Numerical Method for Solving the Boltzmann Equation Using Cubic B-splines

by

Saheba Khurana

B. Sc., Tougaloo College, Mississippi, United States, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies

(Chemistry)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April, 2014

Abstract

A numerical method for solving a one-dimensional linear Boltzmann equation is developed using cubic B-splines. Collision kernels are derived for smooth and rough hard spheres. A complete velocity kernel for spherical particles is shown that is reduced to the smooth, rigid sphere case. Similarly, a collision kernel for the rough hard sphere is derived that depends upon velocity and angular velocity. The exact expression for the rough sphere collision kernel is reduced to an approximate expression that averages over the rotational degrees of freedom in the system. The rough sphere collision kernel tends to the smooth sphere collision kernel in the limit when translational-rotational energy exchange is attenuated. Comparisons between the smooth sphere and approximate rough sphere kernel are made.

Four different representations for the distribution function are presented. The eigenvalues and eigenfunctions of the collision matrix are obtained for various mass ratios and compared with known values. The distribution functions, first and second moments are also evaluated for different mass and temperature ratios. This is done to validate the numerical method and it is shown that this method is accurate and well-behaved.

In addition to smooth and rough hard spheres, the collision kernels are used to model the Maxwell molecule. Here, a variety of mass ratios and initial energies are used to test the capability of the numerical method. Massive tracers are set to high initial energies, representing kinetic energy loss experiments with biomolecules in experimental mass spectrometry. The validity of the Fokker-Planck expression for the Rayleigh gas is also tested. Drag coefficients are calculated and compared to analytic expressions. It is shown that these values are well predicted for massive tracers but show a more complex behaviour for small mass ratios especially at higher energies. The numerical method produced well converged values, even when the tracers were initialized far from equilibrium.

In general this numerical method produces sparse matrices and can be easily generalized to higher dimensions that can be cast into efficient parallel algorithms. Future work has been planned that involves the use of this numerical method for a multi-dimension linear Boltzmann equation.

Preface

The work presented in this thesis has been published or is in preparation for publishing by the author of this thesis, Saheba Khurana, under the guidance and in co-authorship with research supervisor Prof. Mark Thachuk. Saheba Khurana is the first author and the main writer of the published works and those in the process of publishing.

Chapters 1-3 are introductory chapters. Chapter 1 gives the general background of the research. Chapter 2 presents in more comprehensive detail the theoretical background of the numerical method. Details of the computational method are given in Chapter 3. Chapters 4 and 5 present the results, followed by Chapter 6 with details of future work and concluding remarks. Details of the code are given in the Appendix.

The analytical details given in Chapter 2 are expanded versions of material presented in the publications by Saheba Khurana and Mark Thachuk, "A numerical solution of the linear Boltzmann equation using cubic B-splines", *J. Chem. Phys.* **136**, 094103 (2012), "Kernels of the linear Boltzmann equation for spherical particles and rough hard sphere particles", *J. Chem. Phys.* **139**, 164122 (2013) and "Drag coefficients from the Boltzmann equation for hot and massive tracers", (*in preparation*).

In Chapter 3, the methodology has been published as part of publications by Saheba Khurana and Mark Thachuk, "A numerical solution of the linear Boltzmann equation using cubic B-splines", *J. Chem. Phys.* **136**, 094103 (2012), and "Drag coefficients from the Boltzmann equation for hot and massive tracers", (*in preparation*) with a few more modifications presented in the same paper.

Chapters 4 and 5 present the results. Results in Chapter 4 are part of a publication by Saheba Khurana and Mark Thachuk, "A numerical solution of the linear Boltzmann equation using cubic B-splines", *J. Chem. Phys.* **136**, 094103 (2012), and Chapter 5 presents results from "Drag coefficients from the Boltzmann equation for hot and massive tracers", (*in preparation*).

Chapter 6 presents future work that will be completed by Saheba Khurana as a first author, under the supervision of Prof. Mark Thachuk, followed by concluding remarks.

Table of Contents

Appendices

List of Tables

List of Figures

List of Symbols

A	collision frequency factor
α	reduced moment of inertia
$\mathbf{c}$	particle velocity in lab frame
χ	scattering angle
χ	used in moment of inertia expression in Section 2.3
δ	Dirac delta function
Δ	variable used in Eq. 2.38
Δ	spacing between intervals of points for evaluation in Chapter 3
ϵ	azimuthal angle
f	distribution function of tracer
$\mathcal{F}$	distribution function of bath
$\mathbf{F}$	external field
f_{hs}	complete distribution function as given in Eq. (3.1)
$f_{hs,1}$	distribution function expanded about equilibrium function 1 as given in Eq. (3.2)
$f_{hs,2}$	distribution function expanded in the exponent as given in Eq. (3.3)
$f_{hs,3}$	distribution function expanded about exponential tail of equilibrium function as given in Eq. (3.4)
$\mathbf{g}$	relative velocity
γ	mass ratio
I	moment of inertia
$I_0(z)$	modified Bessel function of zeroth order
$\mathbf{J}$	impulse
k	Boltzmann constant
$\hat{\mathbf{k}}$	unit vector
$K(x, x')$	collision kernel
$\tilde{K}_{hs}$	original collision kernel
$K_{hs,1}$	modified collision kernel using $f_{hs,1}$
$K_{hs,2}$	modified collision kernel using $f_{hs,2}$
$K_{hs,3}$	modified collision kernel using $f_{hs,3}$
m	mass

$M(a, b, z)$	Kummer's (confluent hypergeometric) function
n_1	number density
$\mathbf{r}$	distance between particles
σ	diameter of particle
$\sigma(g, \chi)$	scattering cross section
ω	angular velocity
P_l	Legendre polynomials
ϕ	polar angle
$\Phi(x)$	error function
Q, R	expression used with mass ratio
T	temperature of bath
θ	polar angle
μ	reduced mass
$\mathbf{v}$	relative velocity of sphere at points of contact
x	scaled energy
Y_{lm}	spherical harmonic functions
$Z(x)$	energy dependent collision frequency

Acknowledgements

First and foremost I would like to express my sincere gratitude to my supervisor Dr. Mark Thachuk for his invaluable guidance, knowledge and support in completing my thesis work. His expertise has provided me a great deal of knowledge and his advice in editing this thesis is also greatly appreciated.

Thank you to my parents Pam Khurana and B. M. Singh, for providing a loving, nurturing and enriching environment for me to grow up in and being the perfect role models. Thank you for making my education your priority and always supporting me in my endeavours. I am here today only because of your encouragement, love and support and I hope to continue making you proud.

Thank you to Dr. Bernard Shizgal and Dr. Donald Douglas for insightful discussions in regards to the work completed. I also thank my committee members for reading and giving valuable advice on the work in this thesis. My special thanks to the members of the Chemistry Department for creating an enjoyable and inspirational environment, from which I have learnt a tremendous amount.

This thesis work was completed with funding provided by Natural Sciences and Engineering Research Council of Canada (NSERC).

This thesis is dedicated
to
my parents

Chapter 1

Introduction and Motivation

This chapter includes a brief introduction to the problems that are modelled by the Boltzmann equation in kinetic theory. This is followed by an introduction to some of the popular numerical methods of solving the Boltzmann equation in Section 1.1. The motivation behind the work presented in this thesis is given in Section 1.2.

Kinetic theory deals with the evolution of the state of a system and provides the laws of macroscopic phenomena based on the hypothesis of molecular structure of a system and dynamical laws of discrete molecules. Equilibrium statistical mechanics in comparison determines the average state of a system that is in thermal equilibrium. The kinetic theory of gases developed by Maxwell, Boltzmann, Krönig and Clausius is built on some important assumptions such as the systems consist of many discrete particles, and the dynamics and interaction of these particles are known. The dynamical laws are classical and can be used to derive the macroscopic quantities of the system. A gas is described by the distribution function $F(\mathbf{c}, \mathbf{r}, t)$ and calculations are made to obtain average physical quantities [1–7].

A number of problems in kinetic theory lead to a hierarchy of equations. Using Newton's equations of motion, the Liouville equation can be derived that governs the time evolution of the ensemble of particles. Using the Bougoliubov-Born-Green-Kirkwood-Yvon (BBGKY) theory, the Boltzmann equation is derived from the Liouville equation assuming uncorrelated binary collisions among the particles. Using the Chapman-Enskog theory, which treats systems close to equilibrium, leads to the Navier-Stokes equation from the Boltzmann equation. The Boltzmann equation is not applicable when correlated collisions are important and thus can be used for the rarefied or dilute gas regime.

The Boltzmann equation has been used to describe many problems where the motion of particles is given by classical mechanics. It can account for external fields, the particles can be non-interacting or be a chemically reactive mixture of multiple species. Some of these problems include studies of plasmas, ion mobility, rarefied gases, and micro-scale gas flows [8] that are significant in engineering. In another example, the motion of gas molecules was simulated using the Direct Simulation Monte Carlo method and the interaction between a gas molecule and the wall was studied using Molecular Dynamics [9]. The Galerkin numerical method of solving the Boltzmann equation is used to monitor the relaxation phenomena of a binary gas mixture [10]. A model for the Boltzmann equation is proposed that describes chemical reactions that proceed within a set activation energy [11]. Reactions

such as these are significant in combustion phenomena and other fluid dynamic processes in fields of astrophysics, organic chemistry, biophysics, chemical physics, and enzymology. These are just a subset of the wide range of problems that are modelled by the Boltzmann equation.

Experimental studies done by Chen *et al.* [12] and Covey and Douglas [13] monitor the kinetic energy decay of biomolecules at high initial energies. Information from this decay is then used to obtain cross section values for the biomolecules. The cross sections can further give insight into the structure of the molecule and this information can be used to understand the role of biomolecules in living cells. The numerical method given in this thesis models the kinetic energy decay of such experiments where the tracers are initialized at high energies.

Furthermore, studies can also include the effect of external fields, similar to ion mobility problems [2, 7]. The tracers can be influenced by electric and magnetic fields with arbitrary spatial and temporal dependence. To correctly describe this problem, a minimum of two dimensions is required, those being the tracer velocity parallel and perpendicular to the field direction. Similarly, problems involving atoms in intense laser fields can be also be modelled [14]. In such systems the atomic beams can be manipulated to produce ideal laser beams for chemical kinetics and gas-surface interaction studies. Studies such as these also require a multi-dimensional numerical scheme to be used to study the deflection of the laser beam. Therefore there can be a numerical scheme that takes into account field effects, that would account for the multi-dimensional aspect of the problem.

In most numerical methods, the tracers in the system are only slightly perturbed from equilibrium. The distribution functions of the tracers considered in this study, to test this numerical method, are far from equilibrium, and their relaxation to equilibrium is monitored. These initial distribution functions may not even describe a steady state and are anisotropic in velocity and spatial variables. This type of problem is not particularly well suited for the numerical methods that are available currently. In addition to the requirements for problems involving high energy tracers, the expression of the collision kernel in the linear Boltzmann equation is known to contain an analytical cusp that also adds to difficulty in converging results.

With these conditions in mind, a newer method is needed that imposes as few constraints on the distribution function as possible. This type of method should be able to use different types of expressions for the collision integral, to effectively deal with the cusp in the collision kernel and yet is simple and accurate enough to apply to the problems described above. The method presented in this thesis uses B-splines, and also meets the criteria set earlier. In this formulation, cubic B-splines are employed in a collocation scheme that approximate the distribution function as a piece-wise continuous function of cubic polynomials. Therefore, they do not rely on the behaviour of the underlying function except that it can be well-

represented locally by a Taylor series expansion. It will be shown that the B-spline functions are localized in space, and allow the evaluation of the collision integral to be divided into smaller localized domains. Due to this construction, the cusp in the collision integral can also be dealt with.

1.1 Methods of Solving the Boltzmann Equation

Owing to the number of problems described by the Boltzmann equation, there are a number of numerical methods that have been developed to solve it. The only exact solution of this equation is the Maxwellian distribution for the equilibrium case. To solve the Boltzmann equation a number of approximations are made to simplify the expression. These are further used in the numerical methods to solve the systems. These numerical methods and their numerous variations have been very successful in accomplishing the task of solving the Boltzmann equation. At the same time, there is no one method that can be considered a universal method to solve every problem that involves the Boltzmann equation. The methods that have been developed work very well primarily because they are specifically designed for a particular problem with certain limitations and conditions, which in turn provides accurate results. A comparative study, by Kowalczyk *et al.* [15] uses the shock wave problem to compare results obtained using different numerical schemes.

In general, the numerical methods used to solve kinetic equations can be divided into particle-based methods (section 2.1.1) and methods that solve the Boltzmann equation directly for the distribution function (section 2.1.2). These methods include the Direct Simulation Monte Carlo (DSMC) and the Quadrature Discretization Method (QDM). Note, that these two methods are in no way the only methods available. Variations of these and other methods are used to solve the Boltzmann equation. A brief description of these methods provides a general picture of the approach taken by numerical schemes used to solve the Boltzmann equation.

1.1.1 Direct Simulation Monte Carlo (DSMC)

A very popular particle-based numerical method is the Direct Simulation Monte Carlo [16–22]. There have been numerous applications of this numerical method in rarefied gas dynamics and aerospace studies, through its many variations. For example, a hydrogen-oxygen detonation study used DSMC simulations to study molecular level gas detonation, important in applications of the propagation of detonation waves [23]. The flow of a granular gas can be simulated using DSMC [24], which can be used to study more complex problems in granular flow. The uniform shear flow in a granular gas is stationary along with a constant temperature and linear flow velocity when compared to a traditional gas. In aerospace

science the DSMC method has been used to study multiscale aerothermodynamic flows to design optimal space capsules for entry into planetary atmospheres [25]. Similarly, during re-entry a weak plasma is formed in the shock layer and the analysis of this plasma has also been done using DSMC [26].

In principle, DSMC works by coarse-graining the system into simulated (coarse-grained) particles that represent groups of real particles. The simulated particles are assigned positions and velocities and are allowed to move in time. The collision dynamics occurring amongst the coarse-grained particles can be described by a number of different algorithms. Average quantities are obtained by averaging over the states of the coarse-grained particles. The coarseness in this method increases the fluctuations in the obtained quantities. Systems with large Knudsen numbers, such as rarefied gases are particularly well suited for the DSMC. The Knudsen number (K_n) is a measure of the degree of rarefaction of a gas. It is a ratio of the mean free path (average distance travelled by the molecules in between collisions) to the characteristic dimension. In the case where $K_n \to 0$, the distribution is of the Maxwellian form and in the opposite limit, where $K_n \to \infty$, the system falls into the collisionless or free-molecule flow regime.

In rarefied systems, the coarse-graining is physically justified. In principle, the description of ion mobility experiments presents a challenge because the concentration of ions is trace so that each ion moves as a separate entity, that is uninfluenced by the other ions. The movement of such an ion is difficult to coarse-grain. With the fluctuations arising in the DSMC method, ion mobilities would be harder to converge to great accuracy. Also statistical errors can arise in DSMC owing to the different parameters that are used in the simulations [27].

The issues arising from the coarse-grained model can be completely avoided by using fully atomistic classical Molecular Dynamics (MD). Using MD, accurate distribution functions for rotating solid bodies in the presence of constant electric fields, have been obtained [28–30]. In these models, the system reaches a steady state and quantities are obtained at this steady state over many time steps until converged average values are obtained. In the case where time varying fields are used, this approach becomes difficult because in order to obtain accurate enough averages at each time, large ensembles of trajectories are required. Calculations such as those that involve time varying fields become a lot more intense than those where averages have to be obtained for systems with constant fields.

In general, particle-based methods such as those mentioned above are not the best for determining distribution functions for the ion mobility problem when spatially and temporally varying fields are present. These particle based methods are statistical in nature which make it challenging to obtain converged and accurate distribution functions, especially in the tails of the distribution. Due to these limitations, methods that solve for the distribution function directly are expected to perform better for the ion mobility problem.

A number of different methods are available, but one method of particular interest is the Quadrature Discretization Method.

1.1.2 Quadrature Discretization Method (QDM)

The Quadrature Discretization Method [31–40] is based on the quadrature method for numerically evaluating integrals. This method uses a weight function to generate a polynomial basis set from which a quadrature scheme is constructed. The QDM solves integro-differential equations by representing the distribution function at the set of quadrature points which are the roots of these polynomials. The polynomials are orthogonal with respect to the weight function. The number of quadratures equals the number of sets of orthogonal polynomials. Some of the familiar quadratures are based on classical polynomials such as the Legendre, Laguerre, Hermite, Chebyshev, and Jacobi polynomials. Each of these sets have different intervals and weight functions, and can be used for particular systems as needed. The set of orthogonal polynomials can be constructed for any interval and any weight function. A set of expansion coefficients are obtained that are also required to be evaluated using the polynomials and the distribution function that are evaluated at the quadrature points.

The collision integral and all other terms in the Boltzmann equation are then evaluated using this numerical scheme. The convergence of the distribution functions is dependent on the choice of the basis set used. If the basis matches the problem well, spectral convergence is obtained which in turn produces highly accurate distribution functions, whereas, if the basis does not match the problem well, difficulties can arise. As an example, if the basis set chosen has exponentially decaying tails at large distances but the distribution function does not, they will not accurately represent the distribution function.

The QDM has also proven its versatility. For example, variations of this method have been used to solve the Fokker-Planck equation [41]. In this study, eigenvalues and eigenfunctions were obtained for the Fokker-Planck equation which was used to model the time evolution of the probability density function. In the study of Shizgal and Blackmore [36], the Boltzmann equation describes the perturbation from equilibrium that occurs due to the loss of energetic atoms in a planetary atmosphere. Radiative transfer equations obtained from the Boltzmann equation describing phonon transport have also been solved using a variation of the QDM [42]. It has been used to solve the Boltzmann equation and also has been used as a numerical method to solve the Schrödinger equation [32]. In that case, the basis functions are used to expand the wave function. These are just a few examples of problems where variations of the QDM have been used to solve the Boltzmann equation or its approximations.

In ion mobility, it is known that the tails of the distribution function decay algebraically

at large velocities [43]. The cusp in the collision kernel can cause difficulties when integrating basis functions over the entire range of the integral. This cusp can cause inaccuracies and insufficiently converged values of the distribution function. The boundary conditions used in the QDM are intrinsic, that is, they are dependent on the choice of quadrature used, and modifications to these conditions cannot be made. The form of the Boltzmann equation, such as the linear or linearized Boltzmann equation, or any other forms of the collision kernel can be used in the QDM, but are dependent on the generation the quadratures. Therefore, the quadratures are dependent on the form of the expressions used. This ensures an accurate development of the quadrature, that is specific to the problem.

1.2 Motivation

The numerical method using cubic B-splines is presented here that is based on the motion of tracer particles in a bath, where the bath considered is at equilibrium, a system that is described by the Boltzmann equation. The tracer particles are in trace amounts, therefore collisions between tracers are negligible. For this numerical method it is convenient both theoretically and computationally to use the linear Boltzmann equation in the form of a collision kernel. An approach like this has been used in previous studies, such as, describing the mobility of gas phase ions in low fields, the kinetic energy relaxation of hot particles, the effect of velocity-changing collisions, or the collisional broadening of spectral lines [44–51]

Spline functions have been used in many fields, but seem to not have been pursued for the Boltzmann equation. In the literature, the only reference by Siewert [52], uses Hermite cubic splines to solve the Chapman-Enskog equations for viscosity and heat transfer. In that study, Burnett functions were computed that were based on rigid sphere collisions and the linear Boltzmann equation. The work presented in this thesis develops a B-spline method for a reduced dimensional problem using collision kernels for the smooth and rough hard sphere.

The distribution function can be expanded in several different ways, of which four different formulations are introduced, and the relative merits of each will be discussed. A complete Boltzmann equation can involve up to ten dimensions (three each for velocity, position and angular momenta, and one for time). The method given here can be easily extended to higher dimensions that allow fine details to be studied, such as spatial fluctuations induced by arbitrary external electric and magnetic fields. As discussed in Chapter 6, the extension of the numerical method to higher dimensions will be the aim of future studies.

The linear kernels derived in Chapter 2 account to some extent for the internal states. A classical description is used which allows the resulting formulation to be used at temperatures where quantal rotational effects are negligible. Collision kernels come in different

forms and the expressions can be exact or approximate. The smooth hard sphere model considered in this study uses the well known Wigner-Wilkins kernel [53], which accounts only for translational velocity energy exchange between the tracer and bath. In the kernels derived, the tracers are subject to rotations and vibrations whereas in most cases the collision kernels only describe the translational velocity changes that occur due to collisions. Subsequently, a kernel expression is derived for the rough hard sphere model, where the rotational-translational energy exchange is taken into consideration. It is shown that within certain limits, the rough hard sphere kernel expression can be reduced to the smooth hard sphere expression. The Maxwell molecule model introduces a repulsive potential, namely $1/r^4$, via which the tracers must interact. This also introduces a more realistic cross section expression that has to be evaluated as part of the collision kernel.

Chapter 2

The Boltzmann Equation and Derivation of Collision Kernels

This chapter presents the theory behind the formulation of the numerical method. Section 2.1 gives the theory behind the Boltzmann equation and the assumptions that are made to make it valid for the models considered here. Section 2.2 gives a detailed derivation of the smooth hard sphere kernel followed by a derivation of the rough hard sphere kernel in Section 2.3. This leads to Section 2.4 where the derivation for the approximate rough hard sphere kernel is given along with the assumptions that have been used in the derivation. Section 2.5 gives a detailed discussion of the kernels, their form and any numerical difficulties that may occur. Section 2.6 gives a summary of the kernel expressions and collision frequencies for all three models discussed.

2.1 The Boltzmann Equation

Consider an ensemble made up of multiple systems, and each system is represented by a point in phase space. Due to the large number of systems within this ensemble, the number of points become quite dense in the phase space. The distribution can thus be described by a density function, that is a continuous function of the position and momenta coordinates. Normalizing this function density function gives the probability density function, which can be denoted by $F_N(\mathbf{x}, \mathbf{p}, t)$, where N is the number of systems within the ensemble. The evolution of F_N in time is determined by the motion of each ensemble in the phase space. The change, dF_N, in F_N at any given phase point at time t, results from arbitrary, infinitesimal changes in $\mathbf{x}$ and $\mathbf{p}$. This can be written as [3]

$$dF_N = \frac{\partial F_N}{\partial t}dt + \sum_{i=1}^{N} \frac{\partial F_N}{\partial \mathbf{x}_i}d\mathbf{x}_i + \sum_{i=1}^{N} \frac{\partial F_N}{\partial \mathbf{p}_i}d\mathbf{p}_i \ . \tag{2.1}$$

Equation (2.1) must be valid for all infinitesimal changes, so that this change follows the trajectory of the system in phase space. Therefore, $(d\mathbf{x}_i/dt) = \dot{\mathbf{q}}_i$ and $(d\mathbf{p}_i/dt) = \dot{\mathbf{p}}_i$ and

then Eq. (2.1) can be re-written as [3]

$$\frac{dF_N}{dt} = \frac{\partial F_N}{\partial t} + \sum_{i=1}^{N} \left[\frac{\partial F_N}{\partial \mathbf{x}_i} \dot{\mathbf{x}}_i + \frac{\partial F_N}{\partial \mathbf{p}_i} \dot{\mathbf{p}}_i \right] . \tag{2.2}$$

$\partial F_N/\partial t$ is the local change in F_N, that is, the change at the point $\mathbf{x}$ and dF_N/dt is the total change in F_N along the trajectory in the neighbourhood of $\mathbf{x}$. From this, Liouville's equation is given as [3]

$$\frac{dF_N}{dt} = 0 . \tag{2.3}$$

Equation (2.3) states that along the trajectory of any phase point the probability density remains constant in the neighbourhood of the point $\mathbf{x}$. Since F_N remains constant along the trajectory, any function of F_N also has this property. The Liouville equation contains a large number of degrees of freedom, causing computational difficulties. The Liouville equation leads to the BBGKY (Bogoliubov, Born, H. S. Green, Kirkwood and Yvon) equation. The Boltzmann equation is derived from the BBGKY equation.

Developed by Ludwig Boltzmann (1872) the Boltzmann equation governs the evolution of the distribution function for sets of particles at low density [1, 2, 2–7, 54, 55]. The Boltzmann equation is based on the basic assumption that the collision integral is given by the *stosszahlansatz* [1, 2, 6], in which the distribution function f evolves from one state to the other due to collisions between particles. The *stosszahlansatz* gives rise to the assumption that only binary collisions occur between particles and that these are uncorrelated. This is called the molecular chaos assumption. Consider a binary system composed of tracer and bath particles with distribution functions are f and $\mathcal{F}$, respectively. The Boltzmann equations describing such a binary system in general are

$$\frac{\partial f}{\partial t} + \mathbf{c} \cdot \frac{\partial f}{\partial \mathbf{r}} + \frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \mathbf{c}} = \mathcal{C}[f, \mathcal{F}] + \mathcal{C}[f, f] \tag{2.4}$$

and

$$\frac{\partial \mathcal{F}}{\partial t} + \mathbf{C} \cdot \frac{\partial \mathcal{F}}{\partial \mathbf{r}} + \frac{\mathbf{F}}{M} \cdot \frac{\partial \mathcal{F}}{\partial \mathbf{C}} = \mathcal{C}[\mathcal{F}, \mathcal{F}] + \mathcal{C}[\mathcal{F}, f] , \tag{2.5}$$

in which $\mathbf{F}$ is an external field acting on the system and

$$\mathcal{C}[f_a, f_b] = \int [f_a(\mathbf{c_1}')f_b(\mathbf{c_2}') - f_a(\mathbf{c_1})f_b(\mathbf{c_2})]\mathbf{g}\sigma_{\mathbf{12}}(\mathbf{g}, \chi) \sin\chi \mathbf{d}\chi \mathbf{d}\epsilon \mathbf{dc_2} , \tag{2.6}$$

is the collision integral which incorporates the differential scattering cross section $\sigma_{12}(g, \chi)$ between two particles. These integrals account for the interactions between the tracer and bath particles during collisions. The collision integral of Eq. (2.6) can broken down into "loss" and "gain" terms. The expression $\int f_a(\mathbf{c_1})f_b(\mathbf{c_2})\sigma(g, \chi)g \sin\chi d\chi d\epsilon d\mathbf{c_2}$ is the "loss"

term, describing the number of particles that are lost out of the vicinity of shell $\mathbf{c_1}$ due to collisions. Similarly, $\int f_a(\mathbf{c_1}')f_b(\mathbf{c_2}')'\sigma(g,\chi)g\sin\chi d\chi d\epsilon d\mathbf{c_2}'$ is the "gain" term describing the number of particles gained in the velocity shell $\mathbf{c_1}$ due to collisions from particles with initial velocities $\mathbf{c_1}'$ and $\mathbf{c_2}'$. The evolution of the left hand side of Eq. (2.4) is based on external, collisionless factors with $\partial f/\partial t$ describing the time evolution of the distribution function. The term $\mathbf{c}\cdot\partial f/\partial\mathbf{r}$ indicates change in the distribution function due to the movement of particles in and out of the vicinity of the position vector $\mathbf{r}$. The distribution function is assumed to have uniform density, that is, f does not change with $\mathbf{r}$. Due to this assumption this term is ignored. Whereas, the term $\mathbf{F}\cdot\partial f/\partial\mathbf{c}$ describes the change in distribution function due to any external forces $\mathbf{F}$ [1–3, 7]. Therefore, the terms on the left hand side of the Boltzmann equation are not considered in this thesis. This collision kernel on the right hand side of the Boltzmann equation poses more difficulties when it comes to calculations. The collision kernel expressions in the following sections are based only on the terms on the right hand side of the equation.

In Eq. (2.4), the terms $\mathcal{C}[f,\mathcal{F}]$ and $\mathcal{C}[f,f]$ describe the tracer-bath and tracer-tracer interactions. Similarly, in Eq. (2.5), $\mathcal{C}[\mathcal{F},\mathcal{F}]$ and $\mathcal{C}[\mathcal{F},f]$ describe bath-bath and bath-tracer interactions. Because we wish to study tracers in very low concentrations, the $\mathcal{C}[f,f]$ term in Eq. (2.4) and the $\mathcal{C}[\mathcal{F},f]$ term in Eq. (2.5) can be ignored. The bath can then be considered in equilibrium at all times. This results in Eq. (2.4) being reduced to the linear Boltzmann equation.

2.2 Smooth Hard Sphere Collision Kernel

The derivation of the collision kernel for spherical particles has appeared in many forms in literature. For example, the kernel for the linear Boltzmann equation has been derived for a pure gas in Ref. [56]. The first part of the derivation given in the section follows the same idea except it accounts for the mass ratio between the bath and tracer particles. Waldmann [57] expressed the collision integral of the Boltzmann equation using a transition operator formalism which Andersen and Schuler [58] then used to derive the collision kernel for the linear Boltzmann equation for smooth spheres, that also include the spherical component of this kernel. In an analogous approach the kernel was derived for a smooth hard sphere model [59] using a result from Berman [60]. The expression for the general kernel and the first Legendre component of the kernel for the smooth hard sphere model was reported by Kapral and Ross [61], which was based on the work by Mason and Monchick [62] for the linearized Boltzmann equation. The general expansion of the kernel in terms of Legendre components was given by Shizgal and Blackmore [63].

For spherical tracer particles the internal degrees of freedom are ignored. The density carries the spatial dependence of the distribution function completely, therefore only the

velocity dependencies are considered.

The linear Boltzmann equation derived from Eq. (2.4) in the absence of external fields is given as

$$\frac{\partial}{\partial t} f(\mathbf{c}) = \int \left[f(\mathbf{c}') \mathcal{F}^{(0)}(\mathbf{c}_1') - f(\mathbf{c}) \mathcal{F}^{(0)}(\mathbf{c}_1) \right] g \sigma(g, \chi) \sin \chi d\chi d\epsilon d\mathbf{c}_1 \ , \tag{2.7}$$

which is solved for the distribution function. Here, $\mathbf{c}'$ and $\mathbf{c}_1'$ are the pre-collision velocities in an inverse collision where $\mathbf{c}$ and $\mathbf{c}_1$ are the post-collision velocities and variable with subscript "1" refer to the bath particles. Particles with velocities $\mathbf{c}$ have mass m. The relative velocity between the tracer and bath particles is $\mathbf{g} = g\mathbf{\Omega}_\mathbf{g} = \mathbf{c}_1 - \mathbf{c}$, and the scattering angle χ is defined as $\mathbf{g} \cdot \mathbf{g}' = gg' \cos \chi$. In the case of elastic scattering, the scattering cross section $\sigma(g, \chi)$ depends only on the magnitude of $\mathbf{g}$ and the scattering angle and the azimuthal angle ϵ ranges from 0 to 2π, which represents all possible approaches for the initial scattering direction.

In elastic scattering the total momentum and energy are conserved, thus, the dynamics of the elastic collision can be completely expressed in terms of the impulse, $\mathbf{J} = \mu(\mathbf{g} - \mathbf{g}')$, which is imparted by the collision with the reduced mass $\mu = mm_1/(m + m_1)$ and given as

$$\begin{aligned} \mathbf{c}' - \mathbf{c} &= \frac{\mathbf{J}}{m} \ , \\ \mathbf{c}_1' - \mathbf{c}_1 &= -\frac{\mathbf{J}}{m_1} \ , \\ \mathbf{J} &= 2\mu(\mathbf{g} \cdot \hat{\mathbf{k}})\hat{\mathbf{k}} \ , \end{aligned} \tag{2.8}$$

where $\hat{\mathbf{k}}$ is the unit vector that lies along the apse line joining the centers of the colliding particles at the point of closest approach pointing from the bath to the tracer particle. In the case of spherical particles collisions are elastic, therefore, $g = g'$. The equations for the conservation of total momentum and energy are

$$\begin{aligned} m\mathbf{c} + m_1\mathbf{c}_1 &= m\mathbf{c}' + m_1\mathbf{c}_1' \ , \\ mc^2 + m_1c_1^2 &= mc'^2 + m_1c_1'^2 \ , \end{aligned} \tag{2.9}$$

respectively.

The equilibrium distribution function for Eq. (2.7) is

$$\mathcal{F}^{(0)}(\mathbf{c}_1) = n_1 \left(\frac{m_1}{2\pi kT} \right)^{3/2} \exp \left[-\frac{m_1 c_1^2}{2kT} \right] \ , \tag{2.10}$$

where n_1 and T are the number density and temperature of the bath, respectively, and k is

the Boltzmann constant. The goal is to express Eq. (2.7) in the form

$$\frac{\partial}{\partial t} f(\mathbf{c}) = \int K(\mathbf{c}, \mathbf{c}') f(\mathbf{c}') d\mathbf{c}' - Z(\mathbf{c}) f(\mathbf{c}) \ , \tag{2.11}$$

which defines the collision kernel $K(\mathbf{c}, \mathbf{c}')$. Comparing the loss term in Eq. (2.7) with the last term in Eq. (2.11) allows the energy dependent collision frequency $Z(\mathbf{c})$ to be identified as

$$Z(\mathbf{c}) = \int f^{(0)}(\mathbf{c}_1) g \sigma_{int}(g) d\mathbf{c}_1 \ . \tag{2.12}$$

Since $\mathbf{c}_1$ is independent of the scattering angle, the integrals over χ and ϵ can be performed independently, giving the integral cross section

$$\sigma_{int}(g) = \int_0^{2\pi} \int_0^{\pi} \sigma(g, \chi) \sin\chi d\chi d\epsilon \ . \tag{2.13}$$

Since $\mathbf{c}$ is held constant, therefore $d\mathbf{c}_1 = d\mathbf{g}$. It is beneficial to convert the integration over $\mathbf{c}_1$ to one over $\mathbf{g}$ using $c_1^2 = c^2 + g^2 + 2\mathbf{g} \cdot \mathbf{c}$. In order to convert this, a spherical polar coordinate system is considered, one where the polar axis is defined by $\mathbf{c}$, so that $\mathbf{g} \cdot \mathbf{c} = gc\cos\theta$ with θ and ϕ the polar angles specifying the direction of $\mathbf{g}$. Using these definitions and the expressions from Eq. (2.10) for $f^{(0)}$, the integrations over θ and ϕ are performed analytically. By defining the reduced variable $s = \sqrt{m_1/2kT}g$ gives

$$\begin{aligned} Z(\mathbf{c}) &= Z(c) = \frac{n_1}{c\sqrt{\pi}} \left(\frac{2kT}{m_1}\right) \int_0^{\infty} s^2 ds \sigma_{int} \left(\sqrt{\frac{2kT}{m_1}} s\right) \\ &\times \left\{ \exp\left[-\left(s - \sqrt{\frac{m_1}{2kT}} c \right)^2 \right] - \exp\left[-\left(s + \sqrt{\frac{m_1}{2kT}} c \right)^2 \right] \right\} \ , \end{aligned} \tag{2.14}$$

where, $Z(\mathbf{c})$ depends only upon the magnitude of the tracer velocity.

The collision kernel in Eq. (2.11) should be written in a form where the integration is done over $\mathbf{c}'$ rather than $\mathbf{c}_1$ as shown in Eq. (2.7). For this derivation, it is useful to define $\mathbf{C} = \mathbf{c}' - \mathbf{c} = C\Omega_C$, where Ω_C is the angular direction of $\mathbf{C}$ and since $\mathbf{c}$ is being held constant, $d\mathbf{C} = d\mathbf{c}'$. A relationship between g and C can be found, using the definition of $\mathbf{J}$. The first and last relations in Eq. (2.8) define $\mathbf{C} = \mathbf{J}/m$, and using the definition of the scattering angle along with $g = g'$ gives

$$C = \frac{J}{m} = \left(\frac{2\mu}{m}\right) |\mathbf{g} \cdot \hat{\mathbf{k}}| = \left(\frac{\mu}{m}\right) \sqrt{g^2 + g'^2 - 2\mathbf{g} \cdot \mathbf{g}'} = \left(\frac{2\mu}{m}\right) g \sin(\chi/2) \ . \tag{2.15}$$

With a fixed scattering angle χ, this gives

$$\begin{aligned} d\mathbf{c}_1 &= d\mathbf{g} = g^2 dg d\mathbf{\Omega}_g = \left(\frac{m}{2\mu}\right)^2 \frac{1}{\sin^3(\chi/2)} C^2 dC d\mathbf{\Omega}_C = \left(\frac{m}{2\mu}\right)^3 \frac{d\mathbf{C}}{\sin^3(\chi/2)} \quad (2.16) \\ &= \left(\frac{m}{2\mu}\right)^3 \frac{d\mathbf{c}'}{\sin^3(\chi/2)} , \end{aligned}$$

where the expression in Eq. (2.15) has been used to express g in terms of C, and it is also explicitly stated that $\mathbf{\Omega}_C$ and $\mathbf{\Omega}_g$ vary over all space so that the former can be replaced by the latter. Comparing the gain term of the Boltzmann equation in Eq. (2.7) with the first term of the right hand side of Eq. (2.11) and using the relation in Eq. (2.16) gives the kernel as

$$K(\mathbf{c}, \mathbf{c}') = \left(\frac{m}{2\mu}\right)^3 \int f^{(0)}(\mathbf{c}_1') \frac{g\sigma(g,\chi)}{\sin^3(\chi/2)} \sin\chi d\chi d\epsilon . \quad (2.17)$$

Using the expressions in Eq. (2.8), $c_1'^2$ can be expressed within the equilibrium function in terms of $\mathbf{c}$ and $\mathbf{c}'$. This gives $\mathbf{c}_1' = \mathbf{c}' + \mathbf{g}' = \mathbf{c}' + \mathbf{g} - 2(\mathbf{g}\cdot\mathbf{k})\hat{\mathbf{k}}$ which after squaring and simplification gives

$$c_1'^2 = c'^2 + g^2 - 4(\mathbf{g}\cdot\hat{\mathbf{k}})(\mathbf{c}'\cdot\hat{\mathbf{k}}) + 2\mathbf{g}\cdot\mathbf{c}' . \quad (2.18)$$

Taking the dot product of first expression in Eq. (2.8) with $\mathbf{c}'$ gives

$$\mathbf{c}'\cdot(\mathbf{c}'-\mathbf{c}) = \frac{2\mu}{m}(\mathbf{g}\cdot\hat{\mathbf{k}})(\mathbf{c}'\cdot\hat{\mathbf{k}}) . \quad (2.19)$$

The expression for $\mathbf{g}\cdot\mathbf{c}'$ is derived by defining a spherical polar coordinate system where $\hat{\mathbf{k}}$ is along the z-axis and $\mathbf{g}$ lies in the xz plane, giving $\mathbf{g} = g[\sin\psi\hat{\mathbf{x}} + \cos\psi\hat{\mathbf{z}}]$ with ψ being the angle between $\mathbf{g}$ and $\hat{\mathbf{k}}$ [56]. From scattering relations, this angle is related to χ by $\psi = (\pi-\chi)/2$. Since θ (angle between $\hat{\mathbf{k}}$ and $\mathbf{c}'$) and ϕ are the polar angles for $\mathbf{c}'$, $\hat{\mathbf{k}}$ and $\mathbf{g}$ define the scattering plane and ϕ is the azimuthal angle relative to this plane. This angle has exactly the same definition as ϵ over which the scattering cross section is integrated, therefore $\epsilon = \phi$. Combining these expressions together gives $\mathbf{g} = g[\cos(\chi/2)\hat{\mathbf{x}} + \sin(\chi/2)\hat{\mathbf{z}}]$ and $\mathbf{c}' = [\sin\theta\cos\epsilon\hat{\mathbf{x}} + \sin\theta\sin\epsilon\hat{\mathbf{y}} + \cos\theta\hat{\mathbf{z}}]$ giving

$$\begin{aligned} \mathbf{g}\cdot\mathbf{c}' &= gc'\left[\sin\theta\cos(\chi/2)\cos\epsilon + \cos\theta\sin(\chi/2)\right] \quad (2.20) \\ &= \left(\frac{m}{2\mu}\right)\frac{Cc'}{\sin(\chi/2)}\left[\sin\theta\cos(\chi/2)\cos\epsilon + \cos\theta\sin(\chi/2)\right] \\ &= \left(\frac{m}{2\mu}\right)\left[|\mathbf{c}'\times\mathbf{C}|\cot(\chi/2)\cos\epsilon + \mathbf{c}'\cdot\mathbf{C}\right] , \end{aligned}$$

where from Eq. (2.8), $\mathbf{C}$ is parallel to $\hat{\mathbf{k}}$ (since $\mathbf{g}\cdot\hat{\mathbf{k}} \geq 0$ for scattering particles) hence θ is also the angle between $\mathbf{c}'$ and $\mathbf{C}$, so that $\mathbf{c}'\cdot\mathbf{C} = c'C\cos\theta$ and $|\mathbf{c}'\times\mathbf{C}| = c'C\sin\theta$.

Substituting the expressions from Eqs. (2.15), (2.19), and (2.20) into Eqs. (2.17) and (2.18) gives

$$\begin{aligned} K(\mathbf{c},\mathbf{c}') &= \left(\frac{m}{2\mu}\right)^4 |\mathbf{c}'-\mathbf{c}| n_1 \left(\frac{m_1}{2\pi kT}\right)^{3/2} \exp\left[-\frac{m_1}{2kT}c'^2\right] \int_0^{2\pi} d\epsilon \int_0^{\pi} \frac{\sin\chi d\chi}{\sin^4(\chi/2)} \\ &\times \sigma\left[\left(\frac{m}{2\mu}\right)\frac{|\mathbf{c}'-\mathbf{c}|}{\sin(\chi/2)},\chi\right] \exp\left[-\frac{(m+m_1)}{2kT}\right. \\ &\times \left.\left\{|\mathbf{c}'\times\mathbf{c}|\cot(\chi/2)\cos\epsilon - \mathbf{c}'\cdot\left(\mathbf{c}'-\mathbf{c}\right) + \left(\frac{m}{4\mu}\right)\frac{|\mathbf{c}'-\mathbf{c}|^2}{\sin^2(\chi/2)}\right\}\right] . \end{aligned} \tag{2.21}$$

The integration over ϵ can be performed analytically using $\int_0^{2\pi}\exp[z\cos\epsilon]d\epsilon = 2\pi I_0(z)$, where $I_0(z)$ is the modified Bessel function of zeroth order. Completing the integration and making a change of variable using $s = (m/2\mu)^2(m_1/2kT)|\mathbf{c}'-\mathbf{c}|^2\cot^2(\chi/2)$ gives the expression for the kernel as

$$\begin{aligned} K(\mathbf{c},\mathbf{c}') &= \frac{4n_1}{|\mathbf{c}'-\mathbf{c}|}\left(\frac{m}{2\mu}\right)^2\left(\frac{m_1}{2\pi kT}\right)^{1/2}\exp\left[\frac{m}{2kT}c'^2 - \frac{(m+m_1)}{2kT}\left\{\mathbf{c}'\cdot\mathbf{c} + \left(\frac{m}{4\mu}\right)|\mathbf{c}'-\mathbf{c}|^2\right\}\right] \\ &\times \int_0^{\infty} ds e^{-s}\sigma\left[\sqrt{\frac{2kT}{m_1}}\sqrt{\left(\frac{m}{2\mu}\right)^2\left(\frac{m_1}{2kT}\right)|\mathbf{c}'-\mathbf{c}|^2 + s},\right. \\ &\times \left.2\cot^{-1}\left(\sqrt{\frac{2kT}{m_1}\left(\frac{2\mu}{m}\right)\frac{\sqrt{s}}{|\mathbf{c}'-\mathbf{c}|}}\right)\right] I_0\left[2\sqrt{\frac{m_1}{2kT}}\frac{|\mathbf{c}'\times\mathbf{c}|}{|\mathbf{c}'-\mathbf{c}|}\sqrt{s}\right] . \end{aligned} \tag{2.22}$$

These latter two expressions for the kernel agree with those in the literature [58, 62]. In these references the distribution function is expressed as $f(\mathbf{c}) = f^{(0)}(\mathbf{c})\psi(\mathbf{c})$ and the kernels are expressed for the function $\psi(\mathbf{c})$, therefore the kernels in the references are equal to $f^{(0)}(\mathbf{c}')K(\mathbf{c},\mathbf{c}')/f^{(0)}(\mathbf{c})$.

In the case of smooth hard sphere particles, $\sigma(g,\chi) = \sigma_{12}^2/4$ where $\sigma_{12} = (\sigma_1+\sigma)/2$ is the distance between the centers of the two hard spheres at the point of collision. Using the relation $\int_0^{\infty} ds\exp[-s]I_0(\alpha\sqrt{s}) = \exp[\alpha^2/4]$ reduces Eq. (2.22) to

$$\begin{aligned} K_{hs}(\mathbf{c},\mathbf{c}') &= \frac{n_1\sigma_{12}^2}{|\mathbf{c}'-\mathbf{c}|}\left(\frac{m}{2\mu}\right)^2\left(\frac{m_1}{2\pi kT}\right)^{1/2}\exp\left[\frac{m}{2kT}c'^2 - \frac{(m+m_1)}{2kT}\right. \\ &\times \left.\left\{\mathbf{c}'\cdot\mathbf{c} + \left(\frac{m}{4\mu}\right)|\mathbf{c}'-\mathbf{c}|^2 - \left(\frac{\mu}{m}\right)\frac{|\mathbf{c}'\times\mathbf{c}|^2}{|\mathbf{c}'-\mathbf{c}|^2}\right\}\right] . \end{aligned} \tag{2.23}$$

This kernel for the smooth hard sphere can be written in several equivalent forms, with one

particularly compact expression given as [59]

$$K_{hs}(\mathbf{c},\mathbf{c}') = \frac{n_1\sigma_{12}^2}{|\mathbf{c}'-\mathbf{c}|}\left(\frac{m}{2\mu}\right)^2\left(\frac{m_1}{2\pi kT}\right)^{1/2}\exp\left[-\frac{m_1}{2kT}\left(\frac{m}{2\mu}\right)^2\right. \times \left.\left\{|\mathbf{c}'-\mathbf{c}| + \left(\frac{2\mu}{m}\right)\frac{\mathbf{c}'\cdot(\mathbf{c}-\mathbf{c}')}{|\mathbf{c}'-\mathbf{c}|}\right\}^2\right] . \tag{2.24}$$

The full translational velocity dependencies are contained in these kernel expressions. Reduced kernel expressions can be derived from these expressions of the collision kernels, where the new expressions are averaged over various velocity components. One can see that in Eq. (2.22) that $K(\mathbf{c},\mathbf{c}')$ depends only upon c', c, and $y = \cos\Theta = \mathbf{\Omega}_{\mathbf{c}'}\cdot\mathbf{\Omega}_{\mathbf{c}}$, with Θ being the angle between $\mathbf{c}'$ and $\mathbf{c}$. As shown by Blackmore and Shizgal [63], the kernel can be expanded in a series of Legendre polynomials as

$$K(\mathbf{c},\mathbf{c}') = K(c,c',y) = \sum_{l=0}^{\infty}\left(\frac{2l+1}{4\pi}\right)K_l(c,c')P_l(\mathbf{\Omega}_{c'}\cdot\mathbf{\Omega}_c) \tag{2.25}$$

with

$$K_l(c,c') = 2\pi\int_{-1}^{1}K(c,c',y)P_l(y)dy . \tag{2.26}$$

Expanding the distribution function in a series of spherical harmonic functions,

$$f(\mathbf{c}) = \sum_{lm} f_{lm}(c)Y_{lm}(\mathbf{\Omega}_c) , \tag{2.27}$$

with

$$f_{lsm}(c) = \int f(\mathbf{c})Y_{lm}^*(\mathbf{\Omega}_c)d\mathbf{\Omega}_c , \tag{2.28}$$

substituting into Eq. (2.11), multiplying both sides by $Y_{lm}^*(\mathbf{\Omega}_c)$ and then integrating with respect to $\mathbf{\Omega}_c$ gives

$$\frac{\partial}{\partial t}f_{lm}(c) = \sum_{lm'}\int K(\mathbf{c},\mathbf{c}')f_{l'm'}(c')Y_{l'm'}(\mathbf{\Omega}_{c'})Y_{lm}^*(\mathbf{\Omega}_c)c'^2dc'd\mathbf{\Omega}_{c'}d\mathbf{\Omega}_c - Z(c)f_{lm}(c) . \tag{2.29}$$

Replacing the kernel with the expansion in Eq. (2.25) and using the angular momentum addition theorem

$$\sum_{m''=-l''}^{l''}Y_{l''m''}^*(\mathbf{\Omega}_{c'})Y_{l''m''}(\mathbf{\Omega}_c) = \left(\frac{2l''+1}{4\pi}\right)P_{l''}(\mathbf{\Omega}_{c'}\cdot\mathbf{\Omega}_c) , \tag{2.30}$$

allows orthogonality relations between the spherical harmonics to be used, simplifying the

final expression as

$$\frac{\partial}{\partial t} f_{lm}(c) = \int K_l(c,c') f_{lm}(c') c'^2 dc' - Z(c) f_{lm}(c) \ . \tag{2.31}$$

Therefore, the time dependence of the components in $f_{lm}(c)$ are dictated by $K_l(c,c')$ and are also uncoupled. Thus, the higher dimensional integrals of Eq. (2.11) are reduced to a set of uncoupled integro-differential equations with only one dimensional integrations in Eq. (2.31).

If the kinetic energy of the tracer is to be studied, then for any function of c, $h(c)$, using the expansions given above, gives

$$\langle h(c) \rangle = \frac{\int h(c) f(\mathbf{c}) d\mathbf{c}}{\int f(\mathbf{c}) d\mathbf{c}} = \frac{\int_0^\infty h(c) f_{00}(c) c^2 dc}{\int_0^\infty f(c) c^2 dc} \ . \tag{2.32}$$

So, only the spherical component of $f(\mathbf{c})$ is needed and from Eq. (2.31) only the spherical component of the kernel $K_0(c,c')$ is required. Scaled energies, $x = (m/2kT)c^2$ and $x' = (m/2kT)c'^2$ are used to conveniently express the spherical components of the kernel. Including the integrating factor with the definition of the distribution function gives

$$\tilde{f}(x) = \sqrt{\pi x} \left(\frac{2kT}{m} \right)^{3/2} f_{00} \left(\sqrt{\frac{2kTx}{m}} \right) \ , \tag{2.33}$$

$$\tilde{K}(x,x') = \frac{\sqrt{x}}{2} \left(\frac{2kT}{m} \right)^{3/2} K_0 \left(\sqrt{\frac{2kTx}{m}}, \sqrt{\frac{2kTx'}{m}} \right) \ . \tag{2.34}$$

Using these definitions, the equations for the spherical component of the distribution function and for obtaining averages, become

$$\begin{aligned} \frac{\partial}{\partial t} \tilde{f}(x) &= \int_0^\infty \tilde{K}(x,x') \tilde{f}(x') dx' - Z(x) \tilde{f}(x) \ , \qquad (2.35) \\ \langle h(x) \rangle &= \frac{\int_0^\infty h(x) \tilde{f}(x) dx}{\int_0^\infty \tilde{f}(x) dx} \ . \end{aligned}$$

Using the same definitions for $f^{(0)}(c) = (m/2kT)^{3/2} \exp(-mc^2/2kT)$ gives the equilibrium

function as

$$\begin{aligned}\tilde{f}^{(0)}(x) &= \sqrt{\pi x}\left(\frac{2kT}{m}\right)^{3/2} f_{00}^{(0)}\left(\sqrt{\frac{2kTx}{m}}\right) \\ &= 2\pi\sqrt{x}\left(\frac{2kT}{m}\right)^{3/2} f^{(0)}\left(\sqrt{\frac{2kTx}{m}}\right) \\ &= \frac{2}{\sqrt{\pi}}\sqrt{x}e^{-x}\,,\end{aligned} \tag{2.36}$$

with normalization $\int_0^\infty \tilde{f}(x)dx = 1$. Using Eqs. (2.22), (2.26), and (2.34) gives

$$\begin{aligned}\tilde{K}(x,x') &= 2\pi\frac{\sqrt{x}}{2}\left(\frac{2kT}{m}\right)^{3/2}\int_{-1}^{1} dyK\left(\sqrt{\frac{2kTx}{m}},\sqrt{\frac{2kTx'}{m}},y\right) \\ &= A\frac{(1+\gamma)^2}{4\gamma^{3/2}}\sqrt{x}e^{-\gamma x'}\int_{-1}^{1} dy\left(\frac{1}{x'+x-2\sqrt{xx'}y}\right)^{1/2} \\ &\times \exp\left[-(1+\gamma)\left\{-x'+\sqrt{xx'}y+\left(\frac{1+\gamma}{4\gamma}\right)\left(x'+x-2\sqrt{xx'}y\right)\right\}\right] \\ &\times \int_0^\infty dse^{-s}\tilde{\sigma}\left[\sqrt{\frac{2kT}{m}}\left(\left(\frac{(1+\gamma)^2}{4\gamma}\right)\left(x'+x-2\sqrt{xx'}y\right)+s\right)^{1/2},\right. \\ &\times \left.2\cot^{-1}\left(\frac{1\sqrt{\gamma s}}{(1+\gamma)(x'+x-2\sqrt{xx'}y}\right)\right] I_0\left[2\sqrt{\gamma s}\frac{\sqrt{xx'(1-y)^2}}{(x'+x-2\sqrt{xx'}y)^{1/2}}\right]\,,\end{aligned} \tag{2.37}$$

where the scattering cross section has been scaled relative to the hard sphere value so that $\tilde{\sigma}(g,\chi) = 4\sigma(g,\chi)/\sigma_{12}^2$.

The expression for $Z(x)$ is obtained from Eq. (2.14) as

$$Z(x) = \frac{A}{\gamma\sqrt{x}}\int_0^\infty s^2\tilde{\sigma}_{int}\left(\sqrt{\frac{2kT}{m_1}}s\right)\left\{\left[-\left(s-\sqrt{\gamma x}\right)^2\right]-\exp\left[-\left(s+\sqrt{\gamma x}\right)^2\right]\right\}ds\,, \tag{2.38}$$

with the mass ratio defined as $\gamma = m_1/m$, the collision frequency factor as $A = n_1\pi\sigma_{12}^2\sqrt{2kT/(\pi m)}$ and where the cross section has been scaled by the hard sphere value so that $\tilde{\sigma}_{int}(g) = \sigma_{int}(g)/(\pi\sigma_{12}^2)$. In the case of non-hard sphere potentials, σ_{12} is an arbitrary scaling factor chosen to best suit the problem at hand. For the hard sphere potential, $\tilde{\sigma}_{int}(g) = 1$ and Eq. (2.38) can be integrated analytically giving the known result [64, 65],

$$Z_{hs}(x) = \frac{A}{\sqrt{\gamma}}\left[e^{-\gamma x}+\frac{\sqrt{\pi}}{2}\Phi(\sqrt{\gamma x})\left\{\frac{1}{\sqrt{\gamma x}}+2\sqrt{\gamma x}\right\}\right]\,, \tag{2.39}$$

in which $\Phi(x) = (2\sqrt{\pi})\int_0^x \exp(-w^2)dw$ is the error function. The kernel is obtained from Eq. (2.37) using the parameters $Q = (1/\sqrt{\gamma}+\sqrt{\gamma})/2$ and $R = (1/\sqrt{\gamma}-\sqrt{\gamma})/2$ and making

the change of variable $z = (x' + x - 2\sqrt{xx'}y)^{1/2}$ to give

$$\tilde{K}(x,x') = \frac{AQ^2}{\sqrt{\gamma x'}} e^{(Q-R)(Rx'-Qx)} \int_{|\sqrt{x'}-\sqrt{x}|}^{\sqrt{x'}+\sqrt{x}} dz e^{-QRz^2} F(x,x',z;\gamma) \ , \tag{2.40}$$

where

$$F(x,x',z;\gamma) = \int_0^\infty ds e^{-s} \tilde{\sigma} \left[\sqrt{\frac{2kT}{m_1}} \sqrt{s+Q^2z^2}, \cot^{-1}\left(\frac{\sqrt{s}}{qz}\right) \right] I_0\left[(Q-R)]\sqrt{\Delta s}\right] \tag{2.41}$$

and

$$\Delta = 2\left(x'+x\right) - z^2 - \left(\frac{x'-x}{x}\right)^2 \ . \tag{2.42}$$

Since Δ is symmetric with respect to the exchange of x and x', $F(x,x',z;\gamma)$ and the integral in Eq. (2.40) have the same symmetry. The expression given is optimized numerically for treating heavy tracers for which $\gamma < 1$ and $QR > 0$. For lighter tracers with $\gamma > 1$, $QR < 0$ and the value of $\exp(-QRz^2)$ in the integral may become very large, posing numerical challenges during convergence. The kernel is generally well-behaved though it may be helpful to transform to another variable other than z when casting the equations into a numerical algorithm.

2.2.1 Hard Sphere

For a hard sphere potential, $\tilde{\sigma}(g,\chi) = 1$, and using the relation $\int_0^\infty ds \exp[-s] I_0(\alpha\sqrt{s}) = \exp[\alpha^2/4]$, allows the kernel to be evaluated analytically giving the Wigner-Wilkins kernel [53, 58, 64–66], that is

$$\begin{aligned} \tilde{K}_{hs}(x,x') &= \frac{AQ^2}{2}\sqrt{\frac{\pi}{x'}}\left[\Phi\left(Q\sqrt{x}+R\sqrt{x'}\right) + e^{x'-x}\Phi\left(R\sqrt{x}+Q\sqrt{x'}\right)\right. \\ &\pm \left.\left\{\Phi\left(Q\sqrt{x}-R\sqrt{x'}\right) + e^{x'-x}\Phi\left(R\sqrt{x}-Q\sqrt{x'}\right)\right\}\right] \ , \end{aligned} \tag{2.43}$$

in which the "+" and "-" signs are taken when $x < x'$ and $x > x'$, respectively.

2.3 Rough Hard Sphere

In the rough sphere model, rotating hard spheres with non-smooth surfaces are considered, where changes in rotational and translational energies/momenta occur as the particles collide. In this simple model, some internal degrees of freedom are described and inelastic collisions are introduced. The derivation given here does not account for any effects due to vibrational motion. In the rough sphere model it is a known fact that the degree of rotational-translational coupling is greater than in typical molecular systems [2], therefore

a qualitative understanding of the effect of rotational-translational coupling upon the evolution of the distribution function is used in this formulation.

The collision kernel for the rough hard sphere model is derived for a tracer particle dilute in a bath [67, 68] that is assumed to be at equilibrium. This allows the applicability of the linear Boltzmann equation to this model. The kernel depends upon both velocity and angular velocity quantities. Further to this derivation, an expression for the approximate rough hard sphere kernel is derived which assumes that the rotational degrees of freedom of the tracer remain at equilibrium and thus can be treated analytically. With this approximation, the kernel is only dependent upon translational velocities.

The tracer and bath particles in this model have velocities, $\mathbf{c}$, angular velocities, ω, moments of inertia, I, and diameters, σ. The characterization of the rough sphere model is based on its moment of inertia, $\alpha = 4I/(m\sigma^2)$. The values of alpha depend upon the distribution of mass in the tracer particles. With the mass concentrated at the center, $\alpha = 0$, mass concentrated uniformly at the surface of the particle, $\alpha = 2/3$ and with uniform mass density, $\alpha = 2/5$.

As for the smooth hard sphere case, the equations for the collision dynamics in this model can be written using $\mathbf{J}$ as [2, 69–73],

$$\begin{aligned} \mathbf{c}' - \mathbf{c} &= \frac{\mathbf{J}}{m} , \\ \mathbf{c}_\mathbf{1}' - \mathbf{c}_\mathbf{1} &= -\frac{\mathbf{J}}{m_1} , \\ \omega' - \omega &= -\frac{\sigma}{2I_1}\hat{\mathbf{k}} \times \mathbf{J} , \\ \omega_\mathbf{1}' - \omega_\mathbf{1} &= -\frac{\sigma_1}{2I_1}\hat{\mathbf{k}} \times \mathbf{J} , \\ \mathbf{J} &= \frac{2\mu}{1+\mu\chi}\left[\mathbf{v} + \mu\chi\left(\mathbf{v}\cdot\hat{\mathbf{k}}\right)\hat{\mathbf{k}}\right] . \end{aligned} \tag{2.44}$$

with $\chi = 1/(m\alpha) + 1/(m_1\alpha_1)$ (χ used in these equations should not be confused with the scattering angle that has been used previously) and

$$\mathbf{v} = \mathbf{g} - \frac{1}{2}\hat{\mathbf{k}} \times (\sigma_1\omega_1 + \sigma\omega) , \tag{2.45}$$

where $\mathbf{v}$ is the value of the relative velocity of the points of contact of the sphere at the collision. This includes the effects of the relative translational velocity (through $\mathbf{g}$) and the relative surface velocities that arise from rotations of the spheres. Upon collision, the sign of $\mathbf{v}$ changes, i.e. $\mathbf{v}' = -\mathbf{v}$. The collision equations for the total translational momentum,

total angular momentum, and total energy are written as

$$\begin{aligned} m\mathbf{c}+m_1\mathbf{c}_1 &= m\mathbf{c}'+m_1\mathbf{c}_1' \\ I\omega+I_1\omega_1-\frac{m\sigma}{2}\hat{\mathbf{k}}\times\mathbf{c}+\frac{m_1\mathbf{c}_1}{2}\hat{\mathbf{k}}\times\mathbf{c}_1 &= I\omega'+I_1\omega_1'+\frac{m\sigma}{2}\hat{\mathbf{k}}\times\mathbf{c}'+\frac{m_1\sigma_1}{2}\hat{\mathbf{k}}\times\mathbf{c}_1' \\ \frac{1}{2}mc^2+\frac{1}{2}m_1c_1^2+\frac{1}{2}I\omega^2+\frac{1}{2}I_1\omega_1^2 &= \frac{1}{2}mc'^2+\frac{1}{2}m_1c_1'^2+\frac{1}{2}I\omega'^2+\frac{1}{2}I_1\omega_1'^2\ , \end{aligned} \tag{2.46}$$

where the total angular momentum includes contributions from the internal rotations of the particles, as well as their orbiting contributions (of the form $\mathbf{r}\times\mathbf{p}$) at the point of collision. The form of the linear Boltzmann equation for the low density rough hard sphere tracer particle in an uniform, infinite rough sphere bath with no external fields, is given as

$$\frac{\partial f}{\partial t}=\int\left(f'f_1^{(0)'}-ff_1^{(0)}\right)\sigma_{12}^2\left(\mathbf{g}\cdot\hat{\mathbf{k}}\right)d\hat{\mathbf{k}}d\mathbf{c}_1d\omega_1\ , \tag{2.47}$$

where $\sigma_{12}=(\sigma+\sigma_1)/2$. The integral over $\hat{\mathbf{k}}$ includes all values for which $\left(\mathbf{g}\cdot\hat{\mathbf{k}}\right)\geq 0$, and the equilibrium bath distribution function is a generalization of Eq. (2.10),

$$f^{(0)}(\mathbf{c}_1,\omega_1)=n_1\frac{(m_1I_1)^{3/2}}{(2\pi kT)^3}\exp\left[-\frac{1}{2kT}\left(m_1c_1^2+I_1\omega_1^2\right)\right]\ . \tag{2.48}$$

The Boltzmann equation can be written as

$$\frac{\partial}{\partial t}f(\mathbf{c},\omega)=\int K\left(\mathbf{c},\omega,\mathbf{c}',\omega'\right)f\left(\mathbf{c}',\omega'\right)d\mathbf{c}'d\omega'-Z(\mathbf{c},\omega)f(\mathbf{c},\omega)\ , \tag{2.49}$$

with the collision kernel, $K(\mathbf{c},\omega,\mathbf{c}',\omega')$ and the energy dependent collision frequency, $Z(\mathbf{c},\omega)$. Comparing the loss term in Eq. (2.47) with the last term in Eq. (2.49), gives

$$Z(\mathbf{c},\omega)=\int f^{(0)}(\mathbf{c}_1,\omega_1)\,\sigma_{12}^2\left(\mathbf{g}\cdot\hat{\mathbf{k}}\right)d\hat{\mathbf{k}}d\mathbf{c}_1d\omega_1\ . \tag{2.50}$$

Substituting Eq. (2.48) the angular velocity dependence can be integrated analytically. Also, performing the remaining integrals using $c_1^2=c^2+g^2+2\mathbf{g}\cdot\mathbf{c}$ and $\int(\mathbf{g}\cdot\hat{\mathbf{k}})d\hat{\mathbf{k}}=\pi g$ yields the same expression as for the smooth hard sphere, given in Eq. (2.39). This means the presence of translational-rotational coupling in the rough sphere model does not affect the frequency term, and this becomes independent of angular velocity. This is consistent since the smooth and rough sphere fluids have precisely the same structure, resulting from the hard impact at the particle surface.

The collision kernel is obtained from the gain term in the Boltzmann equation, therefore

comparing Eqs. (2.47) and (2.49) gives

$$\int f^{(0)}(\mathbf{c}_1', \omega_1') f(\mathbf{c}', \omega') \sigma_{12}^2 (\mathbf{g} \cdot \hat{\mathbf{k}}) d\hat{\mathbf{k}} \mathbf{c}_1 d\omega_1 = \int K(\mathbf{c}, \omega, \mathbf{c}', \omega') f(\mathbf{c}', \omega') d\mathbf{c}' d\omega' \; . \tag{2.51}$$

The goal is to rewrite the variable dependency on the left hand side of the equation to match that on the right hand side. This is done using the variables $\mathbf{w} = \sigma_1 \omega_1 + \sigma \omega$, $\mathbf{C} = \mathbf{c}' - \mathbf{c}$, and $\mathbf{W} = \omega' - \omega$. Since $\mathbf{c}$ and ω are kept constant in Eq. (2.49), it follows that $d\mathbf{c}_1 d\omega_1 = 1/\sigma_1^3 d\mathbf{g} d\mathbf{w}$ and $d\mathbf{c}' d\omega' = d\mathbf{C} d\mathbf{W}$.

Now, $\mathbf{c}_1$ and ω_1, and therefore $\mathbf{g}$ and $\mathbf{w}$, are independently varying vectors so $d\mathbf{g} d\mathbf{w}$ represents six independent differentials. This is different for $\mathbf{C}$ and $\mathbf{W}$. The first three equations in Eq. (2.44) show that

$$\begin{aligned} \mathbf{J} &= m\mathbf{C} \; , \\ \mathbf{W} &= -\frac{\sigma}{2I} \hat{\mathbf{k}} \times \mathbf{J} = -\frac{m\sigma}{2I} \hat{\mathbf{k}} \times \mathbf{C} \; , \end{aligned} \tag{2.52}$$

so $\mathbf{C} \cdot \mathbf{W} = 0$ regardless of the value of $\mathbf{J}$. Therefore, $\mathbf{C}$ and $\mathbf{W}$ must always be perpendicular, that is, $d\mathbf{C} d\mathbf{W}$ (hence $d\mathbf{c}' d\omega'$) cannot represent six independent differentials because of this constraint. There are only five independently varying quantities. If the right hand side of Eq. (2.51) is to be interpreted as a six-dimensional integral, then it must be the case that the kernel contains a Dirac delta function which effectively reduces the number of independently varying quantities to five. Therefore, the collision kernel must be derived with care and it will be shown how the delta function arises in the derivation.

Different approaches could be taken to derive the collision kernel for the rough hard sphere model. In one approach, one could transform to the magnitudes C and W, which are independently varying quantities, and then transform the angular dependencies minding the orthogonality constraint. In yet another approach, variables other than $\mathbf{C}$ and/or $\mathbf{W}$ could be used to bypass the orthogonality constraint. Several variations of these approaches were tried and given here is the derivation that was the simplest and most straightforward. In this approach, $\mathbf{g}$ is transformed to $\mathbf{C}$, and then the angular frequency part is dealt with.

To begin, a number of relations are derived that are helpful in the derivation. Inverting the last equation in Eq. (2.44) yields $\mathbf{v}$ as a function of $\mathbf{J}$, giving

$$\mathbf{v} = \frac{1 + \mu\chi}{2\mu} \left[\mathbf{J} - \frac{\mu\chi}{1 + \mu\chi} \left(\mathbf{J} \cdot \hat{\mathbf{k}} \right) \hat{\mathbf{k}} \right] \; . \tag{2.53}$$

Combining this equation with Eq. (2.45) and the first equation in Eq. (2.52)gives

$$\mathbf{g} = \left(\frac{g}{2\mu} \right) (1 + \mu\chi) \left[\mathbf{C} - \frac{\mu\chi}{1 + \mu\chi} \left(\mathbf{C} \cdot \hat{\mathbf{k}} \right) \hat{\mathbf{k}} \right] + \frac{1}{2} \hat{\mathbf{k}} \times \mathbf{w} \; . \tag{2.54}$$

From Eq. (2.54), a few useful relations can be derived, namely

$$d\mathbf{g} = \left(\frac{m}{2\mu}\right)^3 (1+\mu\chi)^2 \, d\mathbf{C} \; , \tag{2.55}$$

$$\mathbf{g}\cdot\hat{\mathbf{k}} = \frac{m}{2\mu}\mathbf{C}\cdot\hat{\mathbf{k}} \; , \tag{2.56}$$

$$\mathbf{g}\cdot\mathbf{c} = \left(\frac{m}{2\mu}\right)(1+\mu\chi)\left[\mathbf{C}\cdot\mathbf{c} - \frac{\mu\chi}{1+\mu\chi}\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)\left(\mathbf{c}\cdot\hat{\mathbf{k}}\right)\right] + \frac{1}{2}\mathbf{c}\cdot\hat{\mathbf{k}}\times\mathbf{w} \; , \tag{2.57}$$

$$\begin{aligned} g^2 &= \left(\frac{m\,(1+\mu\chi)}{2\mu}\right)^2 \left[C^2 - \frac{\mu\chi}{(1+\mu\chi)^2}\,(2+\mu\chi)\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)^2\right] \\ &+ \frac{1}{4}\left(\hat{\mathbf{k}}\times\mathbf{w}\right)^2 + \frac{m\,(1+\mu\chi)}{2\mu}\mathbf{C}\cdot\hat{\mathbf{k}}\times\mathbf{w} \; . \end{aligned} \tag{2.58}$$

Using the relations gives in Eqs. (2.55)-(2.58) shows that

$$\sigma_{12}^2\left(\mathbf{g}\cdot\hat{\mathbf{k}}\right)d\hat{\mathbf{k}}d\mathbf{c}_1 d\omega_1 = \frac{\sigma_{12}^2}{\sigma_1^3}\left(\frac{m}{2\mu}\right)^4 (1+\mu\chi)^2\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)d\hat{\mathbf{k}}d\mathbf{C}d\mathbf{w} \; , \tag{2.59}$$

in which it is understood that the integral over $\hat{\mathbf{k}}$ is performed for values where $\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right) \geq 0$. Using the definition of $\mathbf{W}$ from Eq. (2.52) gives in component form

$$\begin{aligned} W_x &= \frac{m\sigma}{2I}\left[C_y\cos\theta - C_z\sin\theta\sin\phi\right] \; , \\ W_y &= \frac{m\sigma}{2I}\left[C_z\sin\theta\cos\phi - C_x\cos\theta\right] \; , \\ W_z &= \frac{m\sigma}{2I}\left[C_x\sin\theta\sin\phi - C_y\sin\theta\cos\phi\right] \; , \end{aligned} \tag{2.60}$$

in which spherical polar coordinates are used to express the component of the unit vector $\hat{\mathbf{k}}$ with $d\hat{\mathbf{k}} = \sin\theta d\theta d\phi$. As discussed earlier, for given $\mathbf{C}$ and $\hat{\mathbf{k}}$, the components of $\mathbf{W}$ are not independent but satisfy, as seen in Eqs. (2.60), $\mathbf{C}\cdot\mathbf{W} = 0$. There, only two of the three components are independent. The next transformation maps θ and ϕ to these two independent components of $\mathbf{W}$, which without loss of generality shall be chosen as W_x and W_y. Calculating the appropriate Jacobian for this transformation using Eqs. (2.60) gives

$$dW_x dW_y = \left(\frac{m\sigma}{2I}\right)^2 C_z\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)\sin\theta\sin\phi = \left(\frac{m\sigma}{2I}\right)^2 C_z\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)d\hat{\mathbf{k}} \; . \tag{2.61}$$

At this point the Dirac delta function is introduced to create an additional integration over

W_z which maintains the dependencies in Eqs. (2.60) that is,

$$\begin{aligned} dW_x dW_y &= \delta\left[W_z + \left(\frac{C_x W_x + C_y W_y}{C_z}\right)\right] dW_x dW_y dW_z \\ &= C_z \delta\left(\mathbf{C}\cdot\mathbf{W}\right) d\mathbf{W} \ . \end{aligned} \tag{2.62}$$

After combining Eqs. (2.59), (2.61), (2.62) allows the left side of Eq. (2.51) to be written

$$\int f^{(0)}(\mathbf{c}_1', \omega_1') f(\mathbf{c}', \omega') \sigma_{12}^2 \left(\mathbf{g}\cdot\hat{\mathbf{k}}\right) d\hat{\mathbf{k}} d\mathbf{c}_1 d\omega$$

$$= \frac{\sigma_{12}^2}{\sigma_1^3}\left(\frac{m}{2\mu}\right)^2 \left(\frac{I}{\mu\sigma}\right)^2 (1+\mu\chi)^2 \int f^{(0)}\left(\mathbf{c}_1', \omega_1'\right), f(\mathbf{c}', \omega') \delta(\mathbf{C}\cdot\mathbf{W}) d\mathbf{w} d\mathbf{C} d\mathbf{W} \ , \tag{2.63}$$

which when compared with the right hand side of Eq. (2.51) and using the definitions for $\mathbf{C}$ and $\mathbf{W}$ identifies the kernel as

$$K(\mathbf{c}, \omega, \mathbf{c}', \omega') = \frac{\sigma_{12}^2}{\sigma_1^3}\left(\frac{m}{2\mu}\right)^2 \left(\frac{I}{\mu\sigma}\right)^2 (1+\mu\chi)^2 \delta\left[\left(\mathbf{c}'-\mathbf{c}\right)\cdot\left(\omega'-\omega\right)\right] \int f^{(0)}(\mathbf{c}_1', \omega_1') d\mathbf{w} \ . \tag{2.64}$$

To evaluate the integral over $\mathbf{w}$, use the conservation of energy expression in Eq. (2.46) to rewrite the Maxwellian as

$$f^{(0)}(\mathbf{c}_1', \omega_1') = n_1 \frac{(m_1 I_1)^{3/2}}{(2\pi k T)^3} \exp\left[-\frac{1}{2kT}\left(mc^2 + I\omega^2 - mc'^2 - I\omega'^2 + m_1 c_1^2 + I_1 \omega_1^2\right)\right] \ . \tag{2.65}$$

The definition of $\mathbf{w}$ yields $\omega^2 = (w^2 - 2\sigma\mathbf{w}\cdot\omega + \sigma^2\omega^2)/\sigma_1^2$ and that of $\mathbf{g}$ yields $c_1^2 = c^2 + g^2 + 2\mathbf{g}\cdot\mathbf{c}$ which can be rewritten using Eqs. (2.57) and (2.58) to eliminate the factors of $\mathbf{g}$. Performing these substitutions and simplifying gives the final expression of the kernel as

$$\begin{aligned} K(\mathbf{c}, \omega, \mathbf{c}', \omega') &= n_1 \frac{\sigma_{12}^2}{\sigma_1^3}\left(\frac{m_1}{2\pi k T}\right)^{3/2}\left(\frac{I_1}{2\pi k T}\right)^{3/2}\left(\frac{m}{2\mu}\right)^2\left(\frac{I}{\mu\sigma}\right)^2 \\ &\times (1+\mu\chi)^2 \delta\left[\left(\mathbf{c}'-\mathbf{c}\right)\cdot\left(\omega'-\omega\right)\right] \\ &\times \exp\left[-\frac{m}{2kT}\left(c^2 - c'^2\right) - \frac{1}{2kT}\left(\omega^2 - \omega'^2\right) - \frac{I_1\sigma^2}{2\sigma_1^2 kT}\omega^2\right] \\ &\times \exp\left[-\frac{m_1}{2kT}\left\{\beta^2 - \left(\frac{m}{2\mu}\right)^2 \mu\chi\left(2+\mu\chi\right)\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)^2 - m\chi\left(\mathbf{c}\cdot\hat{\mathbf{k}}\right)\right\}\right] \\ &\times \int \exp\left[-\frac{I_1}{2\sigma_1^2 kT}\left(w^2 - 2\sigma\omega\cdot\mathbf{w}\right) - \frac{m_1}{2kT}\left\{\frac{1}{4}\left(\hat{\mathbf{k}}\times\mathbf{w}\right)^2 + \beta\cdot\hat{\mathbf{k}}\times\mathbf{w}\right\}\right] d\mathbf{w} \ , \end{aligned} \tag{2.66}$$

in which

$$\beta = \left(\frac{m}{2\mu}\right)(1+\mu\chi)\,\mathbf{C} + \mathbf{c} \ . \tag{2.67}$$

Either before or after performing the integration over $\mathbf{w}$ it is necessary to express the $\hat{\mathbf{k}}$ dependence in terms of $\mathbf{C} = \mathbf{c}' - \mathbf{c}$ and $\mathbf{W} = \omega' - \omega$. This dependence can be obtained by taking the magnitude of the second equation in Eq. (2.52) and rearranging to give

$$\left(\mathbf{C} \cdot \hat{\mathbf{k}}\right)^2 = C^2 - \left(\frac{2I}{m\sigma}\right)^2 W^2 \,. \tag{2.68}$$

Note that for the smooth hard sphere from Eq. (2.15) $(\mathbf{C}\cdot\hat{\mathbf{k}})^2 = C^2$, and comparing with the relation above, and noting Eq. (2.56), shows that for the rough hard sphere, the rotational motion decreases the component of $\mathbf{g}$ along the apse direction $\hat{\mathbf{k}}$. It also shows that this component depends only upon the magnitudes of $\mathbf{C}$ and $\mathbf{W}$ and not upon their directions in space. Taking the vector cross product of $\mathbf{C}$ with the last equation in Eq. (2.52) gives

$$\hat{\mathbf{k}} = \frac{1}{C^2}\left[\left(\mathbf{C} \cdot \hat{\mathbf{k}}\right)\mathbf{C} - \frac{2I}{m\sigma}\mathbf{C}\times\mathbf{W}\right] = \frac{1}{C^2}\left[\sqrt{C^2 - \left(\frac{2I}{m\sigma}\right)^2 W^2}\mathbf{C} - \frac{2I}{m\sigma}\mathbf{C}\times\mathbf{W}\right] \,. \tag{2.69}$$

This last relation allows all the dependence upon $\hat{\mathbf{k}}$ to be expressed in terms of the kernel variable dependencies thus providing a complete description.

From this, it can be seen that the expression for the kernel contains many terms, including those which are functions solely of $\mathbf{C}$ or $\mathbf{W}$ but also those which mix these variable dependencies in both magnitude and angle. As expected, the translational and rotational dependencies are intimately mixed in the kernel preventing any simple factorization. Also, unlike the corresponding kernel for the smooth sphere case which contains only angular dependence through $\mathbf{c}\cdot\mathbf{C}$ (allowing simplifications to be made by expanding the kernel in terms of Legendre polynomials in this angle) the angular dependence in the rough sphere kernel is fully expressed in both the translational and rotational degrees of freedom. This requires expansions in terms of spherical harmonics in both angular degrees of freedom. Such an expansion would produce a set of integro-differential equations similar to Eq. (2.28) except they would couple all components of the kernel. While only the spherical component of the kernel would be necessary to determine the average kinetic energy, the evolution of the distribution function would depend upon not only this spherical component but upon all other components of the kernel as well. While closed form expressions for the kernel, including the spherical component, can be evaluated for the smooth hard sphere model, it appears quite difficult to do so for the rough hard sphere model.

2.4 Approximate Rough Hard Sphere Kernel

The expression for the exact collision kernel for the rough hard sphere is complex. To avoid using the expression in this form, several options were explored to simplify it. The goal of using this collision kernel is to study the kinetic energy loss in large, massive tracer ions moving at high translational energies through a buffer gas. It is expected that energy exchange will dominate, and rotational energy exchange should be less important. It is then reasonable to assume that the rotational degrees of freedom of the tracer as always being in a state of equilibrium, the same as the surrounding bath gas. The distribution function then becomes

$$f(\mathbf{c},\omega) \approx f(\mathbf{c}) \left(\frac{1}{2\pi kT}\right)^{3/2} \exp\left[-\frac{1}{2kT}\omega^2\right] , \tag{2.70}$$

where $f(\mathbf{c})$ is the translational distribution function for the tracer. Substituting Eq. (2.70) into Eq. (2.49) and integrating both sides with respect to ω gives

$$\frac{\partial}{\partial t} f(\mathbf{c}) = \int K_1(\mathbf{c},\mathbf{c}') f(\mathbf{c}') d\mathbf{c}' - Z(\mathbf{c}) f(\mathbf{c}) , \tag{2.71}$$

where $Z(\mathbf{c})$ is given by Eq. (2.39) and the new kernel is given by

$$K_1(\mathbf{c},\mathbf{c}') = \left(\frac{1}{2\pi kT}\right)^{3/2} \int K(\mathbf{c},\omega,\mathbf{c}',\omega') \exp\left[-\frac{1}{2\pi kT}\omega'^2\right] d\omega' d\omega . \tag{2.72}$$

Using the kernel expression in Eq. (2.66) and using the definitions of $\mathbf{C}$ and $\mathbf{W}$ for different quantities, gives

$$\begin{aligned} K_1(\mathbf{c},\mathbf{c}') &= n_1 \left(\frac{m_1}{2\pi kT}\right)^{3/2} \left(\frac{m}{2\mu}\right)^2 \left(\frac{I}{\mu\sigma}\right)^2 (1+\mu\chi)^2 \exp\left[-\frac{m}{2kT}\left(c^2 - c'^2\right)\right] \\ &\times \int d\mathbf{W}\delta(\mathbf{C}\cdot\mathbf{W}) \exp\left[-\frac{1}{2\pi kT}\left\{\beta^2 - \left(\frac{m}{2\mu}\right)^2 \mu\chi\left(2+\mu\chi\right)\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)^2\right.\right. \\ &- \left.\left. m\chi\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)\left(\mathbf{c}\cdot\hat{\mathbf{k}}\right)\ \right\}\right] \left\{\left(\frac{1}{2\pi kT}\right)^{3/2} \left(\frac{I_1}{2\pi kT}\right)^{3/2}\right. \\ &\times \left. \exp\left[-\frac{1}{2kT}\omega^2 - \frac{I_1}{2\sigma_1^2 kT}\left(\sigma^2\omega^2 - 2\sigma\omega\cdot\mathbf{w} + w^2\right)\right]\right\} \\ &\times \exp\left[-\frac{m_1}{2kT}\left(\frac{1}{4}\left(\hat{\mathbf{k}}\times\mathbf{w}\right)^2 + \beta\cdot\hat{\mathbf{k}}\times\mathbf{w}\right)\right] d\mathbf{w} . \end{aligned} \tag{2.73}$$

The integral over ω can be done analytically using the relation $4\chi = (I_1\sigma^2 + I\sigma_1^2)/(II_1)$, with the result that the quantity within braces becomes $\sigma_1^3/(8\pi\chi kT)^{3/2}\exp[-w^2/(8\chi kT)]$. Inserting this result back into the expression for the kernel, and also using Eqs. (2.61) and

(2.62) to transform from integration over $\mathbf{W}$ to that over $\hat{\mathbf{k}}$ gives

$$\begin{aligned} K_1(\mathbf{c},\mathbf{c}') &= \frac{n_1\sigma_{12}^2}{(8\pi kT)^{3/2}}\left(\frac{m_1}{2\pi kT}\right)^{3/2}\left(\frac{m}{2\mu}\right)^4(1+\mu\chi)^2\exp\left[-\frac{m}{2kT}\left(c^2-c'^2\right)\right] \\ &\times \int\exp\left[-\frac{m_1}{2kT}\left\{\beta^2-\left(\frac{m}{2\mu}\right)^2\mu\chi\left(2+\mu\chi\right)\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)\left(\mathbf{c}\cdot\hat{\mathbf{k}}\right)\right\}\right] \\ &\times \left\{\int\exp\left[-\frac{1}{8\chi kT}w^2-\frac{m_1}{2kT}\left(\frac{1}{4}\left(\hat{\mathbf{k}}\times\mathbf{w}\right)^2+\boldsymbol{\beta}\cdot\hat{\mathbf{k}}\times\mathbf{w}\right)\right]d\mathbf{w}\right\}\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)d\hat{\mathbf{k}}\,, \end{aligned} \tag{2.74}$$

where the integral over $\hat{\mathbf{k}}$ includes all values for which $(\mathbf{C}\cdot\hat{\mathbf{k}})\geq 0$. The integral over $\mathbf{w}$ can be performed taking $\hat{\mathbf{k}}$ and $\boldsymbol{\beta}$ as fixed variables. Let $\hat{\mathbf{k}}$ define the polar axis of a spherical polar coordinate system and let $\boldsymbol{\beta}$ lie in the xz plane so that $\boldsymbol{\beta}=\beta(\sin\theta_\beta\hat{\mathbf{x}}+\cos\theta_\beta\hat{\mathbf{z}})$ with θ_β the angle between $\hat{\mathbf{k}}$ and $\boldsymbol{\beta}$. If θ and ϕ are the polar angles associated with $\mathbf{w}$ then $\hat{\mathbf{k}}\times\mathbf{w}=w(-\sin\theta\sin\phi\hat{\mathbf{x}}+\sin\theta\cos\phi\hat{\mathbf{y}})$, $(\hat{\mathbf{k}}\times\mathbf{w})^2=w^2\sin\theta^2$, and $\boldsymbol{\beta}\cdot\hat{\mathbf{k}}\times\mathbf{w}=-\beta w\sin\theta\sin\phi_\beta=-|\boldsymbol{\beta}\times\hat{\mathbf{k}}|w\sin\theta\sin\phi$. Using these relations in the integral over $\mathbf{w}$ and integrating over ϕ using $\int_0^{2\pi}\exp[\pm z\sin\phi]d\phi=2\pi I_0(z)$ with $I_0(z)$ a modified Bessel function, then gives the contents within the braces as

$$2\pi\int_0^\pi\sin\theta d\theta\int_0^\infty w^2dw\exp\left[-\frac{1}{8\chi kT}w^2\left(1+m_1\chi\sin^2\theta\right)\right]I_0\left(\frac{m_1}{2kT}|\boldsymbol{\beta}\times\hat{\mathbf{k}}|w\sin\theta\right)\,. \tag{2.75}$$

The integration over w can be performed by making a change of variable $y=w^2$ and using the relation [74]

$$\int_0^\infty dyy^{\mu-1/2}e^{\alpha y}I_{2\nu}\left(2\rho\sqrt{y}\right)=\frac{\Gamma\left(\mu+\nu+1/2\right)}{\Gamma\left(2\nu+1\right)}\frac{\rho^\nu}{\alpha^{\mu+\nu+1/2}}M\left(\mu+\nu+\frac{1}{2},1+2\nu,\frac{\rho^2}{\alpha}\right)\,, \tag{2.76}$$

in which $M(a,b,z)$ is Kummer's (confluent hypergeometric) function, so that Eq. (2.75) becomes

$$(8\chi kT)^{3/2}\int_0^{\pi/2}d\theta\frac{\sin\theta}{\left(1+m_1\chi\sin^2\theta\right)^{3/2}}M\left(\frac{3}{2},1,\frac{m_1^2\chi}{2kT}\frac{\sin^2\theta}{1+m_1\chi\sin^2\theta}|\boldsymbol{\beta}\times\hat{\mathbf{k}}|^2\right)\,, \tag{2.77}$$

in which it has been recognized that since the integrand depends only upon $\sin\theta$ that the integration over θ is symmetric about $\theta=\pi/2$. The integral above can be solved [75] yielding the final expression

$$\frac{(8\pi\chi kT)^2}{1+m_1\chi}\exp\left[\frac{m_1\chi}{(1+m_1\chi)}\frac{m_1}{2kT}|\boldsymbol{\beta}\times\hat{\mathbf{k}}|^2\right]\,. \tag{2.78}$$

Incorporating these results in the expression for the kernel and using $|\boldsymbol{\beta}\times\hat{\mathbf{k}}|^2 = \beta^2 - (\boldsymbol{\beta}\cdot\hat{\mathbf{k}})^2$ then gives

$$\begin{aligned} K_1(\mathbf{c},\mathbf{c}') &= n_1\sigma_{12}^2\left(\frac{m_1}{2\pi kT}\right)^{3/2}\left(\frac{m}{2\mu}\right)^4\frac{(1\mu\chi)^2}{(1+m_1\chi)} \\ &\times \exp\left[-\frac{m}{2kT}\left(c^2-c'^2\right)-\frac{1}{(1+m_1\chi)}\frac{m_1}{2kT}\beta^2\right]\left\{\int\exp\left[-\frac{m_1}{2kT}\left\{\frac{m_1\chi}{1+m_1\chi}\left(\boldsymbol{\beta}\cdot\hat{\mathbf{k}}\right)^2\right.\right.\right. \\ &- \left.\left.\left.\left(\frac{m}{2\mu}\right)^2\mu\chi\left(2+\mu\chi\right)\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)^2 - m\chi\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)\left(\mathbf{c}\cdot\hat{\mathbf{k}}\right)\right\}\right]\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)d\hat{\mathbf{k}}\right\}\ . \end{aligned} \tag{2.79}$$

To perform the integration over $\hat{\mathbf{k}}$ it is convenient to choose the direction of $\mathbf{C}$ as the polar axis in a spherical polar coordinate system and let $\mathbf{c}$ be in the xz plane. Since the integral over $\hat{\mathbf{k}}$ must be performed for $(\mathbf{C}\cdot\hat{\mathbf{k}}) \geq 0$ the polar angle θ should range from 0 to $\pi/2$ while the angle ϕ should range from 0 to 2π. With these definitions we get

$$\begin{aligned} \mathbf{C}\cdot\hat{\mathbf{k}} &= C\cos\theta\ , \\ \mathbf{c}\cdot\hat{\mathbf{k}} &= c\left(\sin\theta\cos\phi\sin\theta_c+\cos\theta\cos\theta_c\right) = \frac{|\mathbf{c}\times\mathbf{C}|}{\mathbf{C}}\sin\theta\cos\phi+\frac{(\mathbf{c}\cdot\mathbf{C})}{\mathbf{C}}\cos\theta\ , \\ \boldsymbol{\beta}\cdot\hat{\mathbf{k}} &= \left(\frac{m}{2\mu}\right)(1+\mu\chi)\left(\mathbf{C}\cdot\hat{\mathbf{k}}\right)+\left(\mathbf{c}\cdot\hat{\mathbf{k}}\right)\ , \end{aligned} \tag{2.80}$$

in which θ_c is the angle between $\mathbf{C}$ and $\mathbf{c}$, and the integral over $\hat{\mathbf{k}}$ within the braces of Eq. (2.79) becomes

$$\begin{aligned} C\int_0^{2\pi} d\phi \int_0^{\pi/2}\sin\cos\theta d\theta\exp\left[-\frac{m_1}{2kT}\left\{\left(\frac{m}{2\mu}\right)^2\left(1-\frac{(1+\mu\chi)^2}{1+m_1\chi}\right)C^2\cos^2\theta\right.\right. \\ + \frac{m_1\chi}{1+m_1\chi}\left(\frac{|\mathbf{c}\times\mathbf{C}|^2}{C^2}\sin^2\theta\cos^2\phi+(\mathbf{c}\cdot\mathbf{C})\cos^2\theta\left[1+\frac{(\mathbf{c}\cdot\mathbf{C})}{C^2}\right]\right. \\ \left.\left. + |\mathbf{c}\times\mathbf{C}|\sin\theta\cos\theta\cos\phi\left[1+2\frac{(\mathbf{c}\cdot\mathbf{C})}{C^2}\right]\right)\right\}\right]\ . \end{aligned} \tag{2.81}$$

The only dependence upon ϕ appears through terms containing $\cos\phi$, and $\int_0^{2\pi}\xi(\cos\phi)d\phi = 2\int_0^{\pi}\xi(\cos\phi)d\phi$ for any function ξ. Transform the integral over ϕ using this relation and $t = \phi/\pi$. In addition, insert the result for the integral over $\hat{\mathbf{k}}$ into the expression for the kernel, rearrange the resulting terms, including writing the expression for β^2 using Eq. (2.67) and substitute $\mathbf{C} = \mathbf{c}' - \mathbf{c}$ to give

$$\begin{aligned} K_1(\mathbf{c},\mathbf{c}') = K_1(c,c',y) &= \frac{n_1\sigma_{12}^2}{|\mathbf{c}'-\mathbf{c}|}\left(\frac{m_1}{2\pi kT}\right)^{1/2}\left(\frac{m}{2\mu}\right)^2 F_1 \\ &\times \exp\left[\frac{m}{2kT}c'^2-\frac{(m+m_1)}{2kT}\left\{\mathbf{c}'\cdot\mathbf{c}+\left(\frac{m}{4\mu}\right)|\mathbf{c}'-\mathbf{c}|^2\right\}\right]\ , \end{aligned} \tag{2.82}$$

in which

$$
\begin{aligned}
F_1 &= 2|\mathbf{c}'-\mathbf{c}|^2\left(\frac{m_1}{2kT}\right)\left(\frac{2}{2\mu}\right)^2\frac{(1+\mu\chi)^2}{(1+m_1\chi)} \qquad (2.83)\\
&\times \int_0^1 dt\int_0^{\pi/2}\sin\theta\cos\theta d\theta\exp\left[-\frac{m_1}{2kT}\left\{\frac{1}{4}\left[\left(\frac{m}{m_1}\right)m\chi-\frac{m_1\chi}{1+m_1\chi}\right]|\mathbf{c}'-\mathbf{c}|^2\sin^2\theta\right.\right.\\
&+ \frac{m_1\chi}{1+m_1\chi}\left(\frac{|\mathbf{c}\times\mathbf{c}'|^2}{|\mathbf{c}'-\mathbf{c}|^2}\sin^2\theta\cos^2(\pi t)-\left(\mathbf{c}'\cdot\mathbf{c}\right)\sin^2\theta-\frac{|\mathbf{c}\times\mathbf{c}'|^2}{|\mathbf{c}'\times\mathbf{c}|^2}\cos^2\theta\right.\\
&+ \left.\left.\left.\left(c'^2-c^2\right)\frac{|\mathbf{c}\times\mathbf{c}'|}{|\mathbf{c}'-\mathbf{c}|^2}\sin\theta\cos\theta\cos(\pi t)\right)\right\}\right] .
\end{aligned}
$$

In obtaining this result, several useful relations were used, such as $(m/2\mu)^2[1-(1+\mu\chi)^2/(1+m_1\chi)] = [m_1\chi/(1+m_1\chi)-(m/m_1)m\chi]/4$, $(\mathbf{c}\cdot\mathbf{C})[1+(\mathbf{c}\cdot\mathbf{C})/C^2] = (\mathbf{c}'\cdot\mathbf{c})-|\mathbf{c}\times\mathbf{c}'|^2$, and $[1+2(\mathbf{c}\cdot\mathbf{C})/C^2] = (c'^2-c^2)/|\mathbf{c}'-\mathbf{c}|^2$. This shows a simplified expression for the general kernel in Eq. (2.66). Looking at the expression in Eq. (2.82), shows that $K_1(\mathbf{c},\mathbf{c}')$ depends only upon the magnitudes c and c', and $y=\cos\Theta = \boldsymbol{\Omega}_{c'}\cdot\boldsymbol{\Omega}_c$ with Θ the angle between $\mathbf{c}$ and $\mathbf{c}'$. In other words, the Boltzmann equation reduces to a form with a kernel operator with the same dependencies found in Eq. (2.22) for the spherical case. The same analysis employed with that equation can be applied here, that is the kernel can be expanded in a series of Legendre polynomials, and the Boltzmann equation reduced to a series of uncoupled, one-dimensional integro-differential equations. Using analogous definitions shown in the equations for the angular components of the kernel gives the spherical component of the kernel as

$$
\begin{aligned}
\tilde{K}_1(x,x') &= 2\pi\frac{\sqrt{\pi}}{2}\left(\frac{2kT}{m}\right)^{3/2}\int_{-1}^1 dyK_1\left(\sqrt{\frac{2kTx}{m}},\sqrt{\frac{2kTx'}{m}},y\right) \qquad (2.84)\\
&= \frac{AQ^2}{\sqrt{\gamma}}\sqrt{x}\int_{-1}^1\frac{dy}{\left(x'+x-2\sqrt{xx'}y\right)^{1/2}}F_1\\
&\times \exp\left[-R^2\left(x'+x-2\sqrt{xx'}y\right)-x+(1-\gamma)\sqrt{xx'}y\right] .
\end{aligned}
$$

Making the change of variable $z=(x'+x-2\sqrt{xx'}y)^{1/2}$ and simplifying gives the final expression for the kernel as

$$
\tilde{K}_1(x,x') = \frac{AQ^2}{\sqrt{\gamma x'}}e^{(Q-R)(Rx'-Qx)}\int_{|\sqrt{x'}-\sqrt{x}|}^{\sqrt{x'}+\sqrt{x}}dze^{-QRz^2}F_1(x,x',z;\gamma,\mu\chi) . \qquad (2.85)
$$

The expression for $F_1(x,x',z;\gamma,\mu\chi)$ is obtained by rewriting Eq. (2.83) in terms of $x=(m/2kT)c^2$, $x'=(m/2kT)c'^2$, and $z^2=(m/2kT)|\mathbf{c}'-\mathbf{c}|^2$, and writing the parameter dependencies in terms of the mass ratio $\gamma=m_1/m$ and $\mu\chi$. Note that $m_1\chi=(1+\gamma)\mu\chi$

and $(m/\mu) = (1+\gamma)/\gamma$. With these variable substitutions, and transforming the integral over θ using $s = \sin^2\theta$ gives the final expression for F_1 as

$$\begin{aligned} F_1(x,x',z;\mu\chi) &= \frac{(1+\gamma)^2}{4\gamma}\frac{(1+\mu\chi)^2}{[1+(1+\gamma)\,\mu\chi]}z^2\int_0^1 dt\int ds \\ &\times \exp\left[-\frac{(1+\gamma)\,\mu\chi}{4\gamma}z^2 s + \frac{\gamma\,(1+\gamma)\,\mu\chi}{4\,[1+(1+\gamma)\,\mu\chi]}\right. \\ &\times \left.\left\{\Delta\left[1-s\cos^2(\pi t)\right] - 2\left(\frac{x'-x}{z}\right)\sqrt{\Delta s(1-s)}\cos(\pi t) + \left(\frac{x'-x}{z}\right)^2 s\right\}\right], \end{aligned} \tag{2.86}$$

with Δ given by Eq. (2.42). Note that similar to the spherical kernel of Eq. (2.40) the expression above shows that $F_1(x,x',z;\gamma,\mu\chi) = F_1(x',x,z;\gamma,\mu\chi)$ so that the integral in Eq. (2.85) is symmetric with respect to the exchange of x and x'.

Examining the expression for $\mathbf{J}$ from the last expression in Eq. (2.44) shows that when $\mu\chi \to \infty$, $\mathbf{J} \to 2\mu(\mathbf{g}\cdot\hat{\mathbf{k}})\hat{\mathbf{k}}$, that is the impulse shown in the last expression of Eq. (2.8) for the smooth hard sphere system. The inelasticity of the rough hard sphere model is negligible as $\mu\chi$ becomes very large. Therefore, in this limit the rough hard sphere kernel should reduce to the smooth hard sphere. This is shown by transforming F_1 in Eq. (2.86) using $s = (m/2kT)(m\chi/4)|\mathbf{c}'-\mathbf{c}|^2\sin^2\theta$ to give

$$\begin{aligned} F_1 &= \left(\frac{m_1}{\mu}\right)\frac{(1+\mu\chi)^2}{\mu\chi\,(1+m_1\chi)}\int_0^1 dt\int_0^{\frac{m}{2kT}\frac{m\chi}{4}|\mathbf{c}'-\mathbf{c}|^2} ds e^{-s} \\ &\times \exp\left[\frac{m_1}{2kT}\left(\frac{m_1\chi}{1+m_1\chi}\right)\frac{|\mathbf{c}\times\mathbf{c}'|^2}{|\mathbf{c}-\mathbf{c}|^2}\right] \\ &\times \exp\left[-\left(\frac{2m_1}{m}\right)^2\frac{1}{1+m_1\chi}\left\{\frac{|\mathbf{c}\times\mathbf{c}'|^2}{|\mathbf{c}'-\mathbf{c}|^4}s\left[1+\cos^2(\pi t)\right] - \left(\frac{1}{4}+\frac{(\mathbf{c}'\cdot\mathbf{c})}{|\mathbf{c}'-\mathbf{c}|^2}\right)s\right.\right. \\ &+ \left.\left.\sqrt{\frac{m}{2kT}}\frac{\sqrt{m\chi}}{2}\left(c'^2-c^2\right)\frac{|\mathbf{c}\times\mathbf{c}'|}{|\mathbf{c}'-\mathbf{c}|^3}\sqrt{s}\cos(\pi t)\sqrt{1-\left(\frac{2kT}{m}\right)\left(\frac{4}{m\chi}\right)\frac{s}{|\mathbf{c}'-\mathbf{c}|^2}}\right\}\right] \end{aligned} \tag{2.87}$$

In the limit when $\mu\chi$ is large, $m_1\chi$ is also large, and the prefactor in front of the integral in F_1 approaches unity, while all the terms is the exponentials, apart from the first two, approach zero due to the factor of $1/(1+m_1\chi)$ which multiplies them, that is,

$$\lim_{\mu\chi\to\infty} F_1 = \int_0^1 dt\int_0^\infty ds e^{-s}\exp\left[\frac{m_1}{2kT}\frac{|\mathbf{c}\times\mathbf{c}'|^2}{|\mathbf{c}-\mathbf{c}|^2}\right] - \exp\left[\frac{m_1}{2kT}\frac{|\mathbf{c}\times\mathbf{c}'|^2}{|\mathbf{c}'-\mathbf{c}|^2}\right]. \tag{2.88}$$

When this result is incorporated into Eq. (2.82), the smooth hard sphere kernel, $K_{hs}(\mathbf{c},\mathbf{c}')$ of Eq. (2.23) is obtained. Thus, the approximate rough hard sphere kernel approaches the smooth hard sphere one in the correct limit.

2.5 Discussion of Kernels

The analytical results are given for a number of different kernels. From these derivations a number of issues can arise when using these expressions numerically. Care must be taken when the limits x and x' approach zero in translating the spherical kernel of Eq. (2.40) into a numerical algorithm. For this kernel when $x \to 0$ with any value of fixed x', transforming the integral in Eq. (2.40) using $p = (z - \sqrt{x'})/\sqrt{x}$ gives

$$\sqrt{x} \int_{-1}^{1} dp \exp\left[-QR\left(p\sqrt{x} + \sqrt{x'}\right)^2\right] F\left(x, x', p\sqrt{x} + \sqrt{x'}; \gamma\right) \ . \tag{2.89}$$

In determining the limits of the integral over p, the relation $x < x'$ is used, which is always satisfied when x is small enough and x' has a finite value. The values of p are always of order unity so when $x \to 0$, the factors of $p\sqrt{x}$ become negligible, reducing the above expression to $2\sqrt{x}\exp(-QRx')F(0, x', \sqrt{x'}; \gamma)$. When $x = 0$ and $z = \sqrt{x'}$, Eq. (2.42) shows that $\Delta = 0$ which when used in the expression of Eq. (2.41) and noting $\lim_{z\to 0} I_0(z) = 1$, gives the limit of the above expression as

$$2\sqrt{x}e^{-QRx'} \int_0^\infty ds e^{-s} \tilde{\sigma}\left[\sqrt{\frac{2kT}{m_1}}\sqrt{s + Q^2x'}, 2\cot^{-1}\left(\frac{\sqrt{s}}{Q\sqrt{s}}\right)\right] \ . \tag{2.90}$$

The leading factor of $\sqrt{x}$ causes the kernel to approach zero when $x \to 0$ and since $F(x, x', z; \gamma) = F(x', x, z; \gamma)$ the preceding analysis shows when $x' \to 0$ the same limit as that given above will be obtained except with x replaced everywhere with x'. In this case, the leading factor of $\sqrt{x'}$ cancels with the same term in the denominator of the prefactor in Eq. (2.40) to give a finite result. In summary, the kernel has the following limiting values:

$$\begin{aligned} \tilde{K}(0, x') &= 0 \\ \tilde{K}(x, 0) &= \frac{2AQ^2}{\sqrt{\gamma}} \int_0^\infty ds e^{-s} \tilde{\sigma}\left[\frac{2kT}{m_1}\sqrt{s + Q^2x}, \cot^{-1}\left(\frac{\sqrt{s}}{Q\sqrt{x}}\right)\right] \ . \end{aligned} \tag{2.91}$$

For the smooth hard sphere, the last equation gives $\tilde{K}_{hs}(x, 0) = (2AQ^2/\sqrt{\gamma})\exp(-Q^2x)$ showing an exponential decay as energy increases. More generally, the equations can be used to examine the value of $\tilde{K}(0, 0)$. A limit of the last equation gives

$$\lim_{x\to 0} \tilde{K}(x, 0) = \frac{2AQ^2}{\sqrt{\gamma}} \int_0^\infty ds e^{-s} \tilde{\sigma}\left[\sqrt{\frac{2kTs}{m_1}}, 0\right] \ . \tag{2.92}$$

The forward-scattering cross section for most interaction potentials can be quite large therefore this limit can be quite significant. Comparing the two expressions in Eq. (2.91) shows

that $\lim_{x'\to 0}\tilde{K}(0,x') \neq \lim_{x\to 0}\tilde{K}(x,0)$; the kernel has a point discontinuity at $x = x' = 0$. This point does not affect the values of any integrals because it is of measure zero; however, care must be taken when casting equations in numerical algorithms since the kernel will not be a continuous function at zero energy.

In an analogous procedure, the limits of $\tilde{K}_1(x,x')$ when x or x' is small must be determined. Consider the limit when x is small (with finite x') and transform the integral over z in the kernel of Eq. (2.85) using $p = (z - \sqrt{x'})/\sqrt{x}$ to give

$$\sqrt{x}\int_{-1}^{1} dp \exp\left[-QR\left(p\sqrt{x}+\sqrt{x'}\right)^2\right] F_1\left(x,x',p\sqrt{x}+\sqrt{x'};\gamma,\mu\chi\right) . \tag{2.93}$$

In the limit when x is small, this reduces to $2\sqrt{x}e^{-QRx'}F_1(0,x',\sqrt{x'};\gamma,\mu\chi)$. As in the spherical case, the factor of $\sqrt{x}$ will cause the kernel to approach zero when $x \to 0$. The value of $F_1(0,x',\sqrt{x'};\gamma,\mu\chi)$ can be determined analytically using Eq. (2.86) and noting that in this limit $\Delta \to 0$. Performing the necessary algebra gives

$$F_1(0,x',\sqrt{x'};\gamma,\mu\chi) = \frac{(1+\mu\chi)^2}{\mu\chi\,(1+\mu\chi-\gamma)}\left[1-\exp\left\{-\frac{\mu\chi\,(1+\mu\chi-\gamma)}{[1+(1+\gamma)\,\mu\chi]}Q^2x'\right\}\right] \tag{2.94}$$

Again using $F_1(x,x',z;\gamma,\mu\chi) = F_1(x',x,z;\gamma,\mu\chi)$ gives the limit when x' is small to be identical with the above except with x' replacing x everywhere. The leading factor for $\sqrt{x'}$ then produces a finite limit.

In summary, putting together all this information gives the following limiting values for the kernel

$$\begin{aligned}
\tilde{K}_1(0,x') &= 0 , \qquad (2.95)\\
\tilde{K}_1(x,0) &= \frac{2AQ^2}{\sqrt{\gamma}}e^{-Q^2x}\frac{(1+\mu\chi)^2}{\mu\chi\,(1+\mu\chi-\gamma)}\left[1-\exp\left\{-\frac{\mu\chi\,(1+\mu\chi-\gamma)}{[1+(1+\gamma)\,\mu\chi]}Q^2x\right\}\right] .
\end{aligned}$$

A number of features can be seen in these expressions. For any finite value of x, $\lim_{\mu\chi\to\infty}\tilde{K}_1(x,0) = (2AQ^2/\sqrt{\gamma})\exp(-Q^2x)$ which agrees with the smooth sphere result. This is another confirmation that the rough hard sphere results reduce to the smooth hard sphere ones when $\mu\chi$ tends towards infinity. However, for any finite $\mu\chi$, the equations show $\lim_{x'\to 0}\tilde{K}_1(0,x') = \lim_{x\to 0}\tilde{K}_1(x,0) = 0$ so unlike the spherical kernel, $\tilde{K}(x,x')$, the approximate rough sphere kernel is continuous and zero at $x = x' = 0$.

In addition to discontinuities in the limits of zero energy, the kernels may also contain cusps. This can be seen by examining the derivative of the kernel expressions. For example,

the partial derivative of Eq. (2.37) with respect to x gives

$$\begin{aligned}\frac{\partial}{\partial x}\tilde{K}(x,x') &= -Q\left(Q-R\right)\tilde{K}(x,x') + \frac{AQ^2}{\sqrt{\gamma x'}}e^{(Q-R)(Rx'-Qx)}\int_{|\sqrt{x'}-\sqrt{x}|}^{\sqrt{x'}+\sqrt{x}} dz e^{-QRz^2} \\ &\times \frac{\partial}{\partial x}F(x,x',z;\gamma) + \frac{AQ^2}{2\sqrt{\gamma x x'}}\left[e^{-\left(R\sqrt{x'}+Q\sqrt{x}\right)^2}F(x,x',\sqrt{x'}+\sqrt{x};\gamma)\right. \\ &\times \left.\frac{|\sqrt{x'}-\sqrt{x}|}{\sqrt{x'}-\sqrt{x}}e^{-\left(R\sqrt{x'}-Q\sqrt{x}\right)^2}F(x,x',\sqrt{x'}-\sqrt{x};\gamma)\right] .\end{aligned} \tag{2.96}$$

As mentioned above, this integral over the scattering cross section in the forward direction can take on a large values, implying that the last term in the expression for the derivative of the kernel will indeed cause a cusp at $x = x'$ as this value adds or subtracts on either side. The value of this same integral was also responsible for the discontinuity of the kernel at zero energy. Thus, both the cusp and the point discontinuity at zero arises from the same physical contribution, namely, the scattering cross section in the forward direction.

As analogous procedure applied to the approximate rough sphere kernel would yield an expression identical with Eq. (2.96) except with $\tilde{K}(x,x')$ replaced with $\tilde{K}_1(x,x')$ and $F(x,x',z'\gamma)$ with $F_1(x,x',z;\mu\chi)$. Using the definition of the latter gives

$$\begin{aligned}F_1(x,x',\sqrt{x'}\pm\sqrt{x};\gamma,\mu\chi) &= \frac{(1+\gamma)^2}{4\gamma}\frac{(1+\mu\chi)^2}{\left[1+(1+\gamma)\,\mu\chi\right]}\left(\sqrt{x'}\pm\sqrt{x}\right)^2 \\ &\times \int_0^1 ds \exp\left[-\frac{(1+\gamma)\,\mu\chi}{4\gamma}\left(\sqrt{x'}\pm\sqrt{x}\right)^2 s\right. \\ &+ \left.\frac{\gamma\,(1+\gamma)\,\mu\chi}{4\left[1+(1+\gamma)\,\mu\chi\right]}\left(\frac{x'-x}{\sqrt{x'}\pm\sqrt{x}}\right)^2 s\right] .\end{aligned} \tag{2.97}$$

A limiting procedure gives $\lim_{x\to x'} F_1(x,x',\sqrt{x'}-\sqrt{x};\gamma,\mu\chi) = 0$. Since the factor appearing in the expression for the derivative of the kernel is zero, and since this term would normally be responsible for producing a cusp, the approximate rough hard sphere kernel has no cusp at $x = x'$. Instead the kernel is a smooth function everywhere, including in the limit of zero energy. Physically, this results because the rough hard sphere model undergoes translational-rotational coupling even for glancing collisions, thus affecting scattering in the forward direction.

A comparison of the energy dependence of the smooth and approximate rough sphere kernels is made in Figs. 2.1 and 2.2 for two different mass ratios and for a range of $\mu\chi$ values. Note that the largest values of the reduced moments of inertia occur when $\alpha = \alpha_1 = 2/3$, corresponding to the maximum amount of translational-rotational energy exchange. For these values $\mu\chi = 1.5$, and for any other values $\mu\chi > 1.5$. Thus, the minimum value of $\mu\chi$, corresponding to the maximum amount of translational-rotational energy exchange

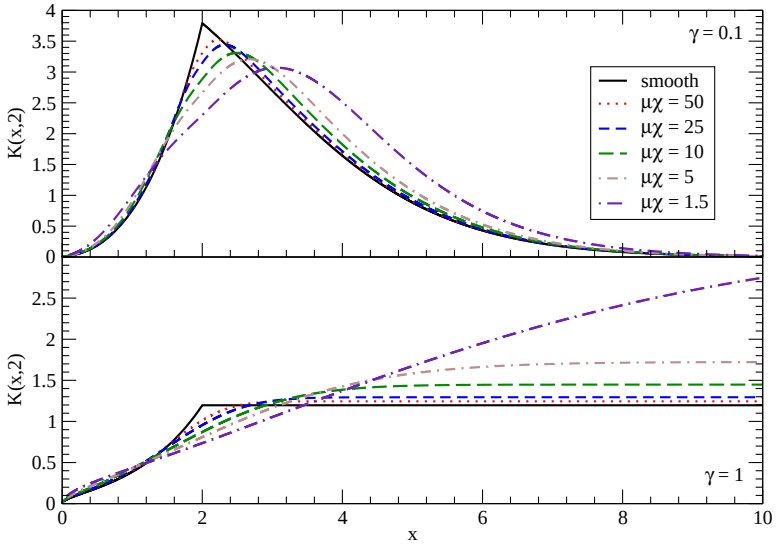

Figure 2.1: The spherical component of the kernel plotted as a function of reduced energy, x, for the initial reduced energy $x' = 2$. The upper panel plots values for the mass ratio $\gamma = 0.1$, and the lower panel for $\gamma = 1$. The values for the smooth hard sphere kernel of Eq. (2.23) are shown by black lines while the coloured lines plot the values for the rough hard sphere kernel of Eq. (2.85) for values of $\mu\chi$ varying from 1.5 to 50.

decreases and the system eventually approaches the smooth sphere results in the infinite limit.

As seen in Fig. 2.1, when the bath and tracer particles have the same mass ($\gamma = 1$) the kernel for the smooth hard sphere has values which extend to large energies, indicating the wide range of energy transfer which can occur upon collision. In comparison, the curve for the approximate rough sphere kernel with $\mu\chi = 1.5$ is shifted to larger energies and is much broader. This reflects the additional energy transfer possibilities in the rough case due to translational-rotational coupling, which is strongest for this value of $\mu\chi$. As the value of $\mu\chi$ is increased, the curves for the rough sphere kernel shift to the left, and begin to adopt the shape of the smooth sphere curve. For $\mu\chi = 50$, the curves are very close, with the largest differences appearing near the cusp at $x = 2$. When the tracer is ten times heavier than a bath particle ($\gamma = 0.1$), the upper panel of Fig. 2.1 shows that the kernel values for the smooth sphere are strongly peaked near $x = 2$, therefore, the probability is greatest for small energy transfers, as expected from the kinematic constraints imposed by the mass difference. The curve for the approximate rough hard sphere kernel with $\mu\chi = 1.5$ is shifted to the right and is broadened. However, this shift and broadening is not as pronounced as it was for the $\gamma = 1$ case. Again, the rough hard sphere values approach the smooth ones as $\mu\chi$ increases.

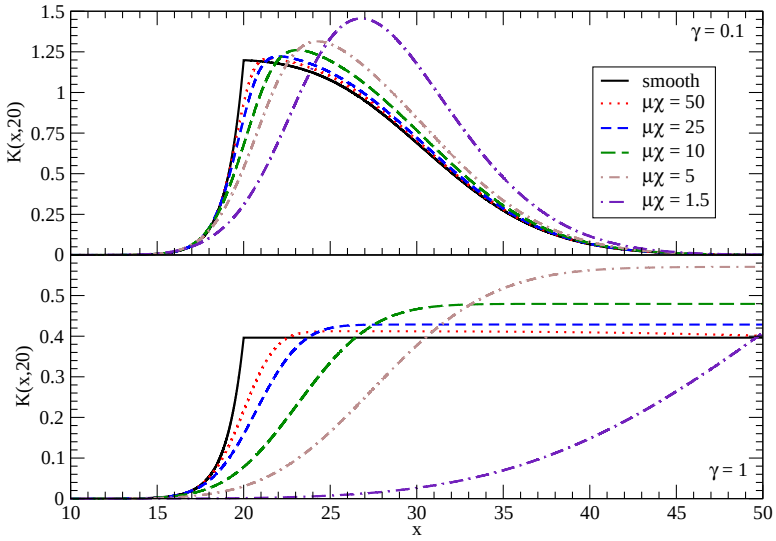

Figure 2.2: Same as Fig. 2.1 except for an initial reduced energy $x' = 20$.

A similar comparison is made in Fig. 2.2 except for a higher energy. In this case, one is examining tracer energies of $x = 20$ corresponding to values 20 times higher than the average thermal bath energy. For both $\gamma = 1$ and $\gamma = 0.1$ the smooth sphere results show a cusp at $x = 20$ with values dropping off rapidly for $x < 20$ and less rapidly for $x > 20$. As in Fig. 2.1, the approximate rough sphere kernel values approach the smooth ones as the value of $\mu\chi$ increases. For $\mu\chi = 1.5$, the curve for $\gamma = 0.1$ is smooth and shifted slightly to the right, again reflecting the added energy exchange possible between the translational and rotational degrees of freedom. This curve gradually shifts to the left and becomes more asymmetric as $\mu\chi$ increases. In contrast, for $\gamma = 1$ the curve for $\mu\chi = 1.5$ is significantly different from the smooth sphere result, peaking at values far to the right of the plotted axis. This curve and to a certain extent the corresponding curve at low energy in Fig. 2.1 show unrealistic behaviour as a result of using the approximate kernel for a mass ratio that is too small. Recall that the approximate kernel was based upon the physical assumption that the rotational degrees of freedom of the tracer remained in equilibrium. This assumption should be poor unless the tracer is heavy, that is, $\gamma \ll 1$. The curves in Figs. 2.1 and 2.2 show this to be true.

2.6 Summary of Collision Kernels

For the linear Boltzmann equation describing the distribution for a spherical tracer particle of low concentration in a bath gas, an expression for the scattering kernel (Eq. (2.17)) has been derived in general. The kernel was also expanded in an angular basis, and the explicit

expression for the spherically averaged component was given (Eq. (2.37)). Although these results are well known, the presentation given here collects various bits and pieces from different sources into a single, complete result. Specific expressions for the smooth hard sphere model were also given.

As a new result, the general kernel (Eq. (2.66)) for the rough hard sphere model was derived. This kernel depends upon both translational and angular velocities, and has a number of constraints which must be imposed. Although more complicated than the smooth sphere result, it is possible to express the rough sphere kernel in closed form. In this model, the degree of translational-rotational coupling is controlled by a single parameter $\mu\chi$, related to the reduced moments of inertia of the bath and tracer particles.

To simplify the expressions somewhat, an approximation to the exact rough sphere kernel was introduced by treating the rotational degrees of freedom of the tracer as an equilibrium bath. In this approximation, the angular velocity dependence in the full kernel can be treated analytically, resulting in a reduced kernel (Eq. 2.82)) depending only upon translational degrees of freedom. This kernel was also expanded in an angular basis and the explicit expression for the spherically-averaged component was given (Eq. (2.85)). This latter result tends to the smooth sphere result when $\mu\chi$ approaches infinity, which is the limit where translational-rotational energy exchange tends towards zero. In that limit, the approximation using in deriving Eq. (2.82) becomes exact. The approximate kernel should only be used for treating systems with heavy tracers, that is small values of γ, since it becomes unphysical for large γ.

Due to non-zero scattering in the forward direction, the general spherical kernel contains a cusp at $x = x'$ and a point discontinuity at $x = x' = 0$. When casting equations into a numerical algorithm, special care must be used to avoid complications due to these two features. In particular, the accurate integration of the kernel near $x = x'$ can be challenging. Because of the nature of the collision dynamics in the rough hard sphere model, the approximate kernel has neither a cusp nor the point discontinuity for all finite $\mu\chi$.

Although the degree of translational-rotational coupling in the rough hard sphere model is greater than that in typical molecules, it is a good model for studying the effects of inelasticity.

Table 2.1: Summary of the collision kernel $K(x,x')$ and collision frequency $\nu(x)$, for the smooth hard sphere, hard sphere and the rough hard sphere models [76].

	Collision kernel, $K(x,x'))$	Collision frequency, $Z(x)$
Smooth Hard Sphere, $\tilde{K}_{sph}(x,x')$	$\frac{AQ^2}{\sqrt{x'\gamma}} e^{(Q-R)(Rx'-Qx)} \int_{\lvert\sqrt{x'}-\sqrt{x}\rvert}^{\sqrt{x'}+\sqrt{x}} dz e^{QRz^2} \times$ $\int_0^\infty ds e^{-s} F(x,x',z;\gamma)$	$\frac{A}{\gamma\sqrt{x}} \int_0^\infty s^2 \tilde{\sigma}_{int}\left(\sqrt{\frac{2kT}{m_1}}s\right) \left\{\exp\left[-\left(s-\sqrt{x\gamma}\right)^2\right]\right.$ $\left. -\exp\left[-\left(s+\sqrt{x\gamma}\right)^2\right]\right\} ds$
Hard Sphere (Wigner-Wilkins), $\tilde{K}_{hs}(x,x')$	$\frac{AQ^2}{2}\sqrt{\frac{\pi}{x'}} \left\{\Phi\left(Q\sqrt{x}+R\sqrt{x'}\right) + e^{x'-x}\Phi\left(R\sqrt{x}+Q\sqrt{x'}\right)\right.$ $\left. \pm \left[\Phi\left(Q\sqrt{x}-R\sqrt{x'}\right) + e^{x'-x}\Phi\left(R\sqrt{x}-Q\sqrt{x'}\right)\right]\right\}$	$\frac{A}{\sqrt{\gamma}} \left\{\left[2\sqrt{x\gamma}+\frac{1}{\sqrt{x\gamma}}\right] \frac{\sqrt{\pi}}{2}\Phi\left(\sqrt{x\gamma}\right) + e^{-x\gamma}\right\}$
Rough Hard Sphere, $\tilde{K}_{rhs}(x,x')$	$\frac{AQ^2}{\sqrt{x'\gamma}} e^{(Q-R)(Rx'-Qx)} \int_{\lvert\sqrt{x'}-\sqrt{x}\rvert}^{\sqrt{x'}+\sqrt{x}} dz e^{-QRz^2} \times$ $F_1(x,x',z;\gamma,\mu\chi)$	$Z_{rhs}(x) = Z_{hs}(x)$

Table 2.2: Summary of the limiting cases of the collision kernel $K(0,x')$ and $K(x,0)$ for the smooth hard sphere, hard sphere and rough hard sphere models [76].

	$K(x,0)$	$K(0,x')$
Smooth Hard Sphere, $K_{sph}(x,x')$	$\frac{2AQ^2}{\sqrt{\gamma}}\int_0^\infty dse^{-s}\times$ $\tilde{\sigma}\left[\sqrt{\frac{2kT}{m_1}}\sqrt{s+Q^2x},\cot^{-1}\left(\frac{\sqrt{s}}{Q\sqrt{x}}\right)\right]$	0
Hard Sphere (Wigner-Wilkins), $K_{hs}(x,x')$	$\frac{2AQ^2}{\sqrt{\gamma}}e^{-R^2x}$	0
Rough Hard Sphere, $K_{rhs}(x,x')$	$\frac{2AQ^2(1+\mu\chi)^2}{\sqrt{\gamma}\mu\chi(1+\mu\chi-\gamma)}e^{-Q^2x}\times$ $\left[1-\exp\left\{-\frac{\mu\chi(1+\mu\chi-\gamma)}{[1+(1+\gamma)\mu\chi]}Q^2x\right\}\right]$	0

2.7 Maxwell Molecule

The Maxwell model is developed for particles that interact via a $1/r^4$ potential [2, 77, 78] and is used widely in kinetic theory. In comparison to other more general models, this model is special because the eigenvalues are known.

Unlike the spherical and rough hard sphere cases which represent specular and diffuse scattering [12], the Maxwell molecule represents a more realistic potential for molecules, for which the cross section was evaluated from [77, 78]

$$W_r bdb = W_r\sigma(W_r,\chi)\sin\chi d\chi = -\gamma(\chi)\cos(\chi/2)d\chi\ , \tag{2.98}$$

where

$$\gamma(\chi) = 4\sin(\chi/2)\sigma(W_r,\chi)W_r = \left(\frac{A_4}{kT}\right)^{1/2}\frac{s_0ds_0}{\cos(\chi/2)d\chi}\ , \tag{2.99}$$

and

$$s_0 = s_0(W_r) = b\left(\frac{kTW_r^2}{A_4}\right)^{1/4}\ . \tag{2.100}$$

The value of A_4 is given by

$$A_4 = \frac{V(r)}{r^4}\ , \tag{2.101}$$

where the value of the potential energy $V(r) = V_0$ and inverse potential via which the

Maxwell molecules interact, $r = \sigma_{12}$.

From the expressions [78]

$$\chi(s_0) = \pi - 2\int_0^{s_0} ds \left[1 - s^2 - \left(\frac{s}{s_0}\right)^4\right]^{-1/2} , \tag{2.102}$$

and

$$1 - s_c^2 - \left(\frac{s_c}{s_0}\right)^4 = 0 , \tag{2.103}$$

where $s = b/r$ with b is the impact parameter. Defining new variables $w = s/s_c = t$ and $m = s_c^2 + 1$ and substituting into Eq. (2.102) gives

$$\chi(m) = \pi - 2s_c \int_0^1 \left[1 + mt^4 - (m+1)t^2\right]^{-1/2} dt , \tag{2.104}$$

where the entire integrand is in the form of an elliptic integral. Re-writing Eq. (2.104) in terms of s_c and using the limits of m gives

$$\chi(s_c) = \pi - \frac{2s_c}{\sqrt{2 - s_c^2}} K\left(\frac{1 - s_c^2}{2 - s_c^2}\right) , \tag{2.105}$$

where K is a first order elliptic integral. Defining

$$\frac{1 - s_c^2}{2 - s_c^2} = \sin^2\theta , \tag{2.106}$$

gives $s_c = \sqrt{\cos 2\theta}/\cos\theta$, $s_0 = \sqrt{2\cot 2\theta}$ and Eq. (2.105) becomes

$$\chi(\theta) = \pi - 2\sqrt{\cos 2\theta} K(\sin^2\theta) . \tag{2.107}$$

Differentiating Eq. (2.107) gives,

$$\frac{d\chi}{d\theta} = \frac{4\left[\cos^2\theta K(\sin^2\theta) - \cos 2\theta E(\sin^2\theta)\right]}{\sin 2\theta\sqrt{\cos 2\theta}} , \tag{2.108}$$

where E is the elliptic integral of the second kind given by

$$\frac{dK(m)}{dm} = \frac{[E(m) - (1-m)K(m)]}{2m(1-m)} . \tag{2.109}$$

Substituting Eq. (2.108), the cross section $\sigma(W_r, \chi)$ in Eq. (2.98) can be written as

$$\tilde{\sigma}_{Maxwell}\left(\sqrt{\frac{2kT}{m_1}}\tilde{g}, \chi\right) = 2\sqrt{\frac{V_0}{kT}}\frac{\sqrt{1+\gamma}}{\tilde{g}}\frac{\sqrt{\cos 2\theta}}{\sin\chi \sin 2\theta}\frac{1}{\left[\cos^2\theta K(\sin^2\theta) - \cos 2\theta E(\sin^2\theta)\right]} . \tag{2.110}$$

The classical differential cross section above diverges as θ^5 or $(\chi^{5/2})$ when $\theta \to 0$ ($\chi \to 0$), that is in the forward scattering direction. This is a known problem with all classical differential cross sections for potentials which extend to infinite distances. Correct quantum mechanical differential cross sections do not have this singularity. In fact, the quantum and classical cross sections for the Maxwell molecule agree very well except in the forward direction [79]. Since the cross section divergence occurs as $\chi^{5/2}$, a function is required to multiply with this factor. In addition to removing divergence, this factor should also approach a value of 1 very quickly, in order to keep the values of larger θ unaltered. To overcome this divergence, the cross section is multiplied by the hyperbolic tangent function $\tanh(\beta\chi^{5/2})$, where β is a coefficient that determines how quickly the values of the cross section become finite. A plot of the cross section for different values for β is given in Fig. 2.3. Note that θ ranges from 0 to $\pi/4$ as χ ranges from 0 to π. As the value of β becomes larger, the cross section values are more accurately calculated. The solid black curve shows the actual cross section, which diverges as $\chi \to 0$. Introducing the hyperbolic tangent function forces the cross section values to a finite value at $\chi = 0$. The value of $\beta = 20$ was set in this study because this value removes the singularity at $\chi = 0$ and reproduces the lowest eigenvalues of the Maxwell molecule kernel (which was determined in a separate calculation) to within a few percent. It also gives a strong peak in the forward scattering direction that mimics realistic cross sections and will provide a good test of the numerical method.

For the case of the Maxwell molecule, the scattering angle expression in Eq. (2.107) is evaluated using pre-calculated values of θ on a grid of χ and interpolated as needed. The cusp at $x = x'$ is very prominent for this model, and the integration over this point requires care.

2.8 Fokker-Planck Expressions

As shown by Andersen and Schuler [58] the Boltzmann equation can be cast into a Fokker-Planck equation when γ is either very large (Lorentz limit) or very small (Rayleigh limit). The system used in this study falls under the Rayleigh limit, and therefore from this point forward, Rayleigh limit expressions and results will be presented. For the Rayleigh gas, the relaxation of the mean energy is given as [58] (in the present notation),

$$\frac{d\langle x\rangle(t')}{dt'} = -k_R[\langle x\rangle(t') - \langle x\rangle(\infty)] = -k_R[\langle x\rangle(t') - 3/2] , \tag{2.111}$$

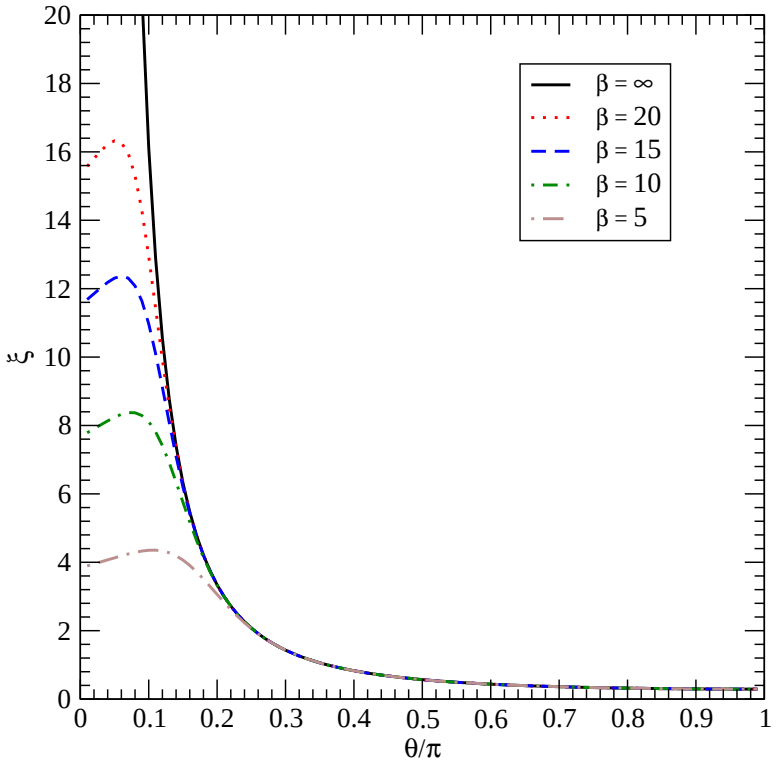

Figure 2.3: The value of $\xi = \tilde{g}/2 \cdot \sqrt{kt/V_0} \cdot 1/\sqrt{1+\gamma}\tilde{\sigma}_{Maxwell}$ from Eq. (2.110) for values of $\beta = 5$, 10, 15, 20. The solid curve for $\beta = \infty$ corresponds to Eq. (2.110) directly.

with $k_R = 16\sqrt{\gamma}/3$ for any initial distribution function. The above results are valid when

$$\gamma < \langle x \rangle_0 < \gamma^{-1} \,, \tag{2.112}$$

and $0 \leq \langle x \rangle_0 \leq \infty$ with $\langle x \rangle_0$ the initial energy of the system. Using the B-spline numerical scheme, we wish to determine the validity of Eq. (2.111), and to identify the range where this expression breaks down.

2.9 Drag Coefficients

The drag force on a tracer is given by

$$\vec{F} = m\frac{d\vec{v}}{dt} = -\frac{C_D \mathcal{A} n_1 m_1 v}{2}\vec{v} \,, \tag{2.113}$$

in which v is the net speed and $\mathcal{A}$ the cross sectional area [12] of the tracer. Drag coefficients, C_D, are obtained by considering the momentum transfer during a collision and integrating over all scattering angles and a thermal distribution of speeds. Taking the derivative of the net average kinetic energy for the tracer $E_k - (3/2)kT = mv^2/2$, setting $\mathcal{A} = \pi\sigma_{12}^2$ and using the definition of the collision frequency factor, A, and Eq. (2.113) gives

$$\frac{dE_k}{dt} = -\mathcal{A} C_D \gamma \sqrt{\frac{\pi}{kT}} \left(E_k - \frac{3}{2}kT \right)^{3/2} \,, \tag{2.114}$$

which when converted to reduced units gives

$$\frac{d\langle \tilde{x} \rangle}{dt'} = -C_D \gamma \sqrt{\pi} \langle \tilde{x} \rangle^{3/2} \,, \tag{2.115}$$

with $\langle \tilde{x} \rangle = \langle x \rangle (t') - 3/2$. The time t' is scaled using the collision frequency, given as, $t' = At$. The speed ratio s of the tracer is then $s = v\sqrt{m_1/2kT} = \sqrt{\gamma \langle \tilde{x} \rangle}$.

For $s \ll 1$, Epstein [80] has given the drag coefficients for specular and diffuse scattering as

$$\begin{aligned} C_{sp,ep} &= \frac{16}{3\sqrt{\pi}} \frac{1}{s} \,, \\ C_{diff,ep} &= \frac{13}{9} C_{sp,ep} \,. \end{aligned} \tag{2.116}$$

Specular scattering is considered for collisions that occur between smooth hard sphere, whereas the diffuse scattering occurs for collisions between rough hard spheres. Expressions

for general speed ratios are [81]

$$C_{sp} = \frac{2}{\sqrt{\pi}}\frac{e^{-s^2}}{s}\left(1+\frac{1}{2s^2}\right)+2\left(1+\frac{1}{s^2}-\frac{1}{4s^4}\right)\Phi(s)\,, \tag{2.117}$$

$$C_{diff} = \frac{2\sqrt{\pi}}{3s}+C_{sp}\,,$$

in which $\Phi(s)$ is the error function. These expressions are within a few percent of the Epstein ones when $s \leq 0.05$. Note that smooth hard spheres undergo specular scattering only and $C_{sp,ep}$ substituted into Eq. (2.115) gives Eq. (2.111). This is consistent from the derivation of the Fokker-Planck equation that assumes a heavy tracer moving within a specified range of energies. From known diffusion coefficients for the rough hard sphere, one can show that $C_{rough} = [(2+\mu\chi)/(1+\mu\chi)]C_{sp}$. For $\mu\chi = 1.5$, which corresponds to the maximum amount of translational-rotational coupling this gives $C_{rough} = (7/5)C_{sp}$ which is within a few percent of $(13/9)C_{sp}$. Thus, the rough hard sphere model with $\mu\chi = 1.5$ can be used to model diffuse scattering conditions. More precisely, $C_{rough} = 7/5 \cdot 9/13 \cdot C_{diff}$.

Chapter 3

Methodology of Numerical Method and Cubic B-Splines

A detailed description of the numerical method is given in this chapter. Section 3.1 gives details on four different transformations that were used for the collision kernels to make numerical convergence easier. Each of these transformations is advantageous in different scenarios, which are discussed. Section 3.2 gives details on the construction of the numerical method using B-spline polynomials as a basis set. These polynomials are used to expand the distribution function. Also given are the conditions and parameters used to validate the method.

3.1 Varied Collision Kernels

The collision kernel was transformed into four different forms. This was done in Ref. [82] for the hard sphere collision kernel, but these transformations are general and can be used for any of the models discussed. This was done to avoid certain terms in the kernel expressions that could cause difficulties with numerical convergence.

For the general spherical kernel of by Eq. (2.40). The equilibrium distribution function is

$$f^{(0)}(x) = 2\sqrt{\frac{x}{\pi}}e^{-x} \ . \tag{3.1}$$

From a numerical point of view the exponential and root terms in Eq. (2.40) can be problematic, and to bypass these two the following three forms of the distribution function were considered,

$$f(x,t) = 2\sqrt{\frac{x}{\pi}}e^{-x}f_1(x,t) \ , \tag{3.2}$$

$$f(x,t) = 2\sqrt{\frac{x}{\pi}}e^{-f_2(x,t)} \ , \tag{3.3}$$

$$f(x,t) = 2\sqrt{\frac{x}{\pi}}f_3(x,t) \ , \tag{3.4}$$

where, at equilibrium $f_1^{(0)}(x,t) = 1$, $f_2^{(0)}(x,t) = x$ and $f_3^{(0)}(x,t) = e^{-x}$. Substituting

Eq. (3.2) or (3.4) into Eq. (2.35) gives

$$\frac{\partial}{\partial t} f_\alpha(x,t) = \int_0^\infty K_\alpha(x,x') f_\alpha(x',t) dx' - Z(x) f_\alpha(x,t) \ , \tag{3.5}$$

in which $\alpha = 1, 3$ and

$$K_1(x,x') = \sqrt{\frac{x'}{x}} e^{x-x'} \tilde{K}(x,x') \ , \tag{3.6}$$

$$K_3(x,x') = \sqrt{\frac{x'}{x}} \tilde{K}(x,x') \ . \tag{3.7}$$

Similarly substituting Eq. (3.3) into Eq. (2.35) gives

$$\frac{\partial}{\partial t} f_2(x,t) = -\int_0^\infty K_2(x,x') e^{f_2(x,t) - f_2(x',t)} dx' + Z(x) \ , \tag{3.8}$$

To test these various transformations, the smooth hard sphere kernel $\tilde{K}_{hs}(x,x')$ of Eq. (2.43) was employed. Following the same analogies as in Section 2.6 surrounding Eqs. (2.89) and (2.90) gives

$$K_{hs,1}(x,0) = K_{hs,3}(x,0) = 0 \ , \tag{3.9}$$

$$K_{hs,1}(0,x') = 2\frac{AQ^2}{\sqrt{\gamma}} e^{-R^2 x'} \ , \tag{3.10}$$

$$K_{hs,3}(0,x') = 2\frac{AQ^2}{\sqrt{\gamma}} e^{-Q^2 x'} \ . \tag{3.11}$$

Similarly, Eq. (2.39) gives $Z_{hs}(0) = 2A/\sqrt{\gamma}$.

3.2 Analytical Details of the Algorithm

A computer algorithm is developed in Fortran 90 to test the analytical results presented in Chapter 2. This method is constructed using B-splines and was published in [82] and [83]. B-splines are *basis splines* [84–87] that are simple piece-wise, continuous polynomial functions. In the method developed here, third order polynomial functions have been used, therefore cubic B-splines. The order of the B-splines can be increased by odd powers and therefore quintic or septic B-splines can also be used. This would require modification of the structure of the matrices and boundary conditions, but the general layout of the numerical method does not change. Using a higher order set of B-splines would improve accuracy, but the results obtained using third order B-splines have given accuracy sufficient for the purpose of developing this method. B-spline functions are constructed over a finite domain.

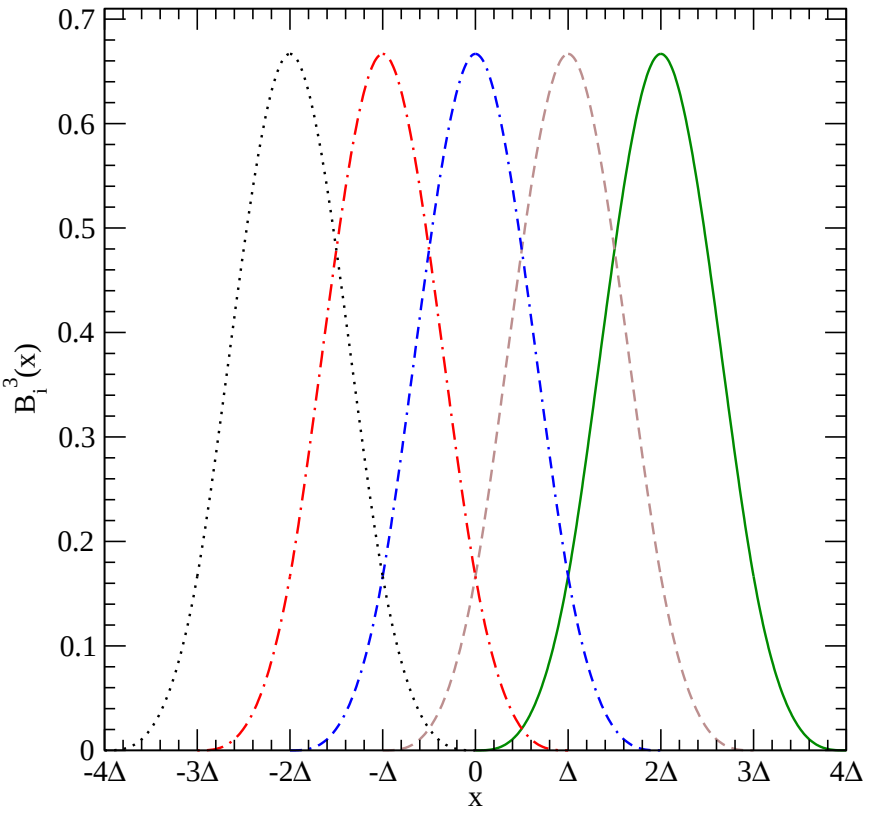

Figure 3.1: Cubic B-spline functions, $B_i^3(x)$, for $i = -4, -3, -2, -1, 0$ defined on a grid of x values.

As an example, the set of B-spline functions used in this study are

$$B_i^3(x) = \frac{1}{6}\begin{cases} w_i^3, & 0 \le w_i < 1 \\ w_i^3 - 4(w_i - 1)^3, & 1 \le w_i < 2 \\ (4 - w_i)^3 - 4(3 - w_i)^3, & 2 \le w_i < 3 \\ (4 - w_i)^3, & 3 \le w_i \le 4 \\ 0, & \text{otherwise} \end{cases} . \tag{3.12}$$

in which $w_i = (x/\Delta) - i$. A graphical representation of these functions is given in Fig. 3.1. In this methodology, the B-spline functions are defined over a grid of equally spaced knots or intervals, but in general, these functions are flexible enough to be described over arbitrarily spaced intervals. The space between each interval is Δ (which should not be confused for Δ given in Eq. (2.42)). As can be seen in Fig. 3.1, the B-spline functions span over four units on the grid, and outside the domain of four units, the functions have a value of 0. The index i labels the left most point of the spline function and matches it to the point on the grid. Simply put, for $i = 0$, $B_0^3(x)$ begins at $x = 0$ and extends to $x = x_4 = 4\Delta$. In the

present case, the grid over which the B-spline functions are described starts from $x_0 = 0$ and extends to $x_n = S$, a finite value. From Fig. 3.1, it should be noted that for all $i \geq n$ extending past the end of the grid, $B_i^3(x)$ will not contribute to f. At the same time, one should note there will be some contribution to f at grid values beyond $x = x_0$ from $B_{-3}^3(x)$, $B_{-2}^3(x)$, and $B_{-1}^3(x)$. Therefore, the index i in Eq. (3.12) ranges from $i = -3$ to $i = n - 1$, to fully include the cubic B-splines contributing to the distribution function values on the grid. Although there are B-splines off the grid that contribute in the evaluation of the coefficients, that is, with index $i = -3, -2$ and -1, the evaluation of the coefficients are confined to the grid. Calculations that extend to the left of the defined grid give un-physical values, and therefore do not contribute to the calculations.

The unknown distribution function $f(x,t)$ is expanded in terms of the B-spline functions of Eq. (3.12) [82],

$$f(x,t) \doteq \sum_{i=-3}^{n-1} c_i(t) B_i^3(x) \,, \tag{3.13}$$

where $c_i(t)$ are time-dependent coefficients and f can be either f_1, f_2 or f_3. Substituting Eq. (3.13) into Eq. (3.5) gives

$$\sum_{i=-3}^{n-1} \frac{dc_i(t)}{dt} B_i^3(x) = \sum_{i=-3}^{n-1} c_i(t) \left[\int_0^S K(x,x') B_i^3(x') dx' - Z(x) B_i^3(x) \right] \,, \tag{3.14}$$

in which K can be any of $\tilde{K}_{hs}$, K_1, K_2, or K_3 or the more general formulations $\tilde{K}$ and $\tilde{K}_1$, given by Eqs. (2.40) and (2.85).

Using a collocation scheme, the equality in Eq. (3.14) is forced to hold at the grid points $x = x_j = j\Delta$, where $j = 0, \ldots, n$, giving $n + 1$ constraints for the time dependent coefficients, $c_i(t)$. Two additional constraints are required in this formulation since there are $n + 3$ unknown coefficients. To obtain these two extra constraints, a very general and flexible process is used.

Constraint 1: One possibility is to enforce the equality in Eq. (3.14) at the midpoint between the first and the second point, $x_{1/2}$, on the grid and the midpoint between the last and second-to-last points, $x_{n-1/2}$ on the grid. Evaluating Eq. (3.14) at $x = x_{1/2}$, gives

$$\sum_{i=-3}^{n-1} \frac{dc_i}{dt} B_i^3(x_{1/2}) = \sum_{i=-3}^{n-1} c_i \int_0^S K(x_{1/2},x') B_i^3(x') dx' - \sum_{i=-3}^{n-1} c_i Z(x_{1/2}) B_i^3(x_{1/2}) \tag{3.15}$$

which reduces to

$$\frac{1}{6}\left[\frac{1}{8}\frac{dc_{-3}}{dt}+\frac{23}{8}\frac{dc_{-2}}{dt}+\frac{23}{8}\frac{dc_{-1}}{dt}+\frac{1}{8}\frac{dc_0}{dt}\right]=\sum_{i=-3}^{n-1}K_{1/2i}c_i-\frac{Z_{1/2}}{6}\left[\frac{1}{8}c_{-3}+\frac{23}{8}c_{-2}+\frac{23}{8}c_{-1}+\frac{1}{8}c_0\right]\ . \tag{3.16}$$

Equation (3.16) is obtained by considering values of i where $B_i^3(x_{1/2})$ is non-zero, that is, $i = -3, \ldots, 0$ and calculating the values of the spline functions using Eq. (3.12). An expression for $x_{n-1/2}$ is obtained in a similar fashion.

Constraint 2: Obtaining the constraints is a very flexible method, therefore as a second choice the spatial and temporal derivatives of f, can also be chosen, that is,

$$\frac{\partial^2}{\partial x \partial t}f(x,t)=\sum_{i=-3}^{n-1}\frac{d}{dt}c_i(t)\frac{d}{dx}B_i^3(x)\ . \tag{3.17}$$

In this constraint the derivatives are forced to be zero at the grid end points, $x = x_0$, x_n. With these constraints set, the first and last rows of the **K** and **Z** matrices have zeroes as their elements and the corresponding elements of matrix **B** are derived from the known values of the spline derivatives at the grid points, again determined from Eq. (3.12). This constraint is not used in the present work, but given to show the flexibility of the method. This constraint was tried but does not work well because it does not match the behaviour of the distribution function well enough. For this reason, this constraint was abandoned. The eigenvalues obtained using constraint 2 were not accurate when compared to those in the literature. This occurred because the derivative of the B-splines was forced to be zero at the given points. The derivatives of the B-splines may not have this value at the points and this introduces a discrepancy and affects the accurate evaluation of the eigenvalues.

With either of these constraints, Eq. (3.14) can be cast into matrix notation as

$$\frac{d\mathbf{c}}{dt}=\mathbf{L}\mathbf{c}\ , \tag{3.18}$$

with

$$\mathbf{L}=\mathbf{B^{-1}}(\mathbf{K}-\mathbf{Z}\mathbf{B})\ . \tag{3.19}$$

Since the matrix **L** is time independent, the formal solution of Eq. (3.18) is

$$\mathbf{c}_i(t)=e^{-\mathbf{L}\mathbf{t}}\mathbf{c}(t=0)=\mathbf{U}e^{\mathbf{\Lambda} t}\mathbf{U}^{-1}\mathbf{c}(t=0)\ , \tag{3.20}$$

with

$$\mathbf{\Lambda}=\mathbf{U}^{-1}\mathbf{L}\mathbf{U}\ , \tag{3.21}$$

in which $\Lambda_{ij} = \lambda_i\delta_{ij}$ with λ_i the eigenvalues of the matrix **L** and **U** its transformation

matrix.

In the f_2 case, the formulation given above does not apply because writing f_2 as part of the exponent term destroys the linearity with respect to the spline coefficients. Therefore, in this particular case the matrix formulation of Eq. (3.8) becomes

$$\frac{d\mathbf{c}}{dt} = \mathbf{B^{-1}}(\tilde{\mathbf{Z}} - \tilde{\mathbf{K}}) \; , \tag{3.22}$$

where $\mathbf{B}$ is a square matrix and $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{Z}}$ are column vectors. The column matrices $\tilde{\mathbf{Z}}$ and $\tilde{\mathbf{K}}$ have elements $\tilde{\mathbf{Z}} = Z_j$ and

$$\tilde{K}_j = \int_0^S K_3(x_j, x') e^{[f_{hs,2}(x_j,t) - f_{hs,2}(x',t)]} dx' \; . \tag{3.23}$$

For this case a simple finite difference scheme is used to integrate Eq. (3.22), namely

$$\mathbf{c}(t + \Delta t) = \mathbf{c}(t) + \frac{d\mathbf{c}}{dt}\Delta t \; . \tag{3.24}$$

Equation (3.24) can also be used to integrate Eq. (3.18).

In the evaluation of the time dependent coefficients, the first set of coefficients are evaluated at $t = 0$, which are further used in the iterations above. The matrices $\mathbf{B}$, $\mathbf{K}$, and $\mathbf{Z}$ are square matrices with dimensions $(n+3) \times (n+3)$, whereas for the f_2 formulation, $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{Z}}$ have dimensions $(n+3) \times 1$.

To show a more detailed derivation of the matrix formulation, consider the form of Eq. (3.14) when $x = x_j$, that is

$$\sum_{i=-3}^{n-1} \frac{dc_i}{dt} B_i^3(x_j) = \sum_{i=-3}^{n-1} c_i \int_0^S K(x_j, x') B_i^3(x') dx' - \sum_{i=-3}^{n-1} c_i Z(x_j) B_i^3(x_j)$$

$$\frac{1}{6}\left[\frac{dc_{j-3}}{dt} + 4\frac{dc_{j-2}}{dt} + \frac{dc_{j-1}}{dt}\right] = \sum_{i=-3}^{n-1} K_{ji} c_i - \frac{Z_j}{6}[c_{j-3} + 4c_{j-2} + c_{j-1}] \; , \tag{3.25}$$

in which

$$K_{ji} = \int_0^S K(x_j, x') B_i^3(x') dx' \; . \tag{3.26}$$

The second line of Eq. (3.25) is obtained using the values in Eq. (3.12) since $B_i^3(x_j)$ is non-zero only for $i = j-3$, $j-2$ and $j-1$. Mapping the above equations to matrix notation is done as follows. If k and l are taken to be the indices for the rows and columns of an $(n+3)$ $\times$ $(n+3)$ square matrix, respectively, choose $k = 2, \ldots, n+2$ to correspond with $j = 0$, $\ldots$, n and $l = 1, \ldots, n+3$ to correspond with $i = -3, -2, \ldots, n-1$. Therefore, $k = j+2$ and $l = i+4$. This provides a mapping from (j, i) labels to matrix indices (k, l). The values

for $k = 1$ and $k = n + 3$ are used to assign the values of the two extra constraints. The matrices used in Eq. (3.19) with constraint 1, then become

$$\begin{aligned}
\mathbf{B} &= \frac{1}{6}\begin{pmatrix}
\frac{1}{8} & \frac{23}{8} & \frac{23}{8} & \frac{1}{8} & \dots & 0 & 0 \\
1 & 4 & 1 & 0 & \dots & 0 & 0 \\
0 & 1 & 4 & 1 & \dots & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\
0 & 0 & \dots & 1 & 4 & 1 & 0 \\
0 & 0 & \dots & 0 & 1 & 4 & 1 \\
0 & 0 & \dots & \frac{1}{8} & \frac{23}{8} & \frac{23}{8} & \frac{1}{8}
\end{pmatrix}, \\
\mathbf{Z} &= \begin{pmatrix}
Z_{1/2} & 0 & 0 & \dots & 0 & 0 \\
0 & Z_0 & 0 & \dots & 0 & 0 \\
0 & 0 & Z_1 & \dots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \dots & Z_n & 0 \\
0 & 0 & 0 & \dots & 0 & Z_{n-1/2}
\end{pmatrix}, \\
\mathbf{c} &= \begin{pmatrix}
c_{-3} \\ c_{-2} \\ c_{-1} \\ c_0 \\ \vdots \\ c_{n-1}
\end{pmatrix}. \qquad (3.27)
\end{aligned}$$

To obtain the initial coefficients $c_i(t = 0)$, Eq. (3.13) is used,

$$\mathbf{c}_i(t = 0) = \mathbf{B}^{-1} f \ , \qquad (3.28)$$

where $f_k = f(x_j, t = 0)$ with $j = k - 2$ for $k = 2$, ..., $n + 2$ and $j = 1/2$ for $k = 1$ and $j = n - 1/2$ for $k = n + 3$ which match the boundary conditions using constraint 1.

3.2.1 Evaluation of Z

Ideally, if the grid is infinitely large, that is if S is infinite, then $Z_j = Z(x_j)$, with $Z(x)$ given by the definition of A for the collision frequency. This equality between $Z(x)$ and the integral of the collision kernels in Eqs. (3.5) or (3.8) over x' must be held in order to maintain a constant norm for the distribution function. At equilibrium in this limit $\mathbf{K} = \mathbf{ZB}$. But, since S is finite in the numerical method, this equality breaks down at equilibrium and the norm may not be preserved if Eq. (2.38) is used for evaluating $Z(x)$. To overcome this, the

values of Z are calculated using

$$Z_j = \int_0^S K_i(x_j, x')dx' \ . \tag{3.29}$$

This forces the norm to be preserved but does not guarantee $\mathbf{K} = \mathbf{ZB}$ will also be satisfied at equilibrium. Only when the representation of the equilibrium function is accurate enough, will this equality be satisfied.

3.3 Initial Functions and Conditions

Several initial functions were used to test the numerical method, namely a Maxwellian, Gaussian and bimodal functions. The Maxwellian function was set at different temperatures as compared to the temperature of the bath. The results obtained with this particular form of the initial function are presented in Chapter 4. The form of Maxwellian function used is

$$f(x, t = 0) = 2\sqrt{\frac{x}{\pi}}\alpha^{3/2}e^{-\alpha x} \ , \tag{3.30}$$

where α is the temperature ratio of the tracer to the bath. The form of Eq. (3.30) is given for the f_{hs} formulation and is appropriately transformed into the $f_{hs,1}$, $f_{hs,2}$ and $f_{hs,3}$ formulations using Eqs. (3.2), (3.3) and (3.4). To validate the flexibility of the numerical method, a more extreme initial function with a bimodal distribution was used, representing peaks at two different energies, with the form

$$f(x, t = 0) = \exp[-0.2(x-2)^2] + \exp[-0.2(x-8)^2] \ . \tag{3.31}$$

The Gaussian distribution function used to obtain the results given in Chapter 5 is

$$f(x, t = 0) = e^{-(x-x_0)^2} \ , \tag{3.32}$$

where x_0 is the initial energy of the system.

3.4 Use of Quadratures

Any numerical scheme can be used to evaluate the integrals given in the collision kernel expressions in Eqs. (2.40), (2.85), (3.15), or (3.29). For the results presented in Chapter 4 Gauss-Legendre quadrature schemes were used. For the results presented in Chapter 5 for the integrations over z in the collision kernel expressions, once again Gauss-Legendre quadratures were used, and those integrals over the variable s were done using Gauss-

Laguerre quadrature. Since the domain of $B_i^3(x)$ is finite, the integrand is non-zero only over a very narrow range on the grid. Therefore, the use of the Gauss-Legendre quadrature scheme is best suited. Because the piecewise continuous nature of the B-splines affects convergence of the quadrature scheme, the integrals are divided into pieces extending over a single grid width Δ. This ensures that each integral is considered to be a separate entity that is independent of all other intervals. Some intervals may require more points than others to converge, and by separating the integrals into smaller intervals the appropriate number of points can be used in the evaluation. The number of quadrature points are increased until the difference in the integrated values is less than a certain tolerance, typically set at 10^{-13} for results in Chapter 4 and 10^{-9} for results in Chapter 5. Using B-splines provided an advantage in dealing with problematic integrands, such as the cusp at $x = x'$ in the collision kernel, in a localized manner.

3.5 Code Details

The code was written in Fortran 90 and is about 5500 lines. The code was divided into subroutines, functions and modules that evaluate individual parts of the calculation. The code requires the use of some routines given in Numerical Recipes [88]. These subroutines include *polint* for polynomial interpolation, *gaulag* for the evaluation of the Gauss-Laguerre quadrature abscissae and weights and *gauss* for the evaluation of the Gauss-Legendre quadrature abscissae and weights. Using these routines, the points and weights of the quadratures were evaluated at the very beginning of the program and stored. These quadratures were used as needed in the evaluation of the individual integrals. A level of tolerance was set and the evaluated values of the integrals were checked against these tolerances to confirm their convergence. If convergence was not obtained, the number of points and weights was increased. For some calculations, the number of points for the quadratures was set at a particular value. The values obtained with this condition converged well, and thus allowed the higher level integration to use quadrature points as needed for convergence.

To parallelize certain parts of the code, especially large matrix evaluations, OPENMP routines were used. This allowed the use of multiple processors for calculations. The evaluation of the matrices in Chapter 4 took anywhere from a few seconds to about 4 hours. The results for the f_1 case completed the fastest and those for the f_{hs} case took the longest amount of time. The matrices ranged in size from 20×20 to 1000×1000. For the results in Chapter 5, parallelization was introduced. This paralellization was done for evaluation of the matrix $\mathbf{L}$ in Eq. (3.19). A separate program was written to complete the evaluation of matrix $\mathbf{K}$ in Eq. (3.19). In this the elements of $\mathbf{K}$ were evaluated and then the entire matrix was read into the main program. Using two-dimensional linear interpolation, the correct value at the points of evaluation was obtained, and then further used in the

calculation. The evaluation of matrix **K** would take upto three days, depending on how large the matrices were. The largest matrix that was evaluated was 2000×2000. This method of evaluating the matrix **L** was later abandoned since the tolerance on convergence was relaxed in the lower level evaluations of the integrals. This allowed only the main program to be used for all calculations. The smallest matrix evaluated was 100×100 and the largest 2000×2000. The calculations run for the smooth hard sphere were completed between 30 minutes to about 2 hours depending on the initial energy that used. It would take longer to complete the calculation for the higher energy cases. The rough hard sphere cases were completed between 2 - 6 hours, and the Maxwell model took the longest to run, from 4 - 40 hours. In comparison to other numerical methods available, this method worked quite well in completing the time propagation and giving well converged values. The QDM method would still be much faster than the B-spline method. The number of processors were also controlled in the submission script. The maximum number of processors used were 12, that all ran on the same node.

C subroutines were used for the gamma and error functions. LAPACK subroutines were called for the matrix operations such as matrix decomposition, matrix diagonalization, eigenvalue and eigenvector evaluations. Intel and Portland Group compilers were used to compile and debug the entire code. The code is given in Appendix A.

Chapter 4

Validating the Numerical Method using Eigenvalues and Moments

This chapter gives the results used to test the accuracy of the method given in Chapter 3. A comparison between the eigenvalues and eigenfunctions is shown for the different formulations of the collision kernels discussed in Section 3.1, and these values are also compared to previously published results that were obtained using QDM ([89]). Also given are the distribution functions and moments for the different collision kernel formulations. Using the results the advantages and disadvantages of each formulation of the collision kernel is discussed at the end of the chapter.

4.1 Eigenvalues and Eigenfunctions

For the $\tilde{K}_{hs}$, K_1 and K_3 formulations of the collision kernel, the collision matrix $\mathbf{L}$ in Eq. (3.19) is diagonalized to obtained eigenvalues and corresponding eigenfunctions. The eigenvalue spectrum for a hard sphere gas has been evaluated [65] and shown in this thesis are the discrete values that are less than 1. This discrete spectrum gives an eigenvalue of 0 that is indicative of the equilibrium distribution function. The consecutive values calculated are negative and increase. In order to obtain convergence, the smaller values decrease and the values closer to 1 tend to pile up around this value.

Table 4.1 contains the eigenvalues obtained using the $\tilde{K}_{hs}$ kernel expression in Eq. (2.43). It shows convergence of the first two non-zero eigenvalues for four different mass ratios of $\gamma = 1/8$, $1/2$, 1 and 8. The grid size S and the number of points n vary. The data in Table 4.1 shows the effect of changing the grid size and number of points on the convergence of the eigenvalues. Different combinations of (S, n) are chosen to monitor convergence in multiple ways. Keeping the value of S fixed and increasing values of n shows the effects of decreasing the grid spacing on convergence. In contrast, keeping the ratio S/n constant shows the effect of increasing the size of the grid while maintaining a fixed grid spacing. From Table 4.1, it can be deduced that a grid size S of a maximum of 20 units will be sufficient to obtain convergence for the eigenvalues. By keeping the values of S constant and increasing the values of n the convergence of the eigenvalues improves. This is further

shown in Tables 4.2 and 4.3.

Table 4.1: The first and second non-zero eigenvalues obtained for different mass ratios, $\gamma = 1/8$, $1/2$, 1 and 8, by diagonalizing $\mathbf{L}$ using the matrix representation of the $K_{hs,1}$ kernel of Eq. (3.6). The eigenvalues are normalized by $Z_{hs}(0)$. The grid used for each diagonalization spans from 0 to S with n intervals. The accurate values are QDM results from Shizgal *et al.* [89].

	S	n	$\gamma = \frac{1}{8}$	$\gamma = \frac{1}{2}$	$\gamma = 1$	$\gamma = 8$
$\lambda_1/Z_{hs}(0)$	10	10	0.2796	0.6895	0.8137	0.5641
	10	50	0.2796	0.6896	0.8191	0.6016
	20	20	0.2784	0.6891	0.8188	0.5639
	20	100	0.2784	0.6891	0.8190	0.6132
	30	150	0.2784	0.6889	0.8158	0.5894
Accurate values			0.2784	0.6891	0.8190	0.6139
$\lambda_2/Z_{hs}(0)$	10	10	0.4978	0.9157	0.9778	0.9572
	10	50	0.4984	0.9212	0.9760	0.9385
	20	20	0.4882	0.9199	0.9717	0.9561
	20	100	0.4882	0.9208	0.9789	0.9382
	30	150	0.4882	0.9208	0.9760	0.9382
Accurate values			0.4882	0.9208	0.9797	0.9667

Tables 4.2 and 4.3 show the eigenvalues of the collision kernels in the $\tilde{K}_{hs}$, $K_{hs,1}$ and $K_{hs,3}$ formulations for different mass ratios. In Tables 4.2 and 4.3 the same four mass ratios are tested by varying the number of points n, ranging from $10 - 600$ all with $S = 20$. It is shown that convergence is obtainable by keeping the grid size constant and varying the number of points. The accurate values indicated are obtained using QDM in Ref. [89]. In all three tables the discrete eigenvalues are reported and are normalized by the frequency $Z_{hs}(0)$. Keeping in accordance with significant figures these values are rounded up.

Tables 4.2 and 4.3 show a number of trends. The first non-zero eigenvalue λ_1 is obtained accurately for all mass ratios in the $K_{hs,1}$ and $K_{hs,3}$ formulations. For the smaller mass ratios of $\gamma = 1/8$ and $\gamma = 1/2$, the convergence is obtained very quickly. In the case of the $K_{hs,1}$ formulation fairly accurate eigenvalues are obtained for $n = 2$ even, and the $K_{hs,3}$ formulation requires a minimum of $n = 60$. In comparison, the $\tilde{K}_{hs}$ formulation goes as high as $n = 600$ and is still not converged. As the mass ratios increase, namely $\gamma = 1$ and $\gamma = 8$, a greater number of points are required for convergence. In the $K_{hs,3}$ case the convergence is rapid since the non-polynomial nature of the $\sqrt{x}$ is removed from the distribution function.

Therefore, the convergence is almost as good as the $K_{hs,1}$ case. It should be noted that convergence is the worst for $\tilde{K}_{hs}$ formulation for all mass ratios and for $\gamma = 8$ the values are even less converged at this limit. The trends are similar for the second and third non-zero eigenvalues, except they are correspondingly more difficult to fully converge. This is especially noticeable for the largest mass ratio $\gamma = 8$ where it becomes more difficult to obtain eigenvalues with a small number of points. In fact, for this particular mass ratio λ_3 was unobtainable until a minimum value of $n = 600$ was used. Even setting n to a high number, the eigenvalue was not converged to the exact value, but it is possible to obtain the exact value by further increasing the number of points.

Overall, the values given in Table 4.1 show similar patterns to those in Table 4.2 and 4.3. Between Tables 4.1 and 4.2 and 4.3, it can be confirmed that the B-spline method is stable and does converge to the exact eigenvalues, but the rate of convergence is heavily dependent on the particular formulation used. The $K_{hs,1}$ and $K_{hs,3}$ formulations converge much faster than $\tilde{K}_{hs}$ formulation. Comparison between Tables 4.1 and 4.2 and 4.3 shows that the B-spline method is quite flexible to accommodate different choices of parameters. However, it is clear that Constraint 1 discussed in Chapter 3 is the better choice to use to obtain converged eigenvalues and has been used in Tables 4.2 and 4.3.

The first and second eigenfunctions obtained from the $\tilde{K}_{hs}$, $K_{hs,1}$ and $K_{hs,3}$ formulations are compared with the exact ones in Figs. 4.1, 4.2 and 4.3. In the $\tilde{K}_{hs}$ and $K_{hs,3}$ formulations the eigenfunctions oscillate at small distance and have exponentially decaying tails at large distances.

As shown by Shizgal *et al.* [89] there are very steep oscillations that occur close to $x = 0$, evident in Figs. 4.1 and 4.3 and are difficult to represent in the $\tilde{K}_{hs}$ formulation but in the $K_{hs,3}$ formulation they are represented very well. As can be seen in Fig. 4.1, the eigenfunctions in the $\tilde{K}_{hs}$ formulation are good for small values of γ and become increasingly worse as γ increases, but in the $K_{hs,3}$ formulation the eigenfunctions are represented well for the entire grid for all mass ratios. The curves for the eigenfunctions were generated using $S = 20$ and $n = 200$, so with these rather generous parameters, convergence is poor in $\tilde{K}_{hs}$ formulation but not for $K_{hs,3}$. In comparison, for the eigenfunctions obtained using the $K_{hs,1}$ formulation in Fig. 4.2, the exponential decay in the tails evident in the $\tilde{K}_{hs}$ formulation is gradually decreased. All eigenfunctions have tails at large x that are smooth and slowly varying. In general, the curves are simple in shape and the eigenfunction obtained using the $K_{hs,1}$ formulation reproduce the exact values very well.

4.2 Distribution Functions

Figures 4.4 and 4.5 show the time dependence of the distribution functions for all four formulations (f_{hs}, $f_{hs,1}$, $f_{hs,2}$ and $f_{hs,3}$) starting with initial function given in Eq. (3.30),

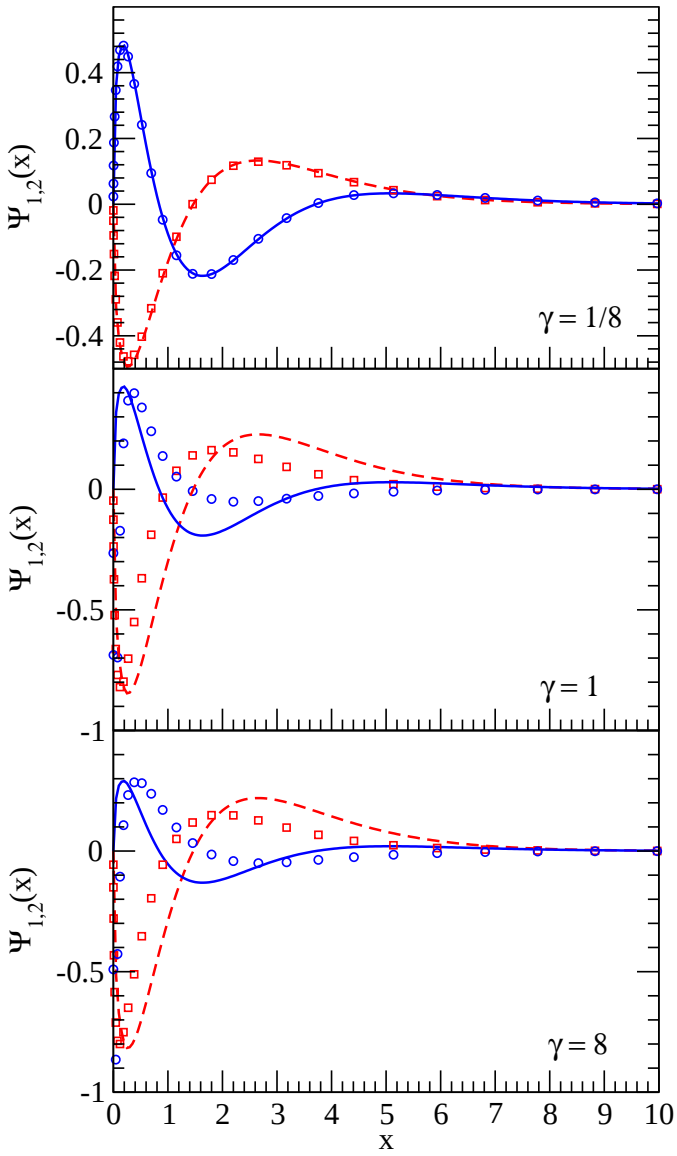

Figure 4.1: Plots showing the accuracy of the eigenfunctions associated with the two lowest, non-zero eigenvalues for different mass ratios. The red squares and blue circles are values obtained from a highly accurate QDM method. The dashed red and solid blue curves represent the first and second eigenfunctions obtained by diagonalizing the matrix representation of the kernel $\tilde{K}_{hs}$, using a grid spanning from 0 to 20 with 200 points.

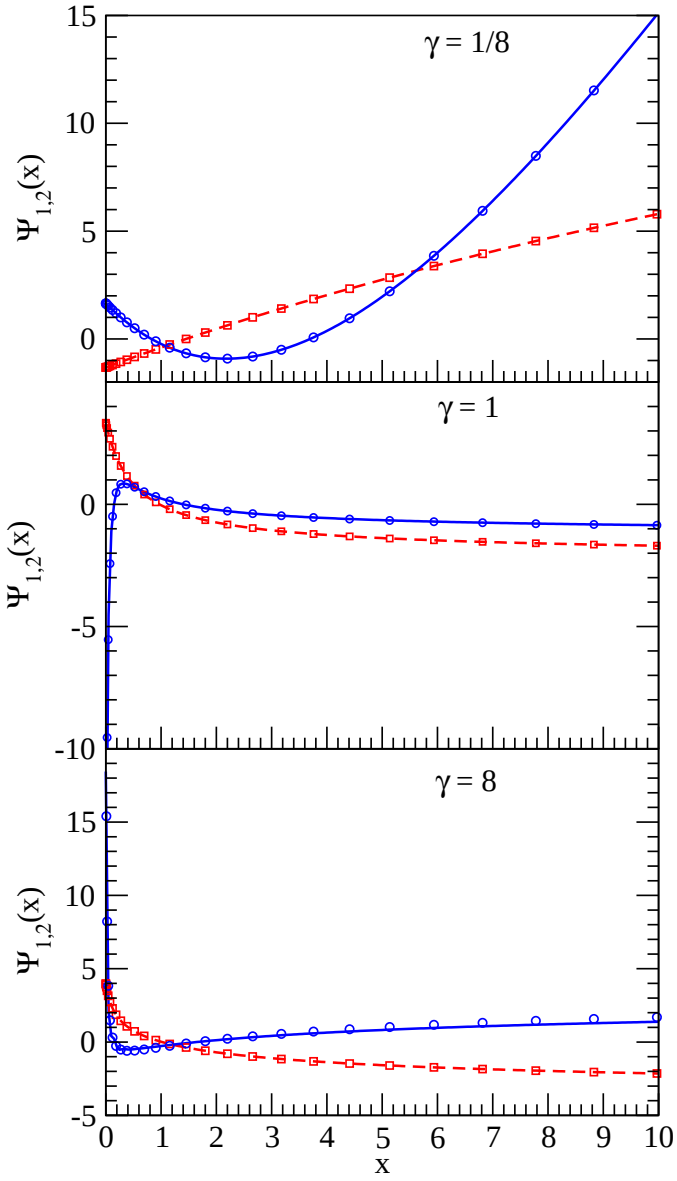

Figure 4.2: Same as Fig. 4.1 except the curves were obtained by diagonalizing the kernel $K_{hs,1}$.

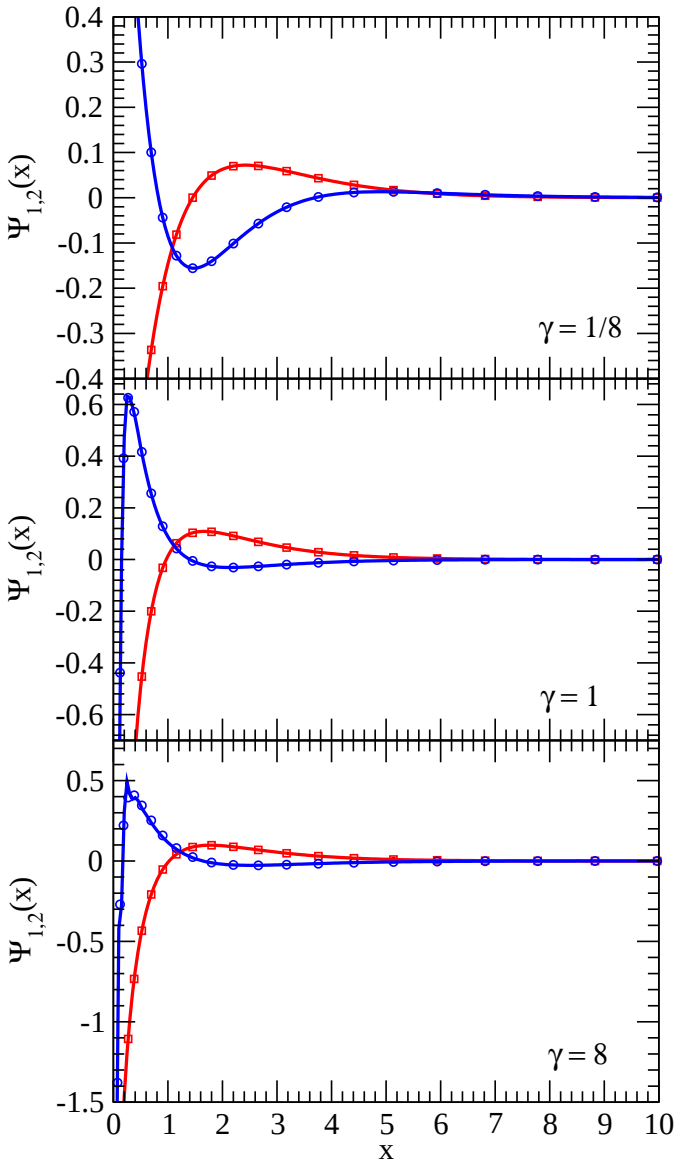

Figure 4.3: Same as Fig. 4.1 except the curves were obtained by diagonalizing the kernel $K_{hs,3}$.

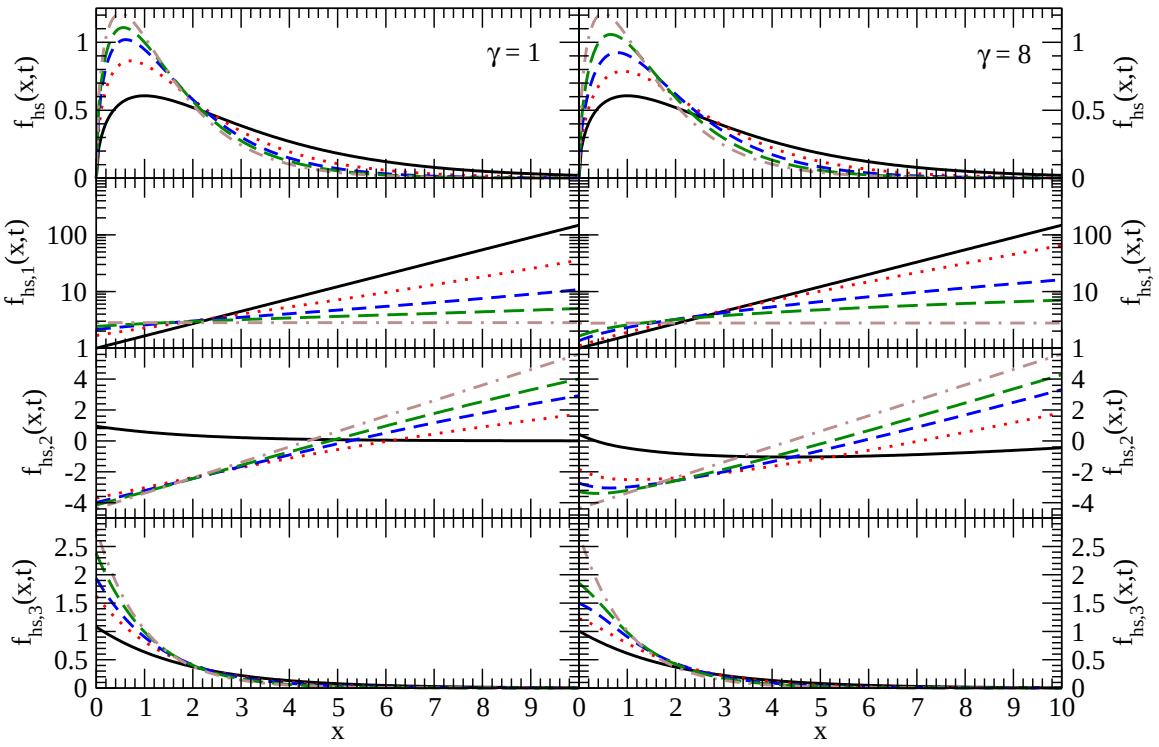

Figure 4.4: Snapshots of the time dependence of distribution functions started with the initial function of Eq. (3.30) with $\alpha = 1/2$ and $\gamma = 1$ and 8. The progression of the curves begins from the black solid curves to the final equilibrium state given by the brown dot-dashed curves. The top, middle and bottom panels give the distribution functions in the $\tilde{K}_{hs}$, $K_{hs,1}$, and $K_{hs,2}$ and $K_{hs,3}$ representations, respectively. The y-axis of the second panel is logarithmic. The black curves are given for $t = 0$ and the brown curves represent the last time step in the calculations. A greater number of curves are obtained, but since they overlap each other, only a few are shown to indicate the progression of the distribution function. The time steps varied from $10 - 100$ for the the different formulations shown.

with temperature ratios of 1/2 and 2 and the two mass ratios $\gamma = 1$ and 8.

Figure 4.6 shows the same time dependent distribution functions in the left panel as given in Fig. 4.4 for $\gamma = 1$ and $\alpha = 1/2$. In the right panel, the full distribution function as given by Eqs. (3.2)−(3.4) are given. The curves in the right panel should all be identical to each other. This figure shows the difference of expanding the distribution in the different formulations, does not effect the full distribution function. Therefore, each of the formulations are numerically different from each other, but analytically identical. Each of the curves shown in this figure are snapshots taken at identical times in each case.

Figure 4.7 shows the time dependent distribution function initialized using the bimodal distribution function in Eq. (3.31) for all four formulations and $\gamma = 1$. With this initial distribution the tracer has two regions of energy and must relax to equilibrium. Depending

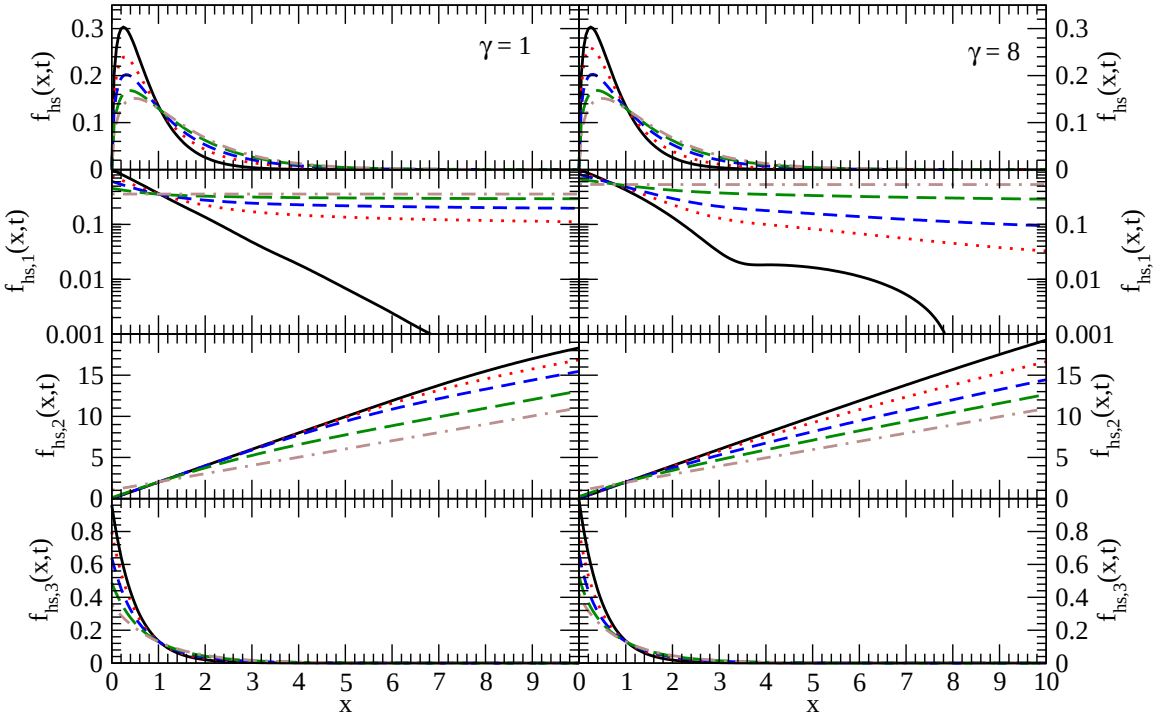

Figure 4.5: Same as Fig. 4.4, except with $\alpha = 2$ and $\gamma = 1$ and 8.

on the formulation used, there is variation in the general shapes of the distribution functions.

The top panels of all the figures show the $\tilde{K}_{hs}$ formulation. In this representation the distribution functions have exponentially decaying tails and the initial functions decay to a Maxwellian distribution function. The graphs show the time evolution of the distribution smoothly moving towards equilibrium. The second panels show the distribution functions obtained using the $K_{hs,1}$ formulation, where the initial functions are increasing for a lower temperature ratio and decreasing for $\alpha = 2$. In both cases these distributions must relax to a value of 1 at equilibrium. The third panels show the distribution functions in the $K_{hs,2}$ formulations. In this representation, the B-splines expand a function in the exponent and therefore the initial distribution function starts with a slope different from 1 and as the distribution approaches equilibrium, the slope becomes exactly 1, the correct limiting value. The bottom panel in all the figures shows the distribution function in the $K_{hs,3}$ formulation. In this formulation the initial distribution function are exponential functions and at equilibrium the distribution is an exponential.

In Fig. 4.7 the top panel very clearly shows the two peaks, where the first peak must grow and the second peak must shrink to relax to a Maxwellian distribution at equilibrium. In the second panel for the $K_{hs,1}$ formulation, the initial distribution starts off with a non-uniform curve and steadily approaches a value of 1 at equilibrium. Similarly, in the $K_{hs,2}$

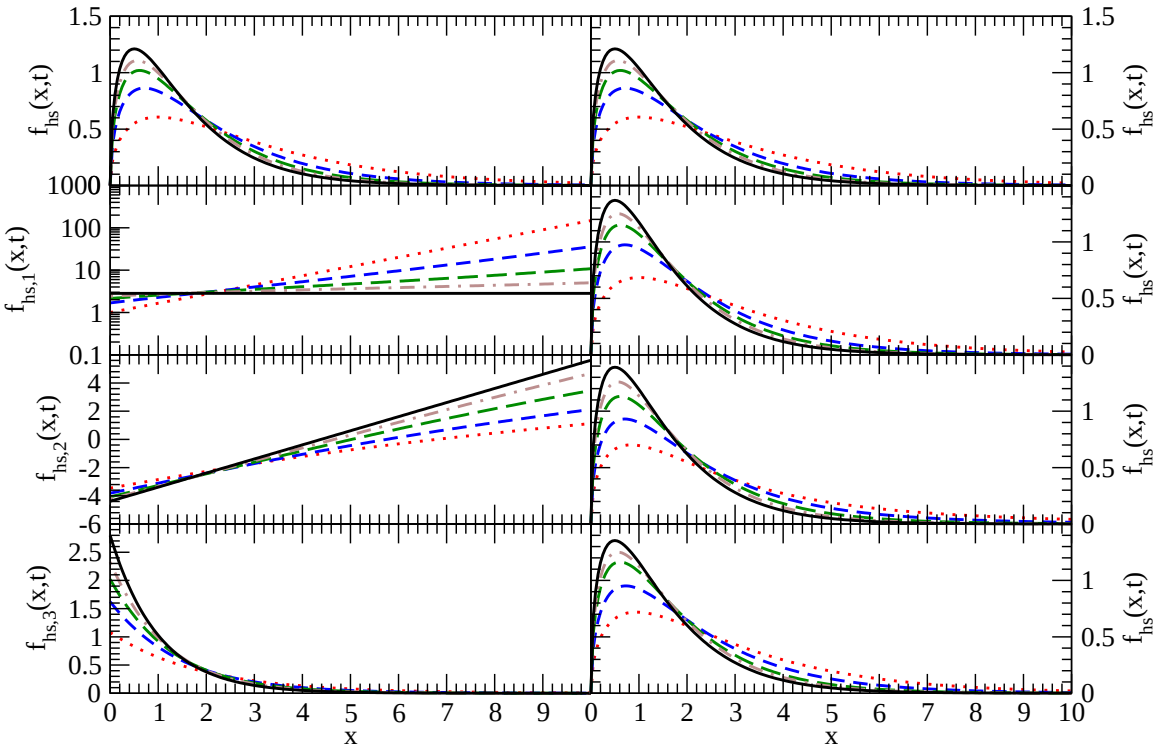

Figure 4.6: Distribution functions for $\alpha = 1/2$ and $\gamma = 1$. The left panels are the same as those in Fig. 4.4, that is, they show the progression of the distribution function to equilibrium of the different formulations, K_{hs}, $K_{hs,1}$, $K_{hs,2}$ and $K_{hs,3}$. The right side panels show the complete distribution function as given by Eqs. (3.2)−(3.4). The red dotted curve is the initial distribution function that progress to the equilibrium function given by the black solid curve. The curves shown are plotted at identical relaxation times for all the cases.

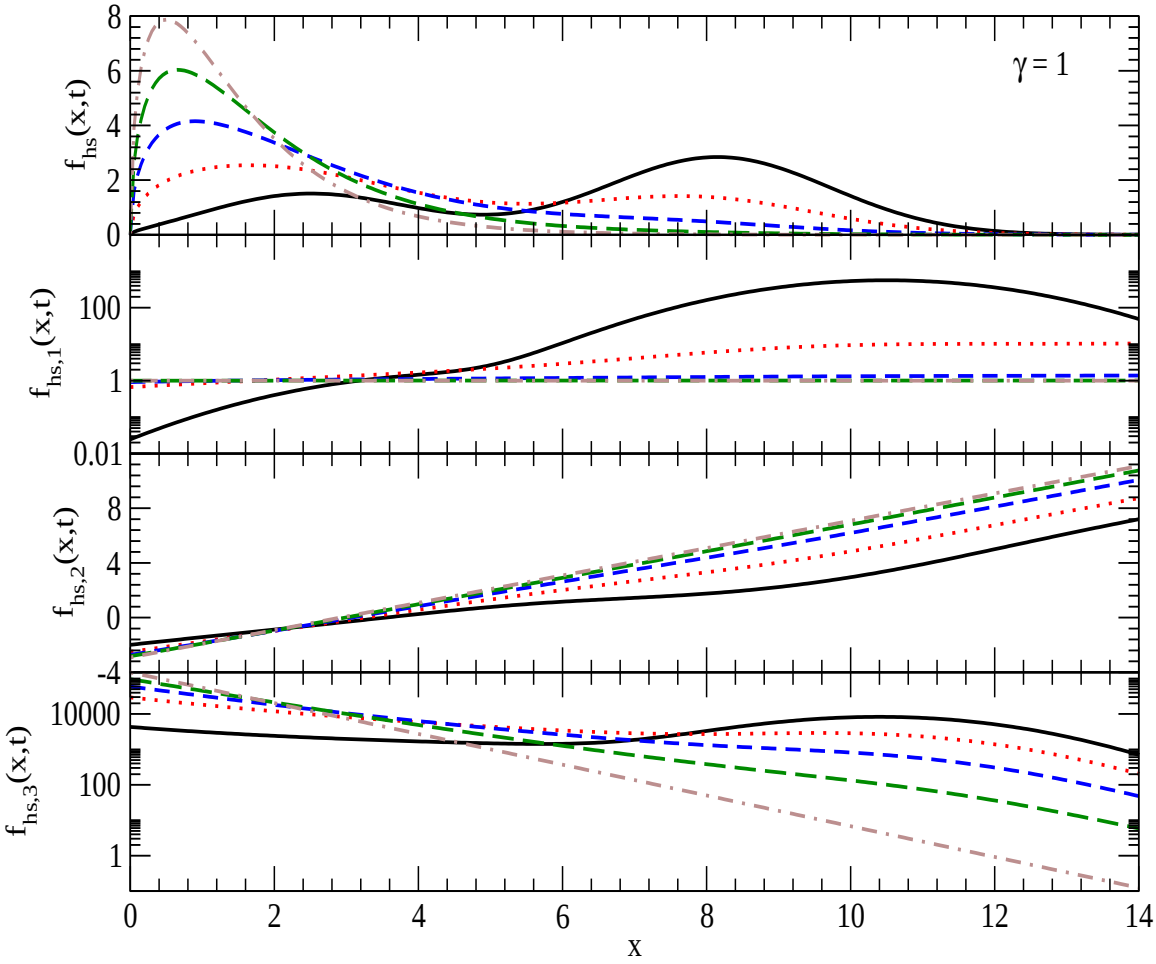

Figure 4.7: Same as Fig. 4.4, except using an initial function given by Eq. (3.31) with $\gamma = 1$.

formulation shown in the third panel, the curve is non-uniform initially and then approaches a slope of exactly 1 at equilibrium. In the last panel, the initial function has a similar shape to the black curve in the top panel. As the distribution approaches equilibrium the curve has an exponential shape. Overall the time dependence in all formulations is well behaved. The curves are smooth over the entire grid of spatial values and remain at equilibrium at long times. The distribution function at equilibrium for $K_{hs,1}$ and $K_{hs,2}$ is a linear function which is represented by the B-splines exactly. In comparison, the other formulations have more difficult forms, and even in these cases the B-splines are easily able to represent these functions. Therefore, depending on the case each of the formulation can have its advantages and disadvantages.

4.3 Moments

For the calculation of the moments a standard calculation was done using Eq. (3.30) with $\alpha = 0.5$ and 2, and the bimodal distribution using Eq. (3.31), for $\gamma = 1/8$, 1, and 8, where the parameters for the total time, and grid size S, were determined. Multiple calculations were done to determine the minimum number of the points, n, required for the moment values to be varied by more than 0.1% from the standard results. Therefore, the value of n was reduced until the value of the first moment was within 0.1% of the equilibrium value of $3/2kT$.

The values for the zeroth, first and second moments were calculated for distribution functions of different mass ratios and initial conditions, but graphical representations of the first and second moments are shown in Figs. 4.8 and 4.9. T^* can be taken to be the ratio between the tracer and bath temperature, and at equilibrium $T^* = 1$. A normalization factor of $T^* = (2/3)\langle y \rangle$ is used to normalize the first moment as shown in Figures 4.8 and 4.9, where $\langle y \rangle$ is the average value of the moment at a particular time. The second moment is calculated using $\xi = \langle y^2 \rangle - (5/3)\langle y \rangle^2$. At equilibrium, for a Maxwellian distribution function, $\xi = 0$, whereas at other times, ξ can be taken to be the measure of deviation of the distribution function from equilibrium. It should be noted though, that in general a value of $\xi = 0$ does not guarantee a Maxwellian distribution. This relationship between the values of T and ξ is also shown in Figures 4.8 and 4.9. The moments are shown to converge quite well as the initial functions approach equilibrium. This is indicated by the curves in Fig. 4.8 where the deviation of the normalized moment values from Maxwellian decreases at longer times.

As can be seen in Figure 4.8, the values of T^* start either at 2 or 0.5, approach 1 smoothly, and remain at that value for long times. As expected, the decay of the moments for a higher mass ratio is slower than that for a smaller mass ratio. The values of ξ show small deviations from Maxwellian at small times, but then quickly approach the equilibrium

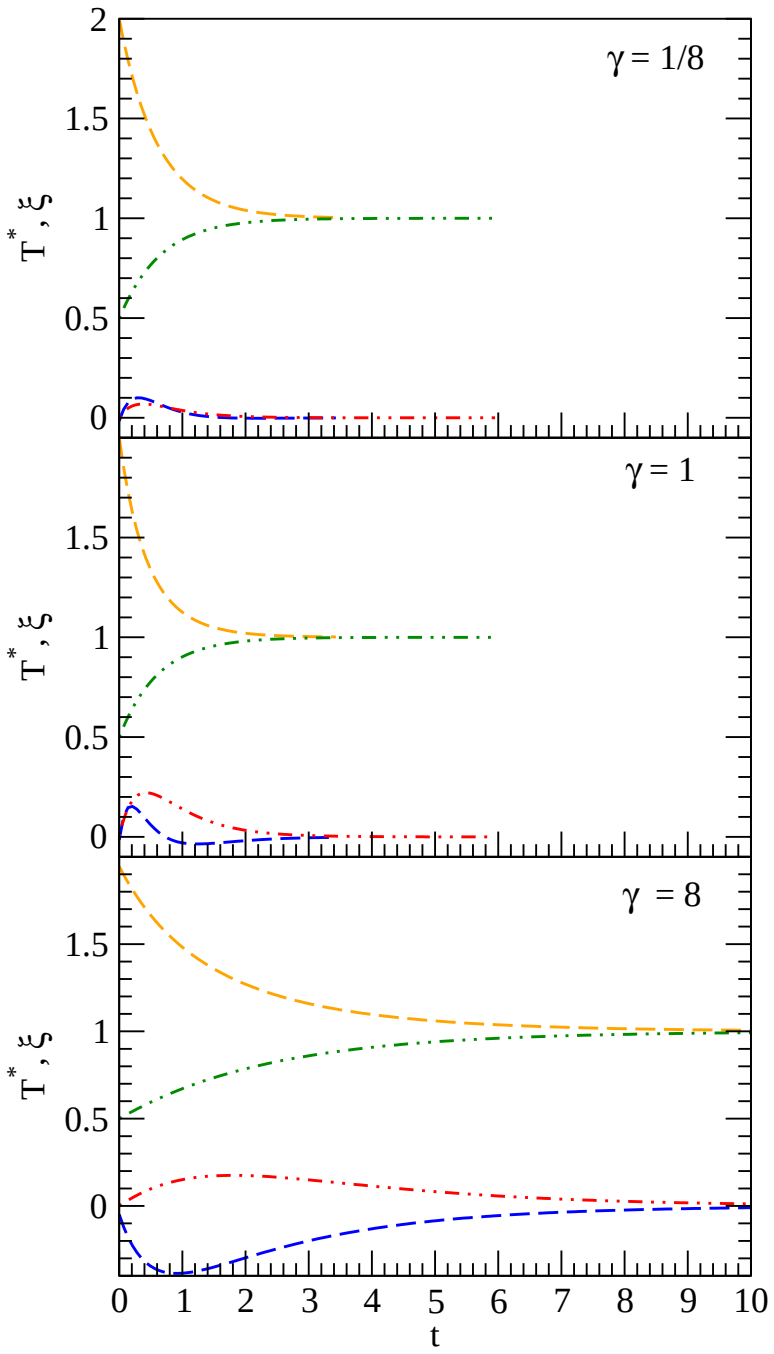

Figure 4.8: Plots of T^* and ξ for the distribution functions for different mass ratios using the initial function of Eq. (3.30). In all three panels, the blue and yellow dashed curves use initial functions with $\alpha = 1/2$, and red and green dot-dashed curves are for $\alpha = 2$. The yellow dashed and green dot-dashed curves plots values of T^*. The blue dashed and red dot-dashed curves plot values of ξ.

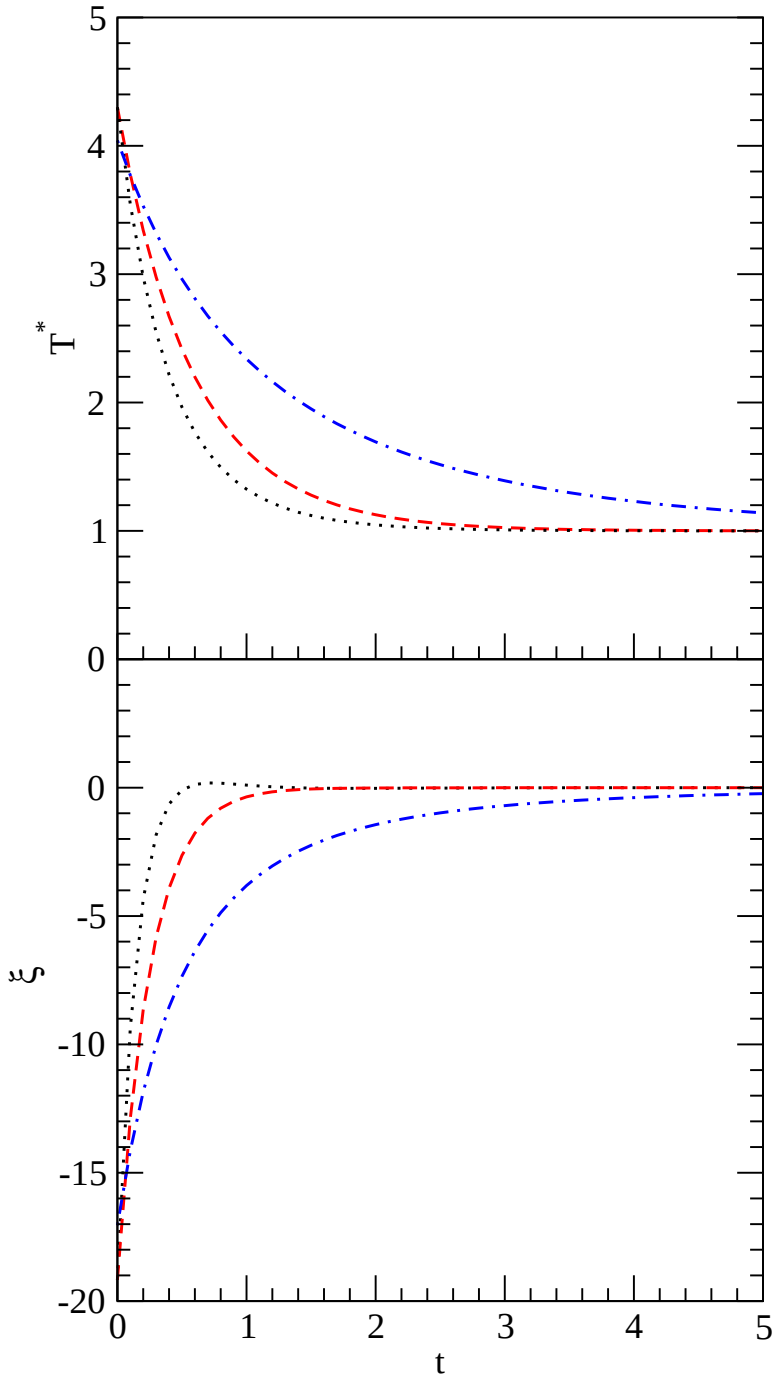

Figure 4.9: Plots of T^* and ξ for the distribution functions for different mass ratios using the initial function of Eq. (3.31). In both panels, the red dashed, black dotted, and blue dot-dashed curves are for mass ratios $\gamma = 1/8$, 1 and 8, respectively.

value of zero. In Figure 4.9, the initial function is far from equilibrium, and therefore a more pronounced change is visible. The ξ values are significantly less than zero but increase quickly to zero, with the low mass ratio increasing faster than the higher mass ratio. The values of T^* once again show the decay of an initially hot distribution to the equilibrium value.

Although not shown here, it was found that moments calculated with the $f_{hs,1}$, $f_{hs,2}$ and $f_{hs,3}$ formulations, all converged using $n = 10 - 100$. For the f_{hs} formulation, n was much larger and in fact it was quite challenging to converge the moments to within 0.1% accuracy. Even though the $f_{hs,1}$ and $f_{hs,2}$ formulations converged quite well, the $f_{hs,2}$ formulation required many time steps to propagate to the equilibrium state along with a very small time step. The time propagator used for the $f_{hs,2}$ formulation is quite simple and better performance can be obtained by using a more efficient time propagator, such as the Runge-Kutta. Overall, in terms of time steps and convergence, the $f_{hs,1}$ formulation works the best. Also, generally for all the formulations the moments were more easily converged rather than eigenfunctions and eigenvalues.

4.4 Discussion

In this chapter results from solving the linear Boltzmann equation of Eq. (2.7) using B-splines are shown for four different formulations of the kernel. These were used to test accuracies in predicting moments, eigenvalues and eigenfunctions. The results indeed validate the use of B-splines as a basis set. Using a variety of forms of the distribution functions for a number of different combinations of mass and temperature ratios, proves that this numerical method is quite flexible and can represent any form of the distribution function. The flexibility of this method also allows the size of the grid, the order of the B-splines, the number of collocation points and boundary conditions, to be changed independently according to the needs of the problem. The points at which the integrals are evaluated are equally spaced in the results shown, but the use of B-splines does not require this condition to be satisfied either. The collision kernel is known to have an analytical cusp at $x = x'$. The use of B-splines did not cause any problem in dealing with this cusp or any other irregularities that may have been present. This is due to the fact that the evaluation of the integrals is done over very small domains and the kernel can be represented very well at or near the cusp. Since this irregular behaviour is localized, very few integrations are affected by it, and special techniques can be used in these cases to deal with any such problems. The numerical results obtained using the different formulations are nearly identical to each other, but each formulation has its advantages and disadvantages.

Even though different formulations are used, the convergence obtained from each of them is not identical. Therefore, depending on the problem at hand, an appropriate kernel

formulation can be used that is the most advantageous. In the original expression of the collision kernel for $\tilde{K}_{hs}$, there are exponential and $\sqrt{x}$ terms and these formulations help with bypassing these terms. Results obtained using the original expression of the collision kernel required the maximum number of collocation points among all the formulations presented. The $\tilde{K}_{hs}$ formulation does not produce satisfactory results. Eigenvalues and eigenfunctions show poor convergence and the B-splines do not represent the distribution function well. Contributions from a majority of grid points is minimal since they lie in the tail of the distribution function. Therefore, only a small number of points are actually significant in constructing an accurate representation of the distribution function. Thus, large values of n are required for the system to relax to equilibrium.

To better balance the contribution of all grid points to the distribution function, the K_1 formulation was constructed, where the exponential term was factored out and thus at equilibrium the distribution function is precisely 1, which is exactly represented by the polynomial nature of the B-splines using only two points. With this formulation, the eigenvalues and eigenfunctions produced are very accurate and converge very rapidly to the exact published values with very few collocation points. Since this formulation removes the decaying tails in the equilibrium function, all grid points contribute significantly and evenly in the construction of the distribution function. The results are more accurate with smaller n as compared to the $\tilde{K}_{hs}$ formulation. The formulation of K_1 can work very well, but its limitations lie in the fact that the system can only be slightly perturbed from equilibrium.

As will be shown in Chapter 5, when tracers are subject to high initial energies, representing the initial distribution functions with the K_1 formulation can be a problem because as implied by Eq. (3.1) $f_1(x, t = 0) = \frac{1}{2}\sqrt{\frac{\pi}{x}}e^x f(x, t = 0)$. The exponential factor approaches very large values which can be difficult to store in standard variables if $f(x, t = 0)$ is peaked at large x. Propagating such large values is also difficult. To counter this problem, the K_3 formulation was designed to remove the $\sqrt{x}$ factor in $\tilde{K}_{hs}$ that causes poor convergence but leaves the remaining dependencies in f untouched. As in the f_{hs} case, the distribution function relaxes to an exponential at equilibrium and the number of points required to attain equilibrium is comparable to the f_1 case. The convergence is slightly poorer than the f_1 case, but distribution functions with high initial energies can be represented well with this formulation. The eigenvalues and eigenfunctions obtained using K_3 indicate that this representation can produce satisfactory results.

In general, the original expression of the collision kernel $\tilde{K}_{hs}$ should be avoided. The K_3 formulation should be used and is the most similar to $\tilde{K}_{hs}$ and produces fairly accurate results. In both these cases the contribution from the collocation points is uneven and lie mostly in the decaying tails as seen in the eigenfunction plots. To ensure an even contribution from all the collocation points, the K_1 formulation works best but fails in the case where the tracers have high initial energies. In terms of eigenvalues and eigenvectors,

K_1 is the best formulation, since it produces the most accurate results with the least number of collocation points.

In yet another formulation, K_2, the distribution function as the exponential of a B-spline expansion is physically correct. In this formulation of K_2, the B-splines represent an argument of an exponential function. This was done because it guarantees a positive distribution function which is physically correct regardless of the values of f_2. In contrast, all other representations may give physically incorrect negative values for the distribution functions, if the spline is poorly converged and develops oscillations. Since the B-splines are in the exponential, the time-dependent coefficients cannot be isolated during the integration, and therefore no eigenfunctions or eigenvalues can be obtained. Thus, this formulation is removed from the constraint of having to produce these values. Therefore, one can see that obtaining eigenvalues and eigenfunctions is a matter of choice rather than a necessity for this numerical method to work. The moments and distribution functions converge well in this formulation. Once again the number of grid points required to obtain equilibrium are minimal. The only problem encountered with this formulation is that a very small time step is required with the simple propagator given in Eq. (3.24). One can use more sophisticated time propagators to increase this time step.

The construction of the numerical method does not rely upon the particular forms of the kernel, therefore any of the kernel formulations, i.e. $\tilde{K}_{hs}$, K_1, K_2 or K_3, can be used. These formulations were constructed to bypass the numerical difficulties that can affect convergence of the results. The integrations of the collision kernel is divided into smaller intervals that are independent of other integrals on the grid. To make the numerical method more efficient, a parallel algorithm can be set up to perform these integral calculations simultaneously. The results presented are for a one-dimensional linear Boltzmann equation but this numerical method can easily be extended to higher dimensions by using B-splines to represent each dimension. Only a few collocation points are used in the different formulations to obtain results in one dimension, therefore increasing the number of collocation points for multiple dimensions should be reasonable. In terms of the computational time, the K_1 formulation was the fastest, and $\tilde{K}_{hs}$ formulation took the longest to finish. Even for this formulation, the calculations completed in a few hours, therefore evaluations in multiple dimensions should only require a reasonable amount of computing time.

Table 4.2: The magnitudes of the first three non-zero eigenvalues obtained for mass ratios, $\gamma = 1/8$ and $1/2$, by diagonalizing $\mathbf{L}$ using the matrix representation of the $\tilde{K}_{hs}$, $K_{hs,1}$ and $K_{hs,3}$ kernels. The eigenvalues are normalized by $Z_{hs}(0)$. The grid used for each diagonalization spans to $S = 20$ with n intervals and spacing Δ between grid points. The accurate values are QDM results from Shizgal *et al.* [89]. The blank cells indicate that convergence was obtained for the particular n and Δ combination. The cells with a dash indicate that no eigenvalues under 1.0000 were obtained.

	n	Δ	$\gamma = \frac{1}{8}$			$\gamma = \frac{1}{2}$		
			$K_{hs,1}$	$\tilde{K}_{hs}$	$K_{hs,3}$	$K_{hs,1}$	$\tilde{K}_{hs}$	$K_{hs,3}$
$\lambda_1/Z_{hs}(0)$	2	10	0.2783	0.6277	–	0.6743	1.0000	0.2025
	5	4	0.2784	0.2157	0.2742	0.6813	0.6524	0.7523
	20	1	0.2784	0.3097	0.2779	0.6889	0.7283	0.6875
	60	0.33		0.2861	0.2785	0.6891	0.6972	0.6892
	200	0.10		0.2799	0.2784	0.6891	0.6906	0.6891
	600	0.03		0.2787	0.2784		0.6894	0.6891
Accurate values				0.2784			0.6891	
$\lambda_2/Z_{hs}(0)$	2	10	0.4894	0.6277	–	–	–	–
	5	4	0.4878	0.6248	–	0.9319	1.0000	0.7523
	20	1	0.4882	0.5201	0.4855	0.9119	0.9757	0.9150
	60	0.33	0.4882	0.4963	0.4882	0.9203	0.9349	0.9200
	200	0.10		0.4898	0.4882	0.9208	0.9234	0.9208
	600	0.03		0.4885		0.9208	0.9214	0.9208
Accurate values				0.4882			0.9208	
$\lambda_3/Z_{hs}(0)$	2	10	0.6707	1.0000	0.6291	–	–	–
	5	4	0.6481	0.6248	–	–	–	–
	20	1	0.6462	0.6777	0.6415	0.9996	1.0000	0.9802
	60	0.33	0.6466	0.6548	0.6465	0.9795	1.0000	0.9798
	200	0.10	0.6466	0.6482	0.6466	0.9827	0.9884	0.9827
	600	0.03		0.6469	0.6466	0.9832	0.9843	0.9832
Accurate values				0.6465			0.9832	

Table 4.3: Same as Table 4.2 except for $\gamma = 1$ and 8.

	n	Δ	$\gamma = 1$			$\gamma = 8$		
			$K_{hs,1}$	$\tilde{K}_{hs}$	$K_{hs,3}$	$K_{hs,1}$	$\tilde{K}_{hs}$	$K_{hs,3}$
$\lambda_1/Z_{hs}(0)$	2	10	0.8842	1.0000	0.1992	0.5023	1.0000	–
	5	4	0.7801	1.0000	–	0.4757	0.4726	0.4474
	20	1	0.8136	0.8848	0.8101	0.5639	0.7956	0.5544
	60	0.33	0.8190	0.8325	0.8189	0.6097	0.6584	0.6085
	200	0.10	0.8190	0.8213	0.8190	0.6139	0.6213	0.6139
	600	0.03		0.8195	0.8190	0.6139	0.6153	0.6139
Accurate values			0.8190			0.6139		
$\lambda_2/Z_{hs}(0)$	2	10	–	–	–	–	–	–
	5	4	–	0.8512	0.9298	–	1.0000	–
	20	1	0.9777	1.0000	0.9841	0.9561	1.0000	–
	60	0.33	0.9727	1.0000	0.9732	0.9292	1.0000	0.9330
	200	0.10	0.9789	0.9874	0.9789	0.9526	1.0000	0.9523
	600	0.03	0.9797	0.9812	0.9797	0.9650	0.9770	0.9650
Accurate values			0.9797			0.9667		
$\lambda_3/Z_{hs}(0)$	2	10	–	–	–	–	–	–
	5	4	–	–	–	–	–	–
	20	1	–	–	–	–	–	–
	60	0.33	–	–	–	–	–	–
	200	0.10	0.9994	–	0.9996	–	–	–
	600	0.03	0.9977	1.0000	0.9977	1.0046	1.0000	1.0047
Accurate values			0.9984			0.9997		

Chapter 5

Kinetic Energy Relaxation of Heavy Tracers

In this chapter the numerical method constructed using B-splines is further scrutinized by subjecting the heavy tracers to high initial energies and monitoring the relaxation to equilibrium. The smooth and rough hard sphere and Maxwell molecule models are used to test the experimental results of Douglas *et al.* [12, 13] in mass spectrometry, where gas-phase biomolecular ions, including motilin, ubiquitin, cyctochrome *c*, myoglobin and bovine serum albumin, are generated using pneumatically assisted electrospray (ion spray). Trace amounts of these biomolecules are injected into a neutral bath at a high initial energy of $10i$ eV, where i is the number of charges on the ion, and this energy when converted to the units used in our studies is $x_0 \simeq 386$. As they travel the length of the cell, the ions relax towards equilibrium due to collisions. By measuring the kinetic energy loss of the ions exiting the cell and modelling the drag coefficient, cross sections of the order $10^3 - 10^4 \mathring{A}$ can be calculated. One of the goals of this study is to compare their type of analysis with results obtained by solving the Boltzmann equation directly.

Andersen *et al.* [58] cast the relaxation of hard sphere Rayleigh and Lorentz gases in the form of a Fokker-Planck equation. In the Rayleigh gas, a trace number of heavy particles are dispersed in a bath of lighter particles. They predicted a Rayleigh gas should relax at a constant rate regardless of the initial energy of the tracer particles, and gave an expression for the rate. In the present study, the range for the validity for the Fokker-Planck results is tested with different mass ratios and initial energies.

Drag coefficients are often modelled using specular and diffuse scattering processes, which in the present case be modelled by smooth and rough hard spheres, respectively. These particular models represent two scattering limits in kinetic theory. In addition, the Maxwell model [77, 78] is also used, to represent a more realistic scattering potential allowing the size of the tracer to vary in the collision. By comparing the results from these three models the effect of elastic and inelastic collisions can be gauged, as well as effect of soft walls in the interaction between the tracer and bath.

The initial function used to obtain these results is given by Eq. (3.32). To test the limits of the numerical method a range of energies were tested, with $x_0 = 5$ and 350. These

two initial energies test both lower and upper energy regimes. The mass of the tracers used in the experimental studies were high, around 16000 Da. To mimic such particles tracers with masses 50, 100, 200 and 400 times greater than the bath are used, giving mass ratios $\gamma = 0.02$, 0.01, 0.005 and 0.0025, respectively. The K_3 collision kernel formulation is primarily used to obtain these results. Shown in this chapter are the distribution functions for the three models for the two different energies. Also, the Fokker-Planck expression for the Rayleigh gas derived by Andersen and Schuler [58] is also tested to determine its limits of applicability. Further, the drag coefficients for the three different models are calculated and compared to published theoretical and experimental results.

5.1 Eigenvalues of Smooth and Approximate Rough Hard Sphere

The eigenvalues of the kernels for the smooth hard sphere and approximate rough hard sphere in Eqs. (2.40) and (2.85), respectively are given in Table 5.1 for $\gamma = 1$ and 0.1. When scaled by $Z_{hs}(0) = 2A/\sqrt{\gamma}$, the kernel $\tilde{K}_{hs}(x, x')$ has a discrete spectrum below

Table 5.1: Eigenvalues (absolute values) of the smooth and approximate rough hard sphere kernels for two different mass ratios and a range of $\mu\chi$ values. Eigenvalues are scaled by $2A/\sqrt{\gamma}$ and only up to six non-zero scaled values less than unity are tabulated. An entry of "..." indicates that no eigenvalue was found. The values for $\mu\chi = \infty$ are for the smooth hard sphere kernel.

$\mu\chi$	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
$\gamma = 1$						
1.5	...	...	...	...	...	...
5	...	...	...	...	...	...
10	0.9488	...	...	...	...	...
25	0.8787	...	...	...	...	...
50	0.8503	...	...	...	...	...
∞	0.8190	0.9797	0.9977	...	...	...
$\gamma = 0.1$						
1.5	0.3287	0.5801	0.7692	0.9081	...	...
5	02724	0.4871	0.6566	0.7895	0.8916	0.9651
10	0.2536	0.4548	0.6148	0.7419	0.8417	0.9171
25	0.2404	0.4316	0.5842	0.7060	0.8024	0.8766
50	0.2356	0.4231	0.5729	0.6924	0.7872	0.8604
∞	0.2307	0.4142	0.5608	0.6779	0.7708	0.8426

unity and a continuous one above [64]. The first few discrete values for the smooth sphere are listed in Table 5.1 in the rows labelled with $\mu\chi = \infty$. The spectrum of $\tilde{K}_1(x, x')$ is not

known but for the sake of comparison, its eigenvalues are scaled in the same way and only those values less than unity are tabulated. Examining values in Table 5.1 shows that for finite $\mu\chi$ the number of eigenvalues less than unity for the approximate rough hard sphere kernel is less than that for the smooth sphere one, and the corresponding eigenvalues are larger. This implies relaxation in the rough system should be faster than in the smooth one; an expected result given the possibility for translational-rotational energy exchange in the former. The values for the rough case do approach those for the smooth one as $\mu\chi$ increases, although this convergence is much faster for $\gamma = 0.1$ than for $\gamma = 1$.

5.2 Distribution Functions

In Figs. 5.1 and 5.2 representative time evolutions of the distribution function are shown for $\gamma = 0.02$ and 0.0025, respectively, with initial energies $x_0 = 5$ and $x_0 = 350$ for the smooth hard sphere, rough hard sphere and Maxwell models. In the left panels, the solid black curves show the distribution function after the first few time steps and are close to Gaussian shapes. In a very short time (dotted red curves) the peaks lower and the widths expand. As the distributions move to equilibrium the peaks continue to lower and the widths become greater, until at equilibrium, (violet dot-dashed curves) the functions adopt an exponential form. This trend is identical for all three models tested, except for the Maxwell model, the distribution function is slightly broader at the first few time steps. In the right hand panels, the solid blacks curves represent the distribution function after a few time steps and show sharp peaks at $x_0 = 350$, with a small bud forming at lower energy. These peaks quickly shift and broaden (dotted red curves). As time increases the widths of the distributions remain approximately constant, but the peaks shift to lower energy until at equilibrium the distributions adopt an exponential decay represented by the maroon dashed curves. This trend is similar for both the smooth and rough hard sphere models while the Maxwell molecule shows a greater relaxation at the initial times. This means an ensemble of the tracers tend to relax as a group with some energy width. This width defines the resolution possible in a kinetic energy loss experiment.

The nature of the distribution functions is consistent with Monte Carlo calculations of Douglas *et al.* [13]. Although the distribution curves shown in this experimental study compares two different masses of the tracer, similar trends can be seen with the representative distribution functions in Figs. 5.1 and 5.2. From Ref. [13], the more massive the tracer, the narrower the distribution, and with a lighter tracer the distribution is more spread out. In both Figures 5.1 and 5.2 we note that with a lower initial energy, the distribution is more spread out and relaxation to equilibrium is gradual. With the higher initial energy, the distribution is narrower and relaxes faster to equilibrium. This behaviour is consistent for a lower mass ratio of $\gamma = 0.0025$ as well.

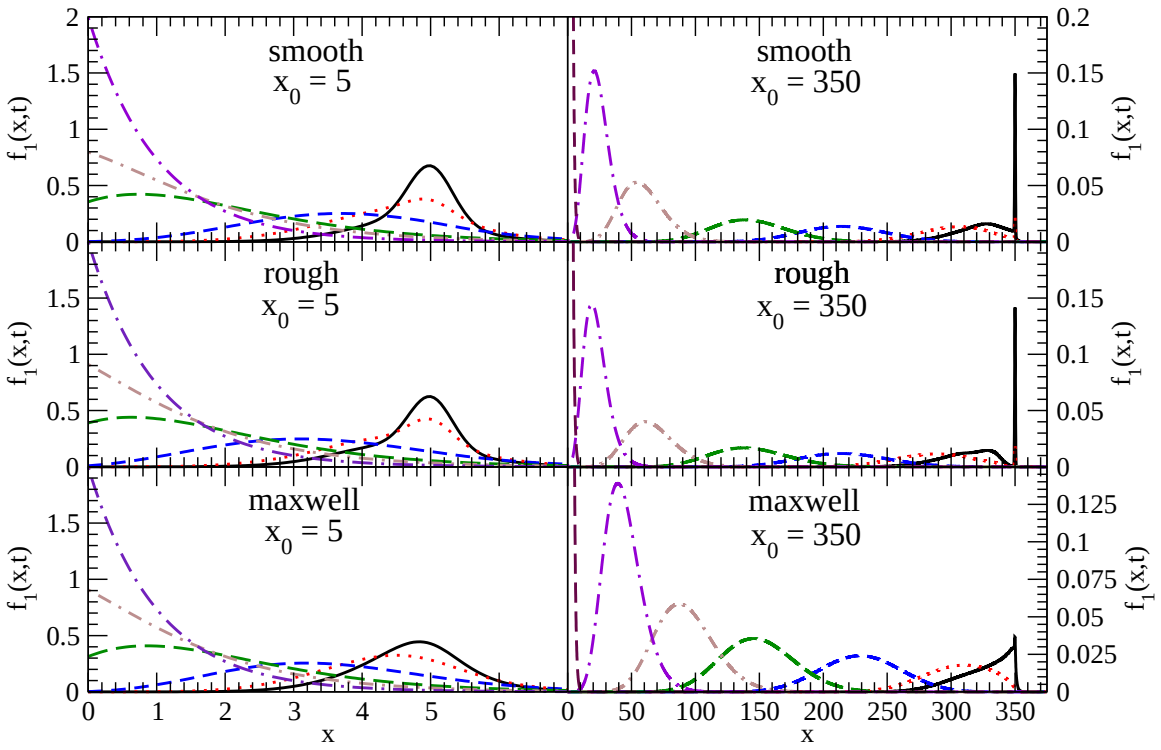

Figure 5.1: The time evolution of the distribution function for $\gamma = 0.02$ with initial energies $x_0 = 5$ (left panels) and $x_0 = 350$ (right panels). The top, middle and bottom panels show the relaxation of the distribution function to equilibrium for the smooth hard sphere, rough hard sphere and Maxwell molecule cases, respectively. The evolution starts from the solid black curves and moves to the dash-dot violet curves (left panels) or dashed maroon curves (right panels).

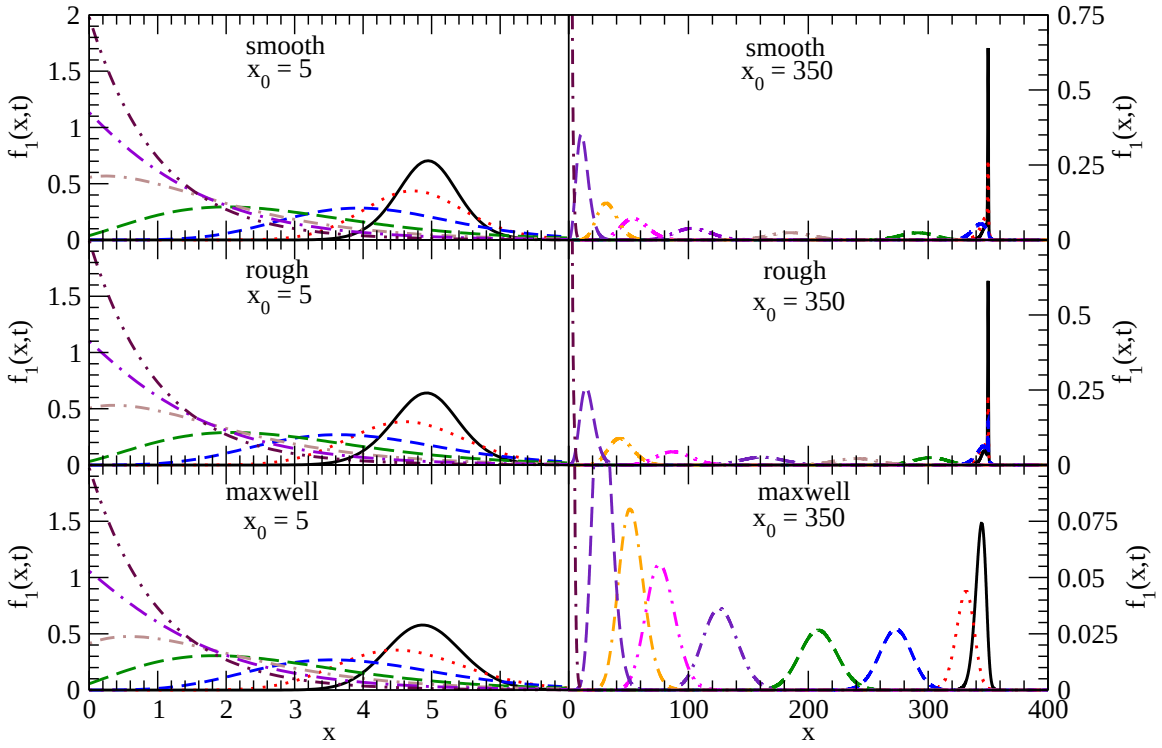

Figure 5.2: Same as Fig. 5.1 but for $\gamma = 0.0025$.

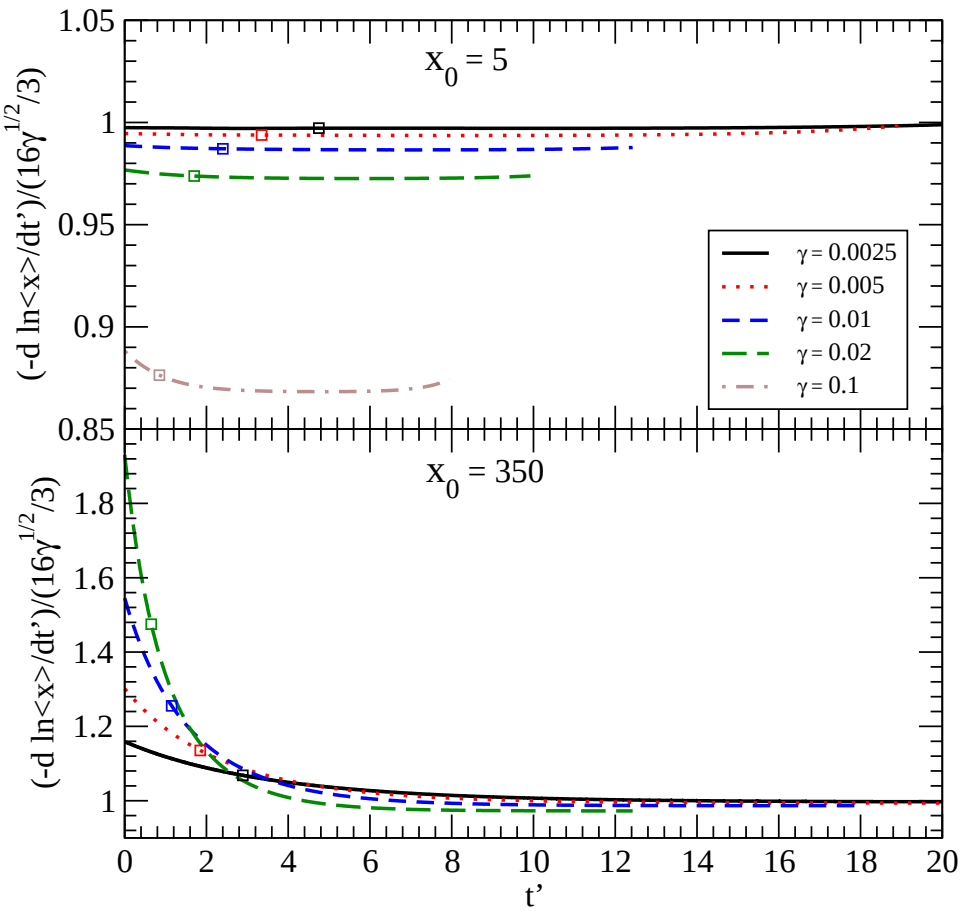

Figure 5.3: Derivatives of the reduced average kinetic energy normalized by $16\sqrt{\gamma}/3$ as a function of t' for $\gamma = 0.02$, 0.01, 0.005 0.0025 and 0.1 and $x_0 = 5$ (top panel) and $x_0 = 350$ (bottom panel), for the smooth hard sphere case. The square symbols indicate the points at which the kinetic energies are 50% of their initial values.

5.3 Kinetic Energy Derivative

Figure 5.3 shows the derivatives of the reduced average kinetic energy motivated by Eq. (2.111) for different mass ratios and energy regimes. The values are normalized by $16\sqrt{\gamma}/3$, the Fokker-Planck results of Eq. (2.111). The results in Fig. 5.3 test the accuracy of Eq. (2.111) and the bound of Eq. (2.112). In the case where Eq. (2.111) is valid and stands correct for a particular mass ratio, the values of derivatives of the first moment will equal exactly $16\sqrt{\gamma}/3$. If the Fokker-Planck equation is valid for the cases tested, the curves shown in Fig. 5.3 will have constant values at 1, and all the curves would overlap each other. The only exception in this case would be the curves in the lower panel, because for $\gamma = 0.02$, 0.01 and 0.005, Eq. (2.112) is not valid.

In the top panel, the relation in Eq. (2.112) is satisfied for all the values of γ shown when $x_0 = 5$. Thus, one expects the Fokker-Planck decay constant to be accurate. However, while the curves are generally constant, they deviate from unity as γ increases, that is, as the tracer becomes lighter. For the smallest mass ratio $\gamma = 0.0025$, shown by the solid back curve the value is very close to unity while for the brown dot-dashed curve representing $\gamma = 0.1$ it drops to approximately 0.89.

In the bottom panel the relation of Eq. (2.112) is satisfied only for $\gamma = 0.0025$ when $x_0 = 350$. Thus, the curves in this panel show the behaviour when Eq. (2.112) is not satisfied. Unlike the curves in the top panel, all those in the lower panel decay as time increases, with a rate which increases as γ increases. In other words, the initial decay of the kinetic energy is much larger than predicted by Eq. (2.111) and does not evolve with a constant decay rate. Eventually though, at long enough times, these rates do become constant and equal to those predicted by Eq. (2.111), as the kinetic energy of the system decreases to a point where Eq. (2.112) becomes valid.

The square symbols on the curves indicate the times at which the kinetic energy has dropped to 50% of its initial value. As seen in the upper panel of Fig. 5.3, using Eq. (2.111) when Eq. (2.112) is satisfied gives a rate which deviates by some percentage from k_R depending on γ. However, as seen in the lower panel, Eq. (2.111) is qualitatively incorrect during most of the relaxation process when Eq. (2.112) is not satisfied. In other words, in the case Eq. (2.111) becomes accurate only when the system is closer to equilibrium for the higher mass ratios.

5.4 Drag Coefficients

The normalized drag coefficients for the smooth and hard sphere models and the Maxwell model given by Eq. (2.117) are shown in Figs. 5.4 and 5.5 for $x_0 = 5$ and 350, respectively. The curves are normalized by C_{sp} and C_{rough} for the smooth/Maxwell and rough hard spheres models, respectively. In the top panel of Fig. 5.4, for the smaller mass ratios (i.e heavier tracer particle), the curves are noticeably closer to unity. The most massive tracer with $\gamma = 0.0025$, represented by the solid black curve is closest to 1, whereas the lightest tracer with $\gamma = 0.1$, represented by the brown dot-dashed curve is furthest away from 1. The normalized curves in this panel have a constant ratio over the entire time. In the middle panel of the same figure, all the curves are greater than but within a few percent of unity. The smallest mass ratio, $\gamma = 0.0025$, shown by the solid black curve is the furthest away from unity, whereas the lightest tracer with the highest mass ratio, $\gamma = 0.02$, shown by the green dashed curve, is the closest to 1. At higher values of s, where the system is in the initial stages of relaxation, the curves dip lower, and this feature is most prominent for $\gamma = 0.02$. In the approach to equilibrium (i.e. as we move to the left on the curves), the

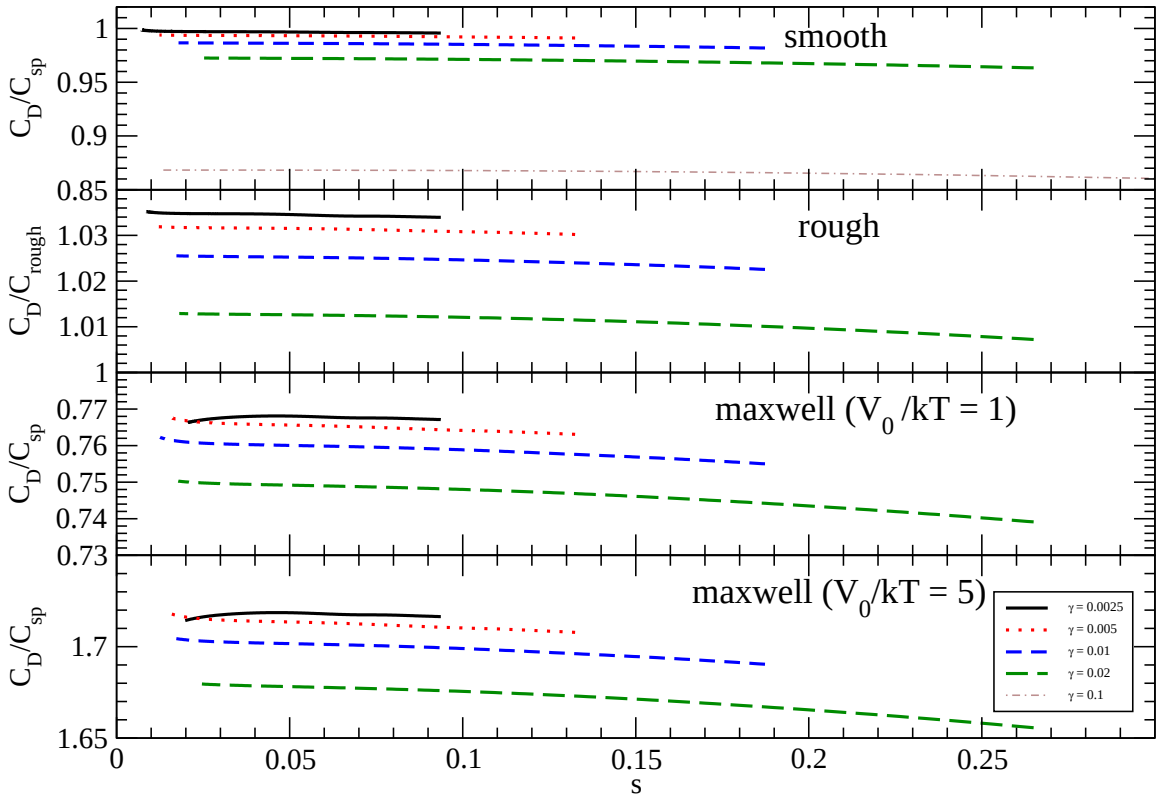

Figure 5.4: Ratios of calculated drag coefficients to analytical predictions for a variety of mass ratios and initial reduced energies of $x_0 = 5$. The top, middle and bottom panels are for the smooth and rough hard spheres and the Maxwell molecule, respectively.

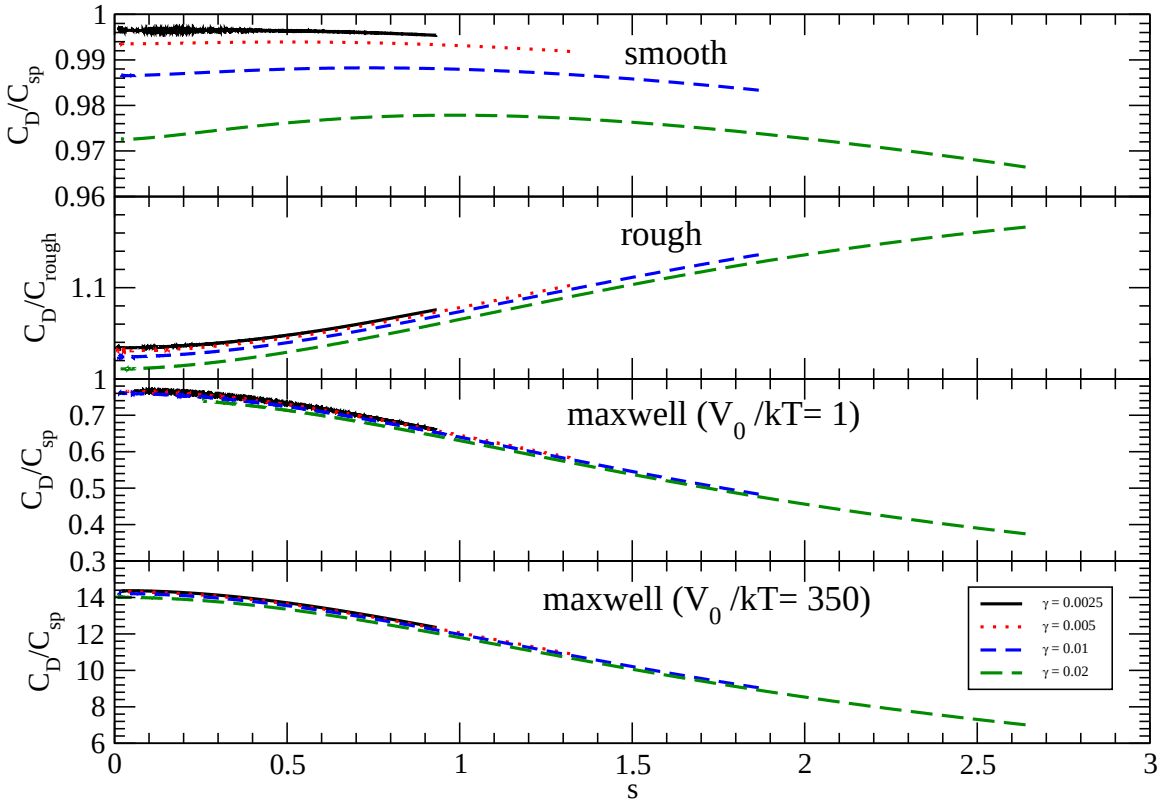

Figure 5.5: Same as Fig. 5.4 except for an initial reduced energy $x_0 = 350$.

ratios become constant. Overall, the normalized curves stay constant for the entire time shown, indicating that the values of the derivative do not change.

In the case of the Maxwell model in the bottom panels of the same figure, the normalized curves are either significantly less than 1 for $V_0/kT = 1$ at $0.750 - 0.769$ and greater than 1 for $V_0/kT = 5$ at $0.65 - 1.72$. The values of $K_{sph}(x, x')$ and $Z_{sph}(x)$ scale with $\sqrt{V_0/kT}$ for the Maxwell molecules as shown by the dependence of the cross section in Eq. (2.110). This can be see in Fig. 5.4 since the limiting values for $V_0/kT = 5$ are $\sqrt{5}$ as large as those for $V_0/kT = 1$. Similarly, in the bottom panels of Fig. 5.5, limiting curves for $V_0/kT = 350$ are $\sqrt{350}$ times those for $V_0/kT = 1$. As defined here, for the Maxwell molecule potential, $V(\sigma_{12}) = V_0$. Thus, the curves for $V_0/kT = 1$ imply the scaling factor σ_{12} is approximately the distance at which the potential is equal to the thermal energy of the bath. That is, $\pi\sigma_{12}^2$ is approximately the cross section the tracer would have at equilibrium. In Figs. 5.4 and 5.5, the curves for $V_0/kT = 1$ are consistently below unity, at approximately 0.75 indicating that C_{sp} is overestimating the kinetic energy decay rate. This is physically reasonable because at $x_0 = 5$, the actual tracer size is smaller than the σ_{12} value at equilibrium, leading to a reduced drag and hence slower rate of decay. Curves with $V_0/kT = 5$ estimate σ_{12} at the initial collision energy $x_0 = 5$ so the conversion in Fig. 5.4 is above unity because the actual decay rate is greater than that predicted by C_{sp}. The situation is analogous but accentuated in Fig. 5.5 for $V_0/kT = 350$. Overall, the results for the smooth hard sphere in the top panels of Figs. 5.4 and 5.5 show that for heavy enough tracers, C_{sp} describes the relaxation well for all speed ratios. This is also consistent with the agreement seen in Fig. 5.3 with the Fokker-Planck predictions. The ratios in the middle panels for the rough hard sphere are consistently too large. This is possibly due to the approximate nature of the collision kernel. For heavy tracers it was argued there should be a larger change in translational energy than rotational energy. Therefore, the rotational degrees of freedom were considered to be at equilibrium. The presence of this cold rotational bath could be causing slightly greater drag than would be expected for the exact rough sphere kernel.

Equations (2.116) and (2.117) were derived assuming an infinitely heavy tracer. This assumption can be relaxed by replacing m_1 with $\mu = \gamma/\sqrt{1+\gamma}$ in the definition of s [90]. Making such a change though causes the ratios in Fig. 5.4 to decrease, moving further from unity. Thus, the deviations seen for example in the upper panel of Fig. 5.3 cannot be corrected by accounting for this mass effect.

5.5 Discussion

This chapter is a more rigorous test of the numerical method given in Chapter 3. It tests the collision kernels that were derived in Chapter 2 for the spherical and rough hard spheres given by Eqs. (2.40) and (2.85), respectively. Overall, the method produced well converged

values for a variety of mass ratios and initial energies for the smooth and rough hard sphere and Maxwell models, which also included conditions that were far from equilibrium.

The expressions given by Andersen and Schuler [58] are derived exactly for the Rayleigh limit, where the tracers are much more massive than the bath gas. The curves shown in Fig. 5.3 show where these expressions can be used for mass ratios close to but not exactly in the Rayleigh limit. The relation given in Eq. (2.112) must be satisfied first before a constant decay rate occurs. In the case where Eq. (2.112) is not satisfied, the energy decay is a lot greater than predicted by Eq. (2.111). This is especially the case for larger mass ratios, γ and this rate of decay tends to decrease over time as the system relaxes. When Eq. (2.112) is satisfied the mass ratio should be less than about 0.0025 where the tracer is 400 times more massive than the bath to give a decay that matches k_R in Eq. (2.111). In the case, where the tracer is only 10 times as massive as the bath, $k_R = 16\sqrt{\gamma}/3$ overestimates the decay rate by about $10 - 15\%$.

The drag coefficients for the three models given in Figs. 5.4 and 5.5 are normalized by the general expression of Stalder [81]. These expressions are valid for infinitely heavy tracers with all values of s. The values of the drag coefficients in this study were calculated using Eq. (2.114) and can be used in energy loss experiments such as those done by Douglas and co-workers [12, 13]. These values of the drag coefficients can be used to obtain cross section values although, accurate values of the drag coefficient C_D are required. It is seen that the exact expressions in Eq. (2.117) work well for hard sphere systems, but fail to take into account the change in effective tracer diameter as the energy changes, and is evident for the Maxwell molecule case in the two lower panels of Figs. 5.4 and 5.5. In these cases, $V_0/kT = 1$ results show that using C_{sp} overestimates the energy decay rate by about 30%, therefore the cross section from the expressions in Eq. (2.117) are underestimated by about the same amount. This is highlighted by the fact that the effective cross sections decrease as energy increases. In the experimental studies, Chen *et al.* [12] found their estimated cross sections, from high energy relaxation, were 30% smaller than those determined by ion mobility experiments performed at energies closer to equilibrium.

In principle, with a good intermolecular potential model, the solutions from solving the Boltzmann equation can be used to correct these differences. One can choose, for example, the Lennard-Jones intermolecular potential which contains an attractive well. Using this potential does not alter the numerical method, except the cross section expression appropriate to the Lennard-Jones potential would have to be substituted in the evaluation of the collision kernel. For a completely realistic model one should also include the effects of inelastic collisions. The results obtained from these simulations can be used in experiments, since the Boltzmann equation is solved exactly. The models are used to define the collision dynamics between the tracers, and the drag coefficients obtained come directly from solving the Boltzmann equation and are not based on any assumptions. Therefore, kinetic energy

loss experiments modelled by these simulations can be used to obtain exact cross section values which can further provide insight into the structure of the tracers. If the tracers are considered to be biomolecules, information about the structure of these biomolecules can be obtained. In experiments, it is assumed that biomolecules in the gas phase retain similar properties as they would in a liquid medium, therefore the results obtained from gaseous studies can be translated to understanding biomolecules in the cells of living organisms.

Chapter 6

Future Work and Concluding Remarks

6.1 Future Work

The theory and numerical method given in Chapters 2 and 3, respectively, are presented for the one-dimensional linear Boltzmann equation. The B-splines posed no problems with obtaining convergence for all the results given in Chapters 4 and 5.

In the next project the numerical method will be extended to solve a two-dimensional linear Boltzmann equation, where two velocity components, for example the parallel and perpendicular components, are represented. For this case the two-dimensional linear Boltzmann equation can be written as

$$\frac{\partial}{\partial t}f(x,y,t) = \int K(x,x',y,y')f(x',y',t)dx'dy' - Z(x,y)f(x,y,t)\ , \tag{6.1}$$

where, as before, the distribution function $f(x,y,t)$ is expanded using B-splines as

$$f(x,y,t) = \sum_{n=-3}^{p-1}\sum_{m=-3}^{q-1} \mathbf{c}_{nm}(t)B_n^3(x)B_m^3(y)\ . \tag{6.2}$$

Substituting Eq. (6.2) into Eq. (6.1) gives

$$\begin{aligned}\sum_{n=-3}^{p-1}\sum_{m=-3}^{q-1}\frac{d}{dt}\mathbf{c}_{nm}(t)B_n^3(x)B_m^3(y) &= \sum_{n=-3}^{p-1}\sum_{m=-3}^{q-1}\mathbf{c}_{nm}(t) \\ &\times \Big[\int_0^S K(x,x',y,y')B_n^3(x')B_m^3(y')dx'dy' \\ &- Z(x,y)B_n^3(x)B_m^3(y)\Big]\ .\end{aligned} \tag{6.3}$$

In Eq. (6.3) the collision kernels can be transformed using formulations similar to those given in Chapter 3. As shown before, the collocation scheme forces the equality in Eq. (6.3) to hold at the grid points (x_k, y_l) where $x_k = k\Delta$ $(k = 0,\ldots,p)$ and $y_l = l\Delta$ $(l = 0,\ldots,q)$, giving $(p+1)(q+1)$ constraints. However, there are $(p+3)(q+3)$ unknown coefficients

$c_{nm}(t)$, so that an additional $2(p+q+4)$ constraints are needed. As before, choosing these constraints is a general process. In analogy with constraint 1 in Section 3.2, the equality in Eq. (6.3) can also be forced at values of (x, y) off the grid, such as, $(x_{1/2}, y_l)$, $(x_{p-1/2}, y_l)$, $(x_k, y_{1/2})$ and $(x_k, y_{q-1/2})$. This gives an additional $2(q+1)+2(p+1)$ constraints. The 4 remaining constraints can follow by imposing the equality in Eq. (6.3) at the points $(x_{1/2}, y_{1/2})$, $(x_{1/2}, y_{q-1/2})$, $(x_{p-1/2}, y_{1/2})$ and $(x_{p-1/2}, y_{q-1/2})$. This set of additional constraints are on the grid completely at the half grid points near the grid edges. With these constraints imposed, Eq. (6.3) can be cast into matrix notation, as

$$\frac{d}{dt}\mathbf{c}_{nm} = \sum \mathbf{L}_{nkml}\mathbf{c}_{kl} \ , \tag{6.4}$$

where $\mathbf{L}_{nkml}$ is a fourth rank tensor and $\mathbf{c}_{nm}$ and $\mathbf{c}_{kl}$ are second rank tensors (matrices).

In comparison the code used for the one-dimensional Boltzmann equation works very smoothly and the computational time required for the calculations to complete was very reasonable. The B-spline formulation changes, such that the matrices given in Eq. (3.27) become fourth rank tensors and the column vector becomes a matrix and will have to incorporate the two variables that have to be integrated. This will make the computing ideally squared of the original time that was required. This would still be on the order of a few hours to a few days. It was the Maxwell model calculations that took the longest amount of time for the calculations to complete. Additionally, this numerical method can further expanded to the full three dimensional linear Boltzmann equation.

Extending the numerical method to a multi-dimensional problem can account for movement of tracers under the effects of external electric and magnetic fields. Since only the kinetic energy decay of tracers was modelled in these studies, a one-dimensional velocity dependent collision kernel is sufficient to describe the collisions. In the case of ion mobility, two components of velocity, that is, perpendicular and parallel components of velocity are required to describe the tracer-bath collisions. Furthermore, the collision induced alignment of tracers can be modelled with a two-dimensional linear Boltzmann equation.

6.2 Concluding Remarks

This thesis presents a numerical method for solving the Boltzmann equation by expanding the distribution function in a basis of cubic B-splines. Collision kernels for the spherical and rough hard spheres have also been derived, and the accuracy of the numerical method has been validated. The formulation of the distribution function can be done in a number of different ways, of which four have been shown in Chapter 3. These formulations can be chosen according to the needs of the problem at hand. Eigenvalues, eigenfunctions and moment values have been obtained using the formulations, and have been compared to the

values that have been published in earlier studies. To test this numerical method the initial distribution functions chosen were far from equilibrium and the mass ratios of the bath to the tracer varied. With the values obtained using different combinations of the initial parameters, it can be concluded that this numerical method is efficient and evaluated well converged values for distribution functions, eigenvalues and moments.

In the second part of the work presented, the numerical method was further scrutinized to monitor the kinetic energy relaxation of heavy tracers that were initialized at high energies. In addition to the smooth and rough hard sphere models, the Maxwell model was also used, which better represents the dynamics between real molecules. These parameters were chosen to mimic mass spectrometry experiments that monitor the relaxation of biomolecules and allow the evaluation of cross sections. The distribution functions and moments were obtained for a Rayleigh gas, for a variety of mass ratios, where the tracer became increasingly heavier than the bath. The derivative of the moments were used to determine the limits of validity for the expression for the rate of kinetic energy relaxation given by the Fokker-Planck equation for a hard sphere gas. The drag coefficients were obtained for all three models using the calculated moment values. These values were compared against analytical expression given in the literature.

This method was designed to be used for the ion mobility problem where external fields have to be accounted for. With the general requirements of the ion mobility problem described in Chapter 1, this method has shown to satisfy the criteria. This numerical method is flexible and can incorporate a variety of initial conditions. There were no issues in terms of convergence or computational time as the system propagated and relaxed to equilibrium. Although the effects of external fields were not incorporated, the numerical method is independent of the formulation of the collision kernel to be used. This allows the inclusion of external field effects, interactions amongst particles and using a higher dimensional linear Boltzmann collision integral. The formulation for extending this numerical method to higher dimensions has been shown in Section 6.1. The general formulation remains the same, and in general, the structure of the matrices has to change.

The other numerical methods available use approximate expressions of the Boltzmann equation. The B-splines method only assumes a bath at equilibrium and the concentration of the tracer is dilute. This makes collisions between two bath particles and two tracer particles negligible, that is, only collisions between a bath and tracer particle are considered. This allows the Boltzmann equation to be reduced and becomes the starting equation for this numerical method. There are no other approximations made, and thus, this method is exact. The numerical methods discussed, work very well for conditions where the system is only slightly perturbed from equilibrium. In the case where the initial conditions of the system are far from equilibrium, these methods do not produce satisfactory results. The B-spline method produced well-converged values for the cases that were initialized far from

equilibrium as well.

Owing to the small domain over which cubic B-splines are defined, the evaluation of the integrals can be done over narrow domains that allow any anomalies in the analytical expressions to be integrated and give accurate values. For example, the discontinuity that occurs at $x = x'$ in the Boltzmann equation did not cause any difficulties in obtain well converged values. The formulation of this numerical method is independent of the formulation of the expression of the collision kernel and allows the algorithm to be parallelized to multi-dimensions. This is an advantage when considering a higher dimensional linear Boltzmann equation. In terms of the computational times, this numerical method is efficient, since all the different cases were run over a range from a few seconds to a few days, with the Maxwell model taking the maximum amount of time. One of the disadvantages of this numerical method was found in the evaluation of the integrals. Each integral was evaluated individually, and was subject to conditions whereby the value of the absolute error between consecutive integral values had to be within a certain range. This caused the convergence to slow down and thus this tolerance had to be relaxed. Even with this change, accuracy was obtained to $4-5$ digits. This was more of a numerical difficulty rather than the formulation of the numerical method.

In comparison to the QDM method, the B-splines do not depend on the form of the distribution function. It assumes that the distribution is smooth and is well represented in the small domain of four units, over which the spline is expressed. The QDM method generates a quadrature that is appropriate for the problem at hand, and thus works very well for that particular problem. But generating the quadrature can also be tedious and time consuming, computationally. A large number of quadrature points and weights can be required to sufficiently describe the distribution function. The QDM also takes into account the form of the collision kernel used for the problem, which is used in the generation of the quadrature. This way the quadrature is dependent on the form of the kernel and are specific only for a particular form. The B-splines are independent of the collision kernel chose, as shown with the use of the smooth and rough hard sphere kernels. This makes it quite versatile. The B-splines are also used to describe conditions for extremely hot tracers far from equilibrium. It has performed quite well and produced satisfactory results. When comparing results obtained using the QDM and the B-splines, there is no doubt, that values produced by the QDM are very accurate. Once again, this is due to the specific quadrature chosen for the problem. This ensures accuracy. In comparison, the B-splines have produced very good values and are almost as accurate as those produced by the QDM.

Bibliography

[1] T. Koga, *Introduction to Kinetic Theory Stochastic Processes in Gaseous Systems*, (Pergammon Press, 1970).

[2] S. Chapman and T. G. Cowling, *The Mathematical Theory of Non-Uniform Gases*, (Cambridge University Press, 1970).

[3] S. Harris, *An Introduction to the Theory of The Boltzmann Equation*, (Holt, Rinehart and Winston, Inc., 1971).

[4] C. Cercignani, *The Boltzmann Equation and its Applications*, (Springer-Verlag, 1988).

[5] S. Golden, *Elements of the Theory of Gases*, (Addison-Wesley Publishing Company, 1964).

[6] T. Wu, *Kinetic Equations of Gases and Plasmas*, (Addison-Wesley Publishing Company, 1966).

[7] E. A. Mason and E. W. McDaniel, *Transport Properties of Ions in Gases*, (Wiley, New York, 1988).

[8] O. I. Rovenskaya, AIP Conference Proceedings, **1333**, 772 (2011).

[9] H. Takeuchi, K. Yamamoto and T. Hyakutake, AIP Conference Proceedings, **1333**, 486 (2011).

[10] K. Yoshimura and S. Kuwabara, AIP Conference Proceedings, **1333**, 87 (2011).

[11] G. M. Alves, G. M. Kremer and A. Soares, AIP Conference Proceedings, **1333**, 643 (2011).

[12] Yu-Luan Chen, B. A. Collings and D. J. Douglas, J. Am. Soc. Mass Spectrom. **8**, 681-687 (1997).

[13] T. Covey and D. J. Douglas, J. Am. Soc. Mass Spectrom. **4**, 616-623 (1993).

[14] T. C. Lilly, A. D. Ketsdever and S. F. Gimelshein, AIP Conference Proceedings, **1333**, 825 (2011).

[15] P. Kowalczyk, A. Palczewski, G. Russo, and Z. Walenta, Eur. J. of Mech. B/Fluids **27**, 62-74 (2008).

[16] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, (Oxford University Press, 1994).

[17] S. K. Stefanov, J. Sci. Comput. **33**, 677-702 (2011).

[18] A. A. Ganjaei, S. S. Nourazar, J. Mech Sci. & Tech. **23**, 2861-2870, (2009).

[19] C. D. Landon, N. G. Hadjiconstantinou, AIP Conference Proceedings **1333**, 277-282, (2011).

[20] H. A. Al-Mohssen, N. G. Hadjiconstantinou, AIP Conference Proceedings **1084**, 257-262, (2008).

[21] A. V. Bobylev, S. Rjasanow, Eur. J. of Mech. B/Fluids **18**, 869-887 (1999).

[22] J. M. Burt and I. D. Boyd, AIP Conference Proceedings, **1333** 230 (2011).

[23] Y. A. Bondar and M. S. Ivanov, AIP Conference Proceedings, **1333** 1209 (2011).

[24] F. Vega Reyes, AIP Conference Proceedings, **1333** 360 (2011).

[25] E. Jun, J. M. Burt and I. D. Boyd, AIP Conference Proceedings, 557 **1333** (2011).

[26] E. D. Farbar and I. D. Boyd, AIP Conference Proceedings, **1333** 242 (2011).

[27] M. Yu. Plotnikov and E. V. Shkarupa, AIP Conference Proceedings, **1333** (2011).

[28] R. Baranowski and M. Thachuk, J. Chem. Phys. **111**, 10061 (1999).

[29] X. Chen and M. Thachuk, J. Chem. Phys. **124**, 174501 (2006).

[30] R. Baranowski, B. Wagner and M. Thachuk, J. Chem. Phys. **114** (2001).

[31] J. Lo, Pseudospectral Methods in Quandtum and Statistical Mechanics (UBC Thesis) (2002).

[32] B. Shizgal and H. Chen, J. Chem. Phys. **104** 4137 (1996).

[33] B. D. Shizgal, J. Mol. Structure (Theochem) **391** 131 - 139 (1997).

[34] J. Lo and B. D. Shizgal, J. Chem. Phys. **125** 194108 (2006).

[35] K. Leung, B. D. Shizgal and H. Chen, J. Math. Chem. **24**, 291-319 (1998).

[36] B. Shizgal and R. Blackmore, Planet. Space Sci. **34** 279-291 (1986).

[37] B. Shizgal, J. Comp. Phys. **41**, 309-328, (1981).

[38] B. Shizgal, R. Blackmore, J. Comp. Phys. **55**, 313-327, (1984).

[39] A. Clarke and B. Shizgal, Phys. Rev. E **49**, 347-358, (1994).

[40] H. Chen, The Quadrature Discretization Method and its Applications (UBC Thesis) 1998.

[41] R. Blackmore, U. Weinert and B. Shizgal, Trans. Theory and Stat. Phys. **15**, 181-120 (1986).

[42] D. Lemonnier, Microscale and Nanoscale Heat Transfer, Topics Appl. Phys. **107**, 77-106 (2007).

[43] H. Skullerud, J. Phys. B **6**, 728 (1973).

[44] P. R. Berman, J. E. M. Haverkort, and J. P. Woerdman, Phys. Rev. A **34**, 4647 (1986).

[45] J. Park, N. Shafer, and R. Bersohn, J. Chem. Phys. **91**, 7861 (1989).

[46] C. A. Taatjes, J. I. Cline, and S. R. Leone, *ibid.* **93**, 6554 (1990).

[47] K. E. Gibble and A. Gallagher, Phys. Rev. A **43**, 1366 1993.

[48] K. E. Gibble and J. Cooper, *ibid.* **44**, R5335 (1991).

[49] G. Nan and P. L. Houston, J. Chem. Phys. **97**, 7865 (1992).

[50] D. A. Shapiro, J. Phys. B **33**, L43-L49 (2000).

[51] O. V. Belai, O. Y. Schwartz, and D. A. Shapiro, Phys. Rev. A **76**, 012513 (2007).

[52] C. E. Siewert, J. Quant. Spec. & Rad. Trans. **74**, 789-796 (2002).

[53] E. P. Wigner and J. E. Wilkins, U.S. Atomic Energy Commission Report AECD 2275, 1944.

[54] C. Cercignani, *Mathematical Methods in Kinetic Theory*, 2^{nd} Ed., (Plenum Press, 1990).

[55] M. Kac, Foundations of Kinetic Theory, Third Berkeley Symp. on Math. Statist. and Prob., Vol. 3, 171-197 (Univ. of Calif. Press 1956).

[56] J. H. Ferziger and H. G. Kaper, *Mathematical Theory of Transport Processes in Gases* (North-Holland, Amsterdam, 1972).

[57] L. Waldmann in *Handbuch der Physik*, edited by S. Flügge (Springer-Verlag, Berlin, 1958), Vol. 12, p. 348.

[58] K. Andersen and K. E. Schuler, J. Chem. Phys. **40**, 633 (1964).

[59] P. F. Liao, J. E. Bjorkholm, and P. R. Berman, Phys. Rev. A **21**, 1927 (1980).

[60] P. R. Berman, Adv. At. Mol. Phys. **13**, 57 (1978).

[61] R. Kapral and J. Ross, J. Chem. Phys. **52**, 1238 (1970).

[62] L. Monchick and E. A. Mason, Phys. Fluids **10**, 1377 (1967).

[63] B. Shizgal and R. Blackmore, Chem. Phys. **77**, 417 (1983).

[64] M. R. Hoare and C. H. Kaplinsky, J.Chem. Phys. **52**, 3336 (1970).

[65] M. R. Hoare, The Linear Gas, *Advances in Chemical Physics* **20** John Wiley & Sons, 135-214 (1971).

[66] S. E. Nielsen and T. A. Bak, J. Chem. Phys. **41**, 665 (1964).

[67] G. H. Bryan, Rep. Br. Assoc. Advmt. Sci. **64**, 64 (1894).

[68] F. B. Pidduck, Proc. R. Soc. London. Ser. A **101**, 101 (1922).

[69] J. ODell and B. Berne, J. Chem. Phys. **63**, 2376 (1975).

[70] B. Berne, *ibid.* **66**, 2821 (1977).

[71] C. Pangali and B. Berne, *ibid.* **67**, 4590 (1977).

[72] J. Montgomery and B. Berne, iibid. **67**, 4580 (1977).

[73] O. Kravchenko and M. Thachuk, J. Chem. Phys. **134**, 114310 (2011). The value of **J** used in this reference is the negative of that employed in the current work, with particles "1" and "2" being the tracer and bath, respectively.

[74] I. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series, and Products*, 5^{th} ed. (Academic Press, San Diego, 1994), integral (6.643.2).

[75] With a change of variable, $s = (1+m_1\chi)\sin^2\theta/(1+m_1\chi\sin^2\theta)$, and using the properties of Kummer's function giving $M(3/2,1,z) = \exp(z/2)[(1+z)I_0(z/2) + zI_1(z/2)]$, Eq. (2.77) can be written in simplified notation as $(8\pi\chi kT)^{3/2}F_2(\frac{m_1}{2kT}|\boldsymbol{\beta}\times\hat{\mathbf{k}}|^2; m_1\chi)$ in which $F_2(z;a) = \frac{1}{1+a}\int_0^1 ds[e^{bs}/\sqrt{1-s}][(1/2+bs)I_0(bs) + bsI_1(bs)]$ with $b = az/[2(1+a)]$. Using a numerical comparison, and also expanding this function in a

Taylor series up to seventh order, it was determined that $F_z(z;a) = e^{az/(1+a)}/(1+a)$. Using this relation produced the result of Eq. (2.78).

[76] S. Khurana and M. Thachuk, J. Chem. Phys. **139**, 164122 (2013).

[77] J. C. Maxwell, Phil. Trans. R. Soc. Lond. **157**, 49-88 (1867).

[78] F. R. W. McCourt, J. J. M. Beenakker, W. E. Kohler, and I. Kuscer, *Nonequilibrium Phenomena in Polyatomic Gases* Volume I (Clarendon Press, Oxford 1990).

[79] Y. Chang and B. Shizgal, AIP Cong. Proc. (2009).

[80] P. S. Epstein, Phys. Rev. **23**, 710 (1924).

[81] J. R. Stalder and V. J. Zurick *National Advisory Committee for Aeronautics TN*, NACA, 2423 (1951).

[82] S. Khurana and M. Thachuk, J. Chem. Phys. **136**, 094103 (2012).

[83] S. Khurana and M. Thachuk, *in preparation.*

[84] M. N. Schultz, *Spline Analysis*, (Prentice Hall, New Jersey, 1973).

[85] P. M. Prenter, *Splines and Variational Methods*, (John Wiley, New York, 1975).

[86] B. W. Shore, J. Chem. Phys. **58**, 3855-3866, (1973).

[87] L. L. Schumaber, *Spline Functions - Basic Theory*, (Krieger Publishing Company, 1993).

[88] W. Press, S. A. Teuksolky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 1^{st} Ed. (Cambridge University Press, 1986).

[89] B. Shizgal, M. J. Lindenfeld and R. Reeves, J. Chem. Phys. **56**, 249-260 (1981).

[90] Z. Li and H. Wong, Phys. Rev. E. **65** 061207 (2003).

Appendix A

Appendix A: Numerical Code

Given here is the code used to obtain the results given in this thesis.

A.1 Main Program

```
PROGRAM ION_MOBILITY
USE IONMOBILITY
USE MODULE_CROSSEC
IMPLICIT NONE

!This program calls on the module ionmobility, which describes some of the
    variables being used.

REAL(hiprec) :: t_0,w,y_temp,vz_n,y_temp_deriv,y_temp_2_deriv,vz, moment_int,
    moment1, moment2,moment0,fnc_plot,i_plot,a_lim
REAL(hiprec) :: b_lim,z,eval_gauss,moment_mod,moment1_mod,moment2_mod,
    moment2a_mod,x,moment1_diff,eval_gaulag,numdense,sigma12,call1,x1,g,
    check_cross,chi_val,beta,alpha
!vz_0: the starting point of velocity grid
!vz_n: the end point of velocity grid
!t_0: initial time
!w: argument used in bspline function
!y_temp: temporary value of unknown function
!y_temp_2_deriv: second derivative of unknown function
!y_temp_deriv: first derivative of unknown function
!dist_func: distribution function value at a particular point on grid
!vz: velocity grid
INTEGER i,j,nstep,info,n,flag2,flag3,mod_calc,tmp_read,lda,incx,incy
!defining grid for vz
INTEGER,PARAMETER :: n_max_gauss = 3
INTEGER,DIMENSION (:),ALLOCATABLE:: ipiv
REAL(hiprec), DIMENSION (:),ALLOCATABLE ::coeff_prod,temp,coeff_new,nu_h,
    coeff_old,coeff_h
!An array provided as an output
CHARACTER(40):: read_tmp
LOGICAL done
REAL(hiprec),DIMENSION (:,:), ALLOCATABLE :: wr_new,coeff,lmat_check,
    b_init_saved
```

```
!This is a matrix used in the evaluation of the coefficients.
REAL(hiprec),EXTERNAL :: bspline3,exe_func,plot,dist_func,integrate,
    int_interval,integralw,k_matrix,F_1,F_2,crossec
!Bspline is an external function which include the spline functions

!Following variables are entered in an input file separated from this file.
    We can change these parameters as desired when the code is given the
    command to compile.
!vz_n: end point of velocity grid and vz_num is total number of points on
    velocity grid

READ(*,*) vz_0,vz_n,vz_num
!m: mass ratio of bath to ion and colfreq is collision frequency
READ(*,*)m,colfreq
!nstep and h are number of steps and size of interval respectively.
!These two values can be set to any desired value, as the program progresses.
READ(*,*) nstep,h
READ(*,*) spc
!max_smth is the variable that distinguishes between the calculation for
    smooth sphere or maxwell molecule
!max_smth = 1 is smooth sphere
!max_smth = 2 is maxwell molecule
!These are the matrices and array whoses sizes are dependent on the input of
    the values for the variables above.
!Thus these can be changed according to the problem.
!These matrices/arrays are used throughtout different subroutines and
    functions.
n_dim = vz_num+2
ALLOCATE(b_init(0:n_dim,0:n_dim))
ALLOCATE(b_init_saved(0:n_dim,0:n_dim))
ALLOCATE(l1_mat(0:n_dim,0:n_dim))
ALLOCATE(l2_mat(0:n_dim,0:n_dim))
ALLOCATE(vz_grid(0:vz_num))
ALLOCATE(ipiv(0:n_dim))
ALLOCATE(y(0:n_dim))
ALLOCATE(k1(0:n_dim))
ALLOCATE(k2(0:n_dim))
ALLOCATE(k3(0:n_dim))
ALLOCATE(k4(0:n_dim))
ALLOCATE(y1(0:n_dim))
ALLOCATE(temp(0:n_dim))
ALLOCATE(coeff_new(0:n_dim))
ALLOCATE(wr_new(0:n_dim,0:n_dim))
ALLOCATE(coeff(0:n_dim,0:n_dim))
ALLOCATE(coeff_h(0:n_dim))
```

```
OPEN (10,FILE = "ini_func_values")
OPEN (11,FILE = "rk4_values")
OPEN (12,FILE = "dist_func_values")
OPEN (14,FILE = "mat_values_ww")
OPEN (15,FILE = "x_gauss_w_gauss")
OPEN (17,FILE = "x_gaulag_w_gaulag")
OPEN (18,FILE = "moment_values")
OPEN (19,FILE = "Integrand_values")
OPEN (20,FILE = "moment0")
OPEN (21,FILE = "moment1")
OPEN (22,FILE = "moment2")
OPEN (23,FILE = "hspace_coeff")
OPEN (24,FILE = "moment_mod_values")
OPEN (25,FILE = "values_integrand")
OPEN (26,FILE = "initial_func_values")
OPEN (27,FILE = "dist_values")
OPEN (30,FILE = "gaulag_convergence")
!Open all files throughout entire code, this one time, and then data is stored
     accordingly to its respective file.

q = 0.5d0*(1.0d0/sqrt(m) + sqrt(m))
r = 0.5d0*(1.0d0/sqrt(m) - sqrt(m))
!q and r functions of the mass ratio as defined
eval_gauss = int_interval(-1,0,0.0d0,1.0d0,0,0)
!evaluates points and weights of gauss-legendre quadrature

dvz = (vz_n - vz_0)/REAL(vz_num)
IF (spc == 6 .or. spc == 7 .OR. spc == 11) THEN
   done = .TRUE.
   OPEN(38, FILE = 'K_MAT_VALUES_EXTRAP')
   DO WHILE (done)
      READ(38, *,END=10000) kmat_m, kmat_vz_0, kmat_vz_n, kmat_vz_num
      ALLOCATE(kmat_x_val(0:kmat_vz_num))
      ALLOCATE(kmat_y_val(0:kmat_vz_num))
      ALLOCATE(k_mat_val(0:kmat_vz_num,0:kmat_vz_num))
      READ(38,*,END = 10000) kmat_x_val,kmat_y_val,k_mat_val
       kmat_dvz = (kmat_vz_n - kmat_vz_0)/REAL(kmat_vz_num)
   IF (kmat_m == m) THEN
     DONE = .FALSE.
   ELSE
      DEALLOCATE(kmat_x_val)
      DEALLOCATE(kmat_y_val)
      DEALLOCATE(k_mat_val)
   END IF
END DO
```

```
!check mass ratios between input files to determine which set of data to use
    for further calculation
   CLOSE(38)
END IF
IF (spc == 10. .OR. spc == 11) THEN ! spc = 9 and 10 as well
   READ(*,*) mu_chi
END IF
IF (spc == 5 .OR. spc == 6 .OR. spc == 7) THEN
   READ(*,*) numdense,sigma12,max_smth
   eval_gaulag = int_interval(-2,0,0,0,0,0)
   IF (max_smth == 2) THEN
     READ(*,*) a4_tilda
     CALL EVAL_CHI

!OPEN (44, FILE = "max_cross")
!DO chi_val = 0.001d0,pi,0.001d0
   !g = 4.0d0*sqrt(a4_tilda*m*(1.0d0+m))
!       check_cross = crossec(2,1.0d0,chi_val)
!END DO
!CLOSE(44)
   END IF
END IF
!CALL BESSEL
!
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! small loop written to evaluate the 1/r^4 potential for maxwell molecule
!DO x1 = 0.0d0, 1.0d0, 0.001
!    call1 = 1.0d0/(x1**4)
!    WRITE(80, '(e15.10,1x,e35.30)') x1,call1
!END DO
!STOP
!checking k_mat function for spaces 5 and above (those that require
    interpolation/extrapolation using externally generated matrix)
!DO x1 = 0.0d0,10.0d0,0.001d0
!x1 = 275.0d0
   !call1 = k_matrix(spc,2.0d0,x1,coeff)
!    call2 = eval_chi
   !WRITE(57,'E15.10,3X,E20.15') x1,call1
   !WRITE(57,'E15.10,1X,E25.20') x1,call1
!    WRITE(57,'(E15.10,1X,E35.30)') x1,call1
!END DO
!STOP
!
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
!Setting  beginning of velocity grid with 0.
!vz_0 = 0
!With this we set spacings between points on velocity grid. This is dependednt
     on total length of grid,goes back to variables that are input in the
    beginning.
!Spacing between points is uniform.

!dvz = (vz_n - vz_0)/REAL(vz_num)

!This loop creates array for velocity grid.
DO i = 0,vz_num
   vz_grid(i)= vz_0 + REAL(i)*dvz
END DO

b_init = 0.0d0
DO i = 1, n_dim-1
   b_init(i,i) = 4.0d0/6.0d0
   b_init(i,i-1)= 1.0d0/6.0d0
   b_init(i,i+1)= 1.0d0/6.0d0
END DO

!Values of matrix b_init, are based on boundary conditions mentioned above.
    These are the first 4 elements in first row, and last 4 elements in last
    row in matrix.
b_init(0,0) = 1.0d0/48.0d0
b_init(0,3) = 1.0d0/48.0d0
b_init(0,1) = 23.0d0/48.0d0
b_init(0,2) = 23.0d0/48.0d0
b_init(n_dim,n_dim) = 1.0d0/48.0d0
b_init(n_dim,n_dim-3) = 1.0d0/48.0d0
b_init(n_dim,n_dim-1) = 23.0d0/48.0d0
b_init(n_dim,n_dim-2) = 23.0d0/48.0d0

b_init_saved = b_init
ALLOCATE(work(0:n_dim+1))
CALL DGETRF(n_dim+1,n_dim+1,b_init,n_dim+1,ipiv,info)
IF (info /= 0) STOP "DGETRF: FAILED"
CALL DGETRI(n_dim+1,b_init,n_dim+1,ipiv,work,n_dim+1,info)
IF (info /=0) STOP "DGETRI: FAILED"

IF (spc == 3) THEN
   ALLOCATE(nu_h(0:n_dim))
   nu_h = 0.0d0
   DO j = 1, n_dim-1
      nu_h(j) = integrate(3,flag2,vz_grid(0),vz_grid(vz_num),vz_grid(j-1),
          coeff)
```

```
        WRITE(14,*) nu_h(j)
    END DO
    nu_h(0) = integrate(3,flag2,vz_grid(0),vz_grid(vz_num),0.5d0*dvz,coeff)
    WRITE(14,*) nu_h(0)
    nu_h(n_dim) = integrate(3,flag2,vz_grid(0),vz_grid(vz_num),vz_grid(vz_num)
        -0.5d0*dvz,coeff)
    WRITE(14,*) nu_h(n_dim)
ELSE
        CALL WIGNER_WILKINS
!wigner wilkins subroutine contains construction of all matrices needed for
    evaluation of distribution function. More comments on details in
    subroutine file itself. This subroutine evaluates matrix L, which is the
    collision operator that will be used for evaluation of distribution
    function.
END IF

OPEN (29, FILE = "plot_values")
DO i_plot = vz_0,vz_n,0.05
    fnc_plot = plot(i_plot)
    WRITE(29,*) i_plot,fnc_plot
END DO
CLOSE(29)

!This loop evaluates values of time dependent coefficients used in evalution
    of distribution function.
DO i = 0,vz_num
    y(i+1)= exe_func(0,vz_grid(i))
END DO
!These two are first and last values of coefficients in array, based on two
    extra boundary conditions that we have to include.
!These bounday conditions are values of function at mid-points between first
    and second point and mid-point between last and second-to-last point on
    grid.
y(0)= exe_func(0,vz_grid(0)+(dvz/2.0d0))
y(n_dim) = exe_func(0,vz_grid(vz_num)-dvz/2.0d0)

WRITE (10,*)y
!y = MATMUL(b_init,y)
CALL DGEMV('N',n_dim+1,n_dim+1,1.0d0,b_init,n_dim+1,y,1,0.0d0,temp,1)

WRITE (10,*)temp

h2 = h/2.0d0
t_0 = 0.0d0
!temp = y
```

```
!This loop evaluates values of velocity grid, test function, first and second
    derivatives of both the actual function and bspline function, and values
    ofdistribution function.
!This loop also evaluates errors obtained between respective values of test
    function and bspline functions and their derivatives.
DO vz = vz_0, vz_n, 0.05
  !WRITE (10,"(F7.4,3X,E10.4,3X,E10.4,3X,E10.4,3X,E10.4,3X,F8.2,3X,F8.2,3X,E10
      .4,3X,E10.4,3X,F8.2)")vz, dist_func(1,0,vz,y), exe_func(0,vz), dist_func
      (1,1,vz,y), exe_func(1,vz),((dist_func(1,0,vz,y)-exe_func(0,vz))/exe_func
      (0,vz))*100,&
  !((dist_func(1,1,vz,y)-exe_func(1,vz))/exe_func(1,vz))*100, exe_func(2,vz),
      dist_func(1,2,vz,y),((dist_func(1,2,vz,y)- exe_func(2,vz))/exe_func(2,vz
      ))*100
  WRITE(26,"(F11.7,3X,E10.4)")vz, dist_func(1,0,vz,temp)
END DO

!~~~~~~~~~~~~~~~~~~TIME PROPAGATION OF COEFFICIENTS~~~~~~~~~~~~~~~~~~~~~~~~~~
ALLOCATE(coeff_old(0:n_dim))
a_lim = vz_grid(0)
b_lim = vz_grid(vz_num)
coeff_old = temp
moment0 = integrate(1,0,a_lim,b_lim,1,coeff_old)
moment1 = integrate(1,1,a_lim,b_lim,1,coeff_old)/moment0
moment1_mod = log(moment1-1.5d0)
WRITE(21,'(F9.4,3(2X,E14.8))') 0.0d0, moment1, 2.0d0/3.0d0*moment1, moment1_mod
DO j = 1, nstep
   t = t_0 + REAL(j)*h
   IF (spc == 3) THEN
      DO n = 0, n_dim
         IF (n == 0) THEN
            z = 0.5d0*dvz
         ELSE IF (n == n_dim) THEN
            z = vz_grid(vz_num)-0.5d0*dvz
         ELSE
            z = vz_grid(n-1)
            !value of z is what is passed in as the value of y into k_matrix
         END IF
         coeff_h(n) = -1.0d0*integrate(4,0,vz_grid(0),vz_grid(vz_num),z,
             coeff_old)
      END DO
      temp = coeff_h+nu_h
      CALL DGEMV('N',n_dim+1,n_dim+1,1.0,b_init,n_dim+1,temp,1,0.0d0,coeff_new
          ,1)
      !coeff_new = coeff_old + MATMUL(b_init,coeff_h+nu_h)*h
      coeff_new = coeff_old + coeff_new*h
      !WRITE(23,*) coeff_h
```

```
    !WRITE(23,*) nu_h
    !WRITE(23,*) t, coeff_new
    !WRITE(23,"A1")'&'
ELSE IF (spc == 1 .OR. spc == 2 .OR. spc == 4 .OR. spc == 5 .OR. spc == 6 .
    OR. spc == 7 .OR. spc == 9 .OR. spc == 10 .OR. spc == 11) THEN
    !wr_new = 0.0d0
    !this loop creates the diagonalized matrix with the eigenvalues
        evaluated in the l_init file.
    !we are using exact solution to complete time propogation of system.
    !We are evaluating the matrix, and then we obtain time-dependent
        coefficients at any time, by taking product of diagonalized matrix
        and coefficients evaluated at t=0.
    !individual matrices are constructed/solved in l_init file
    !time dependence of distribution function is evaluted here using exact
        expression

    !t = t_0 + REAL(j-1)*h
    !DO i = 0,n_dim
    !    wr_new(i,i) = exp(wr(i)*t)
    !END DO
    !coeff = MATMUL(vr_saved,(MATMUL(wr_new,vr)))
    !coeff_new = MATMUL(coeff,y)
    CALL DGEMV('N',n_dim+1,n_dim+1,1.0d0,l_mat_safe,n_dim+1,coeff_old,1,0.0
        d0,coeff_new,1)
    !coeff_new = coeff_old + MATMUL(l_mat_safe,coeff_old)*h
    coeff_new = coeff_old + coeff_new*h
END IF
!this initializes gaulag subroutine, in which points and weights of gauss-
    laguerre are calculated and then stored in a matrix form.
!an iteration takes places and depending on convergence, appropriate set of
     points and weights are used in evaluation of moments.
IF (spc == 3 .OR. spc == 1 .OR. spc == 4 .OR. spc == 5 .OR. spc == 7 .OR.
    spc == 9 .OR. spc == 10 .OR. spc == 11) THEN
    mod_calc = mod(j,10)
    IF (mod_calc == 0) THEN
        moment0 = integrate(1,0,vz_grid(0),vz_grid(vz_num),1.0d0,coeff_new)
        !evaluates the zeorth order moment (norm)
        WRITE(20,*) t,moment0
        !WRITE(24,*) t,moment0
        !take derivative of the first moment
        moment1 = integrate(1,1,a_lim,b_lim,1,coeff_new)/moment0
        moment1_mod = log(moment1-1.5d0)
        !evaluates the first moment, and is normalized by the norm
        WRITE(21,"(F9.4,3(2X,E14.8))") t,moment1,2.0d0/3.0d0*moment1,
            moment1_mod
        !WRITE(24,*) moment1_mod
```

```
        moment2 = integrate(1,2,a_lim,b_lim,1,coeff_new)/moment0
        moment2_mod = (3.0d0/5.0d0)*moment2/(moment1**2)
        moment2a_mod = moment2 - ((5.0d0/3.0d0)*moment1**2)
        !evaluates the second moment, and is normalized by the  norm
        WRITE(22,*) t,moment2
        WRITE(24,*) t,moment2_mod,moment2a_mod
        !WRITE (22,"A1")'&'
        !DO i = 0, n_dim
            !WRITE(12,*) i, coeff_new(i)
            !We obtain time-dependent coefficients, and  plots of these values
        !END DO
        DO vz = vz_0,vz_n,0.05
            !WRITE (12,"(F10.6,3X,E13.7,3X,E13.7,3X,E13.7)")vz,dist_func(1,0,
                vz,coeff_new)!,dist_func(3,0,vz,coeff_new)!,-1*dist_func(2,0,
                vz,coeff_new)
            WRITE (12,*)vz,dist_func(1,0,vz,coeff_new)!,dist_func(3,0,vz,
                coeff_new)!,-1*dist_func(2,0,vz,coeff_new)
        END DO
        WRITE (12,"(A1)")'&'
    END IF
    coeff_old = coeff_new
ELSE
    moment0 = integrate(1,0,vz_grid(0),vz_grid(vz_num),1.0d0,coeff_new)
    !evaluates zeorth order moment (norm)
    WRITE(20,*) t,moment0
    !WRITE(24,*) t,moment0
    !take derivative of first moment
    moment1 = integrate(1,1,a_lim,b_lim,1,coeff_new)/moment0
    moment1_mod = log(moment1-1.5d0)
    !moment1_mod = (2.0d0/3.0d0)*moment1
    ! evaluates the first moment, and is normalized by the norm
    WRITE(21,"(f9.4,3(2x,e14.8))") t,moment1,2.0d0/3.0d0*moment1,moment1_mod
    !WRITE(21,*) t,moment1,moment1_mod
    !WRITE(24,*) moment1_mod
    moment2 = integrate(1,2,a_lim,b_lim,1,coeff_new)/moment0
    moment2_mod = (3.0d0/5.0d0)*moment2/(moment1**2)
    moment2a_mod = moment2 - ((5.0d0/3.0d0)*moment1**2)
    ! evaluates second moment, and is normalized by norm
    WRITE(22,*) t,moment2
    WRITE(24,*) t,moment2_mod,moment2a_mod
    !WRITE (22,"A1")'&'
    IF (spc == 2 ) THEN
        flag3 = 2
    ELSE IF (spc == 4) THEN
        flag3 = 4
    ELSE IF (spc == 5) THEN
```

```
            flag3 = 5
        END IF
        DO i = 0, n_dim
            !WRITE(12,*) i, coeff_new(i)
            !We obtain time-dependent coefficients, and plots of these values
        END DO
        DO vz = vz_0,vz_n,0.05
            !WRITE (12,"(F10.6,3X,E13.7,3X,E13.7,3X,E13.7)")vz,dist_func(1,0,vz,
                coeff_new),dist_func(flag3,0,vz,coeff_new)!,-1*dist_func(2,0,vz,
                coeff_new)
            WRITE (12,*)vz,dist_func(1,0,vz,coeff_new),dist_func(flag3,0,vz,
                coeff_new)!,-1*dist_func(2,0,vz,coeff_new)
        END DO
        WRITE (12,"(A1)")'&'
        coeff_old = coeff_new
    END IF
END DO
!evaluate time dependent coefficients above with exact mathematical
    formulation.

!This loop evaluates distribution function at points on grid.
!The distribution function is evaluated for original bspline, first derivative
    and second derivative.
CLOSE (10)
CLOSE (11)
CLOSE (12)
CLOSE (14)
CLOSE (15)
CLOSE (17)
CLOSE (18)
CLOSE (19)
CLOSE (20)
CLOSE (21)
CLOSE (22)
CLOSE (23)
CLOSE (24)
CLOSE (25)
CLOSE (26)
CLOSE (27)
CLOSE (30)

10000 IF (spc == 6 .and. done) WRITE(*,*) "MASS_RATIO_NOT_FOUND_IN_
    K_MAT_VALUES_EXTRAP"

END PROGRAM ION_MOBILITY

PROGRAM ANALYZE
```

```
!
!  analyze moment values
!
USE IONMOBILITY
IMPLICIT NONE

REAL(hiprec) :: error,vz,tmp1,tmp2,tmp3,tmp4,j,enrgy,s,eq_rgh1,eq_rgh2,
    eq_smth1,eq_smth2,eq_rgh1a,eq_rgh2a,red_mass,s_red_mass,eq_rgh1_rm,
    eq_rgh2_rm,eq_smth1_rm,eq_smth2_rm,eq_rgh1a_rm,eq_rgh2a_rm
INTEGER i,info
INTEGER,DIMENSION (:),ALLOCATABLE:: ipiv,md
REAL(hiprec),EXTERNAL ::bspline3,dist_func
DOUBLE PRECISION,EXTERNAL:: erf
INTERFACE
        SUBROUTINE POLINT(xa,ya,x,y,dy)
        INTEGER,PARAMETER::I4B = SELECTED_INT_KIND(9),hiprec=KIND(0.0D0)
        REAL(hiprec),DIMENSION(:) :: xa,ya
        REAL(hiprec):: x
        REAL(hiprec):: y,dy
        END SUBROUTINE
END INTERFACE

!  vz_num is the number of moment values to process

READ(5,*) vz_num ,m,enrgy

n_dim = vz_num+2
ALLOCATE(b_init(0:n_dim,0:n_dim))
ALLOCATE(vz_grid(0:vz_num))
ALLOCATE(ipiv(0:n_dim))
ALLOCATE(y(0:n_dim))
ALLOCATE(md(0:2*vz_num))


OPEN (10,FILE = 'coefficients')
OPEN (25,FILE = 'moment_derivative')

!  read in moment values from standard input

DO i = 0,vz_num
   READ(5,*) tmp1, tmp2, tmp3, tmp4
   vz_grid(i)= tmp1
   y(i+1)= tmp4
END DO

dvz = vz_grid(1) - vz_grid(0)
```

```
b_init = 0.0d0
DO i = 1, n_dim-1
    b_init(i,i) = 4.0d0/6.0d0
    b_init(i,i-1)= 1.0d0/6.0d0
    b_init(i,i+1)= 1.0d0/6.0d0
END DO

b_init(0,0) = 1.0d0/48.0d0
b_init(0,3) = 1.0d0/48.0d0
b_init(0,1) = 23.0d0/48.0d0
b_init(0,2) = 23.0d0/48.0d0
b_init(n_dim,n_dim) = 1.0d0/48.0d0
b_init(n_dim,n_dim-3) = 1.0d0/48.0d0
b_init(n_dim,n_dim-1) = 23.0d0/48.0d0
b_init(n_dim,n_dim-2) = 23.0d0/48.0d0

ALLOCATE(work(0:n_dim+1))
CALL DGETRF(n_dim+1,n_dim+1,b_init,n_dim+1,ipiv,info)
IF (info /= 0) STOP "DGETRF: FAILED"
CALL DGETRI(n_dim+1,b_init,n_dim+1,ipiv,work,n_dim+1,info)
IF (info /=0) STOP "DGETRI: FAILED"

! use polint to get values at midpoints for boundary conditions
CALL POLINT(vz_grid(0:5),y(1:6),(vz_grid(0)+vz_grid(1))*0.5d0,y(0),error)
        !write(*,*) y(0),error
CALL POLINT(vz_grid(vz_num-6:vz_num),y(n_dim-7:n_dim-1),(vz_grid(vz_num)+
    vz_grid(vz_num-1))*0.5d0,y(n_dim),error)
        !WRITE(*,*) y(n_dim),error

WRITE (10,*)y
y = MATMUL(b_init,y)
WRITE (10,*)y

OPEN (40,FILE = 'moment_derivative_a')
DO i = 0, 2*vz_num
    vz = REAL(i)*0.5d0*dvz
    tmp1 = dist_func(1,0,vz,y)
    tmp2 = dist_func(1,1,vz,y)
    red_mass = m/(1.0d0+m)
    s = sqrt(m*(exp(tmp1)))
    s_red_mass = sqrt(red_mass*(exp(tmp1)))
! speed
    eq_smth1 = 16.0d0/(3.0d0*sqrt(pi)*s)
    eq_smth1_rm = 16.0d0/(3.0d0*sqrt(pi)*s_red_mass)
! epstein result for smooth sphere
```

```
      eq_smth2 = ((2.0d0*exp(-(s*s)))/(sqrt(pi)*s))*(1.0d0+(0.5d0/(s*s)))+2.0d0
          *(1.0d0+(1.0d0/(s*s))-(1.0d0/(4.0d0*s**4)))*erf(s)
      eq_smth2_rm = (2.0d0*exp(-s_red_mass*s_red_mass)/(sqrt(pi)*s_red_mass))
          *(1.0d0+0.5d0/(s_red_mass*s_red_mass))+2.0d0*(1.0d0+1.0d0/(s_red_mass*
          s_red_mass)-1.0d0/(4.0d0*s_red_mass**4))*erf(s_red_mass)
! stalder result for smooth sphere
      eq_rgh1 = eq_smth1*13.0d0/9.0d0
      eq_rgh1_rm = eq_smth1*13.0d0/9.0d0
      eq_rgh1a = eq_smth1*7.0d0/5.0d0
      eq_rgh1a_rm = eq_smth1*7.0d0/5.0d0
! epstein result for rough sphere
      eq_rgh2 = eq_smth2 + (2.0d0*sqrt(pi))/(3.0d0*s)
      eq_rgh2_rm = eq_smth2 + (2.0d0*sqrt(pi))/(3.0d0*s_red_mass)
      eq_rgh2a = eq_rgh2*(7.0d0/5.0d0)*(9.0d0/13.0d0)
      eq_rgh2a_rm = eq_rgh2_rm*7.0d0/5.0d0
! stalder result for rough sphere
      WRITE(25,'(F11.5,3X,E22.15,3X,E22.15,3X,E22.15)') vz, tmp1, tmp2,tmp2
          /(-16.0d0/(3.0d0*sqrt(1/m)))
! prints out the fokker planck results: time, value of moment, derivative of
     moment, ratio of derivative with actual fokker planck result
      WRITE(26,'(F11.5,3X,E15.10,3X,E15.10,3X,E13.7,3X,E13.7,3X,E13.7,3X,E13.7)')
           vz, sqrt(m*(exp(tmp1))), 1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2,eq_smth1
          ,eq_smth2,(1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2)/eq_smth1,(1/(-m*sqrt(
          pi))/sqrt(exp(tmp1))*tmp2)/eq_smth2
! fort.26 is for the smooth sphere results
! prints values for vz, speed, drag coefficients, epstein result for drag
     coefficients, stalder result for drag coefficient, ratio of drag
     coefficient to the epstein result
      WRITE(27,'(F11.5,3X,E15.10,3X,E15.10,3X,E13.7,3X,E13.7,3X,E13.7,3X,E13.7,3X
          ,E13.7,3X,E13.7)') vz, sqrt(m*(exp(tmp1))), 1/(-m*sqrt(pi))/sqrt(exp(
          tmp1))*tmp2,eq_rgh1,eq_rgh2,(1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2)/
          eq_rgh1,(1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2)/eq_rgh1a,(1/(-m*sqrt(pi)
          )/sqrt(exp(tmp1))*tmp2)/eq_rgh2,(1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2)/
          eq_rgh2a
! fort.27 prints the same results as above for fort.26 except for the rough
     sphere, the last column multiplies the rough coefficient with 7/5 from
     theory.
      !WRITE(25,'(F11.5,3X,E13.7,3X,E13.7,3X,E13.7)') vz, dist_func(1,0,vz,y),
          dist_func(1,1,vz,y),dist_func(1,1,vz,y)/(-16.0d0/(3.0d0*sqrt(1/m)))
      !WRITE(40,'(F11.5,3X,E13.7)') vz,dist_func(1,1,vz,y)*(16.0d0*sqrt(m)/3.0d0)
      WRITE(28,'(F11.5,3X,E15.10,3X,E15.10,3X,E13.7,3X,E13.7,3X,E13.7,3X,E13.7)')
           vz, s_red_mass, 1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2,eq_smth1_rm,
          eq_smth2_rm,(1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2)/eq_smth1_rm,(1/(-m*
          sqrt(pi))/sqrt(exp(tmp1))*tmp2)/eq_smth2_rm
      WRITE(29,'(F11.5,3X,E15.10,3X,E15.10,3X,E13.7,3X,E13.7,3X,E13.7,3X,E13.7,3X
          ,E13.7,3X,E13.7)') vz, s_red_mass, 1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2
```

```
        ,eq_rgh1_rm,eq_rgh2_rm,(1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2)/
        eq_rgh1_rm,(1/(-m*sqrt(pi))/sqrt(exp(tmp1))*tmp2)/eq_rgh1a_rm,(1/(-m*
        sqrt(pi))/sqrt(exp(tmp1))*tmp2)/eq_rgh2_rm,(1/(-m*sqrt(pi))/sqrt(exp(
        tmp1))*tmp2)/eq_rgh2a_rm
  WRITE(30,'(F11.5,3X,E15.10,3X,E15.10,3X,E15.10)') vz,sqrt(m*(exp(tmp1))),
      eq_rgh1/eq_rgh2,eq_rgh1a/eq_rgh2a
END DO
CLOSE (10)
CLOSE (25)
CLOSE (40)
END PROGRAM ANALYZE
```

A.2 Subroutines

```
SUBROUTINE WIGNER_WILKINS
USE IONMOBILITY
USE MODULE_INTEGRATE
USE MODULE_CROSSEC
IMPLICIT NONE

!The subroutine calls on the ionmobility module, which describes some of the
    variables used. This subroutine is designed to be generic

CHARACTER:: jobvl,jobvr
LOGICAL,EXTERNAL:: select
REAL(hiprec) :: a_lim,b_lim,v,vz,temp,eval_gauss,z,temp1
!a_lim: lower limit of integration found using maximum values of interval
    given
!b_lim: higher limit of integration found using the minimum of values of
    interval given
!v: velocity
REAL(hiprec),DIMENSION (:), ALLOCATABLE ::y_deriv,ipiv,wl,wi
!y_deriv: array containing coefficients for bspline function
!ipiv: an array delivered upon output
REAL(hiprec),DIMENSION (:,:), ALLOCATABLE:: k_mat,b_til,b_til_values,nu,b_p,b,
    l_mat,vl,wr_new,vr_transpose,nu1,diff,l_mat1,l_mat1_safe,c
!k_mat: evaluates matrix K in evaluation of L matrix; this contains evaluation
     of wigner-wilkins kernel
!b_til: matrix used in the evaluation of the L matrix
!b_til_values: this matrix saves original b_til matrix, since inverse
    subroutine destroys original matrix, and we need the b_til matrix in its
    original form as well for evaluation of L matrix
!nu: one of the matrices used in evaluation of L matrix
!b: matrix used in evaluation of L matrix
!b_p: matrix used in evaluation of L matrix
INTEGER i,j,k,info,lwork,a,flag1,flag2,ldc,n
```

```
! i,j,k: integers used in the loops of the code,
! lwork is a parameter in the LAPACK routines, to set the size of the matrices
     being evaluated.
REAL(hiprec),DIMENSION(0:n_dim):: coeff
REAL(hiprec),EXTERNAL :: exe_func,int_interval,integrate,moments,dist_func
! exe_func: external function that has initial functions being used for the
    code
! integrate: external function that checks intervals for evaluation of
    integrands
! moments: external function that evaluates respective moments
! dsit_func: external function that evaluates the distribution function
LOGICAL :: done
DOUBLE PRECISION, EXTERNAL:: erf
!READ(67)
!RETURN

ALLOCATE(ipiv(0:n_dim))
ALLOCATE(y_deriv(0:n_dim))
ALLOCATE(k_mat(0:n_dim,0:n_dim))
!ALLOCATE(b_til(0:n_dim,0:n_dim))
!ALLOCATE(b_til_values(0:n_dim,0:n_dim))
ALLOCATE(nu(0:n_dim,0:n_dim))
ALLOCATE(b_p(0:n_dim,0:n_dim))
ALLOCATE(b(0:n_dim,0:n_dim))
ALLOCATE(l_mat(0:n_dim,0:n_dim))
ALLOCATE(l_mat_safe(0:n_dim,0:n_dim))

!above elements are due the mid-points used as two extra conditions needed for
     evaluation of coefficients
!b_til_values = b_til
!This forms b_tilda matrix used in evaluation of L matrix
!The values of elements are set to zero initially
!This is a band matrix. Diagonal elements in matrix have values 1,4,1
!The first and last rows of above matrix has 1,4,1 as first three and last
    three elements respectively

b_p = 0.0d0
b = 0.0d0

!$omp parallel shared(n_dim,dvz,b,b_p) private(i)
!$omp do schedule(auto)

DO i = 1,n_dim-1
    b_p(i,i-1) = -0.5d0/dvz
    b_p(i,i+1) = 0.5d0/dvz
    b(i,i-1) = 1.0d0/6.0d0
```

```
    b(i,i) = 4.0d0/6.0d0
    b(i,i+1) = 1.0d0/6.0d0
END DO

!$omp end do
!$omp end parallel

!WRITE(14,*) 'B_P'
!CALL L_INIT_VALUES(b_p)
!This is b_prime matrix used in evaluation of L matrix
!The values of the elements of this matrix are initially set to zero
!This is a band matrix, in which first and third element of band is -1.0 and
    1.0 respectively

!b = 0.0d0
!DO i = 1,n_dim-1
!    b(i,i-1) = 1.0d0/6.0d0
!    b(i,i) = 4.0d0/6.0d0
!    b(i,i+1) = 1.0d0/6.0d0
!END DO
b(0,0) = 1.0d0/48.0d0
b(0,3) = 1.0d0/48.0d0
b(0,1) = 23.0d0/48.0d0
b(0,2) = 23.0d0/48.0d0
b(n_dim,n_dim) = 1.0d0/48.0d0
b(n_dim,n_dim-3) = 1.0d0/48.0d0
b(n_dim,n_dim-1) = 23.0d0/48.0d0
b(n_dim,n_dim-2) = 23.0d0/48.0d0

!WRITE(14,*) 'B'
!CALL L_INIT_VALUES(b)
!This is a band matrix
!The elements in band are 1/6,4/6,1/6 respectively
!We decided to use mid-points between first two and last two points on
    velocity grid as two extra conditions needed for evaluation of time-
    dependent coefficients.
!This caused a need to modify b matrix, with above given elements.
!First four elements of first row and last four elements of last row are
    defined outside the loop as a special case.

!Unless specifically defined, the matrices above have zeroes everywhere

!This loop forms the K-matrix
!Elements in this matrix are zero everywhere except for element in centre of
    matrix
```

```
!This particular element is evaluted by the following loop, which also
    incorporates an external subroutine
!This external subroutine evaluates weights and points (w and x respectively)
    for evaluation of integral
!Integral is approximated by summation over product of weights and values of
    function at the provided points
!In K-matrix, the indices of the element in the centre are switched around as
    previously described
!The i (column) element is described as i = j-3 and the j (row) element is
    described as j = i-1
!This matrix also uses maximum and minimum values in the evalution of the
    bspline function
!l_matrix_ww: final matrix needed to continue rest of program evaluations
!l_matrix is a product of various matrices computed in this subroutine

k_mat = 0.0d0
!the k_mat matrix is zero everywhere, except for the elements chosen in the
    following loop
!n_gauss is number of points and weights used in evaluation of integral

!eval_gauss = int_interval(-1,0,0,1.0d0,0,0)
!I have chosen to evaluate point and weight only once in every loop and then
    used the assigned value in the kernel expression
!With above call, we evaluate gaussian points and weights for all values of
    n_gauss upto n_gauss_max, which we explicitly set. .
!With above call, points and weights are stored in an array-like fashion, and
    depending on the value of n_gauss, the appropriate set is called for
    evaluation of integrands.

!The entire integral is multiplied by a 1/(sqrt(x)) term. So when x=0, this
    term would cause entire integral to blow up, something we do not want.
!Thus we evaluated the integral just for x, using a Taylor Series expansion
    and derived above expression.

!$omp parallel shared(n_dim,vz_grid,dvz,vz_num,k_mat) private(i,j,z,a_lim,
    b_lim)
!$omp do schedule(auto)

DO j = 0,n_dim
!DO j = 100,100
!be careful to treat the mid-point values carefully...
!we don't want to integrate over the cusp at x=x'

   IF (j == 0) THEN
      z = 0.5d0*dvz
```

```
        k_mat(0,0) = integrate(2,-3,vz_grid(0),z,z,0.0d0) + integrate(2,-3,z,
            vz_grid(1),z,0.0d0)
        k_mat(0,1) = integrate(2,-2,vz_grid(0),z,z,0.0d0) + integrate(2,-2,z,
            vz_grid(2),z,0.0d0)
        k_mat(0,2) = integrate(2,-1,vz_grid(0),z,z,0.0d0) + integrate(2,-1,z,
            vz_grid(3),z,0.0d0)
        k_mat(0,3) = integrate(2,0,vz_grid(0),z,z,0.0d0) + integrate(2,0,z,
            vz_grid(4),z,0.0d0)
        DO i = 1,n_dim-3
!         DO i = 1,100
            a_lim = vz_grid(max(0,i))
            b_lim = vz_grid(min(vz_num,i+4))
            k_mat(0,i+3) = integrate(2,i,a_lim,b_lim,z,0.0d0)
            !WRITE(46,*)j,i,k_mat(0,i+3)
        END DO
    ELSE IF (j == n_dim)THEN
        z = vz_grid(vz_num)-0.5d0*dvz
        k_mat(n_dim,n_dim) = integrate(2,n_dim-3,vz_grid(n_dim-3),z,z,0.0d0) +
            integrate(2,n_dim-3,z,vz_grid(vz_num),z,0.0d0)
        k_mat(n_dim,n_dim-1) = integrate(2,n_dim-4,vz_grid(n_dim-4),z,z,0.0d0) +
             integrate(2,n_dim-4,z,vz_grid(vz_num),z,0.0d0)
        k_mat(n_dim,n_dim-2) = integrate(2,n_dim-5,vz_grid(n_dim-5),z,z,0.0d0) +
             integrate(2,n_dim-5,z,vz_grid(vz_num),z,0.0d0)
        k_mat(n_dim,n_dim-3) = integrate(2,n_dim-6,vz_grid(n_dim-6),z,z,0.0d0) +
             integrate(2,n_dim-6,z,vz_grid(vz_num),z,0.0d0)
        DO i = -3,n_dim-7
            a_lim = vz_grid(max(0,i))
            b_lim = vz_grid(min(vz_num,i+4))
            k_mat(n_dim,i+3) = integrate(2,i,a_lim,b_lim,z,0.0d0)
            !WRITE(46,*)j,i,k_mat(n_dim,i+3)
        END DO
    ELSE
        z = vz_grid(j-1)
        ! the value of z is what is what is passed in as the value of y into
            k_matrix
        DO i = -3,n_dim-3
            a_lim = vz_grid(max(0,i))
            b_lim = vz_grid(min(vz_num,i+4))
            k_mat(j,i+3) = integrate(2,i,a_lim,b_lim,z,0.0d0)
            !WRITE(46,*)j,i,k_mat(j,i+3)
        END DO
    END IF

END DO
!$omp end do
!$omp end parallel
```

```
WRITE(14,*) 'K_MATRIX'
CALL L_INIT_VALUES(k_mat)
!The indices used in the code (i,j) to construct our matrix loops, are
    different than the indices we have used in the notes
WRITE(14,*) 'NU'
nu = 0.0d0
!$omp parallel shared(n_dim,vz_grid,vz_num,nu,dvz) private(j,a_lim,b_lim,z,
    temp,temp1,done)
!$omp do schedule(auto)

DO j = 0, n_dim
   IF (j == 0) THEN
      Z = 0.5D0*dvz
   ELSE IF (j == n_dim) THEN
      z = vz_grid(vz_num)-0.5d0*dvz
   ELSE
      z = vz_grid(j-1)
   END IF
!integrate over whole grid starting at diagonal and working towards positive
    and negative direcrions, stopping when values are converged to a tolerance
    .
!step by 2 dvz to speed things up
   temp = 0.0d0
   IF (j == 1) THEN
      done = .TRUE.
   ELSE
      done = .FALSE.
   END IF
   b_lim = z
   DO WHILE(not(done))
      a_lim = MAX(vz_grid(0),b_lim-2.0d0*dvz)
      temp1 = int_interval(3,0,a_lim,b_lim,z,0.0d0)
      temp = temp + temp1
      b_lim = a_lim
      IF ((abs(temp1/temp) < 1.0d-13) .OR. (b_lim .LE. vz_grid(0))) done = .
          TRUE.
   END DO
   nu(j,j) = temp
   temp = 0.0d0
   IF (j == (n_dim-1)) THEN
      done = .TRUE.
   ELSE
      done = .FALSE.
   END IF
   a_lim = z
```

```
    DO WHILE(not(DONE))
        b_lim = MIN(vz_grid(vz_num),a_lim+2.0d0*dvz)
        temp1 = int_interval(3,0,a_lim,b_lim,z,0.0d0)
        temp = temp + temp1
        a_lim = b_lim
        IF((abs(temp1/temp) < 1.0d-13) .OR. (a_lim .GE. vz_grid(vz_num))) done =
            .TRUE.
    END DO
nu(j,j) = nu(j,j) + temp
END DO
!    nu(j,j) = integrate(3,0,vz_grid(0),vz_grid(vz_num),vz_grid(j-1),0.0d0)
   !WRITE(14,*) nu(j,j)
!END DO
!$omp end do
!$omp end parallel

!nu(0,0) = integrate(3,0,vz_grid(0),vz_grid(vz_num),0.5d0*dvz,0.0d0)
!nu(n_dim,n_dim) = integrate(3,0,vz_grid(0),vz_grid(vz_num),vz_grid(vz_num)
    -0.5d0*dvz,0.0d0)

DO j = 0,n_dim
    WRITE(14,*) nu(j,j)
END DO
!WRITE(14,*) nu(0,0)
!WRITE(14,*) nu(n_dim,n_dim)

!nu (frequency) matrix is evaluated using elements from k_matrix above.
!interval of integration is from 0 to S (finite interval), as opposed to 0 to
    infinity as in theory
!first and last values of frequency matrix are evaluated separately, and those
     values corresponding to midpoints on grid
!nu matrix is a diagonal matrix, with frequency values on diagonal, and these
    values are dependent on velocity grid, and zeroes everywhere else

!l1_mat = MATMUL(nu,b)
CALL DGEMM('N','N',n_dim+1,n_dim+1,n_dim+1,1.0d0,nu,n_dim+1,b,n_dim+1,0.0d0,
    l1_mat,n_dim+1)
WRITE(14,*) 'L1_MAT'
CALL L_INIT_VALUES(l1_mat)

l2_mat = k_mat-l1_mat
WRITE(14,*) 'L2_MAT'
CALL L_INIT_VALUES(l2_mat)
!l1_mat and l2_mat just separates components of defintion of collision
    operator (see comment below)
```

```
!the b_p matrix that requires extra multiplication with time-dependent
    external field.  So far we have not included any effects of field on the
    system

!l_mat = MATMUL(b_init,l2_mat)
CALL DGEMM('N','N',n_dim+1,n_dim+1,n_dim+1,1.0d0,b_init,n_dim+1,l2_mat,n_dim
    +1,0.0d0,l_mat,n_dim+1)
l_mat_safe = l_mat
OPEN(67,FORM = 'UNFORMATTED',FILE='L_MAT_SAFE')
    WRITE(67) l_mat
CLOSE(67)
WRITE(14,*) 'L_MAT'
CALL L_INIT_VALUES(l_mat)
!l_mat matrix is collision operator for which were are evaluating eigenvalues
!This is the final matrix that we need for calculation of distribution
    function
!In our notes collision matrix is defined (using the notations of the code) as
     L = inverse(b_til)*(k_mat-b_p-nu*b).
!All matrices are evaluated in this subroutines itself.
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!!DIAGONALIZATION

!ALLOCATE(vr(0:n_dim,0:n_dim))
!ALLOCATE(vl(0:n_dim,0:n_dim))
!ALLOCATE(vr_saved(0:n_dim,0:n_dim))
!ALLOCATE(wl(0:n_dim))
!ALLOCATE(wr(0:n_dim))
!ALLOCATE(wi(0:n_dim))

!LAPACK subroutine, evaluates right and left eigenvalues of given matrix.
!Evaluating eigenvalues of collision operator l_mat.
!For our particular matrix, we should obtain only right eigenvalues, and left
    eigenvalues should have a value of zero.

!CALL DGEEV('N','V',n_dim+1,l_mat,n_dim+1,wr,wi,vl,n_dim+1,vr,n_dim+1,work,-1,
    info)
!lwork = work(0)

!DEALLOCATE(work)
!ALLOCATE(work(1:lwork))
!CALL DGEEV('N','V',n_dim+1,l_mat,n_dim+1,wr,wi,vl,n_dim+1,vr,n_dim+1,work,
    lwork,info)

!Eigenvalues are scaled using following evaluation
!WRITE(*,*) 'REAL EIGENVALUES (NU): SCALED'
!WRITE(*,*) wr/(2.0d0*colfreq/sqrt(m))
```

```
!WRITE(*,*) 'REAL EIGENVALUES OF MAXWELL (NU): SCALED'
!WRITE(*,*) wr/(colfreq*sqrt(1.0d0+m)*sqrt(a4_tilda)/sqrt(m*pi)) ! maxwell
    scaled eigenvalues
!STOP
!WRITE(*,*) 'REAL EIGENVALUES (NU): NOT SCALED'
!WRITE(*,*) wr
!WRITE(*,*) 'IMAGINARY EIGENVALUES (NU1): SCALED'
!WRITE(*,*) 'IMAGINARY EIGENVALUES OF MAXWELL (NU1): SCALED'
!WRITE(*,*) wi/(colfreq*sqrt(1.0d0+m)*sqrt(a4_tilda)/sqrt(m*pi)) !maxwell
    imaginary scaled eigenvalues
!WRITE(*,*) wi/(2.0d0*colfreq/sqrt(m))
!WRITE(*,*) 'IMAGINARY EIGENVALUES (NU1): NOT SCALED'
!WRITE(*,*) wi

!vr are eigenvectors that are evaluated by subroutine DGEEV, for l_mat matrix.
!WRITE(14,*) 'vr'
!CALL L_INIT_VALUES(vr)
!vr_saved = vr

!CALL DGETRF(n_dim+1,n_dim+1,vr,n_dim+1,ipiv,info)
!IF (info /= 0) STOP "DGETRF: FAILED"
!CALL DGETRI(n_dim+1,vr,n_dim+1,ipiv,work,n_dim+1,info)
!IF (info /=0) STOP "DGETRI: FAILED"
!evaluates inverse of above tranformation matrix

!a_lim = vz_grid(0)
!b_lim = vz_grid(vz_num)
!   DO i = 0,n_dim
!      WRITE (33,"(A1,x,I3)") '&',i
!       DO vz = vz_0,vz_grid(vz_num),0.05
!           WRITE(33,"(F7.4,3X,E10.4,3X,E10.4,3X,E10.4,3X,E10.4,3X,F8.2,3X,F8
    .2,3X,E10.4,3X,E10.4,3X,F8.2)")vz,dist_func(1,0,vz,vr_saved(:,1)),
    dist_func(1,0,vz,vr_saved(:,2))!dist_func(2,1,vz,vr_saved(:,i))
!       END DO
!       a = 0
!       temp = integrate(1,0,a_lim,b_lim,1.0d0,vr_saved(:,i))
!       WRITE(34,*) i,temp
!   END DO

!WRITE(14,*) 'CHECK INVERSE OF VR'
!CALL L_INIT_VALUES(MATMUL(vr_saved,vr))
!CALL DGEMM('N','N',n_dim+1,n_dim+1,n_dim+1,1.0d0,vr_saved,n_dim+1,vr,n_dim
    +1,0.0d0,c,n_dim+1)

!END DIAGONALIZATION ROUTINE
```

```
!
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

!y1 = MATMUL(l_mat_safe,y)
! for runge-kutta
!WRITE(14,*) 'L_MAT & Y'
!WRITE(14,'(13(1pE9.2,X))')(y1(i)/sqrt(a_lim),i=0,n_dim)

!y_deriv = MATMUL(k_mat,y)
! for runge-kutta
!WRITE(14,*) 'K_MAT & Y'
!WRITE (14,'(13(1pE9.2,X))')(y_deriv(i),i=0,n_dim)

!l1_mat= MATMUL(nu,b)
!y1 = MATMUL(l1_mat,y)
! for runge-kutta
!WRITE(14,*) 'Y, B, NU'
!WRITE(14,'(13(1pE9.2,X))')(y1(i),i=0,n_dim)

!WRITE(14,*) 'nu'
!DO i = 0,100
   !WRITE(14,*) nu(i,i)
!END DO

END SUBROUTINE WIGNER_WILKINS

SUBROUTINE L_INIT_VALUES(x)
USE IONMOBILITY
IMPLICIT NONE

! this subroutine helps in printing out the values of the matrices that are
    being evaluated elsewhere.
! the format is set up to the proper matrix form, to make it easier to read
    the matrices

CHARACTER (len=10) f

REAL(hiprec), DIMENSION (0:n_dim,0:n_dim):: x

INTEGER :: i,j

DO i = 0,n_dim
WRITE(14,'(13(E12.6,X))')(x(i,j),j=0,n_dim)
END DO

END SUBROUTINE
```

```
SUBROUTINE GAUSS_WW (n_gauss, w_gauss, x_gauss, x1, x2)
!USE MODULE_INTEGRATE
IMPLICIT NONE

! This subroutine calls on the ionmobility module which
! describes some of the variables used
! This subroutine will evaluate the points x_gauss and
! the weights w_gauss
! These weights and points will be used the the
! subroutine l_init, to approximate an integral used in
! the evaluation of the element in the K matrix.

INTEGER,PARAMETER:: hiprec = KIND(0.0D0)

REAL(hiprec),DIMENSION (1:n_gauss) :: w_gauss, x_gauss
! w_gauss: the weights calculated in this subroutine
! x_gauss: the points calculated in this subroutine

REAL(hiprec) :: x1, x2, z1, p1, p2, pp, z, p3, max_val, min_val
! z1:
! p1: the desired Legendre polynomial
! p2: Legendre polynomial of one lower order
! pp: derivative of p1: Legendre polynomial
! z: calculation of the legendre polynomial at this particular value
! p3: polynomial that is two orders lower
! max_val: maximum value of integration
! min_val: minimum value of inegration

INTEGER i, j, n_gauss, mod_calc, m1
! n_gauss:the total number of points used in the summation of the integrand
! mod_calc: calculates the mod of n_gauss
! m1:sets a condition for the values of n_gauss to be used

REAL(hiprec), PARAMETER:: eps = 1.0D-15, pi = 3.1459265359d0

WRITE(15,*)"x1 = ",x1, "x2 = ",x2, "n_gauss = ", n_gauss

mod_calc = mod (n_gauss,2)

IF (mod_calc == 0)THEN
   m1 = n_gauss/2

ELSE
   m1 = (n_gauss+1)/2
END IF
```

```
max_val = 0.5d0*(x2+x1)
min_val = 0.5d0*(x2-x1)

DO i = 1,m1
    z = cos(pi*(REAL(i)-0.25d0)/(REAL(n_gauss)+0.5d0))
    z1 = 2.0d0
    DO WHILE (ABS(z-z1)>eps)
        p1 = 1.0d0
        p2 = 0.0d0
        DO j = 1, n_gauss
            p3=p2
            p2=p1
            p1 =((2.0d0*REAL(j)-1.0d0)*z*p2-(REAL(j)-1.0d0)*p3)/REAL(j)
        END DO
        pp = REAL(n_gauss)*(z*p1-p2)/(z*z-1.0d0)
        z1 = z
        z = z1-p1/pp
    END DO
    x_gauss(i) = max_val-min_val*z
    x_gauss(n_gauss+1-(i)) = max_val+min_val*z
    w_gauss(i)= 2.0d0*min_val/((1.0d0-z*z)*pp*pp)
    w_gauss(n_gauss+1-i)= w_gauss(i)

END DO
WRITE(15,*)"points"
WRITE(15,*)(x_gauss(j),j=1,n_gauss)
WRITE(15,*)"weights"
WRITE(15,*)(w_gauss(j),j=1,n_gauss)

END SUBROUTINE

SUBROUTINE POLINT(xa,ya,x,y,dy)
!USE nrutil, ONLY: assert_eq,iminloc,nerror
IMPLICIT NONE

!This subroutine evaluates the derivatives of the moments at the mid-point
    between the first two points on the grid and the mid point between the two
     last points on the grid.

! this is an interpolation to evaluate these two points.
! this subroutine is being used as a check to see if the results obtained are
    similar to those in the Andersen-Schuler (1964) paper.

INTEGER,PARAMETER:: I4B = SELECTED_INT_KIND(9), hiprec= KIND(0.0D0)
REAL(hiprec),DIMENSION(:),INTENT(in):: xa,ya
REAL(hiprec),INTENT(in):: x
REAL(hiprec),INTENT(out):: y,dy
```

```
!INTEGER(I4B),EXTERNAL:: assert_eq, iminloc
INTEGER(I4B):: m,n,ns,nstep
REAL(hiprec),DIMENSION(size(xa)):: c,d,den,ho

INTERFACE
        FUNCTION IMINLOC(arr)
        INTEGER,PARAMETER:: hiprec = KIND(0.0D0), I4B = SELECTED_INT_KIND(9)
        REAL(hiprec),DIMENSION(:),INTENT(in):: arr
        END FUNCTION
END INTERFACE

n = size(xa)
!n = assert_eq(size(xa),size(ya),'point')
c = ya
d = ya
ho = xa-x
ns = iminloc(ABS(x-xa))
y = ya(ns)
ns = ns-1
DO m = 1,n-1
    den(1:n-m) = ho(1:n-m)-ho(1+m:n)
    IF (ANY(den(1:n-m) == 0.0d0)) THEN
       CALL NERROR('point: calculation failure')
END IF
den(1:n-m) = (c(2:n-m+1)-d(1:n-m))/den(1:n-m)
       d(1:n-m) = ho(1+m:n)*den(1:n-m)
       c(1:n-m) = ho(1:n-m)*den(1:n-m)
       IF (2*ns < n-m) THEN
          dy = c(ns+1)
       ELSE
          dy = d(ns)
          ns = ns-1
       END IF
       y = y+dy
END DO
END SUBROUTINE POLINT

FUNCTION IMINLOC(arr)
INTEGER,PARAMETER::hiprec = KIND(0.0D0), I4B = SELECTED_INT_KIND(9)
REAL(hiprec),DIMENSION(:),INTENT(in):: arr
INTEGER(I4B),DIMENSION(1):: imin
INTEGER(I4B):: iminloc
imin=minloc(arr(:))
iminloc=imin(1)
END FUNCTION IMINLOC
```

```
FUNCTION ASSERT_EQ(nn,string)
  CHARACTER(len=*),INTENT(in) :: string
  INTEGER,DIMENSION(:),INTENT(in):: nn
  INTEGER :: assert_eq
  IF (all(nn(2:) == nn(1))) THEN
     assert_eqn = nn(1)
  ELSE
     WRITE(*,*) 'NERROR: an assert_eq failed with this tag:',string
     STOP 'program terminated by assert_eqn'
  END IF
END FUNCTION ASSERT_EQ

SUBROUTINE NERROR(string)
  CHARACTER(len=*),INTENT(in) :: string
  WRITE (*,*) 'nerror:', string
  STOP 'program terminated by nerror'
END SUBROUTINE NERROR

SUBROUTINE EVAL_CHI
USE MODULE_CROSSEC
IMPLICIT NONE

REAL(hiprec):: error, alpha, chi_val, theta_val
REAL(hiprec),EXTERNAL:: elliptic
INTEGER i,j,j_l,j_u

INTERFACE
     SUBROUTINE POLINT(xa,ya,x,y,dy)
     INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
     REAL(hiprec),DIMENSION(:):: xa,ya
     REAL(hiprec) :: x,y,dy
     END SUBROUTINE
END INTERFACE

ALLOCATE(theta(0:1000))
ALLOCATE(chi(0:1000))
ALLOCATE(cross_func(0:1000))

! first determine the mapping between chi and theta

OPEN (42, FILE = "chi_values")

chi = 0.0d0
theta = 0.0d0
i = 0
write(42,*) theta(0), chi(0)
DO theta_val = 0.001d0,pi/4.0d0,0.001d0
```

```
    i = i+1
    theta(i) = theta_val
    chi(i) = pi - 2.0d0*sqrt(cos(2.0d0*theta(i)))*elliptic(cos(theta(i)),1.0d0)
    write(42,*) theta(i), chi(i)
END DO
n_val = i+1
theta(n_val) = pi/4.0d0
chi(n_val) = pi
write(42,*) theta(n_val), chi(n_val)

CLOSE(42)

! now fill cross_func with values needed for the cross section calculation
! use a uniformly spaced chi grid for this

alpha = 20.00d0
n_chi = 100
dchi = pi/float(n_chi)
do i = 1, n_chi-1
    chi_val = float(i)*dchi
!
!  find chi value in chi array
!
    j = n_val
    DO WHILE (chi(j) > chi_val)
       j = j - 1
    END DO
    j_l = max(0,j-4)
    j_u = min(n_val,j+4)
!
!  interpolate value of theta from the chi and theta arrays
!
    CALL POLINT(chi(j_l:j_u),theta(j_l:j_u),chi_val,theta_val,error)
!
!  use theta and chi values to calculate crosssection function values
!
    cross_func(i) = sqrt(cos(2.0d0*theta_val))/sin(chi_val)/sin(2.0d0*theta_val
       )/( (cos(theta_val))**2*elliptic(cos(theta_val),1.0d0)-cos(2.0d0*
       theta_val)*elliptic(cos(theta_val),(cos(theta_val))**2) )
!
!  now add damping factor to make forward scattering cross section finite
!
!   write(58,*) chi_val/pi, cross_func(i), tanh(alpha*chi_val**2.5d0)*
    cross_func(i)
    write(58,*) chi_val, cross_func(i), tanh(alpha*chi_val**2.5d0)*cross_func(i
       )
```

```
    cross_func(i) = tanh(alpha*chi_val**2.5d0)*cross_func(i)
end do

!  add analytic values at endpoints

cross_func(0) = sqrt(3.0d0*pi)/4.0d0*alpha
cross_func(n_chi) = (1.0d0/elliptic(1/sqrt(2.0d0),1.0d0))**2

!write(58,*) pi/pi, cross_func(n_chi)
write(58,*) pi, cross_func(n_chi)
!write(58,*) 0.0d0/pi, cross_func(0)
write(58,*) 0.0d0, cross_func(0)

! now fill chi array with values matching those of cross_func

chi = 0.0d0
do i = 0, n_chi
    chi(i) = float(i)*dchi
end do

END SUBROUTINE EVAL_CHI
```

A.3 Functions

```
FUNCTION INTEGRATE(flag1,flag2,a,b,z,coeff)
USE IONMOBILITY
IMPLICIT NONE

!This function tests whether interval for integration is greater than one unit
     on the velocity grid.
!If so, loop breaks down interval, so that integration is done over 1 unit at
    a time.
!It then adds values of integrand, to give a final value for given limits of
    integration.
!We also have 0.5 intervals because of the special conditions we have added
    that involve the mid-points between 0 and 1 and  n_dim-1 and n_dim.
!For this case integration is carried out over half an interval within given
    limits, and then values are added to give a number for given limits of
    integration.

REAL(hiprec):: a,b,integrate,z,a1,b1
!a/b: lower/upper limits of integration interval that are passed from previous
     subroutine
!a1/b1: newer lower/upper limits of integration that are passed on to next
    function
LOGICAL done
```

```
INTEGER flag1,flag2,index
!index: calculates index of the point on left of the a on velocity grid
REAL(hiprec),DIMENSION(0:n_dim),INTENT(OUT):: coeff
REAL(hiprec), EXTERNAL:: int_interval

IF (a < vz_grid(0)) THEN
!checks to make sure that beginning on the interval for integration is not
    less than beginning of grid.
   WRITE(*,*) "ERROR: CHECK INTERVALS FOR INTEGRATION"
STOP
END IF
!returns error if above analysis fails

!if all is well with analysis done above, function continues in checking
    interval for integration
   index = INT((a-vz_grid(0))/dvz) + 1
   a1 = a
   b1 = vz_grid(index)
   IF (b1 > b) THEN
      b1 = b
      done = .TRUE.
   ELSE
      done = .FALSE.
   END IF
   integrate = int_interval(flag1,flag2,a1,b1,z,coeff)
   !write(32,*) 'saheba1',a1,b1,integrate
   a1 = b1
   DO WHILE(NOT(done))
      IF (b-a1 > dvz) THEN
         index = index+1
         b1 =  vz_grid(index)
!with following call to function integrate2, we are evaluating gaussian points
    and weights first for the set interval using values of a and b.
!this prevents gaussian points and weights to be re-evaluated repeatedly and
    can be use for entire loop as long as a and b do not change.

         integrate = integrate + int_interval(flag1,flag2,a1,b1,z,coeff)
         !WRITE(32,*)a1,b1,integrate
         a1 = b1
      ELSE
         b1 = b
         integrate = integrate + int_interval(flag1,flag2,a1,b1,z,coeff)
          !WRITE(32,*)'saheba',a1,b1,integrate
         done = .TRUE.
      END IF
   END DO
```

```
 END FUNCTION INTEGRATE

FUNCTION INT_INTERVAL(flag1,flag2,a,b,z,coeff)
USE IONMOBILITY
USE INTEGRATE_GAULAG
USE MODULE_INTEGRATE
IMPLICIT NONE

!this subroutine has different scenarios where  kernel is evaluated in
!fspace: f(x) = 1
!gspace: f(x) = 2*sqrt(pi/x)*exp(-x)*g(x); where g(x) = 1
!hspace: f(x) = 2*sqrt(pi/x)*exp(-h(x)); where h(x) = x

REAL(hiprec):: a,b,wgt,pt,j,int_interval,integrand,integrand_last,z,moment_val
    ,moment_val_last,x,k_mat_y
!a: upper limit of integration
!b: lower limit of integration
REAL(hiprec), EXTERNAL:: k_matrix,dist_func,bspline3
!bspline3: external function that contains the expressions for the bspline
    functions, their first and second derivatives
REAL(hiprec),DIMENSION(0:n_dim),INTENT(OUT):: coeff
INTEGER flag1,flag2,k,i,flag3,n,n_max_gauss_check
LOGICAL done

SELECT CASE(flag1)
CASE (-1)
!evaluates gaussian points and weights for given limits of integral.
!we avoid re-evaluation of points and weight for same interval.
   !a = 0.0d0
   !b = 1.0d0
!above values for a and b are explicitly defined, to ensure that points and
    weights are evaluated for a generic interval of 0 to 1
!these are then scaled later on depending on interval we want to evaluate the
    integrands over
   DO n = 1, n_max_gauss
      npts_gauss(n) = 2**(n+2)
      CALL GAUSS_WW(npts_gauss(n),w_gauss(n,:),x_gauss(n,:),a,b)
   END DO
CASE(-2)
   DO n = 1,n_max_gaulag
      npts_gaulag(n) = 2**(n+2)
      CALL GAULAG(x_gaulag(n,:),w_gaulag(n,:),0.0d0,npts_gaulag(n))
   END DO


CASE(1)
!MOMENT CALCULATION
```

```
!flag2 = order of moment
!moment calculation in fspace
IF (spc == 1) THEN
   flag3 = 1
ELSE IF (spc == 2)THEN
   flag3 = 2
   IF (z < 0.0d0) THEN
      flag3 = 1
!negative z means calculate moment in native space
   END IF
ELSE IF (spc == 3) THEN
   flag3 = 3
   IF (z < 0.0d0) THEN
      flag3 = 1
   END IF
ELSE IF (spc == 4) THEN
   flag3 = 4
   IF (z < 0.0d0) THEN
      flag3 = 1
   END IF
ELSE IF (spc == 5 .OR. spc == 6 .OR. spc == 7 .OR. spc == 10 .OR. spc == 11)
   THEN
   flag3 = 5
   IF (z < 0.0d0) THEN
      flag3 = 1
   END IF
END IF
   moment_val_last = 0.0d0
   DO n = 1,n_max_gauss
      moment_val = 0.0d0
      DO k = 1,npts_gauss(n)
         pt = x_gauss(n,k)*(b-a)+a
         wgt = w_gauss(n,k)*(b-a)
!points and weights are scaled depending on interval of integral evaluation
         moment_val = moment_val + wgt*(pt**flag2)*dist_func(flag3,0,pt,coeff)
!evaluate moments at different points as evaluated by gauss-legendre
    subroutine
      END DO
      !WRITE(18,*) a,b,npts_gauss(n),moment_val
      if( ( abs(( moment_val - moment_val_last )/moment_val ) < 1.0d-13 ) .or.
           ( abs( moment_val - moment_val_last ) < 1.0d-14 ) ) exit

!this test checks the convergence of the integrands.
!we can set tolerance to any desired value.  So far tolerance is sufficient
    for double precision values.
      moment_val_last = moment_val
```

```
        !WRITE(18,*) n,moment_val
    END DO
    int_interval = moment_val
    IF (n == n_max_gauss+1) THEN
        !WRITE(19,*) "Value not converged", "flag1 = ",flag1,"i = ",i,"a = ",a,"
            b = ",b, "j = ",j
!a little test does show that once the loop is complete, we would have
    n_max_gauss+1 as the last value, hence we can use the above test to check
    whether converged value is appropriate
    END IF
    !WRITE (18,"(A1)")'&'

CASE(2)
!evaluation of integrals in f-space, when spc (defined in the input) equals 1
!g-space integration of kernel when spc = 2
!integration in both cases is done with respect to variable x

    IF (spc == 3) THEN
        WRITE(*,*) "ERROR: WE ARE IN CASE 2 IN INT_INTERVAL!"
        STOP
    END IF

    !IF (spc == 1 )THEN
    !    flag3 = 1
    !ELSE IF (spc == 2) THEN
    !    flag3 = 2
    !ELSE IF (spc == 4) THEN
    !    flag3 = 4
    !ELSE IF (spc == 5) THEN
    !    flag3 = 5
    !ELSE IF (spc == 6) THEN
    !    flag3 = 6
    !ELSE IF (spc == 7) THEN
    !    flag3 = 7
    !ELSE IF (spc == 9) THEN
    !    flag3 = 9
    !ELSE IF (spc == 10) THEN
    !    flag3 = 10
    !END IF
        integrand_last = 0.0d0
        !IF (z < 1.0d-14) THEN
        !    n_max_gauss_check = n_max_gauss
        !ELSE
        !    n_max_gauss_check = 4
        !END IF
        DO n = 1, n_max_gauss
```

```
        !DO n = 1,n_max_gauss_check
           integrand = 0.0d0
           DO k = 1,npts_gauss(n)
              wgt = w_gauss(n,k)*(b-a)
              pt = x_gauss(n,k)*(b-a)+a
              x = (pt - vz_grid(0) )/dvz - REAL(flag2)
!flag2 carries the index of the spline
              integrand = integrand + wgt*k_matrix(spc,pt,z,coeff)*bspline3(0,x)
              !WRITE(38,*) pt,wgt,z,integrand
           END DO
!          WRITE(19,*) npts_gauss(n),integrand!,"gauss"
            !WRITE(19,'(f8.3,3X,F8.3,3X,I3,3X,E15.9)') a,b,n,integrand
           IF( ( abs(( integrand - integrand_last )/integrand ) < 1.0d-9 ) .OR.
               ( abs( integrand - integrand_last ) < 1.0d-14 )) EXIT
           integrand_last = integrand
        END DO
        int_interval = integrand
        IF (n == n_max_gauss+1) THEN
        !WRITE(19,*) "Value not converged", "spline index = ",flag2, "a = ",a,"b
            = ",b, "y = ",z
        END IF

CASE (3)
!this case is for integration of just the kernel by itself, with respect to y
!no bspline functions are used in integration, and points and weights of gauss
    -legendre quadrature are used.
!IF (spc == 1) THEN
IF (spc == 1 .OR. spc ==2 .OR. spc == 3 .OR. spc ==4) THEN
    flag3 = 1
ELSE IF (spc == 5 .OR. spc == 6 .OR. spc == 7) THEN
    flag3 = 8
ELSE IF (spc == 9 .OR. spc == 10 .OR. spc == 11) THEN
    flag3 = 9
END IF
    integrand_last = 0.0d0
    !IF (z < 1.0d-14) THEN
    !   n_max_gauss_check = n_max_gauss
    !ELSE
    !   n_max_gauss_check = 4
    !END IF
   DO n = 1, n_max_gauss ! uses the full 512 points in the integration..set in
        module_integrate
     !DO n = 1,n_max_gauss_check
        integrand = 0.0d0
        DO k = 1,npts_gauss(n)
           wgt = w_gauss(n,k)*(b-a)
```

```
            pt = x_gauss(n,k)*(b-a)+a
            integrand = integrand + wgt*k_matrix(flag3,z,pt,coeff)
!this integration for nu matrix is done using first case in k_matrix
!make the change for the integrand call using the spc == 5 and set the flag to
    a variable.
        END DO
        !WRITE(19,"(i5,1x,1pd20.12,1x,f10.4,1x,f10.4)") npts_gauss(n),integrand,
            a, b
        IF( ( abs(( integrand - integrand_last )/integrand ) < 1.0d-9 ) .OR. (
            abs( integrand - integrand_last ) < 1.0d-14 )) EXIT
        integrand_last = integrand
    END DO
    int_interval = integrand
    IF (n == n_max_gauss+1) THEN
        !WRITE(19,*) "Value not converged", "flag1 = ",flag1,"i = ",i,"a = ",a,"
            b = ",b, "j = ",j
    END IF

CASE(4)
!this is the evaluation of integrals in the h-space
integrand_last = 0.0d0
DO n = 1,n_max_gauss
    integrand = 0.0d0
    DO k = 1,npts_gauss(n)
        wgt = w_gauss(n,k)*(b-a)
        pt = x_gauss(n,k)*(b-a)+a
        integrand = integrand + wgt*k_matrix(3,pt,z,coeff)
    END DO
    !WRITE(19,*) npts_gauss(n),integrand
    IF( ( abs(( integrand - integrand_last )/integrand) < 1.0d-13 ) .OR. ( abs(
        integrand - integrand_last ) < 1.0d-14 )) EXIT
    integrand_last = integrand
    END DO
    int_interval = integrand
    IF (n == n_max_gauss+1) THEN
        !WRITE(19,*)"Value not converged", "flag1 = ",flag1,"i = ",i,"a = ",a,"b
            = ",b,"j = ",j
    END IF
END SELECT
END FUNCTION INT_INTERVAL

RECURSIVE FUNCTION K_MATRIX(a,x,x1,coeff) RESULT(k_matrix_calc)
!FUNCTION K_MATRIX(a,x,y,coeff)
USE IONMOBILITY
USE MODULE_INTEGRATE
IMPLICIT NONE
```

```
!This is a function, that has the different expressions for the 4 different
    integrals we are using, that make up the Wigner Wilkins kernel.
! Case 1 is the form of the expression for the initial function, g(x)=1
! Case 2 is the form of the expression for the initial function, g(x)=exp(-x)

! All vales are in double precision
REAL(hiprec):: pt,wgt,j,k_matrix1,k_matrix2,x,a_lim,b_lim,add,diff,constant,b,
    integrand,integrand_last,vz_n,yval,xval,f11,f22,f12,f21,constant1,
    constant2,deltax,k_matrix_calc,x1

!REAL(hiprec):: k_matrix7

REAL(hiprec),DIMENSION(0:n_dim):: coeff

INTEGER i,a,k,n
! integer a selects the case according to the initial function
! integer n selects the integrand expression that needs to evaluated

REAL(hiprec), EXTERNAL:: bspline3
! This is an external function that the different expressions for the third
    order bspline funcyion.

DOUBLE PRECISION, EXTERNAL:: erf,dist_func,integralw,bilinear,F_2

SELECT CASE (a)
CASE(1)
! f-space kernel
   IF (ABS(x) < 1.0d-12) THEN
      k_matrix_calc = 2.0d0*colfreq*q*q/sqrt(m)*exp( -q*q*x1 )
! limit of x going to zero
   ELSE
      k_matrix1 = erf( q*sqrt(x1) + r*sqrt(x) ) + exp(x-x1)*erf( r*sqrt(x1) +
          q*sqrt(x) )

      k_matrix2 = erf( q*sqrt(x1) - r*sqrt(x) ) + exp(x-x1)*erf( r*sqrt(x1) -
          q*sqrt(x) )

      IF (x > x1) THEN
         k_matrix_calc = colfreq*q*q/2.0d0*sqrt(pi/x)*&
              (k_matrix1 + k_matrix2)
      ELSE
         k_matrix_calc = colfreq*q*q/2.0d0*sqrt(pi/x)*&
              (k_matrix1 - k_matrix2)

      END IF
   END IF
```

```
CASE(2)
! g-space kernel
   IF(ABS(x1) < 1.0d-12) THEN
      k_matrix_calc = 2.0d0*colfreq*q*q/sqrt(m)*exp( -q*q*x )
   ELSE
      k_matrix1 = exp(x1-x)*( erf( q*sqrt(x1) + r*sqrt(x) ) ) + erf( r*sqrt(x1
          ) + q*sqrt(x) )

      k_matrix2 = exp(x1-x)*( erf( q*sqrt(x1) - r*sqrt(x) ) ) + erf( r*sqrt(x1
          ) - q*sqrt(x) )

      IF (x > x1 ) THEN
         k_matrix_calc = colfreq*q*q/2.0d0*sqrt(pi/x1)*&
              (k_matrix1 + k_matrix2)
      ELSE
         k_matrix_calc = colfreq*q*q/2.0d0*sqrt(pi/x1)*&
              (k_matrix1 - k_matrix2)
      END IF
   END IF

CASE(3)
! h-space kernel
   IF(ABS(x1) < 1.0d-12) THEN
         k_matrix_calc = 2.0d0*colfreq*q*q/sqrt(m)*(exp(dist_func(1,0,0.0d0,
             coeff)-dist_func(1,0,x,coeff) - r*r*x) )
   ELSE
         k_matrix1 = exp(dist_func(1,0,x1,coeff)-dist_func(1,0,x,coeff) )*&
                     (&
                     erf( q*sqrt(x1) + r*sqrt(x) ) + exp( x-x1 )*erf( r*sqrt(
                         x1) + q*sqrt(x) )&
                     )

         k_matrix2 = exp(dist_func(1,0,x1,coeff)-dist_func(1,0,x,coeff) )*&
                     (&
                      erf(q*sqrt(x1) - r*sqrt(x) ) + exp(x-x1)*erf(r*sqrt(x1)
                          - q*sqrt(x) )&
                      )
      IF (x > x1) THEN
         k_matrix_calc = colfreq*q*q*0.5d0*sqrt(pi/x1)*&
                    (k_matrix1 + k_matrix2)
      ELSE
         k_matrix_calc = colfreq*q*q*0.5d0*sqrt(pi/x1)*&
                    (k_matrix1 - k_matrix2)
      END IF
   END IF
```

```
CASE(4)
   IF (ABS(x1) < 1.0d-12) THEN
          k_matrix_calc = 2.0d0*colfreq*q*q/sqrt(m)*exp(-r*r*x)
   ELSE
          k_matrix1 = erf( q*sqrt(x1) + r*sqrt(x) ) + exp(x-x1)*erf( r*sqrt(x1
              ) + q*sqrt(x) )
          k_matrix2 = erf( q*sqrt(x1) - r*sqrt(x) ) + exp(x-x1)*erf( r*sqrt(x1
              ) - q*sqrt(x) )
          IF (x > x1) THEN
             k_matrix_calc = colfreq*q*q*0.5d0*sqrt(pi/x1)*&
                    (k_matrix1 + k_matrix2)
          ELSE
             k_matrix_calc = colfreq*q*q*0.5d0*sqrt(pi/x1)*&
                    (k_matrix1 - k_matrix2)
          END IF
  END IF

CASE(5)
!the general case kernel
   IF (ABS(x) < 1.0d-12) THEN
      k_matrix_calc = 0.0d0
   ELSE IF (ABS(x1) < 1.0d-12) THEN
      k_matrix_calc = (2.0d0*colfreq*q*q/sqrt(m))*exp(-r*r*x)*integralw(0.0d0,
          x,0.0d0)
   ELSE
      integrand_last = 0.0d0
      constant = (colfreq*q*q/sqrt(m*x1))*exp((q-r)*(r*x-q*x1))
      a_lim = ABS(sqrt(x)-sqrt(x1))
      b_lim = sqrt(x)+sqrt(x1)
      !DO n = 4,4
     DO n = 1, n_max_gauss
        integrand = 0.0d0
        DO k = 1, npts_gauss(n)
           wgt = w_gauss(n,k)*(b_lim-a_lim)
           pt = x_gauss(n,k)*(b_lim-a_lim)+a_lim
           integrand = integrand + wgt*constant*exp(-q*r*pt*pt)*integralw(pt,
               x,x1)
        END DO
        if( (n > 1) .AND. (( abs(( integrand - integrand_last )/integrand ) <
            1.0d-10 ) .or. ( abs( integrand - integrand_last ) < 1.0d-14 )))
            exit

        integrand_last = integrand
     !WRITE(57,'(E15.9,3X,E15.9,3X,I3,3X,E15.9)') x,x1,n,integrand
     END DO
     k_matrix_calc = integrand
```

```
    !    stop
        IF (n == n_max_gauss + 1) THEN
!           WRITE(19,*) "K_MAT CASE: 5 VALUE NOT CONVERGED"
        END IF
    END IF

CASE(6)
! evaluation of the kmatrix for the Maxwell molecule and the general case
    smooth sphere
! can also be evaluated using case 5 above, but it would take a lot longer to
    run the calculations
! spc = 6
! these arrays are the ones evaluated by the k_mat_calc program, and creates
    the arrays of arbitrary values
! these arrays are used to interpolate the value to be used in the calculation
     of the integrands

    i = INT((x-kmat_vz_0)/kmat_dvz)
    k = INT((x1-kmat_vz_0)/kmat_dvz)
    xval = x - kmat_x_val(i)
    yval = x1 - kmat_y_val(k)
IF (i < 0) THEN
    i = 0
END IF
IF (i >= kmat_vz_num) THEN
    i = kmat_vz_num - 1
END IF
IF (k < 0) THEN
    k = 0
END IF
IF (k >= kmat_vz_num) THEN
    k = kmat_vz_num-1
END IF
! the above indices are evaluated based on the values of x and y that are
    passed in, to identify which elements in the arrays should be used for
    interpolation

    constant = (colfreq*q*q/sqrt(m*x1))*exp((q-r)*(r*x-q*x1))
    constant1 = log(constant)

    IF ( i == k) THEN

! interpolation above or below the diagonal but not crossing the diagonal due
    the presence of the cusp
! uses 3 points in 2D for interpolation
```

```
      IF (i == 0 .AND. yval > xval) THEN

         constant = (colfreq*q*q/sqrt(m*kmat_y_val(1)))*exp((q-r)*(r*x-q*
             kmat_y_val(1)))
         constant1 = log(constant)
! triangle below the diagonal
         f11 = 0.0d0
         f12 = k_mat_val(i,k+1)
         f21 = 0.0d0
         f22 = constant*k_mat_val(i+1,k+1)
         k_matrix_calc = exp(bilinear(1,xval,yval,f11,f12,f21,f22))

      ELSE IF (i == 0 .AND. xval >= yval) THEN

         constant = (colfreq*q*q/sqrt(m*kmat_y_val(1)))*exp((q-r)*(r*x-q*
             kmat_y_val(1)))
         constant1 = log(constant)

         f11 = k_mat_val(i,k)
         f12 = k_mat_val(i+1,k)
         f21 = 0.0d0
         f22 = constant*exp(k_mat_val(i+1,k+1))
         k_matrix_calc = bilinear(1,xval,yval,f11,f12,f21,f22)

      ELSE

! case for all other points along the diagonal
         IF (xval >= yval) THEN
            f11 = k_mat_val(i,k)
            f12 = k_mat_val(i+1,k)
            f21 = 0.0d0
            f22 = k_mat_val(i+1,k+1)

         ELSE

            f11 = k_mat_val(i+1,k+1)
            f12 = k_mat_val(i,k)
            f21 = 0.0d0
            f22 = k_mat_val(i,k+1)
! this is an exception for the notation beacuse of the variables used in the
    formula in bilinear.f90
! the value of f21 is what is used in the interpolation
         END IF

         k_matrix_calc = constant1+bilinear(1,xval,yval,f11,f12,f21,f22)
         k_matrix_calc = exp(k_matrix_calc)
```

```
      END IF
! normal bilinear interpolation in 2D using 4 points
   ELSE

      IF (k == 0) THEN
! special case for the first row of the matrix that is evaluated when the
    value of y is 0
! therefore the first row in the matrix is evaluated with a limiting case and
    therefore needs to be dealt with a special case for interpolation

         constant = (colfreq*q*q/sqrt(m*kmat_y_val(1)))*exp((q-r)*(r*x-q*
             kmat_y_val(1)))
         constant1 = log(constant)
         f11 = k_mat_val(i,0)
         f12 = constant*(exp(k_mat_val(i,1)))
         f21 = k_mat_val(i+1,0)
         f22 = constant*(exp(k_mat_val(i+1,1)))
         k_matrix_calc = bilinear(2,xval,yval,f11,f12,f21,f22)

      ELSE IF (i == 0) THEN
! special case for the limit of x approaching 0, therefore in the k_mat_val
    matrix this corresponds to the first column that is evaluated to be all
    zeroes

         constant = (colfreq*q*q/sqrt(m*x1))*exp((q-r)*(r*kmat_x_val(1)-q*x1))
         f11 = k_mat_val(i,k)
         f12 = k_mat_val(i,k+1)
         f21 = exp(k_mat_val(i+1,k))
         f22 = exp(k_mat_val(i+1,k+1))
         k_matrix_calc = constant*bilinear(2,xval,yval,f11,f12,f21,f22)

      ELSE

! for all other points and elements in the k_mat_val matrix
         f11 = k_mat_val(i,k)
         f12 = k_mat_val(i,k+1)
         f21 = k_mat_val(i+1,k)
         f22 = k_mat_val(i+1,k+1)
         k_matrix_calc = constant1+bilinear(2,xval,yval,f11,f12,f21,f22)
         !k_matrix = constant*bilinear(2,xval,yval,f11,f12,f21,f22)
         k_matrix_calc = exp(k_matrix_calc)
      END IF
   END IF

!    WRITE(56,*) x,x1,f11,f12,f21,f22,k_matrix_calc
```

```
    !WRITE(56,*) constant*exp(f11),constant*exp(f12), constant*exp(f21),
        constant*exp(f22)
    !WRITE(56,*) xval, yval, k_matrix

!stop

CASE(7)
! this is a hybrid case for evaluating the k_matrix
! this will call case 5 or case 6 depending on the values of x and y
! it will evaluate values around the diagonal of the matrix using case 5 and
    the rest using case 6
    i = INT((x-kmat_vz_0)/kmat_dvz)
    k = INT((x1-kmat_vz_0)/kmat_dvz)
IF (i < 0) THEN
    i = 0
END IF
IF (i >= kmat_vz_num) THEN
    i = kmat_vz_num-1
END IF
IF (k < 0) THEN
    k = 0
END IF
IF (k >= kmat_vz_num) THEN
    k = kmat_vz_num-1
END IF

    IF (abs(k-i) < 10) THEN
! evaluation of the k_matrix within the band around the diagonal
! of course once again only
        k_matrix_calc = k_matrix(5,x,x1,coeff)
    ELSE
! evaluation of the k_matrix everywhere else in the matrix
        k_matrix_calc = k_matrix(6,x,x1,coeff)
END IF
!k_matrix1 = k_matrix

CASE(8)
! f-space formulation for the general kernel
!calculation of the nu matrix for spaces 5,6,7
! uses the general kernel with the sqrt(x) factor included in it
! integration is done over the variable y
! x and y variable are switched around
! this will use case 2 in integralw for the frequency calculation

IF (ABS(x) < 1.0d-12) THEN
```

```
        k_matrix_calc =  (2.0d0*colfreq*q*q/sqrt(m))*exp(-q*q*x1)*integralw(0.0
            d0,x1,0.0d0)
    ELSE IF (ABS(x1) < 1.0d-12) THEN
        k_matrix_calc = 0.0d0
    ELSE
        integrand_last = 0.0d0
        constant = (colfreq*q*q/sqrt(x*m))*exp((q-r)*(r*x-q*x1))
        a_lim = ABS(sqrt(x)-sqrt(x1))
        b_lim = sqrt(x)+sqrt(x1)
        !DO n = 4,4
        DO n = 1, n_max_gauss
          integrand = 0.0d0
          DO k = 1, npts_gauss(n)
            wgt = w_gauss(n,k)*(b_lim-a_lim)
            pt = x_gauss(n,k)*(b_lim-a_lim)+a_lim
            integrand = integrand + wgt*constant*exp(-q*r*pt*pt)*integralw(pt,
                x,x1)
          END DO
          if((n > 1) .and. (( abs(( integrand - integrand_last )/integrand ) <
              1.0d-13 ) .or. ( abs( integrand - integrand_last ) < 1.0d-14 )))
              exit
          integrand_last = integrand
        END DO
        k_matrix_calc = integrand
!       WRITE(57,*) x,x1,k_matrix
    !   stop
        IF (n == n_max_gauss + 1) THEN
!         WRITE(19,*) "K_MAT CASE: 5 VALUE NOT CONVERGED"
        END IF
    END IF

CASE(9)
!rough hard sphere approximate kernel - f-space representation
    !constant1 = 1.5 !\mu\chi value
    IF (ABS(x1) < 1.0d-12) THEN
        k_matrix_calc = 0.0d0
    ELSE IF (ABS(x) < 1.0d-12) THEN
!       k_matrix = (2.0d0*colfreq*q*q/sqrt(m))*exp(-q*q*x1)*F_2(0.0d0,y,sqrt(x1
    ),m,mu_chi)
        k_matrix_calc = (2.0d0*colfreq*q*q/sqrt(m))*exp(-q*q*x1)*((1.0d0+mu_chi)
            **2)/mu_chi/(1.0d0+mu_chi-m)*(1.0d0 - exp(-mu_chi*q*q*(1.0d0+mu_chi-
            m)*x1/(1.0d0+(1.0d0+m)*mu_chi)))
    ELSE
        integrand_last = 0.0d0
        constant= colfreq*q*q/sqrt(m*x)*exp((q-r)*(r*x-q*x1))
        a_lim = ABS(sqrt(x)-sqrt(x1))
```

```
        b_lim = sqrt(x)+sqrt(x1)
        DO n = 4,4
            !DO n = 1,n_max_gauss
            integrand = 0.0d0
            DO k = 1, npts_gauss(n)
                wgt = w_gauss(n,k)*(b_lim-a_lim)
                pt = x_gauss(n,k)*(b_lim-a_lim)+a_lim
                integrand = integrand +wgt*exp(-q*r*pt*pt)*F_2(x,x1,pt,m,mu_chi)
            END DO
            if( ( abs(( integrand - integrand_last )/integrand ) < 1.0d-13 ) .or.
                ( abs( integrand - integrand_last ) < 1.0d-14 )) exit
            integrand_last = integrand
        END DO
        k_matrix_calc = constant*integrand
        !WRITE(57,*) x,y,k_matrix
        IF (n == n_max_gauss + 1) THEN
!           WRITE(19,*) "K_MAT CASE: 9 VALUE NOT CONVERGED"
        END IF
    END IF

CASE (10)
!rough hard sphere approximate kernel - g-space representation
!    constant1 = 1.5 !\mu\chi value
    IF (ABS(x) < 1.0d-12) THEN
        k_matrix_calc = 0.0d0
    ELSE IF (ABS(x1) < 1.0d-12) THEN
!
!   careful here with this one....
!
!       k_matrix = (2.0d0*colfreq*q*q/sqrt(m))*exp(-r*r*x)*F_2(x,0.0d0,sqrt(x),m
    ,mu_chi)
        k_matrix_calc = (2.0d0*colfreq*q*q/sqrt(m))*exp(-r*r*x)*(1.0d0+mu_chi)
            **2/mu_chi/(1.0d0+mu_chi-m)*(1.0d0 - exp(-mu_chi*(1.0d0+mu_chi-m)*q*
            q*x/(1.0d0+(1.0d0+m)*mu_chi)))

    ELSE
        integrand_last = 0.0d0
        constant = colfreq*q*q/sqrt(m*x1)*exp((q-r)*(r*x-q*x1))
        a_lim = ABS(sqrt(x)-sqrt(x1))
        b_lim = sqrt(x)+sqrt(x1)
        DO n = 4,4
            !DO n = 1, n_max_gauss
            integrand = 0.0d0
            DO k = 1, npts_gauss(n)
                wgt = w_gauss(n,k)*(b_lim-a_lim)
                pt = x_gauss(n,k)*(b_lim-a_lim)+a_lim
```

```
            integrand = integrand + wgt*exp(-q*r*pt*pt)*F_2(x,x1,pt,m,mu_chi)
          END DO
          if( ( abs(( integrand - integrand_last )/integrand ) < 1.0d-10 ) .or. (
               abs( integrand - integrand_last ) < 1.0d-14 ))  exit
          integrand_last = integrand
        END DO
        k_matrix_calc = constant*integrand
        !WRITE(57,*) x,x1,k_matrix
        IF (n == n_max_gauss + 1) THEN
!           WRITE(19,*) "K_MAT CASE: 10 VALUE NOT CONVERGED"
        END IF
      END IF

CASE(11)
! this is a hybrid case for evaluating the k_matrix
! this will call case 10 or case 6 depending on the values of x and x1
! it will evaluate values around the diagonal of the matrix using case 10 and
    the rest using case 6
      i = INT((x-kmat_vz_0)/kmat_dvz)
      k = INT((x1-kmat_vz_0)/kmat_dvz)
IF (i < 0) THEN
      i = 0
END IF
IF (i >= kmat_vz_num) THEN
      i = kmat_vz_num-1
END IF
IF (k < 0) THEN
      k = 0
END IF
IF (k >= kmat_vz_num) THEN
      k = kmat_vz_num-1
END IF
      IF (abs(k-i) < 10) THEN
! evaluation of the k_matrix within the band around the diagonal
! of course once again only
        k_matrix_calc = k_matrix(10,x,x1,coeff)
      ELSE
! evaluation of the k_matrix everywhere else in the matrix
        k_matrix_calc = k_matrix(6,x,x1,coeff)
END IF
!k_matrix1 = k_matrix

END SELECT

END FUNCTION K_MATRIX

FUNCTION CROSSEC(n,g,chi_val)
```

```
USE IONMOBILITY
USE MODULE_CROSSEC
IMPLICIT NONE

REAL(hiprec):: g,crossec,constant,error, chi_val

INTEGER i,n,i_u,i_l

REAL(hiprec),EXTERNAL::elliptic

INTERFACE
    SUBROUTINE POLINT(xa,ya,x,y,dy)
    INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
    REAL(hiprec),DIMENSION(:):: xa,ya
    REAL(hiprec) :: x,y,dy
    END SUBROUTINE
END INTERFACE

SELECT CASE(n)

CASE(1)
! cross section for the smooth sphere case

crossec = 1.0d0

CASE(2)

!  constant includes the factor of sqrt(2kT/m_1) from the general call

constant = 2.0d0*sqrt(a4_tilda*(1.0d0+m))/g
!constant = 1.0d0

!  locate grid points around chi value of interest and use polint to
!  interpolate cross_func value to get final cross section

i = int(chi_val/dchi)
i_l = max(0,i-4)
i_u = min(n_chi,i+4)

CALL POLINT(chi(i_l:i_u),cross_func(i_l:i_u),chi_val,crossec,error)

crossec = constant*crossec

END SELECT

END FUNCTION
```

```
FUNCTION INTEGRALW(a,x,x1)
USE IONMOBILITY
USE MODULE_INTEGRATE
USE INTEGRATE_GAULAG
IMPLICIT NONE

!following loop initializes gauss laguerre subroutine that would calculate the
     points and weights for the quadrature
!these are evaluaed and stored in a matrix

REAL(hiprec) :: a,x,integrandw,integrandw_last,integralw,pt,wgt,constant,
    sigma12,numdense,a_lim,b_lim,x1,vz_n,constant1
REAL(hiprec), EXTERNAL:: crossec
DOUBLE PRECISION,EXTERNAL :: bessi0 ! bessel function
INTEGER i,k,n

! this case is used for integration of spc 5,6,7 but in f4 formulation of
    general kernel

integrandw_last = 0.0d0
constant1 = (q-r)*(sqrt(2.0d0*(x+x1)-a*a-((x-x1)/a)**2))
DO i = 5,5
!DO i = 1,n_max_gaulag
   integrandw = 0.0d0
   DO k = 1,npts_gaulag(i)
      wgt = w_gaulag(i,k)
      pt = x_gaulag(i,k)
   IF (abs (x1) < 1.0d-12) THEN
      integrandw =integrandw + wgt*crossec(max_smth,sqrt(q*q*x+pt),2.0d0*atan(
          q*sqrt(x/pt)))
   ELSE
      integrandw = integrandw + wgt*bessi0(constant1*sqrt(pt))*crossec(
          max_smth,sqrt(q*q*a*a+pt),2.0d0*atan(q*a/sqrt(pt)))
   END IF
   END DO
   IF( ( abs(( integrandw - integrandw_last )/integrandw ) < 1.0d-13 ) .OR. (
       abs( integrandw - integrandw_last ) < 1.0d-14 )) EXIT

   integrandw_last = integrandw
END DO
integralw = integrandw
IF (n == n_max_gaulag + 1) THEN
   WRITE(30,*) "VALUE_NOT_CONVERGED"
END IF

!CLOSE (30)
```

```
END FUNCTION INTEGRALW

!FUNCTION ELLIPTIC(qqc,pp,aa,bb)
FUNCTION ELLIPTIC(qqc,bb)
IMPLICIT NONE

INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
REAL(hiprec):: elliptic,qqc,a,b,f,g,em,p,qc,bb,aa,pp,q,e
REAL(hiprec),PARAMETER:: ca = 0.000000003d0, pi02 = 1.5707963268d0

IF (qqc == 0) PAUSE 'failure_in_elliptic'

qc = ABS(qqc)
b = bb
p = 1.0d0
e = qc
em = 1.0d0

f = 1.0d0
a = 1.0d0 + b
g = e
b = b + g
b = b + b
p = g + 1.0d0
g = em
em = qc+em
DO WHILE (ABS(g-qc) > g*ca)
    qc = sqrt(e)
    qc = qc+qc
    e = qc*em

    f = a
    a = a+b/p
    g = e/p
    b = b+f*g
    b = b+b
    p = g+p
    g = em
    em = qc+em
END do
elliptic = pi02*(b+a*em)/(em*(em+p))

END FUNCTION

FUNCTION BILINEAR(a,x,x1,f11,f12,f21,f22)
USE IONMOBILITY
IMPLICIT NONE
```

```
REAL(hiprec):: f11,f12,f21,f22,x,x1,bilinear

INTEGER a

SELECT CASE(a)
CASE(1)
! triangle interpolation
bilinear = f11 + (1.0d0/kmat_dvz)*(x*(f12-f22) - x1*(f12-f22))
!bilinear =   f11-(1.0d0/kmat_dvz)*((x*(f11-f21))-(y*(f21-f22)))

! check the formula

CASE(2)
! this is the 2D biliear interpolation using 4 poin/ts around the values of x
    and y
!bilinear = f11 + &
!           (1.0d0/kmat_dvz)*(f12*(x + y)) + f21*x +f12*y + &
!           (1.0d0/(kmat_dvz*kmat_dvz))*(x*y)*(f11-f21-f12+f22)
bilinear = f11 + &
           (1.0d0/kmat_dvz)*((f11-f21)*(-x) + (f11-f12)*(-x1)) + &
           (1.0d0/(kmat_dvz*kmat_dvz))*(x*x1)*(f11-f21-f12+f22)


END SELECT

END FUNCTION

FUNCTION F_1(z,a)
USE IONMOBILITY
USE MODULE_INTEGRATE
IMPLICIT NONE

REAL(hiprec) :: a,z,integrand,integrand_last,F_1,pt,wgt,constant,constant1

REAL(hiprec), EXTERNAL:: bessi0, bessi1

INTEGER n,i

constant = a*z/(1.0d0+a)
F_1 = exp(constant)/(1.0d0+a)

return

!
!  I discovered numerically that the result of the integration below is the
!  very simple function given above....  Lucky!!!!
```

```
!
!  I've left the code below in case one wants to go back to the integration
!  and check things numerically
!
constant = a*z/2.0d0/(1.0d0+a)
integrand_last = 0.0d0
!DO n = 3,3
DO n = 1,n_max_gauss
   integrand = 0.0d0
   DO i = 1,npts_gauss(n)
      wgt = w_gauss(n,i)
      pt = x_gauss(n,i)
      constant1 = constant*(1.0d0-pt*pt)
      integrand = integrand + wgt*exp(constant1)*( (1.0d0+2.0d0*constant1)*
          bessi0(constant1) + 2.0d0*constant1*bessi1(constant1) )
   END DO
   write(30,*) npts_gauss(n), integrand
   IF (abs ( ( integrand - integrand_last )/integrand ) < 1.0d-13) THEN
   EXIT
   END IF
   integrand_last = integrand
END DO
F_1 = integrand/(1.0d0+a)
IF ( n == n_max_gauss + 1 ) THEN
   WRITE(30,*) "VALUE_NOT_CONVERGED"
END IF
write(53,*) constant, integrand, log(integrand)

!CLOSE (30)
END FUNCTION F_1

FUNCTION F_2(x,x1,z,a1,a2)
USE IONMOBILITY
USE MODULE_INTEGRATE
IMPLICIT NONE

REAL(hiprec) :: a1,a2,x,x1,z,integrand,integrand_last,F_2,pt1,pt2,wgt1,wgt2,
    alpha,delta,constant,constant1,constant2,cost, cost1

INTEGER n1,n2,i1,i2

alpha = (1.0d0+a1)*a2
constant = (x-x1)/z
delta = 2.0d0*(x+x1)-z*z-constant*constant
constant1 = alpha/(1.0d0+alpha)*a1/4.0d0
constant2 = alpha*z*z/4.0d0/a1
```

```
integrand_last = 0.0d0
DO n1 = 3,3
!DO n1 = 1,n_max_gauss
   DO n2 = 3,3
   !DO n2 = 1,n_max_gauss
      integrand = 0.0d0
      DO i1 = 1,npts_gauss(n1)
      DO i2 = 1,npts_gauss(n2)
         wgt1 = w_gauss(n1,i1)
         pt1 = x_gauss(n1,i1)
         wgt2 = w_gauss(n2,i2)
         pt2 = x_gauss(n2,i2)
         cost1 = pi*pt2
         cost = cos(cost1)
         integrand = integrand + wgt1*wgt2*exp(constant1*(delta*(1.0d0-cost*
             cost*pt1)-2.0d0*constant*sqrt(delta)*cost*sqrt(pt1*(1.0d0-pt1))+
             constant*constant*pt1)-constant2*pt1)
      END DO
      END DO

      if( ( abs( ( integrand - integrand_last )/integrand ) < 1.0d-13 ) .or. (
           abs( integrand - integrand_last ) < 1.0d-14 ) ) exit

      integrand_last = integrand
   END DO
END DO
F_2 = integrand*(1.0d0+a1)*(1.0d0+a1)*(1.0d0+a2)*(1.0d0+a2)*z*z/4.0d0/a1/(1.0
    d0+alpha)
IF ( ( n1 == n_max_gauss + 1 ) .or. ( n2 == n_max_gauss + 1 )) THEN
   WRITE(30,*) "VALUE_NOT_CONVERGED"
END IF

!CLOSE (30)
END FUNCTION F_2

FUNCTION bessi0(x)
IMPLICIT NONE
INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
REAL(hiprec) bessi0,x,ax,y
REAL(hiprec),PARAMETER:: p1 = 1.0d0,        p2 = 3.5156229d0, p3 = 3.0899424d0,
                         p4 = 1.2067492d0, p5 = 0.2659732d0, p6 =
    0.360768d-1,                          p7 = 0.45813d-2
REAL(hiprec),PARAMETER:: q1 = 0.39894228d0, q2 = 0.1328592d-1, q3 = 0.225319d
    -2,                    q4 = -0.157565d-2, q5 = 0.916281d-2, q6
    =-0.2057706d-1,                        q7 = 0.2635537d-1, q8 = -0.1647633
    d-1, q9 = 0.392377d-2
ax = abs(x)
```

```
if( ax.lt.3.75d0 ) then
   y = (x/3.75d0)**2
   bessi0 = p1+y*(p2+y*(p3+y*(p4+y*(p5+y*(p6+y*p7)))))
else
   y = 3.75d0/ax
   bessi0 = (exp(ax)/sqrt(ax))*(q1+y*(q2+y*(q3+y*(q4+y*(q5+y*(q6+y*(q7+y*(q8+y
      *q9)))))))))
endif
END FUNCTION bessi0
!  (C) Copr. 1986-92 Numerical Recipes Software !"#D&'.

FUNCTION BESSI0_S(x)
IMPLICIT NONE

INTEGER, PARAMETER:: hiprec = KIND(0.0D0)

REAL(hiprec) :: bessi0_s,ax

REAL(hiprec), INTENT(IN) :: x

REAL(hiprec),DIMENSION(7):: P = (/1.0D0,3.5156229D0,3.0899424D0,1.2067492D0,
    0.2659732D0,0.0360768D0,0.0045813D0/)

REAL(hiprec),DIMENSION(9):: Q = (/0.39894228D0,0.01328592D0,0.00225319D0
    ,-0.00157565D0, 0.00916281D0, -0.02057706D0, 0.02635537D0, -0.01647633D0,
    0.00392377D0/)

!REAL(hiprec), EXTERNAL:: poly

INTERFACE
   FUNCTION POLY(x,coeffs)
   INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
   REAL(hiprec),DIMENSION(:):: coeffs
   REAL(hiprec):: x
   END FUNCTION
END INTERFACE

ax = abs(x)
IF (ax < 3.75d0 ) THEN
   bessi0_s = poly((x/3.75d0)**2,p)
ELSE
   bessi0_s = (exp(ax)/sqrt(ax))*poly((3.75d0/ax),q)
END IF

END FUNCTION

FUNCTION POLY(x,coeffs)
```

```
IMPLICIT NONE

INTEGER, PARAMETER:: hiprec = KIND(0.0D0)

INTEGER, PARAMETER :: npar_poly = 8

REAL(hiprec), INTENT(IN) :: x

REAL(hiprec), DIMENSION(:), INTENT(IN) :: coeffs

REAL(hiprec) :: poly,pow

REAL(hiprec), DIMENSION(:), ALLOCATABLE:: vec

INTEGER i,n,nn

!DO x = 0.0d0,10.0d0,0.00001d0

n = size(coeffs)
IF (n <= 0) THEN
    poly = 0.0d0
ELSE IF (n < npar_poly) THEN
    poly = coeffs(n)
    DO i = n-1,1,-1
        poly = x*poly+coeffs(i)
    END DO
ELSE
    ALLOCATE(vec(n+1))
    pow = x
    vec(1:n) = coeffs
    DO
       vec(n+1) = 0.0d0
       nn = ishft(n+1,-1)
       vec(1:nn) = vec(1:n:2)+pow*vec(2:n+1:2)
       IF (nn == 1) EXIT
       pow = pow*pow
       n = nn
   END DO
    poly = vec(1)
   DEALLOCATE(vec)
END IF
!write(53,*) x, poly
!end do
!stop
END FUNCTION

FUNCTION EXE_FUNC(n,x)
```

```
USE IONMOBILITY
IMPLICIT NONE

REAL(hiprec),PARAMETER:: alpha = 1.0d0!,pi = 3.14159265359

! All values in this subroutine are set to double precision
! This external function subroutine defines the initial function to be used in
    the ion mobility problem.
! This is the starting point in the Runge-Kutta method of numerical analysis.
! The first derivative of this function starts the iteration process.

REAL(hiprec) :: exe_func,x

INTEGER n
! this is the function we choose

IF (spc == 2) THEN
   IF (n == 0) THEN
      exe_func = 1.0d0
      !exe_func = exp(-x) !alpha = 2
      !exe_func = exp(0.5d0*x) !alpha = 0.5
      !exe_func = exp(0.98d0*x) ! alpha = 50
      !exe_func = exp(x)*(exp(-0.2d0*(x-2.0d0)**2)+exp(-0.2d0*(x-8.0d0)**2))!
          alpha = odd function
   ELSE
      exe_func = 0.0d0
      !exe_func = -exp(-x)
      !exe_func = 0.5d0*exp(0.5d0*x)
      !exe_func = 0.98d0*exp(0.98d0*x)
      !exe_func = exp(x-0.2d0*(x-2.0d0)**2)*(1.0d0-0.4d0*(x-2.0d0)) + exp(x
          -0.2d0*(x-8.0d0)**2)*(1.0d0-0.4d0*(x-8.0d0))
   END IF

ELSE IF (spc == 1 .OR. spc == 8 .OR. spc == 9) THEN
   IF (n == 0) THEN
      exe_func = exp(-x)*sqrt(x) ! alpha = 1
      !exe_func = sqrt(x)*exp(-0.5d0*x) !alpha = 0.5
      !exe_func = sqrt(x)*exp(-2.0d0*x) !alpha = 2
      !exe_func = sqrt(x)*exp(-0.02d0*x) ! alpha = 50
      !exe_func = exp(-0.1d0*(x-2.0d0)**2) + 1.5d0*exp(-0.1d0*(x-8.0d0)**2)!
          alpha = odd function
   ELSE
      exe_func = -exp(0.5d0*x)*(0.5d0*sqrt(0.5d0*x)-1/(sqrt(0.5d0*x)))
      !exe_func = -0.5d0*exp(-0.5d0*x)*(1/sqrt(x) - sqrt(x))
      !exe_func = -2.0d0*exp(-2.0d0*x)*(1/sqrt(x) - sqrt(x))
      !exe_func = -0.020d0*exp(-0.02d0*x)*(1/sqrt(x) - sqrt(x))
```

```
        !exe_func = -0.2d0*(x-2.0d0)*exp(-0.1d0*(x-2.0d0)**2) - 0.3d0*(x-8.0d0)*
            exp                    (-0.1d0*(x-8.0d0)**2)
    END IF
ELSE IF (spc == 3) THEN
    IF (n == 0) THEN
        exe_func = x
        !exe_func = 2.0d0*x !alpha = 2
        !exe_func = exp(-0.5d0*x) ! alpha = 0.5
        !exe_func = exp(-0.02d0*x) !alpha = 50
        !exe_func = -log(exp(-0.2d0*(x-2.0d0)**2) + exp(-0.2d0*(x-8.0d0)**2))!
            alpha = odd function
    ELSE
        exe_func = 1.0d0
        !exe_func = 2.0d0
        !exe_func = -0.5d0*exp(-0.5d0*x)
        !exe_func = -0.02d0*exp(-0.02d0*x)
        !exe_func = 2.4d0/(exp(-0.2d0*(x-2.0d0)**2) + exp(-0.2d0*(x-8.0d0)**2))
    END IF
ELSE IF (spc == 4 .OR. spc == 5 .OR. spc == 6 .OR. spc == 7 .OR. spc == 10 .OR
    . spc ==11) THEN
    IF (n == 0) THEN
        !exe_func = exp(-x)
        !exe_func = exp(-0.5d0*x) ! alpha = 2
        !exe_func = exp(-0.98d0*x) !alpha = 50
        !exe_func = exp(-2.0d0*x) ! alpha = 0.5
        !exe_func = exp(x)*(exp(-0.2d0*(x-2.0d0)**2)+exp(-0.2d0*(x-8.0d0)**2))!
            alpha = odd function

        exe_func = exp(-5.0d0*(x-350.0d0)**2) !newf4 !gaussian function used for
             spaces 5,6,7
      ! exe_func = sqrt(x)*exp(-0.5d0*(x-5.0d0)**2) !gaussian*sqrt
    ELSE
        !exe_func = -exp(-x)
        !exe_func = -0.5d0*exp(-0.5d0*x)
        !exe_func = -0.98d0*exp(-98.0d0*x)
        !exe_func = -2.0d0*exp(-2.0d0*x)
        !exe_func = exp(x-0.2d0*(x-2.0d0)**2)*(1.0d0-0.4d0*(x-2.0d0)) + exp(x
            -0.2d0*(x-8.0d0)**2)*(1.0d0-0.4d0*(x-8.0d0))
        exe_func = (x-350.0d0)*exp(-5.0d0*(x-350.0d0)**2)!derivative of newf4
        !exe_func = (0.5d0/sqrt(x))*exp(-0.5d0*(x-5.0d0)**2) + sqrt(x)*(x-5.0d0)
            *exp(-0.5d0*(x-5.0d0)**2) ! derivative of gaussian*sqrt
    END IF
!ELSE IF (spc == 5 .OR. spc == 6 .OR. spc == 7) THEN
!       IF (n == 0) THEN
!           !exe_func = exp(-x*x)!equilibrium function
!           !exe_func = exp(-0.5d0*(x)**2) !gaussian function
```

```
!          exe_func = exp(-5.0d0*(x-350.0d0)**2) !gaussian function high energy
          !exe_func = exp(-0.5d0*(x-5.0d0)**2) !gaussian function low energy
!       ELSE
          !exe_func = -2.0d0*x*exp(x*x)
          !exe_func = -x*exp(-0.5d0*(x)**2)
!          exe_func = -(x-350.0d0)*(exp(-0.5d0*(x-350.0d0)**2))
          !exe_func = -(x-5.0d0)*(exp(-0.5d0*(x-5.0d0)**2))
!       END IF
END IF

END FUNCTION

RECURSIVE FUNCTION DIST_FUNC(a,n,vz,coeff) RESULT(dist_func1)
USE IONMOBILITY
IMPLICIT NONE

!This external function is used in the main program ion_mobility.
!This function evaluates distribution function at points on velocity grid.
!w: the argument used to evaluate the bspline function.
!n determines whether the original function, the first derivative, or the
    second derivative of the third order bspline is used in the evaluation.
!Distribution function is divided by constant dvz raised to the power of n.
!This function calls on external function BSPLINE3 which contains third order
    spline expressions.

REAL(hiprec)::w,vz,index,dist_func2,dist_func1
REAL(hiprec),EXTERNAL:: bspline3
REAL(hiprec),DIMENSION(0:n_dim):: coeff
INTEGER i,n,a,j

dist_func1 = 0.0d0
IF ((vz_grid(0) > vz).OR.(vz > vz_grid(vz_num)))RETURN

index = (vz-vz_grid(0))/dvz

SELECT CASE(a)

CASE(1)
!evaluates distribution function at given grid point in native space
   DO i = INT(index)-3, INT(index)
      w = index - REAL(i)
      dist_func1 = dist_func1 + coeff(i+3)*BSPLINE3(n,w)
   END DO
    dist_func1 = dist_func1/dvz**n

CASE (2)
!this case evaluates the moments as desired.
```

```
!moments are evaluated in terms of f
!we check size of grid, since some of the moments can fall of the end of grid.
!only moments that actually lie on the grid are needed
!following check ensures that only the relevant points are evaluated
!gspace undone to fspace
      IF (n == 0) THEN
        dist_func1 = 2.0d0*sqrt(vz/pi)*exp(-vz)*dist_func(1,0,vz,coeff)
      ELSE
         dist_func1 = 2.0d0/sqrt(pi)*exp(-vz)*(sqrt(vz)*dist_func(1,1,vz,coeff
             ) + (0.5d0*sqrt(1/vz) - sqrt(vz))*dist_func(1,0,vz,coeff))
      END IF


CASE(3)
!converts the calculation of distribution function from h-space to f-space
      IF (n == 0) THEN
         dist_func1 = 2.0d0*sqrt(vz/pi)*exp(-1.0d0*dist_func(1,0,vz,coeff))
      ELSE
         dist_func1 = 2.0d0*exp(-1.0d0*dist_func(1,0,vz,coeff))/sqrt(pi)*(0.5
             d0/sqrt(vz) - sqrt(vz)*dist_func(1,1,vz,coeff))
      END IF
CASE(4)
!fprime space comments
!this case evaluates the moments as desired.
!the moments are evaluated in terms of f
!check size of grid, since some of the moments can fall of the end of the grid
    .
!only moments that actually lie on the grid are needed
!following check ensures that only the relevant points are evaluated
!gspace undone to fspace
      IF (n == 0) THEN
        dist_func1 = 2.0d0*sqrt(vz/pi)*dist_func(1,0,vz,coeff)
      ELSE
         dist_func1 = 2.0d0/sqrt(pi)*(0.5d0/sqrt(vz)*dist_func(1,0,vz,coeff) +
              sqrt(vz)*dist_func(1,1,vz,coeff))
      END IF
CASE(5)
!general case of the kernel
   IF (n == 0) THEN
      dist_func1 = 2.0d0*sqrt(vz/pi)*dist_func(1,0,vz,coeff)
   ELSE
      dist_func1 = 2.0d0/sqrt(pi)*(0.5d0/sqrt(vz)*dist_func(1,0,vz,coeff) +
          sqrt(vz)*dist_func(1,1,vz,coeff))
   END IF
END SELECT
!dist_func1 = dist_func
```

```
END FUNCTION

FUNCTION BSPLINE3(n,w)
USE IONMOBILITY
IMPLICIT NONE

! This is an external function describing the piecewise bspline function.
! The function is described over four intervals, where it has a different form
    .
! Depending on the value of w, the argument, the appropriate case of the
    bspline! is called.
! The bspline function has values over only four points, and zero everywhere
! else.
! w: argument used to evaluate the bspline expression
! n determines which case is to be called upon during the evaluation.

REAL(KIND = hiprec)::bspline3,w

INTEGER n

IF (w < 0)THEN
      BSPLINE3 = 0.0d0
      RETURN
END IF

SELECT CASE (n)

!This case has the expressions for the original cubic spline function.
CASE (0)
   SELECT CASE (int(w))
      CASE (0)
         bspline3 = 1.0d0/6.0d0*(w**3)
      CASE (1)
         bspline3 = 1.0d0/6.0d0*((w**3)-4.0d0*(w-1.0d0)**3)
      CASE (2)
         bspline3 = 1.0d0/6.0d0*((4.0d0-w)**3-4.0d0*(3.0d0-w)**3)
      CASE (3)
         bspline3 = 1.0d0/6.0d0*(4.0d0-w)**3
      CASE DEFAULT
         bspline3 = 0.0d0
   END SELECT

! This case has the expressions for the first derivative of the cubic splines.
CASE (1)
   SELECT CASE (int(w))
      CASE (0)
         bspline3 = 0.5d0*(w*w)
```

```
        CASE (1)
            bspline3 = 0.5d0*((w*w)-4.0d0*(w-1.0d0)**2)
        CASE (2)
            bspline3 = -0.5d0*((4.0d0-w)**2-4.0d0*(3.0d0-w)**2)
        CASE (3)
            bspline3 = -0.5d0*(4.0d0-w)**2
        CASE DEFAULT
            bspline3 = 0.0d0
    END SELECT

! This case consists of the second derivative expressions of the cubic splines
    .
CASE (2)
    SELECT CASE (int(w))
        CASE (0)
            bspline3 = w
        CASE (1)
            bspline3 = (4.0d0-3.0d0*w)
        CASE (2)
            bspline3 = (3.0d0*w-8.0d0)
        CASE (3)
            bspline3 = (4.0d0-w)
        CASE DEFAULT
            bspline3 = 0.0d0
    END SELECT
END SELECT

END FUNCTION

FUNCTION bessi1(x)
IMPLICIT NONE
INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
REAL(hiprec) bessi1,x,ax,y
REAL(hiprec),PARAMETER:: p1 = 0.5d0,        p2 = 0.87890594d0, p3 = 0.51498869
    d0,                          p4 = 0.15084934d0, p5 = 0.2658733d-1, p6 =
    0.301532d-2,                          p7 = 0.32411d-3
REAL(hiprec),PARAMETER:: q1 = 0.39894228d0, q2 =-0.3988024d-1, q3 =-0.362018d
    -2,                          q4 = 0.163801d-2, q5 =-0.1031555d-1, q6 =
    0.2282967d-1,                          q7 =-0.2895312d-1, q8 = 0.1787654d
    -1, q9 = -0.420059d-2

ax = abs(x)
if( ax.lt.3.75d0 ) then
    y = (x/3.75d0)**2
    bessi1 = x*(p1+y*(p2+y*(p3+y*(p4+y*(p5+y*(p6+y*p7))))))
else
    y = 3.75d0/ax
```

```
    bessi1 = (exp(ax)/sqrt(ax))*(q1+y*(q2+y*(q3+y*(q4+y*(q5+y*(q6+y*(q7+y*(q8+y
        *q9)))))))))
    if( x.lt.0.0d0 ) bessi1 = -bessi1
endif
END FUNCTION bessi1
!  (C) Copr. 1986-92 Numerical Recipes Software !"#D&'.
```

A.4 Modules

```
MODULE IONMOBILITY
IMPLICIT NONE

INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
REAL(hiprec)::t,h,h2,dvz,vz_0,m,colfreq,q,r,kmat_vz_0,kmat_vz_n,kmat_dvz,
    kmat_m,mu_chi
!t:time
!h: time step
!dvz: size of the interval in the velocity grid
! m: mass ratio of the bath gas to the ion
! colfreq: the collision frequency
INTEGER:: n_dim, vz_num,spc,max_smth,kmat_vz_num
!vz_num: the total number of points on the velocity grid
!n_dim: size of the arrays . We get extra two values due to the conditions on
    the derivatives. These extra values are needed for the cubic splines.
REAL(hiprec),PARAMETER::pi = 3.14159265359
!REAL(KIND = hiprec), DIMENSION(1:1100):: x_gauss,w_gauss
! x_gauss: the points evaluated by the gaussian quadrature used in the
    evaluation of the integrands
! w_gauss: the weights evaluated by the gaussian quadrature used in the
    evaluation of the integrands
REAL(hiprec),DIMENSION(:),ALLOCATABLE:: y,k1,k2,k3,k4,y1,vz_grid,work,wr,
    work1,wr1,kmat_x_val,kmat_y_val
!y: value of the derivative at the interval
!k1: the first evaluation of the derivative
!k2: the second evaluatin of the derivative
!k3: the third evaluation of the derivative
!k4: the fourth evaluation of the derivative
!y1: value of the derivative after the evaluation at the four points within
    the ! interval.  This also becomes the next point of evaluation in the
    Runge-Kutta
REAL(hiprec),DIMENSION(:,:),ALLOCATABLE::l1_mat,l2_mat,vr,vr_saved,vr_saved1,
    vr1,l11_mat,l21_mat,b_init,l_mat_safe,k_mat_val
!l1_mat: the field-dependent matrix product of two matrices, inverse of b_til
    and b_p
!l2_mat: the product of matrices, inverse of b_til, k_mat, and nu and b
! vr_saved: the array containing the eigenfunctions of the eigenvalues of L
```

```
! vr: the array containing the inverse of vr_saved

END MODULE IONMOBILITY

MODULE MODULE_INTEGRATE
USE IONMOBILITY
IMPLICIT NONE

INTEGER,PARAMETER:: n_max_gauss = 7, n_max_trap = 25
INTEGER,DIMENSION(1:n_max_gauss):: npts_gauss
INTEGER npts_trap
REAL(hiprec), DIMENSION(1:n_max_gauss,1:2**(n_max_gauss+2)):: x_gauss,w_gauss
! x_gauss: the points evaluated by the gaussian quadrature used in the
    evaluation of the integrands
! w_gauss: the weights evaluated by the gaussian quadrature used in the
    evaluation of the integrands

END MODULE

MODULE MODULE_CROSSEC
USE IONMOBILITY
IMPLICIT NONE


REAL(hiprec),DIMENSION(:),ALLOCATABLE:: theta,chi,cross_func

INTEGER n_val,n_chi

REAL(hiprec):: dchi,a4_tilda

END MODULE MODULE_CROSSEC

MODULE INTERFACES
USE IONMOBILITY
IMPLICIT NONE


!INTERFACE int_dist_func
!   REAL(hiprec)

!INTERFACE moments1
!   REAL(hiprec) FUNCTION MOMENTS(a,coeff1,a_max,b_min)
!     INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
!     REAL(hiprec):: a_max,b_min
!     INTEGER a
!     REAL(hiprec) coeff1(:)
!   END FUNCTION MOMENTS
```

```
!END INTERFACE

INTERFACE moments_chk1
   REAL(hiprec) FUNCTION MOMENTS_CHK(a,coeff1,a_max,b_min)

use ionmobility
!      INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
       REAL(hiprec):: a_max, b_min
       INTEGER a
       REAL(hiprec) coeff1(:)
   END FUNCTION MOMENTS_CHK
END INTERFACE

INTERFACE dist_func1
   REAL(hiprec) FUNCTION DIST_FUNC(a,n,vz,coeff1)

use ionmobility
!      INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
       INTEGER a,n
       REAL(hiprec) coeff1(:)
       REAL(hiprec):: vz
   END FUNCTION DIST_FUNC
END INTERFACE

INTERFACE
   SUBROUTINE GAULAG(x,w,alf,n)

use ionmobility
 !     INTEGER, PARAMETER:: hiprec = KIND(0.0D0)
       INTEGER n
       REAL(hiprec):: alf
       REAL(hiprec)  x(:),w(:)
   END SUBROUTINE GAULAG
END INTERFACE

END MODULE INTERFACES

MODULE INTEGRATE_GAULAG
USE IONMOBILITY
IMPLICIT NONE

INTEGER,PARAMETER::n_max_gaulag= 5
REAL(hiprec):: alf
! the parameter used by the gaulag subroutine in the evaluation of the points
    and weights
! for the purpose of this code, alf = 0
INTEGER,DIMENSION(1:n_max_gaulag) :: npts_gaulag
```

```
REAL(hiprec), DIMENSION(1:n_max_gaulag,1:2**(n_max_gaulag+2)):: x_gaulag,
    w_gaulag
! the arrays that will store the points (x_gaulag) and weights (w_gaulag)
! all the points and weights for the different values of n are constructed in
    the initalization
! the soubroutine moments calls the different values of n, and depending on
    convergence, the appropriate set of points and weights are used in the
    evaluation of the moments

END MODULE INTEGRATE_GAULAG
```

A.5 C Subroutines

```
/*
   C program to call the error functions in the C math library
*/

#include <math.h>

double erf_(double *arg)
{
   return(erf(*arg));
}

double erfc_(double *arg)
{
   return(erfc(*arg));
}

/*
   C program to call the gamma function in the C math library
*/

#include <math.h>

double lgamma_(double *arg)
{
   return(lgamma(*arg));
}
/*
double gammln_s_(double *arg)
{
  return(gammln_s(*arg));
}
*/
```

A.6 Input File

```
0 10 100 vz_n vz_num
0.02 1.0 mass ratio colfreq
40 0.005 nstep h
1 spc
1.5 mu_chi (rough)
1.0 1.0 1 numdense sigma12 max_smth (spherical)
1 a4_tilda (maxwell)
```

A.7 Makefile

```
F90 = ifort
#
#  Intel flags
#
#LIBS   = -L/global/software/intel/mkl72/lib/32 -lmkl_lapack
#LIBS = -L/global/software/intel/mkl/lib/intel64 -lmkl_lapack
#pgf90 flags
#LIBS = -llapack -lblas -lpgftnrtl -lrt -lpthread -lm
OFLAGS = -mkl -fast -Bstatic -openmp
#optimization
#OFLAGS =  -openmp -mkl -Bstatic
#(debugger)
F90FLAGS  = -c ${OFLAGS}
#
#  Source and object files for mobility code
#
OBJECTS2 = module.o module_integrate.o module_crossec.o interfaces.o bessi1.o\
           l_init_print.o module_gaulag.o ion_mobility.o gauss.o bspline3.o \
           rk4.o deriv.o field.o exe_func.o dist_func.o erf_.o l_init.o \
           k_mat.o integrate.o cnvrg_chk.o int_interval.o gammaln_.o \
           mat_chk.o gaulag.o plot_function.o polint.o chi.o elliptic.o\
           integralw.o bessel.o crossec.o bilinear.o F_1.o F_2.o bessi0.o \
           chi.o
           #ion_mobility1.o k_mat_calc.o k_mat_calc_func.o\

OBJECTS3 = module.o analyze.o bspline3.o polint.o dist_func.o drg_coeff.o \
           erf_.o

OBJECTS4 = module.o module_gaulag.o module_integrate.o module_crossec.o chi.o\
           k_mat_calc.o crossec.o gammaln_.o gauss.o dist_func.o bspline3.o\
           bessi0.o integralw.o elliptic.o k_mat_calc_func.o polint.o F_2.o\
           gaulag.o
```

```
OBJECTS5 = module_crossec.o chi.o elliptic.o

#OBJECTS6 = kmat_check.o module.o module_integrate.o integralw.o crossec.o
    bessel.o module_gaulag.o gammaln.o elliptic.o

%.o: %.f90
        $(F90) $(F90FLAGS) -o $@ $<

#-------------------------DESCRIPTION PART ---------------------
ion_mobility_ww: $(OBJECTS2)
        ${F90} ${OFLAGS} -o ion_mobility_ww ${OBJECTS2} ${LIBS}

analyze: $(OBJECTS3)
        ${F90} ${OFLAGS} -o analyze ${OBJECTS3} ${LIBS}

k_mat_calc: $(OBJECTS4)
        ${F90} ${OFLAGS} -o k_mat_calc ${OBJECTS4} ${LIBS}

elliptic: $(OBJECTS5)
#       ${F90} ${OFLAGS} -o elliptic ${OBJECTS5} ${LIBS}

#kmat_chk: $(OBJECTS6)
#        ${F90} ${OFLAGS} -o kmat_chk ${OBJECTS6} ${LIBS}

clean:
        rm *.o ion_mobility_ww
```

A.8 Submission Script

```
#!/bin/csh
#
#PBS -l nodes=1:fast:ppn=6
#PBS -N maxwell_external
#PBS -o output
#PBS -e error

cd $PBS_O_WORKDIR
setenv OMP_NUM_THREADS 6

#/tmp/K_MAT_VALUES_EXTRAP
#./ion_mobility_ww < input > Integrand_values
./k_mat_calc < input_calc
```