1991

Investigating the motions and energies of ions confined in a uniform magnetic field

Amara Lynn Graps
San Jose State University

Recommended Citation

Graps, Amara Lynn, "Investigating the motions and energies of ions confined in a uniform magnetic field" (1991). *Master's Theses*. 196.
DOI: https://doi.org/10.31979/etd.vy9s-v5bb
https://scholarworks.sjsu.edu/etd_theses/196

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 1345803

Investigating the motions and energies of ions confined in a uniform magnetic field

Graps, Amara Lynn, M.S.

San Jose State University, 1991

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

INVESTIGATING THE MOTIONS AND ENERGIES OF IONS CONFINED IN A UNIFORM MAGNETIC FIELD

A Thesis

Presented to

The Faculty of the Department of Physics

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Amara Lynn Graps

August, 1991

APPROVED FOR THE DEPARTMENT OF PHYSICS

Patrick Hamill

Dr. Patrick Hamill

Fred Witteborn

Dr. Fred Witteborn

Alejandro Garcia

Dr. Alejandro Garcia

APPROVED FOR THE UNIVERSITY

Serena W. Stanford

ABSTRACT

INVESTIGATING THE MOTIONS AND ENERGIES OF IONS CONFINED IN A UNIFORM MAGNETIC FIELD

by Amara Lynn Graps

The research described in this thesis is aimed at understanding the motion and energies of charged particles trapped in a constant magnetic field B. The study entailed writing a computer simulation of the motion of three ions confined in a constant, vertical B-field. While the simulation is general enough to allow n charged particles of any charge and mass, the cases constructed were only for hydrogen ions. Computed quantities for the ions included positions, velocities, kinetic and total energies.

The calculated positions and velocities elucidate how a system of three charged particles behave under the classical assumption of the Lorentz force. We were especially interested in the variation with particle density of the kinetic energy exchange (coupling) between the energy related to motion parallel to and orthogonal to the confining magnetic field. It is found that as the density of the three particle system decreases, the coupling decreases.

ACKNOWLEDGMENTS

This manuscript is dedicated to my father Alexander Graps. I am grateful for his love-of-life philosophy because he has shown me that the universe is ultimately a benevolent place.

There are several others that I could not have completed this thesis without their support. To Fred Witteborn, who gave me the topic of the thesis and a great deal of insight on the ions' dynamics. To Forrest Bennett who never failed to give me encouragement and a listening ear throughout my years in graduate school. To all three of my thesis advisors, who have had to endure reading this manuscript on very short notice. And, finally, to my best friend Vince Kerchner who never doubted that I would complete this and gave me a balance and perspective that I could not have gotten anywhere else.

TABLE OF CONTENTS

Appendixes

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The equivalence principle of the general theory of relativity states that the ratio of gravitational mass to inertial mass is unity for all materials, even antimatter. The equivalence principle has been verified to better than one part in 10^{11} for neutral matter (1), but has not been fully verified for the charge constituents of neutral matter. It is not expected that a different gravitational force would be exerted on ions, but if a difference is found, then the equivalence principle would be violated, and the general theory of relativity would have to be re-examined.

An experiment is currently being conducted at Los Alamos (with collaborators at the NASA Ames Research Center) that measures the Earth's gravitational effect on antiprotons in a uniform magnetic field. Since the force of gravity on charged particles is much less than the electromagnetic forces, a clever method must be devised to measure the gravitational acceleration. The clever method was devised by W. M. Fairbank and F. C. Witteborn (3) and is called the time-of-flight (TOF) method. In this method, ions are emitted from the bottom of a vertical metal tube ("drift tube"), they are guided along the axis by a very uniform, time-constant magnetic field maintained by a superconducting solenoid, and then are detected at the top of the drift tube by an ion multiplier detector. The Los Alamos experiment uses this method, and their experimental design is shown in Figure (1). The time between the emission and the detection determine the TOF of each ion in the original pulse. Since the ions have a spread in energy ΔT_z $(= (1/2)m_0\Delta v_z^2)$, some will have short flight times and some will have long flight times and a distribution in arrival time will result (4). If we assume that a constant vertical force F acts on the particles, then the maximum TOF is given by:

$$t_{\text{max}} = \sqrt{\frac{2hm_0}{F}}$$

where h is the height of the particles' path and m_0 is the mass of the ion. The slowest ions to reach the detector are those for which the total energy is equal to the gravitational potential energy m_0gh (i.e. the kinetic energy, and therefore the vertical velocity v_z, is zero). At some cut-off time $t_{\text{max}} = \sqrt{2h/g}$, no more ions will be detected; consequently, the gravitational constant can be determined from t_{max}.

Although the ions are prevented from leaving the axis of the tube by the magnetic field, the magnetic moment induced by the ions' spin and orbital motion interacts with small magnetic field gradients (10^{-5} in the free-fall experiment) to produce vertical forces much larger than gravity. Therefore, the potential energy will no longer be just m_0gh. It will have additional components due to the rotational motion of the ions.

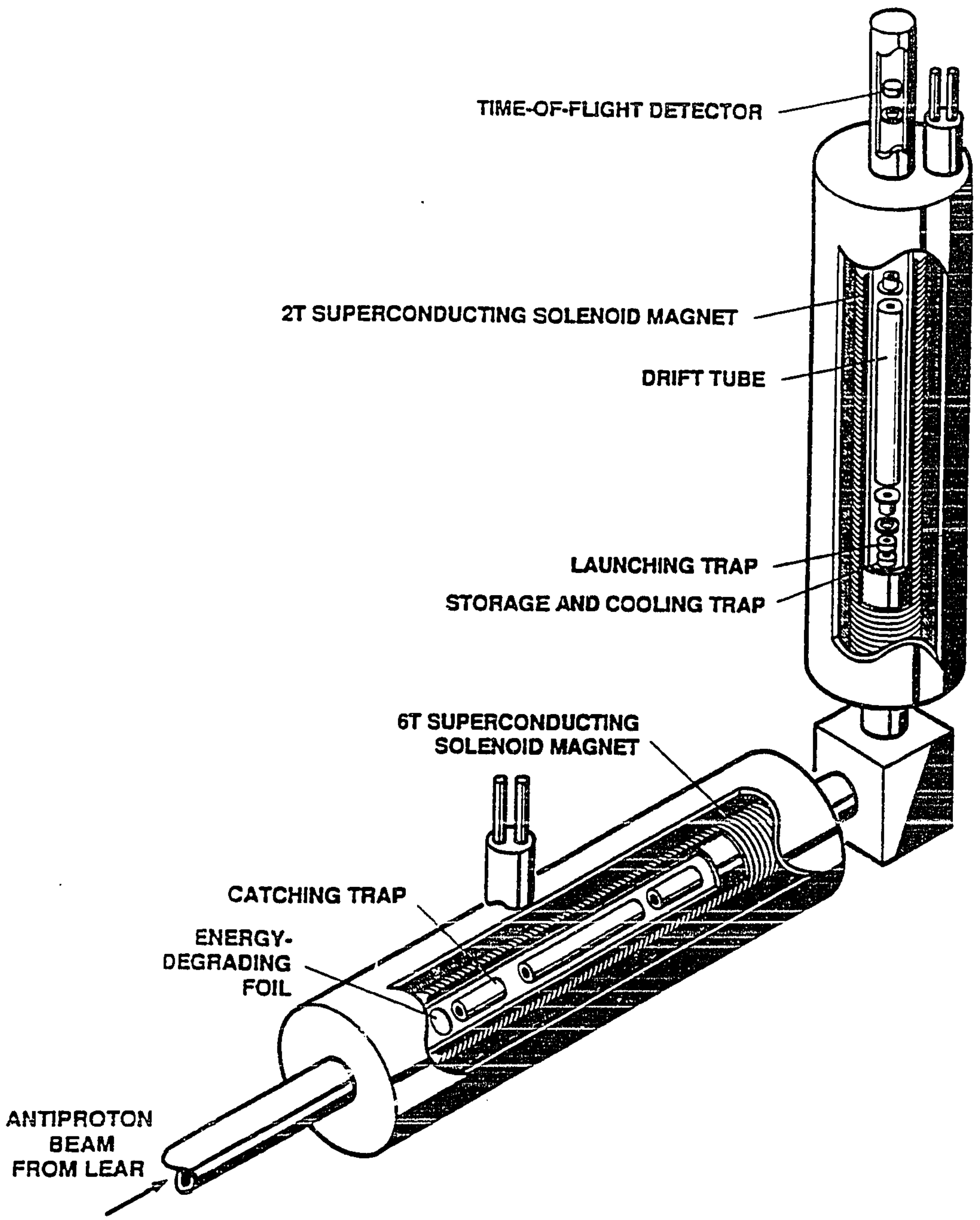

Figure 1. Los Alamos' design for their TOF experiment (2).

The research for this thesis is intended to help the Los Alamos and NASA Ames investigators by illustrating the coupling between different degrees of freedom following the release of a burst of charged particles in a magnetic field. The thesis describes a three-dimensional simulation of the interaction of three charged particles trapped in a constant B-field. When the charged particles interact, they exchange energy; specifically, the vertical z kinetic energy T_z will be transferred to rotational xy kinetic energy $T_\theta (= (1/2) m_0 v_\theta^2)$. The amount transferred will depend on the proximity of the charged particles and possibly other parameters. The result of the analysis will be a measure of the dependence of the change in T_θ with respect to the separation of the ions on the magnetic field lines. The distribution of these values will determine the requirement on the magnetic field homogeneity needed to insure that mgh is much greater than the change in potential energy experienced by an ion of velocity v_θ as the ion moves through small B-field inhomogeneities expected in the free-fall experiment. And the distribution of T_z values will determine the number of pulses required to measure t_{max}.

CHAPTER 2

Physical Theory

This section explains the theory used in the computer simulation of the interacting charged particles. Additional physical concepts are presented that will aid in the understanding of how the kinetic energy relates to the B-field and to the TOF measurement. And, finally, we answer a question of interest to the TOF experimenters on the cooling of accelerating particles. The physical theory in this section is a classical treatment, and because the simulation considers slow-moving ions, a nonrelativistic treatment.

Force Equations

The physical theory is based on the electromagnetic interaction of n ions of charge q_n. The Coulomb force acting on one charged particle q_1 due to all of the other charges $(q_{n\neq 1})$ is

$$\vec{F}_1 = \frac{q_1}{4\pi\varepsilon_0}\sum_{i\neq 1}^{n}\frac{q_i\vec{r}_{1i}}{r_{1i}^3} \quad ,$$

where $\vec{r}_{1i} = (\vec{r}_1 - \vec{r}_i)$ is their separation and ε_0 is the electrical permittivity of free space. If magnetic fields are present, then the force generated on the particle q_1 is the Lorentz force

$$\vec{F}_1 = q_1 \left(\sum_{i \neq 1}^{n} \frac{\vec{r}_{1i}}{4\pi\varepsilon_0} \frac{q_i}{r_{1i}^3} + \vec{v}_1 \times \vec{B} \right) \quad ,$$

where $\vec{v}_1$ is the velocity of particle 1, and $\vec{B}$ is the value of the magnetic induction at the location of the charge (i.e. the magnetic field strength).

The first term inside the parenthesis is the electric field strength $E_{i \neq 1}$ at $q_{i \neq 1}$. Since the Lorentz force obeys the superposition principle, the total force for n particles is, accordingly,

$$\vec{F} = q_0 \left(\vec{E} + \vec{v} \times \vec{B} \right) \quad , \tag{1}$$

where $\vec{E}$ and $\vec{B}$ are due to to charges <u>other</u> than q_0, and the total force $\vec{F}$ is the sum of the forces due to each particle that interacts with q_0.

Gyroscopic Radius

The motion of an individual ion is a helix of constant pitch around the magnetic field lines (5). This path is caused by combining the two motions of: 1) velocity parallel to B and 2) velocity perpendicular to B. To determine the radius of a particle's orbit, we look along the z-axis of the ion's motion so that its path projected on the xy plane is a circle. The radius of the circle can then be calculated by equating the magnitude of the magnetic force with the magnitude of the centripetal force

$$|\vec{F}_m| = |\vec{F}_c| \quad ,$$

$$|q\vec{v} \times \vec{B}| = \frac{m_0 |\vec{v}|^2}{R_g} \quad ,$$

where q is the charge of the ion, $\vec{v}$ is its velocity, m_0 is its mass, and R_g is its gyroscopic radius. Since $\vec{v}$ is perpendicular to $\vec{B}$ and, for circular motion, is equal to $v_\theta = \sqrt{v_x^2 + v_y^2}$ then,

$$\begin{aligned} q v_\theta B &= \frac{m_0 v_\theta^2}{R_g} \quad , \\ R_g &= \frac{m_0 v_\theta}{qB} \quad . \end{aligned} \tag{2}$$

We can also associate the gyroscopic radius with the temperature T of the ion by using the thermal kinetic energy E_k. This relationship is useful because we often know the temperature of the ions when they are released in the experiment (6). For two degrees of freedom (we do not consider the vertical velocity component), the kinetic energy in terms of temperature is

$$\begin{aligned} E_k &= k_B T \quad , \\ &= \frac{1}{2} m_0 (v_x^2 + v_y^2) \quad , \\ &= \frac{1}{2} m_0 (v_\theta^2) \quad , \end{aligned}$$

where k_B is the Boltzmann constant. We can now solve for v_θ, and it is

$$v_\theta = \sqrt{\frac{2 k_B T}{m_0}} \quad . \tag{3}$$

Upon substituting Equation (3) into Equation (2), the root-mean-squared gyroscopic radius in terms of temperature is

$$R_g = \frac{1}{qB} \sqrt{2 m_0 k_B T} \quad . \tag{4}$$

Gyroscopic Period

To ascertain whether the time steps in the computer simulation are small enough to accurately reproduce the helical motion of the ions, one can use the gyroscopic period P_g. The product of the period of an ion and its velocity is $2\pi R_g$, where R_g is the gyroscopic radius (5). However, we know R_g from Equation (2). Therefore,

$$P_g = \frac{2\pi m_0}{qB} \quad . \tag{5}$$

Relationship between T_θ and B-field

In this section I return to the central idea and purpose of the free-fall experiments remarked upon in the Introduction. One of the important problems in the ion free-fall experiments is to obtain ions with low enough energies to have flight times as long as $t_{\max}$. A pulse of ions is emitted at the beginning of each TOF measurement. The pulse expands as a result of space charge repulsion. Energy is exchanged between the central particles and the outer ones with the outer ones carrying away the most kinetic-energy-per-particle. The particles are constrained by a strong vertically-directed magnetic field (B_z). It is clear that collisions in the z-direction will exchange z-directed kinetic energy between particles and that the resulting energy distribution for the slowest particles can be determined from the TOF distribution. What is not so evident is how strongly the orbital kinetic energy (xy plane motion) is coupled to the z-motion. It is important that the xy energy be low because of the interaction of the orbital magnetic moment of the ions and gradients in the magnetic field. Therefore, we will explore the relationship between the orbital kinetic energy T_θ, the magnetic moment m, and the magnetic induction B. For a system of charged particles under the influence of electric and other forces, the relationship between T_θ, m, and B is complicated and cannot be solved analytically. Therefore, numerical solutions, such as those provided by the computer simulation written for this study, are necessary. However, for a single particle in a uniform, homogeneous magnetic field with no electric fields present, the following discussion is valid.

First, we prove that the total kinetic energy is constant. We can do so by examining the vertical velocity v_z and the rotational velocity v_θ. The total kinetic energy KE is the sum of the kinetic motion parallel to B and the kinetic motion perpendicular to B. In other words,

$$KE = \frac{1}{2}m_0 v_z^2 + \frac{1}{2}m_0 v_\theta^2 \quad .$$

The vertical velocity component v_z is unaffected by B because $\vec{v}_z \times \vec{B} = 0$. Since we have a uniform magnetic induction, v_z is constant, and so for a charge $q > 0$, it moves in the same direction as B (5).

The rotational velocity component v_θ is composed of two parts: 1) a constant speed v and 2) a drift velocity $v_\theta = v + v_D$ (5). The constant speed v gives the circular motion about the center of the orbit (or "guiding center"). The drift velocity $v_D = (\vec{E} \times \vec{B})/B^2$ is perpendicular to $\vec{B}$ and is constant also. (Note: the magnitude of v_D is independent of charge and mass of the particle.) Since v_z and v_θ are constant, and the mass of the ion is constant, then the total kinetic energy is constant.

Next, we investigate the ion's dipole moment and its relation to its kinetic energy and the B field. A point charge traveling around the circle illustrated in Figure 2 is equivalent to a current element: $I\mathrm{d}s = qv_\theta$ (5). It will therefore have a magnetic dipole moment $\vec{m}$. The dipole moment points in the direction of the vector angular velocity $\vec{\omega}_g = -(q/m_0)\vec{B}$, so, in Figure 2, is directed into the page.

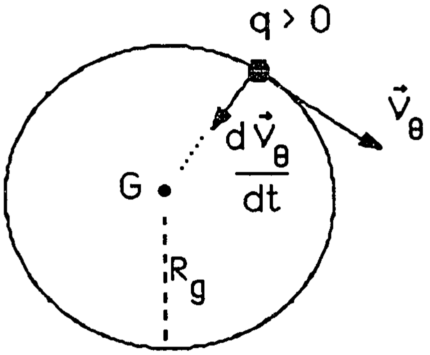

Figure 2. Circular motion of a point charge in a uniform induction $\vec{B}$ pointing out of the paper. The position G refers to the guiding center. The length R_g refers to the gyroscopic radius. The rotational velocity is indicated by v_θ.

The dipole moment of a plane filamentary current, illustrated in Figure 3, is

$$\begin{aligned}\vec{m} &= \frac{I}{2}\oint_C \vec{r}\times d\vec{s} \quad ,\\ \vec{m} &= \frac{I}{2}\int d\vec{a} \quad ,\\ \vec{m} &= \frac{I}{2}\vec{S} \quad ,\\ \vec{m} &= \frac{I}{2}S\hat{n} \quad , \qquad (6)\end{aligned}$$

where S is the total vector area enclosed by the current, I is the equivalent current, and $\hat{n}$ is the normal to its surface. The magnitude of the dipole moment is just the product of the encircling current and the area enclosed by it. The result is independent of the shape of the circuit. For a circular current path, the area $S = \pi R_g^2$, then $\vec{m} = I\pi R_g^2\hat{n}$ (5). Substituting R_g from Equation (2),

$$\vec{m} = I\pi\left(\frac{m_0 v_\theta}{qB}\right)^2 \hat{n} \quad ,$$

$$\vec{m} = 2\pi I \frac{m_0}{q^2 B^2} T_\theta \hat{n} \quad . \tag{7}$$

The equivalent current I is simply the charge divided by the gyroscopic period P_g. Since we know P_g from Equation (5), we can substitute for I into Equation (7). The result gives us a relationship between magnetic dipole moment, rotational kinetic energy and the magnetic induction:

$$\vec{m} = \frac{T_\theta}{B}\hat{n} \tag{8}$$

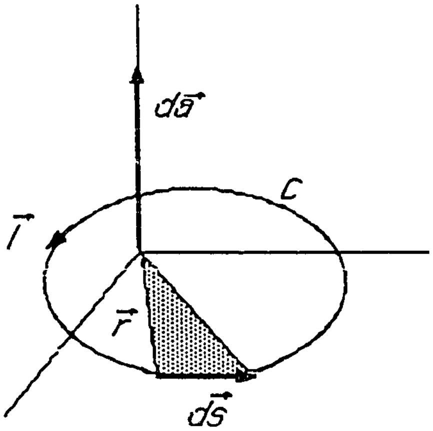

Figure 3. Plane Filamentary Current. The element of area $d\vec{a} = da\hat{n}$, C is the current loop, I is the equivalent current, and $\vec{r}$ and $d\vec{s}$ are the sides of the shaded area to compute half the area of a parallelogram ($(1/2)\vec{r} \times d\vec{s} = d\vec{a}$).

Next, we show that the magnetic moment is approximately constant. The magnitude of the flux enclosed by the circular orbit is

$$\Phi = B\pi R_g^2 \quad , \tag{9}$$

and is constant. Upon substituting R_g from Equation (4) into Equation (9), we learn that the flux can be expressed as

$$\Phi = \frac{\pi m_0^2 v_\theta^2}{q^2 B} \quad .$$

Since $T_\theta = (1/2)mv_\theta^2$, the flux is also

$$\Phi = \frac{2\pi m_0 T_\theta}{q^2 B} \quad , \tag{10}$$

which is still constant. Therefore, one consequence is that if T_θ increases, and all other values are fixed except B, then B must increase to keep Φ constant. From Equation (8), $T_\theta = mB$ where m is the magnitude of the magnetic moment. So we can substitute this expression and get a new relation for the flux

$$\Phi = \frac{2\pi m_0 m}{q^2} \quad . \tag{11}$$

Since Φ is constant, this means that the particle moves in such a way that its magnetic dipole is approximately a constant (5).

Relationship between T_z and t_{max}

This section explains the relationship between the vertical kinetic energy T_z and the maximum flight times t_{max} in the TOF experiment. Denote the number of ions arriving at the detector after time t as $N(t)$. The fraction of ions is a function of vertical kinetic energy T_z which is, in turn, a function of time (7)

$$\frac{dN}{dt} = \frac{dN}{dT_z}\frac{dT_z}{dt} \quad . \tag{12}$$

Let us assume that the distribution dN/dT_z is a constant (7). This first approximation can be made for the following reasons. We know that the number of ions $N(t)$ reaching the detector and the vertical kinetic energy T_z of the ions are both decreasing functions of time, so that $dN/dt < 0$ and $dT_z/dt < 0$. Therefore, $dN/dT_z > 0$. So, the simplest distribution $N(T_z)$, such that $dN/dT_z > 0$, is a line with a positive slope, i.e., $dN(T_z)/dT_z = C$ where C is a constant. Under this assumption, we can rewrite Equation (12) as

$$\frac{dN}{dt} = C\frac{dT_z}{dt} \quad . \tag{13}$$

Denote the total energy by W and the potential energy by PE. The rotational kinetic energy is coupled to the potential energy because as the ion moves through the external magnetic field, its own induced magnetic moment interacts with variations in the B-field to produce time-varying forces that, in turn, cause a change in the electromagnetic potential energy. Accordingly, if we absorb the rotational kinetic energy quantity into the potential energy term, then, by definition, $W = T_z + PE$, where $T_z = (1/2)m_0 v_z^2$. We wish to find t_{max}, which can be derived from v_z. So, by rewriting the definition of total energy,

$$\frac{1}{2}m_0 v_z^2 = W - PE \quad ,$$

$$v_z^2 = \frac{2}{m_0}(W - PE) \quad .$$

Then the vertical velocity is

$$v_z = \sqrt{\frac{2}{m_0}(W - PE)} \quad .$$

The velocity v_z, by definition, is dz/dt. Consequently, the previous equation can be expressed as a function of time (7),

$$\frac{dt}{dz} = \sqrt{\frac{m_0}{2}}\frac{1}{\sqrt{(W - PE)}} \quad ,$$

which can be integrated,

$$t = \sqrt{\frac{m_0}{2}}\int_0^h \frac{dz}{\sqrt{(W - PE)}} \quad , \tag{14}$$

where h is the length of the drift tube. Equation (14) is the most general relationship between t and the energies. We can simplify the relationship by considering only the case $PE \approx 0$ (7). Therefore, $W = T_z = (1/2)m_0 v_z^2$. After solving for t in this case, the vertical velocity of the ion in a (force-free) tube is h/t. We use Equations (13) and (14) to finally arrive the expressions that give us the simplest relationship between T_z and $t_{\max}$

$$T_z = \frac{m_0}{2}\left(\frac{h}{t_{\max}}\right)^2 \quad , \tag{15}$$

$$\frac{dN}{dt} = C\left(\frac{-m_0 h^2}{t_{\max}^3}\right) \quad . \tag{16}$$

Cooling of Accelerating Particles

In this section we ask a peripheral question: Since an ion radiates when it is accelerating, how long does it take to cool from 1000K to 10K? This question is of interest because, in the experiment, we want ions whose final kinetic energies are close to zero, and therefore of low temperature. The ion source emits ions with a roughly thermal distribution characteristic of 1000K. If we could reduce this temperature to 10K, we could get 1 ion (in this case, an antiproton) with an energy of 10^{-7} eV out of about 10^2–10^4 ions. Typically, many hours of data are collected to get a suitable distribuiton of ion flight times. The total rate of radiation of a charge moving in a circle of radius R with constant angular velocity $u = R\omega$ is (8)

$$\frac{dW}{dt} = -\frac{q^2 \dot{u}^2}{6\pi\varepsilon_0 c^3}\frac{1}{(1-u^2/c^2)^2} \quad , \tag{17}$$

$$= -\frac{q^2 \dot{u}^2}{6\pi\varepsilon_0 c^3}\left(\frac{W}{W_0}\right)^4 \quad , \tag{18}$$

where W is the relativistic energy: $W = W_0/\sqrt{1-u^2/c^2}$ and W_0 is the rest energy: $W_0 = m_0 c^2$. For slow-moving ions ($u/c \ll 1$), we can approximate the relativistic term with a binomial expansion (8). After neglecting terms higher than u^2/c^2, the relativistic term becomes

$$\left(1-\frac{u^2}{c^2}\right)^{-1/2} \approx 1+\frac{u^2}{2c^2} \quad .$$

The relativistic energy is now

$$W \cong W_0\left(1+\frac{u^2}{2c^2}\right) \quad ,$$

$$\cong m_0c^2+\frac{m_0u^2}{2} \quad . \tag{19}$$

Since $u = R\omega$, then

$$\left(\frac{W}{W_0}\right)^4 = \left(1+\frac{R^2\omega^2}{2c^2}\right)^4 \quad .$$

Therefore, rate of energy change from Equation (18) is

$$\frac{dW}{dt} = -\left(1+\frac{R^2\omega^2}{2c^2}\right)^4 \frac{q^2\left(R\omega\right)^2}{6\pi\varepsilon_0c^3} \quad . \tag{20}$$

The temperature is introduced by equating the ion's average velocity to $R^2/\omega^2 = 2k_BT/m_0$ where k_B is the Boltzman constant, T is the ion's temperature, and m_0 is its mass. The rate of energy loss is now

$$\frac{dW}{dt} = -\left(1+\frac{k_BT}{m_0c^2}\right)^4 \frac{q^2\left(R\omega\right)^2}{6\pi\varepsilon_0c^3} \quad .$$

Since the cyclotron frequency for this physical situation is $\omega = qB/m_0$, we can finally derive an expression for the rate of energy loss in terms of temperature

$$\frac{dW}{dt} = -\left(1+\frac{k_BT}{m_0c^2}\right)^4 \frac{q^4B^2k_BT}{3\pi\varepsilon_0c^3m_0^3} \quad . \tag{21}$$

However, we want the change in temperature over time dT/dt (9). From Equation (19),

$$\frac{dW}{dt} = k_B \frac{dT}{dt} \quad ,$$
$$-\left(1 + \frac{k_B T}{m_0 c^2}\right)^4 \frac{q^4 B^2 k_B T}{3\pi\varepsilon_0 c^3 m_0^3} = k_B \frac{dT}{dt} \quad .$$

So our desired result is

$$\frac{dT}{dt} = -\left(1 + \frac{k_B T}{m_0 c^2}\right)^4 \frac{q^4 B^2 T}{3\pi\varepsilon_0 c^3 m_0^3} \quad . \tag{22}$$

If we let $A = -(q^4 B^2)/(3\pi\varepsilon_0 c^3 m)^3)$ and $B = k_B/(m_0 c^2)$, then the differential equation to solve is

$$\frac{dT}{dt} = AT(1 + BT)^4 \quad . \tag{23}$$

The Macintosh symbolic math software Maple (10), provided a numerical solution to Equation (23) using the appropriate A's and B's for protons and electrons. For protons, A = -2.50 × 10^{-10} and B = 9.17 × 10^{-14}. The time for them to cool from 1000K to 10K is 1.85 × 10^{10} seconds which is approximately 600 years (!). For electrons, A = -1.54 and B = 1.68 × 10^{-10}. The time for them to cool from 1000K to 10K is 3.0 seconds.

CHAPTER 3

Integration Algorithms

Setting Up the Equations of Motion

Newton's second law describes the kinematics of each ion's motion as

$$\vec{a}\,(t) = \frac{1}{m_0}\vec{F}(\vec{r},\vec{v},t) = \frac{d^2\vec{r}(t)}{dt^2} \quad , \tag{24}$$

where $\vec{a}$ is the ion's acceleration, t is the time, $\vec{F}$ is the net force, $\vec{r}$ is the ion's position, $\vec{v}$ is its velocity, and m_0 is its inertial mass. Since Newton's differential equation is second-order in position, it is usually separated into two coupled first-order equations to apply computer integration algorithms to solve for each ion's motion

$$\vec{v}\,(t) = \frac{d\vec{r}(t)}{dt} \quad , \tag{25}$$

and

$$\vec{a}\,(t) = \frac{d\vec{v}\,(t)}{dt} \quad . \tag{26}$$

Because the acceleration is known from Equation (24), and the force from Equation (1), we substitute these into Equations (25) and (26) to construct the equations of motion

$$\frac{d\vec{r}_n(t)}{dt} = \vec{v}_n(t) \quad , \tag{27}$$

$$\frac{d\vec{v}_n(t)}{dt} = \frac{q_n}{m_{0n}}\left(\vec{E} + \vec{v}_n \times \vec{B}\right) \quad . \tag{28}$$

Consequently, each ion's position and velocity can be calculated by applying integration algorithms to Equations (27) and (28). Note that since the motion is calculated for three dimensions (x, y, z), we have six first-order equations for each ion. This study is concerned with the motion of three particles; therefore, eighteen first-order equations must be integrated for each time step.

The Verlet Algorithm

The first attempt at integrating the equations of motion: Equation (27, 28) was with the Verlet algorithm. This algorithm is commonly used in molecular dynamics simulations. The positions and velocities at each timestep are computed using the following iteration scheme (11)

$$x_{t+1} = x_t + v_t h + \frac{1}{2} a_t h^2 \tag{29}$$

$$v_{t+1} = v_t + \frac{1}{2} h \left(a_t + a_{t+1}\right) \quad , \tag{30}$$

where h ($= \Delta t$) is the size of the integration time step. The derivation for the Verlet algorithm is described in Appendix A. The scheme works by first calculating the new position, then updating the acceleration using this new position, and finally the velocity is calculated using both the old and the new acceleration. The following pseudo-code illustrates these steps.

```
SUB Verlet ($x$, $v$, $a$, $a_{\text{old}}$, $h$)
    Define (force) function $f$.
    $x \leftarrow x + v * h + \frac{1}{2} * a_{\text{old}} * h^2$
    $a \leftarrow f(x, v)$
    $v \leftarrow v + \frac{1}{2} * (a + a_{\text{old}}) * h$
END Verlet
```

Since the new position x_{t+1} is computed using the acceleration a_t and the velocity v_t, the Verlet algorithm is higher order in h than the more commonly-known Euler and Euler-Cromer algorithms. The global truncation error for Verlet is of order h^3 for the position and of order h^2 for the velocity. A better way to implement this algorithm to handle situations when the forces between the particles are very large and the time steps very small is to partially update the velocity using the old acceleration (11). The following pseudo-code illustrates this idea.

```
SUB Verlet ($x$, $v$, $a$, $a_{\text{old}}$, $h$)
    Define (force) function $f$.
    $x \leftarrow x + v * h + \frac{1}{2} * a_{\text{old}} * h^2$
    $v \leftarrow v + \frac{1}{2} * a_{\text{old}} * h$
    $a \leftarrow f(x, v)$
    $v \leftarrow v + \frac{1}{2} * a * h$
END Verlet
```

The Runge-Kutta 4 Algorithm

Computer integration schemes are usually applied to first-order equations of the form

$$\frac{d\vec{r}}{dt} = g(\vec{r}, t) \quad ,$$

for which the solution of $\vec{r}$ as a function of t is desired.

The best-known Runge-Kutta integration scheme is the fourth-order Runge-Kutta method (RK4). In this scheme, the function $g\left(\vec{r},t\right)$ is computed four times within each time step to produce four intermediate values k_1, k_2, k_3, and k_4. These values are then weighted and summed to complete the integration step (12):

$$\begin{aligned}
k_1 &= hg\left(t_n, \vec{r}_n\right) \\
k_2 &= hg\left(t_n + \frac{1}{2}h, \vec{r}_n + \frac{1}{2}k_1\right) \\
k_3 &= hg\left(t_n + \frac{1}{2}h, \vec{r}_n + \frac{1}{2}k_2\right) \\
k_4 &= hg\left(t_n + \frac{1}{2}h, \vec{r}_n + k_3\right) \\
\vec{r}_{n+1} &= \vec{r}_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \quad . \qquad (31)
\end{aligned}$$

The global truncation error for RK4 is of order h^4 and the local truncation error is of order h^5.

The Bulirsch-Stoer Algorithm

Again, we wish to solve the general function

$$\frac{d\vec{r}}{dt} = g\left(\vec{r},t\right) \quad ,$$

for $\vec{r}$. The Bulirsch-Stoer (B-S) method is based on a Richard Extrapolation scheme in which the step size H is divided into n smaller and smaller sub-steps h $(= H/n)$. Each integration step is performed several, perhaps many, times in an iterative fashion by probing $g\left(\vec{r},t\right)$ with different values of h as h goes to zero. The modified midpoint method is used to calculate $d\vec{r}/dt$ within the subintervals of H to solve for the area under the curve of the function and to estimate $\vec{r}_1$, the value of $\vec{r}$ at $t+H$. Next, H is subdivided into additional smaller regions, and the integration is performed again to get a second estimate for $\vec{r}$: $\vec{r}_2$. As n is increased, H is divided into even smaller regions to provide more accurate values $\vec{r}_n$. When enough estimates are available, an extrapolation rule is used to estimate what $\vec{r}_\infty$ (i.e. the value of $\vec{r}$ for an infinite number of sub-steps) would have been had h been permitted to go to zero. This $\vec{r}_\infty$ is our final "true" $\vec{r}$. At this point, the algorithm has completed one step forward in time and is now ready to begin the process all over again during the next time interval (13). The modified midpoint rule is used for integration during each sub-step because its error terms contain only even powers of h. This means that we can gain two orders of magnitude in accuracy by only halving the current time step (14).

The extrapolation rule used to estimate $\vec{r}_n$ fits the function $g(\vec{r},t)$ to a particular analytic form. Bulirsch and Stoer use a rational function of two polynomials of h, e.g.,

$$T^i(h) = \frac{p_0^i + p_1^i h^2 + p_2^i h^4 + \ldots + p_n^i h^{2n}}{q_0^i + q_1^i h^2 + q_2^i h^4 + \ldots + q_n^i h^{2n}} \quad ,$$

where i denotes the sequence of different (smaller) sub-step sizes, and p and q are the coefficients of the two polynomials. The advantage of using rational functions is that such extrapolations can remain good approximations to analytic functions even after the various terms of powers of h all have comparable magnitudes. In other words, h can be so large as to make the whole notion of the "order" of the method meaningless, and the method can still work superbly (14). Bulirsch and Stoer (15) recommend the following set of recurrence formulae to compute the rational function extrapolation values

$$\begin{aligned}
T^i_{-1} &= 0 \\
T^i_0 &= T(h_i, \vec{r}) \\
T^i_k &= T^{i+1}_{k-1} + \frac{T^{i+1}_{k-1} - T^i_{k-1}}{\left(\frac{h_i}{h_{i+k}}\right)^2 \left[1 - \frac{T^{i+1}_{k-1} - T^i_{k-1}}{T^{i+1}_{k-1} - T^{i+1}_{k-2}}\right]}, \qquad k \geq 1
\end{aligned} \tag{32}$$

where k indicates the number of terms desired in the rational function. The elements T^i_k can be arranged in a table, illustrated in Figure 4, in which the first column is used to get the recurrence started, the second column contains the values of the function $\vec{r}$ calculated by the midpoint rule for different h's, and the third and subsequent columns are calculated by the recurrence relation (Equation (32)).

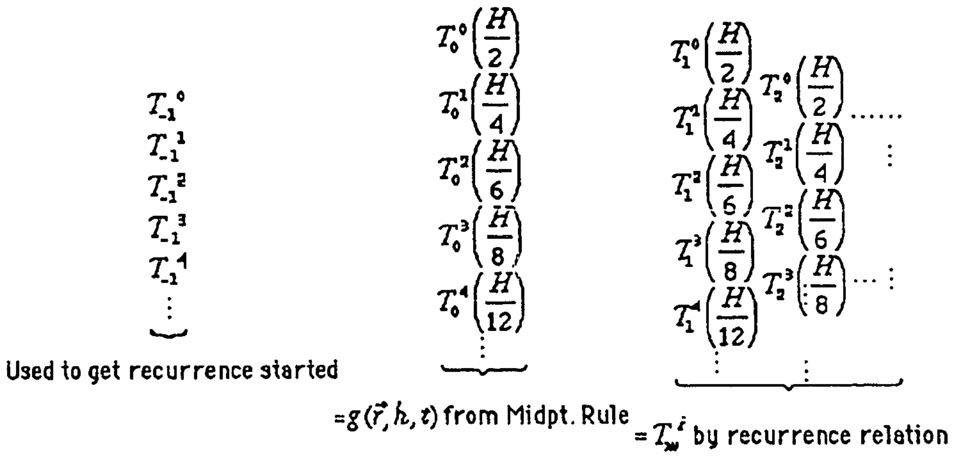
T_{-1}^0
T_{-1}^1
T_{-1}^2
T_{-1}^3
T_{-1}^4
$T_0^0\left(\frac{H}{2}\right)$
$T_0^1\left(\frac{H}{4}\right)$
$T_0^2\left(\frac{H}{6}\right)$
$T_0^3\left(\frac{H}{8}\right)$
$T_0^4\left(\frac{H}{12}\right)$
$T_1^0\left(\frac{H}{2}\right)$
$T_1^1\left(\frac{H}{4}\right)$
$T_1^2\left(\frac{H}{6}\right)$
$T_1^3\left(\frac{H}{8}\right)$
$T_1^4\left(\frac{H}{12}\right)$
$T_2^0\left(\frac{H}{2}\right)$
$T_2^1\left(\frac{H}{4}\right)$
$T_2^2\left(\frac{H}{6}\right)$
$T_2^3\left(\frac{H}{8}\right)$
Used to get recurrence started
$=g(\vec{r}, h, t)$ from Midpt. Rule
$= T_w^i$ by recurrence relation

Figure 4. Recursion Elements in the Bulirsch-Stoer Algorithm.

We work through this table by trying a set of h's in column two, calculating the T's by the midpoint rule, and then extrapolating. The extrapolation returns error estimates, and if the errors are not satisfactory we try a new h by stepping down one row in the second column and repeating the previous step. In effect, each new result from the sequence of midpoint integrations extends the table by adding one diagonal. Since the extrapolation gives us the closest approximation to $\vec{r}$, the error estimates can be calculated by comparing two successive approximations of T within the same time step (i.e. in the same row). The process is stopped (and the current value of h is accepted as a sub-step size), when the two approximations are within a user-defined value *eps* which is input to the integration program.

The utility of this method depends upon its computational cost. Each forward step of size H is much more complicated to perform than that of the classical Runge-Kutta, Verlet or Euler-Cromer integration methods. In addition, each step requires many more evaluations of each differential equation. Therefore, use of the B-S method can only be justified when it takes significantly fewer steps to produce the desired results.

Comparison of Integration Methods

The three integration methods mentioned above were compared by insertion into the computer program Colliding Ions (introduced in Chapter 5) in which the interaction of three charged particles moving within a constant magnetic field is simulated. The initial conditions were those for "Case 1," listed in Table 2 of Chapter 6, and the stopping criterion was when the outer ions were at 1.1 times their initial separation. The simulation was performed once for each of the Verlet, Runge-Kutta 4, and Bulirsch-Stoer methods, and the relevant differences between each run of the program are presented in Table 1.

Table 1. Comparison of Integration Methods.

Test	Verlet	Runge-Kutta 4	Bulirsch-Stoer
Run time (min)	42	43	37
Simulation time (sec)	5.53×10^{-6}	1.10×10^{-5}	1.09×10^{-5}
Step size (sec)	5.00×10^{-10}	1.00×10^{-9}	1.83×10^{-8}
Number of steps	11,060	11,000	738
Energy change (percent)	1.17×10^{8}	-0.36	0.28

All simulations were performed on a Macintosh IIcx computer, which includes a floating point hardware chip. The Verlet simulation could not be completed, as it exhibited numerical instability during the first third of the run, so that the values in this column of the table are extrapolated from the run prior to the onset of instability. The step size for the Verlet and Runge-Kutta 4 methods was a constant, while that reported for the Bulirsch-Stoer is the average value of the variable step size chosen by the routine. Note that while the run time and accuracy of the RK4 and B-S methods are almost equivalent, the RK4 method performed almost 15 times as many iterations. Also, an order of magnitude difference in step size was required so that the RK4 method produced roughly the same accuracy as the B-S method. The Verlet method was unsuitable for this simulation, as an additional order of magnitude reduction in step size (and consequent increase in number of steps and run time) would have been required to eliminate the numerical instability. The energy change indicates how well the total energy was conserved during the run, and is calculated as the difference in total energy over the run divided by the initial total energy.

CHAPTER 4

Variable Time Step Methods

As discussed in Chapter 3, two other integration methods (Verlet and RK4) were attempted before the B-S method was used. Both of these methods use a fixed time step, so that computing all of the x, y and z particle positions and velocities using the small time step required for accuracy within the closest region of interaction would have taken an inordinate amount of computing time. Therefore, a variable time step method was needed to economize on the computational resources. In the first of such methods, only the ions' z motion was computed until some physical quantity indicated that the ions were close enough to strongly interact. At that point, the full x, y and z motions would be calculated. The physical quantities used to indicate when to begin calculating all three coordinate positions and velocities were 1) the relative particle velocity changes and 2) the magnitudes of the electric and magnetic forces between particles.

The scheme for using the relative velocity changes of the ions to alter the step size was: if the velocity change for any of the ions was less than one one-hundredth of a percent, then the time increment was increased by a factor of ten ($\Delta t \ngtr 10^{-5}$ sec), and the xy motion was not calculated. Alternatively, if the change was greater than a percent, then the time increment was decreased by a factor of ten ($\Delta t \nless 10^{-12}$ sec), the simulation was backed up by a single time step, and the x and y positions and velocities were calculated. This scheme, unfortunately, did not allow the particles to approach very closely to one another before beginning the x and y motion calculations, so that a large amount of unnecessary computation was almost always performed, and this method proved unsatisfactory.

The second variable time step scheme was sensitive to the magnitudes of the electric and magnetic forces between particles. In this scheme, three conditions determined the value of the time step. If the electric force magnitude was greater than or equal to a constant ("K") times the magnetic force magnitude, then the simulation would revert by one time step, would set Δt to a small value such as 10^{-9} sec, and then calculate all three cartesian positions and velocities. If this condition was not true, then the time step was increased by a factor of ten ($\Delta t \ngtr 10^{-5}$ sec), and only the z positions and velocities were calculated. Finally, if the ions repelled or passed by one another, the simulation would go back a time step, set Δt to a value such as 10^{-9} sec, and then calculate all three cartesian positions and velocities until the particles were farther apart than their initial separation. One difficulty with this scheme was that the two forces were often different by many orders of magnitude, and that these differences were directly related to the initial velocities of the ions. It therefore became necessary to empirically determine an appropriate value for K based on the initial velocities, so that all of the x, y, and z motions could be computed when the particles were near to one another. With this caveat, this scheme proved to be too cumbersome to be effective.

The beauty of the variable time step algorithm embedded in the B-S method is that it can automatically make fine adjustments to the time steps (by 5–20 percent) based on the strength of the forces between the ions. The two previous attempts at a variable time step method provided only crude adjustments of an order of magnitude.

CHAPTER 5

The Program Colliding Ions

The ions' motions were numerically simulated with a computer program I wrote called Colliding Ions. This program was written with the compiler Think Pascal (16) and runs on a Macintosh computer. Appendix B contains the listing for the main procedure in Colliding Ions. This program calculates the positions and velocities for n charged particles trapped in a constant magnetic field. The positions and velocities are computed by integrating Equations (27, 28) which are, in turn, derived from the Lorentz force: $\vec{F} = q(\vec{E} + \vec{v} \times \vec{B})$. The electric field is calculated from Coulomb's Law,

$$\vec{E}(\vec{r}) = \frac{1}{4\pi\varepsilon_0} \sum_{i=1}^{n} \frac{q_i\,(\vec{r} - \vec{r}_i)}{|\vec{r} - \vec{r}_i|^3} \quad .$$

Since the Lorentz Force satisfies the superposition principle, all of the forces are added for each charged particle. In addition, since Colliding Ions computes cartesian position coordinates: x, y, and z and the cartesian velocities: v_x, v_y, and v_z, eighteen first-order equations are integrated for the three charged particles. The first-order equations are integrated using the Bulirsch-Stoer integration scheme described in Chapter 3. Colliding Ions reads the necessary initial values from an external file called CollidingIonsIn.text. The file contains necessary parameters for the three-dimensional graphics such as look angles, pitch, yaw, roll, focus, size of the axes, magnification for the points, pen patterns for the points and screen background, plus the necessary parameters for the Bulirsch-Stoer scheme such as time, time step, *eps*, the maximum number of columns to use in the extrapolation table, and the maximum number of time steps to try for each step. And, above all, this input file contains the physical parameters for the charged particles such as initial position, velocity, charge and mass. The output of the program is to the screen in the form of a three-dimensional plot, and to three files that contain the computed positions, velocities, and energies for each time step. Appendix C furnishes the CollidingIonsIn.text file for one set of initial conditions, Case 1 listed in Table 2, as an example. The program interface is a fully implemented Macintosh interface with a special borderless window and menu commands that tell the program when to start, stop, or resize the window.

CHAPTER 6

Results

Sixty-three cases with different initial values were run using the Colliding Ions program. All of the cases tested the dynamics of an ion moving up a magnetic field line (particle 1), an ion moving down a field line (particle 3), and a third ion (particle 2) placed in the middle given a slight velocity downward. Figure 5 illustrates the initial dynamics. All three ions had the same mass and charge of 1.673×10^{-27} kg and 1.60×10^{-19} coulombs, respectively, to simulate proton interactions. I varied the initial x separation of particles 1 and 3 so that it ranged from 10^{-3}–10^{-6} meters in order to learn how the resulting motions and kinetic energies changed as a function of particle density.

The initial values for the cases in this study were designed to test the dynamics of relatively slow-moving ions. Their initial z separations and v_z velocities were chosen to illustrate a close interaction between the three ions without too much computation time. Each case took 45–75 minutes on a Mac IIx computer. If the ions were given a fast velocity towards each other, then they would have been more likely to slip by each other with very little interaction. If the ions were given a much slower velocity towards each other, then the computation time would have taken at least an order of magnitude longer. The initial z separation between particles 1 and 3 of only 10^{-3} meters allowed the interaction between the three particles to happen fairly quickly.

If we placed the particles further apart at > 0.1 meters, then the computation would easily have taken days or weeks for each case. If, on the other hand, we placed them closer together, then we would have run into the danger of failing to determine the full exchange between the rotational and z-directed kinetic energy.

From the sixty-three cases, ten cases were sufficient to demonstrate all of the interesting dynamics. The ten cases were selected so that the x initial positions of the outer two particles (Ions 1 and 3), were symmetric about 0. We only varied x because, by symmetry, changes in initial x and y should produce equivalent results. This assumption was tested and proved correct with thirteen out of the original sixty-three cases. Tables 2 and 3 list the initial positions and velocities of the ten sample cases.

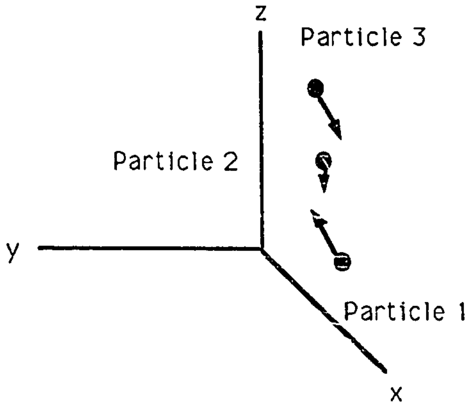

Figure 5. Initial Dynamics for Three-Particle Cases. The magnetic induction B is constant and equal to 2 tesla in the $z > 0$ direction. Particles 1 and 3 are each given a v_z of 100 m/s towards each other. And to start the dynamics with an initial rotational velocity, particle 1 is given v_x=100 m/s, and particle 3 is given v_y = 100 m/s. Particle 2 is only given a v_z = 1 m/s downward. All three have the same mass and charge of 1.673 x 10^{-27} kg and 1.60 x 10^{-19} C, respectively. The initial x separations of particles 1 and 3, the v_x velocity of particle 1, and the v_z velocity of particle 2 (middle one) were varied to learn how the resulting motions and kinetic energies changed.

Each of the sixty-three cases ran until the separation between the outer particles was 1.1 times their initial separation. The conservation of total energy for all cases was excellent, ranging from 0.0008 percent to 0.8 percent. The average relative total energy change was 0.00281 ± 1.47×10^{-5}.

Table 2. Initial Positions for Ten Sample Cases.

Run Label	Ion 1 (x,y,z) m	Ion 2 (x,y,z) m	Ion 3 (x,y,z) m
Case 1	(0, 0, 0)	(0, 0, 5.0×10^{-4})	(0, 0, 1.0×10^{-3})
Case 58	(-3.0×10^{-7}, 0, 0)	(0, 0, 5.0×10^{-4})	(3.0×10^{-7}, 0, 1.0×10^{-3})
Case 59	(-1.0×10^{-6}, 0, 0)	(0, 0, 5.0×10^{-4})	(1.0×10^{-6}, 0, 1.0×10^{-3})
Case 60	(-3.0×10^{-6}, 0, 0)	(0, 0, 5.0×10^{-4})	(3.0×10^{-6}, 0, 1.0×10^{-3})
Case 61	(-1.0×10^{-5}, 0, 0)	(0, 0, 5.0×10^{-4})	(1.0×10^{-5}, 0, 1.0×10^{-3})
Case 62	(-3.0×10^{-5}, 0, 0)	(0, 0, 5.0×10^{-4})	(3.0×10^{-5}, 0, 1.0×10^{-3})
Case 63	(-6.0×10^{-5}, 0, 0)	(0, 0, 5.0×10^{-4})	(6.0×10^{-5}, 0, 1.0×10^{-3})
Case 50	(-1.0×10^{-4}, 0, 0)	(0, 0, 5.0×10^{-4})	(1.0×10^{-4}, 0, 1.0×10^{-3})
Case 51	(-5.0×10^{-4}, 0, 0)	(0, 0, 5.0×10^{-4})	(5.0×10^{-4}, 0, 1.0×10^{-3})
Case 52	(-1.0×10^{-3}, 0, 0)	(0, 0, 5.0×10^{-4})	(1.0×10^{-3}, 0, 1.0×10^{-3})

Table 3. Initial Velocities for Ten Sample Cases.

Run Label	Ion 1 (v_x,v_y,v_z) m/s	Ion 2 (v_x,v_y,v_z) m/s	Ion 3 (v_x,v_y,v_z) m/s
(All Ten Cases)	(100, 0, 100)	(0, 0, -1)	(0, 100, -100)

Motion

The ten sample cases demonstrate how the particles interact with each other when the outer two ions (particles 1 and 3) are initially placed along the same field line, and then, symmetrically in x, moved gradually further apart. The outer two ions have the helical motion described in Chapter 2. The middle particle initially has no cyclical motion, but the torques imparted by the outer two gives it one after a few iterations.

The ten cases can be classed into two kinds of behavior. The first five (Cases 1, 58, 59, 60, and 61 listed in Table 2) show the first kind of behavior. In this, particles 1 and 3 move towards each other, gradually slow down, then repel. The middle particle, like the top particle, also moves towards the bottom ion (particle 1) and repels. However, it then picks up speed and repels from the top ion.

The last four cases (Cases 63, 50, 51 and 52 listed in Table 2) demonstrate the second kind of behavior. In this result, the outer particles do not repel from each other. They only slow down and then continue on their original path. The middle ion in Cases 63 and 50 still repels from the bottom and top ion as before, but then in the last two cases (Cases 51 and 52) it also continues on its original path without repelling.

One case has not yet been discussed, and that is Case 62. Its behavior falls appropriately in the middle of the two classes described above. In this result, Ion 3 and the middle particle, Ion 2, switch places, so that Ion 3 becomes the middle particle. All three particles repel with Ions 1 and 2 repelling from the middle Ion 3, and the middle Ion 3 repelling from the two outer ions.

Figures 6 through 15 illustrate the three-dimensional paths and z-direction velocities for the ten sample cases. I chose to plot the outer Ions 1 and 3 on the same graph, so as their initial separation increased, the x and y axes in the subsequent cases had to be stretched to accommodate the two ions. This is the reason the three-dimensional paths for the last several cases do not show the ions' gyroscopic motion.

Gyroscopic Radius

We utilize the computed xy positions of all three particles to check the theoretical R_g value of Equation 2 by first examining the gyroscopic radius of the outer ions: particles 1 and 3. The theoretical value for both Ions 1 and 3 is 5.228×10^{-7} meters. Since the R_g was derived based on the dynamics of a single ion spiralling along a field line, we would expect the theoretical value to hold initially, and then, perhaps much later, when no other interactions are occurring.

For the ten sample cases, a rotational radius $r(t)$ for each particle was calculated for each $(x(t), y(t))$. The average of those $r(t)$'s ($=R$) is illustrated in Figures 16 and 17 for Ions 1 and 3, respectively. The first few points of $R = 5.19 \times 10^{-9}$ m are reasonably close to the theoretical value. They are slightly less, perhaps due to the interaction with the middle particle. Then the radius values, as a function of $|x|$-separation between the outer ions, traverse through a regime where large excursions in the radius occur, resulting in large error bars. A plausible explanation for this is discussed below. Then, finally, the radius values stabilize at $R = 5.226 \times 10^{-9}$ m which is almost exactly the theoretical gyroscopic radius.

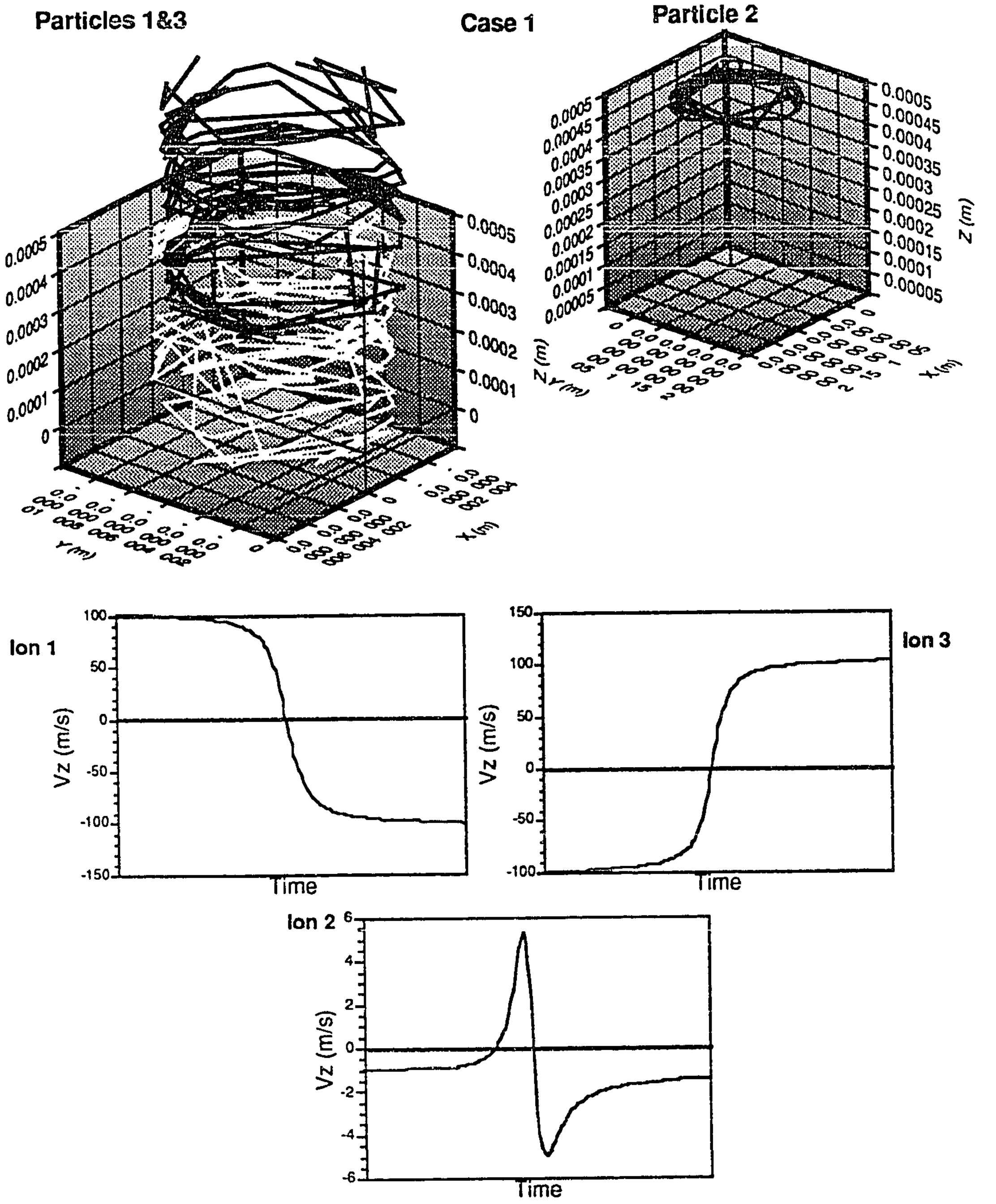

Figure 6. The three-dimensional paths and z-direction velocities for Case 1, where initial $|x| = 0$. The upper left plot show the paths of Ion 1 (white) and Ion 3 (black), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

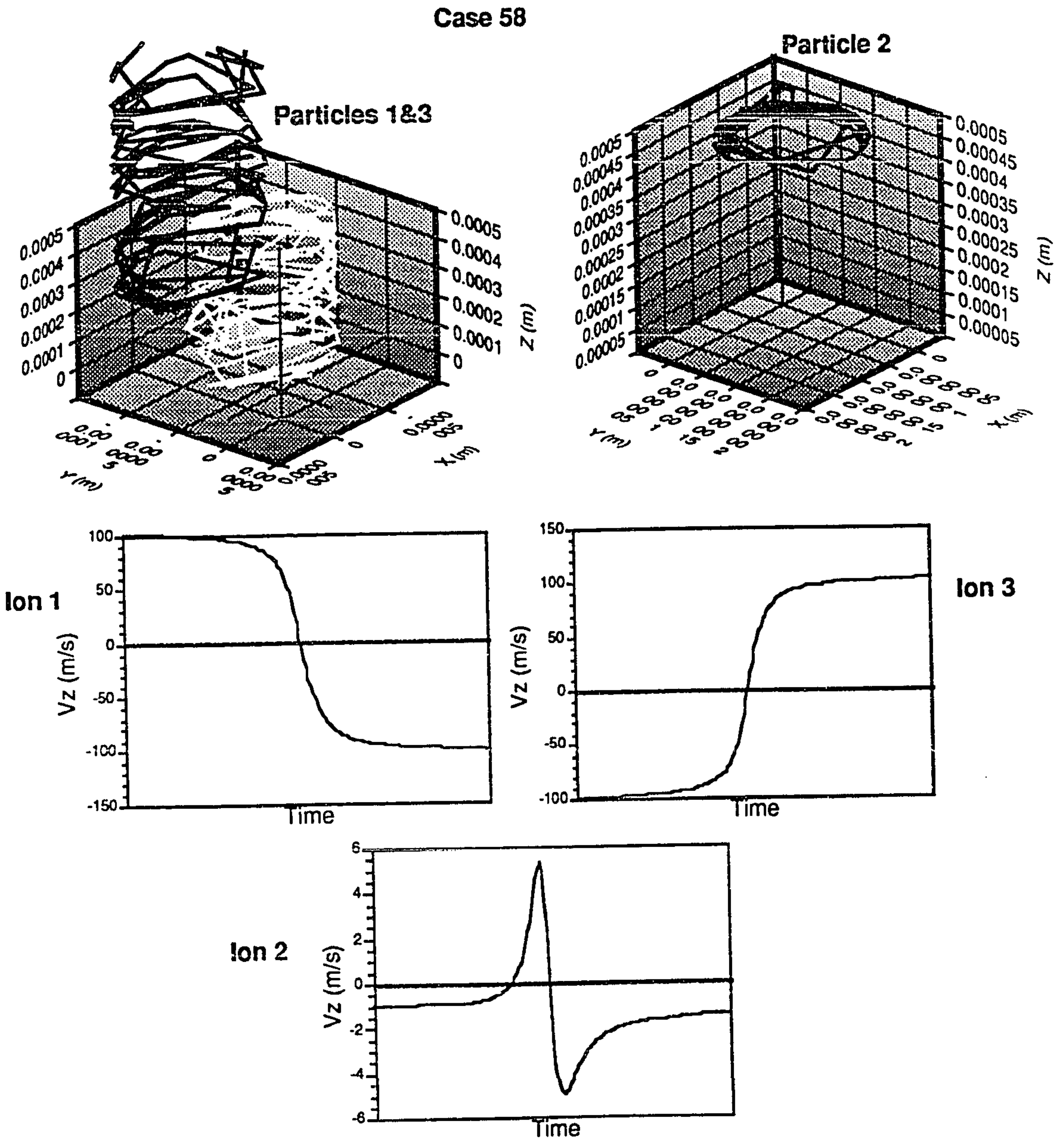

Figure 7. The three-dimensional paths and z-direction velocities for Case 58, where initial $|x| = 3.0 \times 10^{-7}$ m. The upper left plot show the paths of Ion 1 (white) and Ion 3 (black), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

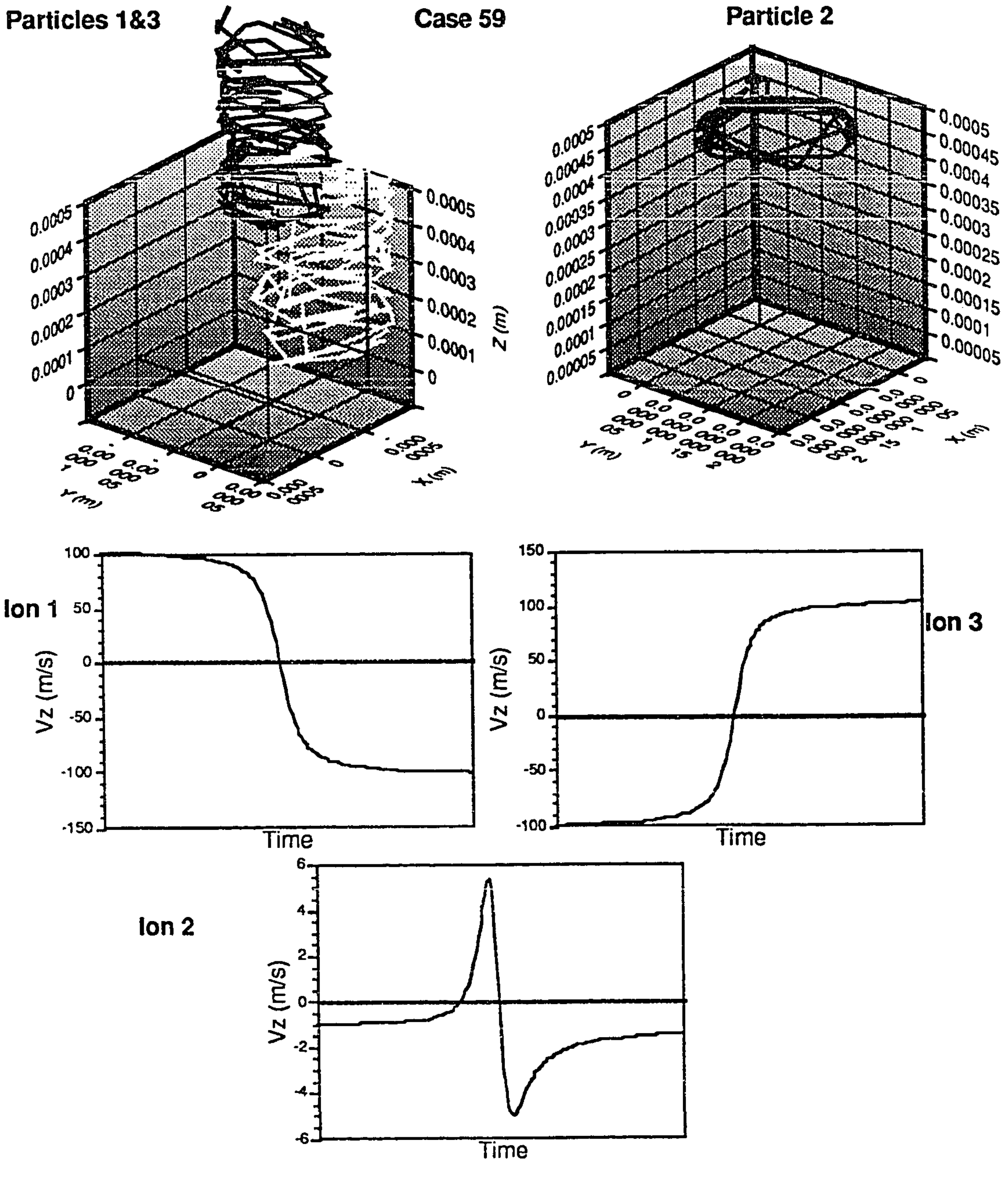

Figure 8. The three-dimensional paths and z-direction velocities for Case 59, where initial $|x| = 1.0 \times 10^{-6}$ m. The upper left plot show the paths of Ion 1 (white) and Ion 3 (black), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

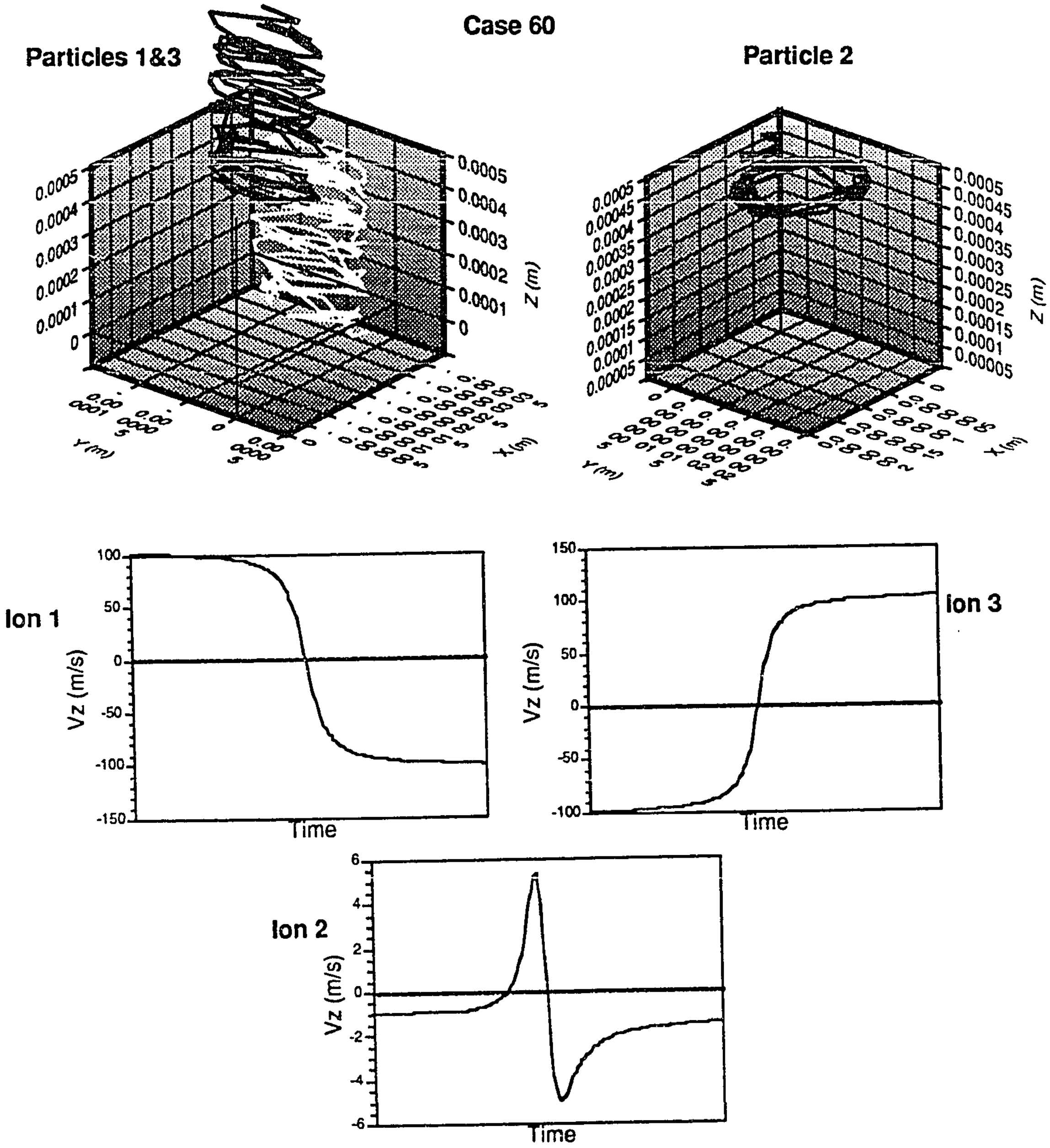

Figure 9. The three-dimensional paths and z-direction velocities for Case 60, where initial $|x| = 3.0 \times 10^{-6}$ m. The upper left plot show the paths of Ion 1 (white) and Ion 3 (black), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

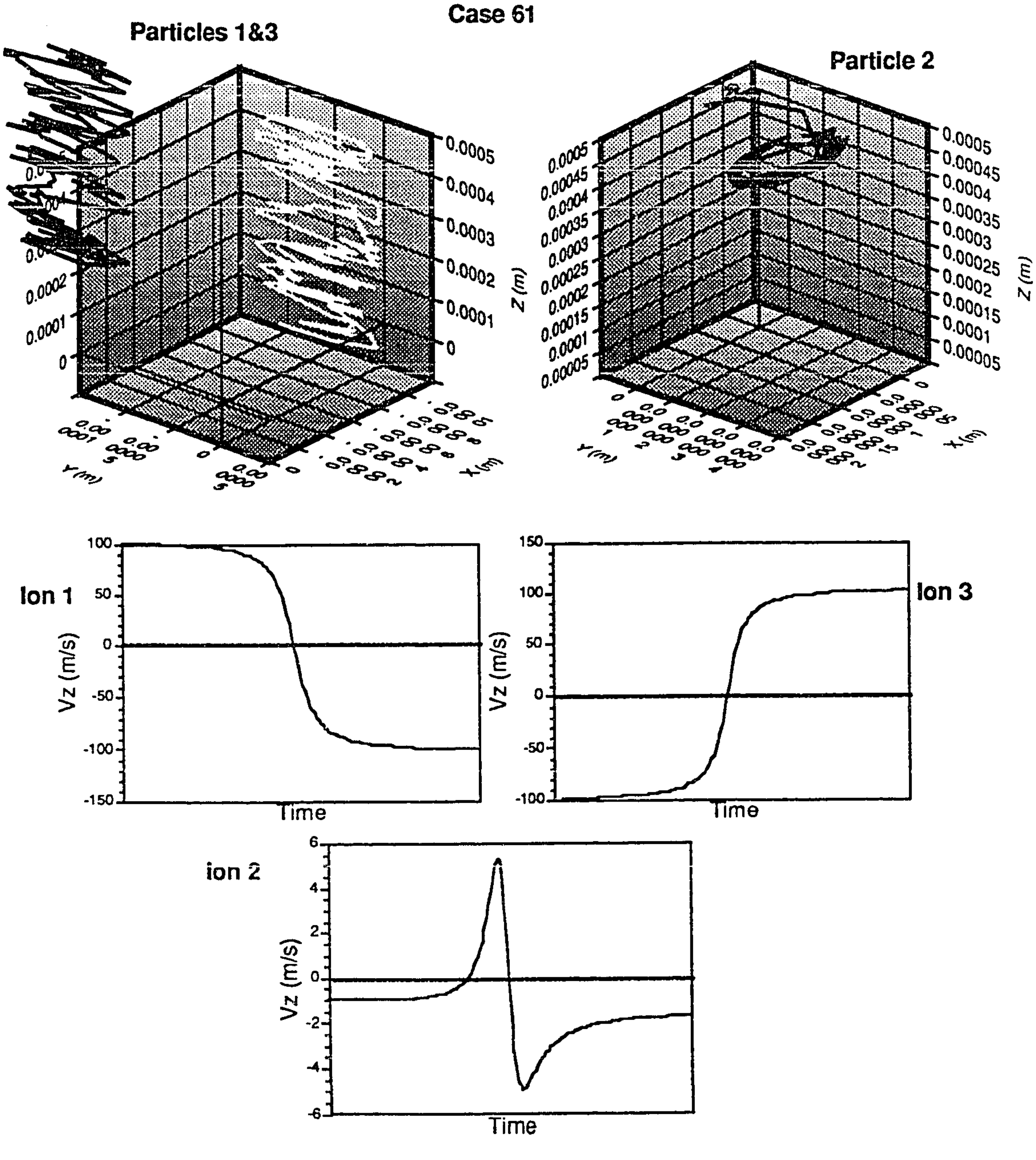

Figure 10. The three-dimensional paths and z-direction velocities for Case 61, where initial $|x| = 1.0 \times 10^{-5}$ m. The upper left plot show the paths of Ion 1 (white) and Ion 3 (black), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

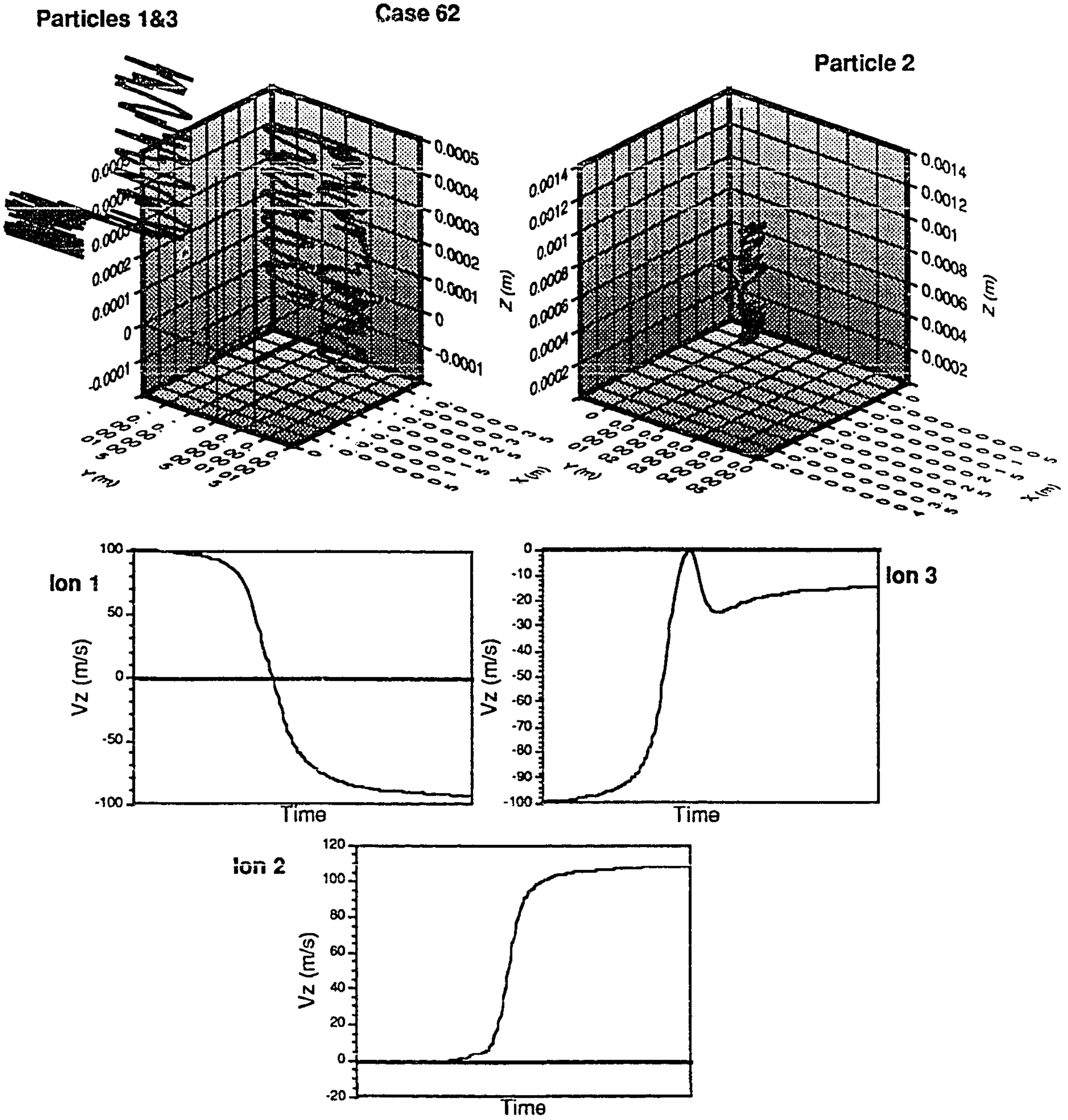

Figure 11. The three-dimensional paths and z-direction velocities for Case 62, where initial $|x| = 3.0 \times 10^{-5}$ m. The upper left plot show the paths of Ion 1 (black-right side) and Ion 3 (black- left side), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

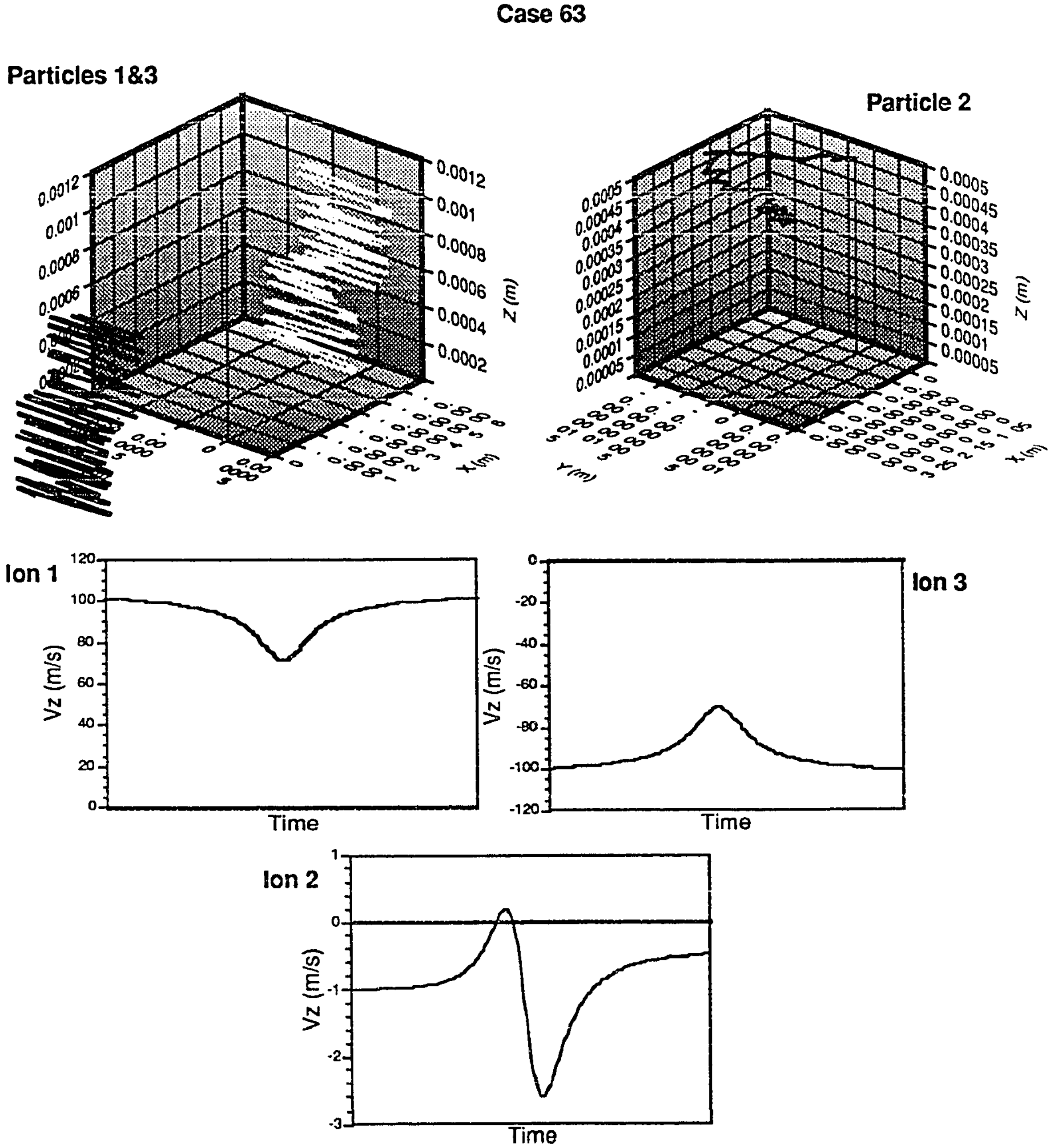

Figure 12. The three-dimensional paths and z-direction velocities for Case 63, where initial $|x| = 6.0 \times 10^{-5}$ m. The upper left plot show the paths of Ion 1 (white) and Ion 3 (black), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

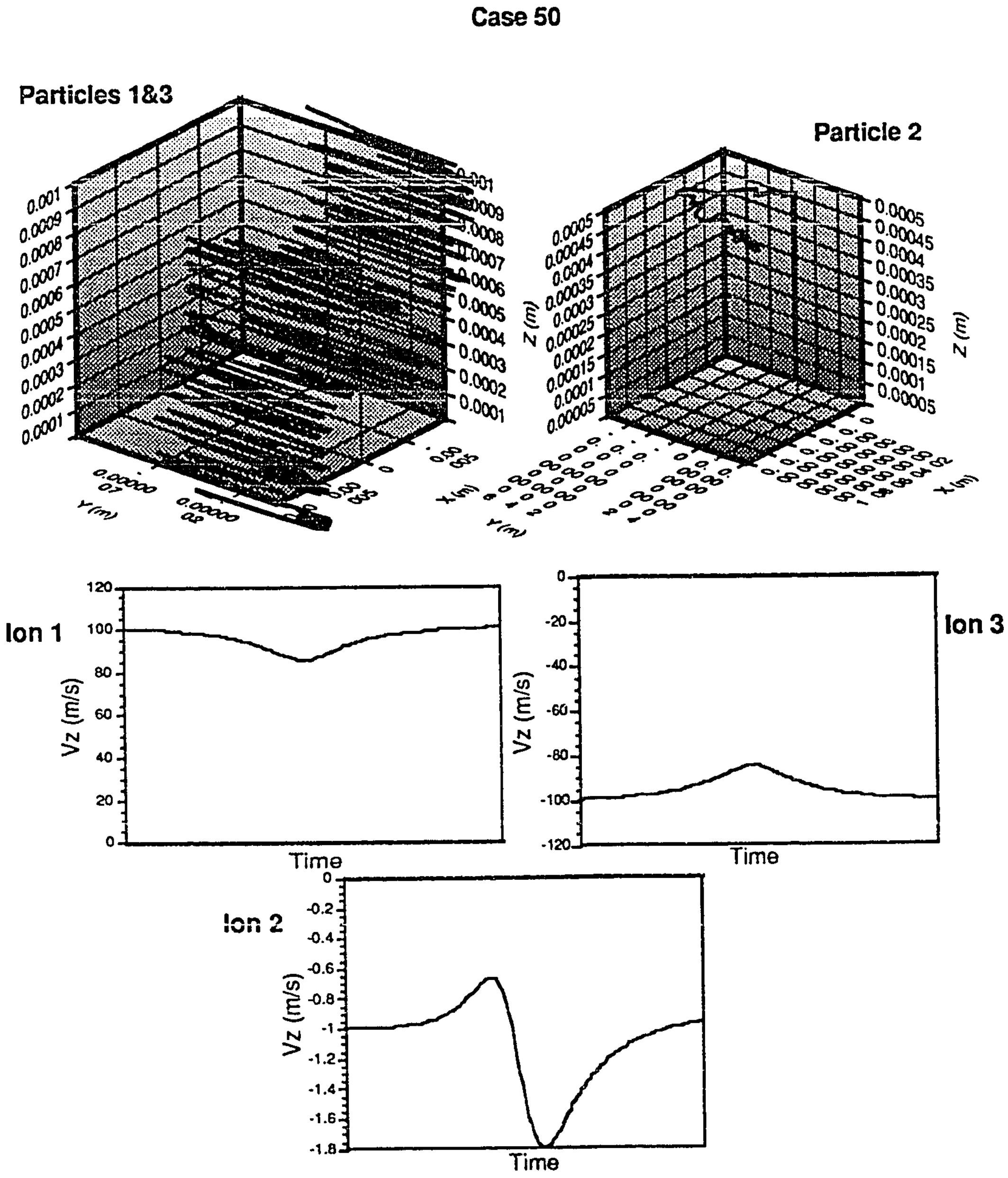

Figure 13. The three-dimensional paths and z-direction velocities for Case 50, where initial $|x| = 1.0 \times 10^{-4}$ m. The upper left plot show the paths of Ion 1 (dark-grey right) and Ion 3 (black left), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

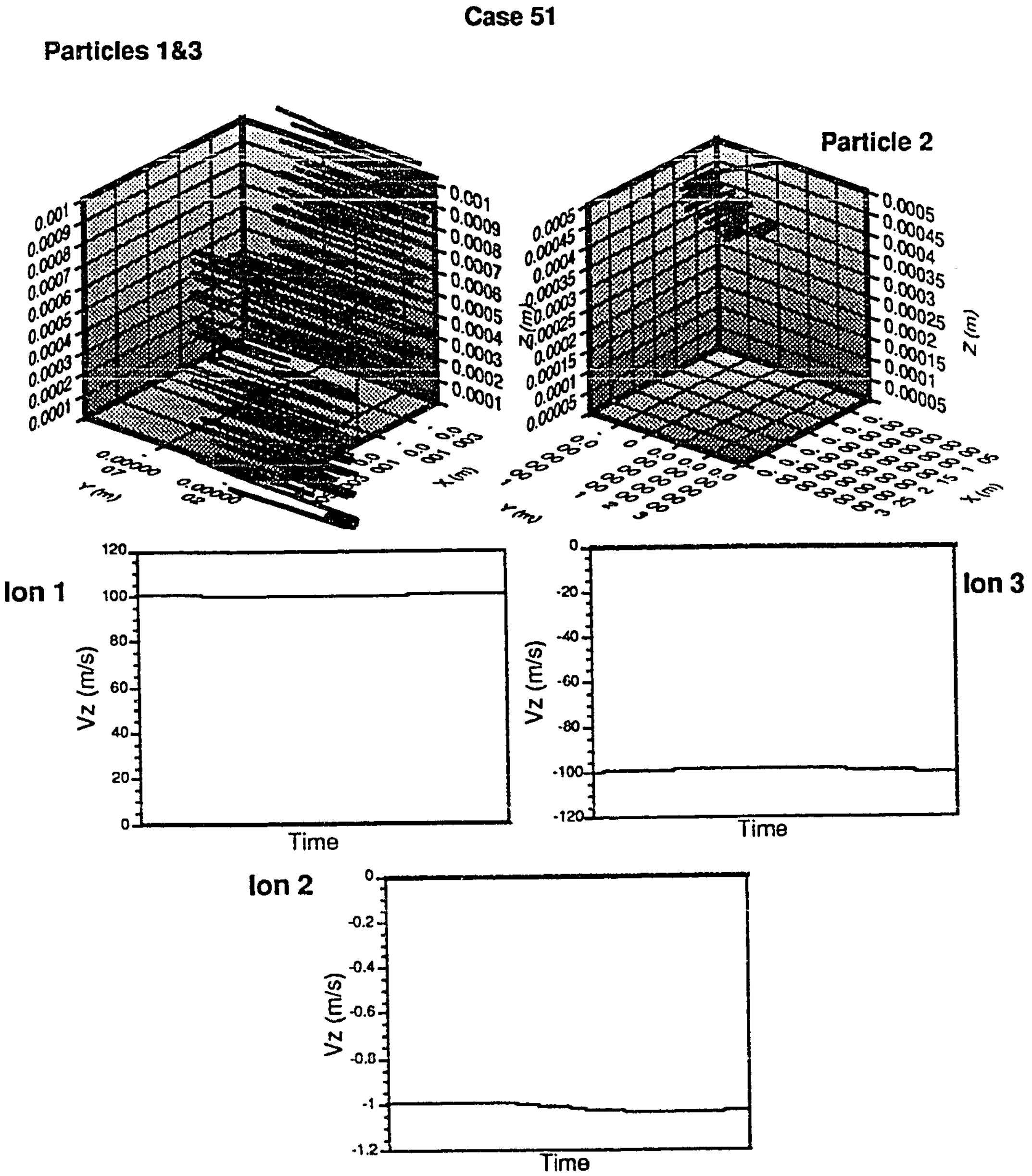

Figure 14. The three-dimensional paths and z-direction velocities for Case 51, where initial $|x| = 5.0 \times 10^{-4}$ m. The upper left plot show the paths of Ion 1 (dark-grey right) and Ion 3 (black left), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

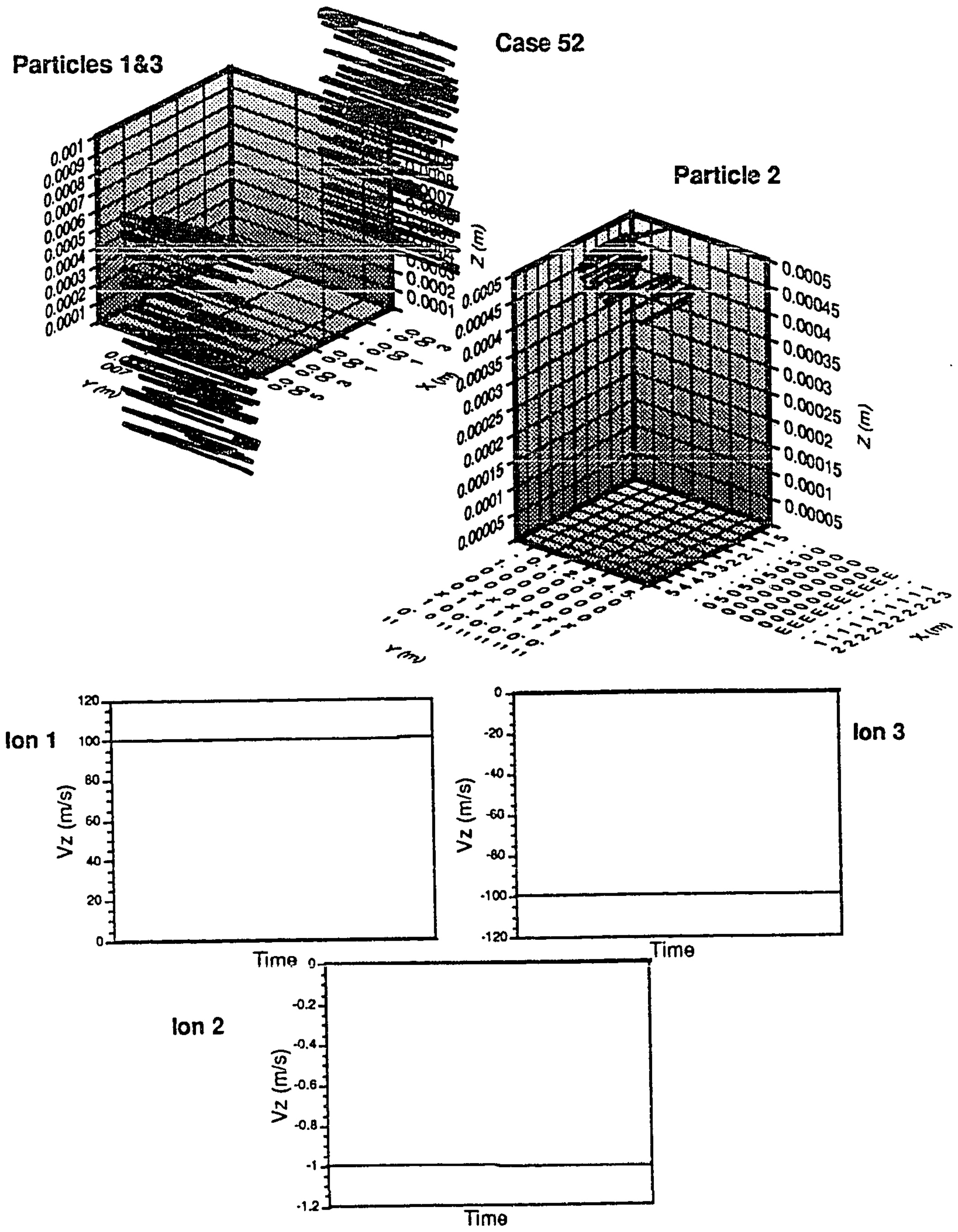

Figure 15. The three-dimensional paths and z-direction velocities for Case 52, where initial $|z| = 1.0 \times 10^{-3}$ m. The upper left plot show the paths of Ion 1 (dark-grey right) and Ion 3 (black left), while the upper right plot shows the path of Ion 2. The bottom three graphs illustrate the z-direction velocities for Ions 1, 3 and 2 respectively.

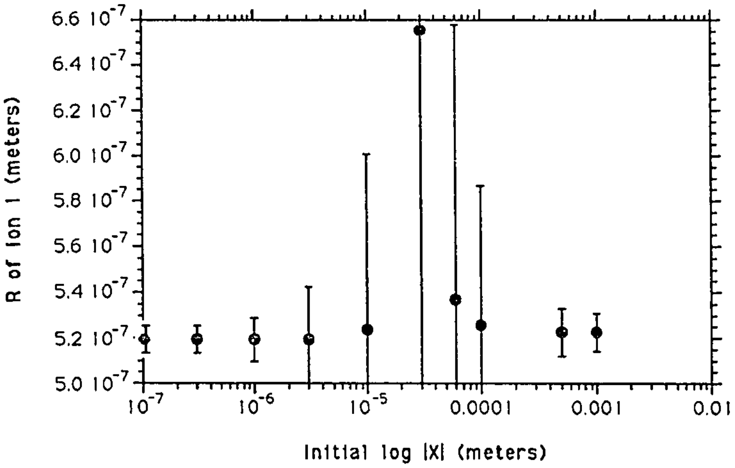

Figure 16. Rotational Radius R vs. Initial $|x|$ Separation of Ion 1 for Sample Cases. The last $R = 5.226 \pm 0.0846 \times 10^{-7}$m is very close to the theoretical $R_g = 5.228 \times 10^{-7}$m.

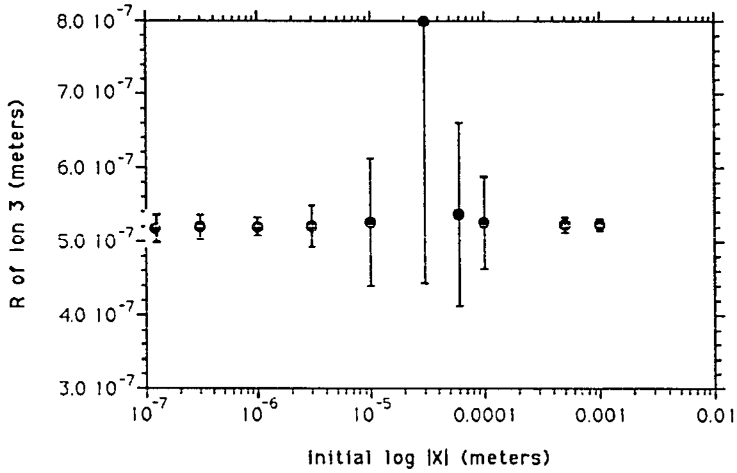

Figure 17. Rotational Radius R vs. Initial $|x|$ Separation of Ion 3 for Sample Cases. The last $R = 5.226 \pm 0.0848 \times 10^{-7}$m is very close to the theoretical $R_g = 5.228 \times 10^{-7}$m.

We explored the variation of $r(t)$ by examining three cases: 1) where the outer particles were on the same field line (Case 1); 2) where the outer particles were on nearby field lines (Case 61); and 3) where the outer particles were on distant field lines (Case 52). These cases are represented in Figures (18, 19, 20). For Figure 18, Ions 1 and 3 were spiraling along the same field line. They interacted by smoothly repelling. Ion 1's gyroscopic radius was, however, permanently altered by its encounter with the middle and/or top particle. Its orbital radius for the first 200 iterations was 5.2261 $\pm$ 0.0578 $\times$ 10^{-7} m. For the last 200 iterations its orbital radius was 5.1622 $\pm$ 0.0348 $\times$ 10^{-7} m.

Figure 19 represents another scenario. They were on nearby field lines, separated by 10^{-5} m. They also repelled each other, but at closest approach, they started greatly perturbing Ion 2's and each other's orbit before moving apart. For the first 100 iterations, Ion 1's orbital radius was 5.2506 $\pm$ 0.862 $\times$ 10^{-7} m. During the middle 30 points, its orbital radius was 5.1959 $\pm$ 0.264 $\times$ 10^{-7} m. And during the last 100 iterations, Ion 1's orbital radius was 5.2158 $\pm$ 0.786 $\times$ 10^{-7} m. So the net effect was during Ion 1's encounter with particles 2 and 3 was to only slightly reduce its orbital radius.

Figure 20 represents the other extreme from Figure 18. In this figure, Ions 1 and 3 were initially on field lines far from each other: 10^{-3} m. None of the three particles was repelled from each other and they continued on their original path. Ion 1 only interacted enough with Ion 2 to have its own orbital radius slightly decreased. Its orbital radius for the first 200 iterations was 5.1339 $\pm$ 0.687 $\times$ 10^{-7} m. And during the last 200 iterations, Ion 1's orbital radius was 5.1337 $\pm$ 0.688 $\times$ 10^{-7} m.

The gyroscopic radius of the middle particle was the least interesting because its theoretical value is 0. In a few iterations, the outer particles started imparting a torque on it and the middle ion began a circular trajectory. We illustrate this with Case 1 in Figure 21 when the initial positions for all three ions were along the same B-field line. In the simulation the orbital radius was 0 until ~ 8×10^{-9} sec. Its radius of orbital motion then gradually increased until it stabilized at ~ 2×10^{-8} m.

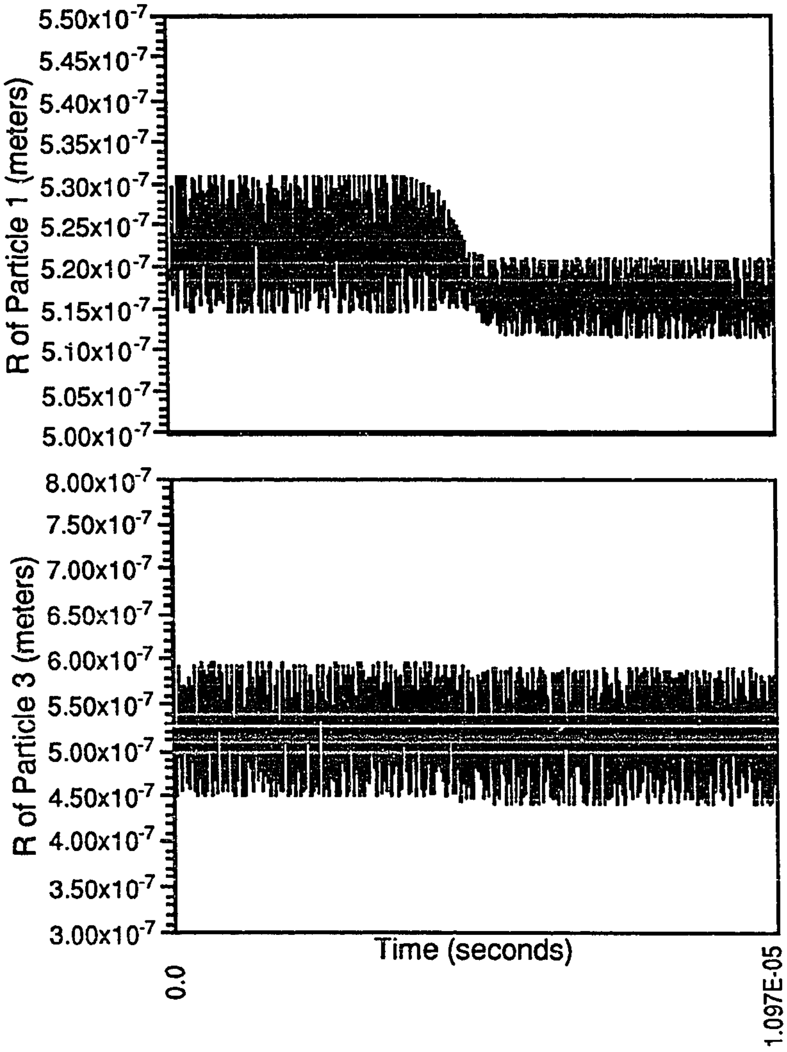

Figure 18. Rotational Radius R vs.Time for Ions 1 and 3 on the Same Field Line (Case 1).

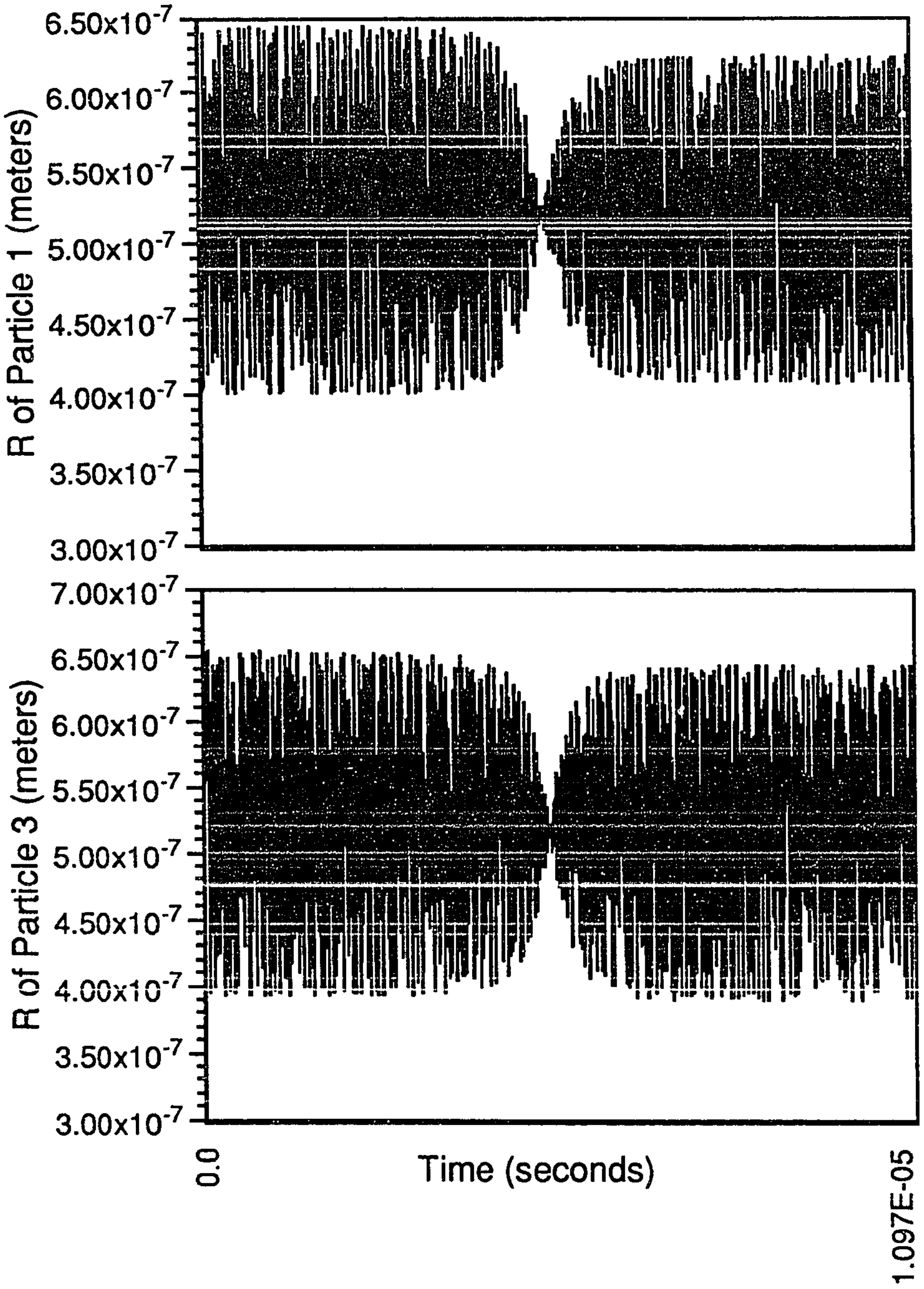

Figure 19. Rotational Radius R vs.Time for Ions 1 and 3 on Nearby Field Lines (Case 61). The initial $|x|$ separation between those ions is 10^{-5} m.

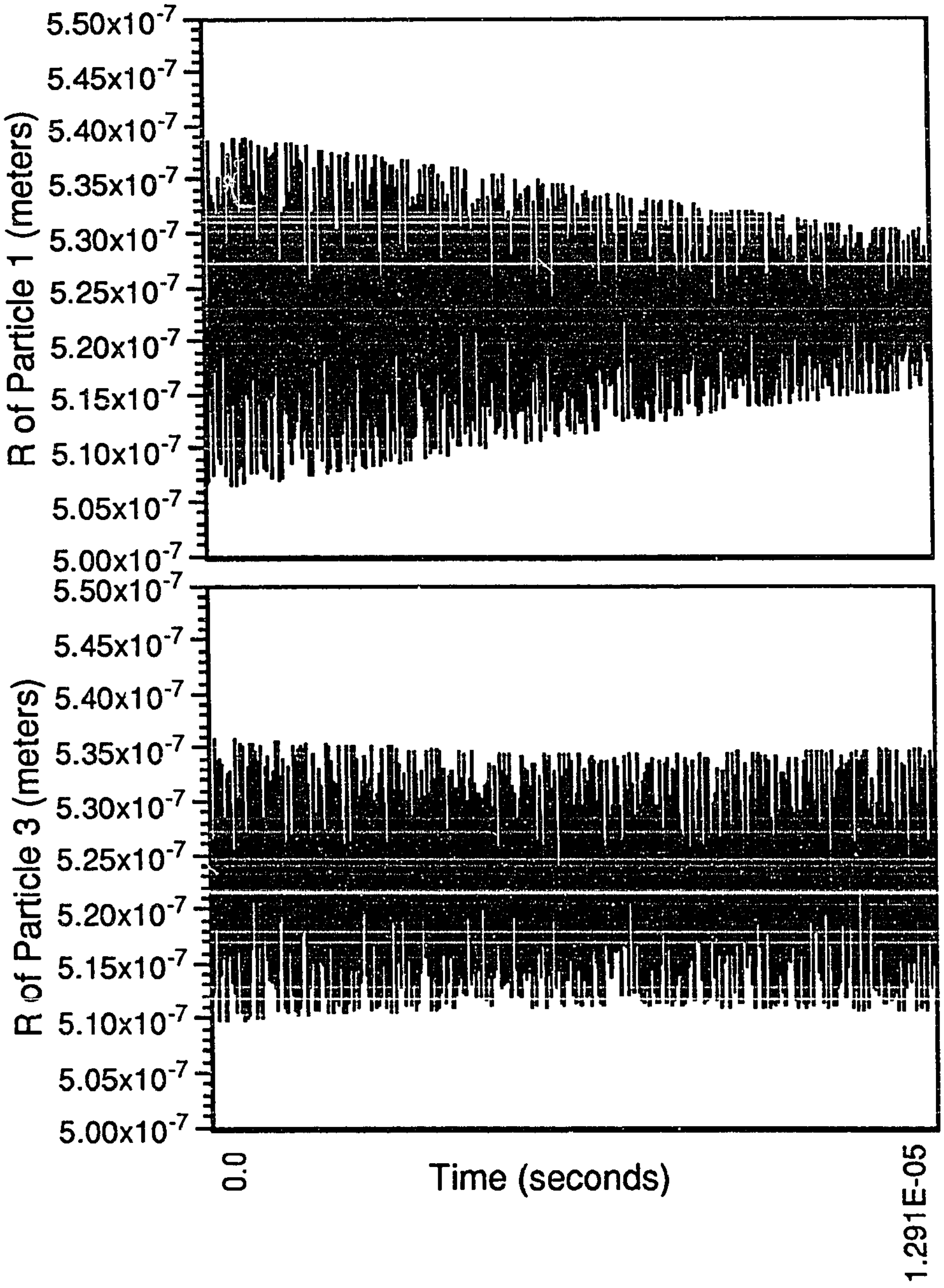

Figure 20. Rotational Radius R vs.Time for Ions 1 and 3 on Distant Field Lines (Case 52). The initial $|x|$ separation between those ions is 10^{-3} m.

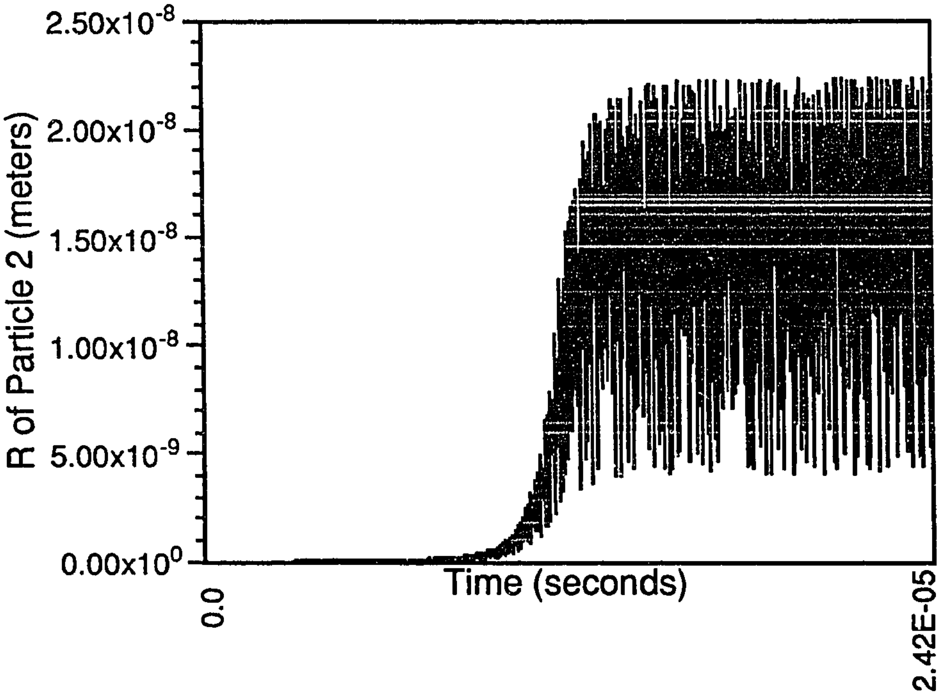

Figure 21. Rotational Radius R vs.Time for Ion 2 on the Same Field Line (Case 1).

Gyroscopic Period

We again utilized the computed xy positions of the particles, but this time to check the theoretical P_g value of Equation (5). Since the equation assumes no dependence on velocity and contains quantities that are all constants in this simulation, the gyroscopic period for all three particles is 3.285×10^{-8} seconds. The periods were empirically found from the data by noting when the xy positions of the particles have gone through a complete cycle. This proved to be somewhat difficult because the B-S method employs a variable time step. So the oscillatory path of each particle is sampled at uneven times, and it is up to the experimenter to decide which xy values show a complete cycle. Figures 22 and 23 illustrate the difficulty. Figure 22 shows the gyroscopic cycle when the motion is computed with a fixed time-step scheme: the Verlet algorithm. The motion is smooth and it is very straightforward to find the completion of one cycle. Figure 23 shows the gyroscopic motion with a variable time-step scheme: the B-S algorithm. The graph demonstrates that each cycle is sampled, at most, three times.

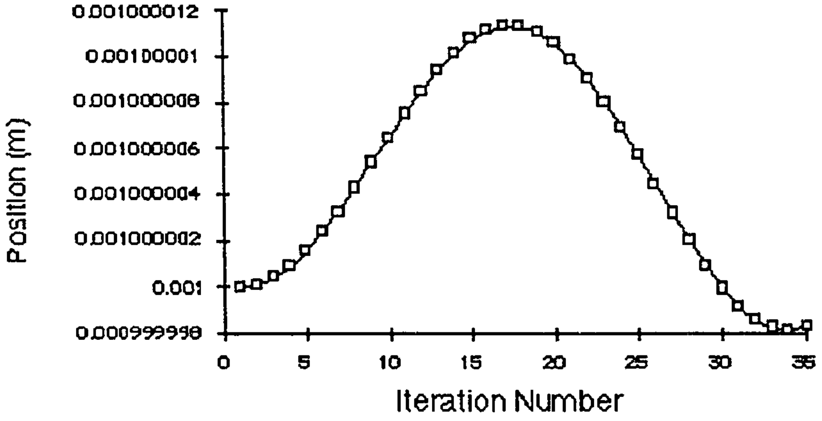

Figure 22. Gyroscopic Motion from a Fixed Time-Step Scheme: the Verlet Algorithm. The cycle here is sampled many times for the experimenter to empirically compute the gyroscopic period.

With this caveat, the periods were empirically checked for the same ion and cases illustrated in Figures (18, 19, 20). For Case 1 (Figure 18), the period was computed at the beginning and at the end of the run. At the beginning, the period was 3.2900 ± 0.0267 × 10^{-8} s, and at the end of the run it was 3.37200 ± 0.123 × 10^{-8} s. So the period has increased. Physically this makes sense because conservation of angular momentum would require that when the period increases, the radius decreases.

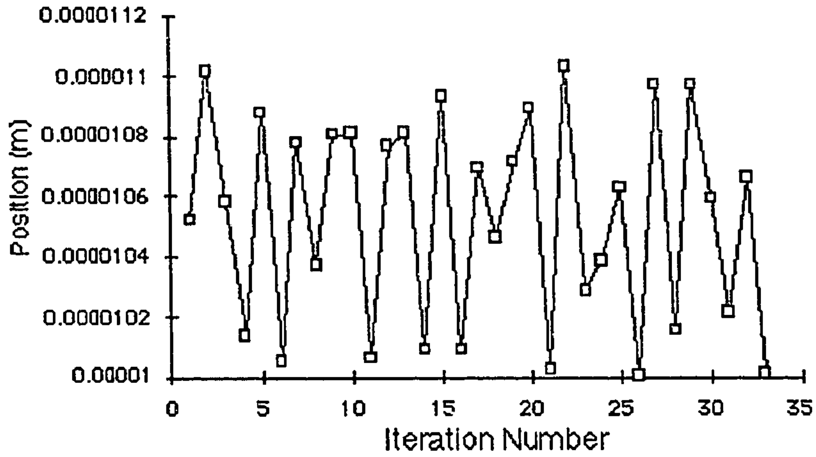

Figure 23. Gyroscopic Motion from a Variable Time-Step Scheme: the B-S Algorithm. The cycle here is sampled, at most, three times for the experimenter to empirically compute the gyroscopic period.

For Case 61 (Figure 19), the periods were checked at the beginning, middle and end of the run. At the beginning, the gyroscopic period was 3.155 ± 0.254 × 10^{-8} s. In the middle of the run, the gyroscopic period was 3.141 ± 0.495 × 10^{-8} s. And at the end of the run, the gyroscopic period was 3.274 ± 0.410 × 10^{-8} s. So for this case, the data slightly supports an increase in the period.

And finally, for Case 52 (Figure 20), the periods were checked at the beginning and at the end. At the beginning of the run, the gyroscopic period was 3.218 ± 0.0974 × 10^{-8} s. At the end, the gyroscopic period was 3.309 ± 0.394 × 10^{-8} s. Here, also, the period seems to have increased, although the gyroscopic radius has only slightly decreased.

Rotational Kinetic Energy

In this and the following section, we explore how the orbital kinetic energy (T_θ) is coupled to the z-motion. For each of the ten sample cases, a T_θ (total) was computed for the three-ion system. Denote $\Delta T_\theta = T_\theta(\text{start}) - T_\theta(\text{end})$. Using this notation, if the final T_θ is less than the beginning T_θ, then $\Delta T_\theta > 0$ (and vice versa). Since the ten sample cases demonstrate initial conditions in which the outer ions are successively further and further apart, we are interested in how the total rotational kinetic energy changes as the separation between the ions increases. Figure 24 illustrates ΔT_θ versus distance apart of the two outer ions. These results indicate that $\Delta T_\theta \rightarrow 0$ as the ions get further apart.

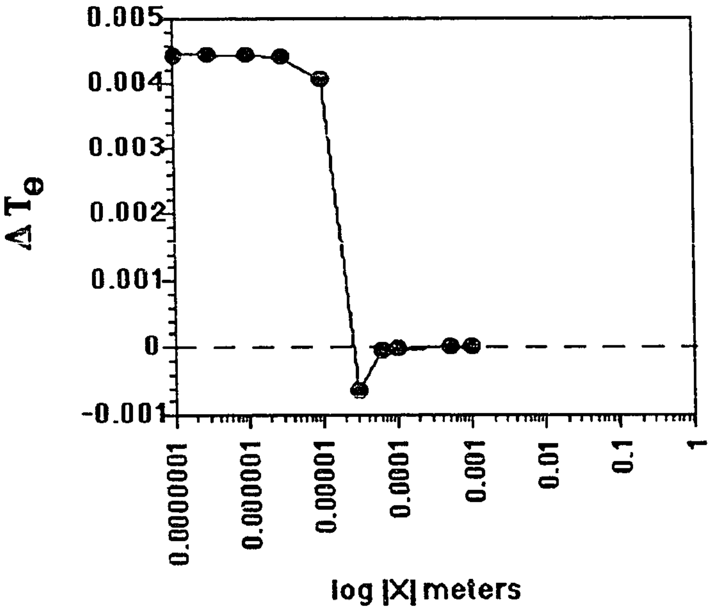

Figure 24. Change in Rotational Kinetic Energy versus Separation of the Outer Ions.

Vertical Kinetic Energy

In the last section, it was demonstrated that the change in orbital kinetic energy decreases as the ions' initial paths are separated. In this section we will describe results on the relationship between the z-motion kinetic energy (T_z) of the inner ion and the two outer ions. Figures 6 through 15 illustrate the z-directed velocity of the middle ion. We can ask the question: does the vertical kinetic energy (which equals $(1/2)m_0v_z^2$) of the middle ion decrease after the ions repelled from each other in these cases? The answer upon examination of the figures is "Yes." This would imply that the middle ion loses vertical kinetic energy to the outer ions after their closest encounter. This result has implications in the TOF experiment on the number of slow-moving ions that remain after the initial pulse of ions.

CHAPTER 7

Conclusions

The purpose of this computer simulation was to explore the dynamics of a small system (three) of charged particles confined in a constant magnetic field under the classical assumption of the Lorentz force. Numerical simulations were necessary to show the motions and energies because the system could not be solved analytically. The TOF experiment at Los Alamos provided motivation to specifically investigate the exchange of rotational and vertical kinetic energies between the ions.

Our goals were accomplished with my Colliding Ions program. It was demonstrated that the ions moved in a helical path, and interacted in ways that were physically plausible. I demonstrated that the gyroscopic radii and periods of the ions' motion could be calculated from the data generated. And finally I demonstrated that the change in rotational kinetic energy decreased as the separation of the ions' trajectories increased, and that the outer ions in this system carried away some of the z-directed kinetic energy from the inner ion.

APPENDIX

A. Verlet Algorithm Derivation (Velocity form)

Because this derivation may be useful to the reader, I will detail the steps for the Verlet algorithm here.

The Euler-Cromer ("last-point approximation") is

$$\left\{ \begin{array}{c} v_{n+1} = v_n + a_n \Delta t \\ x_{n+1} = x_n + v_{n+1} \Delta t \end{array} \right\} \quad .$$

Note that these came from expanding the v_{n+1} ($\equiv v(t_n + \Delta t)$) and x_{n+1} ($\equiv x(t_n + \Delta t)$) in a Taylor series,

$$v_{n+1} = v_n + a_n \Delta t + O\left[(\Delta t)^2\right]$$

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 + O\left[(\Delta t)^3\right] \quad , \tag{33}$$

$$x_{n-1} \left(\equiv x\left(t_n - \Delta t\right)\right) = x_n - v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 - O\left[(\Delta t)^3\right] \quad . \tag{34}$$

Adding the forward (Equation 33) and the reverse (Equation 34) terms give us

$$x_{n+1} = 2x_n - x_{n-1} + a_n (\Delta t)^2 \quad . \tag{35}$$

Subtracting the forward (Equation 33) and the reverse (Equation 34) terms give us

$$v_n = \frac{x_{n+1} - x_{n-1}}{2\Delta t} \quad , \tag{36}$$

or

$$v_{n+1} = \frac{x_{n+2} - x_n}{2\Delta t} \quad . \tag{37}$$

Equations (35) and (37) are the "Verlet Algorithm." However this is not self-starting and it has a round-off problem. Whenever the computer subtracts two quantities of the same order of magnitude, the result loses some numerical precision and it may lead to serious round-off error (11). So we want to derive a form of the Verlet algorithm that is self-starting and minimizes round-off error.

From Equation (35), we add and subtract $\frac{1}{2}x_{n+1}$ from both sides. We get

$$x_{n+1} = x_n + \frac{1}{2}(x_{n+1} - x_{n-1}) - \frac{1}{2}(x_{n-1} + x_{n+1}) + x_n + a_n(\Delta t)^2 \quad . \tag{38}$$

Then, from Equation (37):

$$2\Delta t v_n = (x_{n+1} - x_{n-1}) \quad .$$

Substitute into Equation (38)

$$x_{n+1} = x_n + \frac{1}{2}(2\Delta t v_n) - \frac{1}{2}(x_{n-1} + x_{n+1} - 2x_n) + a_n(\Delta t)^2 \quad . \tag{39}$$

From Equation (35)

$$a_n(\Delta t)^2 = (x_{n+1} + x_{n-1} - 2x_n) \quad ,$$

Substitute the above into Equation (39)

$$x_{n+1} = x_n + v_n(\Delta t) + \frac{1}{2}a_n(\Delta t)^2 \quad . \tag{40}$$

Equation (40) the position iteration for the velocity form of the Verlet algorithm.

To derive the velocity iteration we go back to Equations (35) and (36). Substitute Equation (35) into Equation (36). We get

$$v_{n+1} = \frac{x_{n+1} - x_n + \frac{1}{2}a_n\left(\Delta t\right)^2}{\Delta t} \quad .$$

But we know x_{n+1} from Equation (40). Substituting into the above,

$$v_{n+1} = \frac{v_n \Delta t + \frac{1}{2}\left(a_n\left(\Delta t\right)^2 + a_{n+1}\left(\Delta t\right)^2\right)}{\Delta t} \quad ,$$

or

$$v_{n+1} = v_n + \frac{1}{2}\Delta t\left(a_n + a_{n+1}\right) \quad . \tag{41}$$

Equation (41) the velocity iteration for the velocity form of the Verlet algorithm.

B. Source Program Colliding Ions

```
unit UParticle;

{PURPOSE: This program (Pascal unit) calculates the positions and velocities for n charged }
{particles trapped in a magnetic field. The positions and velocities are found from the Lorentz }
{force: F = q(E + v x B) where E is the Electric field and B is the magnetic field at the location }
{of the charge. The electric field is calculated from Coulomb's Law: E(r) = (1/(4 pi eps_0) ) * }
{(q r )/ (R)^(3) .  Both the Lorentz Force and Coulomb's Law satisfy the superposition principle }
{so all of the E fields and the Forces are added for each charged particle. I calculate cartesian }
{position coordinates: x, y, and z for each particle, and since the force equation is 2nd order, }
{we have 6 first order equations to solve for each charged particle. The first order equations }
{are integrated using the Bulirsch-Stoer integration scheme which uses a variable time-step.  }
{The scheme works on the Richardson extrapolation principle where an initial dt is used to }
{solve the equations.  Then dt is split and we integrate the function with a smaller time step.  }
{We continue this process for up to 10 different time steps.  When the answer to the }
{integration for the current time step changes from the answer for the previous time step}
{by less than a number: "eps", we consider that integration done and an extrapolation using }
{rational fractions is used to extend what the integration values ought to be if dt went to zero.}

{INPUT:}
  {file: 'CollidingIonsIn.text' which contains graphics variables, Bulirsch-Stoer variables, and }
  {positions, velocities, charges, and masses of the ions.}

{OUTPUT:}
  {The following files will be written to:}
    {filename1 := 'Ions_positions.txt(BS)'}
    {filename2 := 'Ions_velocities.txt(BS)'}
    {filename3 := 'Ions_energies.txt(BS)'}

{------------------------------------------------------------------------}
{Amara Graps    3-17-91        Updated: 6-4-91}

{For Masters thesis in Physics, San Jose State University,  Spring 1991}
{------------------------------------------------------------------------}


interface

 uses
  Memtypes, QuickDraw, OSIntf, ToolIntf, FixMath, Graf3D, UReal3D, Sane;

 procedure  ParticleInit;

 procedure  ParticleAxis;

 procedure  ParticleRun;

 procedure  ParticleStop;


{------------------------------------------------------}

implementation

 const
  kMaxParticles = 3;      {Max number of particles}
  KMax_n6 = 18;    {=KMaxParticles*6 = number of particles * number of 1st order }
                     {equations = total number of 1st order equations}
  pi = 3.1415927e0;
  FPE = 1.112626e-10;     { 4 pi eps_0}

 type
  vece = array[1..kMaxParticles] of Extended;
  veci = array[1..kMaxParticles] of Integer;
  vecp = array[1..kMaxParticles] of pattern;

 {for Bulirsch-Stoer integration scheme}
  vecn611 = array[1..kMax_n6, 1..11] of Extended;
  vecn612 = array[1..kMax_n6, 1..12] of Extended;
  quotetype = array[1..11, 1..2] of Extended;
  vecn6 = array[1..kMax_n6] of Extended;

 var
  {for graphics}
  gMyPort3D: Port3D;
  vpLeft, vpTop, vpRight, vpBottom: Extended;
  myPitch, myYaw, myRoll: Extended;
  scaleFactor, focusLen, axisLen, labelSpace, size: Extended;
  x_offset, y_offset, z_offset: Extended;

  background: pattern;
  ParticleDelay: veci;
  axesP: Integer;
  ParticlePat: vecp;

  {keeping track of steps}
  gStepsTotal, gStepsSoFar, start_tick: LongInt;

  {for physics calculations}
  NumParticles, n6: Integer;
  x, y, z, vx, vy, vz, v: vece;
  Q, M, T_per, Elec_x, Elec_y, Elec_z, vtheta, Electheta, KEnergy_z, KEnergy_theta, KEnergy: vece;
  PEnergy, KE, KE_theta, TotEnergy, old_TotE, Del_TotE, B, Start_E: Extended;
  Ex, Ey, Ez, rx, ry, rz, R, R3, R_start, R_min, R_end, dist_frac: Extended;

  {for BS integrator}
  bsy, bsdy, error, ymax: vecn6;
  t, dt, eps, hmin: extended;
  mf, jstart, maxord, maxpts, kflag: Integer;

  {for reading input and writing output}
  write_count: Integer;
  write_on, write_always: Boolean;
  fileout1, fileout2, fileout3, IonsIn: text;
  filename1, filename2, filename3: string[40];
  tick_frac: Longint;

  {misc}
  gDummy: EventRecord;

{*******************************************************}

 function Int2Pattern (i: integer): Pattern;
{PURPOSE: This function assigns pen patterns based on the input integer values.  The user}
{can use different pen patterns to distinguish the different ions as their positions are being}
{plotted.}
{-------------------------------------------------------}
```

```
{F. Bennett     3-90}
{------------------------------------------------------------}

 begin
  case i of
   1:
    Int2Pattern := white;
   2:
    Int2Pattern := ltgray;
   3:
    Int2Pattern := gray;
   4:
    Int2Pattern := dkgray;
   5:
    Int2Pattern := black;
   otherwise
    Int2Pattern := white;
  end;

 end;                {function Int2Pattern}
{************************************************************}

 procedure InitFiles;
{PURPOSE: This program sets up the 3 output files that we will write the results to.  All filenames}
{are global variables.}
{------------------------------------------------------------}
{Amara Graps            3-30-91}
{------------------------------------------------------------}

 begin
  filename1 := 'Ions_positions.txt(BS)';
  filename2 := 'Ions_velocities.txt(BS)';
  filename3 := 'Ions_energies.txt(BS)';
  Open(fileout1, filename1);    {fileout1 now equals output data file}
  Rewrite(fileout1);          {place cursor at beginning of file}
  Writeln(fileout1, ' ');
  Open(fileout2, filename2);
  Rewrite(fileout2);
  Writeln(fileout2, ' ');
  Open(fileout3, filename3);
  Rewrite(fileout3);
  Writeln(fileout3, ' ');

 end;          {procedure InitFiles}

{************************************************************}

 procedure GetVars;
{PURPOSE: This procedure reads from a file in the current folder called "CollidingIonsIn.text"}
{and assigns values for the graphics, the Bulirsh-Stoer integrator, and the positions, velocities,}
{charges, and masses for the ions.  All of the variables being assigned in this procedure are}
{global variables.}
{------------------------------------------------------------}
{Amara Graps,    F. Bennett      3-90          Updated: 3-29-91}
{------------------------------------------------------------}

  var
   i: Integer;
  LocParticlePat, locbackround: Integer;

begin
 open(IonsIn, 'CollidingIonsIn.text');

  {graphics parameters}
 readln(IonsIn, vpLeft, vpTop, vpRight, vpBottom);
 readln(IonsIn, myPitch, myYaw, myRoll);
 readln(IonsIn, scaleFactor);
 readln(IonsIn, focusLen);
 readln(IonsIn, locbackround);
 background := Int2Pattern(locbackround);
 readln(IonsIn, axesP, axisLen, labelSpace);
 readln(IonsIn, size);
 readln(IonsIn, x_offset, y_offset, z_offset);
 readln(IonsIn, gStepsTotal);
 readln(IonsIn, NumParticles);
 for i := 1 to NumParticles do
  begin
   readln(IonsIn, ParticleDelay[i], LocParticlePat);
   ParticlePat[i] := Int2Pattern(LocParticlePat);
  end;

  {Bulirsch-Stoer initial parameters}
 readln(IonsIn, hmin);
 readln(IonsIn, t);
 readln(IonsIn, dt);
 readln(IonsIn, eps);
 readln(IonsIn, mf);
 readln(IonsIn, jstart);
 readln(IonsIn, maxord);
 readln(IonsIn, maxpts);

  {physical variables and misc}
 readln(IonsIn, B);{magnetic field}
 readln(IonsIn, write_always);
 readln(IonsIn, tick_frac);
 readln(IonsIn, dist_frac);
 for i := 1 to NumParticles do
  begin
  {initial positions and velcoities of the charged particles}
   readln(IonsIn, x[i], y[i], z[i], vx[i], vy[i], vz[i], Q[i], M[i]);
  end;

  n6 := NumParticles * 6;       {gives number of 1st order differential eqn's}

  close(IonsIn);

 end;                  {procedure GetVars}
{*******************************************************}

 procedure BS_SetUp;
{PURPOSE: This procedure stuffs the initial positions and velocities from the GetVars procedure}
{into the Bulirsch-Stoer "bsy" array so that the Difsub procedure can calculate a new timestep }
{and the next positions and velocities.  All variables here are global variables.}
{-------------------------------------------------------}
{Amara Graps 3-15-91          }
{-------------------------------------------------------}
```

```
  var
   i: Integer;

 begin

  for i := 0 to (NumParticles - 1) do
   begin
    bsy[6 * i + 1] := x[i + 1];
    bsy[6 * i + 2] := vx[i + 1];
    bsy[6 * i + 3] := y[i + 1];
    bsy[6 * i + 4] := vy[i + 1];
    bsy[6 * i + 5] := z[i + 1];
    bsy[6 * i + 6] := vz[i + 1];
   end;      {for i := 0 to (number of particles)-1}

  for i := 1 to n6 do
   begin
  {Max value of dependent variable = 1 for initial only.  ymax is needed for the BS integrator}
    ymax[i] := 1.0;
   end;      {for i = 1 to the number of differential equations.}

 end;                        {procedure BS_SetUp}

{****************************************************}

 function Minimum (a, b: extended): extended;
{PURPOSE: This function finds the minimum of a and b.}
{-----------------------------------------------------}
{Amara Graps     4-3-91        }
{-----------------------------------------------------}

 begin

  if a <= b then
   Minimum := a
  else
   Minimum := b;

 end;                  {function Minimum}
{****************************************************}

 procedure  Calc_Energies;
{PURPOSE: This subroutine calculates the important physical quantities from the current}
{positions and velocities of the ions. The quantities calculated for each particle are: velocity, }
{magnitude,  rotational velocity (xy direction only),  total kinetic energy, total kinetic energy in }
{the xy direction, kinetic energy in the z direction, rotation period, and electric field in the xyz }
{directions.  The potential energy, total energy and distance between the particles are also }
{calculated. All  variables are global variables.}
{-----------------------------------------------------}
{Amara Graps  11-90         Updated: 4-3-91         }
{-----------------------------------------------------}

  var
   i, j, p: integer;
   R_prev: Extended;

  begin
```

```
PEnergy := 0.0;
KE := 0.0;
KE_theta := 0.0;
R_prev := R_min;

for p := 1 to NumParticles do
 begin
 {initialize Electric field vectors}
  Elec_x[p] := 0.0;
  Elec_y[p] := 0.0;
  Elec_z[p] := 0.0;
 end;
for i := 1 to NumParticles do
 begin
  v[i] := sqrt(sqr(vx[i]) + sqr(vy[i]) + sqr(vz[i]));     {=v_avg for ea particle}
  vtheta[i] := sqrt(sqr(vx[i]) + sqr(vy[i]));                {=v in xy direction only}
  KEnergy[i] := 0.5 * (M[i]) * sqr(v[i]);
  KE := KE + KEnergy[i];                                  {TOTAL Kinetic Energy}
  KEnergy_z[i] := 0.5 * (M[i]) * sqr(vz[i]);               {z direction Kinetic Energy}
  KEnergy_theta[i] := 0.5 * (M[i]) * sqr(vtheta[i]);         {xy direction Kinetic Energy}
  KE_theta := KE_theta + KEnergy_theta[i];                     {total xy direction Kinetic Energy}
  T_per[i] := (2.0e0 * pi * M[i]) / (Q[i] * B);           {period of rotation around B line}

  for j := 1 to NumParticles do
   begin
    If i <> j then
     begin
      rx := x[i] - x[j];
      ry := y[i] - y[j];
      rz := z[i] - z[j];
      R := sqrt(sqr(rx) + sqr(ry) + sqr(rz));     {distance between particles}
      R_min := Minimum(R_prev, R);              {set minimum R value}
      R_prev := R;        {to check R_min next time around}
      R3 := sqr(R) * R;
      PEnergy := PEnergy + ((Q[i] * Q[j]) / R) * (1 / (2 * FPE));      {div by 2 for double }
                                                           {counting}

      Ex := (rx / R3) * (Q[j] / FPE);
      Ey := (ry / R3) * (Q[j] / FPE);
      Ez := (rz / R3) * (Q[j] / FPE);
      Elec_x[i] := Elec_x[i] + Ex;         {electric field in x direction}
      Elec_y[i] := Elec_y[i] + Ey;         {  "     "    y   "}
      Elec_z[i] := Elec_z[i] + Ez;         {  "     "    z   "}
     end;  {i<>j}
   end;     {j := 1 to NumParticles}

  Electheta[i] := sqrt(sqr(Elec_x[i]) + sqr(Elec_y[i]));   {electric field in xy direction only}
 end;    {i:= 1 to NumParticles}

 KE_theta := KE_theta * 1.0E+22;         {total rotational kinetic energy}
 TotEnergy := (KE + PEnergy) * 1.0E+27;            {total energy}

end;                        {procedure Calc_Energies}

{*******************************************************}

procedure RSetUp;
{This procedure sets an Rmin value so that all subsequent R calculations will have a value to compare}
```

```
{to.}
{------------------------------------------------------------------}
{Amara Graps    4-3-91}
{------------------------------------------------------------------}

 begin

  rx := x[1] - x[2];
  ry := y[1] - y[2];
  rz := z[1] - z[2];
  R_min := sqrt(sqr(rx) + sqr(ry) + sqr(rz));      {distance between particles}

 end;       {procedure RSetUp}
{*******************************************************}


 procedure OpenPort3D;
{PURPOSE: This procedure sets up the 3D graphics port.  Since it is more convenient to }
{work with plotting real (extended) values, many of the 2D, 3D routines have been rewritten}
{and are in the unit:  UReal3D}
{--------------------------------------------------------------}
{F.Bennett     3-90}
{--------------------------------------------------------------}


 begin
  Open3DPortR(@gMyPort3D);

  IdentityR;
  LookAtR(vpLeft, vpTop, vpRight, vpBottom);
  PitchR(myPitch);
  YawR(myYaw);
  RollR(myRoll);
  ScaleR(scaleFactor, scaleFactor, scaleFactor);    {1st scaling.. will scale again later}
  ViewAngleR(focusLen);
  BackPat(background);
  EraseRect(thePort^.portRect);
 end;

{*******************************************************}

 procedure Axes;
{PURPOSE: This procedure draws the xyz axes in the  3D graphics port. It also uses the}
{3D routines in the unit UReal3D.}
{--------------------------------------------------------------}
{F.Bennett     3-90}
{--------------------------------------------------------------}


 begin
  PenPat(black);
  PenMode(patXor);
  TextMode(srcXor);

  if axesP <> 0 then
   begin
    moveto3dR(0, 0, 0);
    lineto3dR(axisLen, 0, 0);
    moveto3dR(axisLen + labelSpace, 0, 0);
    DrawChar('x');

    moveto3dR(0, 0, 0);
    lineto3dR(0, axisLen, 0);
    moveto3dR(0, axisLen + labelSpace, 0);
    DrawChar('y');

    moveto3dR(0, 0, 0);
    lineto3dR(0, 0, axisLen);
    moveto3dR(0, 0, axisLen + labelSpace);
    DrawChar('z');
   end;
  PenMode(patCopy);

 end;                         {procedure Axes}

{*****************************************************}

 procedure InitWrite;
{PURPOSE: This procedure write the "header" information to the 3 output files that contain}
{the important information for the thesis.}
{INPUT: All of the global position,velocity vectors, and energies}
{OUTPUT: Writes the initial parameters and other useful information.}
   {The following files will be written to:}
   { filename1 := 'Ions_positions.txt(BS)'}
   {filename2 := 'Ions_velocities.txt(BS)'}
   {filename3 := 'Ions_energies.txt(BS)'}
{-----------------------------------------------------}
{Amara Graps 5-90       Updated: 4-3-91         }
{-----------------------------------------------------}


  const
   SPosx = 'The x positions of particles are: ';
   SPosy = 'The y positions of particles are: ';
   SPosz = 'The z positions of particles are: ';
   SVelx = 'The x velocities of particles are: ';
   SVely = 'The y velocities of particles are: ';
   SVelz = 'The z velocities of particles are: ';
   SRstring = 'The distance between the particles are: ';
   SKE = 'The total kinetic energies of particles are: ';
   SKEz = 'The z kinetic energies of particles are: ';
   SKEt = 'The total rotational kinetic energy*1E+22 of particles is: ';
   STotE = 'The total energy*1E+27 of particles is: ';
   SPer = 'The periods of particles are: ';
   sp20 = '                    ';   {20 spaces}
   sp15 = '               ';   {15 spaces}
   fw = 16;    {fieldwidth}
   dp = 14;    {decimal places}


  var
   s: DecStr;
   f: decform;
   i: Integer;

 begin
```

```
{set up number to string conversion}
f.style := FixedDecimal;
f.digits := 0;

{FILE:  Ions_positions.txt(BS)' }
Writeln(fileout1, ' ');
Writeln(fileout1, 'Results file: ', filename1, ' using BS algorithm');
Writeln(fileout1, ' ');
Write(fileout1, SPosx);
for i := 1 to NumParticles - 1 do
 write(fileout1, x[i], ' ');
Writeln(fileout1, x[NumParticles]);
Write(fileout1, SPosy);
for i := 1 to NumParticles - 1 do
 write(fileout1, y[i], ' ');
Writeln(fileout1, y[NumParticles]);
Write(fileout1, SPosz);
for i := 1 to NumParticles - 1 do
 write(fileout1, z[i], ' ');
Writeln(fileout1, z[NumParticles]);
Writeln(fileout1, SRstring, R);
Writeln(fileout1, ' ');                   ------------------------------------------------------------------');
Writeln(fileout1, '--------------------------------------------------------------------------------------
Write(fileout1, 't            dt            ');
for i := 1 to NumParticles do
 begin
   {write out x titles}
  write(fileout1, 'x');
  num2str(f, i, s);       {convert integer i to string s using format f}
  write(fileout1, s);
  write(fileout1, sp15);
 end;
for i := 1 to NumParticles do
 begin
   {write out y titles}
  write(fileout1, 'y');
  num2str(f, i, s);
  write(fileout1, s);
  write(fileout1, sp15);
 end;
for i := 1 to NumParticles do
 begin
   {write out z titles}
  write(fileout1, 'z');
  num2str(f, i, s);
  write(fileout1, s);
  write(fileout1, sp15);
 end;
writeln(fileout1, '          R');             ------------------------------------------------------------------');
Writeln(fileout1, '--------------------------------------------------------------------------------------


{FILE:  Ions_velocities.txt(BS)' }
Writeln(fileout2, ' ');
Writeln(fileout2, 'Results file: ', filename2, ' using BS algorithm');
Writeln(fileout2, ' ');
Write(fileout2, SVelx);
for i := 1 to NumParticles - 1 do
```

```
 write(fileout2, vx[i], ' ');
Writeln(fileout2, vx[NumParticles]);
Write(fileout2, SVely);
for i := 1 to NumParticles - 1 do
 write(fileout2, vy[i], ' ');
Writeln(fileout2, vy[NumParticles]);
Write(fileout2, SVelz);
for i := 1 to NumParticles - 1 do
 write(fileout2, vz[i], ' ');
Writeln(fileout2, vz[NumParticles]);
Writeln(fileout2, ' ');
Writeln(fileout2,
      '-------------------------------------------------------------------------------------------------------------------
      --------------------');
Write(fileout2, ' t            ');
for i := 1 to NumParticles do
 begin
   {write out vz titles}
  write(fileout2, 'vz');
  num2str(f, i, s);
  write(fileout2, s);
  write(fileout2, sp15);
 end;
for i := 1 to NumParticles do
 begin
   {write out vtheta titles}
  write(fileout2, 'vtheta');
  num2str(f, i, s);
  write(fileout2, s);
  write(fileout2, sp15);
 end;
writeln(fileout2, ' ');
Writeln(fileout2,
      '-------------------------------------------------------------------------------------------------------------------
      --------------------');


{FILE:  Ions_energies.txt(BS)' }
Writeln(fileout3, ' ');
Writeln(fileout3, 'Results file: ', filename3, ' using BS algorithm');
Writeln(fileout3, ' ');
Write(fileout3, SKEz);
for i := 1 to NumParticles - 1 do
 write(fileout3, KEnergy_z[i], ' ');
Writeln(fileout3, KEnergy_z[NumParticles]);
Write(fileout3, SKE);
for i := 1 to NumParticles - 1 do
 write(fileout3, KEnergy[i], ' ');
Writeln(fileout3, KEnergy[NumParticles]);
Writeln(fileout3, SKEt, KE_theta : fw : dp);
Writeln(fileout3, STotE, TotEnergy : fw : dp);
Write(fileout3, SPer);
for i := 1 to NumParticles - 1 do
 write(fileout3, T_per[i], ' ');
Writeln(fileout3, T_per[NumParticles]);
Writeln(fileout3, ' ');
Writeln(fileout3, '------------------------------------------------------------------------------------------------');
Write(fileout3, ' t            ');
```

```
  for i := 1 to NumParticles do
   begin
     {write out KE_z titles}
    write(fileout3, '(1E+22)*KE_z');
    num2str(f, i, s);
    write(fileout3, s);
    write(fileout3, sp15);
   end;
  for i := 1 to NumParticles do
   begin
     {write out KE titles}
    write(fileout3, '(1E+22)*KE_');
    num2str(f, i, s);
    write(fileout3, s);
    write(fileout3, sp15);
   end;
  Writeln(fileout3, 'Tot xy KE*1E+22          TotE*1E+27          Del_TotE');
  Writeln(fileout3, '----------------------------------------------------------------------------------------');


 end;                    {procedure InitWrite}

{*****************************************************}
 procedure WriteResults;
{PURPOSE: This procedure writes the current calculated physical values for the ions}
{into 3 output files.}
{INPUT: All of the global position vectors.}
{OUTPUT: Writes results of position variables in the user-specified file.}
   {The following files will be written to:}
   { filename1 := 'Ions_positions.txt(BS)'}
   {filename2 := 'Ions_velocities.txt(BS)'}
   {filename3 := 'Ions_energies.txt(BS)'}
{--------------------------------------------------}
{Amara Graps 5-90       Updated: 4-3-91          }
{--------------------------------------------------}


  const
   tab = Chr(9);
   ret = Chr(13);
   fw = 16;     {fieldwidth}
   dp = 14;     {decimal places}

  var
   i: Integer;

 begin

  {FILE:  Ions_positions.txt(BS)' }
  Write(fileout1, t : fw : dp, tab, dt : fw, tab);
  for i := 1 to NumParticles do
   begin
     {write out x positions}
    write(fileout1, x[i] : fw : dp, tab);
   end;
  for i := 1 to NumParticles do
   begin
     {write out y positions}
    write(fileout1, y[i] : fw : dp, tab);
   end;
  for i := 1 to NumParticles do
   begin
     {write out z positions}
    write(fileout1, z[i] : fw : dp, tab);
   end;
  Writeln(fileout1, R : fw : dp);

   {FILE:  Ions_velocities.txt(BS)' }
  Write(fileout2, t : fw : dp, tab);
  for i := 1 to NumParticles do
   begin
     {write out vz }
    write(fileout2, vz[i] : fw : dp, tab);
   end;
  for i := 1 to NumParticles - 1 do
   begin
     {write out vtheta}
    write(fileout2, vtheta[i] : fw : dp, tab);
   end;
  Writeln(fileout2, vtheta[NumParticles] : fw : dp);    {last value in row}


   {FILE:  Ions_energies.txt(BS)' }
  Write(fileout3, t : fw : dp, tab);
  for i := 1 to NumParticles do
   begin
     {write out KEnergy_z }
    write(fileout3, (1E+22) * KEnergy_z[i] : fw : dp, tab);
   end;
  for i := 1 to NumParticles do
   begin
     {write out KEnergy}
    write(fileout3, (1E+22) * KEnergy[i] : fw : dp, tab);
   end;
  Writeln(fileout3, KE_theta : fw : dp, tab, TotEnergy : fw : dp, tab, Del_TotE : fw : dp);

 end;                      {procedure WriteResults}
{**********************************************************}

 procedure funder (bs_y: vecn6; var bs_dy: vecn6);
{PURPOSE: This procedure is used in the BS integration schema to provide the 1st order differential}
{equations.}
{INPUT: }
{     bs_y[]: BS vector which holds both the current positions and velocities of}
{             the n charged particles}
{     (global) Elec_x[], Elec_y[], Elec_z[]: Electric field vectors for each of the charged particles}
{     (global) Q[]: vector containing charge values for each of the particles.}
{     (global) M[]: vector containing masses for each of the particles}
{     (global) NumParticles: the number of charged particles}
{OUTPUT: }
{     bs_dy [ ] the velocities and accelerations using Lorentz 's force law of each of}
{             the charged particles}
{-----------------------------------------------------------}
{Amara Graps     3-19-91          }
{-----------------------------------------------------------}
```

```
 var
  Q_M: vece;
  i: integer;

 begin     {procedure funder}

{Note: The bs_y indices contain the positions and velocities in the following manner:}
{bs_y[1] = x[1]}
{bs_y[2] = vx[1]}
{bs_y[3] = y[1]}
{bs_y[4] = vy[1]}
{bs_y[5] = z[1]}
{bs_y[6] = vz[1]}
{Then for particle 2 etc we continue the bsy[] with the next index, e.g. bs_y[7] = x[2]....}

  for i := 0 to (NumParticles - 1) do
   begin

  {velocities}
   bs_dy[6 * i + 1] := bs_y[6 * i + 2];    {"vx"}
   bs_dy[6 * i + 3] := bs_y[6 * i + 4];    {"vy"}
   bs_dy[6 * i + 5] := bs_y[6 * i + 6];    {"vz"}

  {accelerations}
   Q_M[i + 1] := Q[i + 1] / M[i + 1];
   bs_dy[6 * i + 2] := Q_M[i + 1] * (Elec_x[i + 1] + bs_y[6 * i + 4] * B);     {"ax"}
   bs_dy[6 * i + 4] := Q_M[i + 1] * (Elec_y[i + 1] - bs_y[6 * i + 2] * B);     {"ay"}
   bs_dy[6 * i + 6] := Q_M[i + 1] * (Elec_z[i + 1]);    {"az"}

   end;    {for i := 0 to (number of particles)-1}


 end; {procedure funder}
{***************************************************************}

 procedure assign_quot (var quot: quotetype);
{PURPOSE: This procedure assigns the quot matrix for the difsub procedure (BS algorithm), the }
{values that "h" (timestep) will be divided by when smaller timesteps are needed.}
{Gear's Fortran difsub subroutine (in Numerical Initial Value Problems in }
{Ordinary Differential Equations, P-H, (1971).  ) assigned these values in a DATA statement}
{instead of what I have below.}
{INPUT: }
{     quot: blank matrix. }
{OUTPUT:}
{     quot: The values below are assigned.}
{---------------------------------------------------------}
{Amara Graps     3-12-91            }
{---------------------------------------------------------}

 begin
  ;
  quot[1, 1] := 1.0;
  quot[2, 1] := 2.25;
  quot[3, 1] := 4.0;
  quot[4, 1] := 9.0;
  quot[5, 1] := 16.0;
  quot[6, 1] := 36.0;
  quot[7, 1] := 64.0;
  quot[8, 1] := 144.0;
  quot[9, 1] := 256.0;
  quot[10, 1] := 576.0;
  quot[11, 1] := 1024.0;
  quot[1, 2] := 1.0;
  quot[2, 2] := 1.7777777777777777;
  quot[3, 2] := 4.0;
  quot[4, 2] := 7.1111111111111111;
  quot[5, 2] := 16.0;
  quot[6, 2] := 28.444444444444444;
  quot[7, 2] := 64.0;
  quot[8, 2] := 113.77777777777777;
  quot[9, 2] := 256.0;
  quot[10, 2] := 455.11111111111111;
  quot[11, 2] := 1024.0;

 end;    {procedure assign_quot}
{*******************************************************}

 procedure do_integrate (var jhvsv, m2: integer; j, jhvsv1, m, nn: integer; var dyn, ymaxsv, ysave, dy, ymax, yn,
          ynm1: vecn6; var tu, u, b, g, t: extended; var ymaxhv, ynhv, ynm1hv: vecn612);
{PURPOSE:This subroutine integrates by the midpoint method over the range H by 2*M steps.}
{This awful-looking procedure came from Gear's Fortran difsub subroutine in Numerical Initial Value }
{Problems in  Ordinary Differential Equations, P-H, (1971). }
{ INPUT: }
{  YSAVE: The initial values of Y are saved for a restart.}
{  G: (1/2) H/M to use in midpoint calculation}
{  B:}
{  DYN: The initial value of the derivative of Y.}
{  YMAXSV: The saved values of YMAX at the initial point.}
{  M: The number of pairs of sub steps which make up the step H.  M takes the sequence }
{  1,2,3,4,6,8,12,16 etc.}
{  T: independent variable for each Y}
{  NN: The number of first order differential equations}
{  JHVSV: The number of substep sizes for which half-way information has been saved.}
{  JHVSV1: The value of JHVSV from the previous cycle}
{ OUTPUT : }
{  U: the integrated Y by the midpoint rule (?)}
{  YNM1: Y(N-1), The previous value of Y in the midpoint method.}
{  YN: Y(N), The current value of Y in the midoint method.}
{  YMAX: The maximum values of the dependent variables are saved in this array.}
{  M2: SUM OF M's}
{  TU: temporary independent variable}
{  DY: An array of 10 locations which will contain the values derivatives on exit}
{  YNM1HV: The values of YNM1 at the midpoint of the basic interval if the number of sub steps}
{   is divisible by 4.  This information is used to avoid redding the integrationin case the step is halved.}
{  YNHV: The similar values of YN.}
{  YMAXHV: and the same for YMAX.}
{--------------------------------------------------}
{ Amara Graps    3-9-91  Updated: 3-27-91}
{--------------------------------------------------}

  var
   i, k: integer;

 begin    {procedure for_integ}

  if (j > jhvsv1) then
  begin

  {Integrate over the range H by 2*M steps of a midpoint method}

    for i := 1 to nn do
     begin
      ynm1[i] := ysave[i];
      yn[i] := ysave[i] + g * dyn[i];
      ymax[i] := ymaxsv[i];
     end;   {for i}

    m2 := m + m;
    tu := t;

    for k := 2 to m2 do
     begin
      tu := tu + g;

      funder(yn, dy);

      for i := 1 to nn do
       begin
        u := ynm1[i] + b * dy[i];
        ynm1[i] := yn[i];
        yn[i] := u;
        u := abs(u);
        if (u > ymax[i]) then
         ymax[i] := u;
       end;   {for i := 1 to nn}

      if (((k = m) and (jhvsv1 = 0)) and (k <> 3)) then
       begin
        jhvsv := jhvsv + 1;

        for i := 1 to nn do
         begin
          ynhv[i, jhvsv] := yn[i];
          ynm1hv[i, jhvsv] := ynm1[i];
          ymaxhv[i, jhvsv] := ymax[i];
         end; {for i := 1 to nn}

       end;{if (((k = m) and (jhvsv1 = 0)) and (k <> 3))}

     end;   {for k := 2 to m2}

   end   {if j > jhvsv1 }

  else
   begin

{The values of the midpoint integration were saved at the half way point in the previous integration. }
{Use them.}

    for i := 1 to nn do
     begin
      yn[i] := ynhv[i, j];
      ynm1[i] := ynm1hv[i, j];
      ymax[i] := ymaxhv[i, j];
      end;  {for i := 1 to nn}

    end;     {else}

  end;    {of procedure do_integrate}
{***************************************************************}

  procedure do_extrap (var ta, v, b, b1, c, u: extended; i, jodd, l, mf: integer; var extrap: vecn611; var quot:
          quotetype);
{-------------------------------------------------}
{PURPOSE: This subroutine performs extrapolation by either polynomial functions or rational }
{functions for the  Bulirsch-Stoer integration scheme. The extrapolation estimates what the }
{integrated y would have been had the stepsize h been allowed to go to zero.}
{This awful-looking procedure came from Gear's Fortran difsub subroutine in Numerical Initial }
{Value Problems in Ordinary Differential Equations, P-H, (1971). }
{ INPUT:}
{  MF: The method indicator.  The following are allowed:}
{     0: Bulirsch-Stoer rational extrapolation}
{     1: Polynomial extrapolation}
{  L:    The order of extrapolation}
{  EXTRAP(,): the most recent extrapolated values of Y}
{  QUOT(,): The quotients (H(I)/H(I+M))**2 used in the extrapolation.}
{  TA: the final value to be used in extrapolation process}
{  U: the integrated Y by the midpoint rule (?)}
{  V: The first extrapolated value}
{  C: saved TA variable}
{ OUTPUT:}
{  EXTRAP(,): the most recent extrapolated values of Y}
{  TA:}
{  V:}
{-------------------------------------------------}
{ Amara Graps    3-10-91         Updated: 3-27-91}
{-------------------------------------------------}

   var
    k: integer;

  begin      {procedure do_extrap}

   if (mf > 0) then
    begin

    {Perform the extrapolation by polynomial functions on the second and subsequent integrals.}

     for k := 2 to l do
      begin
       ta := ta + (ta - v) / (quot[k, jodd] - 1.0);
       v := extrap[i, k];
       extrap[i, k] := ta;
      end;  {k = 2, l}

    end      {if  mf > 0}

   else
    begin

      {Perform the extrapolation by rational functions on the second and subsequent integrals.}
      for k := 2 to l do
       begin
        b1 := v * quot[k, jodd];
        b := b1 - c;
        u := v;

        if (b <> 0) then
         begin
          b := (c - v) / b;
          u := c * b;
          c := b1 * b;
         end;{if b <> 0}

        v := extrap[i, k];
        extrap[i, k] := u;
        ta := ta + u;

       end;   {k := 2 to l}

     end;        {if mf  not > 0}

   end;  {of procedure do_extrap}
{*************************************************************}

   procedure difsub (nn: integer; var t: extended; var y, dy: vecn6; var h: extended; hmin, eps: extended; mf: integer;
          var ymax, error: vecn6; var kflag: integer; jstart, maxord, maxpts: integer);
{PURPOSE: This procedure integrates a set of nn differential equations using a variable timestep }
{scheme. The scheme works on the Richardson extrapolation principle where an initial h is used to }
{solve the equations.  Then h is split and we integrate the function with a smaller time step.  We }
{continue this process for up to 10 different time steps.  When the answer to the integration from }
{the current time step changes from the answer for the previous time step by less than a number: }
{"eps", we consider that integration  done and an extrapolation using rational fractions is used to }
{extend what the integration values ought to be if dt went to zero.}
{--------------------------------------------------}
{ From C.W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, P-H, (1971)}

{ Modified by Amara Graps to be a Pascal program and to have all calculations in extended precision.}
{ 3-5-91          Updated: 3-17-91}
{--------------------------------------------------}
{ INPUT:}
{  nn: The number of first order differential equations.}
{  t: The independent variable}
{  y: The dependent variables which usually contain both position and velocity.}
{  dy: An array  which will contain the derivatives on exit}
{  h: The step size that should be attempted. It may be  increased or decreased by this subroutine}
{  hmin: The minimum step size that should be allowed on this step.}
{  eps:     The error test constant.  The estimated errors are  required to be less than eps*ymax in }
{           each component. If ymax is originally set to +1 in each component, the  error test will be }
{           relative for those components greater than 1 and absolute for the others.}
{  mf: The method indicator.  The following are allowed:}
{     0: Bulirsch-Stoer rational extrapolation}
{     1: Polynomial extrapolation}
{  ymax: The maximum values of the dependent variables are  saved in this array.  It should be set }
{           to +1 before the first entry. (See the description of eps.)}
{  error: The estimated single step error in each component.}
{  kflag: A completion {ode with the following meanings:}
{     -1: The step was taken with h= hmin, but the requested error was not achieved.}
{     +1: The step was successful.}
{  jstart: An initialization indicator with the meaning:}
{     -1: Repeat the last step, restoring the values of y and ymax that were used the last time.}
{     +1: Take a new step.}
{  maxord: The maximum order of extrapolation allowed.  It must be less than 11.}
{  maxpts: The maximum number of different sub step sized used in  the extrapolation process.}
{OUTPUT:}
{  t:, y, dy, h, ymax, error, kflag, jstart}
{------------------------------------------------------------}

  label
   11, 16;

  const
   {Note: FMAX is a number smaller than the first integer that cannot be represented exactly in }
   {floating point.}
   fmax = 10000000.0;

  var
   ysave, ynm1, yn, dyn, ymaxsv: vecn6;
   quot: quotetype;
   extrap: vecn611;
   ynm1hv, ynhv, ymaxhv: vecn612;
   a, b, g, tu, u, v, ta, c, b1, quotsv, hchnge: extended;
   i, j, jodd, jhvsv, jhvsv1, konv, l, m, mnext, mtwo, m2, k: integer;
   exit_sub: boolean;

 begin     {procedure difsub}

  assign_quot(quot);     {assign value to matrix quot}

  If (jstart < 0) then
   {restore the values of y and ymax for a restart}
   for i := 1 to nn do
    begin
     y[i] := ysave[i];
     ymax[i] := ymaxsv[i];
    end  {for i}

  else
   begin
     {save the values of y and ymax in case a restart is necessary}
    for i := 1 to nn do
     begin
      ysave[i] := y[i];
      ymaxsv[i] := ymax[i];
     end;   {for i}

    funder(y, dyn);
   end;  {else..}

 11:

{The following counters and switches are used}
{  J: The count through the different sub steps G used}
{  JODD: 1 if J is odd, 2 if J is even}
{  JHVSV: The number of substep sizes for which half-way information has been saved.}
{  JHVSV1: The value of JHVSV from the previous cycle}
{  MNEXT: The next value of M}
{  MTWO: The next but one value of M}
{  QUOTSV: The last value of QUOT is irregular due to the fact that the sequence by the multiples 9/4,}
{     16/9 (odd) or 16/9, 9/4 (even until the final multiple of 4). However (H(0)/H(M))**2 is }
{     always M**2.  The regular value of QUOT is saved in QUOTSV, and replaced by M**2.}
{  KONV: Set to +1 initially, and reset to -1 if the error test fails.}

   jhvsv1 := 0;

16:

   kflag := 1;
   exit_sub := false;
   jhvsv := 0;
   a := h + t;
   jodd := 1;
   m := 1;
   mnext := 2;
   mtwo := 3;

   for j := 1 to maxpts do
    begin

     quotsv := quot[j, jodd];
     quot[j, jodd] := m * m;
     konv := 1;
     if (j <= (maxord / 2)) then
      konv := -1;

     if (j <= (maxord + 1)) then
      begin
       l := j;
       hchnge := 1.0 + Num2Extended(maxord + 1 - j) / 6.0;
      end

     else
      begin
       l := maxord + 1;
       hchnge := 0.7071068 * hchnge;
      end;  {else}

     b := h / Num2Extended(m);
     g := b * 0.5;

     do_integrate(jhvsv, m2, j, jhvsv1, m, nn, dyn, ymaxsv, ysave, dy, ymax, yn, ynm1, tu, u, b, g, t, ymaxhv, ynhv,
          ynm1hv);
     funder(yn, dy);

     for i := 1 to nn do
      begin

       v := extrap[i, 1];
        {Calculate the 2nd and subsequent columns in the extrapolation process.}

       ta := (yn[i] + ynm1[i] + g * dy[i]) * 0.5;
       c := ta;

        {Insert the integral as the 1st extrapolated value}
       extrap[i, 1] := ta;
    If (l >= 2) then
     begin

      If ((abs(v) * fmax) < abs(c)) then
       begin
        quot[j, jodd] := quotsv;

        If (abs(h) <= hmin) then
         begin
          kflag := -1;   {exit from subroutine}
          h := h * hchnge;
          t := a;
          exit_sub := true;
             {    writeln('First exit from difsub');}
          Exit(difsub);
         end;  {if abs(h) .le. hmin}

       end; {if abs(v)*fmax < abs(c)}

        {Perform the extrapolation}
      do_extrap(ta, v, b, b1, c, u, i, jodd, l, mf, extrap, quot);
     end;{if l >= 2}

    u := abs(ta);

    If (u > ymax[i]) then
     ymax[i] := u;
    error[i] := abs(y[i] - ta);
    y[i] := ta;

    If (error[i] > eps * ymax[i]) then
     konv := -1;

   end;   {i = 1 to nn}

  quot[j, jodd] := quotsv;

  If (konv > 0) then
   begin
   {Set flag for convergence and exit from subroutine}
    kflag := 1;
    h := h * hchnge;
    t := a;
    exit_sub := true;
      {writeln('Second exit from difsub');}
    Exit(difsub);

   end;   {if (konv >0)}

  jodd := 3 - jodd;
  m := mnext;
  mnext := mtwo;
  mtwo := m + m;

 end;          {for j := 1 to maxpts}

jhvsv1  := jhvsv;
```

```
  if (abs(h) <= hmin) then
   begin
   {h is at its floor value, so set kflag to "did not converge" and exit from subroutine}
    kflag := -1;
    h := h * hchnge;
    t := a;
    exit_sub := true;
     {writeln('Third exit from difsub');}
    Exit(difsub);
   end;    {if (dabs(h) <= hmin}

   {h is not at its floor value, so try again with smaller h}
  h := h * 0.5;
  if (abs(h) > hmin) then
   goto 16;
  h := CopySign(h, hmin);
  goto 11;

 end;    {procedure difsub}


{*****************************************************}

 procedure ShowTime (ticks: longint);
{PURPOSE: This procedure shows (in the corner of the plotting window) how much time the current }
{plotting took.  I'm (AG) not currently using this routine but I'm keeping it around in case I need it.}
{-----------------------------------------------------}
{F. Bennett    3-90}
{-----------------------------------------------------}

  var
   s: DecStr;
   f: decform;

 begin
  PenPat(black);
  TextMode(srcXor);
  f.style := FixedDecimal;
  f.digits := 2;
  num2str(f, ticks / 60, s);
  moveto(10, 15);
  drawstring(s);
  drawstring(' secs');

 end;        {procedure ShowTime}

{*****************************************************}

 procedure ShowDelE (DelE: extended);
{PURPOSE: This procedure shows (in the corner of the plotting window) what the relative}
{total energy difference is from the initial time to the time the plot was stopped.}
{-----------------------------------------------------}
{Amara Graps    3-27-91}
{-----------------------------------------------------}

  var
   s: DecStr;
```

```
    f: decform;

  begin
   PenPat(black);
   TextMode(srcXor);
   f.style := FixedDecimal;
   f.digits := 10;
   num2str(f, DelE, s);
   moveto(10, 15);
   drawstring('Del E= ');
   drawstring(s);

  end;                  {procedure ShowDelE}

{****************************************************}

  procedure  Assign_BSvar;
{PURPOSE: This procedure assigns new positions and velocities from the BS array (output from difsub).}
{From these we can calculate further quantities, plot the points, and store calculated physical results.}
{------------------------------------------------------------}
{Amara Graps     3-25-91}
{------------------------------------------------------------}

   var
    i: Integer;

  begin
   for i := 0 to (NumParticles - 1) do
    begin
     x[i + 1] := bsy[6 * i + 1];
     vx[i + 1] := bsy[6 * i + 2];
     y[i + 1] := bsy[6 * i + 3];
     vy[i + 1] := bsy[6 * i + 4];
     z[i + 1] := bsy[6 * i + 5];
     vz[i + 1] := bsy[6 * i + 6];
    end;      {for i := 0 to (number of particles)-1}
  end;

{****************************************************}
  procedure Do_write (var start_tick, k: Longint);
{PURPOSE: This procedure figures out if the current calculated results should be written to}
{the 3 output files. If it is, then it calls the procedure: WriteResults that does it. The criteria}
{for writing to the output files are contained in the boolean variable write_always.  If write_always}
{is true then we will always right to the output file, however if write_always is false, then we check}
{how much time has passed (by using start_tick), and if one hour has passed,  then we write the next  }
{100 steps of the integration. In addition I want to write out the 1st 100 steps, so these are checked}
{at the same time.  The 100 iterations are counted using the boolean variable 'write_on' and the}
{integer variable: 'write_count.}
{INPUT: }
{      start_tick:   integer value containing the time for the start of the integrations. }
{                 (in Mac tick units).}
{      k:    The current iteration number (step) for the integrations.}
{      (global) Write_always:    boolean variable that is true if we are supposed to write all results }
{                       for the integrations and false if we are only to write after an hour }
{                       has passed.}
{      (global) Write_on: boolean variable is true if we are in the middle of writing results for the case}
{      where we only write several times every hour.}
{      (global) write_count: integer that keeps track of how many steps we have written out for the }
{     case where write_on is true.}
{     (global) tick_frac: number of minutes between writing to the output files }
{OUTPUT: }
{     (global) Write_on:  (see above for criteria how this is set)}
{     (global) Write_count:  incremented by 1 if we are to continue writing to output files.}
{     Calls WriteResults}
{------------------------------------------------------------------}
{Amara Graps     3-20-91                  Updated: 4-3-91   }
{------------------------------------------------------------------}

  var
   ticks: Longint;

 begin

  if (write_always) then
   {always write results}
   begin
    WriteResults;
   end

  else
   begin
    ticks := tickcount - start_tick;    {tickcount = system variable}

    if (ticks >= 36000 * tick_frac) then
{Mac IIx clock = 36000 ticks/min...Mac + clock 3600 ticks/min?...Mac IIcx is what?}
     begin
       {write in files several times an hour}
      write_on := true;
      write_count := 0;
      Writeln(fileout1, 'Iteration: ', k);
      Writeln(fileout2, 'Iteration: ', k);
      Writeln(fileout3, 'Iteration: ', k);
      start_tick := tickcount;     {Set the tickcount to check the next time we want to write }
                                   {out values.}
     end;          {if time has passed}
    if (k = 1) then
     begin
       {Write the 1st 50 points}
      write_on := true;
      write_count := 0;
     end;     {if we're within the 1st 50 integrations}
    if (write_on) then
     {write the next 50 values}
     begin
      write_count := write_count + 1;
      If write_count <= 50 then
       WriteResults
      else
       begin
        write_on := false;
       end;      {else we've exceeded writing 50 values}
     end;     {if write_on..writing the next 50 values}

   end;     {else...write in files only specified times an hour}

 end;     {procedure Do_write}
```

```
{*****************************************************}

 procedure  ParticlePlot;
{PURPOSE: This procedure handles the fine details of the calculations. It integrates the differential }
{equations by calling the BS integrator (difsub), assigns the resulting positions and velocities }
{(AssignBSvar), calculates the electric fields and energies (Calc_Energies), handles the writing }
{to the output files (Do_write), and then plots, in 3D, the results.}
{INPUT:}
{OUTPUT:}
{--------------------------------------------------------------}
{Amara Graps    3-20-91          Updated: 3-31-91    }
{--------------------------------------------------------------}

  var
   i: Integer;
   j: Longint;
   old_x, old_y, old_z: vece;

 begin

  PenMode(patCopy);
  pensize(2, 2);

  for i := 1 to NumParticles do
     {save particles' position if we want to plot with lines connecting the points.}
   begin
    old_x[i] := x[i];
    old_y[i] := y[i];
    old_z[i] := z[i];
   end;

  old_TotE := TotEnergy;

   {Integrate differential equations}
  difsub(n6, t, bsy, bsdy, dt, hmin, eps, mf, ymax, error, kflag, jstart, maxord, maxpts);
  If (kflag = 1) then
   begin
       {we have achieved convergence w/BS algorithm. }
    Assign_BSvar;    {Assign new pos, vel's}
    Calc_Energies;   {Calculate energies, Elec fields}
    Del_TotE := abs(TotEnergy - old_TotE) / old_TotE;
{   t := t + dt;}
    mf := 0;     {set extrapolation to rational functions}
   end
  else
   mf := 1;         {set extrapolation to polynomial functions and try again}

  Do_write(start_tick, gStepsSoFar);       {write out results into output files if conditions }
                                           {are right}

  for i := 1 to NumParticles do
   begin
    MoveTo3DR(size * x[i] + x_offset, size * y[i] + y_offset, size * z[i] + z_offset);
     {MoveTo3DR(old_x[i] + x_offset, old_y[i] + y_offset, old_z[i] + z_offset);}
    PenPat(ParticlePat[i]);
    LineTo3DR(size * x[i] + x_offset, size * y[i] + y_offset, size * z[i] + z_offset);
   end;
  pensize(1, 1);

 end;             {of procedure ParticlePlot}

{*****************************************************}

 procedure ParticleInit;
{PURPOSE: This initializes the 3D graphport as defined in the UReal3D unit, i.e. plotting}
{calls can be made in extended precision.}
{-----------------------------------------------------}
{F. Bennett     3-90}
{-----------------------------------------------------}
 begin
  InitGrf3DR;

 end;           {procedure ParticleInit}

{*****************************************************}

 procedure ParticleRun;
{PURPOSE: This procedure handles the overall procedures for how the ion calculations are  }
{started and stopped.  (whereas ParticlePlot handles the fine details) . }
{-----------------------------------------------------------}
{Amara Graps     3-28-91           Updated: 3-30-91    }
{-----------------------------------------------------------}

  var
   ticks: longint;

  begin

   InitFiles;     {open the output files}
   GetVars;         {get initial values for all variables}
   BS_SetUp;           {stuff positions and velocities into BS vector}
   RSetUp;      {set R min value}
   Calc_Energies;     {calculate energies and Elec fields}
   start_E := TotEnergy;
   InitWrite;         {write preliminary info into output files}
   OpenPort3D;      {set up the 3D graphics port}
   gStepsSoFar := 1;
   start_tick := tickcount;      {tickcount = system variable}
   HideCursor;
   Axes;     {draw axes again}

   case GStepsTotal of

    0:       {step until mouse-click}
     begin
      ShowCursor;
      repeat
       ParticlePlot;
       gStepsSoFar := gStepsSoFar + 1;
      until EventAvail(everyEvent, gDummy);
      ShowDelE((TotEnergy - start_E) / start_E);   {shows conservation of energy during }
                                                    {this time}
      sysbeep(1);
     end;
  -1:  {step until R_end}
   begin
    R_start := R_min;       {set the starting R value}
    R_end := dist_frac * R_start;     {set the ending R value }
    while R_min <= R_end do
     begin
      ParticlePlot;
      gStepsSoFar := gStepsSoFar + 1;
     end;
    ShowDelE((TotEnergy - start_E) / start_E);
    sysbeep(1);
    ShowCursor;
   end;

  otherwise
    { step until gStepsTotal  (gStepsTotal > 0) }
   begin
    while gStepsSoFar <= gStepsTotal do
     begin
      {keep calculating until the number of steps: gStepsTotal is reached}
      ParticlePlot;
      gStepsSoFar := gStepsSoFar + 1;
     end;
    ShowDelE((TotEnergy - start_E) / start_E);  {shows conservation of energy during this time}
    sysbeep(1);
    ShowCursor;
   end;

 end;  {case gStepsTotal}


end;       {of procedure ParticleRun}

{*****************************************************}
 procedure ParticleAxis;
{PURPOSE: This subroutine just draws the axis for the startup of the Colliding ions program.}
{After this point the user can then resize the window or leave it as it is when selecting "Run"}
{or "Shrink" from the Control menu.}
{-----------------------------------------------------------}
{Amara Graps     3-28-91            }
{-----------------------------------------------------------}

 begin
  GetVars;
  OpenPort3D;
  HideCursor;
  Axes;
  ShowCursor;

 end;          {procedure ParticleAxis}

{*****************************************************}

 procedure ParticleStop;
{PURPOSE: This procedure occurs when the user selects "Stop" from the Control menu. It }
{stops the writing to the output files by simply closing them. I would like this routine to eventually }
{stop the particle plotting when "Stop" is selected, but it doesn't do that yet. }
```

```
{------------------------------------------------------------------------}
{Amara Graps     3-29-91             }
{------------------------------------------------------------------------}

 begin
  close(fileout1);
  close(fileout2);
  close(fileout3);

 end;              {procedure ParticleStop}

{**********************************************************}

end.
```

C. Example Input File CollidingIonsIn.text

```
-100     75      100     -40            vpLeft, vpTop, vpRight, vpBottom (3D look angles) <graphics>
120      135     -30                    myPitch(X), myYaw(Y), myRoll(Z) <graphics>
2.35                                 scaleFactor: scales port before axes are drawn <graphics>
30                                   focusLen <graphics>
1                                    BackPat(1-5) <graphics>
1        20      5                   axesP(0,1), axisLen, labelSpace <graphics>
6000                                 size    (magnification for points)
20.0     25.0    20.0                x_offset, y_offset, z_offset  <graphics>
-1                                   steps (0=infinity, -1= R_end)
3                            Number of Particles
0        5                   delay,  PenPat(1-5) <graphics>
0        4                   delay,  PenPat(1-5) <graphics>
0        3                   delay,  PenPat(1-5) <graphics>
1.0e-13                  hmin (sec): smallest timestep <BS integrator>
0.0                      t: beginning time <BS integrator>
1.0e-10                  dt (sec): timestep <BS integrator>
1e-12                    eps: error test constant <BS integrator>
0                        mf: 0 = rational extrap 1 = polynomial extrap <BS integrator>
1                        jstart: start with taking a new step <BS integrator>
10                       maxord: max # of columns in extrapolation table (order).. Up to 10.  <BS integrator>
10                       maxpts: # of H's (=H/2, H/4, etc) to use in extrapolation.  Gear recommends 10 <BS integrator>
2.0e0                    Magnetic field strength (teslas)
true                     true=write to output files at each step, false=write only every hour
15                       ticks fraction: # of minutes between file writes
1.1                      fractional distance of starting R_min = R_end
0.0      0.0      0.0      100.0   0.0      100.0   1.6e-19 1.673e-27
0.0      0.0      5.0e-4   0.0     0.0      -1.0    1.6e-19 1.673e-27
0.0      0.0      1.0e-3   0.0     100.0    -100.0  1.6e-19 1.673e-27
x (m)    y        z        vx(m/sec)        vy      vz      Q (coul)       M (kg)
```

BIBLIOGRAPHY

(1) P. G. Roll, R. Krotkov, and R. H. Dicke, Ann. Phys. **26**, 442 (1964).

(2) R. E. Brown, J. Camp, T. Darling, P. Dyer, T. Goldman, D. B. Holtkamp, M. H. Holzscheiter, R. J. Hughes, N. Jarmie, R. A. Kenefick, N. S. P. King, M. M. Nieto, D. Oakley, R. Ristinen, and F C. Witteborn, "A Measurement of the Gravitational Acceleration of the Antiproton: A Status Report," Los Alamos Internal Report, (1990 unpublished).

(3) Fred C. Witteborn, "Free Fall Experiments with Negative Ions and Electrons," Ph.D. thesis, Stanford University, 1965.

(4) Fred C. Witteborn and William M. Fairbank, "Apparatus for Measuring the Force of Gravity on Freely Falling Electrons," Rev. Sci. Instrum. **48**, 6 (Jan 1977).

(5) Roald K. Wangness, Electromagnetic Fields (Wiley, New York, 1979), pp. 577–591.

(6) Fred C. Witteborn (private communication), 26 April 1991.

(7) Fred C. Witteborn (private communication), 15 September 1989.

(8) Wolfgang K. H. Panofsky and Melba Phillips, Classical Electricity and Magnetism (Addison-Wesley, Cambridge, 1955), pp. 307–308.

(9) Fred C. Witteborn (private communication), 2 November 1989.

(10) Computer software Maple, Version 4.2.1 (Brooks/Cole, Pacific Grove, CA, 1989).

(11) Harvey Gould and Jan Tobochnik, An Introduction to Computer Simulation Methods, Part 1 (Addison-Wesley, Reading, 1988), pp. 108–110.

(12) Germund Dahlquist and Åke Björck, Numerical Methods (Prentice-Hall, Englewood Cliffs, NJ, 1974), pp. 346–347.

(13) F. S. Acton, Numerical Methods that Work (Harper and Row, New York, 1970), pp. 134–136.

(14) William H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Numerical Recipes (Cambridge University Press, 1986), pp. 561–568.

(15) Roland Bulirsch and Josef Stoer, "Numerical Treatment of Ordinary Differential Equations by Extrapolation Methods," Numerische Mathematik **8**, 1–13 (1966).

(16) Computer software Think Pascal, Version 3.0 (Symantec Corporation, Cupertino, 1990).