + aai1 * xr;
                ar2[nn - 1] = aar2 * xr - aai2 * xi;
                ai2[nn - 1] = aar2 * xi + aai2 * xr;
                }

        /* End of calculation of the arrays a1 and a2  */

        ccg(n, n1, nmax, k3, k4, g2);
        m1 = max(-n1 + 1,-n);
        m2 = min(n1 + 1,n);
        m1max = npn6+m2;
        m1min = npn6+m1;
        for (m1 = m1min; m1 <= m1max; ++m1)                {
              bbr1 = 0.0;
              bbi1 = 0.0;
              bbr2 = 0.0;
              bbi2 = 0.0;
              for (nn = nnmin; nn <= nnmax; ++nn)
                    {
                    nnn = nn + 1;
                    sss = g2[m1-1][nnn-1];
                    bbr1 = bbr1+sss * ar1[nn - 1];
                    bbi1 = bbi1+sss * ai1[nn - 1];
                    bbr2 = bbr2+sss * ar2[nn - 1];
                    bbi2 = bbi2+sss * ai2[nn - 1];
                    }
              m1r[nn1-1][m1-1][n-1] = bbr1;
              m1i[nn1-1][m1-1][n-1] = bbi1;
              m2r[nn1-1][m1-1][n-1] = bbr2;
              m2i[nn1-1][m1-1][n-1] = bbi2;
              }
        }
  }

/******  End of calculation of the arrays b1 and b2  ******/

/******  Calculation of the arrays d1, d2, d3, d4 and d5  ******/

for (n = 1; n <= nmax; ++n)
  {
  for (nn = 1; nn <= nmax; ++nn)
        {
        m1 = min(n,nn);
        m1max = npn6+m1;
```

```
m1min = npn6 - m1;
nn1max = nmax1 + min(n,nn);
for (m1 = m1min; m1 <= m1max; ++m1)
      {
      m = -(npn6-m1);
      nn1min = (abs(m - 1)) + 1;
      dd1 = 0.0;
      dd2 = 0.0;
      for (nn1 = nn1min; nn1 <= nn1max; ++nn1)
            {
            xx = ssi[nn1 - 1];
            x1 = m1r[nn1-1][m1-1][n-1];
            x2 = m1i[nn1-1][m1-1][n-1];
            x3 = m1r[nn1-1][m1-1][nn-1];
            x4 = m1i[nn1-1][m1-1][nn-1];
            x5 = m2r[nn1-1][m1-1][n-1];
            x6 = m2i[nn1-1][m1-1][n-1];
            x7 = m2r[nn1-1][m1-1][nn-1];
            x8 = m2i[nn1-1][m1-1][nn-1];
            dd1 = dd1+xx * (x1 * x3 + x2 * x4);
            dd2 = dd2+xx * (x5 * x7 + x6 * x8);
            }
      d1[m1-1][nn-1][n-1] = dd1;
      d2[m1-1][nn-1][n-1] = dd2;
      }

mmax = min(n,nn + 2);
mmin = max(-n,-nn + 2);
m1max = npn6+mmax;
m1min = npn6+mmin;
for (m1 = m1min; m1 <= m1max; ++m1)
      {
      m = -(npn6-m1);
      nn1min = (abs(m - 1)) + 1;
      dd3 = 0.0;
      dd4 = 0.0;
      dd5r = 0.0;
      dd5i = 0.0;
      m2 = npn6 - m + 2;
      for (nn1 = nn1min; nn1 <= nn1max; ++nn1)
            {
            xx = ssi[nn1 - 1];
            x1 = m1r[nn1-1][m1-1][n-1];
            x2 = m1i[nn1-1][m1-1][n-1];
            x3 = m2r[nn1-1][m1-1][n-1];
            x4 = m2i[nn1-1][m1-1][n-1];
            x5 = m1r[nn1-1][m2-1][nn-1];
            x6 = m1i[nn1-1][m2-1][nn-1];
            x7 = m2r[nn1-1][m2-1][nn-1];
            x8 = m2i[nn1-1][m2-1][nn-1];
            dd3 = dd3+xx * (x1 * x5 + x2 * x6);
            dd4 = dd4+xx * (x3 * x7 + x4 * x8);
            dd5r = dd5r+xx * (x3 * x5 + x4 * x6);
            dd5i = dd5i+xx * (x4 * x5 - x3 * x6);
            }
      d3[m1-1][nn-1][n-1] = dd3;
      d4[m1-1][nn-1][n-1] = dd4;
```

```
                d5r[m1-1][nn-1][n-1] = dd5r;
                d5i[m1-1][nn-1][n-1] = dd5i;
                }
            }
        }

/***** End of calculation of the arrays d1, d2, d3, d4 and d5 *****/

/****** Calculation of expansion coefficients ******/

dk = lam * lam / (csca * 4.0 * acos(-1.));
for (l1 = 1; l1 <= l1max; ++l1) //1
        {
        g1l = 0.0;
        g2l = 0.0;
        g3l = 0.0;
        g4l = 0.0;
        g5lr = 0.0;
        g5li = 0.0;
        l = l1 - 1;
        sl = ssi[l1 - 1] * dk;
        for (n = 1; n <= nmax; ++n)
            {
            nnmin = max(1,abs(n-l));
            nnmax = min(nmax,n + l);

            if (nnmax >= nnmin)
                {
                ccg(n, l, nmax, k1, k2, g1);
                if (l >= 2)
                    {
                    ccg(n, l, nmax, k5, k6, g2);
                    }
                nl = n + l;
                for (nn = nnmin; nn <= nnmax; ++nn)
                    {
                    nnn = nn + 1;
                    mmax = min(n,nn);
                    mlmin = npn6 - mmax;
                    mlmax = npn6+mmax;
                    si = ssign[nl + nnn - 1];
                    dm1 = 0.0;
                    dm2 = 0.0;
                    for (m1 = mlmin; m1 <= mlmax; ++m1)
                        {
                        m = -(npn6-m1);
                        if (m >= 0)
                            {
                            sss1 = g1[m1-1][nnn-1];
                            }
                        if (m < 0)
                            {
                            sss1 = g1[npn6-m-1][nnn-1] * si;
                            }
                        dm1 = dm1+sss1 * d1[m1-1][nn-1][n-1];
                        dm2 = dm2+sss1 * d2[m1-1][nn-1][n-1];
                        }
```

```
                        ffn = ff[nn-1][n-1];
                        sss = g1[npn6][nnn-1] * ffn;
                        g1l = g1l+sss * dm1;
                        g2l = g2l+sss * dm2 * si;
                        if (l >= 2)
                              {
                              dm3 = 0.;
                              dm4 = 0.;
                              dm5r = 0.;
                              dm5i = 0.;
                              mmax = min(n,nn+2);
                              mmin = max(-n,-nn+2);
                              m1max = npn6+mmax;
                              m1min = npn6+mmin;
                              for (m1 = m1min; m1 <= m1max; ++m1)
                                    {
                                    m = -(npn6-m1);
                                    sss1 = g2[npn6-m-1][nnn-1];
                                    dm3 = dm3+sss1 * d3[m1-1][nn-1]\
                                          [n-1];
                                    dm4 = dm4+sss1 * d4[m1-1][nn-1]\
                                          [n-1];
                                    dm5r = dm5r+sss1 * d5r[m1-1]\
                                          [nn-1][n-1];
                                    dm5i = dm5i+sss1 * d5i[m1-1]\
                                          [nn-1][n-1];
                                    }
                              g5lr = g5lr-sss * dm5r;
                              g5li = g5li-sss * dm5i;
                              sss = g2[npn4-1][nnn-1] * ffn;
                              g3l = g3l+sss * dm3;
                              g4l = g4l+sss * dm4 * si;
                              }
                        }
                  }
            }

      g1l = g1l*sl;
      g2l = g2l*sl;
      g3l = g3l*sl;
      g4l = g4l*sl;
      g5lr = g5lr*sl;
      g5li = g5li*sl;
      alf1[l1-1] = g1l + g2l;
      alf2[l1-1] = g3l + g4l;
      alf3[l1-1] = g3l - g4l;
      alf4[l1-1] = g1l - g2l;
      bet1[l1-1] = g5lr * 2.;
      bet2[l1-1] = g5li * 2.;
      *lmax = l;
      if (fabs(g1l) < 1e-6)
            {
            break;
            }
      }

for(m = 0; m < npl1; m++)
```

```
      for(n=0;n<npn4;n++)
            free(d1[m][n]);
free(d1);
for(m = 0; m < npl1; m++)
      for(n=0;n<npn4;n++)
            free(d2[m][n]);
free(d2);
for(m = 0; m < npl1; m++)
      for(n=0;n<npn4;n++)
            free(d3[m][n]);
free(d3);
for(m = 0; m < npl1; m++)
      for(n=0;n<npn4;n++)
            free(d4[m][n]);
free(d4);
for(m = 0; m < npl1; m++)
      for(n=0;n<npn4;n++)
            free(d5r[m][n]);
free(d5r);
for(m = 0; m < npl1; m++)
      for(n=0;n<npn4;n++)
            free(d5i[m][n]);
free(d5i);
for(m = 0; m < npl1; m++)
      for(n=0;n<npl1;n++)
            free(m1r[m][n]);
free(m1r);
for(m = 0; m < npl1; m++)
      for(n=0;n<npl1;n++)
            free(m1i[m][n]);
free(m1i);
for(m = 0; m < npl1; m++)
      for(n=0;n<npl1;n++)
            free(m2r[m][n]);
free(m2r);
for(m = 0; m < npl1; m++)
      for(n=0;n<npl1;n++)
            free(m2i[m][n]);
free(m2i);
for(m = 0; m < npl1; m++)
      free(g1[m]);
free(g1);
for(m = 0; m < npl1; m++)
      free(g2[m]);
free(g2);
for(m = 0; m < npn4; m++)
      free(ff[m]);
free(ff);
for(m = 0; m < npn4; m++)
      free(fi[m]);
free(fi);
for(m = 0; m < npn4; m++)
      free(fr[m]);
free(fr);
for(m = 0; m < npl1; m++)
      free(tr1[m]);
free(tr1);
```

```
for(m = 0; m < npl1; m++)
      free(tr2[m]);
free(tr2);
for(m = 0; m < npl1; m++)
      free(ti1[m]);
free(ti1);
for(m = 0; m < npl1; m++)
      free(ti2[m]);
free(ti2);

return 0;
}


//*********************************************************************//
//************************ SIGNUM function ************************//
//*********************************************************************//
// Calculation of the array ssign(n+1) = sign(n)                       //
//*********************************************************************//

int signum(void)
{
int n;

ssign[0] = 1.0;
for (n = 2; n <= 899; ++n)
      {
      ssign[n - 1] = -ssign[n - 2];
      }
return 0;
}


//*********************************************************************//
//************************** ccg function *************************//
//*********************************************************************//
// Calculation of Clebsch - Gordan coefficients.                       //
// Input parameters: n, n1, nmax, k1, k2                               //
// Output parameters: gg(m+npn6, nn+1)                                 //
//*********************************************************************//

int ccg(int n, int n1, int nmax, int k1, int k2, double **gg)
{
/* Local variables */

int l,m,m1,mf,mm, nn, nnf, min, nnl, nnm, nnu, mind,r,s;
double a, b, c, d, c1, c2,*cd, *cu;

/* Memory Allocation */

cd = (double *) calloc(npn5 , sizeof(double));
cu = (double *) calloc(npn5 , sizeof(double));
cd[-1]=0.0;
cu[-1]=0.0;

/* Function Body */
```

```
if (nmax <= npn4 && 0 <= n1 && n1 <= nmax + n && n >= 1 && n <= nmax)
      {
      nnf = min(n + n1,nmax);
      min = npn6 - n;
      mf = npn6+n;

      if (k1 == 1 && k2 == 0)
            {
            min = npn6;
            }

      m=0;

      for (mind = min; mind <= mf; ++mind)
            {
            m=-(npn6-mind);
            mm = m * k1 + k2;
            m1 = mm - m;


            if (abs(m1) <= n1)
                  {
                  nnl = max(abs(mm),abs(n - n1));
                  if (nnl <=nnf)
                        {
                        nnu = n + n1;
                        nnm = (int) ((nnu + nnl) * .5);
                        if (nnu == nnl)
                              {
                              nnm = nnl;
                              }
                        ccgin(n, n1, m, mm, &c);
                        cu[nnl-1] = c;
                        if (nnl != nnf)
                              {
                              c2 = 0.0;
                              c1 = c;
                              for (nn = nnl + 1; nn <= min(nnm,nnf);\
                                    ++nn)
                                    {
                                    a = (double) ((nn + mm) * (nn - \
                                          mm) * (n1 - n + nn));
                                    a = a*(double) ((n - n1 + nn) * \
                                          (n + n1 - nn + 1) * (n + n1 \
                                          + nn + 1));
                                    a = (double) (4* nn * nn) / a;
                                    a = a*(double) ((2*nn + 1) * (2*nn\
                                          - 1));
                                    a = sqrt(a);
                                    b = (double) (m - m1) * .5;
                                    d = 0.0;
                                    if (nn != 1)
                                          {
                                          b = (double) (2*nn * (nn -\
                                                1));
                                          b = (double) ((2*m - mm) * \
                                                nn * (nn - 1) - mm * n\
```

```
                        * (n + 1) + mm * n1 * \
                        (n1 + 1)) / b;
                    d = (double) (4*(nn - 1) * \
                        (nn - 1));
                    d = d*(double) ((2*nn - 3) *\
                        (2*nn - 1));
                    d = (double) ((nn - mm - 1)*\
                        (nn + mm - 1) * (n1 - \
                        n + nn - 1)) / d;
                    d = d*(double) ((n - n1 + nn\
                        - 1) * (n + n1 - nn + \
                        2) * (n + n1 + nn));
                    d = sqrt(d);
                    }
                c = a * (b * c1 - d * c2);
                c2 = c1;
                c1 = c;
                cu[nn-1] = c;
                }
if (nnf > nnm)
        {
        direct(n, m, n1, m1, nnu, mm, &c);
        cd[nnu-1] = c;
        if (nnu != nnm + 1)
        {
        c2 = 0.;
        c1 = c;
        for (nn = nnu - 1; nn >= nnm + 1; --nn)
        {
        a = (double) ((nn - mm + 1) * (nn + mm \
            + 1) * (n1 - n + nn + 1));
        a = a*(double) ((n - n1 + nn + 1) *(n +\
            n1 - nn) * (n + n1 + nn + 2));
        a = (double) (4*(nn + 1) * (nn + 1)) / a;
        a = a*(double) ((2*nn + 1) * (2*nn + 3));
        a = sqrt(a);
        b = (double) (2*(nn + 2) * (nn + 1));
        b = (double) ((2*m - mm) * (nn + 2) * \
            (nn + 1) - mm * n * (n + 1) + \
            mm * n1 * (n1 + 1)) / b;
        d = (double) (4*(nn + 2) * (nn + 2));
        d = d*(double) ((2*nn + 5) * (2*nn + 3));
        d = (double) ((nn + mm + 2) *(nn - mm + \
            2) * (n1 - n + nn + 2)) / d;
        d = d*(double) ((n - n1 + nn + 2) * (n +\
            n1 - nn - 1) * (n + n1 + nn + 3));
        d = sqrt(d);
        c = a * (b * c1 - d * c2);
        c2 = c1;
        c1 = c;
        cd[nn-1] = c;
        }
        }
        }
        }

for (nn = nn1; nn <= nnf; ++nn)
```

```
                                {
                                if (nn <= nnm)
                                      {
                                      gg[mind-1][nn] = cu[nn-1];
                                      }
                                if (nn > nnm)
                                      {
                                      gg[mind-1][nn] = cd[nn-1];
                                      }
                                }
                          }
                    }
              }
        }
else
        {
        printf("ERROR IN SUBROUTINE CCG\n");
        exit(0);
        }
return 0;
}

//*********************************************************************//
//************************* DIRECT function ***************************//
//*********************************************************************//

int direct(int n, int m, int n1, int m1, int nn, int mm, double *c)
{
int i, i1;
double f[900];

f[0] = 0.0;
f[1] = 0.0;
for (i = 3; i<= 900; ++i)
        {
        i1 = i - 1;
        f[i - 1] = f[i1 - 1] + log((double) i1) * 0.5;
        }

*c = f[n * 2] + f[n1 * 2] + f[n + n1 + m + m1] + f[n + n1 - m - m1];
*c = *c - f[(n + n1) * 2] - f[n + m] - f[n - m] - f[n1 + m1] - \
        f[n1 - m1];
*c = exp(*c);
return 0;
}

//*********************************************************************//
//************************** CCGIN function ***************************//
//*********************************************************************//

int ccgin(int n, int n1, int m, int mm, double *g)
{
/* Local variables */

int i,i1,k, l1, m1, l2, l3, n2, m2, m12, n12;
double a,f[900];
```

```
f[0] = 0.0;
f[1] = 0.0;
for (i = 3; i<= 900; ++i)
      {
      i1 = i - 1;
      f[i - 1] = f[i1 - 1] + log((double) i1) * 0.5;
      }
m1 = mm - m;
if (n >= abs(m) && n1 >= abs(m1) && abs(mm) <= n + n1)
      {
      if (fabs(mm) <= fabs(n - n1))
            {
            l1 = n;
            l2 = n1;
            l3 = m;
            if (n1 > n)
                  {
                  k = n;
                  n = n1;
                  n1 = k;
                  k = m;
                  m = m1;
                  m1 = k;
                  }

            n2 = 2*n;
            m2 = 2*m;
            n12 = 2*n1;
            m12 = 2*m1;
            *g = ssign[n1 + m1] * exp(f[n + m] + f[n - m] + f[n12]\
                  + f[n2 - n12 + 1] - f[n2 + 1] - f[n1 + m1] - f[n1\
                  - m1] - f[n - n1 + mm] - f[n - n1 - mm]);

            n = l1;
            n1 = l2;
            m = l3;
            return 0;
            }


      a = 1.;
      l1 = m;
      l2 = mm;
      if (mm < 0)
            {
            mm = -(mm);
            m = -(m);
            m1 = -m1;
            a = ssign[mm + n + n1];
            }

      *g = a * ssign[n + m] * exp(f[2*(mm) + 1] + f[n +n1 - mm] + \
            f[n + m] + f[n1 + m1] - f[n + n1 + mm + 1] - f[n - n1 \
            + mm] - f[-(n) + n1 + mm] - f[n - m] - f[n1 - m1]);
      m = l1;
      mm = l2;
      return 0;
```

```
      }
else
      {
      printf("ERROR IN SUBROUTINE CCGIN");
      exit(0);
      }
}

//*********************************************************************//
//************************** SAREA function ***************************//
//*********************************************************************//

int sarea(double d, double *rat)
{
/* Local variables */
double e, r;

if (d < 1)
      {
      e = sqrt(1. - d * d);
      r = (pow(d, 2.0/3.0) + pow(d,1.0/3.0) * asin(e) / e) * 0.5;
      r = sqrt(r);
      *rat = 1.0 / r;
      return 0;
      }

e = sqrt(1.0 - 1.0 / (d * d));

r = (pow(d, 2.0/3.0) * 2.0 + pow (d, (-4.0 / 3.0)) * log ((e + \
      1.0) / (1.0 - e)) / e) * 0.25;
r = sqrt(r);
*rat = 1.0 / r;
return 0;
}

//*********************************************************************//
//************************** SAREAC function **************************//
//*********************************************************************//

int sareac(double eps, double *rat)
{
*rat = pow(1.5 / eps, 1.0/3.0);
*rat = *rat/sqrt((eps + 2.0) / (eps * 2.0));
return 0;
}


//*********************************************************************//
//************************** GAUSS function ***************************//
//*********************************************************************//
// Calculates the points and weights of the Gaissian quadrature     //
// formula.                                                         //
// n = number of quadrature points                                  //
// z = Gauss division points                                        //
// w = weights of the quadrature formula                            //
//*********************************************************************//
```

```
int gauss(int n, int ind1, int ind2,double *z, double *w)
{
/* Local variables */

int i, j, k, m, ind, niter;
int boolFlag=1;
double a, b, c, f, x, dj, pa, pb, pc, zz,check;

/* Function Body */

a = 1.;
b = 2.;
c = 3.;
ind = n % 2;
k = n / 2 + ind;
f = (double) (n);
for (i= 1; i <= k; ++i)
      {
      m = n + 1 - i;
      if (i == 1)
            {
            x = a - b / ((f + a) * f);
            }
      if (i == 2)
            {
            x = (z[n-1] - a) * 4. + z[n-1];
            }
      if (i == 3)
            {
            x = (z[n - 2] - z[n-1]) * 1.6 + z[n - 2];
            }
      if (i > 3)
            {
            x = (z[m] - z[m + 1]) * c + z[m + 2];
            }
      if (i == k && ind == 1)
            {
            x = 0.0;
            }
      niter = 0;
      check = 1e-16;

      do{
      pb = 1.;
      niter=niter+1;
      if (niter > 100)
            {
            check = check*10.;
            }
      pc = x;
      dj = a;
      for (j = 2; j <= n; ++j)
            {
            dj = dj+a;
            pa = pb;
            pb = pc;
            pc = x * pb + (x * pb - pa) * (dj - a) / dj;
```

```
            }
        pa = a / ((pb - x * pc) * f);
        pb = pa * pc * (a - x * x);
        x = x - pb;
        if (fabs(pb) > check * fabs(x))
            {
            boolFlag=1;
            }
          else
            {
            boolFlag=0;
            }
        }while(boolFlag==1);

        z[m-1] = x;
        w[m-1] = pa * pa * (a - x * x);
        if (ind1 == 0)
            {
            w[m-1] = b * w[m-1];
            }
        if (!(i == k && ind == 1))
            {
            z[i-1] = -z[m-1];
            w[i-1] = w[m-1];
            }
        }

if (ind2 == 1)
        {
        printf("POINTS AND WEIGHTS OF GAUSSIAN QUADRATURE FORMULA \
            OF %d TH ORDER \n",n);

        for (i = 1; i <= k; ++i)
            {
            zz = -z[i-1];
            }
        }

if (ind1 != 0)
        {
        for (i = 1; i<= n; ++i)
            {
            z[i-1] = (a + z[i-1]) / b;
            }
        }

return 0;
}
```

```
//*****************************************************************//
//************************* DISTRB function ***********************//
//*****************************************************************//
// Calculates the number of particles for a particular radius of the//
// required size distribution                                      //
//*****************************************************************//

int distrb(int nnk, double *yy, double *wy, int ndistr, double aa,\ //
           double bb, double gam, double r1, double r2, double   \ //
           *reff, double *veff, double pi)
{
/* Local variables */

int i;
double g, x, y, b2, da, xi, dab, sum;

/* Function Body */

/* Gamma distribution: n(r) = a r^α exp(-br)    */

if (ndistr == 1)
      {
      printf("GAMMA DISTRIBUTION a, b = %f,%f \n",aa,bb);
      b2 = (1.0 - bb * 3.) / bb;
      dab = 1.0 / (aa * bb);
      for (i = 1; i <= nnk; ++i)
            {
            x = yy[i-1];
            x = pow(x, b2) * exp(-x * dab);
            wy[i-1] *= x;
            }
      }

/* Normal distribution: n(r) = 1/((2π)^½ σ_g) exp(-(r - r_g)^2 / (2σ_g^2))    */

if (ndistr == 2)
      {
      printf("NORMAL DISTRIBUTION r_g, sigma_g**2] = %f,%f \n",aa,bb);
      for (i = 1; i <= nnk; ++i)
            {
            x = yy[i-1];
            y = x - aa;
            y = exp(-y * y * 0.5 / bb);
            wy[i-1] *= y;
            }
      }

/* Lognormal distribution: n(r) = 1/((2π)^½ r ln(σ_g)) exp(-(ln r - ln r_g)^2 / (2 ln^2(σ_g)))    */

if (ndistr == 3)
      {
```

```
      printf("LOG NORMAL DISTRIBUTION r_g, [ln(sigma_g)**2] = %f,\
            %f \n",aa,bb);

      da = 1.0 / aa;
      for (i = 1; i <= nnk; ++i)
            {
            x = yy[i-1];
            y = log(x * da);
            y = exp(-y * y * .5 / bb) / x;
            wy[i-1] *= y;
            }
      }

sum = 0.;
for (i = 1; i <= nnk; ++i)
      {
      sum = sum+wy[i-1];
      }
sum = 1. / sum;

for (i = 1; i <= nnk; ++i)
      {
      wy[i-1] = wy[i-1]*sum;
      }
g = 0.;
for (i = 1; i<= nnk; ++i)
      {
      x = yy[i-1];
      g = g+x * x * wy[i-1];
      }
*reff = 0.,
for (i = 1; i <= nnk, ++i)
      {
      x = yy[i-1],
      *reff = *reff+x * x * x * wy[i-1];
      }
*reff /= g;
*veff = 0.;
for (i = 1; i <= nnk; ++i)
      {
      x = yy[i-1];
      xi = x - *reff;
      *veff = *veff + xi * xi * x * x * wy[i-1];
      }
*veff = *veff / (g * *reff * *reff);
return 0;
}

//*****************************************************************//
//************************ HOVENR Function ************************//
//*****************************************************************//
// Comparison of the computed results with the inequalities derived //
// by van der Mee and Hovenier.                                     //
//*****************************************************************//

int hovenr(int l1, double *a1, double *a2, double *a3, double *a4,
double *b1, double *b2)
```

```
{
/* Local variables */

int i, l,ll,kontr;

double c,c1, c2, c3, cc, dl,aa1, aa2, aa3, aa4, bb1, bb2, ddl;

/* Function Body */
for (l = 1; l <= ll; ++l)
      {
      kontr = 1;
      ll = l - 1;
      dl = (double) ll * 2. + 1.;
      ddl = dl * .48;
      aa1 = a1[l-1];
      aa2 = a2[l-1];
      aa3 = a3[l-1];
      aa4 = a4[l-1];
      bb1 = b1[l-1];
      bb2 = b2[l-1];
      if (ll >= 1 && fabs(aa1) >= dl)
            {
            kontr = 2;
            }
      if (fabs(aa2) >= dl)
            {
            kontr = 2;
            }
      if (fabs(aa3) >= dl)
            {
            kontr = 2;
            }
      if (fabs(aa4) >= dl)
            {
            kontr = 2;
            }
      if (fabs(bb1) >= ddl)
            {
            kontr = 2;
            }
      if (fabs(bb2) >= ddl)
            {
            kontr = 2;
            }
      if (kontr == 2)
            {
            printf("TEST FOR VAN DER MEE & HOVENIER IS NOT \
                  SATISFIED, L= %d \n",ll);
            }
      c = -0.1;
      for (i = 1; i <= ll; ++i)
            {
            c=c+ 0.1;
            cc = c * c;
            c1 = cc * bb2 * bb2;
            c2 = c * aa4;
            c3 = c * aa3;
```

```
            if ((dl - c * aa1) * (dl - c * aa2) - cc * bb1 * \
                  bb1 <= -1e-4)
                 {
                  kontr = 2;
                  }
           if ((dl - c2) * (dl - c3) + c1 <= -1e-4)
                  {
                  kontr = 2;
                  }
           if ((dl + c2) * (dl - c3) - c1 <= -1e-4)
                  {
                  kontr = 2;
                  }
           if ((dl - c2) * (dl + c3) - c1 <= -1e-4)
                  {
                  kontr = 2;
                  }
           if (kontr == 2)
                  {
                  printf("TEST FOR VAN DER MEE & HOVENIER IS \
                        NOT SATISFIED, L= %d & A= %f\n",ll,c);
                  }
             }
       }
if (kontr == 1)
       {
       printf("TEST FOR VAN DER MEE & HOVENIER IS SATISFIED\n");
       }
return 0;
}

//*********************************************************************//
//************************** MATR function ****************************//
//*********************************************************************//
// Calculates the scattering matrix elements for given expansion    //
// coefficients.                                                     //
//*********************************************************************//

int matr(double *a1, double *a2, double *a3,double *a4, double *b1,
double *b2, int lmax, int npna)
{
/* Local variables */

int l, n,i1, l1,l1max;

double p, u, f2, f3, d6,p1, p2, p3, p4, da, db, f11, f12, f22, \
      f33, f34, f44,dl, dn, tb, d11, p11, p12, p13, p14, pp1, pp2,\
      pp3, pp4, taa;
double temp;

FILE *fpr,*fpn;
fpr = fopen("matrix.dat","w");
fclose(fpr);
fpn = fopen("matrix.dat","a");

/* Function Body */
```

```
n = npna;
dn = 1.0 / (double) (n - 1);
da = acos(-1.0) * dn;
db = dn * 180.;
l1max = lmax + 1;

for (l1 = 1; l1 <= l1max; ++l1)
      {
      l = l1 - 1;
      }
tb = -db;
taa = -da;

printf("F11 F22   F33   F44   F12   F34\n");

d6 = sqrt(6.) * 0.25;

for (i1 = 1; i1 <= n; ++i1)
      {
      taa += da;
      tb += db;
      u = cos(taa);
      f11 = 0.0;
      f2 = 0.0;
      f3 = 0.0;
      f44 = 0.0;
      f12 = 0.0;
      f34 = 0.0;
      p1 = 0.0;
      p2 = 0.0;
      p3 = 0.0;
      p4 = 0.0;
      pp1 = 1.0;
      pp2 = (u + 1.0) * .25 * (u + 1.0);
      pp3 = (1.0 - u) * .25 * (1.0 - u);
      pp4 = d6 * (u * u - 1.0);
      for (l1 = 1; l1 <= l1max; ++l1)
            {
            l = l1 - 1;
            dl = (double) l;
            dl1 = (double) l1;
            f11 = f11 + a1[l1-1] * pp1;

            f44 = f44 + a4[l1-1] * pp1;
            if (l != lmax)
                  {
                  pl1 = (double) (2*l + 1);
                  p = (pl1 * u * pp1 - dl * p1) / dl1;
                  p1 = pp1;
                  pp1 = p;
                  }
          if (l < 2)
                  {
                  goto L400;
                  }
            f2 = f2 + (a2[l1-1] + a3[l1-1]) * pp2;
            f3 = f3 + (a2[l1-1] - a3[l1-1]) * pp3;
```

```
            f12 = f12 + b1[l1-1] * pp4;
            f34 = f34 + b2[l1-1] * pp4;
            if (l == lmax)
                  {
                  goto L400;
                  }
            pl2 = (double) (l * l1) * u;
            pl3 = (double) (l1 * (l * l - 4));
            pl4 = 1. / (double) (l * (l1 * l1 - 4));
            p = (pl1 * (pl2 - 4.) * pp2 - pl3 * p2) * pl4;
            p2 = pp2;
            pp2 = p;
            p = (pl1 * (pl2 + 4.) * pp3 - pl3 * p3) * pl4;
            p3 = pp3;
            pp3 = p;
            p = (pl1 * u * pp4 - sqrt((double) (l * l - 4)) * \
                  p4) / sqrt((double) (l1 * l1 - 4));
            p4 = pp4;
            pp4 = p;
L400:
          ;
            }
      if(l1==1)
            {
            temp=f11;
            }
      f22 = (f2 + f3) * 0.5;
      f33 = (f2 - f3) * 0.5;
      f22 = f22 / f11;
      f33 = f33 / f11;
      f44 = f44 / f11;
      f12 = -f12 / f11;
      f34 = f34 / f11;
      printf("%f, %f, %f, %f, %f, %f, %f \n", tb, f11/temp, f22, \
            f33, f44, f12, f34);
      fprintf(fpn,"%f, %f, %f, %f, %f, %f, %f \n", tb, f11/temp, \
            f22, f33, f44, f12,f34);
      }
fclose(fpn);
return 0;
}
```

Appendix E

Source code of TUSCAT

TUSCAT (Tezpur University SCATtering Software) is a graphical user interface (GUI) integrated software developed for modeling electromagnetic scattering from small particles and also to yield characteristic properties of real particles from experimental data. The software was programmed in Java and consists of three Java files namely TUScatGUI.java, nonSphericalClass.java and Complex.java. The first file basically deals with the GUI along with the Mie theory part (for light scattering calculations on spherical particles) of the software. The second one calculates the light scattering properties of the spheroidal and cylindrical particles. The file "Complex.java" which aids complex number functions was originally developed by Andrew G. Bennett, Department of Mathematics, Kansas State University, Manhattan, KS 66506 [367] and can be obtained through the weblink

http://www.math.ksu.edu/~bennett/jomacg/c.html.

In this section of the thesis we present the source code of the first two java files of the software. However for proper running of the software all these three files must be compiled together.

A. Source code of TUScatGUI.java

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GridBagLayout;
import java.awt.RenderingHints;
import java.awt.Stroke;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.awt.geom.Line2D;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.text.DecimalFormat;
import java.util.StringTokenizer;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTextField;

public class TUScatGUI extends JPanel {

private static JFrame frame;
private static JComboBox comboShapleList;
private static JComboBox combosizeDisList;
private static JTextField maxrrefractiveIndexTextBox;
private static JTextField maxirefractiveIndexTextBox;
private static JTextField minrrefractiveIndexTextBox;
private static JTextField minirefractiveIndexTextBox;
private static JTextField steprefractiveIndexTextBox;
private static JTextField fixedirefractiveIndexTextBox;
private static JTextField fixedrrefractiveIndexTextBox;

private static JLabel maxrrefractiveIndexTextBoxLabel;
private static JLabel maxirefractiveIndexTextBoxLabel;
private static JLabel minrrefractiveIndexTextBoxLabel;
private static JLabel minirefractiveIndexTextBoxLabel;
private static JLabel steprefractiveIndexTextBoxLabel;
private static JLabel fixedirefractiveIndexTextBoxLabel;
private static JLabel fixedrrefractiveIndexTextBoxLabel;
private static JLabel inputriLabel;
private static JLabel selectionshapeLabel;
private static JLabel realimagLabel;
private static JLabel inputREALRILabel;
private static JLabel inputIMAGRILabel;
private static JLabel inputradLabel;
private static JLabel selectabcdLabel;
private static JLabel inputabcdLabel;
private static JTextField a_bOrC_LTextBox;
private static JTextField accuracyTextBox;
private static JFrame f ;
private static JFrame f2 ;
private static JPanel p;
private static JLabel a_bOrC_LLabel;
private static JLabel accuracyLabel;

private static JTextField rrefractiveIndexTextBox;
private static JTextField irefractiveIndexTextBox;
private static JTextField waveLengthText;
private static JLabel refractiveIndexLabel;
```

```
private static JLabel rrefractiveIndexLabel;
private static JLabel irefractiveIndexLabel;
private static JLabel waveLengthLabel;
private static JTextField lowestPartRadiusText;
private static JTextField highestPartRadiusText;
private static JTextField sigmaText;
private static JTextField modalradiusText;
private static JTextField radiusTextBox;
private static JTextField minAcuracyTextBox;
private static JTextField maxAcuracyTextBox;
private static JTextField evqRadiusTextBox;
private static JTextField stepAcuracyTextBox;
private static JTextField minRadiusTextBox;
private static JTextField maxRadiusTextBox;
private static JTextField stepRadiusTextBox;
private static JTextField radiusImgRealTextBox;
private static JLabel radiusImgRealTextBoxLabel;
private static JLabel minAcuracyTextBoxLabel;
private static JLabel maxAcuracyTextBoxLabel;
private static JLabel evqRadiusTextBoxLabel;
private static JLabel stepAcuracyTextBoxLabel;
private static JLabel minRadiusTextBoxLabel;
private static JLabel maxRadiusTextBoxLabel;
private static JLabel stepRadiusTextBoxLabel;
private static JLabel radiusParameterLabel;
private static JLabel nonspParameterLabel;
private static JLabel sizedistriLabel;
private static JLabel lowestPartRadiusLabel;
private static JLabel highestPartRadiusLabel;
private static JLabel sigmaLabel;
private static JLabel modalradiusLabel;
private static JLabel extinctionCoefficentValue;
private static JLabel scattereingCoefficentValue;
private static JLabel absorptionCoefficentValue;
private static JLabel singleScatteringValue;
private static JLabel assymmetricLabeltValue;
private static JMenu fileMenu = new JMenu("Info");
private static JMenu fileMenu2 = new JMenu("Help");
static final double PI =3.14159265e+0;
static Complex refrelk,refmed;
static Complex s1k[]=new Complex[200];
static Complex s2k[]=new Complex[200];
static double xk;
static int nangk=91,nan,j;
static double
refre,refim,s11,s12,pol,s33,s34,ang,dang,s11nor,rad,wavel;

/*Data Arrays*/
static double angle[]=new double[181];
static double ss11[]=new double[181];
static double ss12[]=new double[181];
static double ss33[]=new double[181];
static double ss34[]=new double[181];
static double ss11Com[]=new double[181];
static double ss12Com[]=new double[181];
static double ss33Com[]=new double[181];
static double ss34Com[]=new double[181];
```

```
static double ss11Min[]=new double[181];
static double ss12Min[]=new double[181];
static double ss33Min[]=new double[181];
static double ss34Min[]=new double[181];
static double d[]=new double[181];
static double qscatxt;
static double qbacktxt;
static double qabstxt;
static double albedotxt;
static double qexttxt;
static double gtxt;
static double qprtxt;
static double gscatxt;

static boolean textSize=false;
static boolean textSizePara=false;
static boolean boolCompare=false;
static boolean boolExp=false;
static boolean boolMonoSphere=true;
static boolean boolNormal=false;
static boolean boolGama=false;
static boolean boolLogNormal=false;
static boolean boolDataUpload=false;
static boolean boolSize=false;
static boolean boolCalculation=false;
static boolean boolOverPlot=false;
static boolean boollogCompare=false;
static boolean boollogExp=false;
static boolean boollogOverPlot=false;
static boolean boolCalculate=false;
static boolean boolsizeRI=false;

static JLabel extinctionCoefficent;
static JLabel scattereingCoefficent;
static JLabel absorptionCoefficent;
static JRadioButton rb1, rb2;
static int np;
static boolean expBool=false;
static double errorMin=-1.0;
static JCheckBox sizeRadioButton;
static JCheckBox refractiveRadioButton;
static JCheckBox sphereRadioButton;
static JCheckBox cyclinderRadioButton;
static JCheckBox spheroidRadioButton;
static JCheckBox abcdRadioButton;
static JCheckBox eqradiusRadioButton;
static JCheckBox realindexRadioButton;
static JCheckBox imagindexRadioButton;
static JSeparator loweraccuracy ;
static JSeparator rightaccuracy ;
static JSeparator leftaccuracy ;
static JSeparator upperaccuracy;
static JSeparator lowerabcdeq;
static JSeparator rightabcdeq;
static JSeparator leftabcdeq;
static JSeparator upperabcdeq ;
static JSeparator lowerabcdin;
```

```
static JSeparator rightabcdin;
static JSeparator leftabcdin;
static JSeparator upperabcdin;
static JSeparator lowerrefin ;
static JSeparator rightrefin;
static JSeparator leftrefin;
static JSeparator upperrefin;
static JSeparator lowershape;
static JSeparator rightshape;
static JSeparator leftshape;
static JSeparator uppershape;
static JSeparator middleabcdin;
static JSeparator lowernonsp ;
static JSeparator rightnonsp;
static JSeparator leftnonsp;
static JSeparator uppernonsp;
static String input;
static String estimatedPara="";

public static void mainFrameTheory()
{
f = new JFrame("TUSCAT: Calculation of Light Scattering by particles");
f.setVisible(false);
f.setResizable(false);
f.setSize(445,720);
expBool=true;
p = new JPanel(new GridBagLayout());
p.setLayout(null);
fileMenu.setBounds(0, 0, 40, 20);
fileMenu2.setBounds(40, 0, 50, 20);
p.add(fileMenu);
p.add(fileMenu2);
JSeparator eastexdata = new JSeparator(JSeparator.VERTICAL);
JSeparator westexdata= new JSeparator(JSeparator.VERTICAL);
JSeparator southrexdata = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlexdata= new JSeparator(JSeparator.HORIZONTAL);
JSeparator southmexdata= new JSeparator(JSeparator.HORIZONTAL);
southrexdata.setBounds(20, 30, 400, 90);
eastexdata.setBounds(20, 30, 200, 70);
westexdata.setBounds(420, 30, 250, 70);
southlexdata.setBounds(20, 55, 400, 90);
southmexdata.setBounds(20, 100, 400, 90);
p.add(eastexdata);
p.add(westexdata);
p.add(southrexdata);
p.add(southlexdata);
p.add(southmexdata);
JLabel theoreticalmodeLabel=new JLabel("THEORETICAL CALCULATION MODE");
theoreticalmodeLabel.setBounds(105,32, 420, 25);
p.add(theoreticalmodeLabel);
JButton buttonexpmodeSwith=new JButton("Switch to Experimental Data
Analysis Mode");
buttonexpmodeSwith.setBounds(30,65, 380, 25);
p.add(buttonexpmodeSwith);
JSeparator northrefrac = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastrefrac = new JSeparator(JSeparator.VERTICAL);
JSeparator westrefrac = new JSeparator(JSeparator.VERTICAL);
```

```
JSeparator southrrefrac = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlrefrac = new JSeparator(JSeparator.HORIZONTAL);
northrefrac.setBounds(20, 165, 400, 90);
eastrefrac.setBounds(420, 115, 200, 50);
westrefrac.setBounds(20, 115, 200, 50);
southrrefrac.setBounds(320, 115, 100, 90);
southlrefrac.setBounds(20, 115, 100, 90);
p.add(northrefrac);
p.add(eastrefrac);
p.add(westrefrac);
p.add(southrrefrac);
p.add(southlrefrac);
JSeparator northshape = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastshape = new JSeparator(JSeparator.VERTICAL);
JSeparator westshape = new JSeparator(JSeparator.VERTICAL);
JSeparator southrshape = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlshape = new JSeparator(JSeparator.HORIZONTAL);
northshape.setBounds(20, 395, 180, 90);
eastshape.setBounds(200, 235, 200, 160);
westshape.setBounds(20, 235, 200, 160);
southrshape.setBounds(185, 235, 15, 90);
southlshape.setBounds(20, 235, 15, 90);
p.add(northshape);
p.add(eastshape);
p.add(westshape);
p.add(southrshape);
p.add(southlshape);
JSeparator northdstn = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastdstn = new JSeparator(JSeparator.VERTICAL);
JSeparator westdstn = new JSeparator(JSeparator.VERTICAL);
JSeparator southrdstn = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southldstn = new JSeparator(JSeparator.HORIZONTAL);
northdstn.setBounds(20, 665, 180, 90);
eastdstn.setBounds(200, 415, 200, 250);
westdstn.setBounds(20, 415, 200, 250);
southrdstn.setBounds(185, 415, 15, 90);
southldstn.setBounds(20, 415, 15, 90);
p.add(northdstn);
p.add(eastdstn);
p.add(westdstn);
p.add(southrdstn);
p.add(southldstn);
refractiveIndexLabel=new JLabel("Refractive index of the Particle") ;
rrefractiveIndexLabel=new JLabel("Real Part") ;
rrefractiveIndexTextBox=new JTextField();
irefractiveIndexLabel=new JLabel("Imaginary Part") ;
irefractiveIndexTextBox=new JTextField();
refractiveIndexLabel.setBounds(135,102, 200, 25);
rrefractiveIndexLabel.setBounds(35,130, 150, 25);
rrefractiveIndexTextBox.setBounds(95,130, 105, 25);
irefractiveIndexLabel.setBounds(220,130, 150, 25);
irefractiveIndexTextBox.setBounds(310,130, 105, 25);
p.add(refractiveIndexLabel);
p.add(rrefractiveIndexLabel);
p.add(rrefractiveIndexTextBox);
p.add(irefractiveIndexLabel);
p.add(irefractiveIndexTextBox);
```

```
JLabel shapeLabel=new JLabel("Particle Geometry") ;
shapeLabel.setBounds(58,223, 200, 25);
String[] comboTypesShap = { "Sphere","Cylinder", "Spheroid"};
comboShapleList=new JComboBox(comboTypesShap);
p.add(comboShapleList);
comboShapleList.setBounds(35,250, 150, 25);
p.add(shapeLabel);
a_bOrC_LLabel=new JLabel("A/B or D/L ratio") ;
a_bOrC_LTextBox=new JTextField();
a_bOrC_LLabel.setBounds(35,285, 150, 25);
a_bOrC_LTextBox.setBounds(35,310, 150, 25);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
p.add(a_bOrC_LLabel);
p.add(a_bOrC_LTextBox);
accuracyLabel=new JLabel("Accuracy of Computation") ;
accuracyTextBox=new JTextField();
accuracyLabel.setBounds(35,340, 150, 25);
accuracyTextBox.setBounds(35,365, 150, 25);
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
p.add(accuracyLabel);
p.add(accuracyTextBox);
sizedistriLabel=new JLabel("Size Distribution") ;
sizedistriLabel.setBounds(65,405, 150, 25);
p.add(sizedistriLabel);
String[] combosizeDis = { "Monodisperse","Gamma",
"Normal","Lognormal"};
combosizeDisList=new JComboBox(combosizeDis);
p.add(combosizeDisList);
combosizeDisList.setBounds(35,435, 150, 25);
radiusParameterLabel=new JLabel("Radius") ;
radiusTextBox=new JTextField();
radiusParameterLabel.setBounds(35,460, 150, 25);
radiusTextBox.setBounds(35,485, 150, 25);
p.add(radiusParameterLabel);
p.add(radiusTextBox);
lowestPartRadiusLabel=new JLabel("Lowest Particle Radius");
lowestPartRadiusText=new JTextField();
lowestPartRadiusLabel.setBounds(35,460, 150, 25);
lowestPartRadiusText.setBounds(35,485, 150, 25);
p.add(lowestPartRadiusText);
p.add(lowestPartRadiusLabel);
lowestPartRadiusLabel.setVisible(false);
lowestPartRadiusText.setVisible(false);
highestPartRadiusLabel=new JLabel("Highest Particle Radius");
highestPartRadiusText=new JTextField();
highestPartRadiusLabel.setBounds(35,510, 150, 25);
highestPartRadiusText.setBounds(35,535, 150, 25);
p.add(highestPartRadiusLabel);
p.add(highestPartRadiusText);
highestPartRadiusLabel.setVisible(false);
highestPartRadiusText.setVisible(false);
highestPartRadiusLabel.setVisible(false);
highestPartRadiusText.setVisible(false);
sigmaLabel=new JLabel("Sigma");
sigmaText=new JTextField();
```

```
sigmaLabel.setBounds(35,560, 150, 25);
sigmaText.setBounds(35,585, 150, 25);
p.add(sigmaLabel);
p.add(sigmaText);
sigmaLabel.setVisible(false);
sigmaText.setVisible(false);
modalradiusLabel=new JLabel("Modal Radius");
modalradiusText=new JTextField();
modalradiusLabel.setBounds(35,610, 150, 25);
modalradiusText.setBounds(35,635, 150, 25);
p.add(modalradiusLabel);
p.add(modalradiusText);
modalradiusLabel.setVisible(false);
modalradiusText.setVisible(false);
JSeparator northincwav = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastincwav = new JSeparator(JSeparator.VERTICAL);
JSeparator westincwav = new JSeparator(JSeparator.VERTICAL);
JSeparator southrincwav = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlincwav = new JSeparator(JSeparator.HORIZONTAL);
northincwav.setBounds(20, 220, 180, 90);
eastincwav.setBounds(200, 180, 200, 41);
westincwav.setBounds(20, 180, 180, 41);
southrincwav.setBounds(175, 180, 25, 90);
southlincwav.setBounds(20, 180, 25, 90);
p.add(northincwav);
p.add(eastincwav);
p.add(westincwav);
p.add(southrincwav);
p.add(southlincwav);
waveLengthLabel=new JLabel("Incident Wavelength") ;
waveLengthText=new JTextField();
waveLengthLabel.setBounds(54,167, 150, 25);
waveLengthText.setBounds(35,190, 150, 25);
p.add(waveLengthLabel);
p.add(waveLengthText);
JSeparator lowermedRI = new JSeparator(JSeparator.HORIZONTAL);
JSeparator rightmedRI = new JSeparator(JSeparator.VERTICAL);
JSeparator leftmedRI = new JSeparator(JSeparator.VERTICAL);
JSeparator uppermedRI = new JSeparator(JSeparator.HORIZONTAL);
lowermedRI.setBounds(220, 220, 200, 90);
rightmedRI.setBounds(420, 180, 200, 41);
leftmedRI.setBounds(220, 180, 200, 41);
uppermedRI.setBounds(220, 180, 200, 90);
p.add(lowermedRI);
p.add(rightmedRI);
p.add(leftmedRI);
p.add(uppermedRI);
JLabel medRILabel1=new JLabel("Medium refractive Index") ;
medRILabel1.setBounds(250,180, 150, 25);
p.add(medRILabel1);
JLabel medRILabel2=new JLabel(" = 1.00 + i x 0.00") ;
medRILabel2.setBounds(270,195, 150, 25);
p.add(medRILabel2);
JSeparator norththdata = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastthdata = new JSeparator(JSeparator.VERTICAL);
JSeparator westthdata= new JSeparator(JSeparator.VERTICAL);
JSeparator southrthdata = new JSeparator(JSeparator.HORIZONTAL);
```

```
norththdata.setBounds(220, 355, 200, 90);
eastthdata.setBounds(220, 235, 200, 120);
westthdata.setBounds(420, 235, 250, 120);
southrthdata.setBounds(220, 235, 200, 90);
p.add(norththdata);
p.add(eastthdata);
p.add(westthdata);
p.add(southrthdata);
JButton buttonCalculate=new JButton("Calculate");
buttonCalculate.setBounds(230,247, 180, 25);
p.add(buttonCalculate);
JButton buttonShowGraph=new JButton("Show Theoretical Plot");
buttonShowGraph.setBounds(230,285, 180, 25);
p.add(buttonShowGraph);
JButton buttonSaveData=new JButton("Save Theoretical Data");
buttonSaveData.setBounds(230,319, 180, 25);
p.add(buttonSaveData);
JSeparator northqext = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastqext = new JSeparator(JSeparator.VERTICAL);
JSeparator westqext = new JSeparator(JSeparator.VERTICAL);
JSeparator southrqext = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlqext = new JSeparator(JSeparator.HORIZONTAL);
northqext.setBounds(220, 421, 200, 90);
eastqext.setBounds(220, 381, 200, 41);
westqext.setBounds(420, 381, 200, 41);
southrqext.setBounds(220, 381, 25, 90);
southlqext.setBounds(395, 381, 25, 90);
p.add(northqext);
p.add(eastqext);
p.add(westqext);
p.add(southrqext);
p.add(southlqext);
JSeparator northqsca = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastqsca= new JSeparator(JSeparator.VERTICAL);
JSeparator westqsca = new JSeparator(JSeparator.VERTICAL);
JSeparator southrqsca = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlqsca = new JSeparator(JSeparator.HORIZONTAL);
northqsca.setBounds(220, 482, 200, 90);
eastqsca.setBounds(220, 441, 200, 41);
westqsca.setBounds(420, 441, 200, 41);
southrqsca.setBounds(220, 441, 25, 90);
southlqsca.setBounds(395, 441, 25, 90);
p.add(northqsca);
p.add(eastqsca);
p.add(westqsca);
p.add(southrqsca);
p.add(southlqsca);
JSeparator northac = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastac= new JSeparator(JSeparator.VERTICAL);
JSeparator westac = new JSeparator(JSeparator.VERTICAL);
JSeparator southrac = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlac = new JSeparator(JSeparator.HORIZONTAL);
northac.setBounds(220, 543, 200, 90);
eastac.setBounds(220, 502, 200, 41);
westac.setBounds(420, 502, 200, 41);
southrac.setBounds(220, 502, 25, 90);
southlac.setBounds(395, 502, 25, 90);
```

```
p.add(northac);
p.add(eastac);
p.add(westac);
p.add(southrac);
p.add(southlac);
JSeparator northssa = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastssa= new JSeparator(JSeparator.VERTICAL);
JSeparator westssa = new JSeparator(JSeparator.VERTICAL);
JSeparator southrssa = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlssa = new JSeparator(JSeparator.HORIZONTAL);
northssa.setBounds(220, 604, 200, 90);
eastssa.setBounds(220, 563, 200, 41);
westssa.setBounds(420, 563, 200, 41);
southrssa.setBounds(220, 563, 25, 90);
southlssa.setBounds(395, 563, 25, 90);
p.add(northssa);
p.add(eastssa);
p.add(westssa);
p.add(southrssa);
p.add(southlssa);
JSeparator northq = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastq= new JSeparator(JSeparator.VERTICAL);
JSeparator westq = new JSeparator(JSeparator.VERTICAL);
JSeparator southrq = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlq = new JSeparator(JSeparator.HORIZONTAL);
northq.setBounds(220, 665, 200, 90);
eastq.setBounds(220, 624, 200, 41);
westq.setBounds(420, 624, 200, 41);
southrq.setBounds(220, 624, 25, 90);
southlq.setBounds(395, 624, 25, 90);
p.add(northq);
p.add(eastq);
p.add(westq);
p.add(southrq);
p.add(southlq);
extinctionCoefficent=new JLabel("Extinction Efficiency") ;
extinctionCoefficentValue=new JLabel("");
extinctionCoefficent.setBounds(260,373, 200, 15);
extinctionCoefficentValue.setBounds(250,392, 200, 25);
p.add(extinctionCoefficent);
p.add(extinctionCoefficentValue);
scattereingCoefficent=new JLabel("Scattering Efficiency") ;
scattereingCoefficentValue=new JLabel();
scattereingCoefficent.setBounds(260,434, 200, 15);
scattereingCoefficentValue.setBounds(250,453, 200, 25);
p.add(scattereingCoefficent);
p.add(scattereingCoefficentValue);
absorptionCoefficent=new JLabel("Absorption Efficiency") ;
absorptionCoefficentValue=new JLabel();
absorptionCoefficent.setBounds(260,494, 200, 15);
absorptionCoefficentValue.setBounds(250,512, 200, 25);
p.add(absorptionCoefficent);
p.add(absorptionCoefficentValue);
JLabel singleScattering=new JLabel("Single Scattering Albedo") ;
singleScatteringValue=new JLabel();
singleScattering.setBounds(250,556, 200, 15);
singleScatteringValue.setBounds(250,573, 200, 25);
```

```
p.add(singleScattering);
p.add(singleScatteringValue);
JLabel assymmetricLabel=new JLabel("Asymmetry Parameter") ;
assymmetricLabeltValue=new JLabel();
assymmetricLabel.setBounds(255,617, 200, 15);
assymmetricLabeltValue.setBounds(250,634, 200, 25);
p.add(assymmetricLabel);
p.add(assymmetricLabeltValue);


buttonexpmodeSwith.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent ae){
f.setVisible(false);
mainFrameExp();
}
});

rb1 = new JRadioButton("Normal", true);
rb2 = new JRadioButton("Logarithmic");
ButtonGroup group = new ButtonGroup();
group.add(rb1); group.add(rb2);
rb1.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolExp)
{
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
}
else if(boolCompare)
{
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
}
else if(boolOverPlot)
{
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
}
if(null==frame)
{
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();

panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
else
{
frame.setVisible(false);
```

```
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
}
});
rb2.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolExp)
{
boollogCompare=false;
boollogExp=true;
boollogOverPlot=false;
}
else if(boolCompare)
{
boollogCompare=true;
boollogExp=false;
boollogOverPlot=false;
}
else if(boolOverPlot)
{
boollogCompare=false;
boollogExp=false;
boollogOverPlot=true;
}
if(null==frame)
{
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
else
{
frame.setVisible(false);
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
```

```
}
});

comboShapleList.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
JComboBox cb = (JComboBox) ae.getSource();
String newSelection = (String) cb.getSelectedItem();
if(newSelection=="Cylinder"||newSelection=="Spheroid")
{
a_bOrC_LLabel.setVisible(true);
a_bOrC_LTextBox.setVisible(true);
accuracyLabel.setVisible(true);
accuracyTextBox.setVisible(true);
radiusParameterLabel.setBounds(35,460, 150, 25);
radiusTextBox.setBounds(35,485, 150, 25);
lowestPartRadiusLabel.setBounds(35,460, 150, 25);
lowestPartRadiusText.setBounds(35,485, 150, 25);
highestPartRadiusLabel.setBounds(35,510, 150, 25);
highestPartRadiusText.setBounds(35,535, 150, 25);
sigmaLabel.setBounds(35,560, 150, 25);
sigmaText.setBounds(35,585, 150, 25);
modalradiusLabel.setBounds(35,610, 150, 25);
modalradiusText.setBounds(35,635, 150, 25);
extinctionCoefficent.setText("Extinction Coefficient");
scattereingCoefficent.setText("Scattering Coefficient");
absorptionCoefficent.setText("Absorption Coefficient");
if(newSelection=="Cylinder")
{
np=-2;
}
else
{
np=-1;
}
boolSize=true;
}
else
{
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
radiusParameterLabel.setBounds(35,460, 150, 25);
radiusTextBox.setBounds(35,485, 150, 25);
lowestPartRadiusLabel.setBounds(35,460, 150, 25);
lowestPartRadiusText.setBounds(35,485, 150, 25);
highestPartRadiusLabel.setBounds(35,510, 150, 25);
highestPartRadiusText.setBounds(35,535, 150, 25);
sigmaLabel.setBounds(35,560, 150, 25);
sigmaText.setBounds(35,585, 150, 25);
modalradiusLabel.setBounds(35,610, 150, 25);
modalradiusText.setBounds(35,635, 150, 25);
extinctionCoefficent.setText("Extinction Efficiency");
scattereingCoefficent.setText("Scattering Efficiency");
absorptionCoefficent.setText("Absorption Efficiency");
boolSize=false;
}
```

```
radiusTextBox.setText("");
a_bOrC_LTextBox.setText("");
accuracyTextBox.setText("");
sigmaText.setText("");
modalradiusText.setText("");
highestPartRadiusText.setText("");
lowestPartRadiusText.setText("");
}
});

combosizeDisList.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
JComboBox cb = (JComboBox) ae.getSource();
String newSelection = (String) cb.getSelectedItem();
if(null!=newSelection&&newSelection.equalsIgnoreCase("Monodisperse"))
{
radiusParameterLabel.setVisible(true);
radiusTextBox.setVisible(true);
lowestPartRadiusLabel.setVisible(false);
lowestPartRadiusText.setVisible(false);
highestPartRadiusLabel.setVisible(false);
highestPartRadiusText.setVisible(false);
highestPartRadiusLabel.setVisible(false);
highestPartRadiusText.setVisible(false);
sigmaLabel.setVisible(false);
sigmaText.setVisible(false);
modalradiusLabel.setVisible(false);
modalradiusText.setVisible(false);
boolMonoSphere=true;
boolNormal=false;
boolGama=false;
boolLogNormal=false;
}
else if(null!=newSelection&&newSelection.equalsIgnoreCase("Gamma"))
{
radiusParameterLabel.setVisible(false);
radiusTextBox.setVisible(false);
lowestPartRadiusLabel.setVisible(true);
lowestPartRadiusText.setVisible(true);
highestPartRadiusLabel.setVisible(true);
highestPartRadiusText.setVisible(true);
highestPartRadiusLabel.setVisible(true);
highestPartRadiusText.setVisible(true);
sigmaLabel.setVisible(true);
sigmaLabel.setText("Alfa");
sigmaText.setVisible(true);
modalradiusLabel.setVisible(true);
modalradiusText.setVisible(true);
boolMonoSphere=false;
boolNormal=false;
boolGama=true;
boolLogNormal=false;
}
else if(null!=newSelection&&newSelection.equalsIgnoreCase("Normal"))
{
radiusParameterLabel.setVisible(false);
radiusTextBox.setVisible(false);
```

```
lowestPartRadiusLabel.setVisible(true);
lowestPartRadiusText.setVisible(true);
highestPartRadiusLabel.setVisible(true);
highestPartRadiusText.setVisible(true);
highestPartRadiusLabel.setVisible(true);
highestPartRadiusText.setVisible(true);
sigmaLabel.setVisible(true);
sigmaLabel.setText("Sigma");
sigmaText.setVisible(true);
modalradiusLabel.setVisible(true);
modalradiusText.setVisible(true);
boolMonoSphere=false;
boolNormal=true;
boolGama=false;
boolLogNormal=false;
}
else if(null!=newSelection&&newSelection.equalsIgnoreCase("Lognormal"))
{
radiusParameterLabel.setVisible(false);
radiusTextBox.setVisible(false);
lowestPartRadiusLabel.setVisible(true);
lowestPartRadiusText.setVisible(true);
highestPartRadiusLabel.setVisible(true);
highestPartRadiusText.setVisible(true);
highestPartRadiusLabel.setVisible(true);
highestPartRadiusText.setVisible(true);
sigmaLabel.setVisible(true);
sigmaText.setVisible(true);
sigmaLabel.setText("Sigma");
modalradiusLabel.setVisible(true);
modalradiusText.setVisible(true);
boolMonoSphere=false;
boolNormal=false;
boolGama=false;
boolLogNormal=true;
}
radiusTextBox.setText("");
a_bOrC_LTextBox.setText("");
accuracyTextBox.setText("");
sigmaText.setText("");
modalradiusText.setText("");
highestPartRadiusText.setText("");
lowestPartRadiusText.setText("");
}
});

WindowListener wndCloser = new WindowAdapter() {
public void windowClosing(WindowEvent e) {
System.exit(0);
}
};
f.addWindowListener(wndCloser);
f.add(p);
f.show();

buttonCalculate.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
```

```
boolCompare=false;
boolExp=true;
boolOverPlot=false;
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
double radlk=0.0;
double radhk=0.0;
double stepk;
int ntotk;
for(int i=0;i<ss11.length;i++)
{
ss11[i]=0.0;
ss12[i]=0.0;
ss33[i]=0.0;
ss34[i]=0.0;
}
if(null!=rrefractiveIndexTextBox&&(null==rrefractiveIndexTextBox.getTex
t()||rrefractiveIndexTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Real Part of
Reflective Index(Particles)");
return;
}
if(null!=irefractiveIndexTextBox&&(null==irefractiveIndexTextBox.getTex
t()||irefractiveIndexTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Imaginary Part of
Reflective Index(Particles)");
return;
}
if(null!=waveLengthText&&(null==waveLengthText.getText()||waveLengthTex
t.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value forWave
LengthField");
return;
}
String strrefre=rrefractiveIndexTextBox.getText();
refre=Double.parseDouble(strrefre);
String strrefim=irefractiveIndexTextBox.getText();
refim=Double.parseDouble(strrefim);
String strrefmedreal="1";
double real=Double.parseDouble(strrefmedreal);
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
if(boolMonoSphere==true)
{
if(null!=radiusTextBox&&(null==radiusTextBox.getText()||radiusTextBox.g
etText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value
forRadius");
```

```
return;
}
String strxk2 =radiusTextBox.getText();
xk=(2.0*PI*Double.parseDouble(strxk2)*real)/Double.parseDouble(strxk1);

if(boolSize==false)
{

/**********************************/
main_Function();
}
else
{
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for A/B Or
C/L Field");
return;
}
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for
Accuracy Field");
return;
}
double axi=Double.parseDouble(strxk2);
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,eps,ddelt,-
1,np,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
}
}
else if(boolGama==true)
{
if(null!=lowestPartRadiusText&&(null==lowestPartRadiusText.getText()||l
owestPartRadiusText.getText().trim().equalsIgnoreCase("")))
```

```
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Highest
Partical Radius Field");
return;
}
if(null!=highestPartRadiusText&&(null==highestPartRadiusText.getText()|
|highestPartRadiusText.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Lowest
Partical Radius Field");
return;
}
if(null!=modalradiusText&&(null==modalradiusText.getText()||modalradius
Text.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Modal
Radius Field");
return;
}
if(null!=sigmaText&&(null==sigmaText.getText()||sigmaText.getText().tri
m().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Sigma
Field");
return;
}
String strradlk=lowestPartRadiusText.getText();
radlk=Double.parseDouble(strradlk);
String strradhk=highestPartRadiusText.getText();
radhk=Double.parseDouble(strradhk);
wavel=Double.parseDouble(strxk1);
String strxk2=modalradiusText.getText();
String stralfa=sigmaText.getText();
double rck=Double.parseDouble(strxk2);
double alfak=Double.parseDouble(stralfa);
if(boolSize==false)
{
extra(stepk,radlk,radhk,ntotk,rck,alfak,3);
}
else
{
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for A/B Or
C/L Field");
return;
}
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for
Accuracy Field");
return;
}
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
```

```
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
String
ret=nonSphericalClass.nonsp(refre,refim,1,rck,radlk,radhk,alfak,eps,dde
lt,5,np,wavel);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
}
}
else if(boolNormal==true)
{
if(null!=lowestPartRadiusText&&(null==lowestPartRadiusText.getText()||l
owestPartRadiusText.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Highest
Partical Radius Field");
return;
}
if(null!=highestPartRadiusText&&(null==highestPartRadiusText.getText()|
|highestPartRadiusText.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Lowest
Partical Radius Field");
return;
}
if(null!=modalradiusText&&(null==modalradiusText.getText()||modalradius
Text.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Modal
Radius Field");
return;
}
if(null!=sigmaText&&(null==sigmaText.getText()||sigmaText.getText().tri
m().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Sigma
Field");
return;
}
String strradlk=lowestPartRadiusText.getText();
radlk=Double.parseDouble(strradlk);
String strradhk=highestPartRadiusText.getText();
radhk=Double.parseDouble(strradhk);

wavel=Double.parseDouble(strxk1);
```

```
String strxk2=modalradiusText.getText();
String stralfa=sigmaText.getText();
double rck=Double.parseDouble(strxk2);
double alfak=Double.parseDouble(stralfa);
if(boolSize==false)
{
extra(stepk,radlk,radhk,ntotk,rck,alfak,3);
}
else
{
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for A/B Or
C/L Field");
return;
}
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for
Accuracy Field");
return;
}
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
String
ret=nonSphericalClass.nonsp(refre,refim,2,rck,radlk,radhk,alfak,eps,dde
lt,5,np,wavel);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
}
}
else if(boolLogNormal==true)
{
if(null!=lowestPartRadiusText&&(null==lowestPartRadiusText.getText()||l
owestPartRadiusText.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Highest
Partical Radius Field");
return;
}
```

```
if(null!=highestPartRadiusText&&(null==highestPartRadiusText.getText()|
|highestPartRadiusText.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Lowest
Partical Radius Field");
return;
}
if(null!=modalradiusText&&(null==modalradiusText.getText()||modalradius
Text.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Modal
Radius Field");
return;
}
if(null!=sigmaText&&(null==sigmaText.getText()||sigmaText.getText().tri
m().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for Sigma
Field");
return;
}
String strradlk=lowestPartRadiusText.getText();
radlk=Double.parseDouble(strradlk);
String strradhk=highestPartRadiusText.getText();
radhk=Double.parseDouble(strradhk);
wavel=Double.parseDouble(strxk1);
String strxk2=modalradiusText.getText();
String stralfa=sigmaText.getText();
double rck=Double.parseDouble(strxk2);
double alfak=Double.parseDouble(stralfa);
if(boolSize==false)
{
extra(stepk,radlk,radhk,ntotk,rck,alfak,3);
}
else
{
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for A/B Or
C/L Field");
return;
}
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter The Value for
Accuracy Field");
return;
}
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
String
ret=nonSphericalClass.nonsp(refre,refim,3,rck,radlk,radhk,alfak,eps,dde
lt,5,np,wavel);
```

```
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
}
}
if(qexttxt!=0.0)
{
boolCalculation=true;
}
DecimalFormat df = new DecimalFormat("#0.0#####");
extinctionCoefficentValue.setText(df.format(qexttxt));
scattereingCoefficentValue.setText(df.format(qscatxt));
absorptionCoefficentValue.setText(df.format(qabstxt));
singleScatteringValue.setText(df.format(albedotxt));
assymmetricLabeltValue.setText(df.format(gtxt));
}
});

buttonShowGraph.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent ae){
if(boolCalculation)
{
boolCompare=false;
boolExp=true;
boolOverPlot=false;
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
if(null==frame)
{
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
rb1.setSelected(true);
rb2.setSelected(false);
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new JScrollPane(new TUScatGUI()));
frame.setSize(1240, 700);
frame.setVisible(true);
}
else
{
frame.setVisible(false);
frame= new JFrame("TUSCAT: Plotting Window");
```

```
JPanel panel = new JPanel();
rb1.setSelected(true);
rb2.setSelected(false);
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new JScrollPane(new TUScatGUI()));
frame.setSize(1240, 700);
frame.setVisible(true);
}
}
else
{
JOptionPane.showMessageDialog(null, "Please Calculate First");
return;
}
}
});

buttonSaveData.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolCalculation)
{
File selectedFile =null;
JFileChooser fileChooser = new JFileChooser();
int returnValue = fileChooser.showSaveDialog(null);
if (returnValue == JFileChooser.APPROVE_OPTION) {
selectedFile = fileChooser.getSelectedFile();
try
{
BufferedWriter out = new BufferedWriter(new
FileWriter(selectedFile.getAbsolutePath()));
if(null!=out){
for(int i=0;i<ss11.length;i++)
{
out.write(Integer.toString(i));
out.write(",");
out.write(Double.toString(ss11[i]));
out.write(",");
out.write(Double.toString(ss12[i]));
out.write(",");
out.write(Double.toString(ss33[i]));
out.write(",");
out.write(Double.toString(ss34[i]));
out.newLine();
}
out.flush();
out.close();
JOptionPane.showMessageDialog(null, "Saved Successfully");
return;
}
else
{
JOptionPane.showMessageDialog(null, "Unable To Create FiLE");
return;
}
```

```
}
catch (IOException ioe) {
JOptionPane.showMessageDialog(null, "Unable To Create FiLE");
return;
}
}
}
else
{
JOptionPane.showMessageDialog(null, "Please Calculate first");
return;
}

}
});
}
public static void mainFrameExp()
{
expBool=false;
boolDataUpload=false;
f2 = new JFrame("TUSCAT: Calculation of Light Scattering by
particles");
f2.setVisible(false);
f2.setResizable(false);
f2.setSize(445, 720);
p = new JPanel(new GridBagLayout());
p.setLayout(null);
fileMenu.setBounds(0, 0, 40, 20);
fileMenu2.setBounds(40, 0, 50, 20);
p.add(fileMenu);
p.add(fileMenu2);
JLabel experimentalLabel=new JLabel("EXPERIMENTAL DATA ANALYSIS MODE");
experimentalLabel.setBounds(105,32, 420, 25);
p.add(experimentalLabel);
JSeparator northexdata = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastexdata = new JSeparator(JSeparator.VERTICAL);
JSeparator westexdata= new JSeparator(JSeparator.VERTICAL);
JSeparator southrexdata = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlexdata= new JSeparator(JSeparator.HORIZONTAL);
southrexdata.setBounds(20, 30, 400, 90);
eastexdata.setBounds(20, 30, 200, 180);
westexdata.setBounds(420, 30, 250, 180);
southlexdata.setBounds(20, 55, 400, 90);
northexdata.setBounds(20, 210, 400, 90);
p.add(northexdata);
p.add(eastexdata);
p.add(westexdata);
p.add(southrexdata);
p.add(southlexdata);
JButton buttonUploadFile=new JButton("Upload Experimental Data");
buttonUploadFile.setBounds(30,70, 180, 25);
p.add(buttonUploadFile);
JButton buttonPlottedGraph=new JButton("Show Experimental Plot");
buttonPlottedGraph.setBounds(230,70, 180, 25);
p.add(buttonPlottedGraph);
JButton buttonErrorCal=new JButton("Calculate Minimum Error");
buttonErrorCal.setBounds(30,105, 180, 25);
```

```
p.add(buttonErrorCal);
JButton buttonoverPlottedGraph=new JButton("Overplot Theoretical
Data");
buttonoverPlottedGraph.setBounds(230,105, 180, 25);
p.add(buttonoverPlottedGraph);
JButton buttonSaveData=new JButton("Save Estimated Paremeters");
buttonSaveData.setBounds(30,140, 380, 25);
p.add(buttonSaveData);
JButton buttonSwith=new JButton("Switch to Theoretical Calculation
Mode");
buttonSwith.setBounds(30,175, 380, 25);
p.add(buttonSwith);
JSeparator northincwav = new JSeparator(JSeparator.HORIZONTAL);
JSeparator eastincwav = new JSeparator(JSeparator.VERTICAL);
JSeparator westincwav = new JSeparator(JSeparator.VERTICAL);
JSeparator southrincwav = new JSeparator(JSeparator.HORIZONTAL);
JSeparator southlincwav = new JSeparator(JSeparator.HORIZONTAL);
northincwav.setBounds(20, 265, 400, 90);
eastincwav.setBounds(420, 220, 200, 45);
westincwav.setBounds(20, 220, 200, 45);
southrincwav.setBounds(20, 220, 400, 90);
p.add(northincwav);
p.add(eastincwav);
p.add(westincwav);
p.add(southrincwav);
p.add(southlincwav);
waveLengthLabel=new JLabel("Wavelength of Incident Light") ;
waveLengthText=new JTextField();
waveLengthLabel.setBounds(35,230, 200, 25);
waveLengthText.setBounds(240,230, 170, 25);
p.add(waveLengthLabel);
p.add(waveLengthText);
JSeparator lowersel = new JSeparator(JSeparator.HORIZONTAL);
JSeparator rightsel = new JSeparator(JSeparator.VERTICAL);
JSeparator leftsel = new JSeparator(JSeparator.VERTICAL);
JSeparator uppersel = new JSeparator(JSeparator.HORIZONTAL);
uppersel.setBounds(20, 275, 400, 90);
lowersel.setBounds(20, 325, 400, 90);
rightsel.setBounds(420, 275, 200, 50);
leftsel.setBounds(20, 275, 200, 50);
p.add(lowersel);
p.add(rightsel);
p.add(leftsel);
p.add(uppersel);
JLabel selectionLabel=new JLabel("Select the parameter to be
estimated") ;
selectionLabel.setBounds(110,280, 250, 25);
sizeRadioButton=new JCheckBox("Size");
sizeRadioButton.setBounds(60,300, 80, 25);
refractiveRadioButton=new JCheckBox("Refractive Index");
refractiveRadioButton.setBounds(240,300, 120, 25);
refractiveRadioButton.setVisible(true);
ButtonGroup groupParameter = new ButtonGroup();
groupParameter.add(sizeRadioButton);
groupParameter.add(refractiveRadioButton);
p.add(selectionLabel);
p.add(sizeRadioButton);
```

```
p.add(refractiveRadioButton);
buttonSwith.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
f2.setVisible(false);
mainFrameTheory();
}
});
buttonSaveData.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolCalculation)
{
File selectedFile =null;
JFileChooser fileChooser = new JFileChooser();
int returnValue = fileChooser.showSaveDialog(null);
if (returnValue == JFileChooser.APPROVE_OPTION)
selectedFile = fileChooser.getSelectedFile();
try
{
BufferedWriter out = new BufferedWriter(new
FileWriter(selectedFile.getAbsolutePath()));
if(null!=out){
out.write(estimatedPara);
out.newLine();
out.flush();
out.close();
estimatedPara="";
JOptionPane.showMessageDialog(null, "Saved Successfully");
return;
}
else
{
JOptionPane.showMessageDialog(null, "Unable To Create FiLE");
return;
}
}
catch (IOException ioe) {
JOptionPane.showMessageDialog(null, "Unable To Create FiLE");
return;
}
}
boolCalculation=false;
}
else
{estimatedPara="";
JOptionPane.showMessageDialog(null, "Please Calculate first");
return;
}
}
});

sizeRadioButton.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
boolsizeRI=true;
upperrefin.setVisible(true);
upperrefin.setBounds(20, 335, 400, 90);
lowerrefin.setVisible(true);
lowerrefin.setBounds(20, 385, 400, 90);
```

```
rightrefin.setVisible(true);
rightrefin.setBounds(420, 335, 200, 50);
leftrefin.setVisible(true);
leftrefin.setBounds(20, 335,200, 50);
uppershape.setVisible(true);
uppershape.setBounds(20, 395, 400, 90);
lowershape.setVisible(true);
lowershape.setBounds(20, 445, 400, 90);
rightshape.setVisible(true);
rightshape.setBounds(420, 395, 200, 50);
leftshape.setVisible(true);
leftshape.setBounds(20, 395,200, 50);
upperaccuracy.setVisible(false);
upperaccuracy.setBounds(20, 455, 400, 90);
loweraccuracy.setVisible(false);
loweraccuracy.setBounds(20, 505, 400, 90);
rightaccuracy.setVisible(false);
rightaccuracy.setBounds(420, 455, 200, 50);
leftaccuracy.setVisible(false);
leftaccuracy.setBounds(20, 455, 200, 50);
upperabcdeq.setVisible(false);
lowerabcdeq.setVisible(false);
rightabcdeq.setVisible(false);
leftabcdeq.setVisible(false);
upperabcdin.setVisible(false);
lowerabcdin.setVisible(false);
rightabcdin.setVisible(false);
leftabcdin.setVisible(false);
middleabcdin.setVisible(false);
uppernonsp.setVisible(false);
lowernonsp.setVisible(false);
rightnonsp.setVisible(false);
leftnonsp.setVisible(false);
fixedrrefractiveIndexTextBoxLabel.setVisible(true);
fixedrrefractiveIndexTextBox.setVisible(true);
fixedrrefractiveIndexTextBoxLabel.setVisible(true);
fixedrrefractiveIndexTextBox.setVisible(true);
fixedirefractiveIndexTextBoxLabel.setVisible(true);
fixedirefractiveIndexTextBox.setVisible(true);
inputriLabel.setVisible(true);
selectionshapeLabel.setVisible(true);
selectionshapeLabel.setBounds(150,400, 250, 25);
sphereRadioButton.setVisible(true);
sphereRadioButton.setBounds(40,420, 80, 25);
spheroidRadioButton.setVisible(true);
spheroidRadioButton.setBounds(180,420, 80, 25);
cyclinderRadioButton.setVisible(true);
cyclinderRadioButton.setBounds(320,420, 80, 25);
realimagLabel.setVisible(false);
realindexRadioButton.setVisible(false);
imagindexRadioButton.setVisible(false);
inputREALRILabel.setVisible(false);
minrrefractiveIndexTextBoxLabel.setVisible(false);
minrrefractiveIndexTextBox.setVisible(false);
maxrrefractiveIndexTextBoxLabel.setVisible(false);
maxrrefractiveIndexTextBox.setVisible(false);
steprefractiveIndexTextBoxLabel.setVisible(false);
```

```
inputIMAGRILabel.setVisible(false);
minirefractiveIndexTextBoxLabel.setVisible(false);
minirefractiveIndexTextBox.setVisible(false);
maxirefractiveIndexTextBoxLabel.setVisible(false);
maxirefractiveIndexTextBox.setVisible(false);
steprefractiveIndexTextBoxLabel.setVisible(false);
steprefractiveIndexTextBox.setVisible(false);
inputradLabel.setVisible(false);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
eqradiusRadioButton.setVisible(false);
abcdRadioButton.setVisible(false);
selectabcdLabel.setVisible(false);
radiusTextBox.setVisible(false);
radiusParameterLabel.setVisible(false);
nonspParameterLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
stepAcuracyTextBox.setVisible(false);
stepAcuracyTextBoxLabel.setVisible(false);
maxAcuracyTextBox.setVisible(false);
maxAcuracyTextBoxLabel.setVisible(false);
minAcuracyTextBox.setVisible(false);
minAcuracyTextBoxLabel.setVisible(false);
inputabcdLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
radiusImgRealTextBox.setVisible(false);
radiusImgRealTextBoxLabel.setVisible(false);
}
});

refractiveRadioButton.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
boolsizeRI=false;
upperrefin.setVisible(true);
upperrefin.setBounds(20, 335, 400, 90);
lowerrefin.setVisible(true);
lowerrefin.setBounds(20, 415, 400, 90);
rightrefin.setVisible(true);
rightrefin.setBounds(420, 335, 200, 80);
leftrefin.setVisible(true);
leftrefin.setBounds(20, 335,200, 80);
uppershape.setVisible(false);
lowershape.setVisible(false);
rightshape.setVisible(false);
leftshape.setVisible(false);
upperaccuracy.setVisible(false);
loweraccuracy.setVisible(false);
```

```
rightaccuracy.setVisible(false);
leftaccuracy.setVisible(false);
upperabcdeq.setVisible(false);
lowerabcdeq.setVisible(false);
rightabcdeq.setVisible(false);
leftabcdeq.setVisible(false);
upperabcdin.setVisible(false);
lowerabcdin.setVisible(false);
rightabcdin.setVisible(false);
leftabcdin.setVisible(false);
middleabcdin.setVisible(false);
uppernonsp.setVisible(false);
lowernonsp.setVisible(false);
rightnonsp.setVisible(false);
leftnonsp.setVisible(false);
fixedrrefractiveIndexTextBoxLabel.setVisible(false);
fixedrrefractiveIndexTextBox.setVisible(false);
fixedrrefractiveIndexTextBoxLabel.setVisible(false);
fixedrrefractiveIndexTextBox.setVisible(false);
fixedirefractiveIndexTextBoxLabel.setVisible(false);
fixedirefractiveIndexTextBox.setVisible(false);
inputriLabel.setVisible(false);
selectionshapeLabel.setVisible(false);
selectionshapeLabel.setBounds(150,455, 250, 25);
sphereRadioButton.setVisible(false);
sphereRadioButton.setBounds(40,475, 80, 25);
spheroidRadioButton.setVisible(false);
spheroidRadioButton.setBounds(180,475, 80, 25);
cyclinderRadioButton.setVisible(false);
cyclinderRadioButton.setBounds(320,475, 80, 25);
realimagLabel.setVisible(true);
realindexRadioButton.setVisible(true);
imagindexRadioButton.setVisible(true);
inputREALRILabel.setVisible(false);
minrrefractiveIndexTextBoxLabel.setVisible(false);
minrrefractiveIndexTextBox.setVisible(false);
maxrrefractiveIndexTextBoxLabel.setVisible(false);
maxrrefractiveIndexTextBox.setVisible(false);
steprefractiveIndexTextBoxLabel.setVisible(false);
inputIMAGRILabel.setVisible(false);
minirefractiveIndexTextBoxLabel.setVisible(false);
minirefractiveIndexTextBox.setVisible(false);
maxirefractiveIndexTextBoxLabel.setVisible(false);
maxirefractiveIndexTextBox.setVisible(false);
steprefractiveIndexTextBoxLabel.setVisible(false);
steprefractiveIndexTextBox.setVisible(false);
inputradLabel.setVisible(false);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
eqradiusRadioButton.setVisible(false);
abcdRadioButton.setVisible(false);
```

```
selectabcdLabel.setVisible(false);
radiusTextBox.setVisible(false);
radiusParameterLabel.setVisible(false);
nonspParameterLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
stepAcuracyTextBox.setVisible(false);
stepAcuracyTextBoxLabel.setVisible(false);
maxAcuracyTextBox.setVisible(false);
maxAcuracyTextBoxLabel.setVisible(false);
minAcuracyTextBox.setVisible(false);
minAcuracyTextBoxLabel.setVisible(false);
inputabcdLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
radiusImgRealTextBox.setVisible(false);
radiusImgRealTextBoxLabel.setVisible(false);
}
});
inputriLabel=new JLabel("Particle Refractive Index") ;
inputriLabel.setBounds(150,332, 250, 25);
inputriLabel.setVisible(false);
p.add(inputriLabel);
fixedrrefractiveIndexTextBoxLabel=new JLabel("Real Part") ;
fixedrrefractiveIndexTextBox=new JTextField();
fixedrrefractiveIndexTextBoxLabel.setBounds(40,355, 100, 25);
fixedrrefractiveIndexTextBox.setVisible(false);
fixedrrefractiveIndexTextBoxLabel.setVisible(false);
fixedrrefractiveIndexTextBox.setBounds(100,355, 75, 25);
p.add(fixedrrefractiveIndexTextBoxLabel);
p.add(fixedrrefractiveIndexTextBox);
fixedirefractiveIndexTextBoxLabel=new JLabel("Imaginary Part") ;
fixedirefractiveIndexTextBox=new JTextField();
fixedirefractiveIndexTextBoxLabel.setBounds(240,355, 160, 25);
fixedirefractiveIndexTextBox.setBounds(335,355, 75, 25);
fixedirefractiveIndexTextBoxLabel.setVisible(false);
fixedirefractiveIndexTextBox.setVisible(false);
p.add(fixedirefractiveIndexTextBoxLabel);
p.add(fixedirefractiveIndexTextBox);
inputradLabel=new JLabel("Expected Range of Radius") ;
inputradLabel.setVisible(false);
inputradLabel.setBounds(140,452, 250, 25);
p.add(inputradLabel);
minRadiusTextBoxLabel=new JLabel("Minimum") ;
minRadiusTextBox=new JTextField();
minRadiusTextBoxLabel.setBounds(40,475, 60, 25);
minRadiusTextBox.setBounds(100,475, 60, 25);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
p.add(minRadiusTextBoxLabel);
p.add(minRadiusTextBox);
maxRadiusTextBoxLabel=new JLabel("Maximum") ;
maxRadiusTextBox=new JTextField();
maxRadiusTextBoxLabel.setBounds(176,475, 60, 25);
maxRadiusTextBox.setBounds(240,475, 60, 25);
```

```
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
p.add(maxRadiusTextBoxLabel);
p.add(maxRadiusTextBox);
stepRadiusTextBoxLabel=new JLabel("Step") ;
stepRadiusTextBox=new JTextField();
stepRadiusTextBoxLabel.setBounds(315,475, 40, 25);
stepRadiusTextBox.setBounds(350,475, 60, 25);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
p.add(stepRadiusTextBoxLabel);
p.add(stepRadiusTextBox);
accuracyLabel=new JLabel("Accuracy of T-matrix Computation") ;
accuracyTextBox=new JTextField();
accuracyLabel.setBounds(40,470, 210, 25);
accuracyTextBox.setBounds(260,470, 150, 25);
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
p.add(accuracyLabel);
p.add(accuracyTextBox);
selectabcdLabel=new JLabel("Select the parameter to be estimated");
selectabcdLabel.setBounds(110,520, 250, 25);
selectabcdLabel.setVisible(false);
ButtonGroup abcdradiusButton=new ButtonGroup();
abcdRadioButton=new JCheckBox("A/B Ratio");
abcdRadioButton.setBounds(60,540, 80, 25);
abcdRadioButton.setVisible(false);
eqradiusRadioButton=new JCheckBox("Equivalent Radius");
eqradiusRadioButton.setBounds(240,540, 140, 25);
eqradiusRadioButton.setVisible(false);
abcdradiusButton.add(abcdRadioButton);
abcdradiusButton.add(eqradiusRadioButton);
p.add(selectabcdLabel);
p.add(abcdRadioButton);
p.add(eqradiusRadioButton);

abcdRadioButton.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent ae){
f2.setSize(445, 720);
upperabcdin.setVisible(true);
lowerabcdin.setVisible(true);
rightabcdin.setVisible(true);
leftabcdin.setVisible(true);
middleabcdin.setVisible(true);
stepAcuracyTextBox.setVisible(true);
stepAcuracyTextBoxLabel.setVisible(true);
maxAcuracyTextBox.setVisible(true);
maxAcuracyTextBoxLabel.setVisible(true);
minAcuracyTextBox.setVisible(true);
minAcuracyTextBoxLabel.setVisible(true);
if(sizeRadioButton.isSelected())
{
if(cyclinderRadioButton.isSelected())
{
inputabcdLabel.setText("Expected Range of D/L Ratio");
}
else
```

```
{
inputabcdLabel.setText("Expected Range of A/B Ratio");
}
}
else if(refractiveRadioButton.isSelected())
{
inputabcdLabel.setText("Volume Equivalent Sphere Radius");
}
inputabcdLabel.setVisible(true);
evqRadiusTextBox.setVisible(true);
evqRadiusTextBoxLabel.setText("Volume Equivalent Radius");
evqRadiusTextBoxLabel.setVisible(true);
evqRadiusTextBoxLabel.setBounds(40,585, 210, 25);
evqRadiusTextBox.setBounds(260,585, 150, 25);
}
});

eqradiusRadioButton.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
f2.setSize(445, 720);
upperabcdin.setVisible(true);
lowerabcdin.setVisible(true);
rightabcdin.setVisible(true);
leftabcdin.setVisible(true);
middleabcdin.setVisible(true);
stepAcuracyTextBox.setVisible(true);
stepAcuracyTextBoxLabel.setVisible(true);
maxAcuracyTextBox.setVisible(true);
maxAcuracyTextBoxLabel.setVisible(true);
minAcuracyTextBox.setVisible(true);
minAcuracyTextBoxLabel.setVisible(true);
inputabcdLabel.setText("Expected Range of Equivalent Radius");
inputabcdLabel.setVisible(true);
evqRadiusTextBox.setVisible(true);
if(sizeRadioButton.isSelected())
{
if(cyclinderRadioButton.isSelected())
{
evqRadiusTextBoxLabel.setText("D/L ratio");
}
else
{
evqRadiusTextBoxLabel.setText("A/B ratio");
}
}
else if(refractiveRadioButton.isSelected())
{
evqRadiusTextBoxLabel.setText("Volume Equivalent Sphere Radius");
}
evqRadiusTextBoxLabel.setVisible(true);
evqRadiusTextBoxLabel.setBounds(40,585, 210, 25);
evqRadiusTextBox.setBounds(260,585, 150, 25);
}
});
lowerrefin = new JSeparator(JSeparator.HORIZONTAL);
rightrefin = new JSeparator(JSeparator.VERTICAL);
leftrefin = new JSeparator(JSeparator.VERTICAL);
```

```
upperrefin = new JSeparator(JSeparator.HORIZONTAL);
upperrefin.setBounds(20, 335, 400, 90);
lowerrefin.setBounds(20, 415, 400, 90);
rightrefin.setBounds(420, 335, 200, 80);
leftrefin.setBounds(20, 335,200, 80);
upperrefin.setVisible(false);
lowerrefin.setVisible(false);
rightrefin.setVisible(false);
leftrefin.setVisible(false);
p.add(upperrefin);
p.add(lowerrefin);
p.add(rightrefin);
p.add(leftrefin);
lowershape = new JSeparator(JSeparator.HORIZONTAL);
rightshape = new JSeparator(JSeparator.VERTICAL);
leftshape = new JSeparator(JSeparator.VERTICAL);
uppershape = new JSeparator(JSeparator.HORIZONTAL);
uppershape.setVisible(false);
lowershape.setVisible(false);
rightshape.setVisible(false);
leftshape.setVisible(false);
uppershape.setBounds(20, 425, 400, 90);
lowershape.setBounds(20,475, 400, 90);
rightshape.setBounds(420, 425, 200, 50);
leftshape.setBounds(20, 425, 200, 50);
p.add(lowershape);
p.add(rightshape);
p.add(leftshape);
p.add(uppershape);
loweraccuracy = new JSeparator(JSeparator.HORIZONTAL);
rightaccuracy = new JSeparator(JSeparator.VERTICAL);
leftaccuracy = new JSeparator(JSeparator.VERTICAL);
upperaccuracy = new JSeparator(JSeparator.HORIZONTAL);
upperaccuracy.setBounds(20, 485, 400, 90);
loweraccuracy.setBounds(20, 535, 400, 90);
rightaccuracy.setBounds(420, 485, 200, 50);
leftaccuracy.setBounds(20, 485,200, 50);
upperaccuracy.setVisible(false);
loweraccuracy.setVisible(false);
rightaccuracy.setVisible(false);
leftaccuracy.setVisible(false);
p.add(upperaccuracy);
p.add(loweraccuracy);
p.add(rightaccuracy);
p.add(leftaccuracy);
lowerabcdeq = new JSeparator(JSeparator.HORIZONTAL);
rightabcdeq = new JSeparator(JSeparator.VERTICAL);
leftabcdeq = new JSeparator(JSeparator.VERTICAL);
upperabcdeq = new JSeparator(JSeparator.HORIZONTAL);
upperabcdeq.setBounds(20, 515, 400, 90);
lowerabcdeq.setBounds(20, 565, 400, 90);
rightabcdeq.setBounds(420, 515, 200, 50);
leftabcdeq.setBounds(20, 515,200, 50);
upperabcdeq.setVisible(false);
lowerabcdeq.setVisible(false);
rightabcdeq.setVisible(false);
leftabcdeq.setVisible(false);
```

```
p.add(upperabcdeq);
p.add(lowerabcdeq);
p.add(rightabcdeq);
p.add(leftabcdeq);
lowerabcdin = new JSeparator(JSeparator.HORIZONTAL);
rightabcdin = new JSeparator(JSeparator.VERTICAL);
leftabcdin = new JSeparator(JSeparator.VERTICAL);
upperabcdin = new JSeparator(JSeparator.HORIZONTAL);
upperabcdin.setBounds(20, 575, 400, 90);
lowerabcdin.setBounds(20, 675, 400, 90);
rightabcdin.setBounds(420, 575, 200, 100);
leftabcdin.setBounds(20, 575,200, 100);
upperabcdin.setVisible(false);
lowerabcdin.setVisible(false);
rightabcdin.setVisible(false);
leftabcdin.setVisible(false);
p.add(upperabcdin);
p.add(lowerabcdin);
p.add(rightabcdin);
p.add(leftabcdin);
middleabcdin = new JSeparator(JSeparator.HORIZONTAL);
middleabcdin.setBounds(20, 615, 400, 90);
middleabcdin.setVisible(false);
p.add(middleabcdin);
lowernonsp = new JSeparator(JSeparator.HORIZONTAL);
rightnonsp = new JSeparator(JSeparator.VERTICAL);
leftnonsp = new JSeparator(JSeparator.VERTICAL);
uppernonsp = new JSeparator(JSeparator.HORIZONTAL);
uppernonsp.setVisible(false);
lowernonsp.setVisible(false);
rightnonsp.setVisible(false);
leftnonsp.setVisible(false);
uppernonsp.setBounds(20, 550, 400, 90);
lowernonsp.setBounds(20, 675, 400, 90);
rightnonsp.setBounds(420, 550, 200, 125);
leftnonsp.setBounds(20, 550,200, 125);
p.add(uppernonsp);
p.add(lowernonsp);
p.add(rightnonsp);
p.add(leftnonsp);
evqRadiusTextBoxLabel=new JLabel();
evqRadiusTextBoxLabel.setText("Volume Equivalent Sphere Radius");
evqRadiusTextBoxLabel.setBounds(40,585, 210, 25);
evqRadiusTextBoxLabel.setVisible(false);
p.add(evqRadiusTextBoxLabel);
evqRadiusTextBox=new JTextField();
evqRadiusTextBox.setBounds(260,585, 150, 25);
evqRadiusTextBox.setVisible(false);
p.add(evqRadiusTextBox);
inputabcdLabel=new JLabel("Expected Range of A/B Ratio");
inputabcdLabel.setBounds(130,615, 250, 25);
inputabcdLabel.setVisible(false);
p.add(inputabcdLabel);
minAcuracyTextBoxLabel=new JLabel();
minAcuracyTextBoxLabel.setText("Minimum");
minAcuracyTextBoxLabel.setBounds(40,645, 60, 25);
minAcuracyTextBoxLabel.setVisible(false);
```

```
p.add(minAcuracyTextBoxLabel);
minAcuracyTextBox=new JTextField(),
minAcuracyTextBox.setBounds(100,645, 60, 25);
minAcuracyTextBox.setVisible(false);
p.add(minAcuracyTextBox);
maxAcuracyTextBoxLabel=new JLabel();
maxAcuracyTextBoxLabel.setText("Maximum");
maxAcuracyTextBoxLabel.setBounds(176,645, 60, 25);
maxAcuracyTextBoxLabel.setVisible(false);
p.add(maxAcuracyTextBoxLabel);
maxAcuracyTextBox=new JTextField();
maxAcuracyTextBox.setBounds(240,645, 60, 25);
maxAcuracyTextBox.setVisible(false);
p.add(maxAcuracyTextBox);
stepAcuracyTextBoxLabel=new JLabel();
stepAcuracyTextBoxLabel.setText("Step");
stepAcuracyTextBoxLabel.setBounds(315,645, 40, 25);
stepAcuracyTextBoxLabel.setVisible(false);
p.add(stepAcuracyTextBoxLabel);
stepAcuracyTextBox=new JTextField();
stepAcuracyTextBox.setBounds(350,645, 60, 25);
stepAcuracyTextBox.setVisible(false);
p.add(stepAcuracyTextBox);
realimagLabel=new JLabel("Select Real or Imaginary Part of the
Refractive Index") ;
realimagLabel.setBounds(70,332, 350, 25);
realimagLabel.setVisible(false);
p.add(realimagLabel);
ButtonGroup refractiveGroup=new ButtonGroup();
realindexRadioButton=new JCheckBox("Real Part");
realindexRadioButton.setVisible(false);
realindexRadioButton.setBounds(60,355, 80, 25);
imagindexRadioButton=new JCheckBox("Imaginary Part");
imagindexRadioButton.setVisible(false);
imagindexRadioButton.setBounds(240,355, 120, 25);
refractiveGroup.add(realindexRadioButton);
refractiveGroup.add(imagindexRadioButton);
radiusImgRealTextBox=new JTextField();
radiusImgRealTextBox.setVisible(false);
radiusImgRealTextBox.setBounds(240,384, 120, 25);
radiusImgRealTextBoxLabel=new JLabel();
radiusImgRealTextBoxLabel.setVisible(false);
radiusImgRealTextBoxLabel.setBounds(60,384, 160, 25);
p.add(realindexRadioButton);
p.add(imagindexRadioButton);
p.add(radiusImgRealTextBoxLabel);
p.add(radiusImgRealTextBox);

realindexRadioButton.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent ae){
f2.setSize(445, 580);
uppershape.setBounds(20, 425, 400, 90);
lowershape.setBounds(20,475, 400, 90);
rightshape.setBounds(420, 425, 200, 50);
leftshape.setBounds(20, 425, 200, 50);
uppershape.setVisible(true);
lowershape.setVisible(true);
```

```
rightshape.setVisible(true);
leftshape.setVisible(true);
upperaccuracy.setBounds(20, 485, 400, 90);
loweraccuracy.setBounds(20, 535, 400, 90);
rightaccuracy.setBounds(420, 485, 200, 50);
leftaccuracy.setBounds(20, 485,200, 50);
upperaccuracy.setVisible(true);
loweraccuracy.setVisible(true);
rightaccuracy.setVisible(true);
leftaccuracy.setVisible(true);
inputREALRILabel.setVisible(true);
minrrefractiveIndexTextBoxLabel.setVisible(true);
minrrefractiveIndexTextBox.setVisible(true);
maxrrefractiveIndexTextBoxLabel.setVisible(true);
maxrrefractiveIndexTextBox.setVisible(true);
steprefractiveIndexTextBoxLabel.setVisible(true);
steprefractiveIndexTextBox.setVisible(true);
inputIMAGRILabel.setVisible(false);
minirefractiveIndexTextBoxLabel.setVisible(false);
minirefractiveIndexTextBox.setVisible(false);
maxirefractiveIndexTextBoxLabel.setVisible(false);
maxirefractiveIndexTextBox.setVisible(false);
selectionshapeLabel.setVisible(true);
selectionshapeLabel.setBounds(150,485, 250, 25);
sphereRadioButton.setVisible(true);
sphereRadioButton.setBounds(40,505, 80, 25);
spheroidRadioButton.setVisible(true);
spheroidRadioButton.setBounds(180,505, 80, 25);
cyclinderRadioButton.setVisible(true);
cyclinderRadioButton.setBounds(320,505, 80, 25);
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
inputradLabel.setVisible(false);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
eqradiusRadioButton.setVisible(false);
abcdRadioButton.setVisible(false);
selectabcdLabel.setVisible(false);
radiusTextBox.setVisible(false);
radiusParameterLabel.setVisible(false);
stepAcuracyTextBox.setVisible(false);
stepAcuracyTextBoxLabel.setVisible(false);
maxAcuracyTextBox.setVisible(false);
maxAcuracyTextBoxLabel.setVisible(false);
minAcuracyTextBox.setVisible(false);
minAcuracyTextBoxLabel.setVisible(false);
inputabcdLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
radiusImgRealTextBoxLabel.setText("Enter the Imaginary part");
radiusImgRealTextBox.setVisible(true);
radiusImgRealTextBoxLabel.setVisible(true);
 }
```

```
});

imagindexRadioButton.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent ae){
f2.setSize(445, 580);
uppershape.setBounds(20, 425, 400, 90);
lowershape.setBounds(20,475, 400, 90);
rightshape.setBounds(420, 425, 200, 50);
leftshape.setBounds(20, 425, 200, 50);
uppershape.setVisible(true);
lowershape.setVisible(true);
rightshape.setVisible(true);
leftshape.setVisible(true);
upperaccuracy.setBounds(20, 485, 400, 90);
loweraccuracy.setBounds(20, 535, 400, 90);
rightaccuracy.setBounds(420, 485, 200, 50);
leftaccuracy.setBounds(20, 485,200, 50);
upperaccuracy.setVisible(true);
loweraccuracy.setVisible(true);
rightaccuracy.setVisible(true);
leftaccuracy.setVisible(true);
inputREALRILabel.setVisible(false);
minrrefractiveIndexTextBoxLabel.setVisible(false);
minrrefractiveIndexTextBox.setVisible(false);
maxrrefractiveIndexTextBoxLabel.setVisible(false);
maxrrefractiveIndexTextBox.setVisible(false);
steprefractiveIndexTextBoxLabel.setVisible(false);
inputIMAGRILabel.setVisible(true);
minirefractiveIndexTextBoxLabel.setVisible(true);
minirefractiveIndexTextBox.setVisible(true);
maxirefractiveIndexTextBoxLabel.setVisible(true);
maxirefractiveIndexTextBox.setVisible(true);
steprefractiveIndexTextBoxLabel.setVisible(true);
steprefractiveIndexTextBox.setVisible(true);
selectionshapeLabel.setVisible(true);
selectionshapeLabel.setBounds(150,485, 250, 25);
sphereRadioButton.setVisible(true);
sphereRadioButton.setBounds(40,505, 80, 25);
spheroidRadioButton.setVisible(true);
spheroidRadioButton.setBounds(180,505, 80, 25);
cyclinderRadioButton.setVisible(true);
cyclinderRadioButton.setBounds(320,505, 80, 25);
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
inputradLabel.setVisible(false);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
eqradiusRadioButton.setVisible(false);
abcdRadioButton.setVisible(false);
selectabcdLabel.setVisible(false);
radiusTextBox.setVisible(false);
radiusParameterLabel.setVisible(false);
stepAcuracyTextBox.setVisible(false);
```

```
stepAcuracyTextBoxLabel.setVisible(false);
maxAcuracyTextBox.setVisible(false);
maxAcuracyTextBoxLabel.setVisible(false);
minAcuracyTextBox.setVisible(false);
minAcuracyTextBoxLabel.setVisible(false);
inputabcdLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
radiusImgRealTextBoxLabel.setText("Enter the real part");
radiusImgRealTextBox.setVisible(true);
radiusImgRealTextBoxLabel.setVisible(true);
}
});
inputREALRILabel=new JLabel("Expected Range of Real Part of the
Refractive Index") ;
inputREALRILabel.setBounds(70,422, 350, 25);
inputREALRILabel.setVisible(false);
p.add(inputREALRILabel);
minrrefractiveIndexTextBoxLabel=new JLabel("Minimum") ;
minrrefractiveIndexTextBox=new JTextField();
minrrefractiveIndexTextBoxLabel.setBounds(40,445, 60, 25);
minrrefractiveIndexTextBox.setBounds(100,445, 60, 25);
minrrefractiveIndexTextBoxLabel.setVisible(false);
minrrefractiveIndexTextBox.setVisible(false);
p.add(minrrefractiveIndexTextBoxLabel);
p.add(minrrefractiveIndexTextBox);
maxrrefractiveIndexTextBoxLabel=new JLabel("Maximum") ;
maxrrefractiveIndexTextBox=new JTextField();
maxrrefractiveIndexTextBoxLabel.setBounds(176,445, 60, 25);
maxrrefractiveIndexTextBox.setBounds(240,445, 60, 25);
maxrrefractiveIndexTextBoxLabel.setVisible(false);
maxrrefractiveIndexTextBox.setVisible(false);
p.add(maxrrefractiveIndexTextBoxLabel);
p.add(maxrrefractiveIndexTextBox);
steprefractiveIndexTextBoxLabel=new JLabel("Step") ;
steprefractiveIndexTextBox=new JTextField();
steprefractiveIndexTextBoxLabel.setBounds(315,445, 40, 25);
steprefractiveIndexTextBox.setBounds(350,445, 60, 25);
steprefractiveIndexTextBoxLabel.setVisible(false);
steprefractiveIndexTextBox.setVisible(false);
p.add(steprefractiveIndexTextBoxLabel);
p.add(steprefractiveIndexTextBox);
inputIMAGRILabel=new JLabel("Expected Range of Imaginary Part of the
Refractive Index") ;
inputIMAGRILabel.setBounds(60,422, 350, 25);
inputIMAGRILabel.setVisible(false);
p.add(inputIMAGRILabel);
minirefractiveIndexTextBoxLabel=new JLabel("Minimum") ;
minirefractiveIndexTextBox=new JTextField();
minirefractiveIndexTextBoxLabel.setBounds(40,445, 60, 25);
minirefractiveIndexTextBox.setBounds(100,445, 60, 25);
minirefractiveIndexTextBoxLabel.setVisible(false);
minirefractiveIndexTextBox.setVisible(false);
p.add(minirefractiveIndexTextBoxLabel);
p.add(minirefractiveIndexTextBox);
maxirefractiveIndexTextBoxLabel=new JLabel("Maximum") ;
maxirefractiveIndexTextBox=new JTextField();
```

```
maxirefractiveIndexTextBoxLabel.setBounds(176,445, 60, 25);
maxirefractiveIndexTextBox.setBounds(240,445, 60, 25);
maxirefractiveIndexTextBoxLabel.setVisible(false);
maxirefractiveIndexTextBox.setVisible(false);
p.add(maxirefractiveIndexTextBoxLabel);
p.add(maxirefractiveIndexTextBox);
steprefractiveIndexTextBoxLabel=new JLabel("Step") ;
steprefractiveIndexTextBox=new JTextField();
steprefractiveIndexTextBoxLabel.setBounds(315,445, 40, 25);
steprefractiveIndexTextBox.setBounds(350,445, 60, 25);
steprefractiveIndexTextBoxLabel.setVisible(false);
steprefractiveIndexTextBox.setVisible(false);
p.add(steprefractiveIndexTextBoxLabel);
p.add(steprefractiveIndexTextBox);
selectionshapeLabel=new JLabel("Select Particle Geometry") ;
selectionshapeLabel.setBounds(150,455, 250, 25);
selectionshapeLabel.setVisible(false);
ButtonGroup shapeGroup=new ButtonGroup();
sphereRadioButton=new JCheckBox("Sphere");
sphereRadioButton.setBounds(40,475, 80, 25);
sphereRadioButton.setVisible(false);
spheroidRadioButton=new JCheckBox("Spheroid");
spheroidRadioButton.setBounds(180,475, 80, 25);
spheroidRadioButton.setVisible(false);
cyclinderRadioButton=new JCheckBox("Cylinder");
cyclinderRadioButton.setBounds(320,475, 80, 25);
cyclinderRadioButton.setVisible(false);
shapeGroup.add(sphereRadioButton);
shapeGroup.add(spheroidRadioButton);
shapeGroup.add(cyclinderRadioButton);
sphereRadioButton.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
upperaccuracy.setVisible(true);
loweraccuracy.setVisible(true);
rightaccuracy.setVisible(true);
leftaccuracy.setVisible(true);
upperabcdeq.setVisible(false);
lowerabcdeq.setVisible(false);
rightabcdeq.setVisible(false);
leftabcdeq.setVisible(false);
upperabcdin.setVisible(false);
lowerabcdin.setVisible(false);
rightabcdin.setVisible(false);
leftabcdin.setVisible(false);
middleabcdin.setVisible(false);
uppernonsp.setVisible(false);
lowernonsp.setVisible(false);
rightnonsp.setVisible(false);
leftnonsp.setVisible(false);
stepAcuracyTextBox.setVisible(false);
stepAcuracyTextBoxLabel.setVisible(false);
maxAcuracyTextBox.setVisible(false);
maxAcuracyTextBoxLabel.setVisible(false);
minAcuracyTextBox.setVisible(false);
minAcuracyTextBoxLabel.setVisible(false);
inputabcdLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
```

```
evqRadiusTextBoxLabel.setVisible(false);
if(!boolsizeRI)
{
radiusTextBox.setVisible(true);
radiusParameterLabel.setVisible(true);
f2.setSize(445, 640);
}
else
{
radiusTextBox.setVisible(false);
radiusParameterLabel.setVisible(false);
f2.setSize(445, 550);
}
if(boolsizeRI)
{
inputradLabel.setVisible(true);
minRadiusTextBoxLabel.setVisible(true);
minRadiusTextBox.setVisible(true);
maxRadiusTextBoxLabel.setVisible(true);
maxRadiusTextBox.setVisible(true);
stepRadiusTextBoxLabel.setVisible(true);
stepRadiusTextBox.setVisible(true);
nonspParameterLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
}
else
{
inputradLabel.setVisible(false);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
nonspParameterLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
upperabcdeq.setBounds(20, 545, 400, 90);
lowerabcdeq.setBounds(20, 595, 400, 90);
rightabcdeq.setBounds(420, 545, 200, 50);
leftabcdeq.setBounds(20, 545,200, 50);
upperabcdeq.setVisible(true);
lowerabcdeq.setVisible(true);
rightabcdeq.setVisible(true);
leftabcdeq.setVisible(true);
}
accuracyLabel.setVisible(false);
accuracyTextBox.setVisible(false);
eqradiusRadioButton.setVisible(false);
abcdRadioButton.setVisible(false);
selectabcdLabel.setVisible(false);
}
```

```
});

spheroidRadioButton.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
upperaccuracy.setVisible(true);
loweraccuracy.setVisible(true);
rightaccuracy.setVisible(true);
leftaccuracy.setVisible(true);
upperabcdin.setVisible(false);
lowerabcdin.setVisible(false);
rightabcdin.setVisible(false);
leftabcdin.setVisible(false);
middleabcdin.setVisible(false);
stepAcuracyTextBox.setVisible(false);
stepAcuracyTextBoxLabel.setVisible(false);
maxAcuracyTextBox.setVisible(false);
maxAcuracyTextBoxLabel.setVisible(false);
minAcuracyTextBox.setVisible(false);
minAcuracyTextBoxLabel.setVisible(false);
inputabcdLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
radiusTextBox.setVisible(false);
radiusParameterLabel.setVisible(false);
inputradLabel.setVisible(false);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
if(boolsizeRI)
{
eqradiusRadioButton.setVisible(true);
abcdRadioButton.setVisible(true);
selectabcdLabel.setVisible(true);
accuracyLabel.setVisible(true);
accuracyTextBox.setVisible(true);
accuracyLabel.setBounds(40,470, 210, 25);
accuracyTextBox.setBounds(260,470, 150, 25);
abcdRadioButton.setText("A/B Ratio");
nonspParameterLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
upperabcdeq.setVisible(true);
lowerabcdeq.setVisible(true);
rightabcdeq.setVisible(true);
leftabcdeq.setVisible(true);
f2.setSize(445, 610);
}
else
{
eqradiusRadioButton.setVisible(false);
abcdRadioButton.setVisible(false);
selectabcdLabel.setVisible(false);
```

```
accuracyLabel.setVisible(true);
accuracyTextBox.setVisible(true);
accuracyLabel.setBounds(35,580, 200, 25);
accuracyTextBox.setBounds(260,580, 150, 25);
nonspParameterLabel.setVisible(true);
evqRadiusTextBox.setVisible(true);
evqRadiusTextBoxLabel.setVisible(true);
evqRadiusTextBox.setBounds(260,640, 150, 25);
evqRadiusTextBoxLabel.setBounds(35,640, 200, 25);
a_bOrC_LLabel.setText("Horizontal to Rotational Axis Ratio, A/B");
a_bOrC_LLabel.setVisible(true);
a_bOrC_LTextBox.setVisible(true);
uppernonsp.setVisible(true);
lowernonsp.setVisible(true);
rightnonsp.setVisible(true);
leftnonsp.setVisible(true);
upperabcdeq.setVisible(false);
lowerabcdeq.setVisible(false);
rightabcdeq.setVisible(false);
leftabcdeq.setVisible(false);
f2.setSize(445, 720);
}
}
});

cyclinderRadioButton.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
upperaccuracy.setVisible(true);
loweraccuracy.setVisible(true);
rightaccuracy.setVisible(true);
leftaccuracy.setVisible(true);
upperabcdin.setVisible(false);
lowerabcdin.setVisible(false);
rightabcdin.setVisible(false);
leftabcdin.setVisible(false);
middleabcdin.setVisible(false);
stepAcuracyTextBox.setVisible(false);
stepAcuracyTextBoxLabel.setVisible(false);
maxAcuracyTextBox.setVisible(false);
maxAcuracyTextBoxLabel.setVisible(false);
minAcuracyTextBox.setVisible(false);
minAcuracyTextBoxLabel.setVisible(false);
inputabcdLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
radiusTextBox.setVisible(false);
radiusParameterLabel.setVisible(false);
inputradLabel.setVisible(false);
minRadiusTextBoxLabel.setVisible(false);
minRadiusTextBox.setVisible(false);
maxRadiusTextBoxLabel.setVisible(false);
maxRadiusTextBox.setVisible(false);
stepRadiusTextBoxLabel.setVisible(false);
stepRadiusTextBox.setVisible(false);
if(boolsizeRI)
{
eqradiusRadioButton.setVisible(true);
```

```
abcdRadioButton.setVisible(true);
selectabcdLabel.setVisible(true);
accuracyLabel.setVisible(true);
accuracyTextBox.setVisible(true);
accuracyLabel.setBounds(40,470, 210, 25);
accuracyTextBox.setBounds(260,470, 150, 25);
abcdRadioButton.setText("D/L Ratio");
nonspParameterLabel.setVisible(false);
evqRadiusTextBox.setVisible(false);
evqRadiusTextBoxLabel.setVisible(false);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
upperabcdeq.setVisible(true);
lowerabcdeq.setVisible(true);
rightabcdeq.setVisible(true);
leftabcdeq.setVisible(true);
f2.setSize(445, 610);
}
else
{
eqradiusRadioButton.setVisible(false);
abcdRadioButton.setVisible(false);
selectabcdLabel.setVisible(false);
accuracyLabel.setVisible(true);
accuracyTextBox.setVisible(true);
accuracyLabel.setBounds(35,580, 200, 25);
accuracyTextBox.setBounds(260,580, 150, 25);
nonspParameterLabel.setVisible(true);
evqRadiusTextBox.setVisible(true);
evqRadiusTextBoxLabel.setVisible(true);
evqRadiusTextBox.setBounds(260,640, 150, 25);
evqRadiusTextBoxLabel.setBounds(35,640, 200, 25);
a_bOrC_LLabel.setText("Diameter to Length Ratio, D/L");
a_bOrC_LLabel.setVisible(true);
a_bOrC_LTextBox.setVisible(true);
uppernonsp.setVisible(true);
lowernonsp.setVisible(true);
rightnonsp.setVisible(true);
leftnonsp.setVisible(true);
upperabcdeq.setVisible(false);
lowerabcdeq.setVisible(false);
rightabcdeq.setVisible(false);
leftabcdeq.setVisible(false);
f2.setSize(445, 720);
}
}
});
p.add(selectionshapeLabel);
p.add(sphereRadioButton);
p.add(spheroidRadioButton);
p.add(cyclinderRadioButton);
radiusParameterLabel=new JLabel("Enter the Radius of the Particle") ;
radiusParameterLabel.setBounds(35,560, 200, 25);
radiusParameterLabel.setVisible(false);
radiusTextBox=new JTextField();
radiusTextBox.setBounds(240,560, 170, 25);
radiusTextBox.setVisible(false);
```

```
p.add(radiusParameterLabel);
p.add(radiusTextBox);
nonspParameterLabel=new JLabel("Enter the Values of the Parameters") ;
nonspParameterLabel.setBounds(120,550, 200, 25);
nonspParameterLabel.setVisible(false);
p.add(nonspParameterLabel);
a_bOrC_LLabel=new JLabel("Horizontal to Rotational Axis Ratio, A/B") ;
a_bOrC_LTextBox=new JTextField();
a_bOrC_LLabel.setBounds(35,610, 220, 25);
a_bOrC_LTextBox.setBounds(260,610, 150, 25);
a_bOrC_LLabel.setVisible(false);
a_bOrC_LTextBox.setVisible(false);
p.add(a_bOrC_LLabel);
p.add(a_bOrC_LTextBox);
evqRadiusTextBoxLabel=new JLabel();
evqRadiusTextBoxLabel.setText("Volume Equivalent Sphere Radius");
evqRadiusTextBoxLabel.setBounds(35,630, 200, 25);
evqRadiusTextBoxLabel.setVisible(false);
p.add(evqRadiusTextBoxLabel);
evqRadiusTextBox=new JTextField();
evqRadiusTextBox.setBounds(260,630, 150, 25);
evqRadiusTextBox.setVisible(false);
p.add(evqRadiusTextBox);

buttonUploadFile.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
File selectedFile =null;
JFileChooser fileChooser = new JFileChooser();
int returnValue = fileChooser.showOpenDialog(null);
if (returnValue == JFileChooser.APPROVE_OPTION) {
selectedFile = fileChooser.getSelectedFile();
}
boolCompare=true;
boolExp=false;
for(int i=0;i<ss11.length;i++)
{
ss11Com[i]=0.0;
ss12Com[i]=0.0;
ss33Com[i]=0.0;
ss34Com[i]=0.0;
}

try{
// Open file
// command line parameter
FileInputStream fstream = new
FileInputStream(selectedFile.getAbsolutePath());
DataInputStream in = new DataInputStream(fstream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
String value;
String strLine;
int count=0;
while ((strLine = br.readLine()) != null){
if(null!=strLine)
{
int countIn=0;
try{
```

```
StringTokenizer st =
new StringTokenizer(strLine, ",");
countIn=0;
while (st.hasMoreElements())
{
String token = st.nextToken();
countIn++;
if(countIn==1)
{
double valueDouble = Double.parseDouble(token.trim());
angle[count]=valueDouble;
}
if(countIn==2)
{
double valueDouble = Double.parseDouble(token.trim());
ss11Com[count]=valueDouble;
}
if(countIn==3)
{
double valueDouble = Double.parseDouble(token.trim());
ss12Com[count]=valueDouble;
}
if(countIn==4)
{
double valueDouble = Double.parseDouble(token.trim());
ss33Com[count]=valueDouble;
}
if(countIn==5)
{
double valueDouble = Double.parseDouble(token.trim());
ss34Com[count]=valueDouble;
}
}
boolDataUpload=true;
}
catch (NumberFormatException exception) {
exception.printStackTrace();
}
count++;
}
}
//Close the input stream
in.close();
}catch (Exception e){//Catch exception if any
System.err.println("Error: " + e.getMessage());
}
}
});

buttonErrorCal.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
double radlk=0.0;
double radhk=0.0;
double stepk;
int ntotk;
boolCalculation=true;
if(!boolDataUpload)
```

```
{
JOptionPane.showMessageDialog(null, "Please Upload Experimental Data");
return;
}
if(null!=waveLengthText&&(null==waveLengthText.getText()||waveLengthTex
t.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Wave Length
Value");
return;
}
if(!sizeRadioButton.isSelected()&& !refractiveRadioButton.isSelected())
{
JOptionPane.showMessageDialog(null, "Please Select Parameter to be
Estimated");
return;
}
if(sizeRadioButton.isSelected())
{
if(null!=fixedrrefractiveIndexTextBox&&(null==fixedrrefractiveIndexText
Box.getText()||fixedrrefractiveIndexTextBox.getText().trim().equalsIgno
reCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Refractive
Index(Real)");
return;
}
if(null!=fixedirefractiveIndexTextBox&&(null==fixedirefractiveIndexText
Box.getText()||fixedirefractiveIndexTextBox.getText().trim().equalsIgno
reCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Refractive
Index(Imaginary)");
return;
}
if(!sphereRadioButton.isSelected()&&
!cyclinderRadioButton.isSelected()&& !spheroidRadioButton.isSelected())
{
JOptionPane.showMessageDialog(null, "Please Select Particle Geometry");
return;
}
if(sphereRadioButton.isSelected())
{
if(null!=minRadiusTextBox&&(null==minRadiusTextBox.getText()||minRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Minimum Radius");
return;
}
if(null!=maxRadiusTextBox&&(null==maxRadiusTextBox.getText()||maxRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Maximum Radius");
return;
}
if(null!=stepRadiusTextBox&&(null==stepRadiusTextBox.getText()||stepRad
iusTextBox.getText().trim().equalsIgnoreCase("")))
```

```
{
JOptionPane.showMessageDialog(null, "Please Enter the Step value for
Radius");
return;
}
String strrefre=fixedrrefractiveIndexTextBox.getText();
refre=Double.parseDouble(strrefre);
String strrefim=fixedirefractiveIndexTextBox.getText();
refim=Double.parseDouble(strrefim);
String strrefmedreal="1";
double real=Double.parseDouble(strrefmedreal);
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
double minRadius= Double.parseDouble( minRadiusTextBox.getText());
double maxRadius= Double.parseDouble( maxRadiusTextBox.getText());
double stepRadius= Double.parseDouble(stepRadiusTextBox.getText());
double valueGet=0.0;
for(double j=minRadius;j<=maxRadius;j=j+stepRadius)
{
xk=(2.0*PI*j*real)/Double.parseDouble(strxk1);
main_Function();
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
```

```
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Refractive Index(Real)
:"+fixedrrefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+fixedirefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Radius :"+valueGet+"\n";
return;
}
if(cyclinderRadioButton.isSelected())
{
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Accuracy of T-
Matrix Computation");
return;
}
if(!abcdRadioButton.isSelected()&& !eqradiusRadioButton.isSelected())
{
JOptionPane.showMessageDialog(null, "Select Parameter to be
estimated");
return;
}
if(abcdRadioButton.isSelected())
{
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Equivalent Sphere
Radius");
return;
}
if(null!=minAcuracyTextBox&&(null==minAcuracyTextBox.getText()||minAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Expected Range of
D/L(Min)");
return;
}
if(null!=maxAcuracyTextBox&&(null==maxAcuracyTextBox.getText()||maxAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Expected Range of
D/L(Max)");
return;
}
if(null!=stepAcuracyTextBox&&(null==stepAcuracyTextBox.getText()||stepA
curacyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Increament
Step");
return;
}
String strrefre=fixedrrefractiveIndexTextBox.getText();
```

```
refre=Double.parseDouble(strrefre);
String strrefim=fixedirefractiveIndexTextBox.getText();
refim=Double.parseDouble(strrefim);
String strrefmedreal="1";
double real=Double.parseDouble(strrefmedreal);
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
double axi=Double.parseDouble(evqRadiusTextBox.getText());
double minRadius= Double.parseDouble( minAcuracyTextBox.getText());
double maxRadius= Double.parseDouble( maxAcuracyTextBox.getText());
double stepRadius= Double.parseDouble(stepAcuracyTextBox.getText());
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
double valueget=0.0;
for(double j=minRadius;j<=maxRadius;j=j+stepRadius)
{
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,j,ddelt,-
1,-2,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
 ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueget=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueget=j;
for(int i=0;i<181;i++)
```

```
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real)
:"+fixedrrefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+fixedirefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere Radius
:"+evqRadiusTextBox.getText()+"\n";
estimatedPara=estimatedPara+"A/B:"+valueget+"\n";
return;
}
if(eqradiusRadioButton.isSelected())
{
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Equivalent D/L");
return;
}
if(null!=minAcuracyTextBox&&(null==minAcuracyTextBox.getText()||minAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent
radius(Min)");
return;
}
if(null!=maxAcuracyTextBox&&(null==maxAcuracyTextBox.getText()||maxAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent
radius(Max)");
return;
}
if(null!=stepAcuracyTextBox
&&(null==stepAcuracyTextBox.getText()||stepAcuracyTextBox.getText().tri
m().equalsIgnoreCase("")))
{
 JOptionPane.showMessageDialog(null, "Please Enter increament Step");
 return;
```

```
}
String strrefre=fixedrrefractiveIndexTextBox.getText();
refre=Double.parseDouble(strrefre);
String strrefim=fixedirefractiveIndexTextBox.getText();
refim=Double.parseDouble(strrefim);
String strrefmedreal="1";
double real=Double.parseDouble(strrefmedreal);
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
double axi=Double.parseDouble(evqRadiusTextBox.getText());
double minRadius= Double.parseDouble( minAcuracyTextBox.getText());
double maxRadius= Double.parseDouble( maxAcuracyTextBox.getText());
double stepRadius= Double.parseDouble(stepAcuracyTextBox.getText());;
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
double valueGet=0.0;
for(double j=minRadius;j<=maxRadius;j=j+stepRadius)
{
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,j,ddelt,-
1,-2,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=errorMin;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
```

```
valueGet=errorMin;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real)
:"+fixedrrefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+fixedirefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"A/B :"+evqRadiusTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere Radius:"+valueGet+"\n";
return;
}
 }
if(spheroidRadioButton.isSelected())
{
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Accuracy of T-
Matrix Computation");
return;
}
if(!abcdRadioButton.isSelected()&& !eqradiusRadioButton.isSelected())
{
JOptionPane.showMessageDialog(null, "Select Parameter to be
estimated");
return;
}
if(abcdRadioButton.isSelected())
{
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Equivalent Sphere
Radius");
return;
}
if(null!=minAcuracyTextBox&&(null==minAcuracyTextBox.getText()||minAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
```

```
{
JOptionPane.showMessageDialog(null, "Please Enter the Expected Range of
A/B(Min)");
return;
}
if(null!=maxAcuracyTextBox&&(null==maxAcuracyTextBox.getText()||maxAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Expected Range of
A/B(Max)");
return;
}
if(null!=stepAcuracyTextBox&&(null==stepAcuracyTextBox.getText()||stepA
curacyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter Increament Step");
return;
}
String strrefre=fixedrrefractiveIndexTextBox.getText();
refre=Double.parseDouble(strrefre);
String strrefim=fixedirefractiveIndexTextBox.getText();
refim=Double.parseDouble(strrefim);
String strrefmedreal="1";
double real=Double.parseDouble(strrefmedreal);
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
double axi=Double.parseDouble(evqRadiusTextBox.getText());
double minRadius= Double.parseDouble( minAcuracyTextBox.getText());
double maxRadius= Double.parseDouble( maxAcuracyTextBox.getText());
double stepRadius= Double.parseDouble(stepAcuracyTextBox.getText());
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
double valueget=0.0;
for(double j=minRadius;j<=maxRadius;j=j+stepRadius)
{
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,j,ddelt,-
1,-1,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
```

```
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueget=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueget=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real)
:"+fixedrrefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+fixedirefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere Radius
:"+evqRadiusTextBox.getText()+"\n";
estimatedPara=éstimatedPara+"A/B:"+valueget+"\n";
return;
}
if(eqradiusRadioButton.isSelected())
{
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter A/B ratio");
return;
}
```

```
if(null!=minAcuracyTextBox&&(null==minAcuracyTextBox.getText()||minAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent
radius(Min)");
return;
}
if(null!=maxAcuracyTextBox&&(null==maxAcuracyTextBox.getText()||maxAcur
acyTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent
radius(Max)");
return;
}
if(null!=stepAcuracyTextBox
&&(null==stepAcuracyTextBox.getText()||stepAcuracyTextBox.getText().tri
m().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter increament Step");
return;
}
String strrefre=fixedrrefractiveIndexTextBox.getText();
refre=Double.parseDouble(strrefre);
String strrefim=fixedirefractiveIndexTextBox.getText();
refim=Double.parseDouble(strrefim);
String strrefmedreal="1";
double real=Double.parseDouble(strrefmedreal);
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
double axi=Double.parseDouble(evqRadiusTextBox.getText());
double minRadius= Double.parseDouble( minAcuracyTextBox.getText());
double maxRadius= Double.parseDouble( maxAcuracyTextBox.getText());
double stepRadius= Double.parseDouble(stepAcuracyTextBox.getText());
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
double valueGet=0.0;
for(double j=minRadius;j<=maxRadius;j=j+stepRadius)
{
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,j,ddelt,-
1,-1,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
```

```
qscatxt=nonSphericalClass.cscat;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real)
:"+fixedrrefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+fixedirefractiveIndexTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"D/L :"+evqRadiusTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere Radius:"+valueGet+"\n";
return;
}
}
}
if(refractiveRadioButton.isSelected())
{
if(!realindexRadioButton.isSelected()&&!imagindexRadioButton.isSelected
())
{
```

```
JOptionPane.showMessageDialog(null, "Select parameter to be
estimated");
return;
}
if(realindexRadioButton.isSelected())
{
if(null!=radiusImgRealTextBox&&(null==radiusImgRealTextBox.getText()||r
adiusImgRealTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter Refractive
Index(Imaginary)");
return;
}
if(null!=minrrefractiveIndexTextBox&&(null==minrrefractiveIndexTextBox.
getText()||minrrefractiveIndexTextBox.getText().trim().equalsIgnoreCase
(""")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Min value for
Refractive Index(Real)");
return;
}
if(null!=maxrrefractiveIndexTextBox&&(null==maxrrefractiveIndexTextBox.
getText()||maxrrefractiveIndexTextBox.getText().trim().equalsIgnoreCase
(""")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Max value for
Refractive Index(Real)");
return;
}
if(null!=steprefractiveIndexTextBox&&(null==steprefractiveIndexTextBox.
getText()||steprefractiveIndexTextBox.getText().trim().equalsIgnoreCase
(""")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Step value for
Refractive Index(Real)");
return;
}
if(!sphereRadioButton.isSelected()&&!cyclinderRadioButton.isSelected()&
&!spheroidRadioButton.isSelected())
{
JOptionPane.showMessageDialog(null, "Enter Particle Geometry");
return;
}
if(sphereRadioButton.isSelected())
{
if(null!=radiusTextBox&&(null==radiusTextBox.getText()||radiusTextBox.g
etText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the value of the
radius");
return;
}
double valueGet=0.0;
double
strrefremax=Double.parseDouble(maxrrefractiveIndexTextBox.getText());
double
strrefremin=Double.parseDouble(minrrefractiveIndexTextBox.getText());
```

```
double
strrefrestep=Double.parseDouble(steprefractiveIndexTextBox.getText());
for(double j=strrefremin;j<=strrefremax;j=j+strrefrestep)
{
refre=j;
String strrefim=radiusImgRealTextBox.getText();
refim=Double.parseDouble(strrefim);
double real=1.0;
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
String strxk2 =radiusTextBox.getText(),
xk=(2.0*PI*Double.parseDouble(strxk2)*real)/Double.parseDouble(strxk1);
main_Function();
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real) :"+valueGet+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+radiusImgRealTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Radius::"+radiusTextBox.getText()+"\n";
```

```
return;
}
if(cyclinderRadioButton.isSelected())
{
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Accuracy of T-
Matrix Computation");
return;
}
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter D/L");
return;
}
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent radius");
return;
}
double valueGet=0.0;
double
strrefremax=Double.parseDouble(maxrrefractiveIndexTextBox.getText());
double
strrefremin=Double.parseDouble(minrrefractiveIndexTextBox.getText());
double
strrefrestep=Double.parseDouble(steprefractiveIndexTextBox.getText());
for(double j=strrefremin;j<=strrefremax;j=j+strrefrestep)
{
refre=j;
String strrefim=radiusImgRealTextBox.getText();
refim=Double.parseDouble(strrefim);
double real=1.0;
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
String strxk2 =evqRadiusTextBox.getText();
xk=(2.0*PI*Double.parseDouble(strxk2)*real)/Double.parseDouble(strxk1);
double axi=Double.parseDouble(strxk2);
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,eps,ddelt,-
1,-2,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
```

```
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.csca;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real) :"+valueGet+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+radiusImgRealTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"D/L :"+a_bOrC_LTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere
Radius:"+evqRadiusTextBox+"\n";
return;
}
```

```
if(spheroidRadioButton.isSelected())
{
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Accuracy of T-
Matrix Computation");
return;
}
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter A/B");
return;
}
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent radius");
return;
}
double
strrefremax=Double.parseDouble(maxrrefractiveIndexTextBox.getText());
double
strrefremin=Double.parseDouble(minrrefractiveIndexTextBox.getText());
double
strrefrestep=Double.parseDouble(steprefractiveIndexTextBox.getText());
double valueGet=0.0;
for(double j=strrefremin;j<=strrefremax;j=j+strrefrestep)
{
refre=j;
String strrefim=radiusImgRealTextBox.getText();
refim=Double.parseDouble(strrefim);
double real=1.0;
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
String strxk2 =evqRadiusTextBox.getText();
xk=(2.0*PI*Double.parseDouble(strxk2)*real)/Double.parseDouble(strxk1);
double axi=Double.parseDouble(strxk2);
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,eps,ddelt,-
1,-1,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
```

```
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.csca;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
 for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real) :"+valueGet+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+radiusImgRealTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"A/B :"+a_bOrC_LTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere
Radius:"+evqRadiusTextBox+"\n";
return;
}
}
if(imagindexRadioButton.isSelected())
```

```
{
if(null!=radiusImgRealTextBox&&(null==radiusImgRealTextBox.getText()||r
adiusImgRealTextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter Refractive
Index(Real)");
return;
}
if(null!=minirefractiveIndexTextBox&&(null==minirefractiveIndexTextBox.
getText()||minirefractiveIndexTextBox.getText().trim().equalsIgnoreCase
("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Min value for
Refractive Index(Imaginary)");
return;
}
if(null!=maxirefractiveIndexTextBox&&(null==maxirefractiveIndexTextBox.
getText()||maxirefractiveIndexTextBox.getText().trim().equalsIgnoreCase
("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Max value for
Refractive Index(Imaginary)");
return;
}
if(null!=steprefractiveIndexTextBox&&(null==steprefractiveIndexTextBox.
getText()||steprefractiveIndexTextBox.getText().trim().equalsIgnoreCase
("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Step value for
Refractive Index(Imaginary)");
return;
}
if(!sphereRadioButton.isSelected()&&!cyclinderRadioButton.isSelected()&
&!spheroidRadioButton.isSelected())
{
JOptionPane.showMessageDialog(null, "Enter Particle Geometry");
return;
}
if(sphereRadioButton.isSelected())
{
if(null!=radiusTextBox&&(null==radiusTextBox.getText()||radiusTextBox.g
etText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the value of the
radius");
return;
}
double valueGet=0.0;
double
strrefremax=Double.parseDouble(maxirefractiveIndexTextBox.getText());
double
strrefremin=Double.parseDouble(minirefractiveIndexTextBox.getText());
double
strrefrestep=Double.parseDouble(steprefractiveIndexTextBox.getText());
for(double j=strrefremin;j<=strrefremax;j=j+strrefrestep)
{
String strrefre=radiusImgRealTextBox.getText();
```

```
refre=Double.parseDouble(strrefre);
refim=j;
double real=1.0;
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
String strxk2 =radiusTextBox.getText();
xk=(2.0*PI*Double.parseDouble(strxk2)*real)/Double.parseDouble(strxk1);
main_Function();
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real) :"+valueGet+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary)
:"+radiusImgRealTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Radius::"+radiusTextBox.getText()+"\n";
return;
}
if(cyclinderRadioButton.isSelected())
{
```

```
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Accuracy of T-
Matrix Computation");
return;
}
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter D/L");
return;
}
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent radius");
return;
}
double
strrefremax=Double.parseDouble(maxirefractiveIndexTextBox.getText());
double
strrefremin=Double.parseDouble(minirefractiveIndexTextBox.getText());
double
strrefrestep=Double.parseDouble(steprefractiveIndexTextBox.getText());
double valueGet=0.0;
for(double j=strrefremin;j<=strrefremax;j=j+strrefrestep)
{
String strrefre=radiusImgRealTextBox.getText();
refre=Double.parseDouble(strrefre);
refim=j;
double real=1.0;
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
String strxk2 =evqRadiusTextBox.getText();
xk=(2.0*PI*Double.parseDouble(strxk2)*real)/Double.parseDouble(strxk1);
double axi=Double.parseDouble(strxk2);
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,eps,ddelt,-
1,-1,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
```

```
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.csca;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real) :"+
radiusImgRealTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary) :"+
valueGet+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"A/B :"+a_bOrC_LTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere
Radius:"+evqRadiusTextBox+"\n";
return;
}
if(spheroidRadioButton.isSelected())
{
```

```
if(null!=accuracyTextBox&&(null==accuracyTextBox.getText()||accuracyTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter the Accuracy of T-
Matrix Computation");
return;
}
if(null!=a_bOrC_LTextBox&&(null==a_bOrC_LTextBox.getText()||a_bOrC_LTex
tBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter A/B");
return;
}
if(null!=evqRadiusTextBox&&(null==evqRadiusTextBox.getText()||evqRadius
TextBox.getText().trim().equalsIgnoreCase("")))
{
JOptionPane.showMessageDialog(null, "Please Enter equivalent radius");
return;
}
double
strrefremax=Double.parseDouble(maxirefractiveIndexTextBox.getText());
double
strrefremin=Double.parseDouble(minirefractiveIndexTextBox.getText());
double
strrefrestep=Double.parseDouble(steprefractiveIndexTextBox.getText());
double valueGet=0.0;
for(double j=strrefremin;j<=strrefremax;j=j+strrefrestep)
{
String strrefre=radiusImgRealTextBox.getText();
refre=Double.parseDouble(strrefre);
refim=j;
double real=1.0;
refmed=new Complex(real,0.0);
refrelk=new Complex(refre,refim);
ntotk=1;
stepk=0.05;
String strxk1=waveLengthText.getText();
double lam=Double.parseDouble(strxk1);
String strxk2 =evqRadiusTextBox.getText();
xk=(2.0*PI*Double.parseDouble(strxk2)*real)/Double.parseDouble(strxk1);
double axi=Double.parseDouble(strxk2);
String streps=a_bOrC_LTextBox.getText();
double eps=Double.parseDouble(streps);
String strddelt=accuracyTextBox.getText();
double ddelt=Double.parseDouble(strddelt);
radlk=0.9999999*axi;
radhk=1.0000001*axi;
String
ret=nonSphericalClass.nonsp(refre,refim,1,axi,radlk,radhk,7,eps,ddelt,-
1,-1,lam);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
JOptionPane.showMessageDialog(null, ret);
return;
}
ss11=nonSphericalClass.ss11;
ss12=nonSphericalClass.ss12;
```

```
ss33=nonSphericalClass.ss33;
ss34=nonSphericalClass.ss34;
qexttxt=nonSphericalClass.cextin;
qscatxt=nonSphericalClass.csca;
qabstxt=nonSphericalClass.cabsin;
albedotxt=nonSphericalClass.walb;
gtxt=nonSphericalClass.asymm;
double sum=0;
for(int i=0;i<181;i++)
{
d[i]=(ss11Com[i]-ss11[i])*(ss11Com[i]-ss11[i]);
sum=sum+d[i];
}
double rmserror;
rmserror=Math.sqrt(sum/181);
if(errorMin==-1.0)
{
errorMin=rmserror;
valueGet=j;
}
else if(rmserror<errorMin)
{
errorMin=rmserror;
valueGet=j;
for(int i=0;i<181;i++)
{
ss11Min[i]=ss11[i];
ss12Min[i]=ss11[i];
ss33Min[i]=ss11[i];
ss34Min[i]=ss11[i];
}
}
}
JOptionPane.showMessageDialog(null, "Minimum Error is"+errorMin);
for(int i=0;i<181;i++)
{
ss11[i]=ss11Min[i];
ss11[i]=ss12Min[i];
ss11[i]=ss33Min[i];
ss11[i]=ss34Min[i];
}
estimatedPara=estimatedPara+"Wave Length Value
:"+waveLengthText.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Real) :"+
radiusImgRealTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Refractive Index(Imaginary) :"+
valueGet+"\n";
estimatedPara=estimatedPara+"Accuracy of T-Matrix Computation
:"+accuracyTextBox.getText()+"\n";
estimatedPara=estimatedPara+"A/B :"+a_bOrC_LTextBox.getText()+"\n";
estimatedPara=estimatedPara+"Equivalent Sphere
Radius:"+evqRadiusTextBox+"\n";
return;
}
}
}
}
```

```
});

buttonoverPlottedGraph.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolDataUpload)
{
boolCompare=false;
boolExp=false;
boolOverPlot=true;
boollogCompare=false;
 boollogExp=false;
 boollogOverPlot=false;
if(null==frame)
{
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
rb1.setSelected(true);
rb2.setSelected(false);
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new JScrollPane(new TUScatGUI()));
frame.setSize(1240, 700);
frame.setVisible(true);
}
else
{
frame.setVisible(false);
JPanel panel = new JPanel();
rb1.setSelected(true);
rb2.setSelected(false);
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new JScrollPane(new TUScatGUI()));
frame.setSize(1240, 700);
frame.setVisible(true);
}
}
else
{
JOptionPane.showMessageDialog(null, "Please Upload File");
return;
}
}
});
rb1 = new JRadioButton("Normal", true);
rb2 = new JRadioButton("Logarithmic");
ButtonGroup group = new ButtonGroup();
group.add(rb1); group.add(rb2);
rb1.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolExp)
{
boollogCompare=false;
```

```
boollogExp=false;
boollogOverPlot=false;
}
else if(boolCompare)
{
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
}
else if(boolOverPlot)
{
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
}
if(null==frame)
{
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
else
{
frame.setVisible(false);
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
}
});
rb2.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolExp)
{
boollogCompare=false;
boollogExp=true;
boollogOverPlot=false;
}
else if(boolCompare)
{
boollogCompare=true;
boollogExp=false;
boollogOverPlot=false;
}
else if(boolOverPlot)
```

```
{
boollogCompare=false;
boollogExp=false;
boollogOverPlot=true;
}
if(null==frame)
{
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
else
{
frame.setVisible(false);
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new TUScatGUI());
frame.setSize(1240, 720);
frame.setVisible(true);
}
}
});

buttonPlottedGraph.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent ae){
if(boolDataUpload)
{
boolCompare=true;
boolExp=false;
boolOverPlot=false;
boollogCompare=false;
boollogExp=false;
boollogOverPlot=false;
if(null==frame)
{
frame= new JFrame("TUSCAT: Plotting Window");
JPanel panel = new JPanel();
rb1.setSelected(true);
rb2.setSelected(false);
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new JScrollPane(new TUScatGUI())),
frame.setSize(1240, 700);
frame.setVisible(true);
}
```

```
else
{
frame.setVisible(false);
JPanel panel = new JPanel();
rb1.setSelected(true);
rb2.setSelected(false);
panel.add(rb1);
panel.add(rb2);
frame.add(panel);
frame.pack();
frame.add(new JScrollPane(new TUScatGUI()));
frame.setSize(1240, 700);
frame.setVisible(true);
}
}
else
{
JOptionPane.showMessageDialog(null, "Please Upload File");
return;
}
}
});

WindowListener wndCloser = new WindowAdapter() {
public void windowClosing(WindowEvent e) {
System.exit(0);
}
};

f2.addWindowListener(wndCloser);
f2.add(p);
f2.show();
}

 public static void main(String[] args) {

String[] choices = { "Select","Theoritical Data Analysis",
"Experimental Data Analysis"};
input = (String) JOptionPane.showInputDialog(null, "Choice your
Option",
"Choice your Option", JOptionPane.QUESTION_MESSAGE, null, // Use
choices, // Array of choices
choices[1]); // Initial choice
if(input!=null&&input.equalsIgnoreCase("Theoritical Data Analysis"))
{
mainFrameTheory();
}
else if(input!=null&&input.equalsIgnoreCase("Experimental Data
Analysis"))
{
mainFrameExp();
}
}

 protected void paintComponent(Graphics g) {
super.paintComponent(g);
Graphics2D g2 = (Graphics2D)g;
```

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
Stroke drawingStroke = new BasicStroke(1, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_BEVEL, 0, new float[]{3}, 0);
DecimalFormat df = new DecimalFormat ("0.#");
double t=0.038;
double tt=0.004;
double ss1190=0.00;
double ss11Exp[]=new double[181];
double ss11ComExp[]=new double[181];
g2.draw(new Line2D.Double(0, 350, 1240, 350));
g2.draw(new Line2D.Double(600, 0, 600, 720));
g.drawRect(50, 50, 540, 250);
g.drawRect(650, 50, 540, 250);
g.drawRect(650, 400, 540, 250);
g.drawRect(50, 400, 540, 250);
g.setColor(Color.BLACK);
int count=0;
for(int i=30;i<=540;i=i+30)
{
count++;
g.drawString(String.valueOf(count*10), 35+i,320 );
}
float doubleValue=0.0f;
g.drawString("0", 47,320 );
g.drawString("0.0", 25,305 );
g.drawString("Scattering Angle(degree)", 260,340 );
g.drawString("Scattering Angle(degree)",850,690 );
g.drawString("Scattering Angle(degree)",850,340 );
g.drawString("Scattering Angle(degree)", 260,690 );
g.drawString("S11 versus scattering angle", 240,30);
g.drawString("-S12/S11 versus scattering angle", 830,30);
g.drawString("S34/S11 versus scattering angle", 240,380);
g.drawString("S33/S11 versus scattering angle", 835,380);
count=0;
for(int i=30;i<=540;i=i+30)
{
count++;
g.drawString(String.valueOf(count*10), 635+i,320 );
}
doubleValue=-1.0f;
for(int i=25;i<=250;i=i+25)
{
doubleValue=doubleValue+0.2f;
int y=(int)(doubleValue*10);
float z= (float)y/10;
if(z>0.0)
{
z=z+0.1f;
}
g.drawString(String.valueOf(z),625,305-i );
}
g.drawString("0", 647,320 );
g.drawString("-1.0", 625,305 );
count=0;
for(int i=30;i<=540;i=i+30)
{
```

```
count++;
g.drawString(String.valueOf(count*10), 635+i,670 );
}
doubleValue=-1.0f;
for(int i=25;i<=250;i=i+25)
{
doubleValue=doubleValue+0.2f;
int y=(int)(doubleValue*10);
float z= (float)y/10;
if(z>0.0)
{
z=z+0.1f;
}
g.drawString(String.valueOf(z),625,655-i );
}
g.drawString("0", 647,670 );
g.drawString("-1.0", 625,655 );
count=0;
for(int i=30;i<=540;i=i+30)
{
count++;
g.drawString(String.valueOf(count*10), 35+i,670 );
}
doubleValue=-1.0f;
for(int i=25;i<=250;i=i+25)
{
doubleValue=doubleValue+0.2f;
int y=(int)(doubleValue*10);
float z= (float)y/10;
if(z>0.0)
{
z=z+0.1f;
}
g.drawString(String.valueOf(z),25,655-i );
}
g.drawString("0", 47,670 );
g.drawString("-1.0", 25,655 );
for(int i=0;i<181;i++)
{
ss11Exp[i]=ss11[i];
}
for(int i=0;i<181;i++)
{
ss11ComExp[i]=ss11Com[i];
}
double maxThValue=0.0;
if(boolExp==true)
{
g.setColor(Color.RED);
ss1190=ss11[31];
if(boollogExp==true)
{
for(int i=0;i<181;i++)
{
ss11Exp[i]=ss11Exp[i]/ss1190;
ss11Exp[i]=Math.log10(ss11Exp[i]);
if(maxThValue<Math.abs(ss11Exp[i]))
```

```
{
maxThValue=Math.abs(ss11Exp[i]);
}
}
doubleValue=0.0f;
g.setColor(Color.BLACK);
for(int i=25;i<=250;i=i+25)
{
doubleValue++;
g.drawString(df.format((doubleValue/10)*maxThValue), 25,305-i );
}
g.setColor(Color.RED);
for(int i=0;i<181;i++)
{
ss11Exp[i]= ss11Exp[i]/maxThValue;
}
double factTh=(maxThValue/250);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor((1+ss11Exp[i])/(.008));
int yy1=(int)Math.floor(((1+ss11Exp[i+1])/(.008)));
g.drawLine((50+i*3),300-yy,(50+(i+1)*3),300-yy1);
}
}
else
{
for(int i=0;i<180;i++)
{
ss11Exp[i]=ss11Exp[i]/ss1190;
if(maxThValue<ss11Exp[i])
{
maxThValue=ss11Exp[i];
}
}
doubleValue=0.0f;
g.setColor(Color.BLACK);
for(int i=25;i<=250;i=i+25)
{
doubleValue++;
g.drawString(df.format((doubleValue/10)*maxThValue), 25,305-i );
}
g.setColor(Color.RED);
double factTh=(maxThValue/250);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor((ss11Exp[i]/(factTh)));
int yy1=(int)Math.floor((ss11Exp[i+1]/(factTh)));
g.drawLine((50+i*3),300-yy,(50+(i+1)*3),300-yy1);
}
}
g.setColor(Color.BLACK);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor(((1+ss12[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss12[i+1])/(0.008)));
g.drawLine((650+i*3),300-yy,(650+(i+1)*3),300-yy1);
}
```

```
g.setColor(Color.BLACK);
g.setColor(Color.RED);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor(((1+ss33[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss33[i+1])/(0.008)));
g.drawLine((650+i*3),650-yy,(650+(i+1)*3),650-yy1);
}
g.setColor(Color.BLACK);
g.setColor(Color.RED);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor(((1+ss34[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss34[i+1])/(0.008)));
g.drawLine((50+i*3),650-yy,(50+(i+1)*3),650-yy1);
}
}
if(boolCompare==true)
{
g.setColor(Color.BLUE);
double ss11Com90=0.00;
int max=(int)(angle[1]-angle[0]);
int init=(180/max)+1;
int j=(int)(angle[0]);
double maxValue;
int minang=(int)angle[0];
int maxang=(int)angle[0];
for(int i=0;i<init;i=i+1)
{
if(maxang<(int)angle[i])
 {
 maxang=(int)angle[i];
 }
if(angle[i]==30.0)
{
ss11Com90=ss11Com[i];
}
}
int initt=(int)((maxang-j)/max);
maxValue=0.00;
if(boollogCompare)
{
for(int i=0;i<=initt;i=i+1)
{
ss11ComExp[i]=ss11ComExp[i]/ss11Com90;
ss11ComExp[i]=Math.log10(ss11ComExp[i]);
if(maxValue<Math.abs(ss11ComExp[i]))
{
maxValue=Math.abs(ss11ComExp[i]);
}
}
doubleValue=0.0f;
g.setColor(Color.BLACK);
for(int i=25;i<=250;i=i+25)
{
doubleValue++;
g.drawString(df.format((doubleValue/10)*maxValue), 25,305-i );
```

```
}
g.setColor(Color.RED);
for(int i=0;i<=initt;i=i+1)
{
ss11ComExp[i]=ss11ComExp[i]/maxValue;
}
if(ss1190==0.00&&ss11Com90==0.00)
{
if(null!=frame)
{
frame.setVisible(false);
}
JOptionPane.showMessageDialog(null, "Unable plot comparative graph
(experimental s11=0.0 at 90 degree)");
return;
}
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss11ComExp[i])/(.008)));
int yy1=(int)Math.floor(((1+ss11ComExp[i+1])/(.008)));
g.drawLine((50+j*3),300-yy,(50+(j+max)*3),300-yy1);
j=j+max;
}
}
else
{
for(int i=0;i<initt;i=i+1)
{
ss11ComExp[i]=ss11ComExp[i]/ss11Com90;
if(maxValue<ss11ComExp[i])
{
maxValue=ss11ComExp[i];
}
}
doubleValue=0.0f;
g.setColor(Color.BLACK);
for(int i=25;i<=250;i=i+25)
{
doubleValue++;
g.drawString(df.format((doubleValue/10)*maxValue), 25,305-i );
}
g.setColor(Color.RED);
double realFact=0.0;
if(maxThValue==0.0)
{
realFact=maxValue/250;
}
else
{
realFact=(Math.max(maxThValue, maxValue))/250;
}
if(ss1190==0.00&&ss11Com90==0.00)
{
if(null!=frame)
{
frame.setVisible(false);
}
```

```
JOptionPane.showMessageDialog(null, "Unable plot comparative graph
(experimental s11=0.0 at 90 degree)");
return;
}
for(int i=0;i<(initt-1);i++)
{
int yy=(int)Math.floor((ss11ComExp[i]/(realFact)));
int yy1=(int)Math.floor((ss11ComExp[i+1]/(realFact)));
g.drawLine((50+j*3),300-yy,(50+(j+max)*3),300-yy1);
j=j+max;
}
}
j=(int)(angle[0]);
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss12Com[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss12Com[i+1])/(0.008)));
g.drawLine((650+j*3),300-yy,(650+(j+max)*3),300-yy1);
j=j+max;
}
j=(int)(angle[0]);
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss33Com[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss33Com[i+1])/(0.008)));
g.drawLine((650+j*3),650-yy,(650+(j+max)*3),650-yy1);
j=j+max;
}
j=(int)(angle[0]);
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss34Com[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss34Com[i+1])/(0.008)));
g.drawLine((50+j*3),650-yy,(50+(j+max)*3),650-yy1);
j=j+max;
}
}
if(boolOverPlot)
{
g.setColor(Color.RED);
ss1190=ss11[31];
maxThValue=0.00;
double factTh=(maxThValue/250);
double ss11Com90=0.00;
int max=(int)(angle[1]-angle[0]);
int init=(180/max)+1;
int j=(int)(angle[0]);
double maxValue
;
int minang=(int)angle[0];
int maxang=(int)angle[0];
for(int i=0;i<init;i=i+1)
{
if(maxang<(int)angle[i])
{
maxang=(int)angle[i];
}
```

```
if(angle[i]==30.0)
{
ss11Com90=ss11ComExp[i];
}
}
int initt=(int)((maxang-j)/max);
maxValue=0.0;
double realFact=0.0;
if(boollogOverPlot)
{
for(int i=0;i<181;i++)
{
ss11Exp[i]=ss11Exp[i]/ss1190;
ss11Exp[i]=Math.log10(ss11Exp[i]);
if(maxThValue<Math.abs(ss11Exp[i]))
{
maxThValue=Math.abs(ss11Exp[i]);
}
}
for(int i=0;i<=initt;i=i+1)
{
ss11ComExp[i]=ss11ComExp[i]/ss11Com90;
ss11ComExp[i]=Math.log10(ss11ComExp[i]);
if(maxValue<Math.abs(ss11ComExp[i]))
{
maxValue=Math.abs(ss11ComExp[i]);
}
}
realFact=(Math.max(maxThValue, maxValue));
doubleValue=0.0f;
g.setColor(Color.BLACK);
for(int i=25;i<=250;i=i+25)
{
doubleValue++;
g.drawString(df.format((doubleValue/10)*realFact), 25,305-i );
}
g.setColor(Color.RED);
for(int i=0;i<181;i++)
{
ss11Exp[i]=ss11Exp[i]/realFact;
}
for(int i=0;i<=initt;i=i+1)
{
ss11ComExp[i]=ss11ComExp[i]/realFact;
}
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor(((1+ss11Exp[i])/(.008)));
int yy1=(int)Math.floor(((1+ss11Exp[i+1])/(.008)));
g.drawLine((50+i*3),300-yy,(50+(i+1)*3),300-yy1);
}
}
else
{
for(int i=0;i<181;i++)
{
ss11Exp[i]=ss11Exp[i]/ss1190;
```

```
if(maxThValue<ss11Exp[i])
{
maxThValue=ss11Exp[i];
}
}
for(int i=0;i<=initt;i=i+1)
{
ss11ComExp[i]=ss11ComExp[i]/ss11Com90;
if(maxValue<ss11ComExp[i])
{
maxValue=ss11ComExp[i];
}
}
doubleValue=0.0f;
g.setColor(Color.BLACK);
for(int i=25;i<=250;i=i+25)
{
doubleValue++;
g.drawString(df.format((doubleValue/10)*Math.max(maxThValue,
maxValue)), 25,305-i );
}
g.setColor(Color.RED);
realFact=(Math.max(maxThValue, maxValue))/250;
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor((ss11Exp[i]/(realFact)));
int yy1=(int)Math.floor((ss11Exp[i+1]/(realFact)));
g.drawLine((50+i*3),300-yy,(50+(i+1)*3),300-yy1);
}
}
g.setColor(Color.BLACK);
g.setColor(Color.RED);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor(((1+ss12[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss12[i+1])/(0.008)));
g.drawLine((650+i*3),300-yy,(650+(i+1)*3),300-yy1);
}
g.setColor(Color.BLACK);
g.setColor(Color.RED);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor(((1+ss33[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss33[i+1])/(0.008)));
g.drawLine((650+i*3),650-yy,(650+(i+1)*3),650-yy1);
}
g.setColor(Color.BLACK);
g.setColor(Color.RED);
for(int i=0;i<180;i++)
{
int yy=(int)Math.floor(((1+ss34[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss34[i+1])/(0.008)));
g.drawLine((50+i*3),650-yy,(50+(i+1)*3),650-yy1);
}
g.setColor(Color.BLUE);
if(ss1190==0.00&&ss11Com90==0.00)
{
```

```
if(null!=frame)
{
frame.setVisible(false);
}
JOptionPane.showMessageDialog(null, "Unable plot comparative graph
(experimental s11=0.0 at 90 degree)");
return;
}
if(boollogOverPlot)
{
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss11ComExp[i])/(.008)));
int yy1=(int)Math.floor(((1+ss11ComExp[i+1])/(.008)));
g.drawLine((50+j*3),300-yy,(50+(j+max)*3),300-yy1);
j=j+max;
}
}
else
{
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor((ss11ComExp[i]/(realFact)));
int yy1=(int)Math.floor((ss11ComExp[i+1]/(realFact)));
g.drawLine((50+j*3),300-yy,(50+(j+max)*3),300-yy1);
j=j+max;
}
}
j=(int)(angle[0]);
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss12Com[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss12Com[i+1])/(0.008)));
g.drawLine((650+j*3),300-yy,(650+(j+max)*3),300-yy1);
j=j+max;
}
j=(int)(angle[0]);
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss33Com[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss33Com[i+1])/(0.008)));
g.drawLine((650+j*3),650-yy,(650+(j+max)*3),650-yy1);
j=j+max;
}
j=(int)(angle[0]);
for(int i=0;i<initt;i++)
{
int yy=(int)Math.floor(((1+ss34Com[i])/(0.008)));
int yy1=(int)Math.floor(((1+ss34Com[i+1])/(0.008)));
g.drawLine((50+j*3),650-yy,(50+(j+max)*3),650-yy1);
j=j+max;
}
}
}

 public static void main_Function()
{
```

```
int count=0;
refrelk=new Complex(refre,refim);
refrelk=refrelk.div(refmed);
dang=(PI/2.0)/(double)(nangk-1);
bhmie();
gtxt=gscatxt/qscatxt;
qprtxt=qexttxt-gscatxt;
qabstxt=qexttxt-qscatxt;
albedotxt=qscatxt/qexttxt;
s11nor=(0.5)*(Math.pow(s2k[1].mod(),2.0)+Math.pow(s1k[1].mod(),2.0));
nan=2*nangk-1;
for(j=1;j<=nan;j++)
{
s11=(0.5)*(Math.pow(s2k[j].mod(),2.0)+Math.pow(s1k[j].mod(),2.0));
s12=(0.5)*(Math.pow(s2k[j].mod(),2.0)-Math.pow(s1k[j].mod(),2.0));
pol=-s12/s11;
s33=(s2k[j].times(s1k[j].conj())).real();
s33=s33/s11;
s34=(s2k[j].times(s1k[j].conj())).imag();
s34=s34/s11;
s11=s11/s11nor;
ang=dang*(j-1.0)*(180.0/PI);
ss11[count]=s11;
ss12[count]=pol;
ss33[count]=s33;
ss34[count]=s34;
count++;
}
}

/*********************************************************************/
/***************BHMIE function for Mie calculations**************/

public static void bhmie()
{
double amu[]=new  double[100];
double theta[]=new  double[100];
double pi[]=new  double[100];
double tau[]=new  double[100];
double pi0[]=new  double[100];
double pi1[]=new  double[100];
Complex y,xi,xi1,ann,bnn,an,bn,an1,an2,comp1,comp2,comp3,comp4,comp5;
Complex d[]=new Complex[3000];
double
psi0,psi1,psi,dn,dx,xstop,nstop,ymod,dang,rn,chi,chi0,chi1,apsi1,
apsi,fn,p,t,qext,qsca,qback,qabs,albedo;
int nn,n,nmax,j,jj;
dx=xk;
an1=new Complex(xk,0.0);
ann=new Complex(0.0,0.0);
bnn=new Complex(0.0,0.0);
an=new Complex(0.0,0.0);
bn=new Complex(0.0,0.0);
y=an1.times(refrelk);
gscatxt=0.0;
if(xk<8)
{
```

```
xstop=xk+(4.0)*Math.pow(xk,0.333)+1.0;
}
else if(xk<=4200)
{
xstop=xk+(4.5)*Math.pow(xk,0.333)+2.0;
}
else
{
xstop=xk+(4.0)*Math.pow(xk,0.333)+2.0;
}
nstop=xstop;
ymod=y.mod();
nmax=(int)(Math.max(xstop,ymod))+15;
dang=(PI/2.0)/(double)(nangk-1);
for(j=1;j<=nangk;j++)
{
theta[j]=((double)(j)-1.0)*dang;
amu[j]=Math.cos(theta[j]);
}
d[nmax]=new Complex(0.0,0.0);
nn=nmax-1;
for(n=1;n<=nn;n=n+1)
{
rn=(double)(nmax-n+1);
an1=new Complex(rn,0.0);
an2=new Complex(1.0,0.0);
d[nmax-n]=an1.div(y).minus(an2.div(d[nmax-n+1].plus(an1.div(y))));
}
for(j=1;j<=nangk;j++)
{
pi0[j]=0.0;
pi1[j]=1.0;
}
nn=2*nangk-1;
for(j=1;j<=nn;j++)
{
s1k[j]=new Complex(0.0,0.0);
s2k[j]=new Complex(0.0,0.0);
}
psi0= Math.cos(dx);
psi1=Math.sin(dx);
chi0= -(Math.sin(xk));
chi1=Math.cos(xk);
apsi1 =psi1;
xi1= new Complex(apsi1,-chi1);
qsca=0.0;
n=1;
boolean flag=true;
while(flag)
{
dn=(double)(n);
rn=(double)(n);
fn=(2.0*rn+1.0)/(rn*(rn+1.0));
psi=(2.0*dn-1.0)*psi1/dx-psi0;
apsi=psi;
chi=(2.0*rn-1.0)*chi1/(xk)-chi0;
xi=new Complex(apsi,-chi);
```

```
an1=d[n].div(refrelk);
comp1=new Complex(rn/(xk),0.0);
comp2=new Complex(apsi,0.0);
comp3=new Complex(apsi1,0.0);
if(n>1)
{
ann=an;
bnn=bn;
}
an1=an1.plus(comp1);
an1=an1.times(comp2);
an1=an1.minus(comp3);
an2=d[n].div(refrelk);
an2=an2.plus(comp1);
an2=an2.times(xi);
an2=an2.minus(xi1);
an=an1.div(an2);
an1=refrelk.times(d[n]);
an1=an1.plus(comp1);
an1=an1.times(comp2);
an1=an1.minus(comp3);
an2=refrelk.times(d[n]);
an2=an2.plus(comp1);
an2=an2.times(xi);
an2=an2.minus(xi1);
bn=an1.div(an2);
qsca=qsca+(2.0*rn+1.0)*(Math.pow(an.mod(),2)+Math.pow(bn.mod(),2));
gscatxt=gscatxt+(2.0*rn+1.0)*((an.times(bn.conj())).real())/((rn+1.0)*r
n);
if(n>1)
{
gscatxt=gscatxt+(rn-
1.0)*(rn+1.0)*(ann.times(an.conj()).plus(bnn.times(bn.conj()))).real()/
rn;
}
for(j=1;j<=nangk;j++)
{
jj=2*nangk-j;
pi[j]=pi1[j];
tau[j]=rn*amu[j]*pi[j]-(rn+1.0)*pi0[j];
p=Math.pow(-1.0,(double)(n-1));
comp1=new Complex(pi[j],0.0);
comp2=new Complex(tau[j],0.0);
comp3=new Complex(fn,0.0);
an1=an.times(comp1);
an2=bn.times(comp2);
an1=comp3.times(an1.plus(an2));
s1k[j]=s1k[j].plus(an1);
t=Math.pow(-1.0,(double)(n));
an1=an.times(comp2);
an2=bn.times(comp1);
an1=comp3.times(an1.plus(an2));
s2k[j]=s2k[j].plus(an1);
if(j!=jj)
{
comp1=new Complex(pi[j],0.0);
comp2=new Complex(tau[j],0.0);
```

```
comp3=new Complex(fn,0.0);
comp4=new Complex(p,0.0);
comp5=new Complex(t,0.0);
s1k[jj]=s1k[jj].plus(comp3.times(an.times(comp1.times(comp4)).plus(bn.t
imes(comp2.times(comp5)))));
s2k[jj]=s2k[jj].plus(comp3.times(an.times(comp2.times(comp5)).plus(bn.t
imes(comp1.times(comp4)))));
}
}
psi0=psi1;
psi1=psi;
apsi1=psi1;
chi0=chi1;
chi1=chi;
xi1=new Complex(apsi1,-chi1);
n=n+1;
rn=(double)(n);
for(j=1;j<=nangk;j++)
{
pi1[j]=((2.0*rn-1.0)/(rn-1.0))*amu[j]*pi[j];
pi1[j]=pi1[j]-rn*pi0[j]/(rn-1.0);
pi0[j]=pi[j];
}
if((n-1-nstop)<0)
{
flag=true;
}
else
{
qsca=(2.0/Math.pow(xk,2))*qsca;
qext=(4.0/Math.pow(xk,2))*s1k[1].real();
qback=(4.0/Math.pow(xk,2))*Math.pow(s1k[2*(nangk)-1].mod(),2);
gscatxt=((4.0/Math.pow(xk, 2))*gscatxt);
qscatxt=qsca;
qexttxt=qext;
qbacktxt=qback;
flag=false;
}
}
return;
}

/*Size distribution functions*/

static double dstn(int ntot, double radl, double radh, double rad,
double step, int dstn, double rc,
double alfa, double sigma, double rg)
{
double l,a,b,gama=1.0,nr,rad1,rad2,rad3,rad4,nrn;
if(dstn==1)
{
b=alfa/rc;
for(l=alfa;l>=1.0e+0;l=l-1.0e+0)
gama= gama*l;
a=ntot/(Math.pow(b,-(alfa+1.0))*gama);
nr=a*Math.pow(rad,alfa)*Math.exp(-b*rad);
}
```

```
else if(dstn==2)
{
rad1=rad-rg;
rad2=Math.pow(rad1,2.0);
rad3=rad2/(2*Math.pow(sigma,2));
rad4=1/Math.exp(rad3);
nrn=rad4/(Math.pow(2.0*PI,0.5)*sigma);
nr=ntot*nrn;
}
else
{
rad1=Math.log(rad/rg);
rad2=Math.pow(rad1,2.0);
rad3=rad2/(2*Math.pow(sigma,2));
rad4=1/Math.exp(rad3);
nrn=rad4/(rad*(Math.pow(2.0*PI,0.5)*sigma));
nr=ntot*nrn;
}
return(nr);
}

/*Light scattering calculations on spherical particles*/

public static void extra(double stepk,double radlk,double radhk,int
ntotk,double rck,double alfak,int dstnk)
{
double s11tot[]=new double[200];
double s12tot[]=new double[200];
double s33tot[]=new double[200];
double s34tot[]=new double[200];
double s11ang[]=new double[200];
for(int i=0;i<s11tot.length;i++)
{
s11tot[i]=0.0;
s12tot[i]=0.0;
s33tot[i]=0.0;
s34tot[i]=0.0;
}
double sigmak=alfak;
double rgk=rck;
double ext=0.0;
double sca=0.0;
double back=0.0;
double gr=0.0;
double gsc=0.0;
xk=0.0;
for(double radk=radlk;radk<=radhk;radk=radk+stepk)
{
double
nr=dstn(ntotk,radlk,radhk,radk,stepk,dstnk,rck,alfak,sigmak,rgk);
nr=stepk*nr;
xk=2.0*PI*radk*(refmed.real())/wavel;
dang=(PI/2.0)/(double)(nangk-1);
bhmie();
gr=gr+PI*Math.pow(radk,2)*nr;
sca=sca+PI*Math.pow(radk,2)*nr*qscatxt;
ext=ext+PI*Math.pow(radk,2)*nr*qexttxt;
```

```
back=back+PI*Math.pow(radk,2)*nr*qbacktxt;
gsc=gsc+PI*Math.pow(radk,2)*nr*gscatxt;
qscatxt=sca/gr;
qexttxt=ext/gr;
qbacktxt=back/gr;
gscatxt=gsc/gr;
gtxt=gscatxt/qscatxt;
qprtxt=qexttxt-gscatxt;
qabstxt=qexttxt-qscatxt;
albedotxt=qscatxt/qexttxt;
nan=2*nangk-1;
for(j=1;j<=nan;j++)
{
s11=0.5e+0*(Math.pow(s2k[j].mod(),2.0)+Math.pow(s1k[j].mod(),2.0));
s11tot[j]=s11tot[j]+s11*nr;
s12=(0.5e+0)*(Math.pow(s2k[j].mod(),2.0)-Math.pow((s1k[j].mod()),2.0));
s12tot[j]=s12tot[j]+s12*nr;
s33=(s2k[j].times(s1k[j].conj())).real();
s33tot[j]=s33tot[j]+s33*nr;
s34=(s2k[j].times((s1k[j].conj()))).imag();
s34tot[j]=s34tot[j]+s34*nr;
}
}
int count=0;
for(j=1;j<=nan;j++)
{
ang=dang*((double)j-1.0e+0)*(180.0e+0/PI);
ss11[count]=s11tot[j]/s11tot[1];
ss12[count]=-s12tot[j]/s11tot[j];
ss33[count]=s33tot[j]/s11tot[j];
ss34[count]=s34tot[j]/s11tot[j];
count++;
}
}

public Dimension getPreferredSize(){
return new Dimension(1240,720);
}
}
```

/*******End of TUScatGUI.java*******/

B. Source code of nonSphericalClass.java

```
public class nonSphericalClass {
public static final int npn1= 100;
public static final int npng1= 300;
public static final int npng2 =2*npng1;
public static final int npn2=2*npn1;
public static final int np1 =npn2+1;
public static final int npn3 =npn1+1;
public static final int npn4= 80;
public static final int npn5= 2*npn4;
```

```
public static final int npn6= npn4+1;
public static final int npl1= npn5+1;
static int ichoice;
static double ss11[]=new double[181];
static double ss12[]=new double[181];
static double ss33[]=new double[181];
static double ss34[]=new double[181];
static double trgqr[][] = new double[npn2][npn2];
static double trgqi[][] = new double[npn2][npn2];
static double qr[][] = new double[npn2][npn2];
static double qi[][] = new double[npn2][npn2];
static double rgqr[][] = new double[npn2][npn2];
static double rgqi[][] = new double[npn2][npn2];
static double ssign[]=new double[900];
static double aa,bb,a,b;
static double ppi;
static double pir;
static double pii;
static int lmax;
static double rat;
static double reff;
static double veff;
static double cond;
static double csca;
static double cextin;
static double cscat;
static double asymm;
static double cabsin;
static double walb;
public static String nonsp(double imrr,double imri,int indistr,double
iaxi,
double ir1,double ir2 ,double ib,double ieps,double iddelt,int
nkmax,int np,double lam) {
String  ret="";
int i;
int l;
int m;
int n;
int l1;
int m1;
int n1;
int n2;
int n11;
int ii;
int n22;
int nk;
int nm;
int l1m;
int nm1;
int nn1;
int nn2;
int nma;
int iax;
int ink;
int nnm;
int inm1;
int nggg;
```

```
int ndgs;
int npna;
int nmin;
int mmax;
int nmax=0;
int ixxx;
int l1max;
int nmax1=0;
int itime;
int ngaus;
int npnax;
int ncheck;
int nnngggg;
int ngauss=0;
int ndistr;
double p;
double r1;
double r2;
double z1;
double z2;
double z3;
double zz1;
double zz2;
double zz3;
double zz4;
double zz5;
double zz6;
double zz7;
double zz8;
double gam;
double dax;
double ax1;
double mri;
double eps;
double mrr;
double xev;
double qsc;
double wgi;
double qxt;
double dsca;
double dn1;
double qsca;
double time;
double wgii;
double dext;
double cext;
double qext;
double qsca1;
double ti1nn;
double qext1;
double tr1nn;
double ddelt;
double dqsca;
double ti1nn1;
double axmax;
double tr1nn1;
double dqext;
```

```
double coeff1;
double wg[]=new double [1000];
double xg[]=new double [1000];
double wg1[]=new double [2000];
double xg1[]=new double [2000];
double r[]=new double[npng2] ;
double s[]=new double[npng2];
double w[]=new double[npng2];
double x[]=new double[npng2];
double an[]=new double[npn1];
double dr[]=new double[npng2] ;
double ss[]=new double[npng2] ;
double be1[]=new double[npl] ;
double be2[]=new double[npl] ;
double al1[]=new double[npl] ;
double al2[]=new double[npl] ;
double al3[]=new double[npl] ;
double al4[]=new double[npl] ;
double ddr[]=new double[npng2] ;
double dri[]=new double[npng2] ;
double drr[]=new double[npng2] ;
double bet1[]=new double[npl] ;
double bet2[]=new double[npl] ;
double alph1[]=new double[npl] ;
double alph2[]=new double[npl] ;
double alph3[]=new double[npl] ;
double alph4[]=new double[npl] ;
double ann[][]=new double[npn1][npn1];

nonSphericalClass nonSphericalClass=new nonSphericalClass();
mrr = imrr;
mri =imri;
ndistr=indistr;
axi = iaxi;
r1=ir1;
r2=ir2;
b = ib;
eps =ieps;
p=Math.acos(-1);
rat=0.5;
npnax = 1;
gam = .5;
ddelt = iddelt;
npna = 181;
ndgs = 2;
ichoice = 2;
ncheck = 0;
if (np == -1 || np == -2)
{
ncheck = 1;
}
if (np > 0 && Math.pow((-1),np) == 1)
{
ncheck = 1;
}
if ((Math.abs(rat - 1.0)) > 0.00000001 && np == -1)
{
```

```
nonSphericalClass.sarea(eps);
}
if ((Math.abs(rat - 1.0)) > 0.00000001 && np == -2)
{
nonSphericalClass.sareac(eps);
}
ddelt = 0.1*ddelt;
tmatBlock tmatblock=new tmatBlock(npng2,npn1);
trtiBlock trtiblock=new trtiBlock(npn2);
rirgigBlock rirgigblock=new rirgigBlock(npn1);
cbassBlock cbassblock=new cbassBlock(npng2,npn1);
nk = (int) (nkmax + 2);
if (nk > 1000)
{
ret="NK , I.E. IS GREATER THAN 1000 : EXECUTION TERMINATED";
return ret;
}
nonSphericalClass.gauss(nk,0,0, xg, wg);
z1 = (r2 - r1) * 0.5;
z2 = (r1 + r2) * 0.5;
z3 = r1 * 0.5;
for (i= 1; i<=nk; i++)
{
xg1[i-1] = z1 * xg[i-1] + z2;
wg1[i-1] = wg[i-1] * z1;
}
nonSphericalClass.distrb(nk, xg1, wg1, ndistr, axi, b, gam, r1, r2, p);
for (i=1;i<=npl;i++)
{
alph1[i - 1] = 0.;
alph2[i - 1] = 0.;
alph3[i - 1] = 0.;
alph4[i - 1] = 0.;
bet1[i - 1] = 0.;
bet2[i - 1] = 0.;
}
cscat = 0.;
cextin = 0.;
l1max = 0;
for (ink = 1; ink <= nk; ink++) {
i = nk - ink + 1;
a = rat * xg1[i - 1];
xev = p * 2. * a / lam;
ixxx =(int)(xev + Math.pow(xev,0.333333)*4.05);
inm1 = Math.max(4,ixxx);
if (inm1 >= npn1)
{
ret="CONVERGENCE IS NOT OBTAINED FOR NPN1  : EXECUTION
TERMINATED"+npn1;
return ret;
}
qext1 = 0.0;
qsca1 = 0.0;
for (nma = inm1; nma <= npn1; nma++)
{
nmax = nma;
mmax = 1;
```

```
ngauss = nmax * ndgs;
if (ngauss > npng1)
{
ret="NGAUSS =  I.E. IS GREATER THAN NPNG1 : EXECUTION TERMINATED";
return ret;
}
nonSphericalClass.constt(ngauss, nmax, mmax, p, x, w, an, ann, s, ss,
np, eps);
ret=nonSphericalClass.vary(lam, mrr, mri, a, eps, np, ngauss, x, p, r,
dr, ddr, drr, dri, nmax,cbassblock);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
return ret;
}
nonSphericalClass.tmatr0(ngauss, x, w, an, ann, s, ss, ppi, pir, pii,
r, dr, ddr, drr, dri, nmax,
ncheck,tmatblock,trtiblock,rirgigblock,cbassblock);
qext = 0.0;
qsca = 0.0;
for (n = 1; n <= nmax; n++)
{
n1 = n + nmax;
tr1nn =trtiblock.tr1[n-1][n-1];
ti1nn = trtiblock.ti1[n-1][n-1];
tr1nn1 = trtiblock.tr1[n1-1][n1-1];
ti1nn1 = trtiblock.ti1[n1-1][n1-1];
dn1 = (double) (2*n + 1);
qsca = qsca + dn1 * (tr1nn * tr1nn + ti1nn * ti1nn + tr1nn1 * tr1nn1 +
ti1nn1 * ti1nn1);
qext = qext + (tr1nn + tr1nn1) * dn1;
}
dsca = (Math.abs((qsca1 - qsca) / qsca));
dext = (Math.abs((qext1 - qext) / qext));
qext1 = qext;
qsca1 = qsca;
nmin = (int) ((double) nmax / 2.0 + 1.);
for (n = nmin; n <= nmax; ++n)
{
n1 = n + nmax;
tr1nn = trtiblock.tr1[n-1][n-1];
ti1nn = trtiblock.ti1[n-1][n-1];
tr1nn1 = trtiblock.tr1[n1-1][n1-1];
ti1nn1 = trtiblock.ti1[n1-1][n1-1];
dn1 = (double) (2*n + 1);
dqsca = dn1 * (tr1nn * tr1nn + ti1nn * ti1nn + tr1nn1 * tr1nn1 + ti1nn1
* ti1nn1);
dqext = (tr1nn + tr1nn1) * dn1;
dqsca = (Math.abs(dqsca / qsca));
dqext = (Math.abs(dqext / qext));
nmax1 = n;
if (dqsca <= ddelt && dqext <= ddelt)
{
break;
}
}
if (dsca <= ddelt && dext <= ddelt)
{
```

```
break;
}
if (nma == 100)
{
ret="CONVERGENCE IS NOT OBTAINED FOR NPN1 =  : EXECUTION TERMINATED
\n"+npn1;
return ret;
}
}
nnnggg = ngauss + 1;
if (ngauss == npng1)
{
System.out.println("WARNING:NGAUSS=NPNG1");
}
mmax = nmax1;
for (ngaus = nnnggg; ngaus <= npng1; ngaus++)
{
ngauss = ngaus;
nggg = 2*ngauss;
nonSphericalClass.constt(ngauss, nmax, mmax, p, x, w, an, ann, s, ss,
np, eps);
ret=nonSphericalClass.vary(lam, mrr, mri, a, eps, np, ngauss, x, p, r,
dr, ddr, drr, dri, nmax,cbassblock);
if(null!=ret&&!ret.equalsIgnoreCase(""))
{
return ret;
}
nonSphericalClass.tmatr0(ngauss, x, w, an, ann, s, ss, ppi, pir, pii,
r,dr, ddr, drr, dri, nmax,
ncheck,tmatblock,trtiblock,rirgigblock,cbassblock);
qext = 0.0;
qsca = 0.0;
for (n = 1; n <= nmax; ++n)
{
n1 = n + nmax;
trlnn =trtiblock. trl[n-1][n-1];
tilnn = trtiblock.til[n-1][n-1];
trlnn1 = trtiblock.trl[n1-1][n1-1];
tilnn1 = trtiblock.til[n1-1][n1-1];
dn1 = (double) (2*n + 1);
qsca = qsca + dn1 * (trlnn * trlnn + tilnn * tilnn + trlnn1 * trlnn1 +
tilnn1 * tilnn1);
qext = qext + (trlnn + trlnn1) * dn1;
}
dsca = (Math.abs((qsca1 - qsca) / qsca));
dext = (Math.abs((qext1 - qext) / qext));
qext1 = qext;
qsca1 = qsca;
if (dsca <= ddelt && dext <= ddelt)
{
break;
}
}
qsca = 0.0;
qext = 0.0;
nnm = 2*nmax;
for (n = 1; n <=nnm; ++n)
```

```
{
qext = qext + trtiblock.tr1[n-1][n-1];
}
if (nmax1 > npn4)
{
ret="NMAX IS GREATER THAN NPN4: EXECUTION TERMINATED";
return ret;
}
gspBlock gspblock=new gspBlock(npn4,npn6);
for (n2 = 1; n2 <= nmax1; n2++)
{
nn2 = n2 + nmax;
for (n1 = 1; n1 <= nmax1; n1++)
{
nn1 = n1 + nmax;
zz1 = trtiblock.tr1[n1-1][n2-1];
gspblock.rt11[0][n1-1][n2-1] = zz1;
zz2 = trtiblock.ti1[n1-1][n2-1];
gspblock.it11[0][n1-1][n2-1]= zz2;
zz3 = trtiblock.tr1[n1-1][nn2-1];
gspblock.rt12[0][n1-1][n2-1] = zz3;
zz4 = trtiblock.ti1[n1-1][nn2-1];
gspblock.it12[0][n1-1][n2-1] = zz4;
zz5 =trtiblock. tr1[nn1-1][n2-1];
gspblock.rt21[0][n1-1][n2-1] = zz5;
zz6 = trtiblock.ti1[nn1-1][n2-1];
gspblock.it21[0][n1-1][n2-1] = zz6;
zz7 = trtiblock.tr1[nn1-1][nn2-1];
gspblock.rt22[0][n1-1][n2-1] = zz7;
zz8 = trtiblock.ti1[nn1-1][nn2-1];
gspblock.it22[0][n1-1][n2-1] = zz8;
qsca = qsca + zz1 * zz1 + zz2 * zz2 + zz3 * zz3 + zz4 * zz4 + zz5 * zz5
+ zz6 * zz6 + zz7 * zz7 + zz8 * zz8;
}
}
for (m = 1; m <= nmax1; ++m)
{
nonSphericalClass.tmatr(m, ngauss, x, w, an, ann, s, ss, ppi, pir, pii,
r, dr, ddr, drr, dri, nmax,
ncheck,tmatblock,trtiblock,rirgigblock,cbassblock);
nm = nmax - m + 1;
nm1 = nmax1 - m + 1;
m1 = m + 1;
qsc = 0.0;
for (n2 = 1; n2 <= nm1; n2++)
{
nn2 = n2 + m - 1;
n22 = n2 + nm;
for (n1 = 1; n1 <= nm1; ++n1)
{
nn1 = n1 + m - 1;
n11 = n1 + nm;
zz1 = trtiblock.tr1[n1-1][n2-1];
gspblock.rt11[m1-1][nn1-1][nn2-1] = zz1;
zz2 = trtiblock.ti1[n1-1][n2-1];
gspblock.it11[m1-1][nn1-1][nn2-1] = zz2;
zz3 = trtiblock.tr1[n1-1][n22-1];
```

```
gspblock.rt12[m1-1][nn1-1][nn2-1] = zz3;
zz4 =trtiblock. til[n1-1][n22-1];
gspblock.it12[m1-1][nn1-1][nn2-1] = zz4;
zz5 =trtiblock. trl[n11-1][n2-1];
gspblock.rt21[m1-1][nn1-1][nn2-1] = zz5;
zz6 = trtiblock.til[n11-1][n2-1];
gspblock.it21[m1-1][nn1-1][nn2-1] = zz6;
zz7 =trtiblock. trl[n11-1][n22-1];
gspblock.rt22[m1-1][nn1-1][nn2-1] = zz7;
zz8 =trtiblock. til[n11-1][n22-1];
gspblock.it22[m1-1][nn1-1][nn2-1] = zz8;
qsc = qsc+ (zz1 * zz1 + zz2 * zz2 + zz3 * zz3 + zz4 * zz4 + zz5 * zz5 +
zz6 * zz6 + zz7 * zz7 + zz8 *
zz8) * 2.;
}
}
nnm = 2*nm;
qxt = 0.0;
for (n = 1; n <= nnm; n++)
{
qxt = qxt+trtiblock.trl[n-1][n-1] * 2.0;
}
qsca = qsca+qsc;
qext = qext+qxt;
}
coeff1 = lam * lam * 0.5 / p;
csca = qsca * coeff1;
cext = -qext * coeff1;
nonSphericalClass.gsp(nmax1, csca, lam, al1, al2, al3, al4, be1,
be2,gspblock);
l1m = lmax + 1;
l1max = Math.max(l1max,l1m);
wgii = wg1[i - 1];
wgi = wgii * csca;
for (l1 = 1; l1 <= l1m; ++l1)
{
alph1[l1 - 1] = alph1[l1 - 1]+al1[l1 - 1] * wg1;
alph2[l1 - 1] = alph2[l1 - 1]+al2[l1 - 1] * wgi;
alph3[l1 - 1] = alph3[l1 - 1]+al3[l1 - 1] * wgi;
alph4[l1 - 1] = alph4[l1 - 1]+al4[l1 - 1] * wgi;
bet1[l1 - 1] = bet1[l1 - 1]+be1[l1 - 1] * wgi;
bet2[l1 - 1] = bet2[l1 - 1]+be2[l1 - 1] * wgi;
}
cscat = cscat+wgi;
cextin = cextin+cext * wgii;
}
for (l1 = 1; l1 <= l1max; ++l1)
{
alph1[l1 - 1] = alph1[l1 - 1]/cscat;
alph2[l1 - 1] = alph2[l1 - 1]/cscat;
alph3[l1 - 1] = alph3[l1 - 1]/cscat;
alph4[l1 - 1] = alph4[l1 - 1]/cscat;
bet1[l1 - 1] = bet1[l1 - 1]/cscat;
bet2[l1 - 1] = bet2[l1 - 1]/cscat;
}
walb = cscat / cextin;
cabsin=cextin-cscat;
```

```
nonSphericalClass.hovenr(l1max, alph1, alph2, alph3, alph4, bet1,
bet2);
asymm = alph1[1] / 3.0;
lmax = l1max - 1;
nonSphericalClass.matr(alph1, alph2, alph3, alph4, bet1, bet2, lmax,
npna);
return ret;
}

public int constt(int ngauss, int nmax, int mmax,double p, double x[],
double w[], double an[],double ann[][], double s[],
  double ss[], int np,double eps)
{
int i;
int n;
int n1;
int ng;
int nn;
int ng1;
int ng2;
double d;
double y;
double dd[]=new double[100];
double xx;
double ddd;
double w1[]=new double[npng2];
double x1[]=new double[npng2];
double x2[]=new double[npng2];
double w2[]=new double[npng2];
boolean flagIn=false;
for (n = 1; n <= nmax; ++n)
{
nn = n * (n + 1);
an[n-1] = (double) nn;
d = Math.sqrt((double) (2*n + 1) / (double) nn);
dd[n - 1] = d;
for (n1 = 1; n1 <= n; ++n1)
{
ddd = d * dd[n1 - 1] * 0.5;
ann[n-1][n1-1] = ddd;
ann[n1-1][n-1] = ddd;
}
}
ng = 2*(ngauss);
if (np == -2)
{
flagIn=true;
}

if(flagIn)
{
ng1 = (int) ((double) (ngauss) / 2.0);
ng2 = ngauss - ng1;
xx = -Math.cos(Math.atan(eps));
gauss(ng1, 0, 0, x1, w1);
gauss(ng2, 0, 0, x2, w2);
for (i = 1; i <= ng1; ++i)
```

```
{
w[i-1] = (xx + 1.0) * 0.5 * w1[i - 1];
x[i-1] = (xx + 1.0) * 0.5 * x1[i - 1] + (xx - 1.0) * 0.5;
}
for (i= 1; i<=ng2; ++i)
{
w[i+ ng1-1] = xx * (-0.5) * w2[i - 1];
x[i+ ng1-1] = xx * (-0.5) * x2[i - 1] + xx * 0.5;
}
for (i = 1; i <= ngauss; ++i)
{
w[ng - i] = w[i-1];
x[ng - i] = -x[i-1];
}
for (i = 1; i <= ngauss; ++i)
{
y = x[i-1];
y = 1.0 / (1.0 - y * y);
ss[i-1] = y;
ss[ng - i] = y;
y = Math.sqrt(y);
s[i-1] = y;
s[ng - i] = y;
}
}
else
{
gauss(ng, 0, 0, x, w);
for (i = 1; i <= ngauss; ++i)
{
y = x[i-1];
y = 1.0 / (1.0 - y * y);
ss[i-1] = y;
ss[ng - i] = y;
y = Math.sqrt(y);
s[i-1] = y;
s[ng - i] = y;
}
}
return 0;
}

public String vary(double lam, double mrr, double mri, double a, double
eps, int np, int ngauss, double x[], double p,
double r[], double dr[], double ddr[], double drr[], double dri[], int
nmax,cbassBlock cbassblock)
{
int i;
int ng;
int nnmax1;
int  nnmax2;
double v;
double  v1;
double  v2;
double ta;
double  tb;
double  pi;
```

```
double  vv;
double  pri;
double  prr;
double z[]=new double[npng2] ;
double zi[]=new double[npng2] ;
double zr[]=new double[npng2] ;
String strret="";
ng = 2*(ngauss);
if (np == -1)
{
rsp1(x, ng, ngauss, a, eps, np, r, dr);
}
if (np == -2)
{
rsp3(x, ng, ngauss, a, eps, r, dr);
}
pi = p * 2. / lam;
ppi = Math.PI * Math.PI;
pir = ppi * mrr;
pii = ppi * mri;
v = 1.0 / (mrr * mrr + mri * mri);
prr = mrr * v;
pri = -(mri) * v;
ta = 0.;
for (i = 1; i <= ng; ++i)
{
vv = Math.sqrt(r[i-1]);
v = vv * pi;
ta = Math.max(ta,v);
vv = 1.0 / v;
ddr[i-1] = vv;
drr[i-1] = prr * vv;
dri[i-1] = pri * vv;
v1 = v * mrr;
v2 = v * mri;
z[i - 1] = v;
zr[i - 1] = v1;
zi[i - 1] = v2;
}
if (nmax > 100)
{
strret="NMAX = "+nmax+", I.E. GREATER THAN NPN1 = "+npn1;
return strret;
}
tb = ta * Math.sqrt(mrr * mrr + mri * mri);
tb = Math.max(tb,(double) (nmax));
nnmax1 = (int) (Math.sqrt((Math.max(ta,(double) (nmax)))) * 1.2 + 3.0);
nnmax2 = (int) (tb +Math.pow(tb, 0.33333) * 4.0 + Math.sqrt(tb) * 1.2);
nnmax2 = nnmax2 - nmax + 5;
bess(z, zr, zi, ng, nmax, nnmax1, nnmax2,cbassblock);
return strret;
}

public int rsp1(double x[], int ng, int ngauss,double rev, double eps,
int np, double r[],double dr[])
{
int i;
```

```
double a;
double c;
double s;
double aa;
double cc;
double ee;
double rr;
double ss;
double ee1;

a = rev * Math.pow(eps, 1.0/3.0);
aa = a * a;
ee = eps * eps;
ee1 = ee - 1.0;
for (i = 1; i <= ngauss; ++i)
{
c = x[i-1];
cc = c * c;
ss = 1.0 - cc;
s = Math.sqrt(ss);
rr = 1.0 / (ss + ee * cc);
r[i-1] = aa * rr;
r[ng - i] = r[i-1];
dr[i-1] = rr * c * s * ee1;
dr[ng - i] = -dr[i-1];
}
return 0;
}

public int rsp3(double x[], int ng, int ngauss,double rev, double eps,
double r[], double dr[])
{
int i;
double a;
double h;
double co;
double si;
double rad;
double rthet;

h = rev * Math.pow((2.0 / (eps * 3.0 * eps)), 1.0/3.0);
a = h * eps;
for (i = 1; i <= ngauss; ++i)
{
co = -x[i-1];
si = Math.sqrt(1.0 - co * co);
if (si / co > a / h)
{
rad = a / si;
rthet = -a * co / (si * si);
r[i-1] = rad * rad;
r[ng - i] = r[i-1];
dr[i-1] = -rthet / rad;
dr[ng - i] = -dr[i-1];
}
else
{
```

```
rad = h / co;
rthet = h * si / (co * co);
r[i-1] = rad * rad;
r[ng - i] = r[i-1];
dr[i-1] = -rthet / rad;
dr[ng - i] = -dr[i-1];
}}
return 0;
}

public int bess(double x[], double xr[], double xi[],int ng, int nmax,
int nnmax1, int nnmax2,cbassBlock cbassblock)
{
int i;
int l;
int n;
double yi;
double yr;
double xx;
double aj[]=new double[npn1];
double ay[]=new double[npn1];
double adj[]=new double[npn1];
double aji[]=new double[npn1];
double ajr[]=new double[npn1];
double ady[]=new double[npn1];
double adji[]=new double[npn1];
double adjr[]=new double[npn1];

for (i = 1; i <= ng; ++i)
{
xx = x[i-1];
rjb(xx, aj, adj, nmax, nnmax1);
ryb(xx, ay, ady, nmax);
yr = xr[i-1];
yi = xi[i-1];
cjb(yr, yi, ajr, aji, adjr, adji, nmax, nnmax2);
for (n = 1; n <= nmax; ++n)
{
cbassblock.j[i-1][n-1] = aj[n - 1];
cbassblock.y[i-1][n-1] = ay[n - 1];
cbassblock.jr[i-1][n-1] = ajr[n - 1];
cbassblock.ji[i-1][n-1] = aji[n - 1];

cbassblock.dj[i-1][n-1] = adj[n - 1];
cbassblock.dy[i-1][n-1] = ady[n - 1];
cbassblock.djr[i-1][n-1] = adjr[n - 1];
cbassblock.dji[i-1][n-1] = adji[n - 1];
}
}

return 0;
}

public int rjb(double x, double y[], double u[],int nmax, int nnmax)
{
int i;
int l;
```

```
int i1;
int l1;
double z[]=new double [800];
double y0;
double z0;
double y1;
double yi;
double xx;
double yi1;

l = nmax + nnmax;
xx = 1.0 / x;
z[l - 1] = 1.0 / ((double) (2*l + 1) * xx);
l1 = l - 1;
for (i = 1; i<= l1; ++i)
{
i1 = l - i;
z[i1 - 1] = 1.0 / ((double) (2*i1 + 1) * xx - z[i1]);
}
z0 = 1.0 / (xx - z[0]);
y0 = z0 * Math.cos(x) * xx;
y1 = y0 * z[0];
u[0] = y0 - y1 * xx;
y[0] = y1;
for (i = 2; i <= nmax; ++i)
{
yi1 = y[i - 2];
yi = yi1 * z[i - 1];
u[i-1] = yi1 - (double) i * yi * xx;
y[i-1] = yi;
}
return 0;
}

public int ryb(double x, double y[], double v[], int nmax)
{
int i;
int nmax1;
double c;
double s;
double x1;
double x2;
double x3;
double y1;

c = Math.cos(x);
s = Math.sin(x);
x1 = 1.0 / x;
x2 = x1 * x1;
x3 = x2 * x1;
y1 = -c * x2 - s * x1;
y[0] = y1;
y[1] = (x3 * -3.0 + x1) * c - x2 * 3.0 * s;
nmax1 = nmax - 1;
for (i= 2; i<= nmax1; ++i)
{
y[i] = (double) (2*i + 1) * x1 * y[i-1] - y[i - 2];
```

```
}
v[0] = -x1 * (c + y1);
for (i= 2; i <= nmax; ++i)
{
v[i-1] = y[i - 2] - (double) i * x1 * y[i-1];
}
return 0;
}

public int cjb(double xr, double xi, double yr[], double yi[], double
ur[], double ui[], int nmax, int nnmax)
{
int i;
int l;
int i1;
int l1;
double ai;
double  ci;
double  ar;
double  cr;
double  qf;
double  qi;
double  ari;
double  cui[]=new double [npn1];
double  cyi[]=new double [npn1];
double  czi[]=new double [1200];
double  cur[]=new double [npn1];
double  cyr[]=new double [npn1];
double  czr[]=new double [1200];
double  cu1i;
double  cy0i;
double  cz0i;
double  cy1i;
double  cu1r;
double  cz0r;
double  cy0r;
double  cy1r;
double  cuii;
double  cyii;
double  cuir;
double  cyir;
double  cxxi;
double  cxxr;
double  xrxi;
double cyi1i;
double  cyi1r;

l = nmax + nnmax;
xrxi = 1.0 / (xr * xr + xi * xi);
cxxr = xr * xrxi;
cxxi = -(xi) * xrxi;
qf = 1.0 / (double) (2*l + 1);
czr[l - 1] = xr * qf;
czi[l - 1] = xi * qf;
l1 = l - 1;
for (i = 1; i <=l1; ++i)
{
```

```
i1 = l - i;
qf = (double) (2*i1 + 1);
ar = qf * cxxr - czr[i1];
ai = qf * cxxi - czi[i1];
ari = 1.0 / (ar * ar + ai * ai);
czr[i1 - 1] = ar * ari;
czi[i1 - 1] = -ai * ari;
}
ar = cxxr - czr[0];
ai = cxxi - czi[0];
ari = 1.0 / (ar * ar + ai * ai);
cz0r = ar * ari;
cz0i = -ai * ari;
cr = Math.cos(xr) * Math.cosh(xi);
ci = -Math.sin(xr) * Math.sinh(xi);
ar = cz0r * cr - cz0i * ci;
ai = cz0i * cr + cz0r * ci;
cy0r = ar * cxxr - ai * cxxi;
cy0i = ai * cxxr + ar * cxxi;
cy1r = cy0r * czr[0] - cy0i * czi[0];
cy1i = cy0i * czr[0] + cy0r * czi[0];
ar = cy1r * cxxr - cy1i * cxxi;
ai = cy1i * cxxr + cy1r * cxxi;
cu1r = cy0r - ar;
cu1i = cy0i - ai;
cyr[0] = cy1r;
cyi[0] = cy1i;
cur[0] = cu1r;
cui[0] = cu1i;
yr[0] = cy1r;
yi[0] = cy1i;
ur[0] = cu1r;
ui[0] = cu1i;
for (i = 2; i <= nmax; ++i)
{
qi = (double) i;
cyi1r = cyr[i - 2];
cyi1i = cyi[i - 2];
cyir = cyi1r * czr[i - 1] - cyi1i * czi[i - 1];
cyii = cyi1i * czr[i - 1] + cyi1r * czi[i - 1];
ar = cyir * cxxr - cyii * cxxi;
ai = cyii * cxxr + cyir * cxxi;
cuir = cyi1r - qi * ar;
cuii = cyi1i - qi * ai;
cyr[i - 1] = cyir;
cyi[i - 1] = cyii;
cur[i - 1] = cuir;
cui[i - 1] = cuii;
yr[i-1] = cyir;
yi[i-1] = cyii;
ur[i-1] = cuir;
ui[i-1] = cuii;
}
return 0;
}
```

```
public int tmatr0(int ngauss, double x[], double w[],double an[],
double ann[][], double s[], double ss[],double ppi, double pir,
  double pii, double r[],double dr[], double ddr[], double drr[],
double dri[], int nmax, int ncheck,tmatBlock tmatblock,trtiBlock
trtiblock,rirgigBlock rirgigblock,
  cbassBlock cbassblock)
{
int i;
int  n;
int i1;
int  i2;
int  k1;
int  k2;
int  n1;
int  n2;
int ng;
int nm;
int mm1;
int kk1;
int  kk2;
int ngss;
int nnmax;
double f1;
double  f2;
double a12;
double  a21;
double  a22;
double si;
double rr[]=new double[npng2];
double aa1;
double  dd1;
double  dd2;
double  b1i;
double  c1i;
double  c2i;
double  b2i;
double  an1;
double  an2;
double  c3i;
double  b3i;
double c4i;
double  b1r;
double  c1r;
double  c2r;
double  b2r;
double  c3r;
double  b3r;
double  dv1[]=new double[npn1];
double  dv2[]=new double[npn2];
double qj1;
double  c4r;
double  b4r;
double  b4i;
double  c5r;
double  c5i;
double  b5r;
double  b5i;
```

```
double qy1;
double  ai12;
double  ai21;
double  ar12;
double  ar21;
double  gi12;
double  sig[]=new double[npn2];
double  gr12;
double  gr21;
double  gi21;
double  d1n1;
double  d2n1;
double  d1n2;
double  d2n2;
double  uri;
double  rri;
double  an12;
double  qdj1;
double  qji2;
double  qjr2;
double  qdy1;
double  tai12;
double  tai21;
double  ddri;
double  tgi12;
double  drii;
double  tgi21;
double  tar12;
double  tar21;
double  tgr12;
double  drri;
double  tgr21;
double  tpii;
double tppi;
double  tpir;
double  qdji2;
double  qdjr2;
double  factor;

mm1 = 1;
nnmax = nmax + nmax;
ng = 2*(ngauss);
ngss = ng;
factor = 1.0;
if (ncheck == 1)
{
ngss = ngauss;
factor = 2.0;
}
else
{
//Continue
}
si = 1.0;
for (n = 1; n <= nnmax; ++n)
{
si = -si;
```

```
sig[n - 1] = si;
}
for (i = 1; i<= ngauss; ++i)
{
i1 = ngauss + i;
i2 = ngauss - i + 1;
vig(x[i1-1], nmax, 0, dv1, dv2);
for (n = 1; n <=nmax; ++n)
{
si = sig[n - 1];
dd1 = dv1[n - 1];
dd2 = dv2[n - 1];
tmatblock.d1[i1-1][n-1] = dd1;
tmatblock.d2[i1-1][n-1] = dd2;
tmatblock.d1[i2-1][n-1] = dd1 * si;
tmatblock.d2[i2-1][n-1] = -dd2 * si;
}
}
for (i = 1; i <= ngss; ++i)
{
rr[i - 1] = w[i-1] * r[i-1];
}
for (n1 = mm1; n1 <=nmax; ++n1)
{
an1 = an[n1-1];
for (n2 = mm1; n2 <= nmax; ++n2)
{
an2 = an[n2-1];
ar12 = 0.0;
ar21 = 0.0;
ai12 = 0.0;
ai21 = 0.0;
gr12 = 0.0;
gr21 = 0.0;
gi12 = 0.0;
gi21 = 0.0;
if (ncheck == 1 && sig[n1 + n2 - 1] < 0.0)
{
an12 = ann[n1-1][n2-1] * factor;
rirgigblock.r12[n1-1][n2-1] = ar12 * an12;
rirgigblock.r21[n1-1][n2-1] = ar21 * an12;
rirgigblock.i12[n1-1][n2-1] = ai12 * an12;
rirgigblock.i21[n1-1][n2-1] = ai21 * an12;
rirgigblock.rg12[n1-1][n2-1] = gr12 * an12;
rirgigblock.rg21[n1-1][n2-1] = gr21 * an12;
rirgigblock.ig12[n1-1][n2-1] = gi12 * an12;
rirgigblock.ig21[n1-1][n2-1] = gi21 * an12;
}
else
{
for (i = 1; i <= ngss; ++i)
{
d1n1 = tmatblock.d1[i-1][n1-1];
d2n1 = tmatblock.d2[i-1][n1-1];
d1n2 = tmatblock.d1[i-1][n2-1];
d2n2 = tmatblock.d2[i-1][n2-1];
a12 = d1n1 * d2n2;
```

```
a21 = d2n1 * d1n2;
a22 = d2n1 * d2n2;
aa1 = a12 + a21;
qj1 = cbassblock.j[i-1][n1-1];
qy1 = cbassblock.y[i-1][n1-1];
qjr2 = cbassblock.jr[i-1][n2-1];
qji2 = cbassblock.ji[i-1][n2-1];
qdjr2 = cbassblock.djr[i-1][n2-1];
qdji2 = cbassblock.dji[i-1][n2-1];
qdj1 = cbassblock.dj[i-1][n1-1];
qdy1 = cbassblock.dy[i-1][n1-1];
c1r = qjr2 * qj1;
c1i = qji2 * qj1;
b1r = c1r - qji2 * qy1;
b1i = c1i + qjr2 * qy1;
c2r = qjr2 * qdj1;
c2i = qji2 * qdj1;
b2r = c2r - qji2 * qdy1;
b2i = c2i + qjr2 * qdy1;
ddri = ddr[i-1];
c3r = ddri * c1r;
c3i = ddri * c1i;
b3r = ddri * b1r;
b3i = ddri * b1i;
c4r = qdjr2 * qj1;
c4i = qdji2 * qj1;
b4r = c4r - qdji2 * qy1;
b4i = c4i + qdjr2 * qy1;
drri = drr[i-1];
drii = dri[i-1];
c5r = c1r * drri - c1i * drii;
c5i = c1i * drri + c1r * drii;
b5r = b1r * drri - b1i * drii;
b5i = b1i * drri + b1r * drii;
uri = dr[i-1];
rri = rr[i - 1];
f1 = rri * a22;
f2 = rri * uri * an1 * a12;
ar12 = ar12 + f1 * b2r + f2 * b3r;
ai12 = ai12 + f1 * b2i + f2 * b3i;
gr12 = gr12 + f1 * c2r + f2 * c3r;
gi12 = gi12 + f1 * c2i + f2 * c3i;
f2 = rri * uri * an2 * a21;
ar21 = ar21 + f1 * b4r + f2 * b5r;
ai21 = ai21 + f1 * b4i + f2 * b5i;
gr21 = gr21 + f1 * c4r + f2 * c5r;
gi21 = gi21 + f1 * c4i + f2 * c5i;
}
an12 = ann[n1-1][n2-1] * factor;
rirgigblock.r12[n1-1][n2-1] = ar12 * an12;
rirgigblock.r21[n1-1][n2-1] = ar21 * an12;
rirgigblock.i12[n1-1][n2-1] = ai12 * an12;
rirgigblock.i21[n1-1][n2-1] = ai21 * an12;
rirgigblock.rg12[n1-1][n2-1] = gr12 * an12;
rirgigblock.rg21[n1-1][n2-1] = gr21 * an12;
rirgigblock.ig12[n1-1][n2-1] = gi12 * an12;
rirgigblock.ig21[n1-1][n2-1] = gi21 * an12;
```

```
}
}
}
tpir = pir;
tpii = pii;
tppi = ppi;
nm = nmax;
for (n1 = mm1; n1 <= nmax; ++n1)
{
k1 = n1 - mm1+1;
kk1 = k1 + nm;
for (n2 = mm1; n2 <= nmax; ++n2)
{
k2 = n2 - mm1+1;
kk2 = k2 + nm;
tar12 = rirgigblock.i12[n1-1][n2-1];
tai12 = -rirgigblock.r12[n1-1][n2-1];
tgr12 = rirgigblock.ig12[n1-1][n2-1];
tgi12 = -rirgigblock.rg12[n1-1][n2-1];
tar21 = -rirgigblock.i21[n1-1][n2-1];
tai21 = rirgigblock.r21[n1-1][n2-1];
tgr21 = -rirgigblock.ig21[n1-1][n2-1];
tgi21 = rirgigblock.rg21[n1-1][n2-1];
trtiblock.tqr[k1-1][k2-1] = tpir * tar21 - tpii * tai21 + tppi * tar12;
trtiblock.tqi[k1-1][k2-1] = tpir * tai21 + tpii * tar21 + tppi * tai12;
trgqr[k1-1][k2-1] = tpir * tgr21 - tpii * tgi21 + tppi * tgr12;
trgqi[k1-1][k2-1] = tpir * tgi21 + tpii * tgr21 + tppi * tgi12;
trtiblock.tqr[k1-1][kk2-1]= 0.0;
trtiblock.tqi[k1-1][kk2-1]= 0.0;
trgqr[k1-1][kk2-1]= 0.0;
trgqi[k1-1][kk2-1]= 0.0;
trtiblock.tqr[kk1-1][k2-1] = 0.0;
trtiblock.tqi[kk1-1][k2-1] = 0.0;
trgqr[kk1-1][k2-1] = 0.0;
trgqi[kk1-1][k2-1] = 0.0;
trtiblock.tqr[kk1-1][kk2-1] = tpir * tar12 - tpii * tai12 + tppi *
tar21;
trtiblock.tqi[kk1-1][kk2-1] = tpir * tai12 + tpii * tar12 + tppi *
tai21;
trgqr[kk1-1][kk2-1] = tpir * tgr12 - tpii * tgi12 + tppi * tgr21;
trgqi[kk1-1][kk2-1] = tpir * tgi12 + tpii * tgr12 + tppi * tgi21;
}
}
nnmax = 2*nm;
for (n1 = 1; n1 <= nnmax; ++n1)
{
for (n2 = 1; n2 <= nnmax; ++n2)
{
qr[n1-1][n2-1] = trtiblock.tqr[n1-1][n2-1];
qi[n1-1][n2-1] = trtiblock.tqi[n1-1][n2-1];
rgqr[n1-1][n2-1] = trgqr[n1-1][n2-1];
rgqi[n1-1][n2-1] = trgqi[n1-1][n2-1];
}
}
tt(nmax, ncheck,trtiblock);
return 0;
}
```

```
public int tmatr(int m, int ngauss, double x[],double w[], double an[],
double ann[][], double s[], double ss[], double ppi,
 double pir, double pii,double r[], double dr[], double ddr[], double
drr[], double dri[], int nmax, int ncheck,tmatBlock tmatblock,trtiBlock
trtiblock,rirgigBlock rirgigblock,cbassBlock cbassblock)
{
int i;
int  n;
int i1;
int  i2;
int k1;
int  n1;
int  n2;
int  k2;
int ng;
int nm;
int mm1;
int kk1;
int  kk2;
int ngss;
int nnmax;
double e1;
double  f1;
double  f2;
double e2;
double  e3;
double a11;
double  a12;
double  a21;
double  a22;
double  ds[]=new double[npng2];
double qm;
double  si;
double  rr[]=new double[npng2];
double  wr;
double  aa1;
double  aa2;
double  dd1;
double  dd2;
double  b1i;
double  c1i;
double  c2i;
double  b2i;
double  an1;
double  an2;
double  c3i;
double  b3i;
double  c4i;
double  b1r;
double  c1r;
double  c2r;
double  b2r;
double  c3r;
double  b3r;
double  dv1[]=new double[npn1];
double  dv2[]=new double[npn1];
```

```
double  qj1;
double  c4r;
double  b4r;
double  b4i;
double  c5r;
double c5i;
double  b5r;
double  b5i;
double  c6r;
double  c6i;
double  b6r;
double  qy1;
double  b6i;
double  c7r;
double  c7i;
double  b7r;
double  b7i;
double  c8r;
double  c8i;
double  b8r;
double  b8i;
double ai11;
double  ai12;
double  ai21;
double  ai22;
double  ar11;
double  ar12;
double  ar21;
double sig[]=new double[npn2];
double ar22;
double  gr11;
double gr12;
double  dss[]=new double[npng2];
double  qmm;
double  gr21;
double  gr22;
double  gi11;
double  gi12;
double  gi21;
double  gi22;
double  d1n1;
double  d2n1;
double  d1n2;
double  d2n2;
double  uri;
double  dsi;
double  rri;
double  an12;
double  qdj1;
double  qji2;
double  qjr2;
double  qdy1;
double  tai11;
double  tai12;
double  tai21;
double  ddri;
double  tai22;
```

```
double  tgi11;
double  tgi12;
double  drii;
double  tar11;
double  tar12;
double  tar21;
double  tgi21;
double  tar22;
double  tgi22;
double  tgr11;
double  tgr12;
double  drri;
double  tgr21;
double  dssi;
double  tpii;
double  tgr22;
double tppi;
double  tpir;
double  qdji2;
double  qdjr2;
double factor;

mm1 = m;
qm = (double) (m);
qmm = qm * qm;
ng = 2*(ngauss);
ngss = ng;
factor = 1.;
if (ncheck == 1)
{
ngss = ngauss;
factor = 2.;
   }
else
{
//Continue
}
si = 1.;
nm = nmax + nmax;
for (n = 1; n <= nm; ++n)
{
si = -si;
sig[n - 1] = si;
}
for (i = 1; i <= ngauss; ++i)
{
i1 = ngauss + i;
i2 = ngauss - i + 1;
vig(x[i1-1], nmax, m, dv1, dv2);
for (n = 1; n <= nmax; ++n)
{
si = sig[n - 1];
dd1 = dv1[n - 1];
dd2 = dv2[n - 1];
tmatblock.d1[i1-1][n-1] = dd1;
tmatblock.d2[i1-1][n-1] = dd2;
tmatblock.d1[i2-1][n-1] = dd1 * si;
```

```
tmatblock.d2[i2-1][n-1] = -dd2 * si;
}
}
for (i = 1; i <= ngss; ++i)
{
wr = w[i-1] * r[i-1];
ds[i - 1] = s[i-1] * qm * wr;
dss[i - 1] = ss[i-1] * qmm;
rr[i - 1] = wr;
}
for (n1 = mm1; n1 <= nmax; ++n1)
{
an1 = an[n1-1];
for (n2 = mm1; n2 <= nmax; ++n2)
{
an2 = an[n2-1];
ar11 = 0.0;
ar12 = 0.0;
ar21 = 0.0;
ar22 = 0.0;
ai11 = 0.0;
ai12 = 0.0;
ai21 = 0.0;
ai22 = 0.0;
gr11 = 0.0;
gr12 = 0.0;
gr21 = 0.0;
gr22 = 0.0;
gi11 = 0.0;
gi12 = 0.0;
gi21 = 0.0;
gi22 = 0.0;
si = sig[n1 + n2 - 1];
for (i = 1; i<= ngss; ++i)
{
d1n1 = tmatblock.d1[i-1][n1-1];
d2n1 = tmatblock.d2[i-1][n1-1];
d1n2 = tmatblock.d1[i-1][n2-1];
d2n2 = tmatblock.d2[i-1][n2-1];
a11 = d1n1 * d1n2;
a12 = d1n1 * d2n2;
a21 = d2n1 * d1n2;
a22 = d2n1 * d2n2;
aa1 = a12 + a21;
aa2 = a11 * dss[i - 1] + a22;
qj1 = cbassblock.j[i-1][n1-1];
qy1 =cbassblock.y[i-1][n1-1];
qjr2 = cbassblock.jr[i-1][n2-1];
qji2 =cbassblock.ji[i-1][n2-1];
qdjr2 = cbassblock.djr[i-1][n2-1];
qdji2 = cbassblock.dji[i-1][n2-1];
qdj1 = cbassblock.dj[i-1][n1-1];
qdy1 = cbassblock.dy[i-1][n1-1];
c1r = qjr2 * qj1;
c1i = qji2 * qj1;
b1r = c1r - qji2 * qy1;
b1i = c1i + qjr2 * qy1;
```

```
c2r = qjr2 * qdj1;
c2i = qji2 * qdj1;
b2r = c2r - qji2 * qdy1;
b2i = c2i + qjr2 * qdy1;
ddri = ddr[i-1];
c3r = ddri * c1r;
c3i = ddri * c1i;
b3r = ddri * b1r;
b3i = ddri * b1i;
c4r = qdjr2 * qj1;
c4i = qdji2 * qj1;
b4r = c4r - qdji2 * qy1;
b4i = c4i + qdjr2 * qy1;
drri = drr[i-1];
drii = dri[i-1];
c5r = c1r * drri - c1i * drii;
c5i = c1i * drri + c1r * drii;
b5r = b1r * drri - b1i * drii;
b5i = b1i * drri + b1r * drii;
c6r = qdjr2 * qdj1;
c6i = qdji2 * qdj1;
b6r = c6r - qdji2 * qdy1;
b6i = c6i + qdjr2 * qdy1;
c7r = c4r * ddri;
c7i = c4i * ddri;
b7r = b4r * ddri;
b7i = b4i * ddri;
c8r = c2r * drri - c2i * drii;
c8i = c2i * drri + c2r * drii;
b8r = b2r * drri - b2i * drii;
b8i = b2i * drri + b2r * drii;
uri = dr[i-1];
dsi = ds[i - 1];
dssi = dss[i - 1];
rri = rr[i - 1];
if (ncheck == 1)
{
if(si > 0.0)
{
f1 = rri * aa2;
f2 = rri * uri * an1 * a12;
ar12 = ar12 + f1 * b2r + f2 * b3r;
ai12 = ai12 + f1 * b2i + f2 * b3i;
gr12 = gr12 + f1 * c2r + f2 * c3r;
gi12 = gi12 + f1 * c2i + f2 * c3i;
f2 = rri * uri * an2 * a21;
ar21 = ar21 + f1 * b4r + f2 * b5r;
ai21 = ai21 + f1 * b4i + f2 * b5i;
gr21 = gr21 + f1 * c4r + f2 * c5r;
gi21 = gi21 + f1 * c4i + f2 * c5i;
}
else
{
e1 = dsi * aa1;
ar11 = ar11+e1 * b1r;
ai11 = ai11+e1 * b1i;
gr11 = gr11+e1 * c1r;
```

```
gi11 = gi11+e1 * c1i;
e2 = dsi * uri * a11;
e3 = e2 * an2;
e2 = e2 * an1;
ar22 = ar22 + e1 * b6r + e2 * b7r + e3 * b8r;
ai22 = ai22 + e1 * b6i + e2 * b7i + e3 * b8i;
gr22 = gr22 + e1 * c6r + e2 * c7r + e3 * c8r;
gi22 = gi22 + e1 * c6i + e2 * c7i + e3 * c8i;
}
}
else
{
e1 = dsi * aa1;
ar11 = ar11+e1 * b1r;
ai11 = ai11+e1 * b1i;
gr11 = gr11+e1 * c1r;
gi11 = gi11+e1 * c1i;
f1 = rri * aa2;
f2 = rri * uri * an1 * a12;
ar12 = ar12 + f1 * b2r + f2 * b3r;
ai12 = ai12 + f1 * b2i + f2 * b3i;
gr12 = gr12 + f1 * c2r + f2 * c3r;
gi12 = gi12 + f1 * c2i + f2 * c3i;
f2 = rri * uri * an2 * a21;
ar21 = ar21 + f1 * b4r + f2 * b5r;
ai21 = ai21 + f1 * b4i + f2 * b5i;
gr21 = gr21 + f1 * c4r + f2 * c5r;
gi21 = gi21 + f1 * c4i + f2 * c5i;
e2 = dsi * uri * a11;
e3 = e2 * an2;
e2 = e2 * an1;
ar22 = ar22 + e1 * b6r + e2 * b7r + e3 * b8r;
ai22 = ai22 + e1 * b6i + e2 * b7i + e3 * b8i;
gr22 = gr22 + e1 * c6r + e2 * c7r + e3 * c8r;
gi22 = gi22 + e1 * c6i + e2 * c7i + e3 * c8i;
}
}
an12 = ann[n1-1][n2-1] * factor;
rirgigblock.r11[n1-1][n2-1] = ar11 * an12;
rirgigblock.r12[n1-1][n2-1] = ar12 * an12;
rirgigblock.r21[n1-1][n2-1] = ar21 * an12;
rirgigblock.r22[n1-1][n2-1] = ar22 * an12;
rirgigblock.i11[n1-1][n2-1] = ai11 * an12;
rirgigblock.i12[n1-1][n2-1] = ai12 * an12;
rirgigblock.i21[n1-1][n2-1] = ai21 * an12;
rirgigblock.i22[n1-1][n2-1] = ai22 * an12;
rirgigblock.rg11[n1-1][n2-1] = gr11 * an12;
rirgigblock.rg12[n1-1][n2-1] = gr12 * an12;
rirgigblock.rg21[n1-1][n2-1] = gr21 * an12;
rirgigblock.rg22[n1-1][n2-1] = gr22 * an12;
rirgigblock.ig11[n1-1][n2-1] = gi11 * an12;
rirgigblock.ig12[n1-1][n2-1] = gi12 * an12;
rirgigblock.ig21[n1-1][n2-1] = gi21 * an12;
rirgigblock.ig22[n1-1][n2-1] = gi22 * an12;
}
}
tpir = pir;
```

```
tpii = pii;
tppi = ppi;
nm = nmax - mm1 + 1;
for (n1 = mm1; n1 <=nmax; ++n1)
{
k1 = n1 - mm1 + 1;
kk1 = k1 + nm;
for (n2 = mm1; n2 <= nmax; ++n2)
{
k2 = n2 - mm1 + 1;
kk2 = k2 + nm;
tar11 = -rirgigblock.r11[n1-1][n2-1];
tai11 = -rirgigblock.i11[n1-1][n2-1];
tgr11 = -rirgigblock.rg11[n1-1][n2-1];
tgi11 = -rirgigblock.ig11[n1-1][n2-1];
tar12 = rirgigblock.i12[n1-1][n2-1];
tai12 = -rirgigblock.r12[n1-1][n2-1];
tgr12 = rirgigblock.ig12[n1-1][n2-1];
tgi12 = -rirgigblock.rg12[n1-1][n2-1];
tar21 = -rirgigblock.i21[n1-1][n2-1];
tai21 = rirgigblock.r21[n1-1][n2-1];
tgr21 = -rirgigblock.ig21[n1-1][n2-1];
tgi21 = rirgigblock.rg21[n1-1][n2-1];
tar22 = -rirgigblock.r22[n1-1][n2-1];
tai22 = -rirgigblock.i22[n1-1][n2-1];
tgr22 = -rirgigblock.rg22[n1-1][n2-1];
tgi22 = -rirgigblock.ig22[n1-1][n2-1];
trtiblock.tqr[k1-1][k2-1] = tpir * tar21 - tpii * tai21 + tppi * tar12;
trtiblock.tqi[k1-1][k2-1] = tpir * tai21 + tpii * tar21 + tppi * tai12;
trgqr[k1-1][k2-1] = tpir * tgr21 - tpii * tgi21 + tppi * tgr12;
trgqi[k1-1][k2-1] = tpir * tgi21 + tpii * tgr21 + tppi * tgi12;
trtiblock.tqr[k1-1][kk2-1] = tpir * tar11 - tpii * tai11 + tppi *
tar22;
trtiblock.tqi[k1-1][kk2-1] = tpir * tai11 + tpii * tar11 + tppi *
tai22;
trgqr[k1-1][kk2-1] = tpir * tgr11 - tpii * tgi11 + tppi * tgr22;
trgqi[k1-1][kk2-1] = tpir * tgi11 + tpii * tgr11 + tppi * tgi22;
trtiblock.tqr[kk1-1][k2-1] = tpir * tar22 - tpii * tai22 + tppi *
tar11;
trtiblock.tqi[kk1-1][k2-1] = tpir * tai22 + tpii * tar22 + tppi *
tai11;
trgqr[kk1-1][k2-1] = tpir * tgr22 - tpii * tgi22 + tppi * tgr11;
trgqi[kk1-1][k2-1] = tpir * tgi22 + tpii * tgr22 + tppi * tgi11;
trtiblock.tqr[kk1-1][kk2-1] = tpir * tar12 - tpii * tai12 + tppi *
tar21;
trtiblock.tqi[kk1-1][kk2-1] = tpir * tai12 + tpii * tar12 + tppi *
tai21;
trgqr[kk1-1][kk2-1] = tpir * tgr12 - tpii * tgi12 + tppi * tgr21;
trgqi[kk1-1][kk2-1] = tpir * tgi12 + tpii * tgr12 + tppi * tgi21;
}
}
nnmax = 2*nm;
for (n1 = 1; n1 <= nnmax; ++n1)
{
for (n2 = 1; n2 <= nnmax; ++n2)
{
qr[n1-1][n2-1] = trtiblock.tqr[n1-1][n2-1];
```

```
qi[n1-1][n2-1] = trtiblock.tqi[n1-1][n2-1];
rgqr[n1-1][n2-1] = trgqr[n1-1][n2-1];
rgqi[n1-1][n2-1] = trgqi[n1-1][n2-1];
}
}
tt(nm, ncheck,trtiblock);
return 0;
}

public int vig(double x, int nmax, int m,double dv1[], double dv2[])
{
int i;
int n;
int i2;
double a;
double d1;
double d2;
double d3;
double qn;
double qs;
double qn1;
double qn2;
double qs1;
double der;
double qmm;
double qnm;
double qnm1;

a = 1.0;
qs = Math.sqrt(1.0 - x * x);
qs1 = 1.0 / qs;
for (n = 1; n <= nmax; ++n)
{
dv1[n-1] = 0.0;
dv2[n-1] = 0.0;
}
if (m == 0)
{
d1 = 1.0;
d2 = x;
for (n = 1; n <= nmax; ++n)
{
qn = (double) n;
qn1 = (double) (n + 1);
qn2 = (double) (2*n + 1);
d3 = (qn2 * x * d2 - qn * d1) / qn1;
der = qs1 * (qn1 * qn / qn2) * (-d1 + d3);
dv1[n-1] = d2;
dv2[n-1] = der;
d1 = d2;
d2 = d3;
}
return 0;
}
qmm = (double) (m * m);
for (i = 1; i <= m; ++i)
{
```

```
i2 = 2*i;
a = a * Math.sqrt((double) (i2 - 1) / (double) i2) * qs;
}
d1 = 0.0;
d2 = a;
for (n = m; n <= nmax; ++n)
{
qn = (double) n;
qn2 = (double) (2*n + 1);
qn1 = (double) (n + 1);
qnm = Math.sqrt(qn * qn - qmm);
qnm1 = Math.sqrt(qn1 * qn1 - qmm);
d3 = (qn2 * x * d2 - qnm * d1) / qnm1;
der = qs1 * (-qn1 * qnm * d1 + qn * qnm1 * d3) / qn2;
dv1[n-1] = d2;
dv2[n-1] = der;
d1 = d2;
d2 = d3;
}
return 0;
}

public int tt(int nmax, int ncheck,trtiBlock trtiblock)
{
int i;
int l;
int n1;
int n2;
int ndim;
int ipvt[]=new int[npn2];
int nnmax;
double b[]=new double[npn2];
double cond;
double work[]=new double[npn2];
double a[][]=new double[npn2][npn2];
double f[][]=new double[npn2][npn2];
double c[][]=new double[npn2][npn2];
double d[][]=new double[npn2][npn2];
double e[][]=new double[npn2][npn2];

ndim = 200;
nnmax = 2*(nmax);
for (n1 = 1; n1 <= nnmax; ++n1)
{
for (n2 = 1; n2 <= nnmax; ++n2)
{
f[n1-1][n2-1] = qi[n1-1][n2-1];
}
}
if (ncheck == 1)
{
inv1(nmax, f, a);
}
else
{
invert(ndim, nnmax, f, a, ipvt, work, b);
}
```

```
prod(qr, a, c, ndim, nnmax);
prod(c, qr, d, ndim, nnmax);
for (n1 = 1; n1 <= nnmax; ++n1)
{
for (n2 = 1; n2 <= nnmax; ++n2)
{
c[n1-1][n2-1] = d[n1-1][n2-1] + qi[n1-1][n2-1];
}
}
if (ncheck == 1)
{
inv1(nmax, c, qi);
}
else
{
invert(ndim, nnmax, c, qi, ipvt, work, b);
}
prod(a, qr, d, ndim, nnmax);
prod(d, qi, qr, ndim, nnmax);
prod(rgqr, qr, a, ndim, nnmax);
prod(rgqi, qi, c, ndim, nnmax);
prod(rgqr, qi, d, ndim, nnmax);
prod(rgqi, qr, e, ndim, nnmax);
for (n1 = 1; n1 <= nnmax; ++n1)
{
for (n2 = 1; n2 <= nnmax; ++n2)
{
trtiblock.tri[n1-1][n2-1] = -a[n1-1][n2-1] - c[n1-1][n2-1];
trtiblock.tii[n1-1][n2-1] = d[n1-1][n2-1] - e[n1-1][n2-1];
}
}
return 0;
}

public int prod(double ax[][], double bx[][], double c[][],int ndim,
int n)
{
int i;
int j;
int k;
int l;
double cij;

for (i = 1; i <= n; ++i)
{
for (j = 1; j <= n; ++j)
{
cij = 0.;
for (k = 1; k <= n; ++k)
{
cij = cij + ax[i-1][k-1] * bx[k-1][j-1];
}
c[i-1][j-1] = cij;
}
}
return 0;
}
```

```
public int inv1(int nmax, double f[][], double a[][])
{
int i;
int  l;
int j;
int  i1;
int  i2;
int  j1;
int  j2;
int  nn1;
int  nn2;
int  ind1[]= new int[npn1];
int  ind2[]= new int[npn1];
int  ndim;
int  ipvt[]= new int[npn1];
int  nnmax;
double b[]= new double[npn1];
double cond;
double work[]=new double[npn1];
double p1[][]= new double[npn1][npn1];
double q1[][]= new double[npn1][npn1];
double p2[][]= new double[npn1][npn1];
double q2[][]= new double[npn1][npn1];

ndim = npn1;
nn1 = (int) (((double) (nmax) - 0.1) * 0.5 + 1.0);
nn2 = nmax - nn1;
for (i = 1; i <= nmax; ++i)
{
ind1[i - 1] = 2*i - 1;
if (i > nn1)
{
ind1[i - 1] = nmax + 2*(i - nn1);
}
ind2[i - 1] = 2*i;
if (i > nn2)
{
ind2[i - 1] = nmax + 2*(i - nn2)-1;
}
}
nnmax = 2*(nmax);
for (i = 1; i <= nmax; ++i)
{
i1 = ind1[i - 1];
i2 = ind2[i - 1];
for (j = 1; j <= nmax; ++j)
{
j1 = ind1[j - 1];
j2 = ind2[j - 1];
q1[j-1][i-1] = f[j1-1][i1-1];
q2[j-1][i-1] = f[j2-1][i2-1];
}
}
invert(ndim, nmax, q1, p1,  ipvt, work, b);
invert(ndim, nmax, q2, p2,  ipvt, work, b);
for (i = 1; i <= nnmax; ++i)
```

```
{
for (j = 1; j <= nnmax; ++j)
{
a[j-1][i-1] = 0.0;
}
}
for (i = 1; i <= nmax; ++i)
{
i1 = ind1[i - 1];
i2 = ind2[i - 1];
for (j = 1; j <= nmax; ++j)
{
j1 = ind1[j - 1];
j2 = ind2[j - 1];
a[j1-1][i1-1] = p1[j-1][i-1];
a[j2-1][i2-1] = p2[j-1][i-1];
}
}
return 0;
}

public int invert(int ndim, int n, double a[][],double xa[][], int
ipvt[], double work[], double b[])
{
int i, j, k, l,m,jo, kb, km1, nm1, kp1;
double t, ek, anorm, ynorm, znorm;

ipvt[n-1] = 1;
if (n != 1)
{
nm1 = n - 1;
anorm = 0.0;
for (j = 1; j <= n; ++j)
{
t = 0.0;
for (i = 1; i <= n; ++i)
{
t = t+ Math.abs(a[i-1][j-1]);
}
if (t > anorm)
{
anorm = t;
}
}
for (k = 1; k <= nm1; ++k)
{
kp1 = k + 1;
m = k;
jo=k;
for (i = kp1; i <= n; ++i)
{
if (Math.abs(a[i-1][k-1]) > Math.abs(a[m-1][k-1]))
{
m = i;
}
if(Math.abs(a[i-1][k-1]) < Math.abs(a[m-1][k-1]))
{
```

```
m=m;
}
}
ipvt[k-1] = m;
if (m != k)
{
ipvt[n-1] = -ipvt[n-1];
}
t = a[m-1][k-1];
a[m-1][k-1] = a[k-1][k-1];
a[k-1][k-1] = t;
if (t != 0.)
{
for (i = kp1; i <= n; ++i)
{
a[i-1][k-1] = -a[i-1][k-1]/ t;
}
for (j = kp1; j <= n; ++j)
{
t = a[m-1][j-1];
a[m-1][j-1] = a[k-1][j-1];
a[k-1][j-1] = t;
if (t != 0.0)
{
for (i= kp1; i <=n; ++i)
{
a[i-1][j-1] = a[i-1][j-1]+ a[i-1][k-1] * t;
}
}
}
}
}
for (k = 1; k <= n; ++k)
{
t = 0.0;
if (k != 1)
{
km1 = k - 1;
for (i = 1; i <= km1; ++i)
{
t = t + a[i-1][k-1] * work[i-1];
}
}
ek = 1.0;
if (t < 0.0)
{
ek = -1.0;
}
if (a[k-1][k-1]== 0.0)
{
cond = 1e+52;

if (cond + 1.0 == cond)
{
System.out.println("THE MATRIX IS SINGULAR FOR THE GIVEN NUMERICAL
ACCURACY COND = "+cond);
}
```

```
for (i= 1; i<= n; ++i)
{
for (j = 1; j <= n; ++j)
{
b[j-1] = 0.0;
if (j == i)
{
b[j-1] = 1.0;
}
}
solve(ndim, n, a, b, ipvt);
for (j = 1; j <= n; ++j)
{
xa[j-1][i-1] = b[j-1];
}
}
return 0;
}
work[k-1] = -(ek + t) / a[k-1][k-1];
}
for (kb = 1; kb <= nm1; ++kb)
{
k = n - kb;
t = 0.0;
kp1 = k + 1;
for (i = kp1; i <= n; ++i)
{
t = t + a[i-1][k-1]* work[k-1];
}
work[k-1] = t;
m = ipvt[k-1];
if (m != k)
{
t = work[m-1];
work[m-1] = work[k-1];
work[k-1] = t;
}
}
ynorm = 0.0;
for (i = 1; i <= n; ++i)
{
ynorm = ynorm+(Math.abs(work[i-1]));
}
solve(ndim, n, a, work, ipvt);
znorm = 0.0;
for (i = 1; i <= n; ++i)
{
znorm = znorm+(Math.abs(work[i-1]));
}
cond = anorm * znorm / ynorm;
if (cond < 1.0)
{
cond = 1.0;
}
}
cond = 1.0;
if (a[0][0] == 0.0)
```

```
{
cond = 1e+52;
}
if (cond + 1.0 == cond)
{
System.out.println("THE MATRIX IS SINGULAR FOR THE GIVEN NUMERICAL
ACCURACY COND = "+cond);
}
for (i= 1; i<= n; ++i)
{
for (j = 1; j <= n; ++j)
{
b[j-1] = 0.0;
if (j == i)
{
b[j-1] = 1.0;
}
}
solve(ndim, n, a, b, ipvt);
for (j = 1; j <= n; ++j)
{
xa[j-1][i-1] = b[j-1];
/* L30: */
}
}
return 0;
}

public int solve(int ndim, int n, double a[][],double b[], int ipvt[])
{
int i;
int  k;
int  m;
int  kb;
int  km1;
int  nm1;
int  kp1;
double t;

if (n != 1)
{
nm1 = n - 1;
for (k = 1; k <= nm1; ++k)
{
kp1 = k + 1;
m = ipvt[k-1];
t = b[m-1];
b[m-1] = b[k-1];
b[k-1] = t;
for (i = kp1; i <= n; ++i)
{
b[i-1] = b[i-1] + a[i-1][k-1] * t;
}
}
for (kb = 1; kb <= nm1; ++kb)
{
km1 = n - kb;
```

```
k = km1 + 1;
b[k-1] = b[k-1] / a[k-1][k-1];
t = -b[k-1];
for (i = 1; i <= km1; ++i)
{
b[i-1] = b[i-1] + a[i-1][k-1] * t;
}
}
}
b[0] = b[0] / a[0][0];
return 0;
}

public int gsp(int nmax, double csca, double lam, double alf1[], double
alf2[], double alf3[], double alf4[], double bet1[],
double bet2[],gspBlock gspblock)
{
int i;
int  j;
int  l;
int  m;
int  n;
int i1;
int  k1;
int  k2;
int  k3;
int  k4;
int  k5;
int  k6;
int  m1;
int  l1;
int  n1;
int  m2;
int  kn;
int  nl;
int  nn;
int  nn1;
int  nnn;
int  mmin;
int  mmax;
int  m1min;
int  l1max;
int  nmax1;
int  m1max;
int nnmin;
int  nnmax;
int  nn1min;
int  nn1max;
double  t1;
double t2;
double t3;
double t4;
double x1;
double x2;
double x3;
double x4;
double x5;
```

```
double x6;
double x7;
double x8;
double ff[][]=new double[npn4][npn4];
double si;
double sj;
double sl;
double xi;
double xr;
double xx;
double dd1;
double tr1[][]=new double[npl1][npn4];
double tr2[][]=new double[npl1][npn4];
double dd2;
double ai1[]=new double[npn4];
double ai2[]=new double[npn4];
double dd3;
double dd4;
double dm1;
double ar1[]=new double[npn4];
double ar2[]=new double[npn4];
double g1l;
double g2l;
double g3l;
double ti1[][]=new double[npl1][npn4];
double ti2[][]=new double[npl1][npn4];
double ri2;
double g4l;
double dm2;
double dm3;
double dm4;
double rr1;
double rr2;
double tt1;
double tt2;
double ffn;
double sig;
double tt3;
double tt4;
double tt5;
double tt6;
double tt7;
double tt8;
double ssi[]=new double[npl];
double ssj[]=new double[npn1];
double sss;
double aai1;
double aai2;
double bbi1;
double bbi2;
double aar1;
double aar2;
double bbr1;
double bbr2;
double dd5i;
double dd5r;
double dm5i;
```

```
double g5li;
double dm5r;
double g5lr;
double sss1=0.0;
double fi[][]=new double[npn4][npn4];
double dk;
double fr[][]=new double[npn4][npn4];
double ri1;
double d1[][][]=new double[npl1][npn4][npn4];
double d2[][][]=new double[npl1][npn4][npn4];
double d3[][][]=new double[npl1][npn4][npn4];
double d4[][][]=new double[npl1][npn4][npn4];
double d5r[][][]=new double[npl1][npn4][npn4];
double d5i[][][]=new double[npl1][npn4][npn4];
double g1[][]=new double[npl1][npn6];
double g2[][]=new double[npl1][npn6];
double m1r[][][]=new double[npl1][npl1][npn4];
double m1i[][][]=new double[npl1][npl1][npn4];
double m2r[][][]=new double[npl1][npl1][npn4];
double m2i[][][]=new double[npl1][npl1][npn4];
Complex ci;
Complex cci;
Complex ccj;
Complex cim[]=new Complex[100];
Complex c1;

signum();
lmax = 2*(nmax);
l1max = lmax + 1;
ci = new Complex(0.0,1.0);
cim[0]=new Complex(ci.real(),ci.imag());
for (i = 2; i <= nmax; ++i)
{
cim[i-1] = cim[i-2].times(ci);
}
ssi[0] = 1.0;
for (i = 1; i<= lmax; ++i)
{
i1 = i + 1;
si = (double) (2*i + 1);
ssi[i1 - 1] = si;
if (i <= nmax)
{
ssj[i - 1] = Math.pow(si,0.5);
}
}
ci=new Complex(-ci.real(),-ci.imag());
for (i = 1; i<= nmax; ++i)
{
si = ssj[i - 1];
cci = new Complex(cim[i-1].real(),cim[i-1].imag());
for (j = 1; j <= nmax; ++j)
{
sj = 1.0 / ssj[j - 1];
c1=new Complex(sj*cim[j-1].real(),sj*cim[j-1].imag());
ccj=c1.div(cci);
fr[j-1][i-1]=ccj.real();
```

```
fi[j-1][i-1]=(ccj.times(ci)).real();
ff[j-1][i-1] = si * sj;
}
}
nmax1 = nmax + 1;
k1 = 1;
k2 = 0;
k3 = 0;
k4 = 1;
k5 = 1;
k6 = 2;
for (n = 1; n <= nmax; ++n)
{
for (nn = 1; nn <= nmax; ++nn)
{
m1max = Math.min(n,nn) + 1;
for (m1 = 1; m1 <= m1max; ++m1)
{
m = m1 - 1;
l1 = npn6+m;
tt1 = gspblock.rt11[m1-1][n-1][nn-1];
tt2 = gspblock.rt12[m1-1][n-1][nn-1];
tt3 = gspblock.rt21[m1-1][n-1][nn-1];
tt4 =gspblock.rt22[m1-1][n-1][nn-1];
tt5 = gspblock.it11[m1-1][n-1][nn-1];
tt6 = gspblock.it12[m1-1][n-1][nn-1];
tt7 = gspblock.it21[m1-1][n-1][nn-1];
tt8 = gspblock.it22[m1-1][n-1][nn-1];
t1 = tt1 + tt2;
t2 = tt3 + tt4;
t3 = tt5 + tt6;
t4 = tt7 + tt8;
tr1[l1-1][nn-1] = t1 + t2;
tr2[l1-1][nn-1] = t1 - t2;
ti1[l1-1][nn-1] = t3 + t4;
ti2[l1-1][nn-1] = t3 - t4;
if (m != 0)
{
l1 = 81 - m;
t1 = tt1 - tt2;
t2 = tt3 - tt4;
t3 = tt5 - tt6;
t4 = tt7 - tt8;
tr1[l1-1][nn-1] = t1 - t2;
tr2[l1-1][nn-1] = t1 + t2;
ti1[l1-1][nn-1] = t3 - t4;
ti2[l1-1][nn-1] = t3 + t4;
}
}
}
nn1max = nmax1 + n;
for (nn1 = 1; nn1 <= nn1max; ++nn1)
{
n1 = nn1 - 1;
ccg(n, n1, nmax, k1, k2, g1);
nnmax = Math.min(nmax,n1 + n);
nnmin = Math.max(1,Math.abs(n - n1));
```

```
kn = n + nn1;
for (nn = nnmin; nn <= nnmax; ++nn)
{
nnn = nn + 1;
sig = ssign[kn + nn - 1];
m1max = npn6+Math.min(n,nn);
aar1 = 0.0;
aar2 = 0.0;
aai1 = 0.0;
aai2 = 0.0;
for (m1 = npn6; m1 <= m1max; ++m1)
{
m = -(npn6-m1);
sss = g1[m1-1][nnn-1];
rr1 = tr1[m1-1][nn-1];
ri1 = ti1[m1-1][nn-1];
rr2 = tr2[m1-1][nn-1];
ri2 = ti2[m1-1][nn-1];
if (m != 0)
{
m2 = npn6 - m;
rr1 = rr1 + tr1[m2-1][nn-1] * sig;
ri1 = ri1 + ti1[m2-1][nn-1] * sig;
rr2 = rr2 + tr2[m2-1][nn-1] * sig;
ri2 = ri2 + ti2[m2-1][nn-1] * sig;
}
aar1 += sss * rr1;
aai1 += sss * ri1;
aar2 += sss * rr2;
aai2 += sss * ri2;
}
xr = fr[nn-1][n-1];
xi = fi[nn-1][n-1];
ar1[nn - 1] = aar1 * xr - aai1 * xi;
ai1[nn - 1] = aar1 * xi + aai1 * xr;
ar2[nn - 1] = aar2 * xr - aai2 * xi;
ai2[nn - 1] = aar2 * xi + aai2 * xr;
}
ccg(n, n1, nmax, k3, k4, g2);
m1 = Math.max(-n1 + 1,-n);
m2 = Math.min(n1 + 1,n);
m1max = npn6+m2;
m1min = npn6+m1;
for (m1 = m1min; m1 <= m1max; ++m1)
{
bbr1 = 0.0;
bbi1 = 0.0;
bbr2 = 0.0;
bbi2 = 0.0;
for (nn = nnmin; nn <= nnmax; ++nn)
{
nnn = nn + 1;
sss = g2[m1-1][nnn-1];
bbr1 = bbr1+sss * ar1[nn - 1];
bbi1 = bbi1+sss * ai1[nn - 1];
bbr2 = bbr2+sss * ar2[nn - 1];
bbi2 = bbi2+sss * ai2[nn - 1];
```

```
}
m1r[nn1-1][m1-1][n-1] = bbr1;
m1i[nn1-1][m1-1][n-1] = bbi1;
m2r[nn1-1][m1-1][n-1] = bbr2;
m2i[nn1-1][m1-1][n-1] = bbi2;
}
}
}
for (n = 1; n <= nmax; ++n)
{
for (nn = 1; nn <= nmax; ++nn)
{
m1 = Math.min(n,nn);
m1max = npn6+m1;
m1min = npn6 - m1;
nn1max = nmax1 + Math.min(n,nn);
for (m1 = m1min; m1 <= m1max; ++m1)
{
m = -(npn6-m1);
nn1min = (Math.abs(m - 1)) + 1;
dd1 = 0.0;
dd2 = 0.0;
for (nn1 = nn1min; nn1 <= nn1max; ++nn1)
{
xx = ssi[nn1 - 1];
x1 = m1r[nn1-1][m1-1][n-1];
x2 = m1i[nn1-1][m1-1][n-1];
x3 = m1r[nn1-1][m1-1][nn-1];
x4 = m1i[nn1-1][m1-1][nn-1];
x5 = m2r[nn1-1][m1-1][n-1];
x6 = m2i[nn1-1][m1-1][n-1];
x7 = m2r[nn1-1][m1-1][nn-1];
x8 = m2i[nn1-1][m1-1][nn-1];
dd1 = dd1+xx * (x1 * x3 + x2 * x4);
dd2 = dd2+xx * (x5 * x7 + x6 * x8);
}
d1[m1-1][nn-1][n-1] = dd1;
d2[m1-1][nn-1][n-1] = dd2;
}
mmax = Math.min(n,nn + 2);
mmin = Math.max(-n,-nn + 2);
m1max = npn6+mmax;
m1min = npn6+mmin;
for (m1 = m1min; m1 <= m1max; ++m1)
{
m = -(npn6-m1);
nn1min = (Math.abs(m - 1)) + 1;
dd3 = 0.0;
dd4 = 0.0;
dd5r = 0.0;
dd5i = 0.0;
m2 = npn6 - m + 2;
for (nn1 = nn1min; nn1 <= nn1max; ++nn1)
{
xx = ssi[nn1 - 1];
x1 = m1r[nn1-1][m1-1][n-1];
x2 = m1i[nn1-1][m1-1][n-1];
```

```
x3 = m2r[nn1-1][m1-1][n-1];
x4 = m2i[nn1-1][m1-1][n-1];
x5 = m1r[nn1-1][m2-1][nn-1];
x6 = m1i[nn1-1][m2-1][nn-1];
x7 = m2r[nn1-1][m2-1][nn-1];
x8 = m2i[nn1-1][m2-1][nn-1];
dd3 = dd3+xx * (x1 * x5 + x2 * x6);
dd4 = dd4+xx * (x3 * x7 + x4 * x8);
dd5r = dd5r+xx * (x3 * x5 + x4 * x6);
dd5i = dd5i+xx * (x4 * x5 - x3 * x6);
}
d3[m1-1][nn-1][n-1] = dd3;
d4[m1-1][nn-1][n-1] = dd4;
d5r[m1-1][nn-1][n-1] = dd5r;
d5i[m1-1][nn-1][n-1] = dd5i;
}
}
}
dk = lam * lam / (csca * 4.0 * Math.acos(-1.));
for (l1 = 1; l1 <= l1max; ++l1)
{
g1l = 0.0;
g2l = 0.0;
g3l = 0.0;
g4l = 0.0;
g5lr = 0.0;
g5li = 0.0;
l = l1 - 1;
sl = ssi[l1 - 1] * dk;
for (n = 1; n <= nmax; ++n)
{
nnmin = Math.max(1,Math.abs(n-l));
nnmax = Math.min(nmax,n + l);
if (nnmax >= nnmin)
{
ccg(n, l, nmax, k1, k2, g1);
if (l >= 2)
{
ccg(n, l, nmax, k5, k6, g2);
}
nl = n + l;
for (nn = nnmin; nn <= nnmax; ++nn)
{
nnn = nn + 1;
mmax = Math.min(n,nn);
m1min = npn6 - mmax;
m1max = npn6+mmax;
si = ssign[nl + nnn - 1];
dm1 = 0.0;
dm2 = 0.0;
for (m1 = m1min; m1 <= m1max; ++m1)
{
m = -(npn6-m1);
if (m >= 0)
{
sss1 = g1[m1-1][nnn-1];
}
```

```
if (m < 0)
{
sss1 = g1[npn6-m-1][nnn-1] * si;
}
dm1 = dm1+sss1 * d1[m1-1][nn-1][n-1];
dm2 = dm2+sss1 * d2[m1-1][nn-1][n-1];
}
ffn = ff[nn-1][n-1];
sss = g1[npn6][nnn-1] * ffn;
g1l = g1l+sss * dm1;
g2l = g2l+sss * dm2 * si;
if (l >= 2)
{
dm3 = 0.;
dm4 = 0.;
dm5r = 0.;
dm5i = 0.;
mmax = Math.min(n,nn+2);
mmin = Math.max(-n,-nn+2);
m1max = npn6+mmax;
m1min = npn6+mmin;
for (m1 = m1min; m1 <= m1max; ++m1)
{
m = -(npn6-m1);
sss1 = g2[npn6-m-1][nnn-1];
dm3 = dm3+sss1 * d3[m1-1][nn-1][n-1];
dm4 = dm4+sss1 * d4[m1-1][nn-1][n-1];
dm5r = dm5r+sss1 * d5r[m1-1][nn-1][n-1];
dm5i = dm5i+sss1 * d5i[m1-1][nn-1][n-1];
}
g5lr = g5lr-sss * dm5r;
g5li = g5li-sss * dm5i;
sss = g2[npn4-1][nnn-1] * ffn;
g3l = g3l+sss * dm3;
g4l = g4l+sss * dm4 * si;
}
}
}
}
g1l = g1l*sl;
g2l = g2l*sl;
g3l = g3l*sl;
g4l = g4l*sl;
g5lr = g5lr*sl;
g5li = g5li*sl;
alf1[l1-1] = g1l + g2l;
alf2[l1-1] = g3l + g4l;
alf3[l1-1] = g3l - g4l;
alf4[l1-1] = g1l - g2l;
bet1[l1-1] = g5lr * 2.;
bet2[l1-1] = g5li * 2.;
lmax = l;
if (Math.abs(g1l) < 1e-6)
{
break;
}
}
```

```
return 0;
}

public int signum()
{
int n;
ssign[0] = 1.0;
for (n = 2; n <= 899; ++n)
{
ssign[n - 1] = -ssign[n - 2];
}
return 0;
}

public int ccg(int n, int n1, int nmax, int k1, int k2, double gg[][])
{
int l;
int m;
int m1;
int mf;
int mm;
int nn;
int nnf;
int min;
int nnl;
int nnm;
int nnu;
int mind;
int r;
int s;
double a;
double b;
double c=0.0;
double d;
double c1;
double c2;
double cd[]=new double[npn5];
double cu[]=new double[npn5];;

if (nmax <= npn4 && 0 <= n1 && n1 <= nmax + n && n >= 1 && n <= nmax)
{
nnf = Math.min(n + n1,nmax);
min = npn6 - n;
mf = npn6+n;
if (k1 == 1 && k2 == 0)
{
min = npn6;
}
m=0;
for (mind = min; mind <= mf; ++mind)
{
m=-(npn6-mind);
mm = m * k1 + k2;
m1 = mm - m;
if (Math.abs(m1) <= n1)
{
nnl = Math.max(Math.abs(mm),Math.abs(n - n1));
```

```
if (nnl <=nnf)
{
nnu = n + n1;
nnm = (int) ((nnu + nnl) * .5);
if (nnu == nnl)
{
nnm = nnl;
}
c=ccgin(n, n1, m, mm, c);
if(nnl-1<0)
{
cu[0] = c;
}
else
{
cu[nnl-1] = c;
}
if (nnl != nnf)
{
c2 = 0.0;
c1 = c;
for (nn = nnl + 1; nn <= Math.min(nnm,nnf); ++nn)
{
a = (double) ((nn + mm) * (nn - mm) * (n1 - n + nn));
a = a*(double) ((n - n1 + nn) * (n + n1 - nn + 1) * (n + n1 + nn + 1));
a = (double) (4* nn * nn) / a;
a = a*(double) ((2*nn + 1) * (2*nn - 1));
a = Math.sqrt(a);
b = (double) (m - m1) * .5;
d = 0.;
if (nn != 1)
{
b = (double) (2*nn * (nn - 1));
b = (double) ((2*m - mm) * nn * (nn - 1) - mm * n * (n + 1) + mm * n1 *
(n1 + 1)) / b;
d = (double) (4*(nn - 1) * (nn - 1));
d = d*(double) ((2*nn - 3) * (2*nn - 1));
d = (double) ((nn - mm - 1) * (nn + mm - 1) * (n1 - n + nn - 1)) / d;
d = d*(double) ((n - n1 + nn - 1) * (n + n1 - nn + 2) * (n + n1 + nn));
d = Math.sqrt(d);
}
c = a * (b * c1 - d * c2);
c2 = c1;
c1 = c;
cu[nn-1] = c;
}
if (nnf > nnm)
{
c=direct(n, m, n1, m1, nnu, mm, c);
cd[nnu-1] = c;
if (nnu != nnm + 1)
{
c2 = 0.;
c1 = c;
for (nn = nnu - 1; nn >= nnm + 1; --nn)
{
a = (double) ((nn - mm + 1) * (nn + mm + 1) * (n1 - n + nn + 1));
```

```
a = a*(double) ((n - n1 + nn + 1) * (n + n1 - nn) * (n + n1 + nn + 2));
a = (double) (4*(nn + 1) * (nn + 1)) / a;
a = a*(double) ((2*nn + 1) * (2*nn + 3));
a = Math.sqrt(a);
b = (double) (2*(nn + 2) * (nn + 1));
b = (double) ((2*m - mm) * (nn + 2) * (nn + 1) - mm * n * (n + 1) + mm
* n1 * (n1 + 1)) / b;
d = (double) (4*(nn + 2) * (nn + 2));
d = d*(double) ((2*nn + 5) * (2*nn + 3));
d = (double) ((nn + mm + 2) * (nn - mm + 2) * (n1 - n + nn + 2)) / d;
d = d*(double) ((n - n1 + nn + 2) * (n + n1 - nn - 1) * (n + n1 + nn +
3));
d = Math.sqrt(d);
c = a * (b * c1 - d * c2);
c2 = c1;
c1 = c;
cd[nn-1] = c;
}
}
}
}
for (nn = nn1; nn <= nnf; ++nn)
{
if (nn <= nnm)
{
if(nn-1<0)
{
gg[mind-1][nn] = cu[0];
}
else
{
gg[mind-1][nn] = cu[nn-1];
}
}
if (nn > nnm)
{
gg[mind-1][nn] = cd[nn-1];
}
}
}
}
}
}
else
{
System.out.println("ERROR IN SUBROUTINE CCG");
System.exit(0);
}
return 0;
}

public double direct(int n, int m, int n1, int m1, int nn, int mm,
double c)
{
int i;
int i1;
double f[]=new double[900];
```

```
f[0] = 0.0;
f[1] = 0.0;
for (i = 3; i<= 900; ++i)
{
i1 = i - 1;
f[i - 1] = f[i1 - 1] + Math.log((double) i1) * 0.5;
}
c = f[n * 2] + f[n1 * 2] + f[n + n1 + m + m1] + f[n + n1 - m - m1];
c = c - f[(n + n1) * 2] - f[n + m] - f[n - m] - f[n1 + m1] - f[n1 -
m1];
c = Math.exp(c);
return c;
}

public double ccgin(int n, int n1, int m, int mm, double g)
{
int i;
int i1;
int k;
int  l1;
int  m1;
int  l2;
int  l3;
int  n2;
int  m2;
int  m12;
int  n12;
double a;
double f[]=new double[900];

f[0] = 0.0;
f[1] = 0.0;
for (i = 3; i<= 900; ++i)
{
i1 = i - 1;
f[i - 1] = f[i1 - 1] + Math.log((double) i1) * 0.5;
}
m1 = mm - m;
if (n >= Math.abs(m) && n1 >= Math.abs(m1) && Math.abs(mm) <= n + n1)
{
if (Math.abs(mm) <= Math.abs(n - n1))
{
l1 = n;
l2 = n1;
l3 = m;
if (n1 > n)
{
k = n;
n = n1;
n1 = k;
k = m;
m = m1;
m1 = k;
}
n2 = 2*n;
m2 = 2*m;
```

```
n12 = 2*n1;
m12 = 2*m1;
g = ssign[n1 + m1] * Math.exp(f[n + m] + f[n - m] + f[n12] + f[n2 - n12
+ 1] - f[n2 + 1] - f[n1 + m1] - f[n1 - m1] -
f[n - n1 + mm] - f[n - n1 - mm]);
n = l1;
n1 = l2;
m = l3;
return g;
}
a = 1.;
l1 = m;
l2 = mm;
if (mm < 0)
{
mm = -(mm);
m = -(m);
m1 = -m1;
a = ssign[mm + n + n1];
}
g = a * ssign[n + m] * Math.exp(f[2*(mm) + 1] + f[n +n1 - mm] + f[n +
m] + f[n1 + m1] - f[n + n1 + mm + 1] -
f[n - n1 + mm] - f[-(n) + n1 + mm] - f[n - m] - f[n1 - m1]);
m = l1;
mm = l2;
return g;
}
else
{
System.out.println("ERROR IN SUBROUTINE CCGIN");
System.exit(0);
}
return 0.0;
}

public int sarea(double d)
{
double e;
double r;
if (d < 1)
{
e = Math.sqrt(1. - d * d);
r = (Math.pow(d, 2.0/3.0) + Math.pow(d,1.0/3.0) * Math.asin(e) / e) *
0.5;
r = Math.sqrt(r);
rat = 1.0 / r;
return 0;
}
e = Math.sqrt(1.0 - 1.0 / (d * d));
r = (Math.pow(d, 2.0/3.0) * 2.0 + Math.pow(d, (-4.0/3.0)) * Math.log((e
+ 1.0) / (1.0 - e)) / e) * 0.25;
r = Math.sqrt(r);
rat = 1.0 / r;
return 0;
}

public int sareac(double eps)
```

```
{
rat = Math.pow(1.5 / eps, 1.0/3.0);
rat = rat/Math.sqrt((eps + 2.0) / (eps * 2.0));
return 0;
}

public int gauss(int n, int ind1, int ind2,double z[], double w[])
{
int i;
int  j;
int  k;
int  m;
int  ind;
int  niter;
int boolFlag=1;
double a;
double  b;
double  c;
double  f;
double  x=0.0;
double  dj;
double  pa;
double  pb;
double  pc;
double  zz;
double check;

a = 1.;
b = 2.;
c = 3.;
ind = n % 2;
k = n / 2 + ind;
f = (double) (n);
for (i= 1; i <= k; ++i)
{
m = n + 1 - i;
if (i == 1)
{
x = a - b / ((f + a) * f);
}
if (i == 2)
{
x = (z[n-1] - a) * 4. + z[n-1];
}
if (i == 3)
{
x = (z[n - 2] - z[n-1]) * 1.6 + z[n - 2];
}
if (i > 3)
{
x = (z[m] - z[m + 1]) * c + z[m + 2];
}
if (i == k && ind == 1)
{
x = 0.;
}
niter = 0;
```

```
check = 1e-16;
do{
pb = 1.;
niter=niter+1;
if (niter > 100)
{
check = check*10.;
}
pc = x;
dj = a;
for (j = 2; j <= n; ++j)
{
dj = dj+a;
pa = pb;
pb = pc;
pc = x * pb + (x * pb - pa) * (dj - a) / dj;
}
pa = a / ((pb - x * pc) * f);
pb = pa * pc * (a - x * x);
x = x - pb;
if (Math.abs(pb) > check * Math.abs(x))
{
boolFlag=1;
}
else
{
boolFlag=0;
}
}while(boolFlag==1);
z[m-1] = x;
w[m-1] = pa * pa * (a - x * x);
if (ind1 == 0)
{
w[m-1] = b * w[m-1];
}
if (!(i == k && ind == 1))
{
z[i-1] = -z[m-1];
w[i-1] = w[m-1];
}
}
if (ind2 == 1)
{
for (i = 1; i <= k; ++i)
{
zz = -z[i-1];
}
}
if (ind1 != 0)
{
for (i = 1; i<= n; ++i)
{
z[i-1] = (a + z[i-1]) / b;
}
}
return 0;
}
```

```
public int distrb(int nnk, double yy[], double wy[], int ndistr, double
aa, double bb, double gam, double r1, double r2,double pi)
{
int i;
double g, x, y, b2, da, xi, dab, sum;

if (ndistr == 1)
{
b2 = (1. - bb * 3.) / bb;
dab = 1. / (aa * bb);
for (i = 1; i <= nnk; ++i)
{
x = yy[i-1];
x = Math.pow(x, b2) * Math.exp(-x * dab);
wy[i-1] *= x;
}
}
if (ndistr == 2)
{
for (i = 1; i <= nnk; ++i)
{
x = yy[i-1];
y = x - aa;
y = Math.exp(-y * y * .5 / bb);
wy[i-1] *= y;
}
}
if (ndistr == 3)
{
da = 1. / aa;
for (i = 1; i <= nnk; ++i)
{
x = yy[i-1];
y = Math.log(x * da);
y = Math.exp(-y * y * .5 / bb) / x;
wy[i-1] *= y;
}
}
sum = 0.;
for (i = 1; i <= nnk; ++i)
{
sum = sum+wy[i-1];
}
sum = 1. / sum;
for (i = 1; i <= nnk; ++i)
{
wy[i-1] = wy[i-1]*sum;
}
g = 0.;
for (i = 1; i<= nnk; ++i)
{
x = yy[i-1];
g = g+x * x * wy[i-1];
}
reff = 0.0;
for (i = 1; i <= nnk; ++i)
```

```
{
x = yy[i-1];
reff = reff+x * x * x * wy[i-1];
}
reff /= g;
veff = 0.;
for (i = 1; i <= nnk; ++i)
{
x = yy[i-1];
xi = x - reff;
veff = veff + xi * xi * x * x * wy[i-1];
}
veff = veff / (g * reff * reff);
return 0;
}

public int hovenr(int l1, double a1[], double a2[], double a3[], double
a4[], double b1[], double b2[])
{
int i;
int  l;
int ll;
int kontr=0;

double c;
double c1;
double  c2;
double  c3;
double  cc;
double  dl;
double aa1;
double  aa2;
double  aa3;
double  aa4;
double  bb1;
double  bb2;
double  ddl;

for (l = 1; l <= l1; ++l)
{
kontr = 1;
ll = l - 1;
dl = (double) ll * 2. + 1.;
ddl = dl * .48;
aa1 = a1[l-1];
aa2 = a2[l-1];
aa3 = a3[l-1];
aa4 = a4[l-1];
bb1 = b1[l-1];
bb2 = b2[l-1];
if (ll >= 1 && Math.abs(aa1) >= dl)
{
kontr = 2;
}
if (Math.abs(aa2) >= dl)
{
kontr = 2;
```

```
}
if (Math.abs(aa3) >= dl)
{
kontr = 2;
}
if (Math.abs(aa4) >= dl)
{
kontr = 2;
}
if (Math.abs(bb1) >= ddl)
{
kontr = 2;
}
if (Math.abs(bb2) >= ddl)
{
kontr = 2;
}
if (kontr == 2)
{
System.out.println("TEST FOR VAN DER MEE & HOVENIER IS NOT SATISFIED,
L="+ll);
}
c = -0.1;
for (i = 1; i <= 11; ++i)
{
c=c+ 0.1;
cc = c * c;
c1 = cc * bb2 * bb2;
c2 = c * aa4;
c3 = c * aa3;
if ((dl - c * aa1) * (dl - c * aa2) - cc * bb1 * bb1 <= -1e-4)
 {
kontr = 2;
}
if ((dl - c2) * (dl - c3) + c1 <= -1e-4)
{
kontr = 2;
}
if ((dl + c2) * (dl - c3) - c1 <= -1e-4)
{
kontr = 2;
}
if ((dl - c2) * (dl + c3) - c1 <= -1e-4)
{
kontr = 2;
}
if (kontr == 2)
{
System.out.println("TEST FOR VAN DER MEE & HOVENIER IS NOT SATISFIED, L
& A = \n"+ll+","+c);
}
}
}
if (kontr == 1)
{
System.out.println("TEST FOR VAN DER MEE & HOVENIER IS SATISFIED");
}
```

```
return 0;
}

public int matr(double a1[], double a2[], double a3[],double a4[],
double b1[], double b2[], int lmax, int npna)
{
int l;
int n;
int i1;
int l1;
int l1max;
double p;
double u;
double f2;
double f3;
double d6;
double p1;
double p2;
double p3;
double p4;
double da;
double db;
double f11;
double f12;
double f22;
double f33;
double f34;
double f44;
double dl;
double dn;
double tb;
double dl1;
double pl1=0.0;
double pl2=0.0;
double pl3=0.0;
double pl4=0.0;
double pp1;
double pp2;
double pp3;
double pp4;
double taa;
int count=0;
double temp=0.0;

n = npna;
dn = 1. / (double) (n - 1);
da = Math.acos(-1.) * dn;
db = dn * 180.;
l1max = lmax + 1;
for (l1 = 1; l1 <= l1max; ++l1)
{
l = l1 - 1;
}
tb = -db;
taa = -da;
d6 = Math.sqrt(6.) * .25;
for (i1 = 1; i1 <= n; ++i1)
```

```
{
taa += da;
tb += db;
u = Math.cos(taa);
f11 = 0.;
f2 = 0.;
f3 = 0.;
f44 = 0.;
f12 = 0.;
f34 = 0.;
p1 = 0.;
p2 = 0.;
p3 = 0.;
p4 = 0.;
pp1 = 1.;
pp2 = (u + 1.) * .25 * (u + 1.);
pp3 = (1. - u) * .25 * (1. - u);
pp4 = d6 * (u * u - 1.);
L400:
for (l1 = 1; l1 <= l1max; ++l1)
{
l = l1 - 1;
dl = (double) l;
dl1 = (double) l1;
f11 = f11 + a1[l1-1] * pp1;
f44 = f44 + a4[l1-1] * pp1;
if (l != lmax)
{
pl1 = (double) (2*l + 1);
p = (pl1 * u * pp1 - dl * p1) / dl1;
p1 = pp1;
pp1 = p;
}
if (l < 2)
{
continue L400;
}
f2 = f2 + (a2[l1-1] + a3[l1-1]) * pp2;
f3 = f3 + (a2[l1-1] - a3[l1-1]) * pp3;
f12 = f12 + b1[l1-1] * pp4;
f34 = f34 + b2[l1-1] * pp4;
if (l == lmax)
{
continue L400;
}
pl2 = (double) (l * l1) * u;
pl3 = (double) (l1 * (l * l - 4));
pl4 = 1. / (double) (l * (l1 * l1 - 4));
p = (pl1 * (pl2 - 4.) * pp2 - pl3 * p2) * pl4;
p2 = pp2;
pp2 = p;
p = (pl1 * (pl2 + 4.) * pp3 - pl3 * p3) * pl4;
p3 = pp3;
pp3 = p;
p = (pl1 * u * pp4 - Math.sqrt((double) (l * l - 4)) * p4) /
Math.sqrt((double) (l1 * l1 - 4));
p4 = pp4;
```

```
pp4 = p;
}
if(i1==1)
{
temp=f11;
}
f22 = (f2 + f3) * .5;
f33 = (f2 - f3) * .5;
f22 = f22 / f11;
f33 = f33 / f11;
f44 = f44 / f11;
f12 = -f12 / f11;
f34 = f34 / f11;
ss11[count]=(f11/temp);
ss12[count]=f12;
ss33[count]=f33;
ss34[count]=f34;
count++;
}
return 0;
}
}
class gspBlock{
double rt11[][][];
double rt12[][][];
double rt21[][][];
double rt22[][][];
double it11[][][];
double it12[][][];
double it21[][][];
double it22[][][];
gspBlock(int npn4,int npn6)
{
rt11=new double[npn6][npn4][npn4];
rt12=new double[npn6][npn4][npn4];
rt21=new double[npn6][npn4][npn4];
rt22=new double[npn6][npn4][npn4];
it11=new double[npn6][npn4][npn4];
it12=new double[npn6][npn4][npn4];
it21=new double[npn6][npn4][npn4];
it22=new double[npn6][npn4][npn4];
}
protected void finalize() throws Throwable
{
super.finalize();
}
}
class tmatBlock
{
double d1[][] ;
double d2[][] ;
tmatBlock(int npng2,int npn1)
{
d1 = new double[npng2][npn1];
d2= new double[npng2][npn1];
}
protected void finalize() throws Throwable
```

```
{
super.finalize();
}
}
class trtiBlock
{
static double tr1[][];
static double ti1[][];
static double tqi[][] ;
static double tqr[][] ;
trtiBlock(int npn2)
{
tr1=new double[npn2][npn2];
ti1=new double[npn2][npn2];
tqi = new double[npn2][npn2];
tqr = new double[npn2][npn2];
}
protected void finalize() throws Throwable
{
super.finalize();
}
}
class rirgigBlock
{
static double r11[][] ;
static double r12[][] ;
static double r21[][] ;
static double r22[][] ;
static double i11[][] ;
static double i12[][] ;
static double i21[][] ;
static double i22[][] ;
static double rg11[][] ;
static double rg12[][] ;
static double rg21[][] ;
static double rg22[][] ;
static double ig11[][] ;
static double ig12[][] ;
static double ig21[][] ;
static double ig22[][] ;
rirgigBlock(int npn1)
{
r11 = new double[npn1][npn1];
r12 = new double[npn1][npn1];
r21 = new double[npn1][npn1];
r22 = new double[npn1][npn1];
i11 = new double[npn1][npn1];
i12 = new double[npn1][npn1];
i21 = new double[npn1][npn1];
i22 = new double[npn1][npn1];
rg11 = new double[npn1][npn1];
rg12 = new double[npn1][npn1];
rg21 = new double[npn1][npn1];
rg22 = new double[npn1][npn1];
ig11 = new double[npn1][npn1];
ig12 = new double[npn1][npn1];
ig21 = new double[npn1][npn1];
```

```
ig22 = new double[npn1][npn1];
}
protected void finalize() throws Throwable
{
super.finalize();
}
}
class cbassBlock
{
static double j[][];
static double y[][];
static double jr[][];
static double ji[][];
static double dj[][];
static double dy[][];
static double djr[][];
static double dji[][];
cbassBlock(int npng2,int npn1)
{
j=new double[npng2][npn1];
y=new double[npng2][npn1];
jr=new double[npng2][npn1];
ji=new double[npng2][npn1];
dj=new double[npng2][npn1];
dy=new double[npng2][npn1];
djr=new double[npng2][npn1];
dji=new double[npng2][npn1];
}
}
```

/********End of nonSphericalClass.java ********/

Appendix F

Grenfell's Method

Mie (1908) theory [1 - 6, 100 - 102] is an exact analytical solution of Maxwell equations for spherical particles, and the most widely used numerical method for light scattering calculations. Although Mie theory is not applicable for nonspherical particles, in many models for light scattering computations when the practical particle shapes and dispersion of shapes are unknown, such nonspherical shapes are represented by equivalent spheres [294]. This also enables the substantial reduction of the computation time by decomposing the scattering problem of nonspherical particles into that of spherical one. Such models are also very much important for calculating the scattering properties of nonspherical shapes having very large size parameter. In such cases conventional methods like T-matrix, FDTD etc. [238, 358, 359, 361] fail to give a convergent solution. In our case also T-matrix code presented in this thesis is capable of calculating light scattering from spheroids and cylinders accurately within size parameter value 20. Therefore to compare the experimental results with hexagonal icelike crystals we used the model described by Grenfell and Warren [294], Neshyba et al [295] and Grenfell et al [296].

Generally, a nonspherical particle is represented by a sphere of same volume, V or by a sphere of same surface area, A. In both the cases the number of equivalent particles is the same as that of the real particles [295, 359]. If A be the projected area a nonspherical particle, the radius of the projected surface area equivalent sphere is given by,

$$r_A = \left(\frac{A}{4\pi}\right)^{\frac{1}{2}} \qquad \text{(E1)}$$

Similarly if A be the projected volume a nonspherical particle, the radius of the projected volume equivalent sphere is given by,

$$r_V = \left(\frac{3V}{4\pi}\right)^{\frac{1}{3}} \qquad \text{(E2)}$$

In the model under consideration represents a nonspherical particle of volume V and surface area A, by a cloud of spheres having same volume to surface area ratio, V/A [294 - 296, 359]. It is noteworthy to mention that in this model the number of equivalent particles is different from that of the original particles and depends on the aspect ratio of the particle under consideration. From equations E1 and E2 the radius of such equal-V/A sphere is given by,

$$r_{VA} = 3\frac{V}{A} \qquad \text{(E3)}$$

Now if n is the number of real particles and n_s be the number of equivalent spheres, it follows,

$$n_s \times \frac{4}{3}\pi r_{VA}^3 = n \times V \qquad \text{(E4)}$$

$$\text{or} \qquad \frac{n_s}{n} = \frac{3V}{4\pi r_{VA}^3} \qquad \text{(E5)}$$

Here n_s may not be an integer. In the calculations of hemispheric reflectance and transmittance, this model where both projected area and volume is conserved was proved to be superior to the other two representations where either the projected area or the volume is conserved [295, 296].

Application to hexagonal columns:

The area of the hexagon and each of the rectangular face of a hexagonal column is given by, $\frac{3\sqrt{3}}{2} \times r^2$ and $h \times r$ where h is the height of the column and

r is the length of one of the hexagon sides. Thus total surface area of a hexagonal column is,

$$A_H = \frac{3\sqrt{3}}{2} r^2 + 6hr \tag{E6}$$

Again the volume of a hexagonal column is

$$V_H = \frac{3\sqrt{3}}{2} r^2 h \tag{E7}$$

Now,

$$\frac{V_H}{A_H} = \frac{\sqrt{3}rh}{2\sqrt{3}r + 4h} \tag{E8}$$

Defining $\Gamma = \frac{h}{2r}$ and using equation E3 and E5,

$$r_{VA} = \frac{3\sqrt{3}r\Gamma}{\sqrt{3}r + 4\Gamma} \tag{E9}$$

$$\frac{n_s}{n} = \frac{\left(4\Gamma + \sqrt{3}\right)^3}{36\Gamma^2} \tag{E10}$$

and

$$V_H = 3\sqrt{3}r^3\Gamma \tag{E11}$$

Thus the number of equivalent spheres per real hexagonal column depends only on length to diameter ratio (aspect ratio). Notably, this method has limitations to be effectively applied for the calculation of light scattering properties of fractal soot aggregates [362]. However, this model has been extensively applied for various ice crystal shapes e.g. circular cylinder, hexagonal prisms, plates. Moreover its accuracy was tested for a spectrum range from 0.2 - 50 μm over the entire range of ice crystal radii and ice water path and it was found that the errors in simulating the hemispheric reflectance and transmittance do not exceed 5% [295, 296].

List of Publications

(a) Papers published/accepted for publication in referred journals

1. **Gogoi, A.**, Choudhury A., and Ahmed, G. A. Mie scattering computation of spherical particles with very large size parameters using an improved program with variable speed and accuracy, *J. Modern Optics* **57** (21), 2192–2202, 2010.
2. Mazumder, N., **Gogoi, A.**, Kalita, R. D., Ahmed, G. A., Buragohain, A. K. and Choudhury A. Luminescence studies of fresh water diatom frustules, *Indian J. Phys.* **84** (6), 665-669, 2010.
3. Roy, S., **Gogoi, A.**, and Ahmed, G. A. Size Dependent Optical Characterization of semiconductor particle: CdS embedded in polymer matrix, *Indian J. Phys.* **84** (10), 1401-1407, 2010.
4. **Gogoi, A.**, Buragohain, A. K., Choudhury A., Ahmed, G. A., Laboratory measurements of light scattering by tropical fresh water diatoms, *J. Quant. Spec. Rad. Trans.* **110**, 1566-1578, 2009.
5. **Gogoi, A.**, Borthakur, L. J., Choudhury, A., Stanciu, G. A. and Ahmed, G. A. Detector array incorporated optical scattering instrument for nephelometric measurements on small particles, *Meas. Sci. Technol.* **20**, 095901 (10pp), 2009.
6. **Gogoi, A.**, Choudhury A., Stanciu, G. A. and Ahmed, G. A. Construction of a Multidetector Array Incorporated Laser Based Scattering System for Ultrafine TiO_2 Characterization, *Journal of Optics*, **38** (2), 67-74, 2009.
7. **Gogoi, A.**, Ahmed, G. A. and Choudhury A. Nanoparticle size characterization by laser light scattering, *Indian J. Phys.* **83** (4), 473-477, 2009.
8. **Gogoi, A.**, Ahmed, G. A. A T-matrix approach for the morphological characterization of spherical nanoparticles using laser, *Indian J. Phys.* **82** (5), 147-150, 2008.

9. **Gogoi, A.**, Das, G., Karak, N., and Choudhury A. and Ahmed, G. A. Measurement of angular scattering function and degree of linear polarization of bentonite clay particles embedded in cylindrical epoxy matrix, *Accepted for publication in: AAPP | Physical, Mathematical, and Natural Sciences*, 2011.

(b) Papers in conference proceedings

1. **Gogoi, A.**, Rajkhowa, P., Choudhury, A. and Ahmed, G. A. Development of a software package for the analysis of electromagnetic scattering from small particles. 58-61; Editors: Muinonen, K., Penttilä, A., Lindqvist, H., Nousiainen, T. and Videen, G. Proc. of 12th Electromagnetic and Light Scattering Conference, June 28-July 2, 2010, University of Helsinki, Finland.
2. **Gogoi, A.**, Choudhury, A. and Ahmed, G. A. Design considerations of a detector array incorporated laser based scattering system for particle characterization, in *Proc. of XXXIII Optical Society of India (OSI) Symposium 2007*, edited by: P P Sahu and P Deb (Tezpur University, Tezpur, Assam, India, December 18-20, 2007), 382 - 384.

(c) Book chapters

1. Ahmed, G. A., Buragohain, A. K., Nath, P. P., **Gogoi, A.**, Mazumder, N., Kalita, R. D. and Choudhury, A. FTIR and luminescence studies of nanoporous diatom frustules, in *Photonics and Quantum Structures*, D. Mohanta and G. A. Ahmed (Ed.), Narosa Publishing House, New Delhi, 2011, 83 - 89. (In press).

(d) Oral presentations in conferences/workshops/ symposia

1. **Gogoi, A.**, Das U., Choudhury, A. and Ahmed, G. A. *Optical scattering properties of spherical ZnS semiconductor nanoparticles: influence of cluster formation,* National Conference on Smart Nanostructures, Department of Physics, Tezpur University, Tezpur, Assam, India, January 18 - 20, 2011
   Authors:
2. **Gogoi, A.**, Choudhury, A. and Ahmed, G. A. *Study of the phase function and linear polarization ratio of ice-analogue crystals using a laser and sensor array based light scattering setup,* The International Conference on Fiber Optics and Photonics - PHOTONICS 2010, Indian Institute of Technology Guwahati, India, December 11-15, 2010.
3. **Gogoi, A.**, Saikia D., Choudhury, A. and Ahmed, G. A. *Laboratory measurements on black carbon and graphite particles in single scattering conditions,* National Conference on Laser and Optical Sciences, D.H.S.K. College, Dibrugarh, Assam, India, 11 - 13 October, 2010.
4. **Gogoi, A.**, Rajkhowa, P., Choudhury, A. and Ahmed, G. A. *Development of a software package for the analysis of electromagnetic scattering from small particles,* 12th Electromagnetic and Light Scattering Conference, June 28 - July 2, 2010, University of Helsinki, Finland.
5. **Gogoi, A.**, Buragohain, A. K., Choudhury, A. and Ahmed, G. A. *Light scattering study of tropical fresh water diatoms,* 11th Electromagnetic and Light Scattering Conference, University of Hertfordshire, Hatfield, UK, September 7-12, 2008.
6. **Gogoi, A.**, Choudhury, A. and Ahmed, G. A. *Nanoparticle size characterization by laser light scattering,* Condensed Matter Days, NIT, Rourkela, Orissa, 29-31 August, 2007.
7. **Gogoi, A.** and Ahmed, G. A., *A T-matrix approach for the morphological characterization of spherical nanoparticles using laser,* V th Conference of Physics Academy of North - East, Gauhati University, Guwahati, 1-2 March, 2007.

(e) Poster presentations in conferences/workshops/ symposia

1. **Gogoi, A.**, Rajkhowa, P., Choudhury, A. and Ahmed, G. A., *TUSCAT: an interactive graphical user interface (GUI) for light scattering studies on spherical and nonspherical particles,* 12th Electromagnetic and Light Scattering Conference, June 28 - July 2, 2010, University of Helsinki, Finland.
2. Ahmed, G. A., Mazumder, N.i, Nath, P. P., **Gogoi, A.**, Kalita, R. D., Buragohain, A. K. and Choudhury, A. *FTIR and luminescence studies of nanoporous diatom frustules,* National Seminar on Photonics and Quantum Structures, Department of Physics, Tezpur University, Tezpur 784028, Assam, India, 4 - 6 November, 2009.
3. **Gogoi, A.**, Choudhury, A. and Ahmed, G. A. *Design Considerations of a Detector Array Incorporated Laser Based Scattering System for Particle Characterization,* XXXIII Optical Society of India (OSI) Symposium on Optics and Optoelectronics, Department of Electronics & Communication Engineering, Tezpur University, Tezpur, Assam, 18-20 December, 2007.