Relation of Electron Scattering Cross-Sections to Drift Measurements in Noble Gases

by

Blake Stacey

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Physics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author . . . .
Department of Physics
May 6, 2005

Certified by . . . .
Ulrich J. Becker
Professor, Department of Physics
Thesis Supervisor

Accepted by . . . .
Prof. David E. Pritchard
Senior Thesis Coordinator, Department of Physics

Relation of Electron Scattering Cross-Sections to Drift Measurements in Noble Gases

by

Blake Stacey

Abstract

We investigate the classic "inverse problem" of extracting collision and scattering cross sections from measurements of electron swarm behavior. A Monte Carlo technique for simulating electron motion through a gas of isotropic scatterers is presented, providing a simplified version of Biagi's MAGBOLTZ algorithm. Using this Monte Carlo software, we examine the thermalization of electron swarms, focusing on their drift velocity and informational entropy, providing justification for a set of analytic expressions for drift measurements which are valid in the hydrodynamic regime. These expressions are then used to estimate the ^{4}He scattering cross section, first by a simple grid interpolation and then through a genetic algorithm (GA). This technique demonstrates that the ^{4}He momentum-transfer cross section in the 0-7 eV range is approximately 6.5 Å^2, with a peak near 2 eV, in agreement with literature values. Empirical cross sections are also presented for Xe and He:CH$_4$(90:10).

Thesis Supervisor: Ulrich J. Becker
Title: Professor, Department of Physics

Acknowledgments

Prof. Mehran Kardar expanded my understanding of kinetic theory. E-mail communications with Prof. Stephen Biagi, University of Liverpool, greatly facilitated my understanding of his MAGBOLTZ code. Eric Downes read and offered valuable comments upon my discussion of electron-swarm entropy. I owe the parenthetical comment in Section 2.2 to Prof. Jeffrey Goldstone, who co-taught my first string theory class alongside Prof. Barton Zwiebach. I thank the many kind spirits who comprise the MIT LNS Drift Gas R&D group, and in particular Prof. Ulrich Becker, whose kind comments and helpful insight made this work possible.

Contents

Chapter 1

Introduction

We consider here the motion of ionization electrons created by the energy loss of a charged particle moving through a gas. This situation corresponds, for example, to present drift chambers, where the drift velocity of the electrons v_d is small compared to their instantaneous random velocity. $v_d \cdot t = x$ determines the location of the ionizing particle's track, where v_d depends upon the gas used, and in particular the electron-atom scattering cross section.

Obtaining collision and scattering cross sections from data on electron swarms liberated by ionization in a gas is a classic "inverse problem." Cross sections are fundamental quantities, dependent upon the energy of incident electrons (typically in the 0.3 to 3 eV range) and the details of atomic states. An electron swarm's properties, its drift velocity, diffusion coefficients or magnetic deflection angle, are derived from the purely local cross section parameters.

The study is motivated by the vast amount of drift velocity data (see, *e.g.,* [4]) compared to the only spotty knowledge of cross section measurements.

Chapter 2

Microscopic Scattering Theory

In this chapter, we begin the discussion of electron swarms and their motion through neutral gases. This field of inquiry is an excellent example of the importance of proper approximations, a topic on which later sections will elaborate. To begin with, we note that in the conditions of a typical drift detector, the de Broglie wavelength of the swarming electrons is much smaller than the spacing between atoms (for gas pressures less than about 100 atmospheres). Consequently, the swarm can be modeled as a set of classical particles, each interacting with only one gas atom at a time [10].
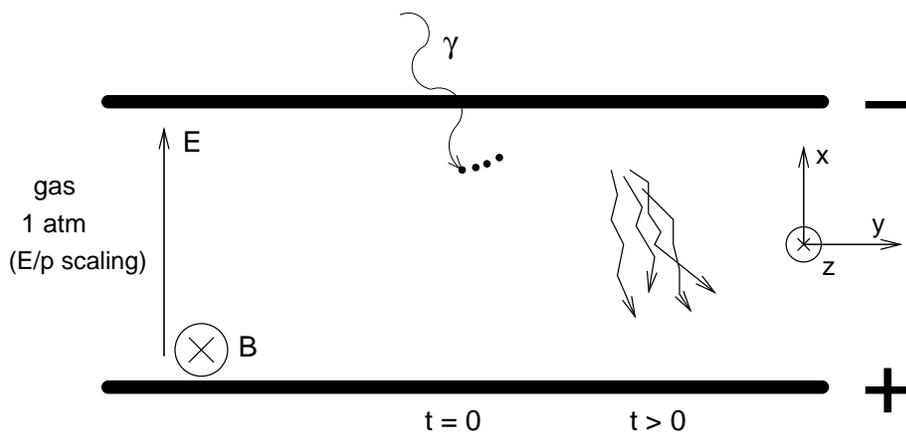

Figure 2-1: Schematic sketch of an electron swarm progressing through a drift chamber.

2.1 Motion in Free Space

Consider a nonrelativistic electron moving through vacuum, subjected to $\vec{E}$ and $\vec{B}$ fields. For our drift chamber geometry, we can choose a coordinate system such that $\vec{B} = B\hat{z}$ and $\vec{E} = E_x\hat{x}$. The electron's acceleration is given by the Lorentz force law,

$$m\frac{d\vec{v}}{dt} = e\vec{E} + e\vec{v} \times \vec{B}. \tag{2.1}$$

In our coordinate system, this simplifies to the three coupled equations

$$\dot{v}_x = \left(\frac{e}{m}\right)E_x + \omega v_y, \tag{2.2}$$

$$\dot{v}_y = -\omega v_x, \tag{2.3}$$

$$\dot{v}_z = 0. \tag{2.4}$$

Decoupling the first two equations gives

$$\ddot{v}_x + \omega^2 v_x = 0, \;\; \ddot{v}_y + \omega^2 v_y + \left(\frac{e}{m}\right)E_x\omega = 0,$$

which can be solved to yield

$$
\begin{aligned}
v_x &= \frac{eE_x}{m\omega}\sin(\omega t) + v_{ox}, \\
v_y &= \frac{eE_x}{m\omega}[\cos(\omega t) - 1] + v_{oy}, \\
v_z &= v_{oz}.
\end{aligned}
\tag{2.5}
$$

Integrating these velocities from an initial time $t = 0$ to a time $t = \Delta t$ later gives expressions for the incremental change in position:

$$
\begin{aligned}
\Delta x &= \frac{eE_x}{m\omega^2}(1 - \cos\omega\Delta t) + v_{ox}\Delta t, \\
\Delta y &= \frac{eE_x}{m\omega^2}\sin\omega\Delta t - \frac{eE_x}{m\omega}\Delta t + v_{oy}\Delta t, \\
\Delta z &= v_{oz}\Delta t.
\end{aligned}
\tag{2.6}
$$

Except for a change of coordinate axes, these formulas are essentially the same as those given in [5] (in the special case that $\vec{E}$ and $\vec{B}$ are orthogonal). Note that the corresponding equations in [5] lack factors of e/m on several instances of the electric field.

2.2 Scattering

The next phenomenon which we must describe is the interaction of an electron with a gas atom or molecule. These events occur with probabilities dependent upon the electron-atom scattering cross section, the gas density and the applied fields. The simplest case is electron motion through a noble gas, in which all scatterers are monatomic. Since the gas mixtures in many drift detectors contain large proportions of noble gases [3, 4], this is a case of considerable practical interest.

The standard practice is to model the noble-gas atoms as hard spheres whose cross sections σ_m depend upon the energy of the incident electron [20]. This is quantum-mechanically justifiable [26] as long as the electron's energy ϵ is less than that required to excite the atom. Conveniently, helium possesses a first excitation energy of ≈ 19 eV; collisions below this upper limit will be elastic. Newtonian kinematics gives the result that a fraction $\lambda(\theta)$ of the electron's energy will be lost to the atom, where θ is the polar angle between the incident and final velocities:

$$\lambda(\theta) = \frac{2mM}{(M+m)^2}(1 - \cos(\theta)). \tag{2.7}$$

Here, m is the electron mass and M is the mass of the target atom [15]. We shall use λ without an argument to denote the fractional energy loss averaged over all angles θ:

$$\lambda = \frac{2mM}{(M+m)^2}. \tag{2.8}$$

For helium gas, this ratio is roughly 2.7×10^{-4}.

Gases not of the noble variety introduce complications. The presence of asymmetric molecules with translational and vibrational modes allows for a variety of inelastic

collisions, each of which is described by an energy-dependent cross section and a characteristic energy loss. In addition, these molecular gases typically ionize more readily than the noble-gas mixture component. This may be convenient in some experimental situations, where such ionizations are necessary (*e.g.,* the apparatus of [3]), but it makes theoretical work more difficult. (In this context, one recalls J. Goldstone's aphorism, delivered to an undergraduate string theory class: "Theorem zero—you can't win.") Common examples of molecular gases present in drift chamber mixtures include N_2 and hydrocarbons like CH_4, C_2H_6, etc. [10].

2.3   Monte Carlo Algorithm

The concepts developed in Sections 2.1 and 2.2 lend themselves readily to a Monte Carlo (MC) implementation. Such an implementation is simple, in principle: define an electron within the computer's memory by its position and velocity coordinates, propagate that electron according to the Lorentz force law Eq. (2.1), decide stochastically whether or not the electron suffers a collision during that timestep, and in the event of a collision randomize the velocity vector with an appropriate energy loss.

Fraser and Matheison [15] give the basic rules for constructing such an MC simulation. C code to implement this construction is given in the Appendix. The key resource for the MC algorithm is a dependable source of random numbers. Using R to denote a random variable uniformly distributed between 0 and 1, we can write formulae for the kinematic quantities necessary to run the simulation. For isotropic scattering—the case covering hard-sphere noble gases, and the case most studied in this paper—the polar angle θ is computed by

$$\theta = \cos^{-1}(1 - 2R), \tag{2.9}$$

while the azimuthal angle ϕ is chosen from a uniform distribution,

$$\phi = 2\pi R. \tag{2.10}$$

The most sophisticated part of the MC algorithm is choosing the proper timestep. If computational efficiency is not a primary goal, one can code the MC so that it estimates the mean time between collisions and chooses a timestep significantly smaller. A somewhat more efficient approach is to use the null-collision technique due to Skullerud [33, 22].

Figure 2-2 shows the result of running the MC code given in the Appendix. Drift velocities were computed by measuring the displacement after a given number of collisions and dividing by the amount of time elapsed. Three observations are noteworthy: first, the drift velocity trace as a function of time $v_d(t)$ shows a transient behavior with a timescale of $\approx 3 \times 10^4$ collisions. MC runs with different starting conditions converge to the same result, after which v_d stays roughly constant for as long as the simulation runs. (In physical time, this transient behavior could exist on the order of a microsecond.) Second, when the MC is run for a range of $\vec{E}$ fields, the results for $\sigma_m = 7\text{Å}^2$ and $\sigma_m = 8\text{Å}^2$ neatly bracket the experimental results for He [14] (using the appropriate λ for He).

2.4   MAGBOLTZ

The *de facto* "industry standard" for calculating electron drift velocities and angles by Monte Carlo simulation is S. F. Biagi's MAGBOLTZ code, which takes as input cross sections defining the gases to be simulated [5]. The most recent version contains a database of 58 gases, whose properties are known with varying degrees of accuracy. Through the MC techniques outlined above, MAGBOLTZ can compute a swarm's drift velocity, Lorentz deflection angle, diffusion coefficients and other properties of interest. The program suffers the disadvantage that its cross-section parameters are stored internally, hard-coded directly into the program, so that modifying the characteristics of a gas involves finding and parsing the proper subroutine within approximately 2.5×10^4 lines of FORTRAN.

The basic principle of computerized science colloquially termed "GIGO" ("Garbage In, Garbage Out") certainly applies to MC swarm calculations. Even the industry
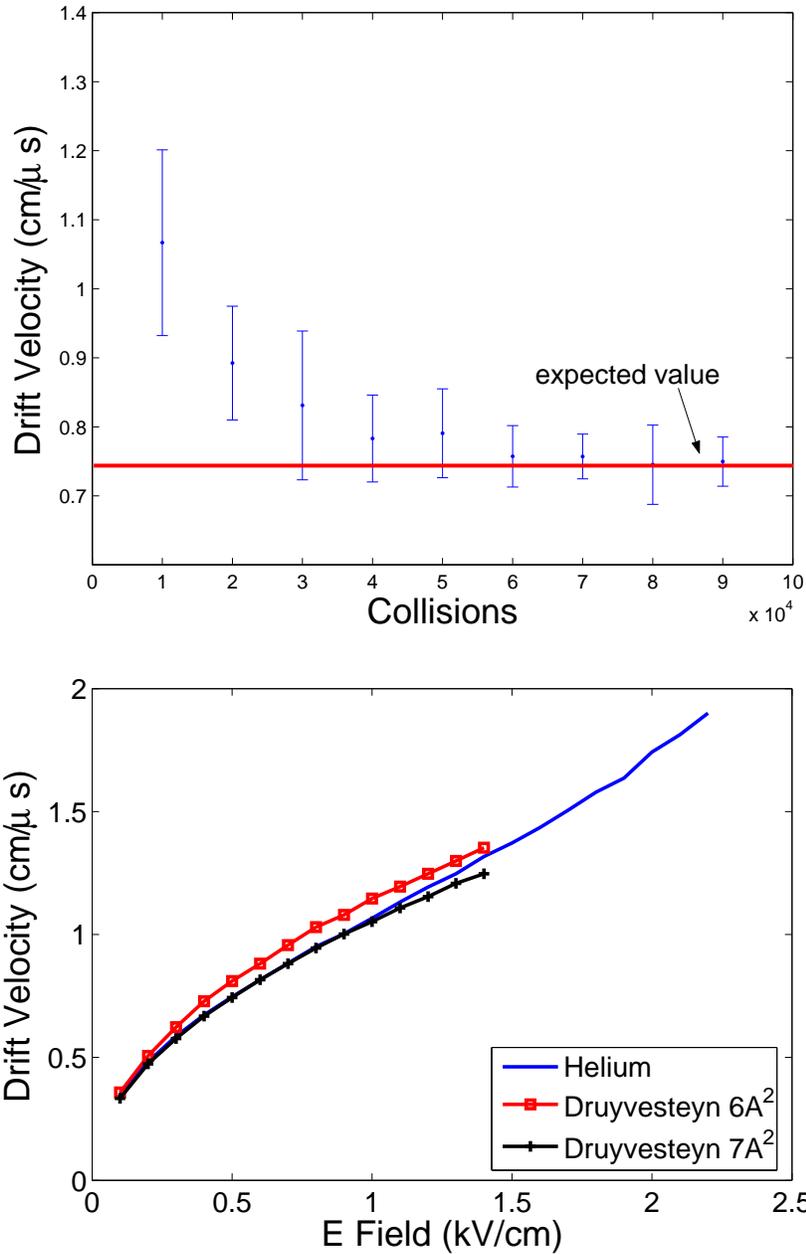
Figure 2-2: Drift velocities computed by Monte Carlo. The upper graph was computed for $E = 0.5$ kV/cm and $\sigma_m = 6\text{Å}^2$.
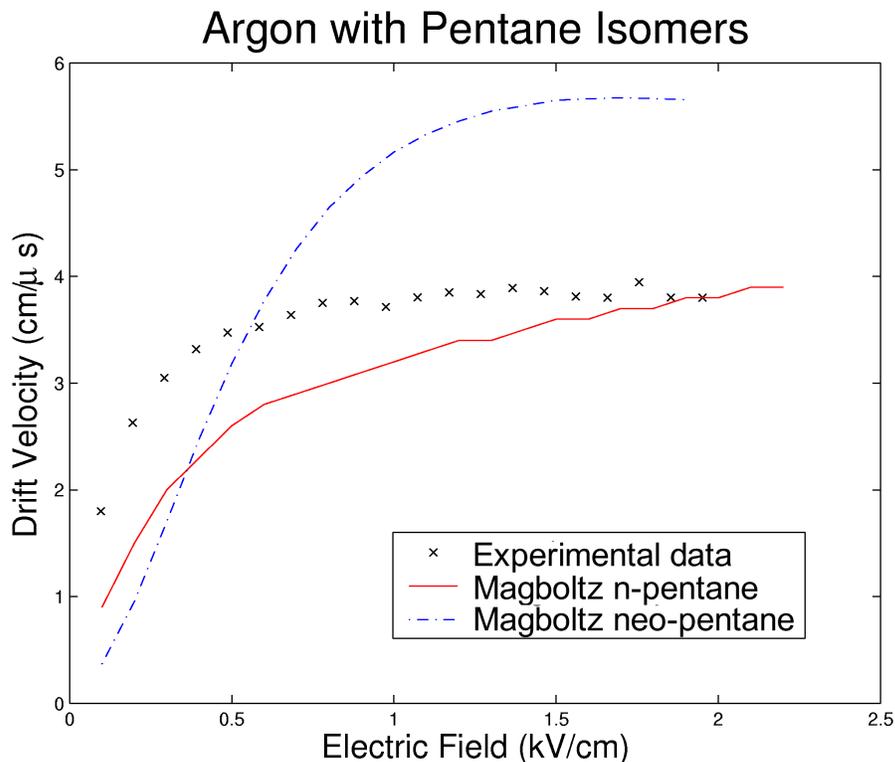
Figure 2-3: Comparison between experimental Ar:n-C_5H_{12} (60:40) mixtures (from [4]) and MAGBOLTZ calculations. At lower E fields, the discrepancy is highly noteworthy.

standard program can only be as good as the cross sections given it. For example, Fig. 2-3 shows a comparison of MAGBOLTZ results and experimental measurements for a 60:40 mix of Ar and n-pentane. Note the discrepancy between the experimental and theoretical curves; while some of the error may be experimental (see [3] for discussion of the experimental procedure and error ranges), part of the discrepancy is theoretical, due most likely to the fact that the n-pentane cross sections are not understood. Note also that the theoretical calculation performed for a 60:40 mix of Ar and neo-C_5H_{12} has a much higher drift velocity over much of the E range. This is due to the neo-pentane molecule's roughly spherical symmetry, which produces a sizeable Ramsauer minimum.

2.5   Monte Carlo Entropies

As Liu and Bardsley observe, the MC method is particularly well-suited to the study of transient effects [21]. We have already seen evidence of such a transient in Fig. 2-2, in which the drift velocity is seen to approach a steady-state value as the simulation progresses. Such a trend suggests a "thermalization" to a hydrodynamic flow situation, where the electron swarm moves in a regular fashion without regard to boundary conditions (in terms of the overall drift or "terminal velocity"; as we shall see, the scale of eE allows for energy fluctuations). When discussing a system's approach to thermal equilibrium, one frequently invokes the concept of entropy. (This is the context of Boltzmann's H-theorem; see the discussion in, *e.g.,* Huang [18].) We ask, therefore, if an MC simulation can be used to give a measure of entropy which we can follow as time progresses.

One way of quantifying the information content (or "disorder") involves the histogram of electron collisions as a function of energy. It is relatively trivial to arrange the MC code to deliver such data; histograms for three different E-field values are shown in Fig. 2-4. One observation easily presents itself: as the magnitude of the E-field is increased, we find electrons scattering at higher energies. (In a language like C which does not provide automatic bounds checking on its arrays, care must be taken when building such histograms, so that electrons do not "scatter" into other regions of memory. Such implementation details are addressed more fully in the Appendix's code listing.) If we treat the normalized histogram $h(\epsilon)$ as a probability distribution, then Boltzmann's formula gives us a measure of the electron's entropy:

$$S_B = -k_B \sum_\epsilon h(\epsilon) \log h(\epsilon). \tag{2.11}$$

We use the subscript on S_B as a reminder that this is a *derived* quantity, of as yet uncertain physical significance.

Fig. 2-5 shows a typical time-evolution trace of $v_d(t)$ compared with a plot of $S_B(t)$ for the same simulation. Note that the two quantities come to "hydrodynamic" values on roughly the same time scale—a promising indicator that we are in fact seeing a
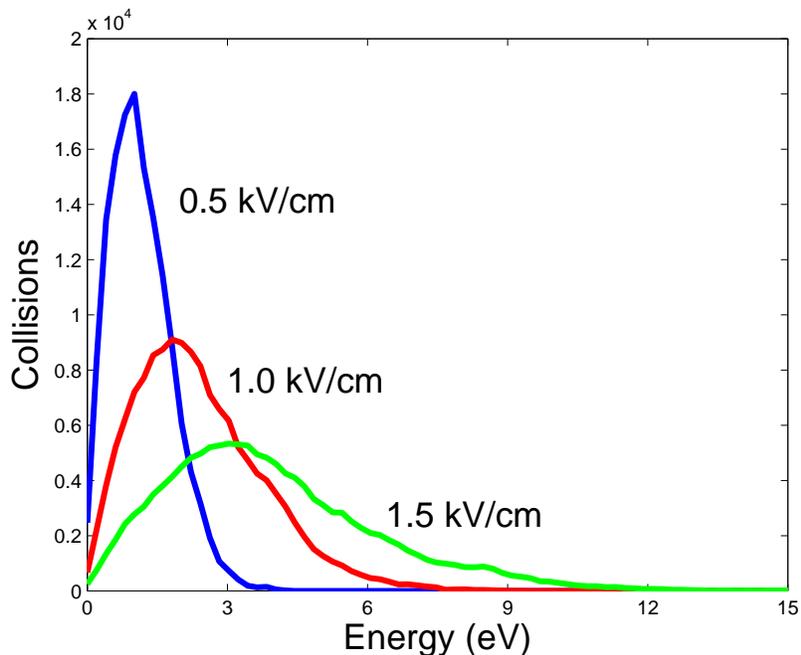
Figure 2-4: Histograms of electron collision energies at three different electric field values. Dividing by 5×10^5 (the total number of collisions) gives the probability distribution $h(\epsilon)$.

kinetic system "thermalizing".

It may appear, however, that the entropy S_B is too simplified a quantity, since it reduces the physics to a one-dimensional energy dependence. We can apply the reasoning of S.-K. Ma [23] to investigate the entropy more fully, as follows.

For the moment, consider a system prepared at a fixed energy ϵ. The entropy of this system is given by the phase-space volume through which the system's trajectory passes:

$$S = k_B \log \Omega(\epsilon). \tag{2.12}$$

Fig. 2-6 illustrates the procedure graphically, for a velocity space described by two axes: sampling the system at many times, we may find that (for example) the sample points fall roughly within the annulus pictured. The entropy, therefore, can be measured by finding the area of the annulus.

Shang-Keng Ma points out that one can estimate this area by counting the number
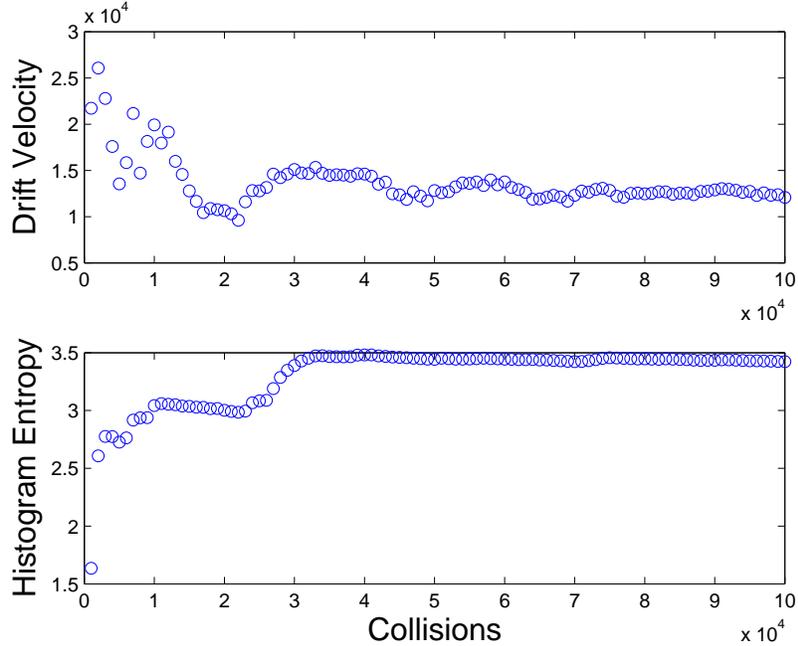
Figure 2-5: Comparison of $v_d(t)$ and $S_B(t)$ thermalization profiles.

of *coincidences*—that is, the number of times two sample points fall within the same bin. (Effects of bin size are considered below.) If we sample N_1 points and find N_2 coincidences, then the probability C_2 of coincidence is given by

$$C_2 = \frac{N_2}{\binom{N_2}{N_1}} = \frac{N_2}{N_1(N_1 - 1)/2}. \tag{2.13}$$

Elementary combinatorics can be used to generalize Eq. (2.13) to coincidences of higher order. The probability C_3 will, for example, be given by

$$C_3 = \frac{N_3}{N_1(N_1 - 1)(N_1 - 2)/6}. \tag{2.14}$$

Ma indicates that, for the fixed-energy case, the entropy is given by the simple relation

$$S = -k_B \log C_2 \qquad \text{(fixed } \epsilon\text{).} \tag{2.15}$$
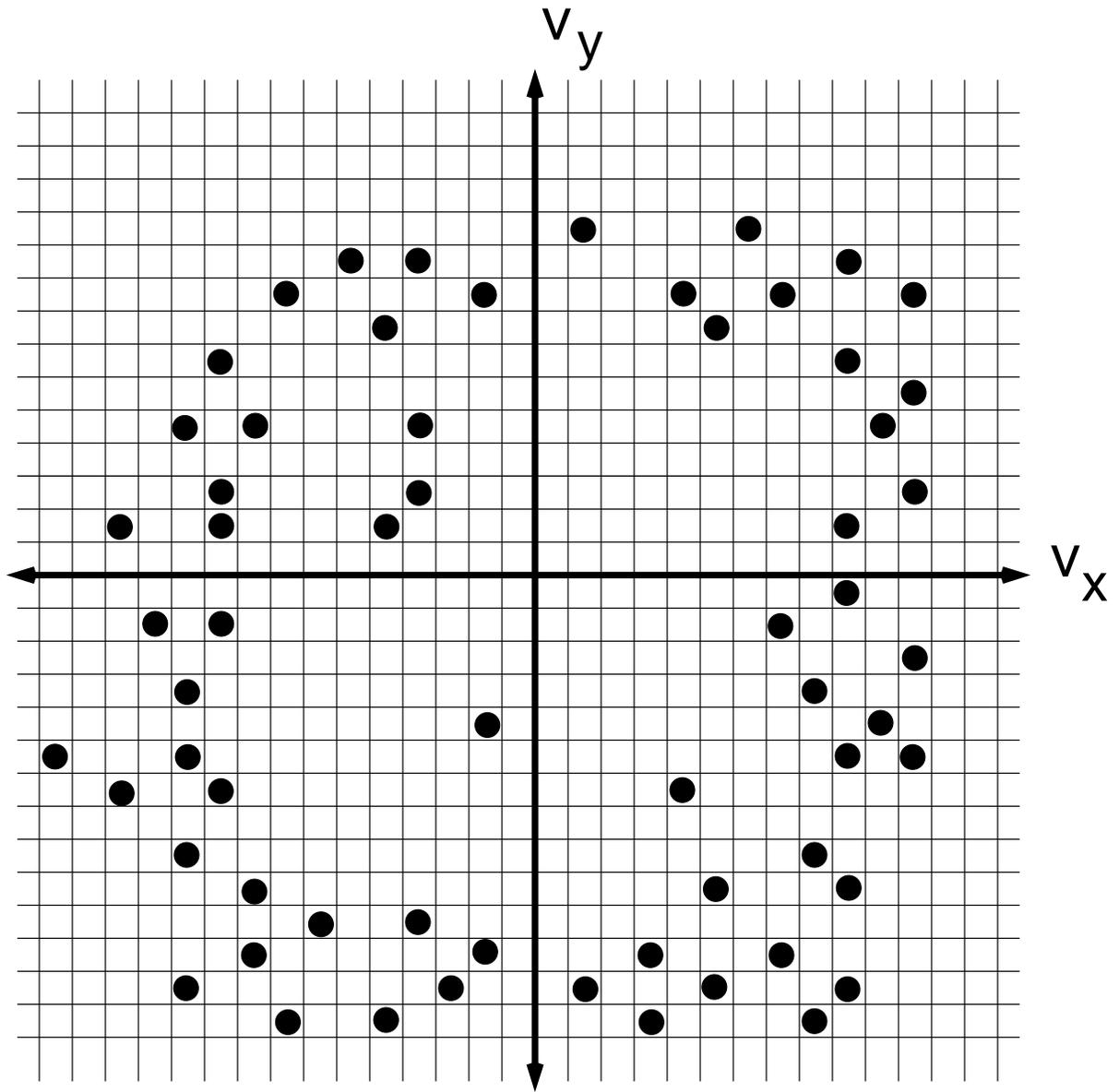
20

Figure 2-6: Shang-Keng Ma's method of measuring entropy.

Although Figure 2-6 illustrates Ma's method in two dimensions, the phase-space grid can be applied to higher dimensions as well. The MC code used in this paper (listed in the Appendix) applies it to all three dimensions of the velocity distribution.

Two caveats are worth mentioning. first, in our case we do not have a system prepared at fixed energy, as Figure 2-4 clearly indicates. Bialas and Czyz [6] generalize Ma's method to this case, which in essence requires counting coincidences of higher orders to provide correction terms to Ma's result. Following an argument they attribute to K. Zyczkowski, these correction terms can be given in terms of the Rényi entropies [29, 19], defined as

$$H_k = -\frac{\log C_k}{k - 1}.$$

(2.16)

Measuring C_2 and C_3 allows an entropy to be estimated using a formula Bialas and Czyz derive to be

$$S_M = H_2 + \frac{1 - \log 2}{\log 2 - \frac{1}{2} \log 3}(H_2 - H_3).$$

(2.17)

Here, the subscript on the "Ma-Bialas-Czyz-Zyczkowski" entropy distinguishes S_M from the S_B shown earlier.

We also note that S_M, as calculated in our MC, only reflects the velocity degrees of freedom, ignoring the positional components of the electron's phase-space trajectory. The theme of neglecting position dependence will be discussed again in Section 3.2. For now, suffice it to say that counting position-space coincidences for a particle which is, after all, being constantly driven forward presents practical and conceptual difficulties. Note that Bialas and Czyz perform the same neglect, though for somewhat different reasons. It is important to remember that both S_M and S_B are both derived quantities, of perhaps more informational than thermodynamic interest.

The MC can be instructed to compute S_M using a sample of collisions, one long enough to include a significant number of counts but short enough that the system's state does not change appreciably during the sample. Fig. 2-2 suggests a time constant on the order of 10^4 collisions, so in this paper both S_B and S_M are calculated at 500 or 1000-collision intervals.

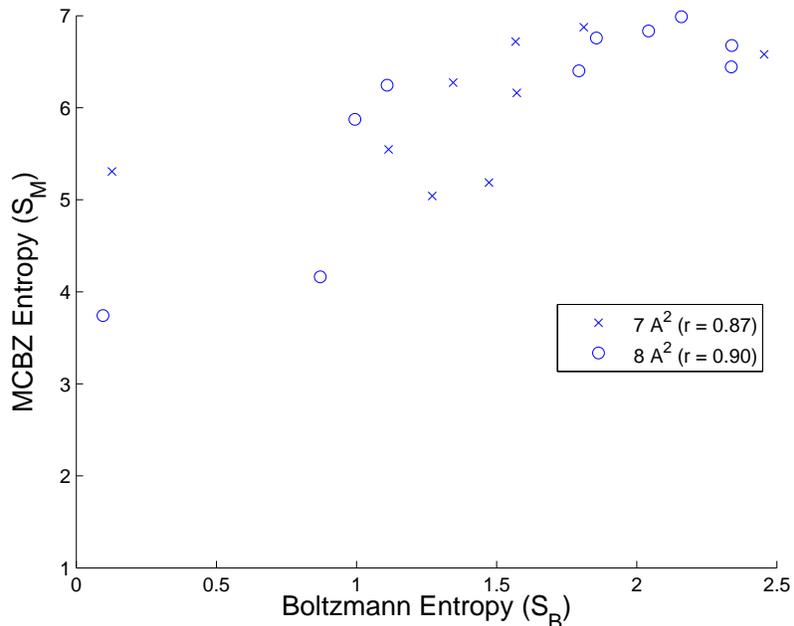Fig. 2-7 shows S_M plotted against S_B for many MC runs, calculated after 10^5

Figure 2-7: Correlation plot of S_M versus S_B. The MBCZ entropy S_M was computed using a v-space grid of $16^3 = 4096$ bins, covering a velocity range from 0 to $2.65 \times 10^6 \frac{m}{s}$ (20 eV) along each axis.

collisions. (The different runs vary the E-field strength and the value of the cross section, which was chosen to be independent of energy.) S_M and S_B are seen to be linearly correlated; knowing only the Boltzmann entropy S_B, one can compute S_M, even though S_M was derived from a higher-dimensional distribution. Note the offset between the horizontal and the vertical scales; this shift is due to the finite resolution of the velocity grid, an effect discussed in Section 6 of [6]. For this correlation analysis, the offset is unimportant.

The correlation between S_B and S_M indicates that we can treat the swarm physics in a lower-dimensional way. As the following chapter will treat in more detail, we can simplify our representation of the electron distribution, giving probabilities in terms of the *energy* (or, equivalently, the speed) rather than the velocity vector. This reduction from a vector problem to a scalar one is an important approximation in the analytic treatment of swarm kinetics, which the following chapter will develop.
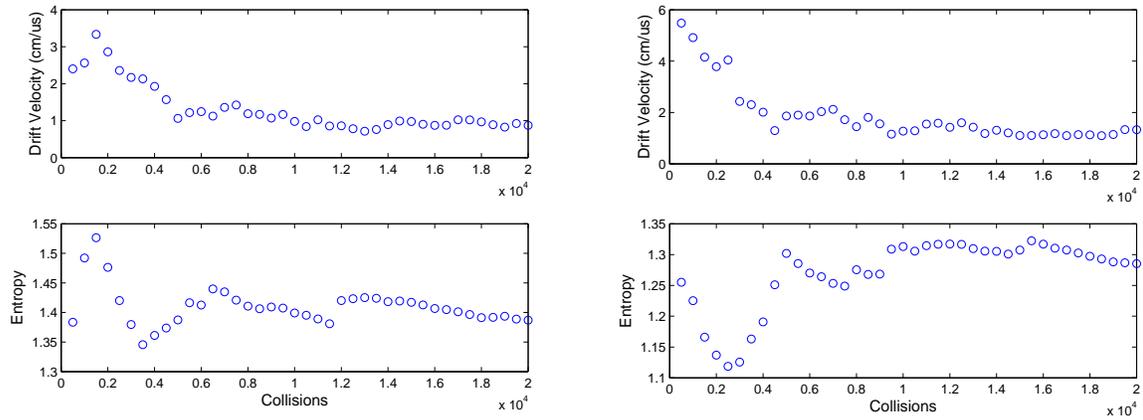
Figure 2-8: Electron-swarm thermalization, shown for simulated ^{4}He atoms. On the left, the simulated electron was subject to an applied field of $E = 1.0$ kV/cm; the electron on the right was driven by a field of $E = 1.5$ kV/cm. Note that in both cases the entropy computed using Eq. (2.11) settles to a steady value after several thousand collisions.

Chapter 3

Analytic Solution of Transport Equations

3.1 Kinetic Theory and Distribution Functions

Because the relatively simple interactions upon which our MC is built lead to simulation results which agree with experiment, it is tempting to consider an analytic treatment based on the same assumptions. MC researches face difficulties: on an abstract level, the lack of equations may produce the impression that the treatment lacks understanding. This, in part, may motivate work such as [27], where algebraic machinery is brought forth and quantities like the "temperature tensor" are defined. On a more practical level, MC calculations take *time*: a workstation must evaluate $\approx 10^5$ collisions, and even the iterations which do not involve an electron-atom impact require trigonometric manipulations, which evaluate relatively slowly. We turn, therefore, to the highly successful kinetic theory of gases, in the hope that it will provide expressions for quantities of interest (such as v_d and α) which can be evaluated faster than performing an MC run.

We consider one species of particle, the electrons, moving stochastically amongst another species, the neutral scatterers. For most of this paper, the scatterers will be hard spheres, distributed evenly on a macroscopic scale. This is a reasonable approximation for noble gases, as discussed above (see Section 2.2). Blum and Rolandi [10]

quote several key results of swarm kinetic theory, though with only minimal deriva-
tions. Fuller explanations can be found in Shkarofsky *et al.* [32] and the review arti-
cle [20]. For pedagogical treatments of kinetic theory in general, see *e.g.* Huang [18]
and Balescu [2].

The state of each electron can be specified by its position $\vec{q}$ and momentum $\vec{p}$,
which together make a six-dimensional phase space. Given a total of N_e electrons,
we can therefore describe the entire swarm in $6N_e$ dimensions. Typically, we define a
phase space density

$$\rho(\vec{p_1}, \vec{q_1}, \vec{p_2}, \vec{q_2}, \ldots, \vec{p}_{N_e}, \vec{q}_{N_e})$$

as the probability that the swarm will be found in the small region of phase space
around the point labeled by the given p and q coordinates [32]. Using $d\Gamma$ to denote
an infinitesimal portion of the $6N_e$-dimensional phase space Γ, we can define an
expectation value for any function $\mathcal{O}$ which depends upon the electron positions and
momenta:

$$\langle \mathcal{O} \rangle = \int \rho(\{\vec{p_i}, \vec{q_i}\}) \mathcal{O}(\{\vec{p_i}, \vec{q_i}\}) d\Gamma. \tag{3.1}$$

This description, however, provides too much information. Were we to compute
the pressure, for example, that the electron swarm exerts on an object, we would
only need to know how likely we are to find *any* of the N_e electrons impacting the
object with a particular velocity. In other words, the quantity of physical interest is
the *one-particle distribution,* the probability of finding any electron at location $\vec{q}$ with
momentum $\vec{p}$. As this quantity may well change over time, we must also consider its
dependence upon t. The one-particle distribution, f_{I}, is defined in terms of the total
probability density ρ:

$$f_{\mathrm{I}}(\vec{p}, \vec{q}, t) = N_e \int \prod_{i=2}^{N_e} d^3\vec{p_i} d^3\vec{q_i} \rho(\vec{p_1} = \vec{p}, \vec{q_1} = \vec{q}, \vec{p_2}, \vec{q_2}, \ldots, \vec{p}_{N_e}, \vec{q}_{N_e}, t). \tag{3.2}$$

The constant N_e in Eq. (3.2) arises from the assumption that the probability density
ρ is symmetric under exchanges of any two electrons.

We may similarly define distribution functions involving more particles, $f_{\mathrm{II}}, f_{\mathrm{III}}$,

and so on. Studying the time evolution of these distributions leads to the Bogoliubov-Born-Green-Kirkwood-Yvon (BBGKY) hierarchy, an interminably long series of equations in which the time derivative of f_I depends upon f_II, the derivative of f_II depends upon f_III and so on. Simplifying this hierarchy to a manageable equation will be the topic of the next section.

3.2 The Generalized Boltzmann Equation

Some standard approximations made in neutral-gas kinetic theory also aply here, and have been borne out by MC (see Section 2.3 above). First, the electron-atom interaction is a *local* one, a short-range process whose timescale is much less than the typical time between collisions. Second, the electrons are sufficently far apart that they do not strongly interact with one another. These approximations, valid for dilute and weakly ionized plasmas (the sort found in drift detectors), allow us to truncate the BBGKY hierarchy. The resulting relation is a generalized form of the Boltzmann kinetic equation, written in terms of the electron distribution function f_I and the gas atom distribution, denoted g:

$$\left(\frac{\partial}{\partial t} + \vec{v}\cdot\nabla_q + \frac{e}{m}(\vec{E} + \vec{v}\times\vec{B})\cdot\nabla_v\right) f_\text{I}(\vec{q},\vec{v},t) = C[f_\text{I}, f_\text{I}] + C[f_\text{I}, g]. \qquad (3.3)$$

The "streaming terms" on the left express how f_I changes over time due to the applied fields [32]. If the electron swarm propagated through free space—and if the electrons did not interact with one another—then Eq. (3.3) would reproduce Liouville's theorem, namely that the probability distribution in phase space flows as an incompressible fluid. However, these electrons are colliding with gas atoms, which intuitively means that an electron may "disappear" from one position in phase space and "reappear" elsewhere with an altered momentum. Therefore, the right-hand side of Eq. (3.3) contains *collision terms,* symbolically denoted with the C operator, which encode how the particle interactions change the swarm distribution f_I.

In the case of a dense swarm—that is, a strongly ionized plasma—we must deal

27

with *long-range* interactions, because the Coulomb potential falls off as $1/r$. Such a case is outside the realm of Eq. (3.3)'s applicability, and must instead be described by a generalized Vlasov equation, analagous to the ones written for the distribution of stars in a galaxy, since gravitational interactions are also $1/r$ potentials. (See, *e.g.*, [7] or [17].) Also, the derivation of Eq. (3.3) depends upon the "assumption of molecular chaos": it is only valid when the two-particle distribution f_{II} can be factored into independent one-particle distributions. Symbolically,

$$f_{\text{II}}(\vec{p}_1, \vec{q}_1, \vec{p}_2, \vec{q}_2) \approx f_{\text{I}}(\vec{p}_1, \vec{q}_1) \cdot f_{\text{I}}(\vec{p}_2, \vec{q}_2). \tag{3.4}$$

This assumption is the source of Eq. (3.3)'s time irreversibility. See [18] and [8]-[9] for perspectives on the loss of information implicit in this assumption.

Here, we follow the standard practice in swarm theory, which neglects the first term—electron-electron interactions—in favor of the second term, $C[f_{\text{I}}, g]$, which expresses the interactions of the light electrons with the heavy, neutral gas atoms [20]. This term can be expected to include a scattering cross-section, representing how likely collisions are to occur, as well as some knowledge of energy losses and momentum transfers, which represent the effect that collisions have on the electrons. A standard result in kinetic theory writes this collision term as an integral over the gas atom's momentum, $\vec{p}_g$. Using $\vec{p}\,'$ and $\vec{p}_g\,'$ to denote the gas and electron momenta *after* the collision (which we could in principle deduce from the specifics of the interaction potential), we can write the operator C as

$$C[f_{\text{I}}, g] = -\int d^3\vec{p}_g d\Omega \frac{d\sigma}{d\Omega} |\vec{v} - \vec{v}_g| [f_{\text{I}}(\vec{p}, \vec{q}) g(\vec{p}_g, \vec{q}) - f_{\text{I}}(\vec{p}\,', \vec{q}) g(\vec{p}_g\,', \vec{q})]. \tag{3.5}$$

Note that all functions carry the same position argument, since we have approximated that the collision occurs at the single point $\vec{q}$. (The time arguments have been suppressed for clarity.)

In principle, we would have to provide a similar formalism for the time evolution of $g(\vec{p}, \vec{q})$, the one-particle distribution for gas atoms. This is in fact not necessary for two important reasons: first, that the electrons are a thousandfold lighter than

the gas atoms, and second, that we have neglected the issues of ionization or electron attachment which may alter the gas atoms' properties after the swarm passes. As explained above, an electron's fractional energy loss per collision is small, but not entirely negligible; this condition is known as a *pseudo-Lorentz gas.* In this case, the effect of the energy loss is much more pronounced on the electron swarm than it is upon the neutral gas, which remains essentially at the Maxwell-Boltzmann distribution,

$$g(\vec{p}, \vec{q}) = \frac{N}{(2\pi M k_B T)^{3/2}} \exp\left(-\frac{p^2}{2M k_B T}\right), \tag{3.6}$$

where M is the mass of a gas atom, as above, and N denotes the number of atoms per unit volume.

Based on the conditions of the typical experimental setup, swarm theories are typically developed in the *hydrodynamic* regime, where the swarm's evolution is unaffected by boundary conditions and has no memory of its initial configuration. (In traditional kinetic theory, hydrodynamic equations govern a system where all expectation values have relaxed to *local* equilibria; the system is then expected to relax on a much longer timescale to some global equilibrium like the Maxwell-Boltzmann distribution.) Working in this regime allows us to neglect the spatial and temporal dependencies of f_I, regarding it as a function of velocity alone:

$$f_\mathrm{I} = f_\mathrm{I}(\vec{v}).$$

For the special case of isotropic scattering, which the Monte Carlo approach treated above, we may expect the distribution to be also isotropic in velocity space. This allows us to write a two-term expression for f_I:

$$f_\mathrm{I}(\vec{v}) = f_0(v) + \frac{\vec{v}}{v} \cdot \vec{f_1}(v). \tag{3.7}$$

This treatment makes intuitive sense for noble gases, and it is supported by the Monte Carlo results derived earlier. However, *it cannot be expected to hold for more complicated scattering molecules.* Recent investigations have focused on the situations

when Eq. (3.7) breaks down, necessitating that the angular dependence of $f_1(v)$ be written with an arbitrarily long expansion in spherical harmonics. This can occur in such a common instance as the inclusion of methane (CH_4) in the neutral gas [37].

We are chiefly concerned with the case that $\vec{E}$ and $\vec{B}$ are perpendicular (with $\vec{B}$ possibly zero). Eq. (3.3) can be solved in much the same way as the ordinary Boltzmann equation. Shkarofsky *et al.* give a detailed exposition, while Blum and Rolandi quote the final answer. Again, the electron distribution as a function of energy $f_0(\epsilon)$ is the exponential of minus a quantity, but here the argument is not an energy divided by $k_B T$. Instead, it is an integral over all energies, depending upon the cross section $\sigma_m(\epsilon)$ and the fractional energy loss per collision $\lambda(\epsilon)$. When the $\vec{E}$ and $\vec{B}$ fields are orthogonal, we find the distribution function is

$$f_0(\epsilon) \propto \exp\left[-\frac{3m}{2e^2 E^2}\int_0^\epsilon \lambda(\epsilon')[\nu^2(\epsilon') + \omega^2]\,d\epsilon'\right] \tag{3.8}$$

where ω is the cyclotron frequency $(e/m)B$ (see reference [1]). $\nu(\epsilon)$ depends upon the number density and the momentum-transfer (or "effective") cross section: $\nu(\epsilon) = N\sigma_m(\epsilon)v$, where $v = \sqrt{2\epsilon/m}$.

The constant of proportionality is fixed by normalization. The definition of the probability distribution implies that the integral of $f_0(v)$ over all velocities is related to the spatial density of electrons, n:

$$4\pi \int_0^\infty v^2 f_0(v)\,dv = n. \tag{3.9}$$

Changing Eq. (3.9) to an integral over energy ϵ is only a matter of changing variables. The density n is found to equal

$$n = 2\pi \left(\frac{2}{m}\right)^{3/2} \int_0^\infty \epsilon^{1/2} f_0(\epsilon)\,d\epsilon. \tag{3.10}$$

For convenience's sake, papers may graph $f_0(\epsilon)$ normalized so that the integral of $\epsilon^{1/2} f_0(\epsilon)$ equals 1.

Knowing $f_0(\epsilon)$, the generalized Boltzmann equation can be used to calculate $f_1(\epsilon)$

(see Shkarofsky for details). The result is found in terms of derivatives of f_0:

$$f_1 = -\frac{v}{\nu}\vec{\nabla}_r f_0 - \frac{e\vec{E}}{m\nu}\frac{\partial f_0}{\partial v}. \tag{3.11}$$

As a first approximation, we shall again neglect the term involving spatial derivatives, focusing on f_0's velocity dependence. With probability distributions in hand, we can calculate expectation values by performing the appropriate integrals. For any vector function $\vec{\mathcal{O}} = \mathcal{O}(v)\frac{\vec{v}}{v}$,

$$\langle\vec{\mathcal{O}}\rangle = \frac{4\pi}{3n}\int \mathcal{O}\vec{f}_1 v^2 dv. \tag{3.12}$$

The drift velocity itself is defined to be

$$\vec{v_d} \equiv \langle\vec{v}\rangle = \frac{4\pi}{3n}\int \vec{f}_1 v^3 dv. \tag{3.13}$$

Generally speaking, the electron swarm's diffusion must be defined by a matrix D_{ij} whose indices range over the $\hat{x}$, $\hat{y}$ and $\hat{z}$ axes. In terms of a velocity integral,

$$D_{ij} = \frac{4\pi}{3n}\int_0^\infty f_0 \frac{v^4}{\nu^2 + \omega^2} \begin{pmatrix} \nu & -\omega & 0 \\ \omega & \nu & 0 \\ 0 & 0 & \frac{\nu^2+\omega^2}{\nu} \end{pmatrix}, \tag{3.14}$$

which simplifies in the $\vec{B} = 0$ case to the single relation

$$\begin{aligned} D_T &= \frac{4\pi}{3n}\int_0^\infty f_0 \frac{v^4}{\nu}dv \tag{3.15}\\ &= \frac{4\pi}{3n}\int_0^\infty f_0 \frac{v^3}{N\sigma(v)}dv. \tag{3.16} \end{aligned}$$

Equivalent relations can be derived in "energy space" as well. Differentiating (3.8) gives

$$\frac{df_0}{d\epsilon} = -\frac{3m}{2e^2 E^2}\lambda(\epsilon)[\nu^2(\epsilon) + \omega^2]f_0(\epsilon). \tag{3.17}$$

The chain rule gives us a relation between this derivative and the derivative taken

with respect to electron velocity.

$$\frac{df_0}{dv} = \frac{d\epsilon}{dv}\frac{df_0}{d\epsilon} = mv\frac{df_0}{d\epsilon}. \tag{3.18}$$

By transforming the integrals to an ϵ dependence, one can find the following expressions for the drift velocity v_d and the transverse diffusion coefficient D_T:

$$v_d = -\frac{1}{3}\left(\frac{2}{m}\right)^{1/2}(eE/N)\int_0^\infty [\sigma_m(\epsilon)]^{-1}\left(\frac{df_0}{d\epsilon}\right)\epsilon d\epsilon, \tag{3.19}$$

$$D_T = (1/3N)\,(2/m)^{1/2}\int_0^\infty [\sigma_m(\epsilon)]^{-1}f_0(\epsilon)\epsilon d\epsilon. \tag{3.20}$$

Eqs. (3.19) and (3.20) are only valid in the $\vec{B} = 0$ case. Furthermore, when $\vec{B}$ is non-zero, a new quantity becomes of interest, the Lorentz deflection angle. When $\vec{E}$ is perpendicular to $\vec{B}$, the Lorentz angle takes on a computable form. The tangent of the angle α is given by the ratio of two integrals,

$$\tan\alpha = \frac{\int_0^\infty \frac{v^3\omega}{\nu^2+\omega^2}\frac{df_0}{dv}dv}{\int_0^\infty \frac{v^3\nu}{\nu^2+\omega^2}\frac{df_0}{dv}dv}. \tag{3.21}$$

It should be noted that all of these swarm parameters depend upon the electric field $\vec{E}$ and magnetic field $\vec{B}$ only through the ratios $\vec{E}/N$ and $\vec{B}/N$. A gas containing Avogadro's number of molecules at the STP volume of 22.4 litres has a number density of roughly 2.69×10^{25} molecules per cubic metre. The experiments listed in [4] report results at a slighly higher temperature, implying a somewhat lower density, $N = 2.47 \times 10^{25}$. For a typical drift-chamber electric field on the order of 1 kV/cm, E/N is on the order of 10^{-21} Vm^2. A convenient unit for E/N is the townsend (Td), which is defined to be 10^{-17} Vcm^2, or 10^{-21} Vm^2. In an analogous manner, White *et al.* define the huxley (Hx) to be 10^{-27}Tm^3, a convenient scale for the density-reduced magnetic field [36].

The integrals in Eqs. (3.8), (3.19) and (3.20) can be evaluated by hand, in certain special cases described more fully below. In the "worst case scenario", the integration can be performed numerically, using a straightforward quadrature approach. (See the

Appendix for a listing of C code which implements this idea.) However, when any of our assumptions are relaxed, the analytic expressions for swarm parameters become much more complicated—if they can be expressed at all. Including the effects of anisotropic and inelastic scattering, for example, or modeling $\vec{E}$ and $\vec{B}$ fields crossed at arbitrary angles leads to formalisms of much greater mathematical intricacy. In some models, the temperature $k_B T$ becomes a tensor [27, 36]. These developments are beyond the scope of this paper.

3.3   Special Cases

Certain special cases have been studied in which the cross-sectional dependence assumes particularly simple forms. For example, when σ_m is constant over its energy range, we obtain the Druyvesteyn distribution:

$$f_0^D(\epsilon) \propto \exp\left[-\frac{3m\lambda N^2}{2e^2 E^2}\left(\frac{\sigma_m^2}{m}\epsilon^2 + \frac{\omega^2}{N^2}\epsilon\right)\right]. \tag{3.22}$$

Also, if we have the cross section decay with increasing energy as $\sigma_m \propto \sigma_0/\sqrt{\epsilon}$, we find that the drift electrons follow a Maxwell-Boltzmann distribution.

$$f_0^M(\epsilon) \propto \exp\left[-\frac{3m\lambda N^2}{2e^2 E^2}\left(\frac{2\sigma_0^2}{m} + \frac{\omega^2}{N^2}\right)\epsilon\right]. \tag{3.23}$$

This allows us to define an electron temperature T_e:

$$k_B T_e = \frac{2e^2 (E/N)^2}{3m\lambda\left(\frac{2\sigma_0^2}{m} + \frac{\omega^2}{N^2}\right)}. \tag{3.24}$$

Assuming the Maxwellian distribution given by (3.23) and (3.24), substituting $f_0^M(\epsilon) \propto \exp(-\epsilon/k_B T)$ into (3.21) shows that

$$\tan\alpha = \frac{\omega}{\nu_0} = \frac{B}{N}\left(\frac{e}{m}\right)\sqrt{\frac{m}{2}}\sigma_0^{-1}. \tag{3.25}$$

Here we have defined ν_0 to equal the constant collision frequency, which in Maxwell's
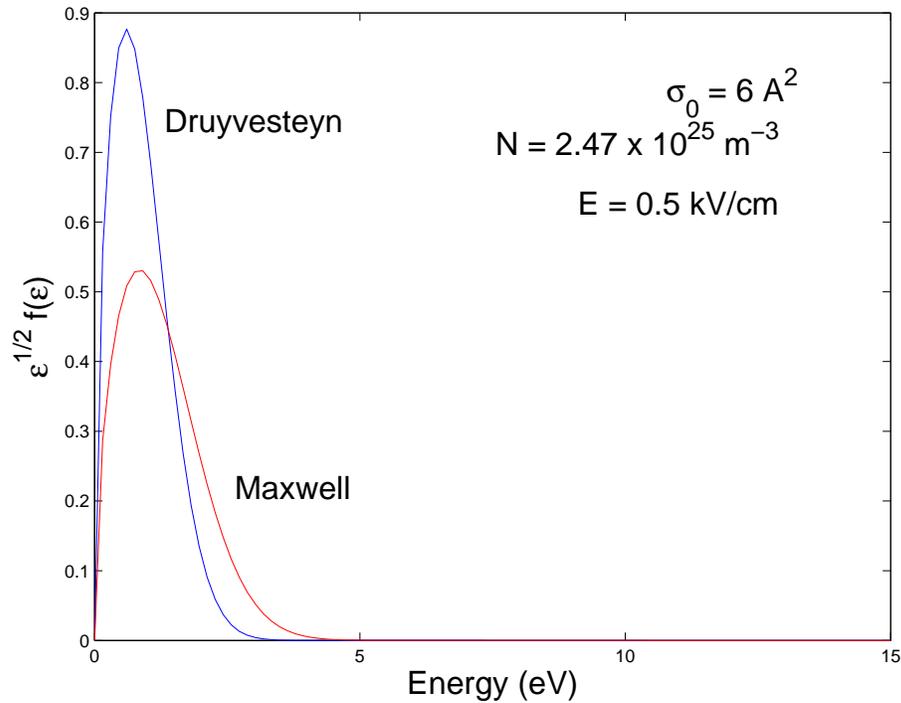
Figure 3-1: Comparison of Maxwellian (3.23) and Druyvesteyn (3.22) distributions, for a typical E-field and gas density.

distribution is independent of the velocity v. This formula makes intuitive sense: the Lorentz angle is a "compromise" between the magnetic field, which pushes the swarm off track, and the scattering effect, which limits its progression. We would thus expect the angle to increase with applied field and decrease with the cross-sectional area, and this is exactly what Equation (3.25) predicts.
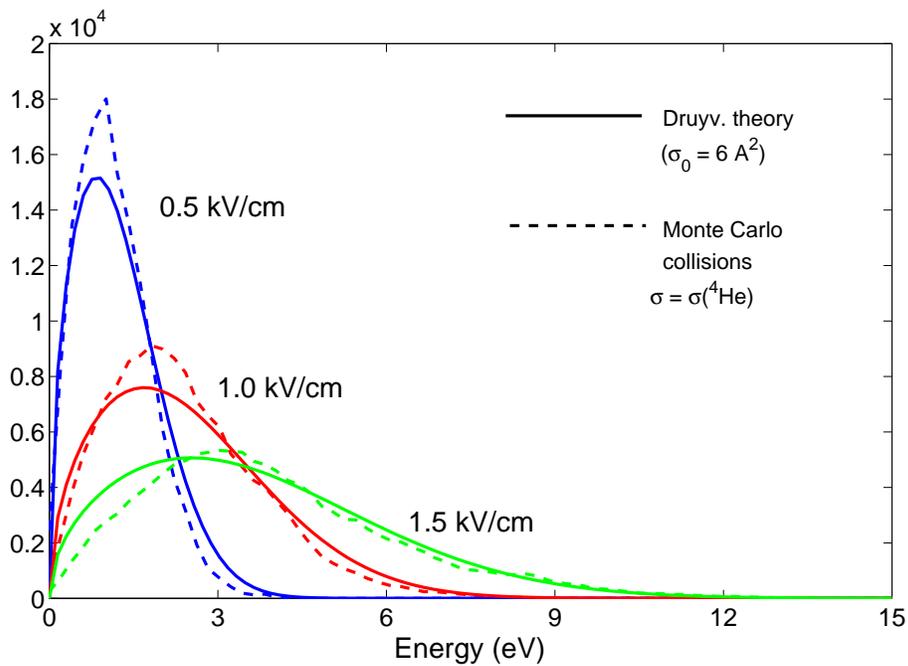
Figure 3-2: Theoretical predictions of the Druyvesteyn model compared with Monte Carlo calculations. The dashed lines indicate a histogram of collision energies, recorded during simulation runs of 5×10^5 collisions. Note that the Druyvesteyn approximation (which is an equilibrium result obtained in the hydrodynamic regime) closely follows the Monte Carlo calculation results.

Chapter 4

The Inverse Problem

4.1 Introduction

In Section 3.2, we presented formulas for calculating swarm parameters like v_d from cross sections $\sigma_m(\epsilon)$. These equations, however, are not readily invertible: we lack an algebraic way of calculating a $\sigma_m(\epsilon)$ curve given a set of v_d measurements. This is a problem of experimental interest, as evidenced by Fig. 2-3. Several approaches have been tried upon this "inverse problem", including trial-and-error [16], numerical optimization [25, 35] and neural networks [24]. In this chapter, we shall present a technique not yet applied to this problem, the Genetic Algorithm (GA), and examine its effectiveness.

In general, suppose that we have a mapping from some set of parameters A to a result B. We wish to invert the mapping and proceed from a B measured experimentally to the best possible set of parameters A. One approach, which we might try if mathematically inverting the mapping is too complicated, is to choose many different sets A, compute what B-values result from each one, and choose which A performs the best. We can then optimize from that point, using some iterative procedure to refine the answer.

The simplest realization of this idea is a grid interpolation. Fig. 2-2 suggests that such an interpolation can extract a zeroth-order approximation to the He cross section. We can estimate that σ_{He} is somewhere between 6 and 7 Å^2. However,

what happens if $\sigma_m(\epsilon)$ is not even approximately constant—say, in the presence of a Ramsauer minimum? If the $\sigma_m(\epsilon)$ curve requires more parameters to specify, we need an optimization routine which can explore that higher-dimensional parameter space, avoiding if possible the "secondary extrema"—where a particular $\sigma_m(\epsilon)$ curve is superior to its near neighbours, but inferior to the globally best-performing curve.

4.2   The Genetic Algorithm

The GA method is one way of addressing these issues. We characterize a gas by a fixed number of cross-section parameters $\{\sigma_i\}$, which in most calculations are the σ values at seven discrete energies. Because the computer can evalutate Eqs. (3.8)-(3.19) quickly, we can construct a population of these gases, 100 or more individuals in number. Initially, the $\{\sigma_i\}$ values for each gas can be chosen randomly. We assign a figure of merit, a "fitness", to each gas by computing its $v_d(\{\sigma_i\})$ values for several E fields and comparing these v_d numbers to experimental results. (A χ^2 comparison is a reasonable tool.) The code then ranks the population in order of fitness. The uniquely "genetic" step of the algorithm is the following: we produce a second generation from the first by "breeding" the gases, exchanging their $\{\sigma_i\}$ values like genes, and weighting the "fitter" gases more strongly. By repeating the evaluation and selection steps for several generations, the individuals within the population explore parameter space, settling on the maxima of the fitness function. We can avoid secondary maxima by introducing mutation operators, portions of code which randomly perturb the $\{\sigma_i\}$ "genes". There are of course many variations on all these procedures; see [13] for more elaborate discussions.

GAs are useful because they are relatively robust and problem-independent: applying a GA to a new problem essentially involves just writing a new function to perform the forward mapping. The chief disadvantage is that a GA is not likely to produce numerical results accurate to a large number of decimal places. This is an inevitable consequence of the procedure's stochasticity.

These issues and other matters of practical applicability are discussed in the User's
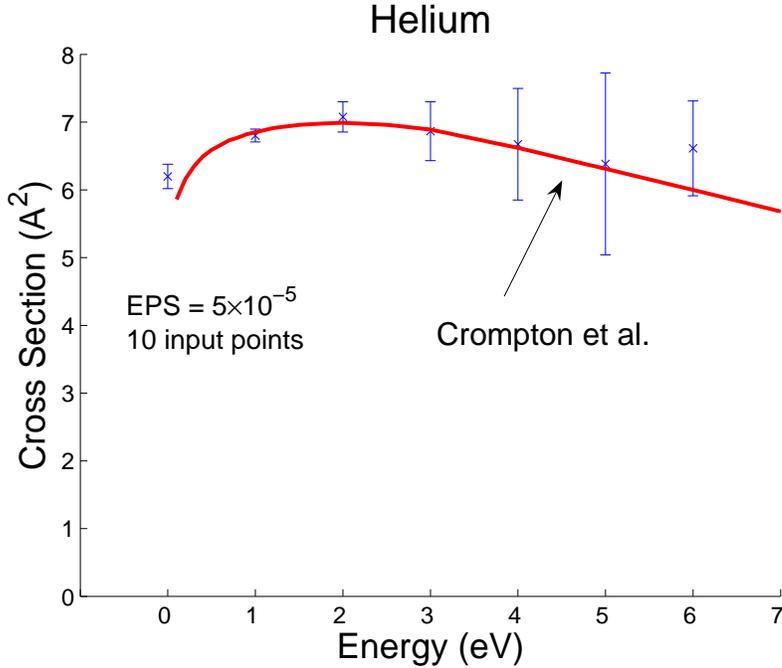
Figure 4-1: Results of GA optimization after 50 generations (data from [14]). The solid line shows the momentum-transfer cross section of Crompton *et al.*, which is in close agreement with the theoretical calculation of Nesbet [26], performed using variational methods in quantum mechanics.

Guide to the GA software used here, Charbonneau and Knapp's PIKAIA [13, 12].

4.3 GA Results for He and Xe

Figure 4-1 shows the result of applying a GA to v_d measurements conducted at 10 different E/N values. After only 50 generations, the GA is able to discover the zeroth-order (Druyvesteyn) approximation and the presence of a peak at low energies. Note that the results of the different GA runs cluster less tightly at higher ϵ, producing a larger uncertainty at 5 eV than at 2 or 3 eV. This effect has a physical cause: as Figure 3-2 indicates, there are fewer electrons colliding at those higher energies, so the system is less sensitive to the exact cross-section value in that ϵ range. *Any other optimization algorithm would suffer the same difficulty;* further refinement requires not a better algorithm but more input data.

The same procedure can be applied to xenon measurements, with data taken from
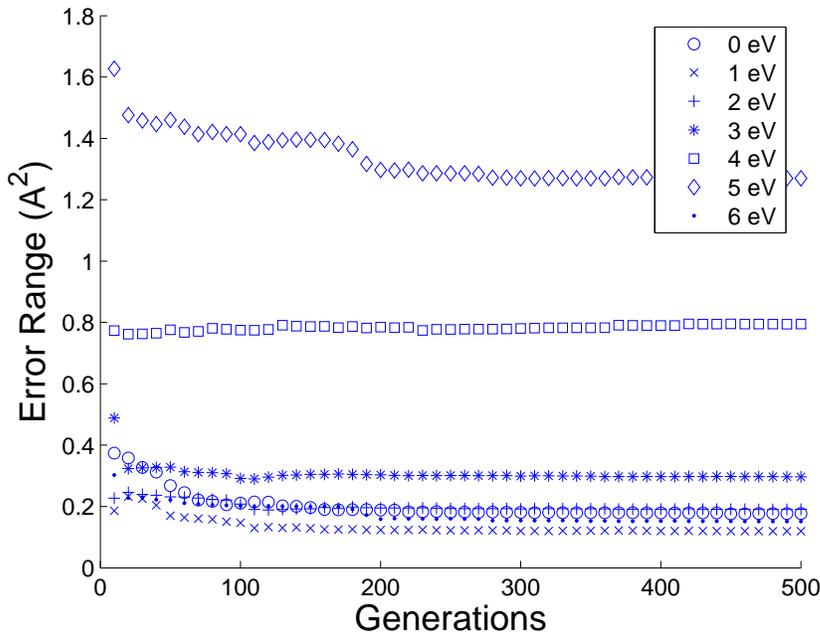
Figure 4-2: GA convergence rates, computed using the same 10 v_d values as in Figure 4-1.

(in this case) [35]. Results, again obtained after 50 generations, are shown in Figure 4-3. As the figure indicates, the GA is able to detect the Ramsauer minimum, although it is not positioned at exactly the same energy as in the literature.

Using the measurements available from the MIT LNS Drift Gas R&D website [4], the GA code can be employed to derive empirical cross sections for mixtures of gases. Results of conducting this procedure on a 90:10 mix of He and CH_4 are shown in Figure 4-4. The Ramsauer minimum in CH_4 [10] reduces the mixture's effective cross section below that of pure He. For the reasons explained above (see Section 3.2 and the discussion after Eq. (3.7)), such a $\sigma_m(\epsilon)$ curve can only be an empirical approximation.
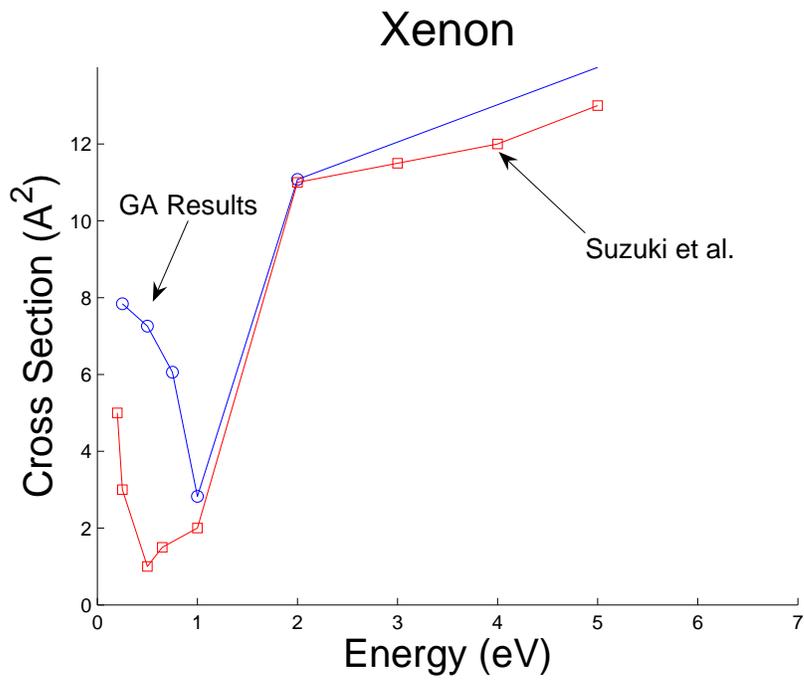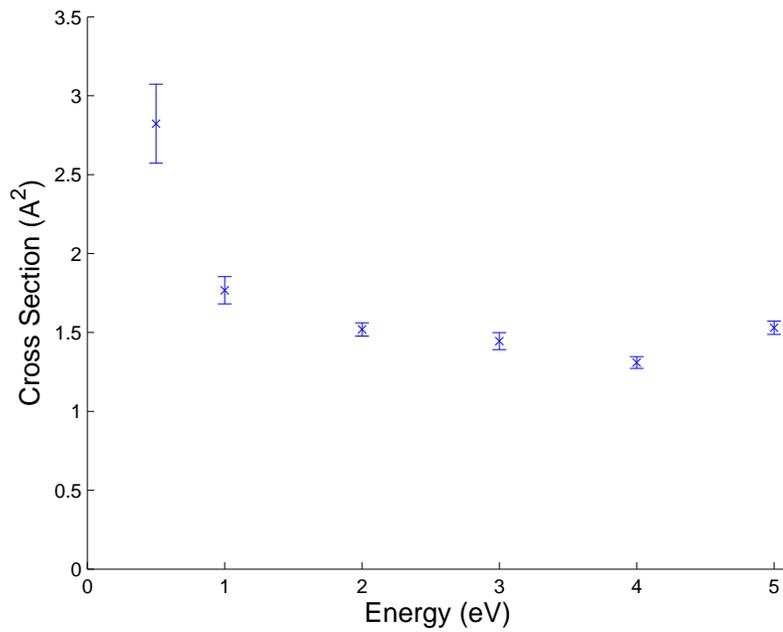
Figure 4-3: GA results when applied to Xe gas.

Figure 4-4: GA results for He:CH$_4$ (90:10).

Chapter 5

Conclusions

Ma's method of calculating MC entropies, with the refinements of Bialas, Czyz and Zyczkowski, establishes the thermalization behavior of an electron swarm moving through hard-sphere scatterers. The success of this technique—which bolsters the two-term approximation Eq. (3.7)—suggests that the same calculational tool can be applied to the cases where the two-term simplification has been seen to break down. This would involve modeling inelastic scattering in addition to the elastic process described in Section 2.2, and also generalizing Eqs. (2.5) and (2.6) to the case that $\vec{E}$ and $\vec{B}$ are not orthogonal. White *et al.* indicate that the error due to truncating the expansion is generally reduced by increasing the angle between $\vec{E}$ and $\vec{B}$, as long as the swarm motion is generally along $\vec{E}$ [36]. Computing the MBCZ entropy in these circumstances may further elucidate the effects of the two-term approximation.

In our calculations, electron energies were typically in the 1-5 eV range, which is considerably higher than the thermal energy $k_B T \approx \frac{1}{40}$eV of the scattering atoms. One could also consider swarm thermalization without a driving $\vec{E}$ field, using the MBCZ entropy to further examine results such as [31] and [11].

The GA, as implemented by the PIKAIA 1.2 code, is able to extract cross-section curves for noble gases. The only essential requirements are input data and computer time; the results of Section 4.3 suggest that a GA code could also be used with higher-term Boltzmann solutions [30, 36, 37] to derive cross sections for molecular gases like CH_4, CF_4, CO_2 and so forth.

Chapter 6

Appendix

6.1   Monte Carlo Code Listing

The file `gas.h` uses preprocessor directives to define various physical constants, in addition to quantities useful for various formulae.

```
// expressions involving physical constants
#define E_MASS      9.109e-31         // electron mass (kg)
#define E_CHARGE    1.6021765e-19     // coulombs
#define E_OVER_M    1.75882012e11     // coulombs per kilogram
#define PI          3.14159265359     // pi
#define FPI_OVER_3 4.18879020479      // 4pi / 3
#define DEG_P_RAD   57.2957795        // 180 / pi
#define RAD_P_DEG   0.0174532925      // pi / 180
#define EV_PER_J    6.2415097e18      // electron volts per joule
#define J_PER_EV    1.6021765e-19     // joules per electron volt
#define AVOGADRO    6.022142e+23      // avogadro's number
#define ANG_PER_M  1e10               // angstroms per metre
#define ANG_PER_M2 1e20               // angstroms^2 per square metre
#define F0_RATIO    53230368          // 3m / 2e^2 (kg/coulomb^2)
#define VEL_PREF    197699            // 1/3 * sqrt(2e/m)

// Monte Carlo constants
#define TIMESTEP    1e-14             // delta-t in seconds
#define MAX_COLS    100000            // maximum collisions
#define REP_INT     1000             // collisions between v,alpha reports
#define ENT_USE     500              // collisions to sample for entropy
#define EPSILON     1e-6             // a nice small number
```

```
#define NUM_COINC   2                       // we record C_2 and C_3
#define MAX_BIN     18                       // how many velocity bins we use
#define RENYI       2.133277                 // constant in an entropy formula


// macros for electron kinematics
#define OMEGA(x)    (E_OVER_M * x)           // cyclotron frequency
#define KINETIC(x)  (0.5*E_MASS*x*x)         // electron kinetic energy
#define KVEL(x)     (sqrt(2*x/E_MASS))       // velocity from kinetic energy



// stuff relating to cross sections
#define NUM_XSEC_POINTS 9                    // how many points we will use
#define MAX_XSEC_AREA   10                   // angstroms^2 which we will not exceed
#define INFINITY        20                   // electron volts
#define INFINITE_JOULES 3.204353e-18         // INFINITY in joules
#define INF_VELOCITY    2.652e6              // KVEL(INFINITE_JOULES)
#define NUM_TABLE_ENTRIES 2000
#define HE_LAMBDA   2.7e-4                    // fractional energy loss (unitless)
#define XE_LAMBDA   8.36e-6
#define DRUYV_XSEC 6e-20;                     // zeroth-order approximation to He-4
```

Also defined in `gas.h` are data structures for representing $\sigma_m(\epsilon)$ curves and electrons.

```
struct xsec_point {
  float energy;         // energy in eV
  float section;        // cross-sectional area (angstroms^2)
  float slope;          // slope of line connecting points
  int num_points;
};

struct electron {
  float pos[3];       // position in 3-space
  float vel[3];       // velocity in 3-space
  float drift_vel;    // drift velocity
  float angle;        // Lorentz angle
  long collisions;    // number of collisions sustained
  long iterations;    // number of times looped through
  float total_time;   // time spent in the gas
  float energy;
};
```

Functions related to electron behavior are contained in `gas.c`. The most sig-

nificant of these are `advance()`, which advances an electron by a given timestep, as described in Section 2.1; `iso_scatter()`, which implements the elastic scattering procedure described in Section 2.2; and `random_walk()`, which uses the two earlier functions to simulate the overall stochastic motion of one electron. `random_walk()` contains the logic to calculate the MCBZ entropy described in Section 2.5.

```
void advance(struct electron * elec, float efield, float bfield,
     float timestep) {
  // advance the given electron by timestep seconds under the influence
  // of the given fields
  // E is along x, B is along z
  float old_pos[3], old_vel[3];
  float argument = 0, omega = 0, sine = 0, cosine = 0, ratio = 0;
  int i = 0;

  for(i = 0; i < 3; i++) {
    old_pos[i] = elec->pos[i];
    old_vel[i] = elec->vel[i];
  }

  omega = OMEGA(bfield);
  argument = omega * timestep;
  cosine = cos(argument); sine = sin(argument);
  if(bfield > EPSILON)
    ratio = efield / bfield;   // ratio has units of velocity

  // update the velocity
  // the z-axis velocity doesn't change (unless by scattering)
  // while the x- and y-components are rotated into each other
  // by the magnetic field -- v_x is also affected by the E-field
  if(bfield > EPSILON) {
    elec->vel[0] = old_vel[0] * cosine + (old_vel[1] + ratio) * sine;
    elec->vel[1] = (old_vel[1] + ratio) * cosine - old_vel[0] * sine - ratio;
  } else {
    elec->vel[0] = old_vel[0] + E_OVER_M * efield * timestep;
    elec->vel[1] = old_vel[1];
  }

  // now, update the position
  elec->pos[2] += (old_vel[2] * timestep);
  if(bfield > EPSILON) {
    elec->pos[0] -= (old_vel[0] * sine - (old_vel[1] + ratio) * (1 - cosine))
      / omega;
```

```c
    elec->pos[1] += ((old_vel[1] + ratio) * sine + old_vel[0] * (cosine - 1)
      - efield * E_OVER_M * timestep) / omega;
  } else {
    elec->pos[0] += old_vel[0] * timestep
      + 0.5 * E_OVER_M * efield * timestep * timestep;
    elec->pos[1] += old_vel[1] * timestep;
  }
}

void iso_scatter(struct electron * elec, float loss_ratio) {
  // isotropic scattering means that we start the electron fresh
  // from the origin.  behavior is described by the polar coordinates
  // (v, theta, phi) where v (the velocity after scattering) is set
  // by the fractional energy loss per collision, lambda.  theta and
  // phi are random numbers.
  float v = 0, costheta = 0, theta = 0, phi = 0;

  // compute the new velocity vector's polar angles
  // 0 <= theta < pi
  // 0 <= phi < 2*pi
  costheta = 1 - 2 * drand48();
  theta = acos(costheta);
  phi = drand48() * 2 * PI;

  // compute the velocity using the energy-loss ratio
  v = KVEL(electron_kinetic(elec) * (1 - loss_ratio * (1 - costheta)));

  // convert polar coordinates to cartesian
  // v_x = v cos(phi) sin(theta)
  // v_y = v sin(phi) sin(theta)
  // v_z = v cos(theta)   (n.b. v_x^2 + v_y^2 + v_z^2 = v^2)
  elec->vel[0] = v * cos(phi) * sin(theta);
  elec->vel[1] = v * sin(phi) * sin(theta);
  elec->vel[2] = v * costheta;
}

void random_walk(struct electron * elec, struct xsec_point * points,
 float efield, float bfield, float density, float energy_loss,
 float timestep, long max_collisions, float max_len) {
  // bounce an electron around inside a gas specified by the given numbers
  // in crossed E- and B-fields.  B is defined to be along z, and E is defined
  // to be along x.
  long collisions = 0, iterations = 0, last_iterations = 0;
  long last_drift_report = 0, n2 = 0, n3 = 0;
  float energy = 0, xsec = 0, prob = 0, roll = 0, total_time = 0;
```

```c
  float h2 = 0, h3 = 0, f0 = 0, entropy = 0, cum_entropy = 0;
  float v_entropy = 0, v_cum_entropy = 0;
  float coincidences[NUM_COINC];
  long coinc_matrix[MAX_BIN][MAX_BIN][MAX_BIN];
  int indices[3];
  int i = 0, j = 0, k = 0;

  // zero the energy histogram
  for(i = 0; i < 100; i++)
    histogram[i] = partial_histogram[i] = 0;

  // zero the collision matrix
  for(i = 0; i < MAX_BIN; i++) {
    for(j = 0; j < MAX_BIN; j++) {
      for(k = 0; k < MAX_BIN; k++) {
coinc_matrix[i][j][k] = 0;
      }
    }
  }

  fprintf(stdout,"results = [\n");

  while((collisions < max_collisions)) {
    // propagate our electron
    advance(elec, efield, bfield, timestep);
    energy = electron_kinetic(elec);

    // test for a collision
    xsec = xsec_momentum_interpol(energy, points);
    prob = (xsec * density * timestep *
        sqrt(elec->vel[0]*elec->vel[0] + elec->vel[1]*elec->vel[1]
      + elec->vel[2]*elec->vel[2]));
    roll = drand48();

    // if a collision has occurred, perform a scattering operation
    if(roll <= prob) {
      collisions++;
      iso_scatter(elec, energy_loss);

      i = (int) ((energy * EV_PER_J) * 99 / 20);
      if(i > 99)
       i = 99;
      histogram[i]++;
      partial_histogram[i]++;
```

```c
      // tally the current collision in the proper place
      if(collisions - last_drift_report <= ENT_USE) {
for(i = 0; i < 3; i++) {
  indices[i] = (int) (elec->vel[i] * MAX_BIN / INF_VELOCITY
      + (MAX_BIN / 2));
  if(indices[i] >= MAX_BIN)
    indices[i] = MAX_BIN - 1;
  if(indices[i] < 0)
    indices[i] = 0;
}

coinc_matrix[indices[0]][indices[1]][indices[2]]++;
i = coinc_matrix[indices[0]][indices[1]][indices[2]];
if(coinc_matrix[indices[0]][indices[1]][indices[2]] >= 2)
  n2 += (i - 1);
if(coinc_matrix[indices[0]][indices[1]][indices[2]] >= 3)
  n3 += (i - 2);
      }
    }

    iterations++;

    // bounding box
    if((fabs(elec->pos[0]) >= max_len) || (fabs(elec->pos[1]) >= max_len)
      || (fabs(elec->pos[2]) >= max_len))
      break;

    // drift measurement report
    if((collisions > 0) && (collisions % REP_INT == 0)
      && (last_drift_report != collisions)) {
      last_drift_report = collisions;
      last_iterations = iterations;

      elec->total_time = iterations * timestep;
      elec->angle = ((atan2(elec->pos[1],elec->pos[0])) * DEG_P_RAD);
      elec->drift_vel = (sqrt(elec->pos[0]*elec->pos[0]
      + elec->pos[1]*elec->pos[1]
      + elec->pos[2]*elec->pos[2]) / elec->total_time);


      coincidences[0] = (float)n2 / (float)(ENT_USE * (ENT_USE - 1) / 2);
      coincidences[1] = (float)n3 / ((float)ENT_USE * (ENT_USE - 1)
    * (ENT_USE - 2) / 6);
```

```
        // compute the Renyi entropies h2 and h3
        h2 = -log(coincidences[0]);
        h3 = -log(coincidences[1]) / 2;

        // compute the "real" entropy from the Renyi entropies
        if(coincidences[1] > 0)
v_entropy = h2 + RENYI * (h2 - h3);
        else
v_entropy = h2;

        // zero the collision matrix
        n2 = n3 = 0;
        for(i = 0; i < MAX_BIN; i++) {
for(j = 0; j < MAX_BIN; j++) {
  for(k = 0; k < MAX_BIN; k++) {
    coinc_matrix[i][j][k] = 0;
  }
}
        }
        entropy = cum_entropy = 0;
        for(i = 0; i < 100; i++) {
if(partial_histogram[i] > 0) {
  f0 = (float)partial_histogram[i] / REP_INT;
  entropy -= f0 * log(f0);
}
partial_histogram[i] = 0;

if(histogram[i] > 0) {
  f0 = ((float)histogram[i] / collisions);
  cum_entropy -= f0 * log(f0);
}
        }
        report(collisions,iterations,elec->drift_vel,
     entropy, cum_entropy, v_entropy, h2);
    }
  }

  elec->iterations = iterations;
  elec->collisions = collisions;
  elec->total_time = iterations * timestep;
  elec->angle = ((atan2(elec->pos[1],elec->pos[0])) * DEG_P_RAD);
  elec->drift_vel = (sqrt(elec->pos[0]*elec->pos[0] + elec->pos[1]*elec->pos[1]
  + elec->pos[2]*elec->pos[2]) / elec->total_time);

  fprintf(stdout,"];\n\nhistogram = [\n");
```

```
  for(i = 0; i < 100; i++)
    fprintf(stdout," %d\n",histogram[i]);

  fprintf(stdout,"];\n");
  fflush(stdout);
}
```

6.2   Analytic Code Listing

The following three functions used to calculate the distribution function $f_0(\epsilon)$. The extended trapezoidal rule algorithm is adapted from *Numerical Recipes in C* [28]; similar logic is used to implement the other integrals given in Section 3.2 for v_d, α and so forth.

```
float distrib_integrand(float energy, struct xsec_point * points,
float energy_loss,
float efield, float bfield) {
  // compute the integrand of the distribution function f_0
  float xsec = 0, omega = 0, kvel_factor = 0;

  xsec = xsec_momentum_interpol(energy, points);

  return energy * (xsec * xsec);
}

float distrib_trapzd(float a, float b, int n,
    float efield, float bfield,
    struct xsec_point * points, float energy_loss) {
  // computes the nth stage of refinement of an extended trapezoidal rule.
  // a and b are limits

  float x,tnm,sum,del;
  static float s;
  int it, j;


  if (n == 1) {
    return (s=0.5*(b-a) * (distrib_integrand(b,points,energy_loss,
      efield,bfield)
   - distrib_integrand(a,points,energy_loss,
```

```
      efield,bfield)));
  } else {
    for (it=1,j=1;j<n-1;j++) it <<= 1;
    tnm = it;
    del = (b-a) / tnm;     // spacing of points to be added
    x = a + 0.5*del;
    for (sum=0.0,j=1;j<=it;j++,x+=del)
      sum += distrib_integrand(x,points,energy_loss,efield,bfield);
    s = 0.5*(s + (b-a) * sum/tnm);     // replaces s by its refined value
    return s;
  }
}


float distribution(float energy, struct xsec_point * points,
   float energy_loss, float efield, float bfield) {
  // Integrate up to find the distribution function
  float argument = 0, integral = 0, dummy_prefactor = 1, constants = 0;
  int j;
  float olds = 0.0;

  // compute the integral
  for (j=1;j<=JMAX;j++) {
    fflush(stderr);
    integral = distrib_trapzd(0,energy,j,efield,bfield,points,energy_loss);
    if (j > 5)
      if (fabs(integral-olds) < EPS*fabs(olds) ||
  (integral == 0.0 && olds == 0.0)) break;
    olds = integral;
  }

  // multiply the integral by the proper constant
  // assumes sigma is in A^2 and eneriges were done in eV
  constants = 3e-40 * energy_loss;
  argument = -(constants / (efield * efield)) * integral;

  // take the exponential
  return exp(argument);
}
```

The following functions wrap the functions which perform the actual numerical integration, giving a convenient interface. `analytic_walk()` takes a pointer to an electron structure (defined in `gas.h` above) and fills its data elements with the ap-

propriate values. `compare()` is used by `pikaia.h` to calculate a figure of merit, or "fitness", by comparing `analytic_walk()`'s results to experimental data.

```
void analytic_walk(struct electron * elec, struct xsec_point * points,
   float efield, float bfield, float density,
   float energy_loss) {
  float driftvel = 0, norm_factor = 0, lorentz_angle = 0, isodiff = 0;
  int i = 0;

  // scale field values by density
  efield = efield / density;
  bfield = bfield / density;

  // normalize by density
  norm_factor = normalize(efield, bfield, points, energy_loss);

  // compute the drift velocity
  driftvel = velocity(efield, bfield, points, energy_loss, norm_factor);

  // compute the isotropic diffusion coefficient
  isodiff = diffusion(efield, bfield, points, energy_loss,
        norm_factor, density);

  // compute the Lorentz angle
  lorentz_angle = angle(efield, bfield, points, energy_loss, prefactor);

  // put the calculated values into the electron struct
  elec->drift_vel = driftvel;
  elec->isodiff = isodiff;
  elec->angle = lorentz_angle;
}


float compare(float efields[], float bfields[], float velocities[],
     float isodiffs[], int num_measurements,
     struct xsec_point * points, float density, float energy_loss) {
  float sse1 = 0, sse2 = 0, difference = 0;
  float *predicted_vels, *predicted_isodiffs;
  struct electron * elec;
  int i = 0, j = 0;

  // create an electron
  elec = (struct electron *) malloc(sizeof(struct electron));
```

```c
  // size arrays
  predicted_vels = (float *) malloc(num_measurements * sizeof(float));
  predicted_isodiffs = (float *) malloc(num_measurements * sizeof(float));

  fprintf(stderr,"v_i = [");
  for(i = 0; i < num_measurements; i++) {
    // initialize the electron
    for(j = 0; j < 3; j++)
      elec->vel[j] = elec->pos[j] = 0;
    elec->collisions = elec->total_time = elec->iterations = 0;
    elec->drift_vel = elec->angle = elec->isodiff = 0;

    // bounce around inside the gas
    analytic_walk(elec, points, efields[i], bfields[i], density, energy_loss);
    predicted_vels[i] = elec->drift_vel;
    predicted_isodiffs[i] = elec->isodiff;
    fprintf(stderr," %.3e", predicted_vels[i]);
    // for ease of explication, only use the velocities right now
    difference = (predicted_vels[i] - velocities[i])
      / (velocities[i]);
    sse1 += (difference * difference);
  }
  fprintf(stderr,"];\n");
  free(elec);
  free(predicted_vels);
  free(predicted_isodiffs);

  return 1/sse1;
}
```

The code in `pikaia.h` interfaces the routines above with the PIKAIA 1.2 genetic
algorithm software. Because PIKAIA is coded in FORTRAN-77, `pikaia.h` depends
upon B. D. Steinmacher-Burow's `cfortran.h` header file [34], which provides tools
for intermingling the two languages.

```c
#ifndef __pikaia_h
#define __pikaia_h

#include "cfortran.h"
#include "gas.h"

// PIKAIA demands parameters to be within [0, 1]
#define PIKSCALE(x)    ((x - MIN_XSEC_AREA) / (MAX_XSEC_AREA - MIN_XSEC_AREA))
```

```c
#define UNPIKSCALE(y)  (MIN_XSEC_AREA + y * (MAX_XSEC_AREA - MIN_XSEC_AREA))

// PIKAIA also expects fitness values to be maximal
// at the desired point, not minimal like chi^2
#define FITNESS(x)      (1 / x)

// global variables (ick)
float efields[NUM_INPUTS];
float bfields[NUM_INPUTS];
float velocities[NUM_INPUTS];
float isodiffs[NUM_INPUTS];

float pik_energies[] = {0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 5.5};

// function prototypes
//int MAIN__(){return 0;}
float fcn (int n, float *xval);
void init_arrays();

void init_arrays() {
  // initialize the arrays holding the experimental numbers
  // which we are trying to match
  int i = 0;

  for(i = 0; i < NUM_INPUTS; i++) {
    bfields[i] = 0;
  }

  // He:CH4 (90:10)
  efields[0] = 0.196390862e5;
  velocities[0] = 0.999e4;
  efields[1] = 0.245488577e5;
  velocities[1] = 1.143e4;
  efields[2] = 0.294586293e5;
  velocities[2] = 1.274e4;
  efields[3] = 0.343684008e5;
  velocities[3] = 1.378e4;
  efields[4] = 0.392781724e5;
  velocities[4] = 1.462e4;
  efields[5] = 0.441879439e5;
  velocities[5] = 1.538e4;
  efields[6] = 0.490977155e5;
  velocities[6] = 1.597e4;
  efields[7] = 0.54007487e5;   velocities[7] = 1.637e4;
  efields[8] = 0.589172586e5;  velocities[8] = 1.685e4;
```

```
    efields[9] = 0.638270301e5;  velocities[9] = 1.72e4;
    efields[10] = 0.687368017e5;  velocities[10] = 1.738e4;
    efields[11] = 0.736465732e5;  velocities[11] = 1.76e4;
    efields[12] = 0.785563448e5;  velocities[12] = 1.778e4;
    efields[13] = 0.834661163e5;  velocities[13] = 1.806e4;
    efields[14] = 0.883758879e5;  velocities[14] = 1.809e4;
    efields[15] = 0.932856594e5;  velocities[15] = 1.824e4;
    efields[16] = 0.98195431e5;  velocities[16] = 1.838e4;
    efields[17] = 1.03105203e5;  velocities[17] = 1.854e4;
    efields[18] = 1.08014974e5;  velocities[18] = 1.872e4;
    efields[19] = 1.12924746e5;  velocities[19] = 1.875e4;
    efields[20] = 1.17834517e5;  velocities[20] = 1.903e4;
    efields[21] = 1.22744289e5;  velocities[21] = 1.898e4;
    efields[22] = 1.2765406e5;  velocities[22] = 1.921e4;
}

float fcn (int n, float *xval) {
  // returns 1 over the reduced SSE
  int i = 0;
  float inv_sse = 0;
  struct xsec_point * points;

  // construct the points structure
  points = (struct xsec_point *) malloc((n+1) * sizeof(struct xsec_point));
  for(i = 0; i < n; i++) {
    //points[i].energy = i;                        // why not?
    points[i].energy = pik_energies[i];
    points[i].section = UNPIKSCALE(xval[i]);
    points[i].num_points = n + 1;
  }
  points[n].energy = 25;
  points[n].section = 10;
  points[n].num_points = n + 1;
  compute_slopes(points);

  fprintf(stderr, "xsec = [\n");
  for(i = 0; i < n; i++) {
    fprintf(stderr," %.2f  %.3f\n", points[i].energy, points[i].section);
  }
  fprintf(stderr, "];\n");
  fflush(stderr);

  // get the sum of squared errors
  inv_sse = compare(efields, bfields, velocities, isodiffs,
    NUM_INPUTS, points, 2.47e25, HE_LAMBDA);
```

```c
    free(points);
    fprintf(stderr, "inv_sse = %.3e\n", inv_sse);
    return inv_sse;
}


// the following wrappers connect our two languages

// calling FORTRAN from C
PROTOCCALLSFSUB7(PIKAIA,pikaia,ROUTINE,INT,FLOATV,FLOATV,PFLOAT,PINT,ROUTINE)
#define PIKAIA(fcn,n,ctrl,xval,fval,status,intreport) \
CCALLSFSUB7(PIKAIA,pikaia,ROUTINE,INT,FLOATV,FLOATV,PFLOAT,PINT,ROUTINE,\
fcn,n,ctrl,xval,fval,status,intreport)

PROTOCCALLSFSUB1(PKINIT,pkinit,INT)
#define PKINIT(seed) CCALLSFSUB1(PKINIT,pkinit,INT,seed)

// calling C from FORTRAN
float pikaiafcn(int *n, float *xval);
float pikaiafcn(int *n, float *xval) {
    return fcn(*n, xval);
}
float pikaiareport(int *n, float *xval, float *fitness, int *igen);
float pikaiareport(int *n, float *xval, float *fitness, int *igen) {
    int num_points = 0, i = 0;

    num_points = *n;
    fprintf(stdout,"interim_%d = [\n", *igen);
    for(i = 0; i < num_points; i++)
        fprintf(stdout,"\t%.2f\t%.4f\n", pik_energies[i], UNPIKSCALE(xval[i]));
    fprintf(stdout,"];\n\n");
    fprintf(stdout,"fitness_%d = %.3e;\n\n", *igen, *fitness);
    fflush(stdout);
}

FCALLSCFUN2(FLOAT,pikaiafcn,PIKAIAFCN,pikaiafcn,INT,FLOATV)

#endif
```

Bibliography

[1] W.P. Allis and H.W. Allen. Theory of the Townsend method of measuring electron diffusion and mobility. *Phys. Rev.*, 52:703–707, 1937.

[2] R. Balescu. *Equilibrium and Non-Equilibrium Statistical Mechanics*. Wiley, 1975.

[3] U. Becker et al. Consistent measurements comparing the drift features of noble gas mixtures. *Nucl. Instr. and Meth. A*, 421:54–59, 1999.

[4] U. Becker et al. Gas R&D Home Page, 2005. `http://cyclotron.mit.edu/drift/www`.

[5] S. F. Biagi. Monte carlo simulation of electron drift and diffusion in counting gases under the influence of electric and magnetic fields. *Nucl. Instr. and Meth. A*, 421:234–240, 1999.

[6] A. Bialas and W. Czyz. Event by event analysis and entropy of multiparticle systems. *Phys. Rev. D*, 61, 2000.

[7] J. Binney and S. Tremaine. *Galactic Dynamics*. Princeton University Press, Princeton, New Jersey, 1987.

[8] J.M. Blatt and A.H. Opie. A new derivation of the Boltzmann transport equation. *J. Phys. A.*, 7:L113–L115, 1974.

[9] J.M. Blatt and A.H. Opie. Non-equilibrium statistical mechanics i.: the Boltzmann transport equation. *J. Phys. A.*, 7:1895–1906, 1974.

[10] W. Blum and L. Rolandi. *Particle Detection with Drift Chambers*. Springer-Verlag, 1993.

[11] I.K. Bronić and M. Kimura. Electron thermalization in rare gases and their mixtures. *J. Chem. Phys.*, 104(22):8973–88, 1996.

[12] P. Charbonneau. *Release Notes for PIKAIA 1.2, NCAR Technical Note 451+STR*. High Altitude Observatory, NCAR, Boulder, Colorado, April 2002.

[13] P. Charbonneau and B. Knapp. *A User's Guide to PIKAIA 1.0, NCAR Technical Note 418+IA*. High Altitude Observatory, NCAR, Boulder, Colorado, December 1995.

[14] R.W. Crompton, M.T. Elford, and A.G. Robertson. *Aust. J. Phys*, 23:667, 1970.

[15] G.W. Fraser and E. Matheison. *Nucl. Instr. and Meth. A*, 247:544, 1986.

[16] L.S. Frost and A.V. Phelps. Momentum-transfer cross sections for slow electrons in He, Ar, Kr, and Xe from transport coefficents. *Phys. Rev. A*, 136(6):1538–1545, 1964.

[17] M. Hénon. *Astron. Astrophys.*, 114:211, 1983.

[18] K. Huang. *Statistical Mechanics*. Wiley, 1987.

[19] P. Jizba and T. Arimitsu. Observability of Rényi's entropy. *Phys. Rev. E*, 69(2), 2004.

[20] K. Kumar. The physics of swarms and some basic questions of kinetic theory. *Phys. Rep.*, 112(5):319–375, 1984.

[21] S.L. Lin and J.N. Bardsley. Ion motion in drift tubes. *J. Chem. Phys.*, 66(2):435–445, 1977.

[22] S. Longo. *Plasma Sources Sci. Technol.*, 9:468–476, 2000.

[23] S.-K. Ma. *Statistical Mechanics*. World Scientific, 1985.

[24] W.L. Morgan. The feasibility of using neural networks to obtain cross sections from electron swarm data. *IEEE Transactions on Plasma Science*, 19(2):250–255, April 1991.

[25] W.L. Morgan. Use of numerical optimization algorithms to obtain cross sections from electron swarm data. *Phys. Rev. A*, 44(3):1677–1681, August 1991.

[26] R.K. Nesbet. Variational calculations of accurate e^--He cross sections below 19 eV. *Phys. Rev. A*, 20(1):58–70, 1979.

[27] K.F. Ness. Multi-term solution of the Boltzmann equation for electron swarms in crossed electric and magnetic fields. *J. Phys. D*, 27:1848–1861, 1994.

[28] W.H. Press et al. *Numerical Recipes in C*. Cambridge University Press, 1992.

[29] A. Rényi. *Acta. Math. Acad. Sci. Hung.*, 10:193, 1959. See also A. Rényi, "On Measures of Information and Entropy", in *Proceedings 4-th Berkeley Symposium Math. Stat. Prob.*, Vol. 1 (1961), pp. 547-561.

[30] R.E. Robson, M. Hildebrandt, and B. Schmidt. Electron transport theory in gases: must it be so difficult? *Nucl. Instr. and Meth. A*, 394:74–86, 1997.

[31] B. Shizgal and D.R.A. McMahon. Electric field dependence of transient electron transport properties in rare-gas moderators. *Phys. Rev. A*, 32(6):3669–80, 1985.

[32] I.P. Shkarofsky, T.W. Johnston, and M.P. Bachynski. *The Particle Kinetics of Plasmas*. Addison-Wesley, Reading, Massachusetts, 1966.

[33] H. R. Skullerud. The stochastic computer simulation of ion motion in a gas subjected to a constant electric field. *J. Phys. D*, 1:1567–1568, 1968.

[34] B.D. Steinmacher-Burow. `cfortran.h` header file, 1990. `http://www-zeus.desy.de/~burow/cfortran/`.

[35] M. Suzuki et al. Momentum transfer cross section of xenon deduced from electron drift velocity data. *J. Phys. D*, 25:50–56, 1992.

[36] R.D. White, K.F. Ness, R.E. Robson, and B. Li. Charged-particle transport in gases in electric and magnetic fields crossed at arbitrary angles: Multiterm solution of Boltzmann's equation. *Phys. Rev. E*, 60(2):2231–49, 1999.

[37] R.D. White, R.E. Robson, and K.F. Ness. On approximations involved in the theory of charged particle transport in gases in electric and magnetic fields at arbitrary angles. *IEEE Transactions on Plasma Science*, 27(5):1249–1253, 1999.