General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Produced by the NASA Center for Aerospace Information (CASI)
PLASM: A COMPUTER CODE FOR SIMULATING CHARGE-EXCHANGE PLASMA PROPAGATION

Final Report and Program Documentation

by

Raymond S. Robinson
William D. Deininger
Dale R. Winder
and
Harold R. Kaufman

January 1982

Department of Physics
Colorado State University
Fort Collins, CO 80523

This code was developed pursuant to Jet Propulsion Laboratory Contract No. 955322.
PLASIM: A COMPUTER CODE FOR SIMULATING CHARGE-EXCHANGE
PLASMA PROPAGATION

Final Report and Program Documentation

by

Raymond S. Robinson
William D. Deininger
Dale R. Winder
and
Harold E. Kaufman

January 1982

Department of Physics
Colorado State University
Fort Collins, CO 80523

This code was developed pursuant to Jet Propulsion Laboratory
Contract No. 955322.
ABSTRACT

The propagation of the charge-exchange plasma for an electrostatic ion thruster is crucial in determining the interaction of that plasma with the associated spacecraft. A model that describes this plasma and its propagation is described, together with a computer code based on this model.

The structure and calling sequence of the code, named PLASIM, is described. An explanation of the program's input and output is included, together with samples of both. The code is written in ANSI Standard Fortran IV.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>1</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>THEORY</td>
<td>3</td>
</tr>
<tr>
<td>PROGRAM STRUCTURE</td>
<td>15</td>
</tr>
<tr>
<td>PLASIM</td>
<td>18</td>
</tr>
<tr>
<td>BLOCK DATA</td>
<td>21</td>
</tr>
<tr>
<td>READER</td>
<td>22</td>
</tr>
<tr>
<td>INIT</td>
<td>22</td>
</tr>
<tr>
<td>CALC</td>
<td>25</td>
</tr>
<tr>
<td>CALCD</td>
<td>28</td>
</tr>
<tr>
<td>BOUND</td>
<td>28</td>
</tr>
<tr>
<td>WRIT</td>
<td>32</td>
</tr>
<tr>
<td>DS</td>
<td>32</td>
</tr>
<tr>
<td>VRSPL and VRSPLT</td>
<td>33</td>
</tr>
<tr>
<td>LNPLT and PLOTN</td>
<td>33</td>
</tr>
<tr>
<td>INPUT AND OUTPUT</td>
<td>34</td>
</tr>
<tr>
<td>Sample Input</td>
<td>36</td>
</tr>
<tr>
<td>VERIFICATION</td>
<td>38</td>
</tr>
<tr>
<td>Analytic Solution</td>
<td>38</td>
</tr>
<tr>
<td>Experimental Solution</td>
<td>38</td>
</tr>
<tr>
<td>Limitations in Use</td>
<td>45</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>51</td>
</tr>
<tr>
<td>APPENDIX A - ANALYTIC SOLUTION</td>
<td>53</td>
</tr>
<tr>
<td>APPENDIX B - BEAM CURRENT DENSITY PROFILES</td>
<td>59</td>
</tr>
<tr>
<td>APPENDIX C - GLOSSARY OF VARIABLES FOR PLASIM</td>
<td>67</td>
</tr>
<tr>
<td>APPENDIX D - COMPUTER CODE LISTING FOR PLASIM</td>
<td>75</td>
</tr>
</tbody>
</table>

*ORIGINAL PAGE IS OF POOR QUALITY*
LIST OF FIGURES

Fig. 1. Coordinate system and dimensions for simulation............ 6
Fig. 2. Geometry for evaluations of distances between paths........ 8
Fig. 3. Calling sequence for program PLASIM......................... 16
Fig. 4. Flow chart of main driver routine, PLASIM.................... 19
Fig. 5. Flow chart of initialization routine, INIT..................... 23
Fig. 6. Flow chart of main calculation routine, CALC.................. 26
Fig. 7. Flow chart of displacement calculation routine, CALCD........ 29
Fig. 8. Simulation for uniform density of charge-exchange ion production. Twenty trajectories simulated in three stages, electron temperature 7.0 eV in ion beam and 3.5 eV in charge-exchange plasma for 5 cm thruster........ 39
Fig. 9. Comparison of radial densities calculated using the computer code and analytic solution......................... 40
Fig. 10. Comparison of radial velocities calculated using the computer code and analytic solution.......................... 41
Fig. 11. Simulated and experimental surveys of electron density for a 5 cm thruster........................................ 42
Fig. 12. Simulated and experimental surveys of electron density for a 15 cm thruster........................................ 43
Fig. 13. Simulated survey of electron density for a 30 cm thruster.......................................................... 44
Fig. 14. Ion trajectories generated using data from the SAMPLE INPUT section for a 5 cm thruster......................... 46
Fig. 15. Ion trajectories generated using data from the SAMPLE INPUT section for a 15 cm thruster......................... 47
Fig. 16. Ion trajectories generated using data from the SAMPLE INPUT section for a 30 cm thruster......................... 48
Fig. 17. Ion trajectories generated by the computer simulation for a 15 cm thruster, one stage, 20 ion trajectories and 80 iterations. Other quantities same as in SAMPLE INPUT section.......................... 49
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 1A</td>
<td>Potential as a function of radial distance</td>
<td>56</td>
</tr>
<tr>
<td>Fig. 2A</td>
<td>Density as a function of radial distance</td>
<td>57</td>
</tr>
<tr>
<td>Fig. 3A</td>
<td>Velocity as a function of radial distance</td>
<td>58</td>
</tr>
<tr>
<td>Fig. 1B</td>
<td>Coordinate system for thruster and ion beam geometry</td>
<td>61</td>
</tr>
<tr>
<td>Fig. 2B</td>
<td>Charge-exchange ion production rates as a function of distance from the grids for extremes of possible beam current density distributions</td>
<td>65</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1B. Density of neutral propellant efflux, \( n(r,z)/n_{o,ref} \) .......................................................... 62

Table 2B. Charge-exchange ion production rates for different beam current density profiles ........................................ 66
INTRODUCTION

Ion thrusters can be used in a variety of primary and auxiliary space-propulsion applications. A thruster produces a charge-exchange plasma which can interact with various systems of the spacecraft. In order to understand these possible interactions, a detailed knowledge of the plasma propagation is required.

The production of charge-exchange ions by thrusters has been understood for some time. Fast ions from the thruster interact with slow neutrals that are also escaping, resulting in the production of ions that initially have only a thermal velocity. The electric fields within the ion beam cause these ions to move approximately radially out of the ion beam. These charge-exchange ions leave the ion beam along with electrons supplied by the neutralizer, the combination constituting the charge-exchange plasma. The propagation of the charge-exchange plasma depends on several factors, including the initial thermal energy of the ions, the distribution of ion production along the beam, and the potentials and geometry of neighboring spacecraft surfaces.

In the THEORY section of this report, the geometry of an idealized spacecraft with an ion thruster is described, together with the simplifications and definitions used in modeling the ion beam. The distribution function used for charge-exchange ion production is also presented, along with the barometric equation that relates the variation in plasma potential to the variation in plasma density. The numerical methods and approximations used for the calculations are then discussed. This section describes the main calculation subroutine, CALC, and the displacement calculation subroutine, CALCD.
In the PROGRAM STRUCTURE section, a flowchart is provided that diagrams the calling sequence of the modules; also presented are detailed descriptions of each of the modules. A guide to using the program is presented in the INPUT AND OUTPUT section of this report. Descriptions of the calculated results are presented in the INPUT AND OUTPUT and VERIFICATION sections.

Also presented is a method of obtaining better resolution in the upstream region. The high-resolution option of the program simulates only the upstream region. This option utilizes previously calculated trajectories as boundaries for the region to be simulated at higher resolution (see notes in the computer code).

The VERIFICATION section of this report also compares experimental and analytic results with those obtained by the computer code. Factors limiting simulation accuracy are also discussed.

An analytic solution is derived for the case of an infinitely long cylindrical beam with a uniform distribution of charge-exchange ion production along the beam. Expressions are obtained for the radial variations in ion density and velocity, permitting a direct comparison with results from the computer code. This analytic solution is described in APPENDIX A and used in the comparison described above.

It should be noted that this final report provides a complete description of the program and supersedes previous reports. All of the information necessary to use the program is contained herein. A glossary of the variables used in the computer code is provided in APPENDIX C and the computer code is listed in APPENDIX D.
THEORY

The interaction of an ion thruster with other components of an electrically propelled spacecraft through the plasma surrounding a spacecraft has been studied for some time. The transport of electrons from the ion beam to a solar-array surface was treated first by Knauer, et al. as an electron space-charge-flow problem. Measured electron currents, though, were found to be much higher than calculated by Knauer. The difference was due to the presence of a charge-exchange plasma.

Charge-exchange ions are produced when fast beam ions pass near slow escaping neutrals. The fast neutrals that result usually present no problem, and escape following the directions they had as ions. The slow charge-exchange ions that are produced, though, initially have only the velocity of the thermal neutrals. Small electric fields within the ion beam result in the charge-exchange ions leaving the beam in approximately radial directions. These charge-exchange ions, together with some escaping electrons, form the charge-exchange plasma that surrounds an electrically propelled spacecraft.

The production rate for the charge-exchange ions was first calculated by Staggs, et al. The capability of the charge-exchange plasma to transport electrons to other parts of the spacecraft was experimentally evaluated by Worlock, et al. Some detailed trajectories of charge-exchange ions have been examined by Komatsu, et al. Experimental studies of the charge-exchange plasma distribution, particularly upstream of the ion-beam direction, have been conducted by Kaufman, and Carruth, et al. Several studies included a correlation of plasma properties in terms of the distance from the thruster and the angle relative to the
beam direction. Theoretical studies of the charge exchange plasma have been carried out by Robinson, et al. and Katz, et al. The latter treat the ions, numerically, as a cold fluid in contrast to the use of calculated ion trajectories and density gradients.

The physical processes involved in the charge-exchange plasma have become well understood as a result of the various studies that have been conducted. The electron population outside of the beam agrees with the "barometric" equation

$$n_e = n_{e,ref} \exp(-qV/kT_e)$$

which was introduced by Sellen, et al. and verified by Ogawa, et al. for the population within the beam and by Kaufman for the population in the charge-exchange plasma. The plasma potential $V$ in Eq. (1) is taken to be zero at the reference electron density $n_{e,ref}$. The electron temperature $T_e$ in the charge-exchange plasma has been found to be about half the value in the ion beam. The electron temperature in the ion beam varies with thruster size and ranges from about 7 eV for a 5 cm thruster to 5 eV for 15 cm and 0.35 eV for 30 cm. Also, $q$ is the elementary unit of charge and $k$ is Boltzman's constant.

The experimental validity of Eq. (1) is consistent with the low density and long mean-free paths in the charge-exchange plasma. The decreasing plasma density with increasing distance from the thruster forms a potential well for the electrons, so that many transits of this region are probable before an electron escapes. The many transits permit randomization of the electron population to a single Maxwellian distribution by Coulomb collisions.
The extent of the charge-exchange plasma is large compared to the Debye shielding distance, which means that the electron density must everywhere be equal to the ion density. Inasmuch as the ions only move outward from the thruster, their motion is essentially collisionless and governed by the potential distribution from Eq. (1).

The approach used in this study has been to assume a cylindrical, axially symmetric ion beam, with the charge-exchange ions leaving the beam with a uniform velocity in the radial direction. The coordinates and simulation boundaries are defined in Fig. 1. The current density of these charge-exchange ions at the cylindrical beam boundary is a function of the distance downstream from the thruster. The total charge-exchange current is distributed among the total number of trajectories, with this total number specified as an input parameter, N. Approximately fifty percent of the charge-exchange ions are generated within one beam radius of the downstream end of the thruster, so about half of the specified trajectories will initially start in this region.

A trajectory represents the path of a single representative ion on which acceleration is produced by electric fields in the plasma. These fields correspond to potential gradients induced by gradients in the plasma density, as indicated by Eq. (1). Density gradients are used in two separate calculations. The component of the density gradient along the path provides a potential gradient which serves to change the ion velocity in that direction, while the component of the density gradient normal to the path provides a potential gradient which modifies the direction of the path. The forces on the ions acting normal and parallel to the path are resolved into x and z components to obtain the resultant force acting on the ion.
Fig. 1. Coordinate system and dimensions for simulation.
In the simulation used herein, the ion path is represented with a stepwise progression away from the beam and the trajectories are advanced from left to right starting from the end of the thruster. It is assumed that, with small enough step sizes, following the ions through one pass of calculations is sufficient. (The validity of this assumption is partially checked later by comparison with experimental and analytic results.) From a physical viewpoint, the ions are moving at, or above, acoustic velocity, so disturbances should not propagate in the upstream direction. Also, the extent of the plasma is very large compared to the Debye shielding distance, hence electric fields at the flow boundaries should not extend into the bulk of the plasma. The distances to neighboring paths are important parameters, in that they are used to determine densities. As indicated in Fig. 2, a normal to the path being incremented is extended in both directions. This normal is used to calculate the distances to neighboring paths, on the right and left of the path being incremented. In this report, right and left are defined in terms of relationships upon leaving the ion beam, with the viewing direction in the direction of charge-exchange ion motion. The distances to the neighboring paths are obtained by calculating the distance from the path being incremented, along the normal to this path, to the neighboring path where the normal intersects it. If the neighboring path was not intersected by a normal, as in the right side of Fig. 2a, then the neighboring path is extended linearly from the last interval and the distance is calculated. If the normal intersects a neighboring path below the final two ion positions on that path, as is the case for the left hand paths in Figs. 2a and 2b, back stepping is used. This allows the distance to be calculated to the line the normal actually intersects.
(a) Linear extrapolation used on right, back stepping used on left.

(b) Paths intersected both sides (path iterated again), back stepping used on left.

Fig. 2. Geometry for evaluations of distances between paths.
If intersections are found on both the right and left, as in Fig. 2b (linear extrapolation not used), the path currently being iterated will be iterated again so it can keep in step with its neighbors. Finally, if the normal intersects the neighboring path between the final two ion positions, as in the right hand path of Fig. 2b, the intersection point is used as it is.

The density is inversely proportional to the distance between neighboring paths, the ion velocity and the radial distance. The latter relationship is due to the axial symmetry and the use of only one trajectory for each axial location. The density on the left is thus given by

\[ n_L = \frac{C}{\Delta d_L x v_i}, \]  

(2)

where \( C \) is a constant depending on operating conditions and the number of trajectories specified, \( \Delta d_L \) is the distance between the path being incremented and the path on the left, \( x \) is the radius and \( v_i \) is the ion velocity. The density on the right is defined in a similar manner, except that \( n_R \) and \( \Delta d_R \) are used. After the radial distance, \( x \), and the ion velocity, \( v_i \), are calculated, the densities to the right and left are calculated using Eq. (2). The two quantities, \( n_L \) and \( n_R \), are averaged to get the plasma density at the point under consideration.

The plasma potentials to the right and left are then calculated using Eq. (1). Here \( n_e \) represents the density just calculated to the right or left and \( n_{e, \text{ref}} \) represents the initial density to the right or left of the path being incremented. The force normal to the path direction is then,

\[ F_\perp = -q v_\perp / \Delta d_s \]  

(3)
where $\Delta V_\perp$ is the difference between the potentials on the right and left and $\Delta d_s$ is the smaller of $\Delta d_L$ and $\Delta d_R$. This choice for $\Delta d_s$ was included to accommodate radical changes in the displacements arising when the perpendicular displacement intersects a boundary. The effect of this choice as compared to averaging the displacements was found to have a negligible effect on interior paths. This force is resolved into $x$ and $z$ components.

The force acting parallel to the path can be calculated in a fashion similar to Eq. (3),

$$F_\parallel = -q\Delta V_\parallel / \Delta d_p$$

(4)

where $\Delta V_\parallel$ is obtained through Eq. (1) with potentials being those at the point presently under consideration and the previous point and $\Delta d_p$ the distance between these two points. This force is also resolved into $x$ and $z$ components.

The normal and parallel forces can also be set equal to the rate of change of momentum:

$$F_\perp, _\parallel = m\Delta v_\perp, _\parallel / \Delta t$$

(5)

with $m$ the ion mass, $\Delta v_\perp, _\parallel$ the velocity component generated normal or parallel to the path direction, and $\Delta t$ the size of the time interval used in the iteration. Equating these two force expressions yields

$$\Delta v_\perp, _\parallel = F_\perp, _\parallel \Delta t / m$$

(6)

These are the velocity changes normal and parallel to the path direction for the present iteration. Similar expressions hold for the $x$ and $z$ components.
The components of the velocity, $\Delta v_x$ and $\Delta v_z$, where $F_{x,z}$ is used instead of $F_L$. The total velocity and its components are calculated from $\Delta v_x$ and $\Delta v_z$. The new ion position is then calculated, using linear expressions, thereby incrementing the path.

It was necessary to consider several special cases in the execution of the displacement algorithm. Three were mentioned in reference to Figs. 2a and 2b in discussing the calculation of distances to neighboring paths. Those were linear extrapolation, back stepping and iterating a path again if intersections were found on both sides. Other cases involve the extreme right and left trajectories that have not intercepted a boundary. Without special treatment, these cases would result in an undefined density on one side of the path because the normal will intercept a boundary instead of another path. The boundary is treated as another path with one exception. If at any time a path would be repelled by the boundary, the direction is left unchanged. This approximates a plasma sheath which would be present at such a boundary.

In general, both the distance between trajectories and the distance from a trajectory to a boundary will be much larger than the Debye distance. The accuracy of the simulation should therefore be considered questionable at any location where the distance between trajectories approaches the distance to a boundary. A better approximation in such a location might be obtained by extrapolating from deeper within the charge-exchange plasma. It would also be possible to use more trajectories, so that the space between them would be reduced.

The distribution of charge-exchange ion production along the axis is assumed to be proportional to the neutral density on the axis. This neutral density for a single thruster (no overlap of neutral effluxes from adjacent thrusters) is $^7,^8$ (see Appendix B).
where \( r_b \) is the beam radius and \( n_o \) is a constant for a given combination of beam diameter and neutral loss rate. This function decreases rapidly with increasing \( z \), due to the rapid divergence of neutral atom paths in free molecular flow. The beam radius, \( r_b \), is an important parameter in this simulation, because approximately half of the total charge-exchange production occurs within about one beam radius of the thruster. This means that half of the charge-exchange trajectories will originate within the same distance.

In determining the locations for the origin of ion trajectories along the axis, the integral of Eq. (7) is used

\[
\int_0^\infty n(z)dz = n_o r_b .
\]  

(8)

The region simulated is finite, so that not all of the integral can be represented. The region to be simulated was defined so that 95 percent of \( n_o r_b \) is contained within this region. For \( N \) trajectories making up 0.95 \( n_o r_b \), with each trajectory located at the median of the density interval that it represents, the following expression holds

\[
\frac{1}{2} \sum_{i=0}^{2N} \int_{z_i}^{z_{i+1}} n(z)dz = \sum_{i=0}^{2N} \int_{z_i}^{z_{i+1}} n(z)dz = 0.95 n_o r_b / 2 .
\]  

(9)

For the first trajectory, the starting point is at the right end of the thruster (left end of the ion beam, see Fig. 1). To calculate each successive \( z \), the expression used is

\[
0.95r_b / 2N = z_{i+1} - (z_{i+1}^2 + r_b^2)^{1/2} - z_i + (z_i^2 + r_b^2)^{1/2} .
\]  

(10)
For the first trajectory, \( i = 0 \) and \( i + 1 = 1 \). The value of \( z_0 \) is the right end of the thruster and the first trajectory is started at \( z_1 \). The second trajectory is started at \( z_2 \), third at \( z_3 \), and so forth.

The initial velocity upon leaving the ion beam is the Bohm velocity,

\[
V_B = \left( \frac{kT_e}{m_i} \right)^{1/2},
\]

where \( T_e \) is the electron temperature in the ion beam and \( m_i \) is the ion mass. The constant, \( C \), in Eq. (2) is obtained using the following procedure. The total production rate of charge-exchange ions, for a uniform beam current density profile, is given by\(^6\)

\[
\dot{N}_{ce} = 2J_b^2(1-n_u)\sigma_{ce}/\pi r_b n_u q^2\bar{v}_o,
\]

where \( J_b \) is the ion-beam current (A), \( n_u \) is the propellant utilization, \( \sigma_{ce} \) is the charge-exchange cross section \((\text{m}^2)\), \( r_b \) is the beam radius \((\text{m})\), \( q \) is the absolute electronic charge \((\text{C})\), and \( \bar{v}_o \) is the mean neutral thermal velocity, \((8kT_o/\pi m_o)^{1/2} \text{ (m/sec)}\). With typical values for Hg neutrals and singly charged ions used,

\[
\dot{N}_{ce} = 6.18 \times 10^{16} J_b^2(1-n_u)/r_b n_u.
\]

The charge-exchange cross section usually decreases with ion energy.

The value used for Eq. (13) corresponds to Hg\(^+\) ions at about 1,000 eV.

The plasma density can be related to this production rate by

\[
n = \dot{N}_{ce}/2\pi A d_m x v_1 N,
\]

\( A \) represents the cross sectional area and \( d_m \) is the mean molecular weight of the neutral gas.
where $\Delta d_m$ is the local mean spacing between trajectories (m), $x$ is the radius (m), $v_i$ is the local ion velocity (m/sec) and $N$ is the number of trajectories simulated. Substituting for $N_{ce}$,

$$n = \left( \frac{J_b^2 (1-\eta_v) q_{ce}}{N_{re} \eta_v q^2 \pi^2 v_0} \right) \frac{1}{x \Delta d_m v_i}$$

where the quantity enclosed in the parentheses is the constant $C$. 
PROGRAM STRUCTURE

The simulation is performed by a computer program written in standard Fortran IV that is listed in APPENDIX D. The main driver program, PLASIM, establishes the sequencing of computations in the simulation and makes calls to the major subroutines which perform the required calculations. This overall flow is illustrated with a flow chart in Fig. 3.

First, all of the fixed parameters are stored in the COMMON blocks by the subprogram BLOCK DATA. The control parameters which define the type of computation, disposition of results and termination, along with physical data used in the simulation, such as the thruster and accelerator system dimensions and the plasma characteristics, are input through subroutine READER. The parameters and data are read from card images.

For a typical simulation, the next call is to subroutine INIT, which initializes the constants and arrays, calculates the coordinates of ion trajectories (paths) at the beam edge and performs the first iteration, thus calculating the second set of coordinates on the paths. Calls are then made to subroutine WRIT to output the heading, thruster schematic, initial parameters and results of the initialization and first iteration.

PLASIM next begins the staging and successive iterations by calls to subroutine CALC. Subroutine WRIT is also called to output a heading for information on the progress of the computation. After the completion of a specified number of iterations, termed a stage, the contents of the coordinate and density arrays are written to an external mass storage file, PATHS. The arrays are reinitialized and the next stage commenced.
Fig. 3. Calling sequence for program PLASIM.
After completion of all the stages, the plotting routines are called according to the status of the control parameters set at the beginning. Another call to READER is made to determine the next action, either another simulation run or a normal termination.

The main calculation subroutine, CALC, sets up an accounting procedure for the paths, the iteration number, the activity of the path, and the boundary condition, in addition to carrying out calculations leading to new ion positions. Displacements from the current path to adjacent paths are returned by a call to subroutine CALCD and then the densities on both sides of the present path are calculated. The potentials, on both sides of the path and at the present and prior ion locations, are obtained, followed by the calculation of forces acting perpendicular and parallel to the path, respectively. The forces are resolved into z and x components to calculate the new z and x components of the ion velocities. A new ion location is computed from the z and x velocities and a call is made to subroutine BOUND to ascertain if it is within the simulation boundaries.

Two plotting methods are included, one for a line printer, LNPLT, and one for a Versatec electrostatic plotter, VRSPL. The subroutines called in VRSPL are described in the appropriate literature. These may be not only device-dependent, but site-specific as well. They are included to indicate a possible preparation procedure to use in presenting the simulation results in graphical form.

There are several error detection segments in the code which output informative messages whenever unrealistic values appear in certain variables (see subroutine BLOCK DATA).
There are three types of simulations defined by the control parameter KEY. KEY=-1 generates a normal simulation with a non-uniform initial density distribution, KEY=1 generates a simulation with a uniform initial density distribution and KEY>0 generates an upstream simulation which utilizes a given path from a previous run as a boundary and simulates paths upstream from that path. A call with KEY=0 causes a normal termination including normal clearing of the plotting buffers. The following subsections discuss the various subroutines in detail.

PLASIM

This is the main driver routine for the simulation of a charge-exchange plasma surrounding a broad beam ion thruster. This routine controls and sequences the computation and defines the program structure and staging by calling various subroutines. It is outlined in Fig. 4. The first statement (two lines) of the program is a non-ANSI statement which declares the files required for input and output. This will be converted to a comment statement to prevent errors because of non-standard FORTRAN.

READER: The first call is to subroutine READER which reads in the run specifications and information. INIT: The next call is to subroutine INIT which initializes various parameters and performs the first iteration. CALC: The next call is to subroutine CALC which computes the ion positions along the trajectories. CALC is called NUMIT times. This completes the first stage. (NUMIT is the maximum number of iterations to be performed on any given path during any particular stage.) Information for the first (NUMIT - 10) iterations is written to an external file, PATHS. Core memory is then reinitialized by transferring the results of the final ten iterations to the core space for the
Fig. 4. Flow chart of main driver routine, PLASTIM.
write contents of core to external file
CALL WRIT (5)

reinitialize core

Fig. 4. Continued, PLASIM.
initial ten. The results of these ten iterations act as a base for another set of \((\text{NUMIT} - 10)\) iterations. \text{CALC} is called another \((\text{NUMIT} - 10)\) times to fill core storage with another set of \text{PATH} coordinates which completes the second stage. Then another set of \((\text{NUMIT} - 10)\) results are written to \text{PATHS}, core is reinitialized again and another stage is run. This is done until all paths intersect boundaries. Finally, the plot indicator, \text{ICLPLT}, is checked to determine which plotting routine to use to output a plot of the \text{PATH} coordinates, if any.

Two blank cards are always placed at the end of the data deck to signal a stop. The word size dependent variables are \text{IFI}, \text{IF2}, \text{INFO}, \text{ITITL}, and \text{IW}. See BLOCK DATA for instructions on word size dependent variables and code generated error messages.

**BLOCK DATA**

This subprogram loads data into labeled common storage at compile time through the data statements. Error message information and the instructions on word size dependence have been included. This dependence was motivated by the requirement for transfer of alphanumeric data for labels to the plotting subroutines of Versatec and IMSL. The basic unit for input was chosen to be 80 characters, a full data card. These 80 characters must be packed continuously into words of an array which will be transferred to the external plotting subroutines. The arrangement described below accomplishes this but requires a change in the DATA statements for computers with word size not equal to 10 characters.

The word size dependent variables are: \text{IFI}, \text{IF2}, \text{IW}, \text{INFO}, and \text{ITITL}. \text{IW} is the number of words required to generate 80 characters. To convert to a machine using different word size, modify only \text{IW}, \text{IFI} and \text{IF2} in first two data statements below. The third data statement
is for I/O buffers and the fourth data statement is for values of constants.

DATA IW, IF1(1), IF1(2) /8, 6H(8A10), 1H /
DATA IF2(1), IF2(2), IF2(3), IF2(4) /10H(8A10/2(4A,4H10)),1H ,1H /
DATA IN, IOUT, IPATHS /5, 6, 7/
DATA BK, Q, PI /1.3806E-23, 1.602E-19, 3.141593/

When a code generated error message is called, the first line of the error output will be of the form,

```
****** ERROR NNN ******
```

where 'NNN' is one of the following integers,

- 207 See Subroutine WRIT
- 410 See Function Subroutine DS
- 412 See Function Subroutine DS
- 521 See Subroutine CALCD
- 522 See Subroutine CALCD
- 523 See Subroutine CALCD
- 524 See Subroutine CALCD
- 525 See Subroutine CALCD
- 526 See Subroutine CALCD
- 527 See Subroutine CALC
- 528 See Subroutine CALCD
- 529 See Subroutine CALCD
- 530 See Subroutine CALC
- 610 See Subroutine BOUND
- 612 See Subroutine BOUND

and the referenced subroutine is the calling routine.

READER

See INPUT AND OUTPUT section.

INIT

This subroutine initializes the necessary variables and performs the first iteration. It is outlined in Fig. 5. The constants, which include the Bohm velocity, velocity of the thermal neutrals and the step size, are defined first. Then the z and x coordinates of the ion
initialize constants

calculate x,z coordinates of ion exit points from beam, define ZBOUND

KEY >0 ?
Yes  high resolution upstream run redefine ZBOUND
No

KEY =-1 ?
Yes  uniform density distribution run recalculate x,z coordinates
No

initialize velocity array calculate second set of ion positions

initialize density arrays

1

Fig. 5. Flow chart of initialization routine, INIT.
Fig. 5. Continued, INIT.
trajectory exit points are obtained and ZBOUND is defined. If this is a
high resolution upstream run (KEY>0), ZBOUND is redefined. If this is a
uniform density run (KEY=-1), the ion trajectory exit points are redefined
to be uniformly spaced. Following this, the velocity arrays and iteration
counter are initialized, the second ion positions on each path are calculated
and all the paths are set to active. The initial densities between the
paths and the initial densities on the paths are then calculated. Finally,
the heading, thruster schematic, initial parameters and results of the init-
ialization and first iteration are output, then control is returned to PLASIM.

CALC

This is the main calculation routine. It is outlined in Fig. 6.
This subroutine uses the arrays in blank common along with subroutine
CALCD to determine the next position of the ion being considered. The
densities and potentials to the right and left of the present path are
calculated first. Then the force acting perpendicular to the present
path is calculated. If the boundaries repel the path, the perpendicular
force is set equal to zero. The potentials and forces acting parallel
to the present path are obtained next. The total force (sum of perpen-
dicular and parallel components) acting on the ion is calculated and then
the velocity components along the x and z axes are obtained. The speed on
path one is normalized to 1.2 times the speed on path two when the speed
on path one is 20% greater than the speed on path two. Finally the next
ion position is calculated. Subroutine BOUND is called to make sure the
new ion position is inside the boundaries. The results are printed every
ICLWRT iterations. IFLAG1 is checked (see CALCD) to see if intersections
were found to both the right and the left. If intersections were found
to both sides, the path is iterated again.
Fig. 6. Flow chart of main calculation routine, CALC.

ORIGINAL PAGE IS OF POOR QUALITY
1. calculate potential on right and left
   - calculate force acting perpendicular to path
   - calculate potentials at present points
   - calculate force acting parallel to path
   - calculate total force and components
   - calculate total velocity and components

   if necessary, normalize speed on path 1 to 1.2 times speed on path 2

   if necessary, normalize speed on path 1 to 1.2 times speed on path 2

2. first trajectory?
   Yes
   No

3. calculate new ion positions
   make sure new ion positions within simulation boundaries
   CALL BOUND

   output results of iteration if desired
   increment iteration counter
   check flags

Fig. 6. Continued, CALC.
CALCD

This subroutine, which is outlined in Fig. 7, computes the perpendicular displacement from the present path to the adjacent paths. The flags are defined in this routine.

First, the flags are initialized and then the slope of the line perpendicular to the present path, at its end point, is calculated. If the path under consideration is the right- or left-most active path, function subroutine DS is used to get the displacement to the boundary.

When an interior path is under consideration, displacements are obtained to the right- and left-hand paths. Here, the slope of the adjacent path, between the last two ion positions, is calculated. The intersection point between the line perpendicular to the path under consideration and the line formed by the last two points of the adjacent path, is calculated. Tests are run to determine if linear extrapolation (adjacent path extended to find an intersection point) or back stepping (interpolation between points on the adjacent path previous to the last two points) is to be used. The flags are set, the displacement of the adjacent path is calculated and then control is returned to subroutine CALC. This subroutine returns the values of the flags, the displacements to the right and left and the angle that the perpendicular makes with the horizontal, to subroutine CALC.

BOUND

This subroutine checks the point \((z,x)\) to see if it lies inside the defined boundaries of the simulation. If \((z,x)\) does lie inside the defined boundaries, no changes are made and control is returned to subroutine CALC. If the point lies outside the boundaries, the path status is set equal to the iteration number. In addition, if \((z,x)\) lies on the
ENTER

initialize flags

- calculate slope of line perpendicular to current path at the end point
- calculate angle the perpendicular makes with the horizontal

calculate displacement to left

left most path? Yes

calculate displacement to boundary on left (IFLAG3), use function DS

No

- set iteration index of left hand path
- make sure left path is active (IFLAG2)

- calculate slope of left hand path between final two ion positions

Yes intersections be found? 2

decrease iteration index of left path by one

No

1

3

Fig. 7. Flow chart of displacement calculation routine, CALCD.
Fig. 7. Continued, CALCD.

1. Calculate displacement to right
2. Calculate intersection of left path line and perpendicular to present path
3. Is intersection point adequate? (IFLAG1, IFLAG2)
4. Set iteration index of right hand path
   Make sure right path is active (IFLAG2)
   Calculate displacement to left
5. Right most path?
   Calculate displacement to boundary on right (IFLAG3) use function DS
   Return
calculate slope of right path between final two ion positions

- calculate intersection point of right hand path line and perpendicular to current path

- is intersection point adequate? (IFLAG1, IFLAG2)

- calculate displacement to right

- define IFLAG3

Fig. 7. Continued, CALCD.
first or last active path and lies outside the boundaries, no changes are made and control is returned to subroutine CALC.

WRIT

This is the main output routine. It consists of five subsections with the value of the passed parameter, KE, determining which section is called. KE=1: the first three pages of output are generated, consisting of the heading, a thruster schematic and the constants and input data. KE=2: the interim status of the major variables is output from subroutine INIT. KE=3: the heading is printed for the results of one pass of calculations. KE=4: a file of path coordinates is created to be used in a high resolution upstream run. KE=5: a file of position-density triplets is created to be used for plotting (see INPUT AND OUTPUT section).

DS

This function subroutine finds the perpendicular displacement from the first or last active path to the boundary. CALCD constructs a perpendicular to the present path at the present point \((z, x)\) with a slope of SLOPEP. Function DS then extrapolates this perpendicular of slope SLOPEP to the left or right (depending on whether the first or last active path is considered) and finds the \(z\) and \(x\) intercepts (ZINT and XINT) along a boundary line. ZINT and XINT are checked to see if they lie on or between the boundary endpoints on the boundary line. If they do, the displacement is calculated, if they do not, ZINT and XINT are calculated along the next boundary line and tested again. This continues until adequate intersection points are found or all boundaries have been considered in which case an error message is output. The perpendicular displacement is returned to subroutine CALCD.
VRSPLT and VRSPLET

These subroutines set up the data in ZION and XION for transfer to the utility plotting package, Versatec. They also draw a schematic of the thruster and beam. Labels for the graph and its axes are also transferred and involve the word-size dependent variables. VRSPLT plots from the external file, PATHS, whereas VRSPLET plots from core memory. These routines are device dependent and may require considerable modification at other installations.

LNPLT and PLOTW

LNPLT sets up the data in ZION and XION for transfer to the utility plotting routine PLOTW which utilizes the line printer. It is also dependent on the type of printer available and may require considerable modification at other installations.
INPUT AND OUTPUT

For each simulation run, seven data cards (or card images) are read by READER. On the second card, if KEY=0, the reading sequence is interrupted and control returns to PLASIM, where normal termination ensues. The input data cards have the following formats and parameters which are defined in the following subsection and in APPENDIX C.

Card 1, FORMAT (see the first data card in BLOCK DATA)
Content: Description of the run, up to 80 characters.

Card 2, FORMAT (7I10)
Content: NUMION, NUMIT, KEY, IOLWRT, ICLPLT, NTOTST, ICLERR.

Card 3, FORMAT (6F10.5)
Content: RBOUND, RT, THLEN, BMCLR, UTIL.

Card 4, FORMAT (5E10.3)
Content: TELIN, TELOUT, TTHNEU, GEXSEC, UMSION.

Card 5, FORMAT (3F10.3)
Content: TIMEMU, XVELMU, ZVELMU.

Card 6, FORMAT (see the second data card in BLOCK DATA)
Content: Label for graphical output.

Card 7, FORMAT (see the second data card in BLOCK DATA)
Content: Axis labels for graphical output.

Due to the dimensions in the COMMON BLOCK, the values of the variables NUMION and NUMIT should not exceed 41 and 150, respectively. Also NUMIT should not be less than 11 and the total number of iterations performed should be less than 8888. The temperatures input on the fourth data card should be in units of eV, all other variables require SI units.

The printed output begins with three pages of identification including a schematic of the simulation region and a statement of the parameters defining the conditions of the run. (See WRIT(1).)
The next two segments consist of INTERIM STATUS, a report of the coordinates, velocity and density for each path after the initialization and first iteration, as prepared by INIT. (See WRIT(2).) A provision is made in subroutine CALC to output the counters, coordinates, velocity and density values for each iteration so that the progress of the simulation may be followed in detail. If the results of each iteration are not desired, the value of the input variable ICLWRT on data card 2 should be set to a value other than one. The value chosen determines how many iterations must occur before these results are output. WRIT(3) outputs the heading for the information output in subroutine CALC.

Graphical output consists of an outline drawing of the thruster and beam boundaries and the paths generated by the computation in the form of dots or lines. The paths are graphs of the coordinates contained in the arrays ZION, XION. These arrays are each a double-subscripted array in which the first subscript identifies the ion path number, and the second one identifies the iteration. The density array is also available for plotting, but provision for such has not been installed.

An appropriate relative scale in terms of the number of paths and the iteration step size must be established to properly model the upstream regions. Very small steps compared to the path spacing are inappropriate as are very large steps compared to the path spacing. The scaling ratio \( S = \frac{\Delta d}{v_B \Delta t} \) compares the path spacing to the axial step size. Appropriate values of this ratio, used herein, for the various thruster sizes are:

- 5 cm thruster: \( S = 2.9 \)
- 15 cm thruster: \( S = 3.9 \)
- 30 cm thruster: \( S = 3.6 \)

These values were obtained by taking \( \Delta d \) as the separation between the 14th and 15th paths, along the beam edge, \( v_B \) as the Bohm velocity and \( \Delta t \) as the time step for each configuration given in the SAMPLE INPUT section. It should be noted that the value of the scaling ratio, \( S = 2.9 \), for the
5 cm thruster was restricted by computer CPU time allocations. The modeling of the upstream region for a 5 cm thruster would be better if more stages were used because for 10 stages, \( S = 3.2 \) and for 11 stages \( S = 3.5 \).

Sample Input

A typical set of input parameters for program PLASIM, to simulate a 5 cm ion thruster using mercury (Hg) propellant, are:

<table>
<thead>
<tr>
<th>Data Card</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 cm thruster, non-uniform density distribution, basic configuration.</td>
</tr>
<tr>
<td>2</td>
<td>40 150 -2 20 3 9 1</td>
</tr>
<tr>
<td>3</td>
<td>.025 .60 .08 .50 .05 .71</td>
</tr>
<tr>
<td>4</td>
<td>7.0E+00 3.5E+00 4.7E-02 6.0E-19 3.34E-23</td>
</tr>
<tr>
<td>5</td>
<td>.75 1.0 0.0</td>
</tr>
<tr>
<td>6</td>
<td>Propagation of a charge-exchange plasma, 5 cm thruster</td>
</tr>
<tr>
<td>7</td>
<td>Distance along beam axis (meters) Radial distance (meters)</td>
</tr>
</tbody>
</table>

A typical set of input parameters for program PLASIM, to simulate a 15 cm thruster using mercury (Hg) propellant, are:

<table>
<thead>
<tr>
<th>Data Card</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15 cm thruster, non-uniform density distribution, basic configuration.</td>
</tr>
<tr>
<td>2</td>
<td>40 150 -2 20 3 4 1</td>
</tr>
<tr>
<td>3</td>
<td>.075 .60 .12 .30 .63 .85</td>
</tr>
<tr>
<td>4</td>
<td>5.0E+00 2.5E+00 4.7E-02 6.0E-19 3.34E-23</td>
</tr>
<tr>
<td>5</td>
<td>.75 1.0 0.0</td>
</tr>
<tr>
<td>6</td>
<td>Propagation of a charge-exchange plasma, 15 cm thruster</td>
</tr>
<tr>
<td>7</td>
<td>Distance along beam axis (meters) Radial distance (meters).</td>
</tr>
</tbody>
</table>
A typical set of input parameters for program PLASIM to simulate a 30 cm ion thruster using mercury (Hg) propellant are,

<table>
<thead>
<tr>
<th>Data Card</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30 cm thruster, non-uniform density distribution, basic configuration.</td>
</tr>
<tr>
<td>2</td>
<td>40 150 -2 20 3 2 1</td>
</tr>
<tr>
<td>3</td>
<td>.14 .60 .20 .40 2.0 0.9</td>
</tr>
<tr>
<td>4</td>
<td>0.350E+00 0.175E+00 4.7E-02 6.0E-19 3.34E-23</td>
</tr>
<tr>
<td>5</td>
<td>.75 1.0 0.0</td>
</tr>
<tr>
<td>6</td>
<td>Propagation of a charge-exchange plasma, 30 cm thruster.</td>
</tr>
<tr>
<td>7</td>
<td>Distance along beam axis (meters) Radial distance (meters).</td>
</tr>
</tbody>
</table>

The values given on data cards 3 and 4, in all of the above cases, correspond to the experimental values for those quantities.
VERIFICATION

Two studies, one analytic and one experimental, are used for the verification of the computer code presented herein. The analytic study was conducted as a support activity for the development of the subject computer code, and is reported in more detail in APPENDIX A.

Analytic Solution

The analytic solution is for the case of an infinitely long cylindrical ion beam with a uniform production of charge-exchange ions along the beam. The density and potential variations are restricted to be only radial so that an analytic solution could be obtained in a straightforward manner.

A computer solution was obtained using the uniform ion density option of the program (KEY=1) and is shown in Fig. 8. The dots occur every 60 iterations. The radial density and velocity from the analytic solution are compared to the radial density and velocity from the simulation, as a function of radial distance, in Figs. 9 and 10, respectively. The agreement between the analytic solution and the computer simulation for the velocity and the density is excellent.

Experimental Solution

Experimental surveys of the plasma density\textsuperscript{8,9} are shown in Figs. 11 and 12 along with comparisons to the computer code for 5 cm and 15 cm thrusters, respectively. The operating conditions used in the experimental studies were duplicated to obtain the isodensity contours generated by the computer simulation. Figure 13 shows the isodensity contours generated by the computer simulation for a 30 cm thruster. No experimental data is presently available for a 30 cm thruster. The ion...
Fig. 8. Simulation for uniform density of charge-exchange ion production. Twenty trajectories simulated in three stages, electron temperature 7.0 eV in ion beam and 3.5 eV in charge-exchange plasma for 5 cm thruster.
Fig. 9. Comparison of radial densities calculated using the computer code and analytic solution.
Fig. 10. Comparison of radial velocities calculated using the computer code and analytic solution.
Fig. 11. Simulated and experimental surveys of electron density for a 5 cm thruster.
Fig. 12. Simulated and experimental surveys of electron density for a 15 cm thruster.
Fig. 13. Simulated survey of electron density for a 30 cm thruster.

Electron density, m⁻³

Simulation

J_b = 2.00A
\( \eta_u = 0.90 \)

Radial distance, m

Axial distance, m

\( r_b \)

\( r_T \)
trajectory directions obtained using the simulation are in good agreement with the experimental measurements of Carruth and Brady.\textsuperscript{10}

Figures 14, 15 and 16 show typical ion trajectories generated by the simulation using the data in the SAMPLE INPUT section for a 5 cm, 15 cm and 30 cm thruster, respectively. The dots occur every 60 iterations.

Figure 17 is a simulation of the 15 cm thruster for use in comparing the computer code contained herein to that discussed in relation to Fig. 8 of the October 1980 report.\textsuperscript{3}

Limitations in Use

Major factors affecting the accuracy of the simulation obtained are the number of ion paths used, the number of iterations performed and the time interval used. When the number of ion paths simulated is increased, either more iterations must be used or the time interval size decreased through use of the variable TIMEMU. This is necessary to keep the spacing between the paths comparable to the distance the ion travels in one iteration, this is accomplished using the scaling ratio, S. If care is not taken in doing this, path crossings will sometimes occur, especially among the paths within one beam radius of the thruster. These path crossings result from plasma properties changing so rapidly that the error in a path location will exceed the local path spacing. The procedure used in carrying out the simulation depends on a "laminar" path structure, that is, no intersection of paths. The existence of any crossed paths, therefore, invalidates local calculations of densities and the other related parameters.

Other limitations are imposed by core storage allocations, external file space, and the CPU time available in a particular machine.
Fig. 14. Ion trajectories generated using data from the SAMPLE INPUT section for a 5 cm thruster.
Fig. 15, Ion trajectories generated using data from the SAMPLE INPUT section for a 15 cm thruster.
Fig. 16. Ion trajectories generated using data from the SAMPLE INPUT section for a 30 cm thruster.
Fig. 17. Ion trajectories generated by the computer simulation for a 15 cm thruster, one stage, 20 ion trajectories and 30 iterations. Other quantities same as in SAMPLE INPUT section.
The present code takes approximately 20 seconds to compile, 150 seconds to execute three stages with NUMION=40, NUMIT=150 and ICLWRT=1, occupies about 140Kbytes of central memory and writes approximately 125,000 characters on an external file for three stages, as above, on a Control Data Cyber 171 computer.

It should be noted that the code could be significantly simplified and shortened if it were translated to Fortran 77 (Fortran V).
REFERENCES


APPENDIX A

ANALYTIC SOLUTION

A theoretical benchmark is valuable for verification of the computer code developed to model the charge-exchange plasma propagation in the vicinity of an operating ion thruster. An analytic solution is developed herein for that purpose.

A cylindrical ion beam is assumed with a length very much greater than \( r_b \), the beam radius. The current density representing positive charge-exchange ion production in the beam is assumed uniform along the beam.

In the region exterior to the beam, three basic physical conditions are assumed to hold for the ion population and/or the plasma as a whole. The first is continuity of ion current represented by

\[
\nabla \cdot \mathbf{j} = 0
\]

where \( \mathbf{j} \) is the ion current density. The barometric equation is also used to relate plasma density to local potential \( V \)

\[
n = n_{o,\text{ref}} \exp\left[\frac{e(V-V_o)}{kT_e}\right]
\]

where \( V_o \) is the potential at the reference density \( n_{o,\text{ref}} \) and \( T_e \) is the electron temperature in the region exterior to the beam. Finally, energy conservation for singly charged ions is represented by

\[
|\nabla V| = \left(\frac{V_o^2}{m_i} - 2e(V-V_o)/m_i\right)^{1/2}
\]
where $\mathbf{v}$ is the ion velocity and $m_i$ is the ion mass. As a boundary condition at the beam edge, ions are assumed to have acquired the Bohm velocity

$$v_o = v_B = \left(\frac{kT_e}{m_i}\right)^{1/2}$$

(A4)

where $T_e$ is the electron temperature in the beam.

The ion current density is related to the streaming velocity by

$$\mathbf{j} = ne\mathbf{v}.$$  \hspace{1cm} (A5)

For the assumed symmetry, the velocity is radial and is

$$\mathbf{v} = v(r)\hat{r}.$$  \hspace{1cm} (A6)

In cylindrical coordinates, Eq. (A1) can be written with the substitution of Eq. (A5) as

$$\frac{1}{r} n(r) v(r) + n(r) \frac{\partial v(r)}{\partial r} \frac{\partial v(r)}{\partial r} + v(r) \frac{\partial n(r)}{\partial r} \frac{\partial v(r)}{\partial r} = 0$$

(A7)

which can be solved for $V(r)$ by eliminating $n(r)$ and $v(r)$ using Eqs. (A2) and (A3). The solution can then be written as

$$(1 - e(V-V_o)/kT_e)^{1/2} \exp(-e(V-V_o)/kT_e) = r/r_b.$$  \hspace{1cm} (A8)

The density and velocity can be calculated using Eq. (A8) along with Eq. (A3) or Eq. (A4).
The solution in terms of potential, density and velocity is displayed in dimensionless form in Figs. 1A, 2A, and 3A.
Fig. 1A. Potential as a function of radial distance.
Fig. 2A. Density as a function of radial distance.
Fig. 3A. Velocity as a function of radial distance.
APPENDIX B

BEAM CURRENT DENSITY PROFILES

Determination of the total rate of charge exchange ion production in the beam volume must take into account both the beam current density profile of the ion thruster and the spatial distribution of neutral propellant atoms in the path of the ion beam.

For theoretical calculations a simplified ion beam current density distribution is usually chosen and axial symmetry assumed. A closed-form solution exists only for the very simplest distribution which has been used for conservative estimates of charge exchange ion production. In this case the current density is given by a Dirac delta function,

\[ j_b = J_b \delta(r) , \]  

\[ (B1) \]

where \( J_b \) is the total beam current and \( \delta(r) \) is the two-dimensional Dirac delta function. This expression essentially places all of the beam current on the thruster axis where the neutral density follows a simple function, thus allowing a closed-form solution.

A current density profile that is more accurate for the larger, multipole thrusters is a uniform beam current density

\[ j_b = J_b / \pi r_b^2 , \]  

\[ (B2) \]

where \( r_b \) is the beam radius. Another profile that approximates the beam of some divergent field thrusters is a parabolic profile

\[ j_b = 2J_b (1 - r^2 / r_b^2) \pi r_b^2 , \]  

\[ (B3) \]
where \( r \) is the distance from the beam axis.

The expressions given above for beam current density are normalized such that

\[
J_b = \frac{2\pi}{\int_0^r \int_0^{2\pi} j_b r dr d\theta} .
\]  

(A4)

A Gaussian profile is also used by some workers where

\[
j_b = \left(\frac{J_b}{\pi r_b^2}\right)e^{-\left(\frac{r}{r_b}\right)^2},
\]  

subject to the normalization:

\[
J_b = \frac{2\pi}{\int_0^r \int_0^{2\pi} j_b r dr d\theta} .
\]  

(A5)

The uniform, parabolic and Gaussian profiles can also accommodate a defined rate of beam spreading as it propagates. The simulation described herein does not include effects from beam spreading.

Neutral gas leaving the accelerator system is taken to be in free molecular flow so that the effective neutral density at an arbitrary point such as that shown in Fig. 1B is proportional to the subtended solid angle of the ion optics as viewed from the point \((r, \theta, z)\).

Effective neutral densities were calculated numerically for an \( r-z \) matrix with a resolution of 0.1 \( r_p \). For the near field, where about half of the charge exchange takes place, the calculated densities are given in dimensionless form in Table 1B.

The calculated distribution of neutral propellant along with the beam current density profile allow a calculation of charge exchange ion production rate per unit length as a function of distance from the ion optics.
Fig. 1B. Coordinate system for thruster and ion beam geometry.
Table 1B. Density of Neutral Propellant Efflux, \( n(r, z)/n_{o, \text{ref}} \).

<table>
<thead>
<tr>
<th>( z/r_b )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r/r_b )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>4.52×10^{-1}</td>
<td>4.03×10^{-1}</td>
<td>3.57×10^{-1}</td>
<td>3.15×10^{-1}</td>
<td>2.77×10^{-1}</td>
<td>2.43×10^{-1}</td>
<td>2.14×10^{-1}</td>
<td>1.88×10^{-1}</td>
<td>1.66×10^{-1}</td>
<td>1.47×10^{-1}</td>
</tr>
<tr>
<td>0.1</td>
<td>4.51</td>
<td>4.02</td>
<td>3.56</td>
<td>3.14</td>
<td>2.76</td>
<td>2.42</td>
<td>2.13</td>
<td>1.87</td>
<td>1.65</td>
<td>1.46</td>
</tr>
<tr>
<td>0.2</td>
<td>4.49</td>
<td>4.00</td>
<td>3.53</td>
<td>3.11</td>
<td>2.73</td>
<td>2.39</td>
<td>2.10</td>
<td>1.85</td>
<td>1.63</td>
<td>1.44</td>
</tr>
<tr>
<td>0.3</td>
<td>4.47</td>
<td>3.96</td>
<td>3.48</td>
<td>3.05</td>
<td>2.67</td>
<td>2.34</td>
<td>2.05</td>
<td>1.80</td>
<td>1.59</td>
<td>1.41</td>
</tr>
<tr>
<td>0.4</td>
<td>4.43</td>
<td>3.90</td>
<td>3.40</td>
<td>2.97</td>
<td>2.59</td>
<td>2.26</td>
<td>1.98</td>
<td>1.74</td>
<td>1.54</td>
<td>1.36</td>
</tr>
<tr>
<td>0.5</td>
<td>4.38</td>
<td>3.81</td>
<td>3.29</td>
<td>2.85</td>
<td>2.47</td>
<td>2.15</td>
<td>1.89</td>
<td>1.66</td>
<td>1.47</td>
<td>1.31</td>
</tr>
<tr>
<td>0.6</td>
<td>4.31</td>
<td>3.68</td>
<td>3.14</td>
<td>2.69</td>
<td>2.33</td>
<td>2.02</td>
<td>1.77</td>
<td>1.57</td>
<td>1.39</td>
<td>1.24</td>
</tr>
<tr>
<td>0.7</td>
<td>4.18</td>
<td>3.47</td>
<td>2.92</td>
<td>2.48</td>
<td>2.14</td>
<td>1.87</td>
<td>1.64</td>
<td>1.45</td>
<td>1.30</td>
<td>1.16</td>
</tr>
<tr>
<td>0.8</td>
<td>3.94</td>
<td>3.15</td>
<td>2.61</td>
<td>2.21</td>
<td>1.92</td>
<td>1.69</td>
<td>1.49</td>
<td>1.33</td>
<td>1.19</td>
<td>1.08</td>
</tr>
<tr>
<td>0.9</td>
<td>3.41</td>
<td>2.64</td>
<td>2.20</td>
<td>1.90</td>
<td>1.67</td>
<td>1.49</td>
<td>1.33</td>
<td>1.20</td>
<td>1.09</td>
<td>9.86×10^{-2}</td>
</tr>
<tr>
<td>1.0</td>
<td>2.15</td>
<td>1.91</td>
<td>1.72</td>
<td>1.55</td>
<td>1.41</td>
<td>1.28</td>
<td>1.17</td>
<td>1.07</td>
<td>9.75×10^{-2}</td>
<td>8.94</td>
</tr>
</tbody>
</table>
As a first approximation, the charge exchange production rate can be calculated in closed form if the Dirac delta function is used for the ion beam profile. The neutral density along the beam axis is given by

\[ n_0(z) = \frac{n_{o,\text{ref}}}{2} \left( 1 - \frac{z}{z^2 + r_b^2} \right), \quad (B7) \]

where \( n_{o,\text{ref}} \) is defined as the density that would provide the correct loss rate of neutral propellant through an opening of the same area as the beam. The neutral loss rate is then

\[ \dot{N}_o = r_b n_{o,\text{ref}} \sqrt{\frac{\pi k T_o}{2 m_o}}, \quad (B8) \]

where \( T_o \) is the propellant temperature, \( m_o \) is the propellant mass, and \( k \) is Boltzmann's constant. The charge exchange ion production rate per unit length is thus

\[ \dot{N}_{CE}(z) = J_b Q_n(z)/e, \quad (B9) \]

for small total production rates. Integrating to obtain the total production rate gives

\[ N_{CE_T} = J_b Q_{n,\text{ref}} r_b/2e, \quad (B10) \]

As the assumed ion beam profile becomes less peaked, progressing from a Dirac delta function through Gaussian and parabolic functions to a uniform distribution, the production rate of charge exchange ions
will diminish as more of the beam passes through the peripheral regions of lower neutral propellant density. The two extremes in production rates as a function of distance thus use a delta function and a uniform function for the beam profile. The results for these two extremes are shown in Fig. 2B. Table 2B gives the calculated total charge exchange ion production rates for these two extremes, as well as the intermediate parabolic ion beam profile. The ion beam profile is clearly not a dominant parameter for charge exchange ion production.

The simulation developed to model the charge exchange plasma propagation can be modified for an arbitrary input for the production rate as a function of distance.
Fig. 2B. Charge-exchange ion production rates as a function of distance from the grids for extremes of possible beam current density distributions.
Table 2B. Charge-Exchange Ion Production Rates for Different Beam Current Density Profiles.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Production Rate ( (J_b Q_o, r_b / 2e) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j_b \delta(r) )</td>
<td>1.00</td>
</tr>
<tr>
<td>( 2J_b (1 - r^2 / r_b^2) / \pi r_b^2 )</td>
<td>0.97</td>
</tr>
<tr>
<td>( J_b / \pi r_b^2 )</td>
<td>0.94</td>
</tr>
</tbody>
</table>
APPENDIX C

GLOSSARY OF VARIABLES FOR PLASIM

This glossary contains definitions of the variables used in the driver program PLASIM, the subroutines BLOCK DATA, READER, INIT, CALC, CALCD, BOUND, and WRIT and the function subroutine DS, including all the variables in the COMMON blocks. The variables used in the plotting subroutines, VRSPLT, VRSPL, LNPLT, and PLOTW are not defined here.
BK: Boltzman's constant (J/K).

BMCUR: Beam current (AMPS).

CEXSEC: Charge exchange cross section (METERS SQUARED).

C: Time interval divided by mass of ion (TIME/UMSION).

COSPAR: Cosine of angle between line parallel to path and horizontal.

COSPER: Cosine of angle between line perpendicular to path and horizontal.

DELTAX: Difference between two x coordinates on present path.

DELTAZ: Difference between two z coordinates on present path.

DETLX: Difference between two x coordinates on left path.

DETLZ: Difference between two z coordinates on left path.

DELTRX: Difference between two x coordinates on right path.

DELTRZ: Difference between two x coordinates on right path.

DELTRZ: Difference between two z coordinates on right path.

DN(I,N): Density on path I at iteration N.

DNI(I): Initial density between paths (I-1) and (I).

DNL: Density to left side of current path.

DNB: Constant used in calculation of the densities, combination of quantities including; BMCUR, CEXSEC, RB, NUMION, UTIL, Q and VELNEU.

DNR: Density to right side of present path.

DS: Perpendicular displacement from present path to boundary.

DSPL: Displacement to left hand path.

DSPLR: Displacement to right hand path.

DUMMY, DUMMYL, DUMMYR: Dummy variables used in the calculation of the intercepts. Contain intermediate results.

DSPLIP: Displacement of ion along path.

F: Total force acting on ion.

FPAR: Force acting parallel to the path.

FPARX: Component of FPAR acting in x direction.
FPARZ: Component of FPAR acting in z direction.
FPER: Force acting perpendicular to the path.
FPERX: Component of FPER acting in x direction.
FPERZ: Component of FPER acting in z direction.
FX: Component of F acting in x direction (FPERX + FPARX).
FZ: Component of F acting in z direction (FPERZ + FPARZ).
I: Allows do loop index, II, to be passed through COMMON, path index.
ICLERR: Used to determine if code generated error messages, for non-fatal errors, are written out.
  = 0, Write messages.
  = 1, Do not write messages.
ICLPLT: Used to determine which (if any) plotting routine is used;
  = 1, Use subroutine VRSPLT.
  = 2, Use subroutine LNPLT.
  = 3, Use both VRSPLT and LNPLT.
  = anything else, No plots.
ICLWRT: Frequency with which results of CALC are output. Write statement in CALC called after every ICLWRT number of iterations.
IDEF: Used to determine when the x and z coordinates of an ion trajectory exit point are defined (every other time).
IDXNEW: New index for right or left-most path.
IFLAG1: Used to determine if intersections were found to both the left and the right without using linear extrapolation,
  = 0, Linear extrapolation not used.
  = 1, Linear extrapolation used on left.
  = 2, Linear extrapolation used on right.
  = 3, Linear extrapolation used in both cases.
IFLAG2: Error flag, tells whether or not the neighboring paths are active,
  = 0, No error, path active.
  > 0, Error exists, value references program statement where error condition originated.
  = -1, Path to left is not active.
  = -2, Path to right is not active.
  = -3, Neither path (right or left) is active.
(Negative values do not indicate an error condition.)
IFLAG3: Used to determine when there is a boundary on the right or left (or both) of the current path.
  = 1, No boundary intersected by the perpendicular to the current path on either side.
  = 2, Boundary intersected on left.
  = 3, Boundary intersected on right.
  = 4, Boundary intersected on both the left and right.

IFLAG4: Trajectory one renormalization flag,
  = 0, Continue iterating as usual.
  = 1, Recalculate position, velocity and density of ion on path 1.

IFVAR: Dummy variable used as an argument in an if statement.

IF1: A format for alpha-numeric data I/O.

IF2: A format for alpha-numeric data I/O.

II: Do loop index.

IL: Index (I) of left-most active path.

IN: Device code for input unit, used in read statements.

INFO(K): Array storing information describing run.

INITDO: Initial do loop index for DO 80 N which calls CALC.

INITIT: Initial iteration index (stage dependent).

IOUT: Device code for output unit, used in write statements.

IPAG: Page number for output.

IPATHS: Device code for output to external files.

IR: Index (I) of right-most active path.

ISAT: Used as path status holder for reading IPATHS and in defining right boundary for high resolution upstream run.

ISTAT(I): Status of path I,
  = 0, Path is active.
  = 1 to NMAX, Path is not active, value is iteration number where boundary was intersected.
  = 8888, Path is not active, error condition.

ITITL(K): Array storing graph title and axis labels.

ITTOTN: Total number of iterations to be performed.
IW: Number of words required to generate 80 characters,
= 8 For computers with a word size of 10.
= 14 For computers with a word size of 6.
(Used in COMMON /IO/ and Routines BLOCK DATA, READER, WRIT,
VRSPLT and VRSP).

J: Used to determine when WRITE-435 is executed (CALC).

KE: Calling parameter for subroutine WRIT,
= 1, Output heading, initial information and data.
= 2, Output interim status of main variables.
= 3, Finish of a pass, output results, start of new pass.
= 4, Create file of path coordinates.
= 5, Create file of position - density triplets.

KEY: Used to determine type of run,
= 0, Finish plots and terminate (2 blank cards will do).
> 0, High resolution upstream pass, right most boundary is
defined using KEY'TH path of file IPATHS (used to
define ZBOUND).
= -1, First pass, uniform density distribution.
< -1, Regular run; first pass, normal non-uniform density
distribution.

LAB: Label containing computer code name, used for output.

LASTDO: Dummy variable denoting last value of do loop index.

LASTIT: Final (last) iteration index (path status dependent).

LR: Determines which side is being considered (DS),
= 1, Looking to the left.
= 2, Looking to the right.

N: (Working) number of iterations on present path, I, total
number of iterations for the present stage.

NIP(I): Total number of completed iterations on path I.

NITP: Iteration pass number, used for output.

NL: (Working) number of iterations on left hand path.

NLML: NL minus one (NL-1), used for indexing left-hand path.

NMAX: NUMPRE divided by four (NUMPRE / 4) (see WRIT(4)).

NR: (Working) number of iterations on right hand path.

NRML: LR minus one (NR-1), used for indexing right-hand path.

NSTAG: Allows do loop index NSTAGE to be passed through COMMON.

NSTAGE: Program stage number.
NSTGMU: Used to add 10 to N after first stage, (N stage multiplier),
NSTAG = 1; NSTGMU = 0
NSTAG > 1; NSTGMU = 1

NTOTST: Maximum number of stages to be run.

NUMION: Number of ion paths.

NUMIT: Maximum number of iterations to be performed on any one
       path during any one stage.

NUMPRE: Number of ion paths (NUMION) from run which created file
        IPATHS (see WRIT(4)).

NUMSTRI: Number of triplets output to be external file.

NUM1: Number of ion paths plus one (NUMION + 1).

NUM2: Two times the number of ion paths plus one (2*NUMION + 1).

PI: Geometrically defined constant, 3.14159265...

PIOV2: PI over (divided by) 2.

Q: Elementary unit of charge (C).

RB: Radius of the beam (METERS).

RBOUND: Radial boundary in positive x direction (METERS).

RB2: Beam radius squared (RB ** 2).

RB95N: 95 percent of the beam radius divided by 2 times the number
       of ion paths (RB * .95 / (2 * NUMION)).

RT: Radius of the thruster (METERS).

SINOV2: SINPER over (divided by) 2.

SINFAR: Sine of angle between line parallel to path and horizontal.

SINPER: Sine of angle between line perpendicular to path and horizontal.

SLOPEL: Slope of path on left between two "working" points.

SLOPEP: Slope of line perpendicular to current path at endpoint.

SLOPER: Slope of path on right between two "working" points.

SPACER: Initial distance between paths in uniform distribution.

TELIN: Temperature of electrons in the ion beam (EV).
TELOUT: Temperature of electrons outside the ion beam (EV).

THETAP: Angle between line with slope SLOPEP and horizontal.

THRLEN: Thruster length (METERS).

TIME: Time interval, defines iteration step size.

TIMEMU: Time multiplier, used to define the time step in terms of multiple of RBOUND / (VELBOH * NUMIT).

TTHNEU: Temperature of thermal neutrals in chamber (EV).

UMSION: Mass of ions (propellant) (KILOGRAMS).

UTIL: Utilization factor of propellant (part of propellant turned into ions).

VCURR: Plasma potential at present point on path I.

VELBET: Present total velocity between path under consideration and neighboring path.

VELBOH: Bohm velocity.

VELBTL: Present total velocity between path under consideration and path on left.

VELBTR: Present total velocity between path under consideration and path on right.

VELNEU: Thermal velocity of the neutrals in the chamber.

VELTOT: Present total velocity of ion.

VELT2: Present total velocity of ion along path 2.

VELTX(I): Present total velocity component in x direction.

VELTZ(I): Present total velocity component in z direction.

VELX: Velocity contribution for this iteration along x direction.

VELZ: Velocity contribution for this iteration along z direction.

VL: Plasma potential on left side of present path.

VPREV: Plasma potential at previous point on path I.

VR: Plasma potential on right side of present path.

X: X coordinate of point to be tested.

XINT: X intersection point on boundary line.
XINTL: X intersection, to left, of lines SLOPEP and SLOPEL.

XINTR: X intersection, to right, of lines SLOPEP and SLOPER.

XION(I,1): X coordinate of ion trajectory exit point I.

XION(I,2): First x coordinate of ion after leaving ion beam along trajectory I.

XION(I,N): Present x position of ion.

XION(I,N+1): Newly calculated x position of ion.

XPOSL: X-position (coordinate) half way along the perpendicular displacement (DSPLL) to the neighboring path on the left.

XPOSr: X-position (coordinate) half way along the perpendicular displacement (DSPLR) to the neighboring path on the right.

XVELMU: X velocity multiplier, used to define initial x velocity in terms of some multiple of the Bohm velocity.

Z: Z coordinate of point to be tested.

ZBOUND: Z boundary to right of thruster.

ZCURR: Current z increment, used to get ion exit points.

ZINT: Z intersection point on boundary line.

ZINTL: Z intersection, to left, of lines SLOPEP and SLOPEL.

ZINTR: Z intersection, to right, of lines SLOPEP and SLOPER.

ZION(I,1): Z coordinate of ion trajectory exit point I.

ZION(I,2): First z coordinate of ion after leaving ion beam along trajectory I.

ZION(I,N): Present z position of ion.

ZION(I,N+1): Newly calculated z position of ion.

ZMAX: Maximum z value on path KEY of IPATHS, defines ZBOUND for high resolution upstream run.

ZPREV: Previous z increment, used to get ion exit points.

ZVELMU: Z velocity multiplier, used to define initial z velocity in terms of some multiple of the Bohm velocity.
APPENDIX D

COMPUTER CODE LISTING FOR PLASIM
PROGRAM PLASM (INPUT, OUTPUT, TAPE5 = INPUT, TAPE6 = OUTPUT, NPARM, PATHS, TAPE7 = PATHS, DEBUG = OUTPUT)

************ DRIVER ROUTINE ************

PROGRAM DESCRIPTION; PROGRAMMER = WILLIAM OEININGER, 9 - 30 - PL
PLACE NOTES ON REVISIONS HERE; INCLUDE DATE, INITIALS, LOCATION
AND CHANGE MADE *** PLEASE ***

THIS IS THE DPFVFP ROUTINE FOR THE SIMULATION OF A CHARGE -
EXCHANGE PLASMA SURROUNDING A BROAD REA H ION THRUSTER. THIS ROUTINE
DEFINES THE PROGRAM STRUCTURE AND STAGING BY CALLING VARIOUS
SUBROUTINES. READER: THE FIRST CALL IS TO SUBROUTINE READER WHICH
READS IN THE RUN SPECIFICATIONS AND INPOPATION. INIT: THE NEXT
CALL IS TO SUBROUTINE INIT WHICH INITIALIZES VARIOUS PARAMETERS AND
CONSTANTS, DEFINES THE ION EXIT POINTS ALONG THE REA H EDGE AND
PERFORMS THE FIRST ITERATION. CALC: THE NEXT CALL IS TO
SUBROUTINE CALC WHICH COMPUTES THE ION POSITIONS ALONG THE
TRAJECTORIES. CALC IS CALLED NUMIT TIMES. THIS COMPLETES THE
FIRST STAGE. (NUMIT IS THE MAXIMUM NUMBER OF ITERATIONS TO BE
PERFORMED ON ANY GIVEN PATH DURING ANY PARTICULAR STAGE).
INFORMATION FOR THE FIRST (NUMIT - 10) ITERATIONS IS WRITTEN TO AN
EXTERNAL FILE, PATHS. CORE MEMORY THEN REINITIALIZED BY
TRANSFERRING THE RESULTS OF THE FINAL TEN ITERATIONS TO THE CORE SPACE
FOR THE INITIAL TEN. THE RESULTS OF THESE TEN ITERATIONS ACT AS A
BASE FOR ANOTHER SET OF (NUMIT - 10) ITERATIONS. CALC IS CALLED
ANOTHER (NUMIT - 10) TIMES TO FILL CORE STORAGE WITH ANOTHER SET OF
PATH COORDINATES WHICH COMPLETES THE SECOND STAGE. THEN CORE IS
REINITIALIZED AGAIN AND ANOTHER STAGE IS RUN. THIS IS DONE UNTIL
THE DESIRED TOTAL NUMBER OF ITERATIONS IS PERFORMED, OR UNTIL
ALL PATHS INTERSECT BOUNDARIES. FINALLY, THE PLOT INDICATOR,
'ICLPLT', IS CHECKED TO DETERMINE WHICH PLOTTING ROUTINE TO USE TO
OUTPUT A PLOT OF THE PATH COORDINATES. IF ANY.

*** NOTE ***
ALWAYS PUT TWO BLANK CARDS AT THE END OF THE DATA DECK TO
SIGNAL A STOP. THE WORD SIZE DEPENDENT VARIABLES ARE
IF: IF0, IFI, INITL AND IN.
SEE BLOCK DATA FOR INSTRUCTIONS ON WORD SIZE
DEPENDENT VARIABLES AND CODE GENERATED ERROR MESSAGES.

********** VARIABLE DICTIONARY **********

DN(K,M) : DENSITY ON PATH K, AT ITERATION M.
ICLPLT : USED TO DETERMINE WHICH (IF ANY) PLOTTING ROUTINE IS
USPD
  = 1 , USE SUBROUTINE VRSPLT.
  = 2 , USE SUBROUTINE LNPLT.
  = 3 , USE BOTH VRSPLT AND LNPLT.
  = ANYTHING ELSE , NO PLOTS.
INIT000 : INITIAL DO LOOP INDEX FOR 'DO 80 N1 WHICH CALLS 'CALC'.
ISTAT(K) : STATUS OF PATH K,
  = 0 , PATH IS ACTIVE.
  = 1 TO NMAX , PATH IS NOT ACTIVE, VALUE IS ITERATION
      NUMBER WHERE BOUNDRY WAS CROSSED.
  = RARR , PATH IS NOT ACTIVE, ERROR CONDITION.
KEY : USED TO DETERMINE TYPE OF RUN,
  = 0 , FINISH PLOTS AND TERMINATE.

ORIGINAL PAGE IS,
OF POOR QUALITY
C > 0 , HIGH RESOLUTION UPSTREAM PASS, RIGHT MOST
BOUNDARY IS DEFINED USING "KEYTH" PATH OF FILE
PATHS (USED TO DEFINE ?BOUND).
-1, FIRST PASS, UNIFORM DISTRIBUTION.
< -1, REGULAR RUN, FIRST PASS, NON-UNIFORM DISTRIBUTION.

NSTAG : ALLWS DO LOOP INDEX 'NSTAGE' TO BE PASSED THROUGH
COMMON.

NSTAGE : PROGRAM STAGE NUMBER.
NTST : MAXIMUM NUMBER OF STAGES TO BE RUN.
NUMION : NUMBER OF ION PATHS TO BE SIMULATED.
NUMIT : MAXIMUM NUMBER OF ITERATIONS TO BE PERFORMED ON ANY ONE
PATH DURING ANY ONE STAGE.

XION(K,M) : X POSITION OF ION ON PATH K AT ITERATION M.
ZION(K,M) : Z POSITION OF ION ON PATH K AT ITERATION M.

*** END OF PROGRAM DESCRIPTION ***

PROGRAM DECLARATION STATEMENTS.
BLANK COMMON FOR LARGE ARRAYS, IO = INPUT-OUTPUT, PARAM = PARAMETERS.

COMMON ZION(41,151),XION(41,151),VELTZ(151),VFLTX(151),
1 NIP(41),DN(41,151),DNT(42),ISTAT(41)
2 COMMON /IO / INIOUT,INFO(14),KEY,ICLPLT,ICLWT,ITIL(26),
3 IPATHS,IV,IF1(7),IF2(4),ICLPR
4 COMMON /PAPAP / NDUMION,NUMIT,RA,RBOUND,RT,TELIN,WMCUR,UTIL,
5 DNUM,CEXSFC,TTHEMU,TIME,TIMEMU,VELMU,VELMU,NSTAG,
6 NSTGHN,NSTOTS,PIDV2

INPUT PARAMETERS, SPECIFICATIONS, INFORMATION. INITIALIZE VARIABLES.
CHECK KEY TO SIGNAL STOP.

1 CALL READER
INI0D = 2
NSTAG = 1

******************************************************************************************

PLOTTING ROUTINE VRSPLT PLOTS FROM CORE MEMORY

IF (KEY .EQ. 0) CALL VRSPLT
******************************************************************************************

PERFORM INITIALIZATION.

CALL INIT

BEGIN STAGING. THE DO LOOP INDEX 'NSTAGE' GIVES THE STAGE NUMBER.

DO 100 NSTAGE = 1, NTST
NSTAG = NSTAGE
IF (NSTAG .EQ. 1) GO TO 70
INI0D = 10

REINITIALIZATION PROCEDURE. THE RESULTS FOR THE FINAL TEN ITERATIONS
ARE TRANSFERRED TO THE CORE AREA FOR THE FIRST TEN ITERATIONS. THESE
TEN ITERATIONS ARE USED AS A BASE FOR ANOTHER (NUMIT = 10) ITERATIONS.
THE TOTAL ITERATION PER PATH COUNTER IS REDUCED BY ONE TO ENSURE

ORIGINAL PAGE IS OF POOR QUALITY
115 C THAT EACH STAGE DOES THE PROPER NUMBER OF ITERATIONS.
C
DO 50 M = 1, 10
DO 40 K = 1, NUMION
   IF (ISTAT(K) .NE. 0) GO TO 40
   MN = N + 141
   XION(K,M) = XION(K,MN)
   ZION(K,M) = ZION(K,MN)
   IF (M .GE. 10) GO TO 30
   DN(K,M) = DN(K,MN)
50  CONTINUE
40  CONTINUE
30  CONTINUE
   DO 60 I = 1, NUMION
      NIP(I) = NIP(I) - 1
60  CONTINUE
C ITERATE "NUMIT" TIMES TO COMPLETE THE FIRST STAGE. ITERATE
C "NUMIT - 10" TIMES TO COMPLETE THE SUCCESSIVE STAGES. THIS
C SECTION CALCULATES THE ION POSITIONS ALONG THE PATHS.
C
70  CONTINUE
   DO 80 NN = INITND, NUMIT
      N = NN
      CALL WRTT(3)
      CALL CALC
80  CONTINUE
C WRITE INFORMATION FOR FIRST (NUMIT - 10) ITERATIONS IN CORE
C TO EXTERNAL FILE. BEGIN NEW STAGE IF DESIRED.
C
C ****************************************
C VRSPLT AND LNPLT PLOT FROM CORE MEMORY
C CALL VRSPLT
C IF (ICLPLT .GT. 1) CALL LNPLT
C ****************************************
C CALL WRIT(5)
C 100 CONTINUE
C DETERMINE PLOTTING ROUTINE TO BE USED, IF ANY. AFTER PLOTTING CHECK
C FOR MORE INPUT (READ MORE DATA IF ANY).
C IF (ICLPLT .EQ. 1) CALL VRSPLT
C IF (ICLPLT .EQ. 3) CALL VRSPLT
160 GO TO 1
END

CARD NR. SEVERITY DETAILS DIAGNOSIS OF PROBLEM
1 ANSI THIS STATEMENT TYPE IS NON-ANSI.
**PROGRAM DESCRIPTION:** PROGRAMMER = WILLIAM DEININGER.
REVISED: (INCLUDE DATE, INITIALS AND DESCRIBE CHANGE ** PLEASE **)

THIS PROGRAM LOADS DATA INTO LABELED COMMON STORAGE AT COMPILE TIME THROUGH THE DATA STATEMENTS.

**VARIABLE DICTIONARY**

- **BLK** : BOLTZMAN'S CONSTANT (J/K).
- **IF1** : A FORMAT FOR ALPHANUMERIC DATA I/O.
- **IF2** : A FORMAT FOR ALPHANUMERIC DATA I/O.
- **IN** : DEVICE CODE FOR INPUT UNIT, USED IN READ STATEMENTS.
- **IOUT** : DEVICE CODE FOR OUTPUT UNIT, USED IN WRITE STATEMENTS.
- **IPATHS** : DEVICE CODE FOR OUTPUT TO EXTERNAL FILES.
- **IW** : NUMBER OF WORDS REQUIRED TO GENERATE 80 CHARACTERS,
  - 8 FOR COMPUTERS WITH A WORD SIZE OF 10
  - 14 FOR COMPUTERS WITH A WORD SIZE OF 16
  (USED IN "COMMON /I0/" AND ROUTINES "BLOCK DATA", "READER", "WRIT", "VRSPLT" AND "VRSPL")
- **PI** : GEOMETRICALLY DEFINED CONSTANT, 3.14159265...
- **Q** : ELEMENTARY UNIT OF CHARGE (C).

**END OF PROGRAM DESCRIPITION AND DICTIONARY**

**PROGRAM DECLARATION STATEMENTS.**

BLANK COMMON FOR LARGE ARRAYS, IO = INPUT-OUTPUT, PARAM = PARAMETERS.

```c
COMMON 2ION(41,151),YION(41,151),VELTZ(151),VELTX(151),
       NIP(41),ONI(41,151),ONI(42),ISTAT(41)
COMMON /IO/ IN,IOUT,INFO(14),KEY,ICLPLT,ICLWRT,ITITL(2R),
       IPATHS,IW,IF1(2),IF2(4),ICLErr
COMMON /PARAM/ N,NUMION,NUMIT,RB,RT,BROUND,RT,TELOUT,MCCLUR,UTIL,
       TELIN,TRLFN,UNSTON,VELBH,ZBOUND,IL,IR,PI,PK,PK85N,
       DNB,CEXSCF,THTHEU,TIME,TIMELU,XVELMU,ZVELMU,NSTAG,
       NSTGMU,NSTST,PIOV
```

**NOTE:** WORD SIZE DEPENDENT VARIABLES;

- **IF1**, **IF2**, **IW**, **INFO**, **ITITL**
- **IW** IS THE NUMBER OF WORDS REQUIRED TO GENERATE 80 CHARACTERS.
- TO CONVERT TO MACHINE USING DIFFERENT WORD SIZE, MODIFY ONLY
- **IW**, **IF1** AND **IF2** IN FIRST TWO DATA STATEMENTS BELOW.

**THIRD DATA STATEMENT FOR I/O BUFFERS, FOURTH DATA STATEMENT FOR VALUES OF CONSTANTS.**

```c
DATA IW,IF1(1),IF1(2),IF2(1),IF2(4),IW)
DATA IF2(1),IF2(2),IF2(3),IF2(4)
DATA IN,ION,IPATHS,IW,6H(6A10),1H
DATA BK,G,P1,1.3E06F-23,1.602E-19,3.141593/
```

**INFORMATION ON PROGRAM GENERATED ERRORS MESSAGES**

**WHEN ERROR CALLED, FIRST LINE OF ERROR OUTPUT WILL BE OF THE FORM,**

```c
****** ERROR NNN ******
```

ORIGINAL PAGE 18
OF POOR QUALITY
WHERE 'NNN' IS ONE OF THE FOLLOWING INTEGERS,

C 207  SEE SUBROUTINE WRIT
C 410  SEE FUNCTION SUBROUTINE DS
C 412  SEE FUNCTION SUBROUTINE DS
C 521  SEE SUBROUTINE CALCD
C 522  SEE SUBROUTINE CALCD
C 523  SEE SUBROUTINE CALCD
C 524  SEE SUBROUTINE CALCD
C 525  SEE SUBROUTINE CALCD
C 526  SEE SUBROUTINE CALCD
C 527  SEE SUBROUTINE CALCD
C 528  SEE SUBROUTINE CALCD
C 529  SEE SUBROUTINE CALCD
C 530  SEE SUBROUTINE CALCD
C 610  SEE SUBROUTINE BOUND
C 612  SEE SUBROUTINE ROUND

AND THE REFERENCED SUBROUTINE IS WHERE THE ERROR IS CALLED FROM.

FATAL' ERRORS ARE GENERALLY FATAL TO PARTICULAR PATH ONLY.

END
SUBROUTINE READER

*************** INPUT (READ) ROUTINE ***************

PROGRAM DESCRIPTION: PROGRAMMER = WILLIAM DEININGER, 7 - 9 - 81.
REVISIONS: (INCLUDE DATE, INITIALS AND DESCRIBE CHANGE ** PLEASE **) 

THIS SUBROUTINE READS THE RUN DESCRIPTION AND SPECIFICATIONS, 
BOUNDARY SPECIFICATIONS, PROPPELLANT AND PLASMA SPECIFICATIONS, GRAPH 
LABELS AND READS FILE IPATHS (IF NECESSARY) TO SUPPLY THE NECESSARY 
INFORMATION FOR HIGH RESOLUTION UPSTREAM RUNS.

******** VARIABLE DICTIONARY ********

10 ACMUR : BEAM CURRENT (AMPS).
 CEXSFC : CHARF EXCHANGE CROSS SECTION (METERS SQUARED).
 ICLPLT : USED TO DETERMINE WHICH (IF ANY) PLOTTING ROUTINE IS 
          USED:  = 1 , USE SUBROUTINE VRSPLT.
          = 2 , USE SUBROUTINE LNPLT.
          = 3 , USE BOTH VRSPLT AND LNPLT.

 ICLWPT : FREQUENCY WITH WHICH RESULTS OF CALC ARE OUTPUT. WRITE 
          STATEMENT IN CALC CALLED AFTER EVERY "ICLWRT" NUMBER 
          OF ITERATIONS.

20 ICLERR : USED TO DETERMINE IF CODE GENERATED ERROR MESSAGES, FOR 
            NON-FATAL ERRORS, ARE WRITTEN OUT.
            = 0, WRITE MESSAGES.
            = 1, DO NOT WRITE MESSAGES.

 INFO(K) : ARRAY STOPING INFORMATION DESCRIBING RUN.

 ISAT : USED AS PATH STATUS HOLDER FOR READING IPATHS AND IN 
        DEFINING RIGHT BOUNDARY FOR HIGH RESOLUTION UPSTREAM RUN.

 ISTAT(I) : STATUS OF PATH I,
            = 0, PATH IS ACTIVE.
            = 1 TO NMAX, PATH IS NOT ACTIVE, VALUE IS ITERATION 
            NUMBER WHERE BOUNDARY WAS CROSSED.
            = ANYTHING ELSE, PATH IS NOT ACTIVE, ERROR CONDITION.

 ITITL(K) : ARRAY STOPING GRAPH TITLE AND AXIS LABLES.

 KEY : USED TO DETERMINE TYPE OF RUN,
       = 0, FINISH PLOTS AND TERMINATE (2 BLANK CARDS WILL DO).
       > 0, HIGH RESOLUTION UPSTREAM PASS, RIGHT MOST BOUNDARY 
           IS DEFINED USING KEY'TH PATH OF FILE IPATHS (USED 
           TO DEFINE ZBOUND). (SEE NOTE BELOW).
       = -1, FIRST PASS, UNIFORM DISTRIBUTION.
       < -1, REGULAR RUN, FIRST PASS, NORMAL NON-UNIFORM 
       DISTRIBUTION.

40 NIP(I) : TOTAL NUMBER OF COMPLETED ITERATIONS ON PATH I.

 NMAX : NUMPRE DIVIDED BY FOUR (NUMPRE / 4) (SEE WRIT(4)).

 NMTOTST : TOTAL NUMBER OF STAGES TO BE RUN.

 NUM1 : NUMBER OF ION PATHS PLUS ONE (NUMION + 1).

 NUMION : NUMBER OF ION PATHS.

 NUMIT : MAXIMUM NUMBER OF ITERATIONS TO BE PREFORMED ON ANY 
         ION PATH DURING ANY ONE STAGE.

 NUMPRF : NUMBER OF ION PATHS (NUMION) FROM RUN WHICH CREATED FILE 
         IPATHS (SEE WRIT(4)).

50 RA : RADIUS OF THE BEAM (METERS).

 RBOUND : RADIAL BOUNDARY IN POSITIVE X DIRECTION (METERS).

 RT : RADIUS OF THE THRUSTER (METERS).

ORIGINAL PAGE IS 
OF POOR QUALITY
**TELIN**  TEMPERATURE OF ELECTRONS IN THE ION BEAM (EV).

**TELOUT**  TEMPERATURE OF ELECTRONS OUTSIDE THE ION BEAM (EV).

**THRLN**  THRUSTER LENGTH (METERS).

**TIMEU**  TIME MULTIPLIER, USED TO DEFINE THE TIME STEP IN TERMS OF SOME MULTIPLE OF RBOUND / (VELBOH * NUMIT).

**TTHNEU**  TEMPERATURE OF THERMAL NEUTRALS IN CHAMBER (EV).

**UMSIN**  MASS OF IONS (PROPPELLANT) (KILOGRAMS).

**UTIL**  UTILIZATION FACTOR OF PROPPELLANT (PART OF PROPPELLANT TURNED INTO IONS).

**XVELMU**  X VELOCITY MULTIPLIER, USED TO DEFINE INITIAL VELOCITY IN TERMS OF SOME MULTIPLE OF THE BOHM VELOCITY.

**ZVELMU**  Z VELOCITY MULTIPLIER, USED TO DEFINE INITIAL VELOCITY IN TERMS OF SOME MULTIPLE OF THE BOHM VELOCITY.

*** END OF PROGRAM DESCRIPTION AND DICTIONARY ***

PROGRAM DECLARATION STATEMENTS.

BLANK COMMON FOR LARGE ARRAYS, IO = INPUT-OUTPUT, PARAM = PARAMETERS.

```
COMMON ION(151,151),XIN(151),VELTZ(151),VELTX(151),
  1 NIT(152),DPN(151),DNI(42),ISTAT(41)
COMMON / IO / IN,IODUT,INFO(14),KEY,ICLPLT,ICLWR,ITITL(28),
  2 IPATHS,IV,IFI(2),IF2(4),ICLRR
COMMON / PARAM / N,NUMION,NUMIT,RB,RBOUND,RT,TELOUT,BMCUR,UTIL,
  3 TELIN,THRLN,IMPSON,VELBOH,ZBOUND,IL,IR,PI,BK,O,BK95N,
  4 DNOB,CEXSEC,TTHNEU,TIME,TIMEU,XVELMU,ZVELMU,NSTAG,
  5 NSTGMU,NTOTST,PI,TV2
```

READ IN 80 COLUMNS OF INFORMATION DESCRIBING RUN. FIRST CHARACTER SHOULD BE A BLANK FOR PRINTER LINE CONTROL.

READ (IN, IFI) (INFO(K), K = 1, IV)

READ IN RUN SPECIFICATIONS AND PARAMETERS.

```
READ (IN, 12) NUMION, NUMIT, KEY, ICLWR, ICLPLT, NTOTST, ICLRR
```

IF (KEY .EQ. 0) RETURN
NUMI = NUMION + 1

READ IN BOUNDARY SPECIFICATIONS.

```
READ (IN, 13) PA, RBOUND, RT, THRLN, BMCUR, UTIL
```

READ IN PROPPELLANT AND PLASMA SPECIFICATIONS. TEMPERATURES (FIRST THREE VARIABLES LISTED) SHOULD BE INPUT IN ELECTRON-VOLTS. THEY WILL BE CONVERTED TO THE DESIRED UNITS FOR CALCULATION BY THE CODE.

```
READ (IN, 14) TFLIN, TELOUT, TTHNEU, CEXSEC, UMSION
```

```
TFLIN = TFLIN * 0
TELOUT = TELOUT * 0
TTHNEU = TTHNEU * 0
```

READ IN THE TIME MULTIPLIER AND THE X AND Z VELOCITY MULTIPLIERS (SEE ABOVE DEFINITIONS).

ORIGINAL PAGE 16
OF POOR QUALITY
READ (IN, 15) TMFNU, XVELMU, ZVELMU
15 FORMAT (3D10.3)

READ TWO CARDS FOR GRAPH, 80 COLUMNS FOR TITLE AND 40 COLUMNS EACH FOR AXIS LABELS. (WPN SIZE DEPENDENT AREA)

IW2 = IW * 2
READ (IN, IF?) (1ITTL(K), K = 1, IW2)

TEST TO SEE IF THIS IS A HIGH RESOLUTION UPSTREAM RUN. IF NOT, RETURN TO DRIVER; IF SO, READ FILE IPATHS AND SET UP BOUNDARY.

IF (KEY) 99, 99, 30

REWIND FILE IPATHS SO IT CAN BE READ FOR ANOTHER PASS. THEN READ THE NUMBER OF ION PATHS IN IPATHS TO BE READ, THE TOTAL NUMBER OF ION PATHS MAKING UP THE FILE IPATHS (NUMPRE = NUMION FROM BEFORE), THE STATUS OF THE PATHS AND THE ARRAY OF PATH COORDINATES.

***** NOTE *****

DUE TO THE DEFINITION OF NMAX (SEE WRIT(4)), KEY MUST BE LESS THAN OR EQUAL TO NMAX FOR A HIGH RESOLUTION UPSTREAM RUN.

ASSIGNMENT OF DEVICE CODE SHOULD BE MODIFIED SO AS NOT TO INTERFER WITH STAGING AND WRIT(5). (SEE WRIT(4))

REWIND IPATHS
READ (IPATHS) NMAX, NUMPRE
READ (IPATHS) (ISTAT(I), I = 1, NMAX)
DO 50 I = 1, NMAX
   ISAT = ISTAT(I)
   READ (IPATHS) (ZION(I,NN), XION(I,NN), NN = 1, ISAT)
50 CONTINUE

DEFINITION OF THE RIGHT BOUNDARY (USED TO DEFINE ZROUND). PATH FOR RIGHT BOUNDARY IS DEFINED BY TRAJECTORY "KEY" OF FILE IPATHS ALONG WITH THE PATH STATUS AND ITERATIONS ON THE PATH.

ISAT = ISTAT(KEY)
DO 70 NN = 1, ISAT
   ZION(NUM1, NN) = ZION(KEY, NN)
   XION(NUM1, NN) = XION(KEY, NN)
70 CONTINUE

NUM(NUM1) = ISAT

CONTINUE
RETURN
FND
**SURROUNTE INIT**

*********** INITIATION ROUTINE ***********

**PROGRAM DESCRIPTION:** PROGRAMMER = WILLIAM DEININGER, 7 = Z = R1

**REVISIONS:** (INCLUDE DATE, INITIALS AND DESCRIBE CHANGE **PLEASE **)


THE INITIAL DENSITIES ARE CALCULATED AND FINALLY THE RESULTS OF THE INITIALIZATION AND FIRST ITERATION ARE OUTPUT.

******* VARIABLE DICTIONARY *******

**DNI(I) : INITIAL DENSITY BETWEEN PATHS (I-1) AND (I).**

**DN0B : CONSTANT USED IN CALCULATION OF THE DENSITIES, COMBINATION OF QUANTITIES INCLUDING; BMCURP, CEXSEC, RB, NUMION, UTIL, O AND VELNEU.**

**IDEF : USED TO DETERMINE WHEN THE X AND Z COORDINATES OF AN ION TRAJECTORY EXIT POINT ARE DEFINED (EVERY OTHER TIME).**

**IL : INDEX (I) OF LEFT MOST ACTIVE PATH.**

**IR : INDEX (I) OF RIGHT MOST ACTIVE PATH.**

**ISAT : ISTAT(I) FOR KEY PATH IN UPSTREAM RUN.**

**ISTAT(I) : STATUS OF PATH I.**

**= 0 , PATH IS ACTIVE.**

**= 1 TO NUMIT , PATH IS NOT ACTIVE, VALUE IS ITERATION NUMBER WHERE BOUNDARY WAS CROSSED.**

**= ERR , PATH IS NOT ACTIVE, ERROR CONDITION.**

**NIP(I) : TOTAL NUMBER OF ITERATIONS PERFORMED ON PATH I.**

**NUM1 : NUMBER OF ION PATHS PLUS ONE (NUMION + 1).**

**NUM2 : TWO TIMES THE NUMBER OF ION PATHS PLUS ONE (2*NUMION + 1).**

**PIOV2 : PI OVER (DIVIDED BY) 2.**

**RB2 : BEAM RADIUS SQUARED (RB ** 2).**

**RB95N : 95 PERCENT OF THE BEAM RADIUS DIVIDED BY 2 TIMES THE NUMBER OF ION PATHS (RB * .95 / (2 * NUMION)).**

**SPACER : INITIAL DISTANCE BETWEEN PATHS IN UNIFORM DISTRIBUTION.**

**TIME : TIME INTERVAL, DEFINES ITERATION STEP SIZE.**

**VELBET : PRESENT TOTAL VELOCITY BETWEEN PATH UNDER CONSIDERATION AND NEIGHBORING PATH.**

**VELBOH : BOHM VELOCITY.**

**VELNEU : THERMAL VELOCITY OF THE NEUTRALS IN THE CHAMBER.**

**VELT1(I) : PRESENT TOTAL VELOCITY COMPONENT IN X DIRECTION.**

**VELT2(I) : PRESENT TOTAL VELOCITY COMPONENT IN Z DIRECTION.**

**XION(I,1) : X COORDINATE OF ION TRAJECTORY EXIT POINT I.**

**XION(I,2) : FIRST X COORDINATE OF ION AFTER LEAVING ION BEAM ALONG TRAJECTORY I.**

**ZBOUND : Z BOUNDARY TO RIGHT OF THRUSTER.**

**ZCURP : PRESENT Z INCREMENT, USED TO GET ION EXIT POINTS.**

**ZION(I,1) : 7 COORDINATE OF ION TRAJECTORY EXIT POINT I.**

**ZION(I,2) : FIRST Z COORDINATE OF ION AFTER LEAVING ION BEAM ALONG TRAJECTORY I.**

**ZMAX : MAXIMUM Z VALUE ON PATH "KEY" OF IPATHS, DEFINES ZBOUND.**

**ORIGINAL PAGE IS OF POOR QUALITY**
FOR HIGH RESOLUTION UPSTREAM RUN,
ZPREV 1 PREVIOUS R INCREMENT, USED TO GET ION EXIT POINTS.

*** END OF PROGRAM DESCRIPTION AND DICTIONARY ***

PROGRAM DECLARATION STATEMENTS.
BLANK COMMON FOR LARGE ARRAYS, IO = INPUT-OUTPUT, PARAM = PARAMETERS.

COMMON ZION(41,151),XION(41,151),VELTZ(151),VELTX(151),
1 HTP(41),DN(41,151),DN1(42),ISTAT(41)
COMMON /IO/INOUT,INFO(14),KEY,ICLPLT,ICLWR,ITITL(28),
2 IPATHS,IW,IF1(7),IF2(4),ICLERR
COMMON /PARAM/N,NUMIT,NUMION,DIR,RT,TELOUT,BCUR,UTIL,
3 TEL,F,THRFN,UPSTON,VELBOH,ZBOUND,IL,TR,PI,BK,QR,29
4 DONR,CEXSEC,THNU,TIME,TIMEMU,VELMU,VELMU,7STAG,
5 NSTGHT,NSTST,PIOV?

DEFINE CONSTANTS INCLUDING BOHM VELOCITY, THERMAL VELOCITY OF THE
NEUTRALS AND THE TIME INTERVAL.

IL = 1
IR = NUMION
NUM1 = NUMION + 1
NUM2 = 2 * NUMION + 1
PIOV2 = PT / ?
RB2 = RP ** ?
RRB2 = RP * 0.95 / (2.0 * FLOAT(NUMION))
6 VFLBOH = SORT (TELIN / UMSION)
VELNU = SORT (THNU / UMSION)
7PREV = 0.0
6DN0B = (HMCUP ** ?) * CEXSEC * (1.0 - UTIL)) / (FLOAT(NUMION)
7 PREV = 0.0
TIME = TIMEMU * RUNOUT / (VELBOH + FLOAT(NUMIT + NTST))

CALCULATION OF THE X AND 7 COORDINATES OF THE ION TRAJECTORY EXIT
POINTS FROM THE REAM. GIVES A NON-UNIFORM DISTRIBUTION OF ION EXIT
POINTS. (2 * NUMION + 1) POINTS ARE CALCULATED, THE EVEN POINTS
ARE USFD AS ION EXIT POINTS, THE LAST VALUE CALCULATED,
(2 * NUMION + 11TH POINT, DEFINES ZBOUND.

DO 100 II = 1, NUM2
ZCURR = 0.5 * (RRB2 + ZPREV - SORT (ZPREV ** 2 + RB2)) -
7 0.5 * PR2 / (RRB2 + ZPREV - SORT (ZPREV ** 2 + RB2))
9 IDEF = MOD (II, 2)
10 IF (IDEF .EQ. 0) GO TO 95
II = (II + 1) / 2
12 XION(T11) = PR
95 ZION(T11) = THERFN + ZCURR
96 ZPREV = ZCURR
100 CONTINUE
ZBOUND = ZION(NUM1, 1)

IF THIS IS A HIGH RESOLUTION UPSTREAM RUN, ZBOUND MUST BE REDEFINED.
THE LARGEST ZION() VALUE ON THE KEY TH PATH IS USED AS ZBOUND.

IF (KEY) 120, 120, 105
105 ZMAX = ZION (KEY, 1)
ISAT = ISTAT(I)
DO 110 M = 2, ISAT
   IF (?ION(KEY, M) .LE. ZMAX) GO TO 110
      ZMAX = ?ION(NUM1, M)
   110 CONTINUE
ZBOUND = 7MAX
 IF KEY EQUALS -1, A UNIFORM DENSITY DISTRIBUTION RUN IS DONE.
 THE INTERVAL BETWEEN THRLEN AND ZBOUND IS BROKEN INTO NUMION+1
 EQUAL INTERVALS. THE ION PATHS START AT THE END OF EACH INTERVAL.
 IF (KEY .NE. -1) GO TO 125
SPACER = (ROUND - THRLEN) / FLOAT(NUM1)
ZION(I,1) = THRLEN + SPACER
DO 122 I = 2, NUMION
   ZION(I,1) = ZION(I-1,1) + SPACER
122 CONTINUE
125 CONTINUE
INITIALIZATION OF THE TOTAL VELOCITY COMPONENTS AND THE TOTAL NUMBER
OF ITERATIONS PER PATH COUNTER (NIP(I)). CALCULATE NEXT ION
 POSITIONS AND SET ALL ION PATHS ACTIVE.
 DO 130 I = 1, NUMION
      VELTX(I) = VELROH * XVELMU
      VELTZ(I) = VELROH * ZVELMU
      NIP(I) = 2
      XION(I,2) = XION(I,1) + VELTX(I) * TIME
      ZION(I,2) = ZION(I,1) + VELTZ(I) * TIME
      ISTAT(I) = 0
130 CONTINUE
INITIALIZATION OF THE DENSITY ARRAY
 DO 180 I = 2, NUM1
      IF (I .EQ. 1) GO TO 152
      IF (I .EQ. NUM1) GO TO 156
      VELRET = (SORT (VELTX(I-1) ** 2 + VELTZ(I-1) ** 2) +
                SORT (VELTX(I) ** 2 + VELTZ(I) ** 2)) / 2.0
      DNI(I) = DNOA / (((XION(I, 1) + XION(I-1, 1)) / 2.0)
              * (ZION(I, 1) - ZION(I-1, 1)) * VELRET)
      GO TO 160
152 DNI(I) = DNOA / ((XION(I, 1) * (ZION(I, 1) - THRLEN))
              * (SORT (VELTX(I) ** 2 + VELTZ(I) ** 2)))
156 DNI(NUM1) = DNOA / ((XION(I-1, 1) * (ZBOUND - ZION(I-1, 1))
              * (SORT (VELTX(I-1) ** 2 + VELTZ(I-1) ** 2)))
160 CONTINUE
INITIALIZATION OF THE DENSITY ARRAY
 DO 180 I = 2, NUM1
      DNI(I-1, 1) = (DNI(I) + DNI(I-1)) / 2.0
180 CONTINUE
RETURN TO DRIVER AFTER OUTPUTTING HEADING, SCHEMATIC OF THRUSTER,
INITIAL PARAMETERS AND THE TWO SETS OF INFORMATION GENERATED BY THIS SUBROUTINE.

175  TELIN = TELIN / 0
176  TELOUT = TELOUT / 0
177  TTHNEU = TTHNEU / 0
178  CALL WRIT(1)
179  TELIN = TELIN * 0
180  TELOUT = TELOUT * 0
181  TTHNEU = TTHNEU * 0
182  N = 1
183  CALL WRIT(2)
184  N = 2
185  CALL WRIT(2)
186  RETURN
187  END
SUBROUTINE CALC

*************** CALCULATION ROUTINE ***************

PROGRAM DESCRIPTION: PROGRAMMER - WILLIAM DEININGER, 5 - 26 - 81

REVISIONS: (INCLUDE DATE, INITIALS AND DESCRIBE CHANGE ** PLEASE **)

THIS SUBROUTINE USES THE ARRAYS IN BLANK COMMON
ALONG WITH SUBROUTINE CALC1D (GETS DISPLACEMENTS TO RIGHT AND LEFT)
TO DETERMINE THE NEXT POSITION OF THE ION BEING CONSIDERED. THE
DENSITIES AND POTENTIALS TO THE RIGHT AND LEFT OF THE CURRENT PATH
ARE CALCULATED FIRST. THEN THE FORCE ACTING PERPENDICULAR TO THE
CURRENT PATH IS CALCULATED. THE POTENTIALS AND FORCES ACTING
PARALLEL TO THE CURRENT PATH ARE OBTAINED NEXT. THE TOTAL FORCE
(SUM OF PERPENDICULAR AND PARALLEL COMPONENTS) ACTING ON THE ION IS
CALCULATED AND THEN THE VELOCITY COMPONENTS ALONG THE X AND Z AXES
ARE OBTAINED. FINALLY THE NEXT ION POSITION IS CALCULATED. SUB-
ROUTINE BOUND IS CALLED TO MAKE SURE THE NEW ION POSITION IS INSIDE
THE BOUNDARIES. THE RESULTS ARE PRINTED (EVERY 'ICLWRT' TIMES) AND
FLAG1 IS CHECKED (SEE CALC1D) TO SEE IF INTERSECTIONS WERE FOUND TO
BOTH THE RIGHT AND THE LEFT. IF INTERSECTIONS WERE FOUND TO BOTH
SIDES, PATH 1 IS ITERATED AGAIN.

******** VARIABLE DICTIONARY ********

TIME INTERVAL DIVIDED BY MASS OF ION (TIME / UNMOWN)
COSPAR: COSINE OF ANGLE BETWEEN LINE PARALLEL TO PATH AND HORIZONTAL.
COSPER: COSINE OF ANGLE BETWEEN LINE PERPENDICULAR TO PATH AND
HORIZONTAL.
D (I,J): DENSITY ON PATH I AT ITERATION J.
DNL: DENSITY TO LEFT SIDE OF CURRENT PATH.
DNOB: CONSTANT USED IN THE DENSITY CALCULATIONS (C IN THE
REPORTS).
DNP: DENSITY TO RIGHT SIDE OF CURRENT PATH.
DSPLIP: DISPLACEMENT OF ION ALONG PATH.
F: TOTAL FORCE ACTING ON ION.
FPAR: FORCE ACTING PARALLEL TO THE PATH.
FPARX: COMPONENT OF FPAR ACTING IN X DIRECTION.
FPARZ: COMPONENT OF FPAR ACTING IN Z DIRECTION.
FPER: FORCE ACTING PERPENDICULAR TO THE PATH.
FPERX: COMPONENT OF FPER ACTING IN X DIRECTION.
FPERZ: COMPONENT OF FPER ACTING IN Z DIRECTION.
FX: COMPONENT OF F ACTING IN X DIRECTION (FPERX + FPARX).
FZ: COMPONENT OF F ACTING IN Z DIRECTION (FPERZ + FPARZ).
I: ALLOWS THE INDEX, II, TO BE PASSED THROUGH COMMON.
P: TOTAL FORCE ACTING ON ION.

C

ORIGINAL PAGE IS OF POOR QUALITY
*** END OF PROGRAM DESCRIPTION AND DICTIONARY ***

PROGRAM DECLARATION STATEMENTS.

BLANK COMMON FOR LARGE ARRAYS, IO = INPUT-OUTPUT, PARAM = PARAMETERS.

COMMON ZION(41,151),XION(41,151),VELT2(151),VELTX(151),
1 NIP(41),ON(41,151),ON(42),ISTAT(41),
2 IPATHS,IV,JF1(2),JF2(4),JICL,RERR
COMMON / IO / IN,IOUT,INFO(14),KEY,ICLPT,ICLRT,ITIAL(28),
3 TELIN,THREL,UMSION,VELBOH,ZBOUND,IL,IR,PB,SB,Q,RB95N,
4 MND,CEP,EQ,EP,EP,ETIM,TIME,TIMEMU,XVELMU,ZVELMU,NSTAG,
5 NSTAGM1,NSTAGM1,PSTAG

DEFINE CONSTANTS, BEGIN ITERATION OF EACH PATH, TEST FOR PATH ONE
RECOMBINATION AND SET THE CURRENT "WORKING" NUMBER OF ITERATIONS
ON PATH I. MAKE SURE THE TOTAL NUMBER OF ITERATIONS PERFORMED ON
PATH I IS NOT TOO LARGE FOR THE CURRENT STAGE AND SEE IF CURRENT
PATH IS ACTIVE.

NOTE: ANY QUANTITIES OPERATED ON BY "AINT" AND MULTIPLIED
OR DIVIDED BY 1 TIMES SOME POWER OF TEN, ARE BEING
TRUNCATED TO AVOID COMPUTER ROUND-OFF ERROR.

C = AINT (TIME / (UMSION * 1.0E+15))
C = C * 10.0F19
NSTAGM1 = 0
IF (NSTAG .GT. 1) NSTAGM1 = 1
IFLAG4 = 0

ORIGINAL PAGE IS OF POOR QUALITY
DO 450 II = 1, NUMION
   I = II
   IF (II .EQ. 2) IFLAG4 = 1
   IF (NIP(II) .GT. ((NSTAG * NUMIT) - ((NSTAG - 1) * 10)))
      1 IFLAG4 = 0
   N = NIP(II) - (NSTAG - 1) * (NUMIT - 10)
   IF (NIP(II) .GT. ((NSTAG * NUMIT) - ((NSTAG - 1) * 10)))
      2 GO TO 450
   IF (ISTAT(I) .NE. 0) GO TO 450
   C GET THE DISPLACEMENTS TO THE RIGHT AND LEFT, CHECK THE ERROR
   C FLAG, BOUNDARY INTERSECTION FLAG. MAKE SURE PARTICLE TRAJECTORY
   C PATHS ARE SMOOTH AND CALCULATE THE NEEDED TRIGONOMETRIC FUNCTIONS
   C OF THETAP.
   CALL CALLC (I, IFLAG1, IFLAG2, IFLAG3, DSPLR, DSPLL, THETAP)
   IF (IFLAG2 .GT. 0) GO TO 499
   IF (IFLAG3 .LE. 0 .OR. IFLAG3 .GE. 5) GO TO 497
   IF (IFLAG3 .EQ. 4) GO TO 460
   COSPAR = AINT (COS (PIOV2 - THETAP) * 1.0E+03)
   COSPER = AINT (COS (THETAP) * 1.0E+03)
   SINPAR = AINT (SIN (PIOV2 - THETAP) * 1.0E+03)
   SINPER = AINT (SIN (THETAP) * 1.0E+03)
   COSPAR = COSPAR / 1.0E+03
   COSPER = COSPER / 1.0E+03
   SINPAR = SINPAR / 1.0E+03
   SINPER = SINPER / 1.0E+03
   C CALCULATION OF THE LEFT AND RIGHT X-POSITIONS HALF WAY ALONG THE
   C PERPENDICULAR DISPLACEMENTS TO THE NEIGHBORING PATHS
   C (X-COORDINATES AT CENTER OF DENSITY CELLS). "IF STATEMENT"
   C DETERMINES THE DIRECTION OF PATH PROPAGATION.
   SINQV2 = SINPER / 2.0
   IF (ZION(T,N) - ZION(I,N-1)) .LT. 314, 310, 316
   C 310 XPOSIL = XION(I,N)
   XPOSIR = XION(I,N)
   GO TO 320
   C 314 XPOSIL = XION(I,N) - DSPLL * SINQV2
   XPOSIR = XION(I,N) + DSPLL * SINQV2
   GO TO 3PO
   C 318 XPOSIL = XION(I,N) + DSPLL * SINQV2
   XPOSIR = XION(I,N) - DSPLL * SINQV2
   C CALCULATION OF THE TOTAL VELOCITIES BETWEEN THE PRESENT PATH
   C AND THE PATHS TO THE RIGHT AND LEFT.
   IF (II .EQ. 1) GO TO 322
   VELTTL = (SORT (VELTX(I-1) ** 2 + VELTZ(I-1) ** 2)
   + SORT (VELTX(I) ** 2 + VELTZ(I) ** 2)) / 2.0
   IF (T .NE. 0, NUMION) GO TO 323
   VELBTR = (SORT (VELTX(I) ** 2 + VELTZ(I) ** 2) +
   321 SORT (VELTX(I+1) ** 2 + VELTZ(I+1) ** 2)) / 2.0
   GO TO 325

ORIGINAL PAGE IS OF POOR QUALITY
VELBTL = SORT (VELTX(I) ** 2 + VELTZ(I) ** 2)
GO TO 321
VELRTR = SORT (VELTX(I) ** 2 + VELTZ(I) ** 2)

CALCULATION OF THE DENSITIES TO THE RIGHT AND LEFT, AND THE AVERAGE DENSITY AT THE CURRENT POINT. MAKE SURE DNL AND DNR ARE NOT ZERO.

DNL = AINT (DNR / (XPOSL * DSPLL * VELBTL))
DNR = AINT (DNR / (XPOSR * DSPLR * VELTRI))
DN(I,N) = (DNL + DNR) / 2.0
IF (DNL .LE. 0.0 OR DNR .LE. 0.0) GO TO 495

WHEN A BOUNDARY IS INTERSECTED ON THE LEFT OR RIGHT, THE DISPLACEMENTS HAVE TO BE CHECKED TO MAKE SURE NO BOUNDARY REPULSION EXISTS. IF THE DISPLACEMENT FROM THE CURRENT PATH TO THE BOUNDARY IS LESS THEN THE DISPLACEMENT BETWEEN THE CURRENT PATH AND THE NEIGHBORING PATH, THE PERPENDICULAR FORCE IS ZERED, OTHERWISE THE PATH PROPAGATES AS USUAL.

IF (IFLAG3 .EQ. 1) GO TO 340
IF (VELT7(I) .EQ. 0.0) GO TO 340
IF (IFLAG3 .EQ. 2) GO TO 328
GO TO 497
IF (DSPLL .LE. DSPLR) GO TO 330
GO TO 340
FPERX = 0.0
FPERZ = 0.0
GO TO 370
IF (DSPLR .LE. DSPLL) GO TO 330


VL = AINT ((ALOG (DNL / DNI(I)) * TELOUT / 0) * 1.0E+04)
VR = AINT ((ALOG (DNR / DNI(I+1)) * TELOUT / 0) * 1.0E+04)
VL = VL / 1.0E+04
VR = VR / 1.0E+04
FPER = AINT ((G * (VL - VR) / DSPLR) * 1.0E+20)
GO TO 350
FPER = AINT ((G * (VL - VR) / DSPLL) * 1.0E+20)
FPERX = FPER * SINPER
FPERZ = FPER * COSPER

CALCULATION OF THE POTENTIALS AND THE FORCE AND ITS COMPONENTS ACTING PARALLEL TO THE PATH.

VCURR = AINT ((DNI(I,N) / DNI(I,N-1)) * TELOUT / 0)
VPREV = AINT ((DNI(I,N-1) / DNI(I,N)) * TELOUT / 0)
DSPLIP = SORT ((XION(I,N) - XION(I,N-1)) ** 2 + (ZION(I,N))
7  - ZION(I,N-1) ** 2)  
230  FPAR = AINT ((0 % (VPREV - VCURR) / DSPLIP) * 1.0E+20)  
      FPAR = FPAR / 1.0E+20  
      FPARX = FPAR * SINPAR  
      FPARZ = FPAR * COSPAR  
235  C CALCULATION OF TOTAL FORCE AND COMPONENTS.  
      FX = FPARX + FPERX  
      FZ = FPARZ + FPERZ  
      F = SORT (FX ** 2 + FZ ** 2)  
240  C CALCULATION OF VELOCITY COMPONENTS FOR THIS ITERATION, TOTAL  
      VELOCITY COMPONENTS AND TOTAL VELOCITY FOR PATH I.  
      VELX = FX * C  
      VELZ = FZ * C  
390  VELX(I) = VELX(I) + VELX  
      VELZ(I) = VELZ(I) + VELZ  
      VELTOT = SORT (VELX(I) ** 2 + VELZ(I) ** 2)  
250  C THIS SECTION MODIFIES THE VELOCITY ON THE FIRST PATH SO THAT ITS  
      NORMALIZED WITH RESPECT TO 1.2 TIMES THE VELOCITY ON THE SECOND  
      PATH IN TERMS OF MAGNITUDE. THE DIRECTION OF THE TRAJECTORY IS  
      LEFT UNCHANGED. THIS NORMALIZATION ONLY OCCURS WHEN THE VELOCITY  
      ON PATH 1 IS MORE THAN 20 PERCENT GREATER THEN THE VELOCITY ON  
      PATH 2. THE VALUE OF 20 PERCENT IS ARBITRARY AND IS BASED ON  
      WHAT GIVES THE SMOOTHEST TRAJECTORIES. THIS PREVENTS PATH ONE  
      FROM ACCELERATING TO FAST.  
      IF (I *EQ. 1 .AND. IFLAG4 *EQ. 1) GO TO 398  
260  GO TO 410  
398  VELTT2 = SORT (VELX(I+1) ** 2 + VELZ(I+1) ** 2) * 1.2  
      IF (VELTOT = VELTT2) 410, 410, 400  
      VELX(I) = (VELX(I) + VELTT2) / VELTOT  
      VELZ(I) = (VELZ(I) + VELTT2) / VELTOT  
      VELTOT = SORT (VELX(I) ** 2 + VELZ(I) ** 2)  
265  C CALCULATION OF THE NEXT ION POSITION (USES LINEAR APPROXIMATION),  
      MAKE SURF IT INSIDE THE BOUNDARIES.  
      XION(I,N+1) = VELX(I) * TIME + XION(I,N)  
      ZION(I,N+1) = VELZ(I) * TIME + ZION(I,N)  
      CALL BOUND (ZION(I,N+1), XION(I,N+1), I)  
270  410  C WRITE THE RESULTS EVERY "ICLWRT" TIMES. IF ISTAT(I) IS NON-ZERO,  
      WRITE THE RESULTS. INCREASE NIP(I) BY ONE FOR NEXT PASS.  
      IF PATH I REACHED A BOUNDARY ON THIS ITERATION, SET NIP(I) TO ITS  
      FINAL VALUE. IF TO MANY ITERATIONS HAVE OCCURRED SET ISTAT(I) = N.  
      TEST TO SEE IF PATH I NEEDS TO BE ITERATED AGAIN (TEST IFLAG1) AND  
      CHECK TO SEE IF ITERATION LIMIT HAS BEEN REACHED. CHECK IFLAG4  
280  C FOR RENORMALIZATION OF PATH I.  
      IF (ICLWRT .LE. 0) GO TO 440  
      J = MOD (NIP(I), ICLWRT)  
      IF (ISTAT(I) .NE. 0) GO TO 425  
285  IF (J .NE. 0) GO TO 440
WRITE (INUT, 435) I, N, NIP(I), ISTAT(I), ZION(I,N+1),
8 XION(I,N+1), VELZ(I), VELX(I), VELTOT, DN(I,N)
$35 FORMAT (1X, I3, 1X, 3(I4, 2X), 6(E13, 6, 2X))
440 NIP(I) = NIP(I) + 1
290 IF (ISTAT(I) .NE. 0) NIP(I) = NIP(I) - 2
IFVAR = (NSTAG * NUMIT + 1) = ((NSTAG = I) * 9)
IF (ISTAT(I) .GT. IFVAR) ISTAT(I) = N
IF (I .EQ. 1 .AND. IFLAG4 .EQ. 1) GO TO 445
IF (IFLAG1 .EQ. 0) GO TO 300
I = 1
IF (ISTAT(I) .NE. 0) GO TO 445
NIP(I) = NIP(I) - 1
GO TO 300
300 445 IFLAG4 = 0
450 CONTINUE
NSTAG = 1
RETURN

C IF A BOUNDARY IS INTERSECTED ON BOTH SIDES OF THE PATH (TO THE RIGHT
C AND LEFT) ALL FORCES ARE SET EQUAL TO ZERO CAUSING THE Y AND Z
C VELOCITY CONTRIBUTIONS FOR THIS ITERATION TO BE ZERO. THE PATH
C PROPAGATES LINEARLY. THE DENSITY IS TREATED AS A CONSTANT.
310 460 VELX = 0.0
VELZ = 0.0
DN(I,N) = DN(I,N-1)
GO TO 390
315 C ERROR EXITS AND ERROR CONDITIONS
C
- ERROR 527 - IFLAG2 IMPROPERLY DEFINED, FATAL.
- ERROR 530 - DN0 OR DN0 EQUAL ZERO OR LESS THAN ZERO, CAUSES
VL OR VR TO BLOW UP, FATAL.
320 C
495 IFLAG2 = 325
WRITE (IOUT, 530) IFLAG1, IFLAG2, IFLAG3, I, N, DN0, DN0,
9 DSPLL, DSPLR, XPOS, XPOS
325 GO TO 499
497 WRITE (IOUT, 530) IFLAG1, IFLAG2, IFLAG3, I, N
499 ISTAT(I) = 8888
GO TO 420
527 FORMAT (/,11X,23H***** ERROR 527 *****,,/,,11X,
1 33HIFLAG3 IMPROPERLY DEFINED (FATAL),/,,11X,
2 27HCALLED FP0! SUBROUTINE CALC,/,11X,8HIFLAG1 = $15,
3 10H IFLAG2 = $15,10H IFLAG3 = $15,5H I = $15,5H N = $15)
530 FORMAT (/,11X,23H***** ERROR 530 *****,,/,,11X,
1 45HDNL OR DN0 LESS THEN OR EQUAL TO ZERO (FATAL),/,,11X,
335 2 27HCALLED FROM SUBROUTINE CALC,/,11X,
3 8HIFLAG1 = $15,10H IFLAG2 = $15,10H IFLAG3 = $15,5H I =,
4 15,5H N = $15,7H DN0 = $E10.3,7H DN0 = $E10.3,,/,,11X,
5 9H DSPLL = $E10.3,9H DSPLR = $E10.3,9H XPOS = $E10.3,
6 9H XPOS = $E10.3)
340 FND

ORIGINAL PAGE IS OF POOR QUALITY
SUBROUTINE CALCD (I, IFLAG1, IFLAG2, IFLAG3, DSPLR, DSPLL, THETAP)

***** CALCULATION OF THE DISPLACEMENTS TO THE RIGHT AND LEFT *****

Program description: programmer - William Deininger, 5 - 19 - 81

Revisions: (Include date, initials and describe change **please**)

This subroutine uses the arrays ZION(), XION() and NIPI() to get the displacements from the current path to the left and right hand paths. The displacement to the left is denoted "DSPLL" and to the right "DSPLR". First the perpendicular is obtained (SLOPEP), then the line formed by the final two points of the path on the left is obtained (SLOPEL), and finally the X and Z intersections, to the left, are calculated (XINTL, ZINTL). Tests are run to make sure the intersection points are "good" and to determine if linear extrapolation was used. The displacement to the left hand path (DSPLL) is then calculated. Likewise for the right hand displacement (DSPLR). Function (subroutine) DS is called to get the displacements for the special cases, e.g. boundaries on the right or left.

--- IFLAG1 is used to determine if intersections were found to both the left and the right without using linear extrapolation;
   IFLAG1 = 0 Linear extrapolation not used.
   IFLAG1 = 1 Linear extrapolation used on left.
   IFLAG1 = 2 Linear extrapolation used on right.
   IFLAG1 = 3 Linear extrapolation used in both cases.

--- IFLAG2 is the error flag and tells whether or not the neighboring paths are active.
   IFLAG2 = 0 No error.
   IFLAG2 > 0 Error exists, value references program statement where error condition originated.
   IFLAG2 = -1 Path to left is not active.
   IFLAG2 = -2 Path to right is not active.
   IFLAG2 = -3 Neither path (right or left) is active.

--- IFLAG3 is used to determine when there is a boundary on the right or left (or both) of the current path.
   IFLAG3 = 1 No boundary intersected by the perpendicular to the current path on either side.
   IFLAG3 = 2 Boundary intersected on left.
   IFLAG3 = 3 Boundary intersected on right.
   IFLAG3 = 4 Boundary intersected on both the left and right.

This subroutine returns: IFLAG1, IFLAG2, IFLAG3, DSPLL, DSPLR, and THETAP.

**** Variable dictionary ****

DELTA: difference between two X coordinates on current path.
DELTAZ: difference between two Z coordinates on current path.
DELTLX: difference between two X coordinates on left path.
DETLTZ: difference between two Z coordinates on left path.

Original page is of poor quality.
DELTRX: DIFFERENCE BETWEEN TWO X COORDINATES ON RIGHT PATH.
DELTRZ: DIFFERENCE BETWEEN TWO Z COORDINATES ON RIGHT PATH.

DSPLL: SEE ABOVE COMMENTS.
DSPLR: SEE ABOVE COMMENTS.
DUMMYP: DUMMY VARIABLES USED IN THE CALCULATION OF THE INTERCEPTS. CONTAIN INTERMEDIATE RESULTS.

IFLAG1: SEE ABOVE COMMENTS.
IFLAG2: SEE ABOVE COMMENTS.
IFLAG3: SEE ABOVE COMMENTS.
IFVAR: DUMMY VARIABLE USED AS AN ARGUMENT IN AN IF STATEMENT.

NITP(I): TOTAL NUMBER OF COMPLETED ITERATIONS ON PATH I.
N: TOTAL NUMBER OF COMPLETED ITERATIONS FOR PRESENT PATH; TOTAL NUMBER OF ITERATIONS FOR PRESENT STAGE.

NL: NUMBER OF COMPLETED ITERATIONS ON LEFT HAND PATH.
NLM1: NL MINUS 1 (NL - 1), USED FOR INDEXING LEFT HAND PATH.
NR: NUMBER OF COMPLETED ITERATIONS ON RIGHT HAND PATH.
NRM1: NR MINUS 1 (NR - 1), USED FOR INDEXING RIGHT HAND PATH.

SLOPEL: SLOPE OF PATH ON LEFT BETWEEN TWO "WORKING" POINTS.
SLOPEP: SLOPE OF LINE PERPENDICULAR TO CURRENT PATH AT ENDPOINT.
SLOPER: SLOPE OF PATH ON RIGHT BETWEEN TWO "WORKING" POINTS.
THETAP: ANGLE BETWEEN LINE WITH SLOPE "SLOPEP" AND HORIZONTAL.

XINTL: X INTERSECTION TO LEFT, OF LINES "SLOPEP" AND "SLOPEL".
XINTR: X INTERSECTION TO RIGHT, OF LINES "SLOPEP" AND "SLOPEL".
ZINTL: Z INTERSECTION TO LEFT, OF LINES "SLOPEP" AND "SLOPEL".
ZINTR: Z INTERSECTION TO RIGHT, OF LINES "SLOPEP" AND "SLOPEL".

*** END OF PROGRAM DESCRIPTION AND DICTIONARY ***

PROGRAM DECLARATION STATEMENTS.
BLANK COMMON FOR LARGE ARRAYS; IO = INPUT-OUTPUT, PARAM = PARAMETERS.

COMMON ZION(41,151),XION(41,151),VELTZ(151),VELTX(151),NIP(41),DN(41,151),DNT(42),ISTAT(41)
COMMON / IO / IN,OUT,INFO(14),KEY,ICLPLT,ICLWR,TYITL(28),IP Thr,TF1(2),TF2(4),ICLERR
COMMON / PARAM / NUMTON,NUMIT,PRB,NPNUM,PRB,HBOUND,HBOUND,INST,SPOS,VELBOH,ZBOUND,IL,IP,Th,PRB,NP,
4 0 DB,CESEC,TTHNEU,TIME,TIMEHU,XVELMU,ZVELHU,NSTAG
5 NSTGRHU,NSTGRUT,PIOV2

INITIALIZE NECESSARY VARIABLES (FLAGS).

IFLAG1 = 0
IFLAG2 = 0
IFLAG3 = 0

CALCULATE SLOPE OF LINE PERPENDICULAR TO THE CURRENT PATH AT ENDPOINT OF CURRENT PATH (NEGATIVE RECIPROCAL OF SLOPE BETWEEN LAST TWO POINTS ON CURRENT PATH) AND ANGLE THIS LINE MAKES WITH HORIZONTAL

DELTAZ = ZION(I,N-1) - ZION(I,N)
DELTAZ = XION(I,N) - XION(I,N-1)

IF (DELTAZ > 0.0) DELTAZ = 1.0E-16
SLOPEP = DELTAX / DELTAX
THETAP = ATAN(SLOPEP)
DUMMYP = XION(I,N) - SLOPEP * ZION(I,N)
TWO SUBSECTIONS FOLLOW; THE FIRST SUBSECTION OBTAINS THE INTER-
SECTION POINT AND DISPLACEMENT TO THE LEFT, THE SECOND SUBSECTION
OBTAINS THE INTERSECTION POINT AND DISPLACEMENT TO THE RIGHT.

CALCULATIONS FOR LEFT:

CHECK TO SEE IF THIS IS THE FIRST ACTIVE PATH AND INITIALIZE
THE COUNTER FOR THE LEFT HAND PATH. THE FIRST ACTIVE PATH MUST
BE HANDLED AS A SPECIAL CASE (IL = 1). CHECK TO SEE IF LEFT HAND
PATH IS ACTIVE.

IF (I - TL) 492, 460, 307
NL = NIP(I-1) - (HSTAG - 1) + (NUMIT - 9)
NLM1 = NL = 1
IF (ISTAT(I-1)) 470, 311, 470
CALCULATE SLOPE OF PATH ON LEFT BETWEEN FINAL TWO (OR TWO
"WORKING") POINTS.

IF (DELTLZ .LE. 0.0) DCLTLZ = 1.0E-16
SLOPEL = DELTLX / DELTLZ
MAKE SURE INTERSECTION CAN BE FOUND, THE SLOPES OF THE TWO LINES
CAN NOT BE THE SAME. STATEMENTS 312 AND THOSE RIGHT BELOW,
BACKSTEP THE LEFT HAND PATH ONE ITERATION, ALLOWING A NEW SLOPE
TO BE CALCULATED. IF BACKSTEPPING IS NOT NEEDED, THE
INTERSECTION POINTS ARE CALCULATED.

IF (SLOPEP = SLOPEL) 313, 312, 313
IF (IFLAG2 .EQ. 312) GO TO 481
IF (IFLAG2 .EQ. 311) GO TO 401
NL = NL = 1
IF (NL .EQ. 1) GO TO 480
NLM1 = NL = 1
GO TO 311
DUMMYL = XION(I-1,NL) - SLOPEL * ZION(I-1,NL)
ZINTL = (DUMMYL - DUMMYP) / (SLOPEP - SLOPEL)
XINTL = SLOPEP * ZINTL + DUMMYP
TEST TO SEE IF INTERSECTION POINTS ARE GOOD, FIRST FIND DIRECTION
OF PATH PROPAGATION, 320 IMPLIES NEGATIVE Z DIRECTION, 325 IMPLIES
POSITIVE Z DIRECTION, 330 IMPLIES VERTICAL (UP OR DOWN) DIRECTION.
THEN TESTS ARE RUN TO SEE IF THE INTERSECTIONS ARE "GOOD", IF
LINEAR EXTRAPOLATION IS USED (SETS IFLAG1), OR IF BACK STEPPING
IS NEEDED. THESE TESTS ARE RUN IN ALL CASES.

IF (ZION(I-1,NL) = ZION(I-1,NLM1)) 320, 330, 325
IF (ZION(I-1,NL) .LE. ZINTL .AND. ZINTL .LE. ZION(I-1,NLM1)) GO TO 355
IF (ZINTL .LT. ZION(I-1,NL)) GO TO 353
IF (ZINTL .GT. ZION(I-1,NLM1)) GO TO 312
IFLAG2 = 322
GO TO 486
97

C 325 IF (ZION(I-1,NLM1) .LE. ZINTL .AND. ZINTL .LE. 
175 ZION(I-1,NL)) GO TO 355
327 IF (ZINTL .GT. ZION(I-1,NL)) GO TO 353
327 IF (ZINTL .LT. ZION(I-1,NLM1)) GO TO 312
IFLAG2 = 327
GO TO 486

180 WHEN STATEMENT 330 IS CALLED WE HAVE TO TEST THE X COMPONENTS
TO FIND THE X DIRECTION OF PROPAGATION; THEN SEE IF THE
INTERSECTIONS ARE "GOOD", IF LINEAR EXTRAPOLATION IS USED,
OR IF BACK STEPPING IS NEEDED.
185 330 IF (XION(I-1,NL) - XION(I-1,NLM1)) 340, 484, 345
C 340 IF (XION(I-1,NL) .LE. XINTL .AND. XINTL .LE. 
8 XION(I-1,NLM1)) GO TO 355
IF (XINTL .LT. XION(I-1,NL)) GO TO 353
190 342 IF (XION(I-1,NL) .GT. XION(I-1,NLM1)) GO TO 312
IFLAG2 = 342
GO TO 486
C 345 9 IF (XION(I-1,NLM1) .LE. XINTL .AND. XINTL .LE. 
195 XION(I-1,NL)) GO TO 355
IF (XINTL .GT. XION(I-1,NL)) GO TO 353
347 IF (XINTL .LT. XION(I-1,NLM1)) GO TO 312
IFLAG2 = 347
GO TO 486
200 STATEMENT 350 RESETS IFLAG2 IF STATEMENT 404 IS REFERENCED.
STATEMENT 353 SETS IFLAG1, STATEMENT 355 CALCULATES THE
DISPLACEMENT TO THE LEFT. MAKE SURE IFLAG2 IS SET PROPERLY.
205 350 IFLAG2 = 0
GO TO 355
353 IFLAG1 = IFLAG1 + 1
355 DSPLL = SORT ((XINTL - XION(I,N)) ** 2 + 1
210 (ZINTL - ZION(I,N)) ** 2)
IF (IFLAG2 .GT. 0) IFLAG2 = IFLAG2 - 312

CALCULATIONS FOR RIGHT.

215 CHECK TO SEE IF THIS IS THE LAST ACTIVE PATH AND INITIALIZE THE
COUNTER FOR THE RIGHT HAND PATH. THE LAST ACTIVE PATH MUST
BE HANDLED AS A SPECIAL CASE (IR = NUMION). CHECK TO SEE IF THE
LEFT HAND PATH IS ACTIVE.
220 358 IF (IR = 1) 499, 461, 359
359 NF = NIP(I+1) = (NSTAG - 1) * (NUMIT - 1)
361 NR = ISTAT(I+1) + 1
358 IF (IR = 1) 472, 361, 472
225 CALCULATE SLOPE OF PATH ON RIGHT BETWEEN FINAL TWO (OR TWO
"WORKING") POINTS.
361 DELTRZ = ZION(I+1,NR) - ZION(I+1,NRm1)
DELTRX = XION(I+1,NR) - XION(I+1,NR1)

IF (DELTRX .EQ. 0.0) DELTRX = 1.0E-16
SLOPEP = DELTRX / DELTRZ

MAKE SURE INTERSECTIONS CAN BE FOUND, THE SLOPES OF THE TWO LINES CANNOT BE THE SAME. STATEMENTS 362 BACKSTEP THE RIGHT HAND PATH ONE ITERATION, ALLOWING A NEW SLOPE TO BE CALCULATED. IF BACKSTEPPING IS NOT NEEDED, THE INTERSECTION POINTS ARE CALCULATED.

IF (SLOPEP = SLOPER) 363, 362, 363

IF (IFLAG2 .GE. 359 .AND. IFLAG2 .LE. 362) GO TO 483
NR = NR = 1
IF (NR .EQ. 1) GO TO 482
NR1 = NR = 1
GO TO 361

DUMMYR = XION(I+1,NR) - SLOPER * ZION(I+1,NR)
ZINTR = (DUMMYR - DUMMYP) / (SLOPEP - SLOPER)
XINTR = SLOPEP * ZINTR + DUMMYP

TEST TO SEE IF INTERSECTION POINTS ARE GOOD. FIRST FIND DIRECTION OF PATH PROPAGATION, 370 IMPLIES NEGATIVE Z DIRECTION, 375 IMPLIES POSITIVE Z DIRECTION, 380 IMPLIES VERTICAL (UP OR DOWN) DIRECTION. AS BEFORE, TESTS ARE RUN TO SEE IF THE INTERSECTIONS ARE "GOOD", IF LINEAR EXTRAPOLATION IS USED (SETS IFLAG1), OR IF BACK STEPPING IS NEEDED. THESE TESTS ARE RUN IN ALL CASES.

IF (ZION(I+1,NR) = ZION(I+1,NR1)) 370, 380, 375

2 IF (ZION(I+1,NR) .LE. ZINTR .AND. ZINTR .LE. ZION(I+1,NR1)) GO TO 397
3 IF (ZINTR .LT. ZION(I+1,NR)) GO TO 395
4 IF (ZINTR .GT. ZION(I+1,NR1)) GO TO 392
IFLAG2 = 372
GO TO 490

IF (ZION(I+1,NR1) .LE. ZINTR .AND. ZINTR .LE. ZION(I+1,NR1)) GO TO 377
2 IF (ZINTR .LT. ZION(I+1,NR1)) GO TO 390
3 IF (ZINTR .GT. ZION(I+1,NR1)) GO TO 395
4 IFLAG2 = 377
GO TO 490

WHEN STATEMENT 380 IS CALLED, WE HAVE TO TEST THE X COMPONENTS IN THE SAME MANOR AS WHEN STATEMENT 330 WAS CALLED TO LOOK TO THE LEFT. NEED TO TEST FOR "GOOD" INTERSECTIONS, LINEAR EXTRAPOLATION AND BACK STEPPING.

IF (XION(I+1,NR) = XION(I+1,NR1)) 385, 400, 390

2 IF (XION(I+1,NR) .LE. XINTR .AND. XINTR .LE. XION(I+1,NR1)) GO TO 397
3 IF (XINTR .LT. XION(I+1,NR)) GO TO 395
4 IF (XINTR .GT. XION(I+1,NR1)) GO TO 362
IFLAG2 = 387
GO TO 490
99

390 IF (XION(I,N+1) .LE. XINTR .AND. XINTR .LE. XION(I-1,N)) GO TO 397
IF (XINTR .GT. XION(I+1,N)) GO TO 395
392 IF (XINTR .LT. XION(I-1,N+1)) GO TO 362
IFLAG2 = 392
GO TO 408
C STATEMENT 394 RESETS IFLAG2 IF STATEMENT 490 IS REFERENCED.
STATEMENT 395 SETS IFLAG1; STATEMENT 397 CALCULATES THE
DISPLACEMENT TO THE RIGHT. MAKE SURE IFLAG2 AND IFLAG3 ARE
PROPERLY SET.
C
394 IFLAG2 = 0
GO TO 397
300 IFLAG1 = IFLAG1 + 2
397 DSPLR = SORT ((XINTR - XION(I,N)) ** 2 +
6 (ZINTR - ZION(I,N)) ** 2)
IF (IFLAG2 .GE. 0) IFLAG2 = IFLAG2 - 362
IF (IFLAG2 .EQ. 0) GO TO 405
GO TO 418
405 IF (IFLAG2 .NE. -2 .AND. IFLAG1 .NE. 2) GO TO 450
IFLAG3 = 4
GO TO 450
C
310 DEFINE IFLAG3.
C
THE FOLLOWING SECTION EXAMINES IFLAG1 AND IFLAG2 SO THAT IFLAG3 CAN BE
DEFINED. IFLAG3 IS USED IN CALC TO DETERMINE WHEN BOUNDARY REPULSION
EXISTS SO THAT THE NECESSARY FORCES CAN BE ZEREO. STATEMENT 4301
BOUNDARIES INTERSECTED ON BOTH SIDES; STATEMENT 4351 BOUNDARY
INTERSECTED ON RIGHT; STATEMENT 4401 BOUNDARY INTERSECTED ON LEFT;
STATEMENT 4451 NO BOUNDARIES INTERSECTED.
C
320 418 IF (IFLAG2) 420, 445, 496
420 IF (IFLAG1) 494, 445, 423
423 IF (IFLAG2 .EQ. -1 .AND. IFLAG1 .EQ. 2) GO TO 445
IF (IFLAG2 .EQ. -1) GO TO 440
IF (IFLAG2 .EQ. -2 .AND. IFLAG1 .EQ. 1) GO TO 445
325 IF (IFLAG2 .EQ. -2) GO TO 435
IF (IFLAG2 .NE. -3) GO TO 496
IF (IFLAG1 .EQ. 1) GO TO 440
IF (IFLAG1 .EQ. 2) GO TO 435
IF (IFLAG1 .EQ. 3) GO TO 430
GO TO 494
330 430 IFLAG3 = 4
GO TO 450
435 IFLAG3 = 3
GO TO 450
335 440 IFLAG3 = 2
GO TO 450
445 IFLAG3 = 1
450 IF (DSPLL .GE. 0.25) GO TO 502
452 IF (DSPLR .GE. 0.25) GO TO 504
455 RETURN
C
C CALCULATIONS FOR (BOUNDARY) SPECIAL CASES.
STATEMENTS 460 AND 461 CALCULATE THE DISPLACEMENT TO THE LEFT
OF THE FIRST ACTIVE PATH TO THE BOUNDARY AND TO THE RIGHT OF THE
LAST ACTIVE PATH TO THE BOUNDARY, RESPECTIVELY. WHEN STATEMENT
461 IS CALLED IFLAG3 HAS TO BE PROPERLY SET.

460 DSPLL = DS (ZION(I,N), XION(I,N), SLOPEP, 0, I)
IF (DSPLL .LE. 1.0E+20) GO TO 490
IFLAG3 = 2
GO TO 35A

461 DSPLR = DS (ZION(I,N), XION(I,N), SLOPEP, 1, I)
IF (DSPLR .LE. 1.0E+20) GO TO 500
IF (IFLAG3 .NE. -1 .AND. IFLAG1 .NE. 1) GO TO 465
IFLAG3 = 4
GO TO 466

465 IFLAG3 = 3
RETURN

OTHER TESTS AND ASSIGNMENTS.

DEFINE IFLAG? IN CASES WHERE TESTED PATH IS INACTIVE.

470 IFLAG2 = -1
GO TO 311

472 IF (IFLAG2) 475, 478, 496
475 IFLAG2 = -3
GO TO 361
478 IFLAG2 = -2
GO TO 361

BACK-STEPPING LOGIC WHEN MORE THEN TEN BACKSTEPS ARE NEEDED ON
A PARTICULAR PATH, ALLOWS USE OF INFORMATION IN CORE STORAGE
NOT YET OVERWRITTEN WITH NEW RESULTS.

FOR LEFT-HAND PATH,

480 IFLAG2 = IFLAG2 + 312
IFVAR = NIP(I-1) - (NSTAG - 1) * (NUMIT - 9)
IF (IFVAR .GE. 140) GO TO 506
NL = NLM1
NL1 = NLM1 - 1
GO TO 311

FOR RIGHT-HAND PATH,

482 IFLAG2 = IFLAG2 + 362
IFVAR = NIP(I+1) - (NSTAG - 1) * (NUMIT - 9)
IF (IFVAR .GE. 140) GO TO 508
NRM1 = 141
GO TO 361

483 IFVAR = NIP(I+1) - (NSTAG - 1) * (NUMIT - 9)
        IF (NRM1 .LE. (IFVAR + 2)) GO TO 508
        NR = NRM1
        NRM1 = NRM1 - 1
        GO TO 361

ERROR CONDITIONS.

***** STATEMENTS 484 THROUGH 500 ARE VARIOUS ERROR EXITS. *****
THE VALUE OF IFLAG2 DETERMINES IF VALUES OUTPUT ARE FOR RIGHT OR
LEFT, SINCE VALUE OF IFLAG2 REFERENCES A PROGRAM STATEMENT.

415 ERROR 521 - STATEMENTS 506, 508 MEAN INDICES NL, NR WERE BACK-
STEPED (130 TIMES) UNTIL CORE STORAGE NO LONGER
CONTAINED VALUES THAT WERE CALCULATED DURING THE
PREVIOUS STAGE. CORRECT COMPARISONS CAN NOT BE
MADE PAST THIS POINT. (FATAL).

420 ERROR 522 - STATEMENTS 486, 488 MEAN THAT THE INTERSECTION POINTS
CONSIDERED IN STATEMENTS 322, 327, 342, 347, 372,
377, 387 AND 392 DO NOT SATISFY ANY LOGICAL
CRITERION. UNPHYSICAL INTERSECTIONS (FATAL).

425 ERROR 523 - STATEMENTS 484, 490 INDICATE THAT POINTS N AND N-1
ARE THE SAME, ION ON LEFT OR RIGHT HAND PATH,
RESPECTIVELY, DID NOT MOVE. UNPHYSICAL UNLESS ION
HAS ZERO VELOCITY AND THERE IS NO NET FORCE
ACTING ON IT (FATAL).

430 ERROR 524 - STATEMENT 492 INDICATES THAT I IS LESS THAN ONE OR
GREATER THEN NUMION. SHOULD NOT OCCUR SINCE I IS
DO LOOP INDEX (FATAL).

435 ERROR 525 - IFLAG1 IMPROPERLY DEFINED (FATAL).

ERROR 526 - IFLAG2 IMPROPERLY DEFINED (FATAL).

ERROR 528 - DSSL1 OR DSSLR COULD NOT BE DEFINED (FATAL).

ERROR 529 - DSSL1 OR DSSLR UNUSUALLY LARGE (NON-FATAL).

484 IFLAG2 = 484

445 IF (VELT(I-1) .EQ. 0.0) AND (VELT(I-1) .EQ. 0.0)
7GO TO 350
WRITE (IOUT, 523) IFLAG2, I, NL, N, ZION(I-1,1NL), ZION(I-1,NLM1),
8 XION(I-1,NL), XION(I-1,NLM1), ZINTL, XINTL, SLOPEP, SLOPEL
GO TO 510

450 486 WRITE (IOUT, 522) IFLAG2, I, NL, N, ZION(I-1,NL), ZION(I-1,NLM1),
9 XION(I-1,NL), XION(I-1,NLM1), ZINTL, XINTL, SLOPEP, SLOPEL
GO TO 510

488 WRITE (IOUT, 522) IFLAG2, I, NR, N, ZION(I+1,NR), ZION(I+1,NRM1),
1 XION(I+1,NR), XION(I+1,NRM1), ZINTR, XINTR, SLOPEP, SLOPER
455 GO TO 510

490 IFLAG2 = 490

ORIGINAL PAGE IS
OF POOR QUALITY
IF (VELTX(I+1) .EQ. 0.0) AND, VELTZ(I+1) .EQ. 0.0)
290 TO 394
WRITE (IOUT, 523) IFLAG2, I, NR, N, ZION(I+1,NR), ZION(I+1,NRM1),
460 XION(I+1,NR), XION(I+1,NRM1), ZINTR, XINTR, SLOPEP, SLOPER
GO TO 510
492 WRITE (IOUT, 524) I, IL, IR, NUMION
GO TO 510
494 WRITE (IOUT, 525) IFLAG1, I, N
496 WRITE (IOUT, 526) IFLAG2, I, N
GO TO 510
498 IFLAG2 = 460
WRITE (IOUT, 528) IFLAG2, I, N
GO TO 510
500 IFLAG2 = 461
WRITE (IOUT, 528) IFLAG2, I, N
GO TO 510
502 IF (ICLERR .EQ. 1) GO TO 452
IFLAG2 = 395
WRITE (IOUT, 529) IFLAG2, I, N, NL, DSPLP, SLOPEP, SLOPEP, XINTL,
4 IFLAG2 = 0
GO TO 452
504 IF (ICLERR .EQ. 1) GO TO 510
IFLAG2 = 397
WRITE (IOUT, 529) IFLAG2, I, N, NR, DSPLR, SLOPER, SLOPER, XINTR,
5 XINTL, DUMMY, DUMMY, XION(I,N), ZION(I,N)
IFLAG2 = 0
GO TO 452
506 WRITE (IOUT, 521) IFLAG2, I, NL, N, SLOPEP, SLOPEP, ZINTL,
6 XINTL, DUMMY, DUMMY, XION(I-1,NL), XION(I-1,NRM1),
7 ZION(I-1,NL), ZION(I-1,NRM1), DSPLL, THETAP
GO TO 510
508 WRITE (IOUT, 521) IFLAG2, I, NR, NRM1, N, SLOPEP, SLOPER, ZINR,
GO TO 510
8 XINTR, DUMMY, DUMMY, XION(I+1,NR), XION(I+1,NRM1),
9 ZION(I+1,NR), ZION(I+1,NRM1), DSPLP, THETAP
510 RETURN

ERROR CONDITION FORMATS, ERROR NUMBER IS FORMAT NUMBER.

C
C 521 FORMAT (/1X,23H******** ERROR 521 ********///11X,)
1 36HNL OR NR RACK STEPPED TO FAR (FATAL)///11X,
2 29HCALLED FROM SUBROUTINE CALCD///11X,
500 3 81HFLAG2 = 15,5H I =,15,13H N(L OR R) =,15,9H N(L OR R),
4 6HRM1 =,15,5H N =,15,10H SLOPEP =,E9,3,12H SLOPEP(L OR R),
5 5H R =,E9,3,16H ZINTL OR R) =,E9,3,14HINTL OR R),
6 5H R) =,E9,3,10H DUMMY =,E9,3,17H DUMMY(L OR R) =,E9,3,
7 /11X,93H ZION(I (= OR +) 1, N(L OR R)) =,E9,3,
505 8 33HXION(I (= OR +) 1, N(L OR R)) =,E9,3,
9 33HZION(I (= OR +) 1, N(L OR R)) =,E9,3,
1 33HZION(I (= OR +) 1, N(L OR R)) =,E9,3,
2 13H DSPL(L OR R) =,E9,3,10H THETAP =,E9,3)
522 FORMAT (/1X,23H******** ERROR 522 ********///11X,)
510 1 39HUNPHYSICAL INTERSECTION POINTS (FATAL)///11X,
2 29HCALLED FROM SUBROUTINE CALCD///11X,
3 81HFLAG2 =,15,5H I =,15,13H N(L OR R) =,15,
4 5H N =,15,33H ZION(I (= OR +) 1, N(L OR R)) =,E9,3,///11X,
**SUBROUTINE** BOUND (Z, X, I)

************ BOUNDARY CHECK ROUTINE ************

PROGRAM DESCRIPTION: PROGRAMMER = WILLIAM DEININGER, 1 - 0 - 92
REVISIONS: (INCLUDE DATE, INITIALS AND DESCRIBE CHANGE ** PLEASE **)

THIS SUBROUTINE CHECKS THE POINT (Z,X) TO SEE IF IT LIES INSIDE THE DEFINED BOUNDARIES OF THE SIMULATION. IF (Z,X) DOES LIE INSIDE THE DEFINED BOUNDARIES, NO CHANGES ARE MADE AND CONTROL IS RETURNED TO SUBROUTINE CALC. IF (Z,X) LIES OUTSIDE THE DEFINED BOUNDARIES, THE PATH STATUS IS SET EQUAL TO THE ITERATION NUMBER. IN ADDITION, IF (Z,X) LIES ON THE FIRST OR LAST ACTIVE PATH AND LIES OUTSIDE THE BOUNDARIES, THE LEFT-MOST (IL) OR RIGHT-MOST (IR) INDEX IS RESET.

******** VARIARLF DICTIONARY ********

I PATH INDEX
INDEX NEW : NEW INDEX FOR RIGHT OR LEFT-MOST PATH.
II DO LOOP INDEX.
IL INDEX OF LEFT-MOST ACTIVE PATH.
IR INDEX OF RIGHT-MOST ACTIVE PATH.
LASTDO COMMON VARIABLE DENOTING LAST VALUE OF DO LOOP INDEX.
X X COORDINATE OF POINT TO BE TESTED.
Z Y COORDINATE OF POINT TO BE TESTED.

*** END OF PROGRAM DESCRIPTION AND DICTIONARY ***

PROGRAM DECLARATION STATEMENTS,

BLANK COMMON FOR LARGE ARRAYS, IO = INPUT-OUTPUT, PARAM = PARAMETERS.

COMMON ZION(161), XION(161), VELZ(161), VELX(161),
1 COMMON P(161), nm(161), ONI(42), ISTAT(42),
2 COMMON / IO / IN/IOU/T,IPDD(14), KEY, ICLPT, ICLRT, ITITL(20),
3 COMMON / PAAM / N, NUMION, NUMRT, RBOUND, RT, TELOUT, BCUR, UTIL,
4 COMMON / PATH, I, PATH(4), CPATH(4), ICLERP,
5 COMMON / INEIOUTrINF0(14), PRV, PLTT, PCL, PLT, PRT, PLT,
6 COMMON / VELB, ZBOUND, IL, IR, PIB, GB, RO15N,
7 COMMON / ND, CESE, YTHNEU, TIME, JONEU, XVELMU, ZVELMU, NSTAG,
8 COMMON / NSTD, NTOOT, PIV02

TEST TO SEE IF THE X COORDINATE IS LESS THEN OR EQUAL TO THE BEAM RADIUS, OR GREATER THEN OR EQUAL TO RBOUND. TEST TO SEE IF THE Z COORDINATE IS LESS THEN OR EQUAL TO 0 (ZERO) OR GREATER THEN OR EQUAL TO THE THRUSTER LENGTH. IF ANY OF THE ABOVE TESTS ARE TRUE, SET ISTAT(I) = N, OTHERWISE RETURN TO SUBROUTINE CALC.

IF (X .LE. RA) GO TO 50
IF (X .GE. RBOUND) GO TO 50
IF (Z .LE. 0.0) GO TO 50
IF (Z .GE. RT) GO TO 50
IF (Z .GE. RBOUND) GO TO 50
IF (Z .LE. THRTEN .AND. X .LE. RT) GO TO 50
RETURN

ISTAT(I) = N

IF (Z,X) IS ON THE FIRST OR LAST ACTIVE PATH AND LIES OUTSIDE THE
C DEFINED BOUNDARIES, RESET THE CORRESPONDING INDEX TO THE INDEX OF
THE NEXT ACTIVE PATH.

FOR LEFT MOST PATH,
---------------------

IF (I - IL) 200, 80, 120
80 CONTINUE
LASTDO = NUMION = IL
DO 100 II = 1, LASTDO
IDXNEW = I + II
IF (ISTAT(IDXNEW)) 210, 105, 100
100 CONTINUE
105 IL = IDXNEW
RETURN

FOR RIGHT MOST PATH,
---------------------

120 IF (I - IR) 160, 125, 200
125 CONTINUE
DO 140 II = 1, IR
IDXNEW = I - II
IF (ISTAT(IDXNEW)) 210, 145, 140
140 CONTINUE
145 IR = IDXNEW
160 RETURN

ERROR CONDITIONS,
-------------------

***** STATEMENTS 200 THROUGH 210 ARE ERROR EXITS. *****
- ERROR 610 - I, IL OR IR DEFINED INCORRECTLY, FATAL.
- ERROR 612 - ISTAT(I) IMPROPERLY DEFINED, FATAL.

200 WRITE (IOUT,610) I, IL, IR
ISTAT(I) = RRRR
GO TO 250
210 WRITE (IOUT,612) I, IDXNEW, ISTAT(I), ISTAT(IDXNEW), N,
NIP(I), NIP(IDXNEW)
100 ISTAT(I) = RRRR
250 RETURN

ERROR CONDITION FORMATS, ERROR NUMBER IS FORMAT NUMBER.

610 FORMAT (/11X,23H****** FORMAT 610 ******,/11X,
1 39HI, IL OR IR DEFINED INCORRECTLY (FATAL),/11X,
2 28HCALLED FROM SUBROUTINE BOUNDARY,/11X,3HI = I5,6H IR = I5)
612 FORMAT (/11X,23H****** ERROR 612 ******,/11X,
1 34HISTAT(I) IMPROPERLY DEFINED (FATAL),/11X,
2 29HCALLED FROM SUBROUTINE BOUNDARY;/11X,3HI = I5,10H IDXNEW = I5,12H ISTAT(I) = I5,
4 17H ISTAT(IDXNEW) = I5,5H N = I5,10H NIP(I) = ,
5 I5,15H NIP(IDXNEW) = I5)
115 END
SUBROUTINE WRIT(KF)
C-- WRIT PRINTS INFORMATION ABOUT THE SIMULATION
C-- KE = 1: OUTPUT HEADING, INITIAL INFORMATION AND DATA
C-- KE = 2: OUTPUT INTERIM STATUS OF MAIN VARIABLES
C-- KE = 3: FINISH OF A PASS, RESULTS, START OF NEW PASS
C-- KE = 4: CREATE FILE OF PATH COORDINATES
C-- KE = 5: CREATE FILE OF POSITION - DENSITY TRIPLETS.
C
C BLANK COMMON FOR LARGE ARRAYS
COMMON ZION(141,151),XTON(141,151),VELTZ(151),VELTX(151),
  1 NIP(41),DNI(41,151),STAT(41)
COMMON / IO / IN,OUT,INFO(14),KEY,ICPLT,ICTLRW,ITIL(28),
  2 IPATHS,IPF1(2),IPF2(4),ICLERR
COMMON / PARAM / NUMION,NUMIT,RTB,BOUNDB,RBOUND,RT,TELOUT,BMCUR,UTIL,
  3 TELIN,THRENNI,VELOM,ROUND,IR,PIBK,GRB95N,
4 DNOB,CEXSC,TTHNF,TIME,TIMEP,XVELMU,ZVELMU,NSTAG,
5 NSTMU,NDTPOS,PTNV2
DATA IPAG,LARPNPAS /0,6HPLASIM,1 /

C------- FORMATS
C
10 FORMAT(///,43HO THIS RUN MAY BE CHARACTERIZED BY INFO/// )
11 FORMAT(1HL///,60X,A13////// )
13 FORMAT(///,17X,33HP L A S M A T S I M U L A T I O N,///
  1 17X,33HA COMPUTER CODE TO DESCRIBE THE PROPAGATION,///
  2 17X,40HP THE JET PROPULSION LABORATORY,///
  3 21X,27H J P L P.,///
  4 17X,47L L A T I O N I NVESTIGATOR///
  5 25X,25HCOLORADO STATE UNIVERSITY,///
  6 26X,23HFORT COLLINS, CO 80523,///
  7 33X,3HFALL 1981///
21 FORMAT(///,27HINITIAL VALUES OF PARAMETERS,,///,
  1 5X,22HSECOND DATA CARD,///,
  2 7X,33HO 6XNUMION,5X,5HNUMIT/,///,
  3 10X,7110,///,
  4 5X,21HFROM THIRD DATA CARD///,
  5 18X,2HRB,4X,4HTHRLEN,5X,5HBMCUR,6X,4HTHELIN,///,
  6 4X,4HTELOUT,4X,4HTHNLNEU,4X,4HCEXSC,4X,4HTHNF,///,
  7 3F10.3///,
  8 5X,21HFROM FIFTH DATA CARD///,
  9 16X,4HTIME,4X,4HTHELIN,///,
21 FORMAT(///,12(5H -2- ),A6,I3///,
  1 10X,13(1H+,13(4H - ),///,
  2 15X,5HCEOS,///,
  3 2(1X,1H+,13(4H - ),///,
21 FORMAT(///,27HINTERIM STATUS -- ITERATION,///,
  1 14,3H OF,14,11H ITERATIONS,///,
C-- WHAT KIND OF CALL IS IT
GOTO (1,2,3,4,5),KF
C-- INITIAL STATE--HEADING AND DATA
1 IPAG=IPAG + 1
WRITE(IOUT,11)LAB,IPAG
WRITE(IOUT,13)
IPAG = IPAG + 1
WRITE(IOUT,14)LAB,IPAG
WRITE(IOUT,11)LAB,IPAG
WRITE(IOUT,10)
WRITE(IOUT,F1) (INFO(K),K=1,IW)
WRITE(IOUT,15) NUMION, NUMIT, KEY, ICLWRT, ICLPLT, NTOTST, ICLERR,
1 RR, PBOUND, RT, THREN, BMCUR, UTIL,
2 TFIN, TFLOUT, TTHNEU, CEXSEC, UMSON,
3 TIMEM, XVELMU, ZVELMU,
4 TIME, VELBOH, ZBOUND
85 RETURN
C-- THIS SECTION PRINTS THE INTERIM STATUS AT THE NTH ITERATION
2 IPAG = IPAG + 1
WRITE(IOUT,21)LAB,IPAG,N,NUMIT
DO 28 I=1,NUMION
28 WRITE(IOUT,22)I,N,ISTAT(I),ZION(I,N),XION(I,N),VELTZ(I),
? VFLTX(I),NN(I,N)
RETURN
C-- THIS SECTION PRINTS RESULT OF A PASS AT NTH EXTRAPOLATION
3 NITP = N + (NSTAG - 1) * (NUMIT + 1) - (10 * (NSTAG - 1))
ITTOTN = (NTOTST * NUMIT + 1) - (NTOTST - 1) * 10
WRITE(IOUT,31) NITP, ITTOTN
RETURN
100 C-- THIS SECTION CREATES FILE OF PATH COORDINATES
CWD DEVICE CODES SHOULD BE CHANGED SO AS NOT TO INTERFER WITH WRIT(5)
4 NMAX=IFIX(FLOAT(NUMION)/4.,)
REWRITE IPATHS
WRITE(IPATHS) NMAX,NUMION
WRITE(IPATHS) (ISTAT(T),I=1,NMAX)
DO 44 I=1,NMAX
ISAT=ISTAT(I)
44 WRITE(IPATHS) (ZION(I,NN),XION(I,NN),NN=1,ISAT)
110 RETURN
C-- (WRIT(5) WRITTEN BY WILLIAM DEININGER)
THIS SECTION WRITES INFORMATION FROM THE FIRST (NUMIT - 9) ITERATIONS IN CORE MEMORY TO AN EXTERNAL FILE. THE INFORMATION IS STORED ON THE EXTERNAL FILE IN "TRIPLETS"; EACH TRIPLET CONTAINS ONE VALUE EACH FOR "XION()", "ZION()" AND "DN()". THIS IS DONE IN PATH MAJOR ORDER. IN OTHER WORDS, ALL THE DESIRED RESULTS FOR ONE PATH ARE OUTPUT BEFORE OUTPUTTING ANY RESULTS FOR NEIGHBORING PATHS.

FIRST THE PATH STATUS IS CHECKED TO MAKE SURE NO ERROR CONDITIONS WERE SET DURING EXECUTION, THEN THE INITIAL ITERATION INDEX IS SET EQUAL TO ONE FOR THE FIRST STAGE AND 10 FOR ALL STAGES THERE AFTER. THE FINAL ITERATION INDEX IS COMPUTED, FOR WHICH THE MAXIMUM VALUE IS (NUMIT + 1) AND OCCURS IF THE PATH IS STILL ACTIVE. IF THE PATH IS ACTIVE (ISTAT = 0), THE NUMBER OF TRIPLETS BECOMES (NUMIT - 9). IF ISTAT IS GREATER THEN ZERO AND BECAME GREATER THEN ZERO IN THE CURRENT STAGE, THE NUMBER OF TRIPLETS BECOMES (ISTAT - 1). IF ISTAT BECAME NON-ZERO IN A PREVIOUS STAGE WE CONSIDER THE NEXT PATH.

AFTER CALCULATING THE NUMBER OF TRIPLETS, THE PATH NUMBER AND NUMBER OF TRIPLETS ARE OUTPUT TO THE EXTERNAL FILE. THEN THE TRIPLETS ARE OUTPUT. THE NEXT PATH IS THEN CONSIDERED, ETC.

ERROR 207 - PATH STATUS IMPROPERLY DEFINED, FATAL.

5 IF (NSTAG .EQ. 1) REWIND IPATHS WRITE (IOUT,35) 00 100 IS = 1, NUMION 140 25 IF (ISTAT(IS) = 00888) 25, 100, 95 INITIT = 1 140 25 IF (NSTAG .GT. 1) INITIT = 10 LASTIT = NIP(IS) - (NSTAG - 1) * (NUMIT - INITIT) 145 25 IF (ISTAT(IS)) 95, 50, 40 IFVAR = (((NSTAG - 1) * NUMIT) + 1 - ((NSTAG - 2) * INITIT) * NSTGML) 145 25 IF (NIP(IS) .LE. IFVAR) GO TO 100 LASTIT = ISTAT(IS) - 1 GO TO 60 150 25 LASTIT = LASTIT - 10 60 NUMTRI = LASTIT MODNTR = (NUMTRI / 3) + 1 WRITE (IPATHS) IS, MODNTR DO 90 NS = 1, NUMTRI 155 85 IF (NS .EQ. 1) GO TO 85 J = MOD (NS, 3) 155 85 IF (J .NE. 0) GO TO 90 85 WRITE (IPATHS) XION(IS,NS), ZION(IS,NS), DN(IS,NS) 90 CONTINUE 160 95 GO TO 100 95 WRITE (IOUT, 207) IS, ISTAT(IS) ISTAT(IS) = 00888 RETURN 100 CONTINUE WRITE (IOUT,36) RETURN 207 FORMAT (/,**11X, 23H****** ERROR 207 ******,**/,**11X, 1 38HPATH STATUS IMPROPERLY DEFINED (FATAL),/,**11X, 2 29HCalled FROM SUBROUTINE READER,**/,**11X, 3 6HISTAT(I4,5H ) = ,15)
FUNCTION DS (Z, X, SLOPE, LR, I)

********** BOUNDARY DISPLACEMENT ROUTINE **********

PROGRAM DESCRIPTION: PROGRAMMER = WILLIAM DEININGER, B = 26 - 01
REVISIONS: INCLUDE DATE, INITIALS AND DESCRIBE CHANGE ** PLEASE **

THIS FUNCTION SUBROUTINE FINDS THE PERPENDICULAR DISPLACEMENT FROM
THE FIRST OR LAST ACTIVE PATH TO THE BOUNDARY. CALCD CONSTRUCTS A
PERPENDICULAR TO THE PATH FROM THE CURRENT POINT (Z,X) WITH SLOPE
"SLOPE". FUNCTION DS THEN EXTRAPOLATES THIS PERPENDICULAR OF SLOPE
"SLOPE" TO THE LEFT OR RIGHT (DEPENDING ON WHETHER WE ARE CONSID -
ERING THE FIRST OR LAST ACTIVE PATH) AND FINDS THE Z AND X INTERCEPTS
(ZINT AND XINT) ALONG THE BOUNDARY LINE. ZINT AND XINT ARE CHECKED
TO SEE IF THEY LIE ON OR BETWEEN THE BOUNDARY ENDPOINTS ON THE
BOUNDARY LINE. IF THEY DO, THE DISPLACEMENT IS CALCULATED, IF THEY
DO NOT, ZINT AND XINT ARE CALCULATED ALONG THE NEXT BOUNDARY LINE
AND TESTED AGAIN. THIS CONTINUES UNTIL "GOOD" INTERSECTION POINTS
ARE FOUND OR ALL BOUNDARIES HAVE BEEN CONSIDERED IN WHICH CASE AN
ERROR MESSAGE IS OUTPUT. THE PERPENDICULAR DISPLACEMENT IS RETURNED
TO CALCD.

******* VARIABLE DICTIONARY *******

DS  : PERPENDICULAR DISPLACEMENT FROM CURRENT POINT TO BOUNDARY.
I  : PATH INDEX.
LR  : DETERMINES WHICH SIDE IS BEING CONSIDERED,
     = 1 LOOKING TO THE LEFT,
     = 2 LOOKING TO THE RIGHT.
XINT : CURRENT X POSITION ON FIRST OR LAST ACTIVE PATH (XION(I,N)).
XINT : X INTERSECTION POINT ON BOUNDARY LINE.
ZINT : CURRENT Z POSITION ON FIRST OR LAST ACTIVE PATH (ZION(I,N)).
ZINT : Z INTERSECTION POINT IN BOUNDARY LINE.

*** END OF PROGRAM DESCRIPTION AND DICTIONARY ***

PROGRAM DECLARATION STATEMENTS.
BLANK COMMON FOR LARGE ARRAYS, IO = INPUT-OUTPUT, PARAM = PARAMETERS.

COMMON ZION(41,151),XION(41,151),VELZ(151),VELX(151),
1 NIP(41),ONI(41,151),ONI(42),ISTAT(41)
COMMON / IO / INIOUT,INFO(14),KEY,ICLPLT,ICLWR,TITLE(26),
2 IPATHS,IF1(2),IF2(4),ICLERR
COMMON / PARAM / N,NUMION,NUMIT,RB,RBOUND,RT,TELOUT,BCUR,UTIL,
3 TELIN,THRL,TMSION,VELBOH,ZBOUND,IL,IP,PI,BK,RB95N,
4 DNOB,CEXSEC,THNEU,TIME,TIMEMU,XVELMU,ZVELMU,NSTAG,
5 NSTGMU,NTOTST,PIOV2

Determine whether the distance on the right or the left is desired.

IF (LR .EQ. 2) GO TO 200

Calculations for left.

-----------------------------

Test for intersections along the end of the thruster between the
beam edge (THRLN, RB) and the thruster corner (THRLN, RT). (FIRST
THE INTERSECTIONS ARE FOUND ALONG THE BOUNDARY LINE AND THEN TESTED
TO SEE IF THEY LIE ON OR BETWEEN THE BOUNDARY ENDPOINTS ON THE
BOUNDARY LINE. (EQUATION OF BOUNDARY LINE: Z = THRLEN)

100 IF (SLOPEP .LT. -1.0E+10) GO TO 110
   ZINT = THRLEN
   XINT = (THRLEN - Z) * SLOPEP + X
   IF (XINT .GE. PB AND XINT .LE. RT) GO TO 300
C
C TEST FOR INTERSECTIONS ALONG THE EDGE OF THE THRUSTER BETWEEN THE
C THRUSTER CORNER (THRLEN, RT) AND THE SPACE CRAFT WALL (O, RT).
C (EQUATION OF BOUNDARY LINE: X = RT)

110 IF (SLOPEP .EQ. 0.0) GO TO 120
   ZINT = (RT - X) / SLOPEP + Z
   XINT = RT
   IF (ZINT .GE. 0.0 AND ZINT .LE. THRLEN) GO TO 300
C
C TEST FOR INTERSECTIONS ALONG THE SPACE CRAFT SURFACE BETWEEN (O, RT)
C AND (O, RBOUND). (EQUATION OF BOUNDARY LINE: Z = 0.0)

120 IF (SLOPEP .LT. -1.0E+10) GO TO 130
   ZINT = 0.0
   XINT = X - 7 * SLOPEP
   IF (XINT .GE. RT AND XINT .LE. RBOUND) GO TO 300
C
C TEST FOR INTERSECTIONS ALONG RBOUND BETWEEN THE SPACE CRAFT WALL
C AND (ZBOUND, RBOUND). (EQUATION OF BOUNDARY LINE: X = RBOUND)

130 IF (SLOPEP .EQ. 0.0) GO TO 400
   ZINT = (RBOUND - X) / SLOPEP + Z
   XINT = RBOUND
   IF (ZINT .GE. 0.0 AND ZINT .LE. ZBOUND) GO TO 300
C
C TEST FOR INTERSECTIONS ALONG THE BEAM EDGE BETWEEN THE THRUSTER
C (THRLEN, RB) AND (ZBOUND, RB). (EQUATION OF BOUNDARY LINE: X = RB)

195 ZINT = (RB - X) / SLOPEP + Z
   XINT = RB
   IF (ZINT .GE. THRLEN AND ZINT .LE. ZBOUND) GO TO 300
   GO TO 400
C
C CALCULATIONS FOR RIGHT.

200 IF (SLOPEP .LT. -1.0E+10) GO TO 210
   ZINT = ZBOUND
   XINT = (ZBOUND - Z) * SLOPEP + X
   IF (XINT .GE. RB AND XINT .LE. RBOUND) GO TO 300
210 IF (VELTZ(I)) = 220, 400, 240
C
INTEGER DS, SORT((Z-ZINT)**2+(X-XINT)**2)
IF (DS.GT.0.25) AN TO 402
RETURN

C ERROR CONDITIONS

410 FORMAT (/11X,'ERROR 410 ***** ERROR 410 *****',/11X,
1 30HDS UNUSUALLY LARGE (NON-FATAL)/,11X,
2 34HCALLED FROM FUNCTION SUBROUTINE DS,/11X,
3 3HXL'=E9.3,5H X =E9.3,10H SLOPEP =E9.3,8H XINT =E9.3,
4 6H 7INT =E9.3,8H VELTZ(I) =E9.3,6H LR =I3)

412 FORMAT (/11X,'ERROR 412 ***** ERROR 412 *****',/11X,
1 30HDS UNUSUALLY LARGE (NON-FATAL)/,11X,
2 34HCALLED FROM FUNCTION SUBROUTINE DS,/11X,
3 4HLR =I3,5H L =I5,6H DS =E9.3,5H Z =E9.3,5H X =,
4 E9.3,10H SLOPEP =E9.3,8H XINT =E9.3,8H ZINT =E9.3)
SUBROUTINE VRSP 

--VRSP USES VERSATEC PLOTTER TO PLOT ARRAYS X() AND Y()

CWD THIS ROUTINE IS SITE DEPENDENT. IT PLOTS THE CONTENTS OF ARRAYS 
CWD XION AND ZION FROM CORE MEMORY.

C BLANK COMMON FOR LARGE ARRAYS
COMMON ZION(41,151),XION(41,151),VZ(151),VX(151)
1,NIP(41),DN(41,151),DN1(42),ISTAT(41)
COMMON / ID / IN, IOUT, INFO(14), KEY, ICLPLT, ICLWRT, ITITL(28)
1,PATH, IN, IF1, IIF2, IICLRP
COMMON / PARAM/ NNUMINT, NUMIT, RBOUND, RT, TELOUT, BMCR, UTIL,
1, TELN, THLEN, UMTON, VELBOH, ZBOUND, IL, RPI, BK, O, RBO5
C-- FIRST ENTRY --SET UP THE SYSTEM, SCALE, AXES AND TITLE IF DESIRED
INTR = INTR+1
IF (KEY.EQ.0) GOTO 3
IF (INTR.EQ.1) CALL PLOTS (O, O, O, .).

C-- SET ORIGIN OF PLOT
CALL PLOT (1., 1., -3)
CALL SETMSG(1)
SAVZ(1) = 0.
SAVX(1) = 0.

SAVZ(2) = RBOUND / ZAXLN
SAVX(2) = RBOUND / XAXLN
ZMAX = ZAXLN * SAVZ(1)
XMAX = XAXLN * SAVX(1)

C WD WORD SIZE DEPENDENT AREA; IW1, IW2 ARE FLAGS IN ITITL
IW1 = IW1 + 1
IW2 = IW1 + IW/2

CALL AXIS(0., 0., ITITL(IW1), -40, ZAXLN, 0., SAVZ(1), SAVZ(2))
CALL AXIS(0., XAXLN, 1H, 1., ZAXLN, 0., SAVZ(1), SAVZ(2))
CALL AXIS(0., 0., ITITL(IW2), 40, XAXLN, 90., SAVX(1), SAVX(2))
CALL SYMBOL(0., 0., 0., 14, ITITL(1), 0., 80)

DO 80 J=2, NUMINT
IF (ZION(J, 1), GT, ZMAX) GO TO 82
80 CONTINUE
82 NUMPT = J-1
IF (KEY.GT.0) NUMPT = NUMINT+1
DO 100 J = 1, NUMPT
NPTS = ISTAT(J)
IF (ISTAT(J), GT, 0) NPTS = NIP(J)
DO 100 NH = 1, NPTS
DZ(NM) = ZION(J, NM)
DX(NH) = XION(J, NH)
100 CONTINUE
DO 120 I = 1, 2
NPT = NPTS + I
DZ(NPT) = SAVZ(1)
DX(NPT) = SAVX(1)
IF (J, EQ, NUMINT+1) ISYM = 0
CALL LINE(DZ, DX, NPTS, INC, LINTYP, ISYM)

ORIGINAL PAGE 17
OF POOR QUALITY
150 CONTINUE
C--DRAW SCHEMATIC OF THRUSTER AND BEAM.
   DX(1)=RT
   DX(2)=RT
   DX(3)=0.
   DX(2)=THPLEN
   DX(3)=THRLEN
   DX(4)=THRLEN
   DX(4)=RB
   DX(5)=RB
   DZ(5)=ZBOUND
   NPTS=5
   DO 200 I=1,2
   NPT=NPTS+I
   DZ(NPT)=SAVZ(I)
   DX(NPT)=SAVX(I)
200 CALL LINE(D7,DX,NPTS,2,0,0)
C--FINISH THIS PLOT AND GO BACK FOR MORE
C--CALL PLOT(0.,0.,-999)
RETURN
C--TERMINATE ALL PLOTTING--RELEASE OUTPUT TO VERSATEC PLOTTER
   3 IF (INTR .LT. 2) RETURN
   CALL PLOT(O.,0.,1+999)
   INTR = 0
   RETURN
END
SUBROUTINE VRSPL
C
C--VRSPL USES VERSATEC PLOTTING TO PLOT ARRAYS X(1) AND Y(1)
C
C
C W D  THIS ROUTINE IS SITE DEPENDENT. IT PLOTS THE CONTENTS OF THE
C W D  ARRAYS XI AND ZI WHICH HAVE BEEN READ FROM THE EXTERNAL
C W D  FILE PATHS IN TERMS OF TRIPLETS.
C
C
C BLANK COMMON FOR LARGE ARRAYS
COMMON XI(41,151),XON(41,151),Y(151),V(151)
1,NIP(41),DN(41,151),DN(41),ISTAT(41)
COMMON / ID / INIOUT,INFO(14),KEY,ICLPLT,ICLWRT,ITITL(28)
1,IPATHS,IP,IF1(2),IF1(4),ICLERR
COMMON/PARAM/N,NUMION,NUMIT,RB,RBOUND,RT,TELOUT,BMCUR,UTIL,
1,T vigil,THLE,FUMSN,VELBOH,ZBOUND,IL,IR,PI,PK,Q,RB95N
1,DOM,CEXSEC,THNEU,TIME,TIMENS,VELMU,ZVELMU,HSTAG
5,NS Hou AJDTST,PIOV
DIMENSION SA(12),SA(2),A(151),XZ(151),Y(151)
COMMON /PARA1
M N
NUMI,NUMIT,RTPRBOUN,RTPELOUT,PBMCUR,UTIL,
1,NSTG,M,TENIN,VTU
SAVZ(1),SAVZ(2),SAVX(1),SAVX(2)
COMMON /PARA2
M N
NUT,NTOTST,PTO
DIMENSION SAVZSZ,SAXLSZ)
DATA ZAXLNPXAXLNPINCPLINTYP
30,SAXLNPXAXLNPINCPLINTYP
DATA INTP/0/

C W D USE INTP TO COUNT ENTRY NUMBER AND AVOID REINITIALIZING
C--FIRST ENTRY--SET UP THE SYSTEM, SCALE, AXES AND TITLE IF DESIRED
C W D 11 - 23 - 61, MUST REWIND FILE BEFORE READING
C W D 11 - 23 - 61, MUST REWIND FILE BEFORE READING
C W D 11 - 23 - 61
C
C W D--SET ORIGIN OF PLOT
CALL PLOT(1,1,1,3)
SAXZ(1)=0.
SAXZ(2)=0.927AXLN
SAXX(2)=RBBOUND/XAXLN
7MAX=ZAXLN=SAVZ(2)

35 CW WOR D--SIZE DEPENDENT AREA; IW1, IW2 ARE FLAG S IN ITITL
IW1=IW1+1
IW2=IW1+IW/2
CALL AXIS(0.0,0.,ITITL(IW1),IAMO),ZAXLN=0.,SAVZ(1),SAVZ(2))
CALL AXIS(0.,0.,ZAXLN=0.,SAVZ(1),SAVZ(2))

CALL AXIS(0.0,0.,ITITL(IW2),40,ZAXLN=0.,SAVZ(1),SAVZ(2))
CALL AXIS(0.0,0.,ITITL(IW2),40,ZAXLN=0.,SAVZ(1),SAVZ(2))
CALL SYMBOL(1,A,0.,0.14,ITITL(1),0.,80)

C W D--DC298I--MOD FOR READING, PLOTTING PATHS FILE
WRITE(IOUT,21) NUMION,NUMIT
NUMI = NUMION +1
DO 55 JS = 1, NTOTST
C
C W D THE FOLLOWING CODE COUNTS THE NUMBER OF TRAJECTORIES WRITTEN
C W D TO FILE PATHS.
C
I C O U N T = 0
IF (JS = JS-1,5,15)
50.9 IFVAR = ((JS-1) +NUMIT +1) - ((JS-2)*10)
DO 7 I=1,NUMION
IF(NIP(I)-IFVAR) 7,7,9
7 CONTINUE
9 INIT00 = I
DO 11 I = 1,NUMION
II = INI00 = I
IF(NIP(I)-IFVAR) 11,11,12
CONTINUE
LASTDO = II
DO 14 IJ = INITDO, LASTDO
   IF (ISTAT(IJ) = FREE) 14, 13, 14
13   IFVAR = (JS - NUMIT + 1) - ((JS - 1) * 10)
   IF (HIP(I) = IFVAR) 5, 6, 14
   ICOUNT = ICOUNT + 1
14 CONTINUE
LASTDO = LASTDO - INITDO - ICOUNT
GO TO 18
15 INITDO = 1
LASTDO = NUMION
CONTINUE
DO 44 J=1, LASTDO
READ(IPATHS) ION, NITER
WRITE(IOUTr2) ION, NITER
44 CONTINUE
CONTINUE
NPT = NITER + I
DO 37 NPT = SAVT(I)
37 DX(NPT) = SAVV(I)
CALL LINE (OZ, DX, NITEP, INC, LINTYP, INTEQ)
CONTINUE
CONTINUE
C- DRAW SCHEMATIC OF THRUSTFR AND BEAM.
70 DX(1) = RT
DX(2) = RT
DX(3) = 0
DX(4) = THRLEN
DX(5) = RR
DX(5) = ZBOUND
100 NPTS = 5
DO 200 I = 1, 2
NPT = NPTS + I
DO 200 DX(NPT) = SAVX(I)
105 CALL LINE (OZ, DX, NPTS, 1, 0, 0)
C- FINISH THIS PLT AND GO BACK FOR MORE
CALL PLOT(0, 0, +999)
RETURN
C- TERMINATE ALL PLOTTING-RELEASE OUTPUT TO VERSATEC PLOTTER
110 IF (INFRT .LT. 2) RETURN
CALL PLOT(0, 0, +999)
INFRT = 0
RETURN
END
SUBROUTINE PLOTW (X,Y,P,L,LAB)

C-- SUM. PLOT BY D. R. WINDER, PHYSICS DEPT., COLO. ST. UNIV.
C PLOT ARRAYS Y VS X, EACH HAVING N POINTS, SELF SCALING.

C== LB IS THE SYMBOL USED FOR THE CURRENT GRAPH--SUGGEST NUMERICAL ORDER
C== E.G., FOR THE FIRST GRAPH, SET IN CALLING PROGRAM: LB=1H1
C== THEN FOR 2ND ONE, RESET IT: LB=1H2, ETC. ONLY ONE CHARACTER, PLEASE.
C== THE FIRST ENTRY IS CRITICAL--IT ESTABLISHES NUMBER OF POINTS, ALSO
C== UPON FIRST ENTRY, MAX AND MIN VALUES ARE FOUND AND ARE USED LATER
C== IF LATER CALLS INVOLVE POINTS OUTSIDE THESE LIMITS, THEY WILL BE
C== TRUNCATED AND AN ERROR MESSAGE PRINTED ON THE PLOT FILE.
C== TO SIGNAL THE LAST GRAPH TO BE PLOTTED, PUT N=99. ALL GRAPHS WILL
C== BE PLOTTED ON ONE SHEET VIA THE ARRAY LINE(103,60). IF ONLY ONE
C== GRAPH IS DESIRED, SET N=-N IN THE CALLING PROGRAM.
C== NOTE ON SCALING: THIS EXPECTS PRINTER WITH 10 CHAR/S/INCH, 6 LINES/IN
C== LAB(16) CONTAINS TITLE (IN WORDS 1-8), X-AXIS LABEL (IN 9-12)
C== AND Y-AXIS LABEL (IN 13-16). THE LAST CALL (N=0) DETERMINES LABELS

C== CALLING PROGRAM MUST SET UP LINE(103,60) AND LAB.

C==NOTE THAT FORMATS ASSUME IND=103, LNG=60, IF OTHERWISE, ADJUST THEN
20 FORMAT(B11,PLOT,22X,8A10,,29X,8XAXIS,,4A10,11X,Y-AXIS,)
2 4A10
25 21 FORMAT(20X,1HI,10(10H...V...X),2H,I)
22 FORMAT(/7X,14PLOT--ON ENTRY,13,12H WITH SYMBOL,1X,A1,,/9X,
2 52HT THE RANGE OF VALUES EXCEEDED THAT SET ON FIRST ENTRY,,/9X,
3 31HORIZONTAL RANGES ARE: (ABSCISSA)14X,10H(DVCDINATE),/12X,
4 13HFIRST ENTRY=,4X,4A10,,/12X,12THIS ENTRY=,9X,4E11,3,,/)
30 23 FORMAT(/7X,13PLOT--ENTRY 13,8H SYMBOL=,4A10,,/14H POINTS, RANGE
2 31HS OF ABSCISSA AND ORDIINATE ARE=/29X,4E11,3)
24 FORMAT(20X,103A1)
25 FORMAT(5X,1PE13,4,2X,103A1)
26 FORMAT(13X,11(1X,1PE9.1))///)

C DIMENSION X(N),Y(N),LINE(103,60),LAB(16),ZX(11)

C== MAX-MIN SECTION--FIRST ENTRY ONLY THE RANGE IS SET FOR X AND Y
40 C
45 XSI=X(1)
YSI=Y(1)
YL1=Y(1)
DO 31 J=2,N
XSI=AMIN1(XSI,X(J))
XLI=AMAX1(XLI,X(J))
YSI=AMIN1(YSI,Y(J))
YL1=AMAX1(YL1,Y(J))
31 IF(NTR,GT.1)GOTO 34
YS=YSI
XLI=XLI
YS=YSI
YL=YL1

TOTALS: 116
GOTO 39
C--NOT THE FIRST ENTRY, SO CHECK RANGES
34 IF(XSL.LT.XS) GOTO 35
IF(XL.LT.XL) GOTO 35
IF(YS.LT.YS) GOTO 35
IF(YL.LT.YL) GOTO 35
GOTO 29
65 C--OUT OF RANGE--WRITE MESSAGE AND SET A FLAG WITH NTR
39 WRITE(IOUTI,22) NTR,LP,XS,XL,YS,YL,XSL,XL,YS1,YL1
NTR= NTR
GOTO 55
C--WRITE MESSAGE ABOUT THIS (GOOD) ENTRY
70 WRITE(IOUTI,93) NTR,LP,N,XSL,XL,YS,YL1
IF(NTR.NE.1) GOTO 55
C--DONE WITH RANGING, NOW SCALING ON FIRST ENTRY ONLY
YSCALE=(XL-YS)/(FLOAT(LWG-2))
YSCALE=(YL-YS)/(FLOAT(LNDG-1))
75 DO 70 J=1,LP
70 X(J)=XS+FLOAT(J-1)*XSCALE*10.
C--BLANK THE PLOT ARRAY LINE ON THE FIRST ENTRY
DO 33 J=1,LP
DO 33 K=1,LP
LINE(J,K)=ILNK
C--BORDERS--LEFT AND RIGHT
DO 44 J=1,LP
LINE(J,1)=IRBD
44 LINE(J,LP)=IRBD
85 CONTINUE
C--FILL IN ARRAY LINE
DO 57 I=1,NN
IX=(X(I)-XS)/XSCALE+1.5
IF(X(I).LT.XS) IX=1
IF(X(I).GT.XL) IX=LP+1
IY=(Y(I)-YS)/YSCALE+5.
IF(Y(I).LT.YS) IY=1
IF(Y(I).GT.YL) IY=LP+1
IY=LP+1
57 LINE(IY,IY)=IR
C--IF NOT THE LAST GRAPH, GO BACK FOR MORE
IF(N.GT.0) GOTO 91
C--MUST BE PLOTTING -- PRINT TITLE, AXES LABELS, AND TOP BORDER
WRITE(IOUTI,21)
WRITE(IOUTI,21)
C--PRINT Y-VALUES AND PLOT
YVAL=YL+YSCALE
DO 73 J=1,LP+2
YVAL=YVAL-YSCALE
WRITE(IOUTI,25) YVAL,LINE(J),LP,LP
YVAL=YVAL-YSCALE
73 WRITE(IOUTI,24) (LINE(J),LP,LP)
WRITE(IOUTI,26) (YVAL,L,LP)
110 C--CLEAN UP THIS MESS AND RETURN
91 NTR=IABS(NTR)
C--IF THIS IS THE LAST GRAPH, RE-INITIALIZE
IF(N.LT.1) NTR=0
RETURN
115 END

ORIGINAL PAGE IS
OF POOR QUALITY
SUBROUTINE LNPLT

CW 7/24/81 LNPLT MODIFIED FOR PLASIM; NOW NO FORMAL PARAMETERS
C LNPLT PREPARES ARRAYS X AND Y FOR PLOTTING VIA PLOTW
C N IS NUMBER OF ENTRIES IN X AND Y ON FIRST ENTRY
C N=0 SIGNALS LAST ENTRY, FOR ONLY 1 ENTRY, ENTER WITH -N
C
DIMENSION LINE(10,60),LAB(16),X(151),Z(151)

CW 7/24/81 CHANGED CONF FOR PLASIM PLOTTING.  ----STP=INCREMENTS
C
CW 7/24/81 BLANK COMMON AND LABELED COMMON FROM READER
COMMON ZION(41,151),XION(41,151),VELTX(151),VETLZ(151)

1, NIP(41),DN(41,151),DN(42),ISTAT(41)
COMMON/IO/ IN,INUT,INFO(14),KEY,ICLPLT,ICLWRIT,ITITL(28)
COMMON/IPATH,ITIMEL,ITIMEX,TIME,TIMEU,XVELMU,ZVELMU,NSTAG
COMMON/I01, INOUT,INFO(14),KEY,ICLPLT,ICLWRIT,ITITL(28)
COMMON/PARAM/NUMION,NUNIT,ND,ROUNDBOUND,RT,TELOUT,BMCUR,UTIL,
3 TELIN,THILEN,USION,VELBOH,ZBOUND,1L,IR,P,IR,1,BK,0,FB95N,
4 DNBO,CNSET,TIMAX,TMIN,TIMEX,TIMENUXVELMU,ZVELMU,NSTAG,
5 NSTAGM,NOTOST,PIOV2

DATA NTP/NSTP,ITRSTP/0,2,2/ NTR=NTR+1
IF(INTRGT0)GOTO 35

CW 7/24/81 PUT LABELS IN LAB FROM ITITL
IN2=2+IN
DO 33 J=1,INP
33 LA9(J)=ITITL(J)
CONTINUE

CW 7/24/81 SET UP DUMMY X,Z FOR PLOTW USING XION,ZION
CW 7/24/81 IF OTHER PLOTS DESIRED, CHANGE NEXT LINES
NIT=NIT+1
CW 7/28/81 AD HOC SET UP MAXMIN FOR PLOTW
DO 39 ION=1,NUMION
NIT=NIT+(NSTAG-1)*(NUMIT-10)
X(I)=XION(ION,1)
Z(I)=ZION(ION,1)
X(NIT)=X(1)
Z(NIT)=Z(1)
35 CONTINUE
DO 39 ITR=1,NITP
X(1)=A'MIN1(X(1),XION(ION,ITR))
Z(1)=AMIN2(Z(1),ZION(ION,ITR))
X(NIT)=AMAX1(X(NIT),XION(ION,ITR))
Z(NIT)=AMAX2(Z(NIT),ZION(ION,ITR))
40 CONTINUE
LB=1H
CALL PLOTW(7,X,NIT,LAB,LB,LINE,LAB)
DO 66 ION=2,NUMION,IONSTP
DO 44 J=1,NIT
44 X(J)=0.0
Z(J)=0.0
NIT=NITP(ION)
LB=SHIFT(ION+54)
DO 55 ITR=1,NIT,ITRSTP
X(ITR)=XION(ION,ITR)
55 Z(ITR)=ZION(ION,ITR)
CW 7/24/81 IF LAST ENTRY TO PLOTW, TELL IT SO WITH N=0
IF(IONGE6,NUMION)NIT=0
66 CALL PLOTW(7,X,NIT,LAB,LINE,LAB)
CW 7/24/81 FINISHED WITH ONE PLOT, RESET FOR NEXT ONE
NTR=0
RETURN

ORIGINAL PAGE IS OF POOR QUALITY
<table>
<thead>
<tr>
<th>CARD NR.</th>
<th>SEVERITY</th>
<th>DETAILS</th>
<th>DIAGNOSIS OF PROBLEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>ANSI</td>
<td></td>
<td>HOLLERITH CONSTANT APPEARS OTHER THAN IN AN ARGUMENT LIST OF A CALL STATEMENT OR IN A DATA STATEMENT.</td>
</tr>
</tbody>
</table>