

REPORT DOCUMENTATION PAGE

0076

Public reporting burden for this collection of information is estimated to average 1 hour per response, including gathering and maintaining the data needed, and completing and reviewing the collection of information. Send collection of information, including suggestions for reducing this burden, to Washington Headquarters Service, Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0100), Washington, DC 20503.

 UFGS
 of this
 person

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE		3. REPORT TYPE AND DATES COVERED FINAL REPORT 01 Aug 92 - 30 Nov 96	
4. TITLE AND SUBTITLE JOINT INDUSTRY-UNIVERSITY PROGRAM FOR COMPUTATIONAL PLASMA RESEARCH				5. FUNDING NUMBERS 61102F 2301/ES	
6. AUTHOR(S) Dr Goplen					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Mission Research Corporation 735 State Street Santa Barbara, CA 93102-0719				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NE 110 Duncan Avenue Suite B115 Bolling AFB DC 20332-8080				10. SPONSORING / MONITORING AGENCY REPORT NUMBER F49620-92-C-0052	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The primary objective of this program was to further the research goals of AFOSR in plasma physics. MRC's role in this program was to collaborate with other AFOSR-sponsored researchers in the development and application of state-of-the-art computational methods to critical research problems. The principal mechanism for this collaboration was the joint research group. The objectives of this group includes not only the performance of superior research, but also the advancement of computational research methods by focusing the combined resources and capabilities of the entire community. The joint research group was established several years ago. Charter members include five universities (Texas Tech, Univ of Michigan, UCLA, MIT, and King's College) in addition to MRC. The research initially focused on problems such as plasma guns and microwave tube research. MRC provided the other participants with codes and related materials and, as their expertise grew, participated with them in their research and analyses and in the formulation of future research and modeling requirements. At present the group has expanded to include 17 universities and 6 government agencies. The scope of research has also expanded to include new topics, including novel beam emission sources and pulsed-power switches.					
14. SUMMARY				15. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED		18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED		19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	
20. LIMITATION OF ABSTRACT					

MRC/WDC-R-380

Copy 84

MAGIC USER'S MANUAL

Bruce Goplen
Larry Ludeking
David Smithe

October 1996

Technical Report

Prepared for: Air Force Office of Scientific Research
Bolling Air Force Base
Washington, D.C. 20332-6448

Contract No.: F49620-92-C-0052

Prepared by: MISSION RESEARCH CORPORATION
8560 Cinderbed Road
Suite 700
Newington, Virginia 22122

Research sponsored by the Air Force Office of Scientific Research under Contract F49620-92-C-0052. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon.

19970203 092

TABLE OF CONTENTS

PART 1. USING MAGIC

1. INTRODUCTION.....	1-1
1.1 MANUAL OBJECTIVE	1-1
1.2 MANUAL ORGANIZATION.....	1-1
1.3 SOFTWARE DESCRIPTION.....	1-2
1.4 ENVIRONMENT	1-2
2. CREATING THE SIMULATION	2-1
2.1 BASIC CONCEPT	2-1
2.2 COMMAND CHECKLIST	2-1
2.3 IMPORTANT CONVENTIONS.....	2-3
2.4 COMMON ERRORS	2-4
3. EXECUTING THE SIMULATION	3-1
3.1 CONTROL KEYS.....	3-1
3.2 OUTPUT FILES.....	3-1
3.3 ERROR MESSAGES	3-2
3.4 CONTINUING THE SIMULATION	3-3
4. MAGIC COMMAND LANGUAGE	4-1
4.1 CHARACTER SET	4-1
4.2 CONSTANTS	4-2
4.3 VARIABLES	4-4
4.4 VARIABLE SUBSTITUTION.....	4-5
4.5 FUNCTIONS	4-6
5. INTERPRETING COMMAND SYNTAX	5-1
5.1 UPPER-CASE ARGUMENTS.....	5-2

5.2 LOWER-CASE ARGUMENTS.....	5-2
5.3 STRING ARGUMENTS	5-3
5.4 FUNCTION ARGUMENTS.....	5-4
5.5 OPTIONAL ARGUMENTS.....	5-4
5.6 ARGUMENT SELECTION.....	5-5
5.7 "WILD-CARD" ARGUMENTS	5-5
5.8 REPEATED ARGUMENTS	5-5

PART 2. MLC COMMANDS

6. VARIABLES AND FUNCTIONS	6-1
7. CONTROL STATEMENTS	7-1
8. I/O UTILITIES	8-1
9. EXECUTION CONTROL	9-1

PART 3. TIME AND SPACE

10. OBJECTS	10-1
11. GRIDS	11-1

PART 4. SPATIAL EXTENSIONS

12. OUTER BOUNDARIES.....	12-1
13. TRANSMISSION LINES	13-1

PART 5. PROPERTIES

14. BULK PROPERTIES	14-1
15. UNIQUE GEOMETRY.....	15-1
16. EMISSION PROCESSES.....	16-1

PART 6. ALGORITHMS

17. ELECTROMAGNETIC FIELDS	17-1
18. CHARGED PARTICLES	18-1
19. OTHER ALGORITHMS.....	19-1

PART 7. OUTPUT

20. OUTPUT CONTROL	20-1
21. PRINTED OUTPUT.....	21-1
22. TIME PLOTS.....	22-1
23. 1-D PLOTS	23-1
24. 2-D PLOTS	24-1
25. DATA	25-1

This page is intentionally left blank.

COMMANDS INDEX

—\$—

\$namelist\$ Command..... 7-9

—A—

AIR-CHEMISTRY Command..... 14-3
 AREA Command..... 10-8
 ASSIGN Command..... 6-3
 AUTOGRID Command..... 11-10

—B—

BLOCK / ENDBLOCK Commands..... 8-3

—C—

CALL / RETURN Commands..... 7-7
 CHARACTER Command..... 6-9
 CIRCUIT Command..... 19-13
 COMMAND Command..... 9-13
 COMMENT / C / Z / ! Commands..... 8-5
 CONDUCTANCE Command..... 14-8
 CONDUCTOR Command..... 14-9
 CONTINUITY CONSERVED..... 18-11
 CONTINUITY CORRECTED..... 18-15
 CONTINUITY LOW_T Command..... 18-18
 CONTINUITY NOT_CONSERVED..... 18-13
 CONTOUR Command..... 24-7
 COURANT Command..... 21-3

—D—

DEBUG Command..... 21-5
 DELETE Command..... 10-14
 DELIMITER Command..... 8-7
 DIAGNOSE Command..... 21-6
 DIELECTRIC Command..... 14-11
 DISPLAY Command..... 24-3
 DO / ENDDO Commands..... 7-3
 DRIVER Command..... 19-12
 DUMP Command..... 25-3
 DURATION Command..... 11-3

—E—

ECHO / NOECHO Commands..... 8-10

EIGENMODE Command..... 19-23
 EMISSION [options] Command..... 16-5
 EMISSION BEAM Command..... 16-8
 EMISSION EXPLOSIVE Command..... 16-12
 EMISSION GYRO Command..... 16-17
 EMISSION HIGH_FIELD Command..... 16-20
 EMISSION PHOTOELECTRIC..... 16-23
 EMISSION ... THERMIONIC..... 16-26
 EMIT Command..... 16-27
 ENERGY Command..... 22-24
 EXCLUDE Command..... 16-29
 EXPORT Command..... 25-6

—F—

FLUX Command..... 22-27
 FOIL Command..... 15-3
 FREESPACE Command..... 12-13
 FUNCTION Command..... 6-10

—G—

GRAPHICS Command..... 20-6
 GRID EXPLICIT Command..... 11-13
 GRID ORIGIN Command..... 11-12
 GRID PADE Command..... 11-19
 GRID QUADRATIC Command..... 11-16
 GRID SIN / COS Command..... 11-21
 GRID UNIFORM Command..... 11-14

—H—

HEADER Command..... 20-3

—I—

IF / ELSEIF / ELSE /ENDIF Commands... 7-5
 IMPORT Command..... 12-18
 INTEGER Command..... 6-7

—J—

JOIN Command..... 13-7
 JOURNAL Command..... 8-11

—K—

KEYBOARD Commands..... 9-14

—L—

LINE Command	10-5
LINPRINT Command.....	21-7
LIST Command.....	10-13
LOOKBACK Command.....	13-11
LORENTZ Command.....	18-7

—M—

MARK Command.....	11-7
MATCH Command	12-15
MATERIAL Command	14-14
MAXWELL BIASED Command.....	17-17
MAXWELL CENTERED Command.....	17-9
MAXWELL HIGH_Q Command	17-13
MAXWELL QUASI_STATIC	17-11
MESSAGE Command	9-7
MODE Command.....	17-7
MOVIE Command.....	25-9

—O—

OBSERVE AIRCHEM Command	22-10
OBSERVE CIRCUIT Command	22-11
OBSERVE ENERGY Command	22-12
OBSERVE FIELD Command.....	22-14
OBSERVE FIELDENRGY Command	22-15
OBSERVE FLUX Command.....	22-16
OBSERVE [options] Command	22-3
OBSERVE POYNTING Command	22-19
OBSERVE QMEA Command.....	22-20
OBSERVE SET_DEFAULT Command...	22-9
OBSERVE STATISTICS Command	22-21
OBSERVE STRUT Command	22-22
OBSERVE TRAMLINE Command.....	22-23
OUTGOING Command.....	12-10

—P—

PARAMETER Command.....	21-8
PARTICLES Command.....	25-12
PAUSE Command	9-8
PERSPECTIVE Command	24-16
PHASESPACE Command	24-22
PHOTON Command	16-31
POINT Command.....	10-3
POISSON Command	19-18
POLARIZER Command	15-6
POPULATE Command.....	19-5

PORT Command.....	12-6
PRESET Command.....	19-8

—R—

RANGE...AIR_CHEMISTRY Command	23-7
RANGE...FIELD Command.....	23-8
RANGE...FLUX Command.....	23-12
RANGE ... [options] Command	23-3
RANGE...PARTICLE Command	23-13
RANGE...POYNTING Command	23-15
RANGE...TRAMLINE Command.....	23-20
REAL Command.....	6-8
RECORD Command	9-9
RESET Command.....	9-10
RESTART Command	9-12

—S—

SHIM Command.....	15-8
SPECIES Command.....	18-5
START Command	9-3
STATISTICS Command	21-9
STOP Command	9-4
STRUT Command	15-10
SYMMETRY Command.....	12-3
SYSTEM Command	10-11

—T—

TAGGING Command	24-30
TERMINATE Command	9-5
TIMER Command.....	11-4
TIME_STEP Command	17-5
TRAJECTORY Command	24-27
TRAMLINE Command.....	13-3

—V—

VECTOR Command	24-12
VOLTAGE Command.....	13-12

VERSION

This User's Manual documents the October 1996 version of MAGIC. The previous version was January 1994. The following new commands have been added:

ASSIGN
POINT, LINE, AREA
AUTOGRID, MARK
GRID, GRID ORIGIN
PORT
STRUT
MAXWELL, TIME_STEP, MODE
LORENTZ, CONTINUITY
DRIVER
EIGENMODE

Some of these new commands are simply revisions of old commands (ASSIGN replaces DEFINE, MAXWELL replaces FIELDS, etc.). The new commands are much simpler, and employ defaults to encourage their use. All of the old commands are retained in the code to ensure "backward compatibility." Thus, old input decks should still run in the new version.

Other new commands reflect a fundamental change in approach. For example, this version inaugurates solid modeling in MAGIC, which means that all spatial objects are now classified either as points, lines, or areas. (Previously, a solid was defined by tracing its surface.) Time and space now are expressed in real, physical (e.g., seconds and meters), as opposed to the previous emphasis on indices. Automatic spatial gridding is available, and there are completely new physical algorithms, such as the eigenmode algorithm and the strut model.

Preface

This Manual marks a substantial departure from its predecessors. Whereas previous editions described commands in alphabetical order, the present Manual groups them according to function and purpose. The distinction is less one of content and more one of our intent.

In making this change, we are motivated by the following observation. By now, the majority of present users have no formal training in how to use MAGIC. The participants in the early seminars have graduated. Many present users rely upon the Manual and old input files for their understanding of what is available, and many of the input files we examine have not changed in years. While this is a testimonial to our "backward compatibility," it also means that our users are not availing themselves of the new commands and algorithms.

The implication is that we must concern ourselves with communicating the many improved features that are becoming available. One alternative remedy would be to disable obsolete features in the code so as to force use of the new techniques. We feel that this would be counter-productive, since many individuals are doing productive work with adequate, if older, methods. Instead, we will try to do a better job of advertising. Hopefully, the organization of the new Manual will help to make clear the choices that are now available.

1. INTRODUCTION

1.1 MANUAL OBJECTIVE

This Manual documents the most recent release of the MAGIC code. The code and Manual are usually released annually. The version (month and year of release) is imprinted on the cover of this document and on virtually all output from the code. Interim versions of the code may be released without documentation to correct errors, to permit use of new algorithms, and to allow beta testing. Users who encounter errors in the code or Manual are encouraged to communicate these to Mission Research Corporation by fax at (703) 339-6953 or e-mail at magic@mrcwdc.com.

Insofar as is possible, MAGIC is designed to be backwards compatible with the Manual. This means that features in previous Manuals will probably work in the latest version of the code. Backwards compatibility is the reason that old input files can continue to run, even though many of the commands are technically obsolete and have been dropped from the Manual. The Manual reflects the future, while the code is a mixture of old and new features.

The Manual assumes that the reader may have no prior experience with MAGIC, but that he is acquainted with computational physics methods in general and electromagnetic, particle-in-cell methods in particular. There is no attempt to motivate or even to justify the conventions and algorithms offered. Instead, the focus is upon how to use the code, and the user having previous experience with MAGIC may wish to omit all of Part 1.

1.2 MANUAL ORGANIZATION

This Manual consists of 7 Parts, and it is further subdivided into 25 Chapters. The names of the Part and Chapter are listed at the top of every page in the Manual.

Part 1 (Using MAGIC) is intended primarily for new users. It focuses on “how to” and is relatively devoid of theory and derivations. You are reading Chapter 1. Chapter 2 describes how to create a simulation. It presents most of the important choices that go into a simulation and includes cross-references to actual commands. It also lists the most important conventions and the most common user errors. Chapter 3 describes how to execute the simulation, how to interpret errors, and how to continue the simulation further in time. Chapter 4 offers an abridged discussion of the MAGIC Command Language (MCL), which includes the use of constants, variables, and functions, and provides a basis for advanced data processing methods. Chapter 5 discusses the rules of syntax, or how to interpret arguments in the Manual to write commands that work.

Parts 2 – 7 contain the actual command descriptions. The Parts and Chapters are organized by function, thus making it easy to compare the available alternatives. Each command includes `command_name`, function, syntax, arguments (definitions), description, restrictions, see also (cross-references), and examples. The Parts are as follows:

- Part 2 (MCL commands) — variables, functions, do-loops, etc.
- Part 3 (time and space) — spatial objects (points, lines, and areas) and grids
- Part 4 (spatial extensions) — boundary conditions and transmission lines
- Part 5 (properties) — conductors, dielectric, resistive properties, emission processes
- Part 6 (algorithms) — electromagnetic, current-density, and particle kinematics
- Part 7 (output) — an extensive selection of simulation output types and methods

1.3 SOFTWARE DESCRIPTION

MAGIC is a two- and one-half-dimensional, finite-difference, time-domain code for simulating plasma physics processes, i.e., those processes which involve interaction between space charge and electromagnetic fields. Beginning from a specified initial state, the code simulates a physical process as it evolves in time. The full set of Maxwell's time-dependent equations are solved to obtain electromagnetic fields. Similarly, the complete Lorentz force equation is solved to obtain relativistic particle trajectories, and the continuity equation is solved to provide current and charge densities for Maxwell's equations. This approach, commonly referred to as electromagnetic particle-in-cell (PIC), provides self-consistence, i.e., interaction between charged particles and electromagnetic fields. In addition, MAGIC has been provided with powerful algorithms to represent structural geometries, material properties, incoming and outgoing waves, particle emission processes, and so forth. As a result, MAGIC (the name derives from early applications as a MAGnetic Insulation Code) is applicable to broad classes of self-consistent plasma physics problems.¹

The MAGIC family of codes includes SOS², a fully-three-dimensional counterpart to MAGIC, and POSTER³, a general-purpose, post-processor. POSTER is designed specifically for data analysis and the production of a wide assortment of sophisticated graphics, including motion pictures and videos. It reads from the MAGIC standard database format. Because of the portability of the database, POSTER can be used for data analysis on platforms other than the one on which the simulation is performed. All codes in the MAGIC family are built on an application-independent software library. This includes the command language interpreter from which the user interface for each code is built and the DUMP utility which provides communication between the codes and the database. The use of this library speeds code development, but more importantly, it helps integrate the system and makes the codes similar to learn and use.

1.4 ENVIRONMENT

The number of computer platforms on which MAGIC has been installed is continually increasing. The principal computer platforms include: Cray, VAX, MicroVax, SUN Sparcstation, PC 486 and Pentium, IBM 6000, HP Workstation, and SGI Workstation. The

¹ B. Goplen, L. Ludeking, D. Smithe, and G. Warren, "User-Configurable MAGIC Code for Electromagnetic PIC Calculations," *Computer Physics Communications*, Vol. 87, Nos. 1 & 2, May 1995, pp. 54-86.

² B. Goplen, K. Heaney, J. McDonald, G. Warren, and R. Worl, "SOS User's Manual - Version October 1988," Mission Research Corporation Report, MRC/WDC-R-158, March 1989.

³ L. Ludeking, G. Warren, J. McDonald and B. Goplen, "POSTER User's Manual," Mission Research Corporation Report, MRC/WDC-R-239, August 1989.

operating systems include: Ultrix and VMS for the VAX, UNICOS for the Cray; and UNIX for the SUN Sparcstation. In addition, MAGIC has been installed on APOLLO, Titan, Cyber, Stardent, and Convex platforms.

NCAR, a commercially available graphics support software product, is required on workstations to obtain graphics output from MAGIC. An interface package for DISSPLA graphics is also available. The PC version of MAGIC is delivered with embedded graphic routines for screen graphics and drivers for postscript output files. MAGIC is generally delivered as a binary library, which is linked to DISSPLA or NCAR at the local site. The PC version of MAGIC is delivered as a compressed executable on a floppy disk.

This page is intentionally left blank.

2. CREATING THE SIMULATION

This Chapter explains how to create a MAGIC simulation. It includes a basic description of electromagnetic PIC, a command checklist, the most important conventions, and finally, the most common errors. Because MAGIC reads commands, `command_names` will be written in upper case to allow easy identification.

2.1 BASIC CONCEPT

MAGIC simulates the interaction between charged particles and electromagnetic fields as they evolve in time and space from some defined initial configuration. The numerical calculation uses the finite-difference method. Time and two-dimensional space are divided into finite grids. From some known initial state, time is advanced by adding a single time step. At each new value of time, Maxwell's equations are solved throughout space to advance the electromagnetic fields in time. Using these new fields, the Lorentz equation is solved to advance the momenta and coordinates of all charged particles in the simulation. The continuity equation is solved to map charge and current densities onto the grid, which will be used as sources for Maxwell's equations on the next time step. This interaction between the fields and particles provides self-consistence.

In addition to this basic framework, MAGIC offers a huge selection of coordinate systems, grids, spatial objects, boundary conditions, material properties, emission processes, numerical algorithms, output, etc. This selection exists for two reasons, first, to cover a wide range of physical phenomena and second, to provide a high level of simulation fidelity. There are important attributes associated with the choices. For example, one electromagnetic algorithm may have superior stability, while another is more accurate. No single selection is best for all applications, and creating a simulation becomes largely a matter of defining the best tradeoffs between the choices. Defaults simplify selections for the novice.

The serious user will encounter cost limitations in his quest for simulation fidelity. A reasonable objective might be to achieve the highest quality consistent with time and cost constraints. To aid in this, a cost algorithm is essential. This algorithm, which can be developed for any platform, is best arrived at by empirical fits to actual simulation cost data. It is simply an equation which expresses time (or money) costs as a function of simulation parameters such as cell number, average particle number, time steps, algorithm selected, etc. This cost algorithm will assist the user in making the tradeoffs essential to the necessary compromise.

2.2 COMMAND CHECKLIST

This checklist covers commands roughly in the order found in Parts 2 – 7. You will typically need commands from all six Parts, and they should be entered in the order listed below. There are other common-sense order requirements; e.g., a function must be defined before it can be used, and so forth.

Identification — You may provide background information to uniquely identify the simulation (HEADER, Ch. 20). This background information will be reproduced on virtually all simulation output. The default identification is blank.

Variables — You may use variables as data entries anywhere in place of constants. However, you must assign values to such variables before they are used (ASSIGN, Ch. 6).

Coordinate system — You may select a coordinate system (SYSTEM, Ch.10). The default is Cartesian, and the other choices are cylindrical, polar, and spherical. The selection will fix the identification of generalized (x1,x2) with physical (e.g., x,y or z,r) coordinates. The third coordinate (x3) is ignorable. (See Important Conventions, below.)

Spatial objects — You may create arbitrary spatial objects consisting of points, lines, and areas of diverse shapes (POINTS, LINES, and AREAS, Ch. 10). The only attribute of spatial objects is geometric. You must use other commands to assign boundary conditions and/or physical properties to such objects or to use them for any other purpose.

Spatial grid — You must construct a grid in both (x1, x2) spatial coordinates, using either automatic or manual gridding (AUTOGRID or GRID, Ch. 11). To use automatic gridding, you must mark at least some of the spatial objects (MARK, Ch. 11). To use manual gridding, you must specify the grid origin (GRID ORIGIN, Ch. 11) followed by arbitrary regions of grid. The spatial grid is the primary determinant of accuracy (the time step is generally unimportant to accuracy), and all spatial phenomena of interest must be appropriately resolved.

Field algorithm — You may replace the default electromagnetic field algorithm (MAXWELL, Ch. 17) and modes (MODE, Ch. 17) and specify the time step (TIME_STEP, Ch. 17). The default algorithms are suitable for relativistic particle simulations. The time step, spatial grid, and algorithm are the primary determinants of numerical stability. If the time step exceeds the Courant criterion, unambiguous catastrophic failure results. Numerical damping is another important property of the electromagnetic algorithm.

Particle algorithms — You may replace the default particle algorithms, which are suitable for relativistic particle simulations. You can alter the kinematics time step, the dimensions (2-D or 3-D), and the relativistic treatment used in the Lorentz algorithm (LORENTZ, Ch. 18) to calculate particle trajectories. You can alter the continuity algorithm (CONTINUITY, Ch. 18) to obtain different local charge conservation and particle noise properties.

Initial conditions — Most simulations start from trivial initial conditions: no fields or particles. However, you must supply any non-trivial initial conditions explicitly by creating particles to represent a plasma and solving the Poisson's equation for the associated electrostatic field (POPULATE, POISSON, and PRESET, Ch. 19).

Boundary conditions — You may assign outer boundaries to spatial lines on the simulation perimeter. (The perimeter shape and location is arbitrary; see Important Conventions, below.)

Most outer boundaries (SYMMETRY, PORT, OUTGOING, etc., Ch. 12) can be applied only to spatial lines. One exception (FREESPACE, Ch. 12) applies only to spatial areas.

Physical properties — You may assign physical properties to spatial objects. Bulk properties, such as infinite conductivity, permittivity, etc., can be assigned only to spatial areas (CONDUCTOR, DIELECTRIC, etc., Ch. 14). Unique (fine-scale) structural properties, such as a foil, polarizer sheath, etc., can be assigned only to spatial lines (FOIL, POLARIZER, etc., Ch. 15). You cannot assign a physical property to a point.

Emission Processes — You may assign particle emission processes to spatial areas (EMIT, Ch. 16). However, you must first specify the process and parameters (EMISSION ... FIELD, THERMIONIC, etc., Ch. 16).

Output — You must explicitly specify any output required (various commands, Ch. 20–25). A timer is used to trigger output at different times during the simulation (TIMER, Ch. 11).

Time — You must specify the time period to be covered by the simulation (DURATION, Ch. 11).

Finally — You must terminate the input file to execute the simulation (START, Ch. 9) or to stop processing (STOP, Ch. 9).

2.3 IMPORTANT CONVENTIONS

Units — Input and output for MAGIC uses the MKSA system of units. The only exception is generalized particle momentum, which omits rest mass from the definition. In general, the correct physical units are stated in the command argument definitions and in all printed and plotted output. You may enter constants using other units (ASSIGN, Ch. 6); however, they will automatically be converted to MKSA values as they are processed.

Generalized coordinates — For simplicity, the code uses generalized coordinates (x_1, x_2) to describe two-dimensional space. The convention identifying generalized coordinates with physical coordinates is given in the table which follows.

System	(x_1, x_2, x_3)
Cartesian	(x, y, z)
cylindrical	(z, r, θ)
polar	(r, θ, z)
spherical	(r, θ, ϕ)

The third coordinate (x_3) always represents a coordinate of symmetry; that is, physical variations depend only on x_1 and x_2 . The third coordinate is ignorable, and need not be defined by the user. However, spatial grids are required for the first two coordinates. The particular identification of generalized coordinates with physical coordinates is based upon cyclic permutation. It is simple to remember if one starts with a right-hand system, e.g., (r, θ, z),

and cyclically permutes it until the ignorable coordinate falls in the third position, e.g., (z, r, θ) , where θ is ignorable. The “length” of the ignorable coordinate is one meter in Cartesian and polar coordinates and 2π radians in cylindrical and spherical. All variable signs, e.g., for B_θ where θ is ignorable, are correctly given by the right-hand rule. Also, we refer to field components and other variables by their generalized coordinate, e.g., B_3 rather than B_θ .

Contiguous perimeter — The outer perimeter of the active simulation area must be contiguous, i.e., the area must be completely enclosed. The perimeter must consist only of areas assigned as conductors and outer boundaries, such as symmetries. However, it can use any combination of these, and it can be of any shape; i.e., it does not need to follow the extremes of the grid. The critical issue is that the perimeter not contain any “holes.”

2.4 COMMON ERRORS

Some of the most commonly experienced errors are listed below. The severity of the result can vary substantially. Some errors (e.g., stability) result in catastrophic failure, while others simply diminish accuracy and are hard to recognize.

Failure to read the LOG file — Execution (see Chapter 3) always results in a LOG file containing all error messages (flagged with ***). Since not all errors cause the code to stop, searching the LOG file for errors is essential to simulation integrity.

Keyword duplication — Variable names which duplicate command names or previously defined function names can cause serious errors. In assigning variables, it is best to choose unusual variable names to minimize the possibility of duplication.

Contiguous perimeter violation — The outer perimeter of the active simulation area must be contiguous, i.e., the area must be completely enclosed. The critical issue is that the perimeter not contain any “holes,” which could give wholly invalid results without necessarily producing an error signature. The code does not check this perimeter; it is entirely the responsibility of the user. However, you can use the DISPLAY PERIMETER command to inspect the perimeter visually (DISPLAY, Ch. 24).

Courant violations — There are several limitations on the size of the time step. The electromagnetic (Maxwell) algorithm fails catastrophically at 100% of Courant. You will know if this happens, since the exponential growth usually causes a system overflow. In certain coordinate systems and grids, failure may occur as low as 82% of Courant. Particles which travel more than one cell size in one time step will decouple the particle integer and coordinate space. This can show up as an unexpected acceleration and disappearance, and you may not notice it. In some boundary conditions (PORT and OUTGOING, Ch. 12), the time step must be less than the cell width divided by the phase velocity. These violations are more common where the choice of an implicit electromagnetic algorithm allows a larger time step. Some violations (but not all) are detected by the code and produce error messages.

Resolution errors — Inaccuracy can result if the cell-to-cell variation in any axis exceeds 25%, or the aspect ratio of the unit cell exceeds five to one. In addition, physical phenomena of interest must be adequately resolved by the time step and the spatial grid.

Outer boundary location errors — Applying an outer boundary condition too near a structural singularity, e.g., a corner or protrusion, can result in artificial scattering and even instability. Generally, outer boundaries expect the outgoing wave to be well behaved, and they do not handle fringe fields well. The best approach is to apply the outer boundary at the end of a channel of some length, and to avoid abrupt discontinuities in structure near the boundary.

Failure to test — This error results from the construction of an extremely large or complicated simulation without testing any of the models and features employed. Because of nonlinearity, it is difficult to recognize even the most basic errors in a complex simulation. Therefore, it is good practice to test simulation components under idealized conditions.

This page is intentionally left blank.

3. EXECUTING THE SIMULATION

MAGIC can be run in interactive or batch modes. To run interactively, you first start MAGIC and enter blanks for the input and output files. You then enter the commands manually, one at a time. To run batch, you first create a file (or edit an existing file) named `file_name.MGC` which contains all the required commands (as described in Chapter 2). You then start MAGIC and enter `file_name.MGC`. (File_name is an arbitrary name that you choose for the MGC file; it will automatically be used to name all output files, as well.)

This Chapter describes how to interact with the simulation during execution, what to do with the output files, how to interpret error messages, and how to continue the simulation for a longer period of time.

3.1 CONTROL KEYS

During execution, you may interact with the simulation through the keyboard (KEYBOARD, Ch. 9). You press the f6-key to send graphical output to the monitor (screen mode) or the Alt-f6-key to send it to the PS graphics file (printer mode). (The output mode can also be set by commands (GRAPHICS SCREEN and GRAPHICS PRINTER, Ch. 20).) Pressing other designated keys will override a command time-trigger (TIMER, Ch. 11) and create output precisely at the current time step. This allows you to examine results from a simulation in progress, rather than waiting for completion. The designated keys and their functions are as follows:

- Esc — terminates simulation being executed.
- f1 — creates new record in REC file (see below) and continues
- f2 — creates new REC file record and RST file and terminates
- f3 — turns PAUSE ON for graphics (screen mode only)
- f4 — turns PAUSE OFF for graphics (screen mode only)
- f5 — writes all time history plots up to current time step
- f6 — turns screen mode on and printer mode off
- Alt-f6 — turns printer mode on and screen mode off
- f7 — writes all time history plots and terminates simulation
- f8 — writes all phase-space plots
- f9 — writes all contour plots
- f10 — writes all perspective plots
- f11 — writes all range plots
- f12 — writes all vector plots

In addition to the designated control keys, you can use an intrinsic function named LASTKEY in commands to provide customized interactive capability (FUNCTION, Ch. 6).

3.2 OUTPUT FILES

Depending upon the commands in the MGC file, a single execution may produce as many as nine output files:

file_name.LOG — “log” of processed commands, error messages, and PRINT data.
file_name.SUM — “summary” of designated simulation variables and final values.
file_name.PS — “postscript” printer file.
file_name.TOC — “table-of-contents” to post-process FLD, GRD, and PAR files.
file_name.FLD — “field” data for post-processing.
file_name.GRD — “grid” and miscellaneous data for post-processing.
file_name.PAR — “particle” data for post-processing.
file_name.REC — “record” of the simulation to allow continuation.
file_name.RST — “restart” to call REC and initiate the continuation.

Each execution always produces a LOG file and a SUM file. The LOG file contains all processed commands; each data entry is written and analyzed on a separate line. Error messages are flagged with ***, to allow a quick search for errors. PRINT options in various commands may also write output to this file as the simulation progresses (STATISTICS, LINPRINT, etc., Ch. 21). Finally, basic simulation parameters, such as the number of cells, etc., are recorded. The SUM file records the final values of any simulation variables which have been specifically designated as simulation parameters (PARAMETER, Ch. 21).

The other files are produced only if triggered by certain commands in the MGC file. The PS printer file is produced if a graphical output command is executed (various output commands, Ch. 22–24) while the system is in printer mode (see Control Keys, above). The type of printer file depends on the operating system. On the PC generating POSTSCRIPT commands, the file is called PS. On UNIX workstations using NCAR-GKS graphics, the file is called GMETA. Workstations typically provide a post-processor which allows the user to view plots or to route them to a printer.

The TOC, FLD, GRD, and PAR files are produced only if post-processing is anticipated (DUMP, Ch. 25). Basically, any data that can be plotted directly can also be routed to one of these files. After the simulation, the files can be read into POSTER for processing and display, including motion pictures. The REC and RST files are both produced if the possibility of continuing the simulation for a longer period of time is anticipated (RECORD, Ch. 9). How to continue the simulation is discussed more fully below.

3.3 ERROR MESSAGES

MAGIC tests individual commands as well as simulation consistency. When individual commands are read from the MGC file, the following errors can be detected.

invalid command_name — misspelling or abbreviating the command_name
invalid key words — misspelling (abbreviations are allowed within the command)
overused commands — exceeding the maximum number of repetitions of a command
incorrect number of data entries — providing too few or too many data entries
incorrect data type — entering character data for integer or real data, or vice versa
out-of-range number — exceeding the platform word accuracy

undefined variable — trying to use a variable before assigning it a value
undefined function — trying to use a function before defining it

If no command errors are detected, MAGIC next looks at the simulation as a whole for the following errors:

spatial grid arrays (number of 1-D cells) exceeded
field arrays (number of 2-D cells) exceeded
Courant stability criterion (time step) in electromagnetic algorithm exceeded
Courant stability criterion (time step) in boundary condition exceeded
particle kinematics (2- or 3-D) inconsistent with mode (TE or TM) or applied fields

If errors of any type are found, they will be flagged with *** in the LOG file, and the simulation will be terminated. You may also specify other error-related conditions for termination (TERMINATE, Ch. 9). However, if no errors are detected, the simulation will continue.

MAGIC does not detect all errors. There are many desirable but unanticipated uses which are not safe, because they have never been tested. There are also common errors that are easy to warn against but difficult to test for. Chapter 2 includes a list of some of these. For now, the primary responsibility for error detection must rest with the user. Most realistic simulations contain conflicting processes and nonlinear interactions and are so complex that it is very difficult to recognize errors, much less fix them. Therefore, we recommend that you build a simulation in parts and test each part extensively under idealized conditions.

3.4 CONTINUING THE SIMULATION

A simulation may be continued in separate executions, *ad infinitum*. Perhaps the hoped-for effect did not materialize in the original simulation, or perhaps it did materialize, but you didn't specify adequate output. Continuation allows you to run out farther in time without repeating the original simulation.

In any simulation, you may elect to write an REC file at periodic intervals (RECORD, Ch. 9). The REC file contains the complete state of the simulation at a point in time. Only the most recent record is preserved, and any earlier records will be overwritten. On termination, an RST file is automatically produced. The RST file contains a tailor-made set of commands to continue the simulation, should you so desire.

To continue, you simply do minor editing to the RST file, change the file extension name from RST to MGC, and execute it. Commands in RST will call the REC file and initialize the simulation, beginning from the last recorded state. Commands in RST act as though they are appended to the original MGC file; they either supersede, change, or add to the original commands. Please be aware that there are limits to what you can change. For example, you cannot add a new boundary condition, or new spatial objects, or change the spatial grid. Basically, you are limited to adding new graphical output.

This page is intentionally left blank.

4. MAGIC COMMAND LANGUAGE

Input data for the MAGIC code is based upon the MAGIC Command Language (MCL). MCL is a sophisticated scripting language with many powerful features. This Chapter presents an abridged description of these features, which will be sufficient for the vast majority of applications. It emphasizes the importance of variables in creating input data. Users who require or desire to explore the more advanced features of MCL are referred to the POSTER User's Manual⁴ and/or advised to contact the authors.

A typical MAGIC command in an input file has the form

```
command_name data_entry data_entry ... ;
```

This Chapter describes the capabilities of MCL to read data. Most of these capabilities can be exercised using commands found in Chapter 6 (Variables and Functions).

4.1 CHARACTER SET

The character set which MCL can read is based upon FORTRAN. In general, individual data entries may contain upper- and lower-case letters (MCL is case-insensitive),

```
A B C D E F G ... Z a b c d e f g ... z
```

digits,

```
0 1 2 3 4 5 6 7 8 9
```

and the eighteen special characters,

```
. % & < > ? _ ~ ` @ # ^ \ | { } [ ]
```

The other special characters are reserved and have special meaning within MCL. For example, the following seven characters are reserved for mathematical expression operations (the asterisk is also used for "wild-card" operations).

=	variable and function definition
+	addition
-	subtraction
*	multiplication (*) or exponentiation (**)
/	division
()	mathematical and dummy argument grouping (always in pairs)

⁴ L. Ludeking, Gary Warren, and Bruce Goplen, "POSTER User's Manual," Mission Research Corporation Report, MRC/WDC-R-245, August, 1993.

The rest of the special characters are reserved for use as delimiters. For example, the command illustrated at the beginning of this chapter is terminated (delimited) with a semi-colon, and the individual data entries are separated (delimited) with blanks (commas are equivalent). A complete list of the MCL delimiters follows.

;	command termination delimiter
blank	data entry delimiter (interchangeable with comma)
,	data entry delimiter (interchangeable with blank)
“”	character string delimiters (always in pairs)
‘’	variable substitution delimiters (always in pairs)
:	format delimiter (when used inside variable substitution delimiters)
!	comment delimiter
\$	namelist delimiter

It is possible to re-define some, but not all, of the delimiter characters. For example, if an existing namelist file uses & as a delimiter instead of \$, the MCL delimiter can simply be re-defined in order to read the data.

4.2 CONSTANTS

MCL can read constants (literal data) of the following type: integer, real, character, and character string. The resulting word length and accuracy are machine-dependent.

An integer constant consists of one or more digits preceded by an optional sign (+ or -). If a real constant is entered for an integer, it will be truncated to integer value. Some integer constant examples are

+100 100 -9999

A real constant contains one or more digits with a decimal point located anywhere. The digits may be preceded by a sign (+ or -) and followed by an exponent letter (e or E) and more digits representing an exponent (which may also be signed). In addition, a real constant may be appended with an optional metric prefix (pico, nano, etc.) and a physical unit (inch, foot, etc.). (The constant, metric prefix, and physical unit may be separated by underscore (_) characters, if desired.) Table 4.1 lists all metric prefixes and physical units which are recognized by MCL, and the internal conversion factors. Upon reading appended units, MCL automatically converts the constant to the proper MKSA value. Some examples of real constants are

0.000001 1.0e-6 1micron 1_micro_meter 1micrometer 1e-3millim

Each of these data entries will be converted to 10^{-6} (meters) as it is read by MCL. (They are all equal.) Note that only those prefixes and units listed in Table 4.1 will be recognized by MCL.

Table 4.1. Prefixes and units with conversion values.

Prefixes

Prefix	Factor	Prefix	Factor	Prefix	Factor	Prefix	Factor
atto	10^{-18}	micro	10^{-6}	kilo	10^{+3}	eta	10^{+15}
femto	10^{-15}	milli	10^{-3}	mega	10^{+6}	exa	10^{+18}
pico	10^{-12}	centi	10^{-2}	giga	10^{+9}		
nano	10^{-9}	deci	10^{-1}	tera	10^{+12}		

Physical constants

Unit	Factor
sec	1
second	1
c	299792458
rad	1
radian	1
pi	3.14159265359
deg	pi/180
degree	pi/180

Length

Unit	Factor
mil	2.54×10^{-5}
mils	2.54×10^{-5}
inch	2.54×10^{-2}
foot	.0254x12
feet	.0254x12
micron	10^{-6}
mm	10^{-3}
cm	10^{-2}
m	1
meter	1
km	10^{+3}

Electromagnetic

Unit	Factor
a	1
amp	1
volt	1
kv	$1 \times 10^{+3}$
tesla	1
gauss	1×10^{-4}

Energy

Unit	Factor
ev	$1.6021892 \times 10^{-19}$
kev	$1.6021892 \times 10^{-16}$
mev	$1.6021892 \times 10^{-13}$
gev	$1.6021892 \times 10^{-10}$
joule	1
watt	1

Frequency

Unit	Factor
hertz	1
hz	1
khz	$1 \times 10^{+3}$
mhz	$1 \times 10^{+6}$
ghz	$1 \times 10^{+9}$
thz	$1 \times 10^{+12}$

A character constant consists of an uninterrupted string of up to 32 characters from the character set described above. The constant must begin with an alphabetical letter (A – Z) and can contain letters, digits, and special characters (but not delimiters or mathematical expression operators). Almost all of the command names (TITLE, STOP, etc.) are character constants. Other examples include

```
alfa Charlie i293[x]
```

The last data entry should not be confused with a function; it is simply a character constant.

A character string consists of a number of character constants (including embedded blanks and special characters) delimited by double quotation marks. The allowable length of a character string is machine-dependent. The string may contain any of the special characters (including delimiters and expression operators) except for the string delimiter itself. The case of any letters within the string will be preserved. If it contains a variable substitution delimiter, then the indicated substitution (see Variable Substitution, below) will be performed as the string is read. Typical uses of character strings include file names, titles, and axis labels. An example of a character string is

```
“Double-gap klystron design #07”
```

4.3 VARIABLES

Variables may be used in place of constants wherever a data entry is required. Their use is advocated for a number of reasons. First, it allows input data to be self documenting. Although MCL supports an extensive commenting capability, variables provide meaning to the data directly. Secondly, fundamental constants and parameters can be defined once and used many places in an input file. It is much simpler (and safer) to define $\pi = 3.14159$ and to use π wherever it is needed than it is to repeat a long string of digits. Third, the practice results in input files which are re-usable. By defining basic parameters (lengths, frequencies, etc.) as variables, one can quickly change data and run a new case without searching the entire file for constants.

MCL can read integer, real, and character variables. The type is usually determined by the variable name. If the variable begins with the letters i – n, it is an integer variable. Otherwise, it is real. To circumvent this convention, or to define character variables, one simply uses type-declaration commands (see Chapter 6). Examples of integer, real, and character variables are

```
Index = 999 ; pi = 3.14158 ; character alfa ; alfa = “bravo” ;
```

Note that the character variable (alfa) has been declared as a type character prior to its definition; otherwise, “bravo” would be read as a real variable.

In addition to user-defined variables, MCL supports a number of system variables which are internally defined but can be accessed and used in the input file. System variables all contain the prefix ISYS\$ (for integer variables) or SYS\$ (for real variables).

4.4 VARIABLE SUBSTITUTION

Variable substitution is a very powerful MCL feature which provides a capability similar to that of arrays. The default delimiter for variable substitution is the single quotation mark. When MCL encounters these delimiters, it substitutes the current value of the enclosed variable. The following example

```
do i = 1, 3 ;  
    x'i' = i ;  
enddo ;
```

will produce three real variables named x1, x2, and x3, with real values, 1, 2, and 3, respectively. Although the example uses an integer variable (i), real and character variables can also be substituted.

It is also possible to specify the substitution format. The formats available for the three data types are

```
type integer - In.m  
type real - Fn.m, En.m, and Gn.m  
type character - An.m
```

The standard FORTRAN rules apply for integer and real variables. For character variables, n is the number of characters to substitute, and m is the position in the existing character. As a simple example, if the statement in the block above is changed to read

```
x'i:I2' = i ;
```

then the resulting variable names would be x01, x02, and x03.

Variable substitution is also the only means of performing character string concatenation. In the following example, a substring is extracted and used to create a new character variable

```
CHARACTER FULLNAME LASTNAME ;  
FULLNAME = "David Jones" ;  
LASTNAME = "'FULLNAME:A5.7'" ;
```

The character variable LASTNAME will contain "Jones."

4.5 FUNCTIONS

Many commands require specification of a function. MCL can read a general mathematical expression (which depends on dummy arguments). The name of the function can then be entered into the command which requires the function. MCL supports the standard Fortran intrinsic mathematical functions. These, and any previously user-defined constants, variables, and functions can be included in the expression.

An example of a function command is

```
FUNCTION V(t) = sin ( omega * t ) ;
```

where omega has previously been defined as a real variable. Once this function has been entered, it can be applied wherever required simply with the data entry

V

Note that the data entry does not include parentheses or the dummy argument(s).

5. INTERPRETING COMMAND SYNTAX

This Manual describes commands which use the following form of syntax:

COMMAND_NAME argument argument ... ;

This chapter will explain how to interpret the syntax to write commands successfully. The basic rules are very simple.

- Wherever the command syntax offers an argument, you should replace the argument with a data entry. (Conceptually, this is equivalent to assigning a value to the argument.)
- Upper-case arguments indicate keywords, and the data entry should duplicate the argument exactly. (The command name is the pre-eminent example.)
- Lower-case arguments should be replaced with data entries or your choice, using integer, real, or character data, which may be either constant or variable.
- Double quotation marks “ ” enclosing an argument require that a character string be entered. (The quotation marks must be included in the input data.)
- Parentheses () embedded within an argument require that the name of a previously defined function be entered. (The parentheses themselves are not entered.)
- Brackets [] enclose optional arguments. (The brackets themselves are not entered.)
- Braces { } enclose a list of arguments requiring selection; only one data entry is made in response. (The braces themselves are not entered.)
- An asterisk argument * permits an asterisk “wild card” entry, which allows several options to be selected with a single data entry.
- Ellipsis ... indicate that more data of the preceding type may be entered.

For most users, adherence to these basic rules will suffice, and you may wish to skip the rest of this chapter. Naturally, there are variations on the basic rules which allow greater flexibility in creating input data. The remainder of the chapter will discuss some of these variations and present some examples.

5.1 UPPER-CASE ARGUMENTS

Any argument in upper case represents a keyword. This requires that the data entry duplicate the spelling of the argument exactly. Since MCL is case-insensitive, lower-case input data is acceptable, so long as the spelling is correct. All command names must be written out in full (not truncated). For example, if the command syntax reads

```
STOP ;  
  
then  
    STOP ;  
or  
    Stop ;
```

or any similar variation will be acceptable, but the name must be written out in full because STOP is a command name (the first argument in a command).

For upper-case arguments offering options or selection within the command, truncation of the data entry is acceptable. However, the data entry must duplicate enough of one argument to differentiate it from the other possibilities. For example, if the command syntax contains the arguments

```
{ YES, NO },  
  
then you can select the YES option by supplying the data entry  
  
Y
```

because it can be differentiated from the other possibility (N or NO) .

5.2 LOWER-CASE ARGUMENTS

A lower-case argument usually requires a specific type of input data. Generally, the type (integer, real, or character) will be indicated by the first letter of the argument or by the meaning which it conveys. (Functions and strings will be discussed separately.)

If the first letter of an argument is i – n, then an integer is usually required. (The integer can be either an integer constant or an integer variable.) For example, if the command syntax contains the argument

```
index_limit,  
  
where both first letter and meaning suggest an integer, then a legal data entry would be the  
integer constant  
  
1000 .
```

Or, you can first place an implicit ASSIGN command (see Variables and Functions Chapter) in the input file to create an integer variable,

```
LIMIT = 1000 .
```

Then you can enter the integer variable (LIMIT) instead of the integer constant (1000). Note that a variable must be assigned a value before it can be used, i. e., the ASSIGN command must come first in the input file.

If the first letter of the argument is not i – n, then real data is usually required. (Again, real data can be either a constant or a variable.) For example, if the command syntax contains the argument,

```
angle,
```

suggesting a real value, then the real constant,

```
3.14159,
```

would provide a legal response. Equivalently, you can define a real variable and enter the variable instead of the constant.

The argument context may indicate that character data is required. (Again, the character data can be either a constant or a variable.) For example, if the command syntax contains the argument,

```
object_name,
```

then it is clear that character (rather than real) input data is required. A legal response is the character constant,

```
Line_A.
```

Equivalently, you can define a character variable to be equal to Line_A and then enter the character variable as input data.

5.3 STRING ARGUMENTS

Double quotation marks “ ” enclosing an argument require that a character string be entered in the input data. The entry must include the quotation marks as delimiters. An example is the HEADER command which encloses the argument, remarks, in double quotes,

```
HEADER REMARKS “remarks” ;
```

This is clearly a requirement for a character string which will provide background information on the simulation. A legal response would be

HEADER REMARKS "TWT - 1099, Mk II, 3/1/96";

Note that the double quotation marks are included as part of the data entry. This character string with embedded blanks and special characters will be preserved exactly as it appears in the command. Other applications requiring character strings include the COMMENT command and certain file specifications.

5.4 FUNCTION ARGUMENTS

Parentheses () embedded within an argument require that the name of a function be entered. Before you can enter this name, you must first create the function by placing a FUNCTION command (see Variables and Functions Chapter) in the input data file. For example, if the command syntax contains the argument,

voltage(time),

the embedded parentheses indicate that a function is needed. (From the argument meaning, you might surmise that you are being asked for a voltage as a function of time.) Let us assume that you choose a sinusoidal dependence with an amplitude of 100 V and a frequency of 10 GHz. You place in the input file a FUNCTION command,

FUNCTION V(t) = 100. * SIN (2 * 3.14159 * 1E10 * t) ;

(A function expression will accept constants, variables, mathematical operators, intrinsic functions, and even functions defined in previous commands.) Now, you can simply enter the name of the function (V) in the original command. Note that the data entry is V and not V(t); the dummy argument and parentheses are not entered.

For certain arguments which call for a function, it may be acceptable to enter a constant or a variable instead of a function. (This is equivalent to defining a function equal to a constant, but more convenient.) Those cases where constant data is acceptable will be explicitly identified in the argument description.

5.5 OPTIONAL ARGUMENTS

Brackets [] are used to enclose optional argument(s). For example, if the command syntax contains the argument,

[FFT],

then legal data entries include either FFT or nothing (blank). If the brackets enclose a number of arguments, the same number of data entries (or none) must be entered. Note that the brackets themselves are not entered in the input data.

5.6 ARGUMENT SELECTION

Braces { } are used to enclose a list of arguments and to mandate a selection. In response, you must make one (and only one) data entry. For example, if the command syntax contains the arguments,

{ A, B, C },

then legal responses include only A, B, or C. No other (e. g., a blank) response is allowed, because you must select from the enclosed list. The braces themselves are not entered.

5.7 “WILD-CARD” ARGUMENTS

A “wild-card” argument (*) allows you to select more than one option with a single data entry. During processing, the asterisk is replaced with all possible variations allowed in the selection. If an argument within the bracket or braces contains an asterisk (*), then wild-card entry is allowed. For example, if a command syntax includes

{ TE, TM, * }

then legal entries include TE, TM, or *. The * entry provides the means to select both TE and TM. Note that wild-card entry is supported only where explicitly stated in the syntax.

5.8 REPEATED ARGUMENTS

Ellipsis ... indicate that more data of the type suggested by the preceding argument(s) may be entered. For example, if the command syntax contains

index ...

then several integers (but at least one) should be entered. In general, the argument grouping which precedes the ellipsis defines the scope of the data which is to be repeated. For example, if the command syntax contains the arguments,

x, y ...

then it is clear that one or more pairs of data should be entered. By contrast, the command syntax

x y ...

would suggest a single data entry for x followed by one or more data entries for y.

This page is intentionally left blank.

6. VARIABLES AND FUNCTIONS

This Chapter covers the following commands:

ASSIGN
INTEGER
REAL
CHARACTER
FUNCTION

You use these commands to create variables and functions. Once created, a variable may be used as a data entry anywhere in place of a constant. A function may be referenced simply by entering its name.

Variables are of type integer, real, or character, and the first occurrence of a variable in the input file irrevocably determines its type. You can declare it explicitly using a type declaration command (INTEGER, REAL, or CHARACTER), or it will be declared implicitly the first time that you assign a value to the variable. Either way, once the type has been declared, it cannot be altered by any subsequent command. Trying to change the type explicitly will produce an error. Trying to change it implicitly may alter the assigned value or produce an error.

The ASSIGN command (previously known as DEFINE) is used to assign a value to a variable (variable = value). The value may be changed at will, consistent with the original type declaration. ASSIGN provides implicit type declaration. If the variable name begins with i – n, it is type integer. Otherwise, it is type real. But if the assigned value is enclosed in double quotation marks, then the variable is type character. The ASSIGN command also supports pseudo-array (multi-variable) capabilities and the optional specification of physical units.

The FUNCTION command is used to create functions out of mathematical expressions or numerical data. Virtually any expression which is legal in FORTRAN can be entered with this command. In addition to constants, variables, and intrinsic mathematical functions, this command will accept any function previously entered in the input file. It will even accept time-history data and user instructions during the simulation, thus providing feedback loop capability. The FUNCTION command also supports true vector (multi-value) capability.

This page is intentionally left blank.

ASSIGN Command

Function: Assigns a value to a variable.

Syntax:

```
[ASSIGN] variable_name = expression, ... ;  
[ASSIGN] variable_name = "string", ... ;  
[ASSIGN] variable_name, ... ;
```

Arguments:

variable_name	-	name of integer, real, or character variable.
expression	-	arithmetic expression in the FORTRAN style containing constants, variables, intrinsic functions, user-specified functions, the mathematical operators add (+), subtract (-), multiply (*), divide (/), and exponentiate (**), and the Boolean operators .GT., .LT., .LE., .GE., .EQ., and .NE.
string	-	a character string (with double-quotes), or the name of a previously defined character variable (without quotes).

Description:

The ASSIGN command (previously known as DEFINE) assigns a constant value to a variable. It can also be used to display the present value of a previously assigned variable. As indicated by the syntax brackets, entry of the keyword, ASSIGN, is optional. If it is omitted, we refer to this as an "implicit" ASSIGN command.

The ASSIGN command can be used with integer, real, and character variables. Unless the type is declared explicitly in a type declaration command, it is declared implicitly in the first ASSIGN command. The implicit rules involve the variable_name and the assigned value. The first character in variable_name must be a letter. If the letter is i – n, the variable is type integer. If the letter is a – h or o – z, the variable is type real. But if the assigned value is enclosed by double quotes, the variable is type character. To override these rules, you must use one of the type declaration commands. Once the variable type is declared, it can never be changed. However, the value assigned to the variable can be changed at will, using more ASSIGN commands.

An integer or real variable is assigned the value resulting from evaluation of an expression. The expression may include constants, previously assigned variables, basic FORTRAN and Boolean operators, intrinsic mathematical functions, and user-defined functions (see FUNCTION command). The ASSIGN command converts the resulting value from real to integer and vice versa, as appropriate for the variable type. ASSIGN also allows specification of physical units, which may be appended directly onto the value or separated from it by an underscore character. (When physical units are included in any data entry, the numerical value will automatically be converted to the equivalent MKSA value.)

ASSIGN Command

For a character variable, double quotes around a string are required to preserve letter case, blanks, and other special characters in the string. If the value (string) is another character variable, then the double quotes are not used; however, both variables must be previously declared to be type character. The length of the string is the total number of characters between the quotes, including blanks. The maximum allowable length is 132 characters.

The ASSIGN command also supports a pseudo-array (multi-variable) capability. As the syntax indicates, it is possible to enter more than one expression, but only one variable_name. What happens in this event is that MCL automatically creates additional variables by appending an underscore and increasing integers to variable_name (see Examples, below). Note that this is not a true array, since the variables all have different names. The pseudo-array capability is available for all variable types. Commas are required to separate the multiple data entries (spaces cannot be used as delimiters in this command).

Once a variable has been assigned a value, the variable can be used as a data entry anywhere in place of a constant. There are some restrictions. For example, if the selected variable_name duplicates an internal character constant (any keyword, such as a command name), then it replaces that constant. Thus, it is possible to inadvertently redefine a keyword. (FORTRAN behaves in a similar manner; e.g., if you define a function named SIN, it replaces the intrinsic trigonometric function.) Also, real and integer variables should never be used as data entries where a character string is expected, and vice versa. Furthermore, variable names should never duplicate valid character string data entries, since this will result in ambiguous interpretation.

Restrictions:

1. A variable must be assigned a value before it is used.
2. Type declaration is implicit in an ASSIGN command, and it is explicit in type declaration commands. Type declaration (implicit or explicit) occurs wherever variable_name first appears in the input file.
3. The variable_name should not duplicate any character constant in the input file. For example, DATA cannot be used for a variable_name if the DATA option in a FUNCTION command is used.
4. There is no string concatenation operator; instead, variable substitution is used to assemble strings (see Examples, below).

See Also:

INTEGER, Ch. 6
REAL, Ch. 6
CHARACTER, Ch. 6
FUNCTION, Ch. 6

ASSIGN Command**Examples:**

1. The following two ASSIGN commands are equivalent.

```
ASSIGN c = 2.993e8 ;    ! This is a conventional ASSIGN command.
c = 2.993e8 ;          ! This one is an implicit ASSIGN command.
```

2. The following two commands with appended physical units are also equivalent.

```
x = 10cm ;      x = 10_cm ;
```

Both data entries will be converted to 0.1 (meters) as they are read.

3. The following sequence of commands computes the relativistic momentum of an electron with a velocity of 10^6 m/sec.

```
V = 1.E6 ;
C = 2.998e8 ;    RM = 9.1E-31 ;    ! Speed of light and rest mass.
Gamma = (1 - (V/C)**2) ** (-.5) ;
P = Gamma * RM * V ;
```

4. The following commands illustrate the pseudo-array (multi-variable) capability. The single command

```
x = 1.0, 4.0, 9.0 ;
```

will produce precisely the same results as the following six commands.

```
INTEGER x_dim ;    x_dim = 3 ;    x = 1.0 ;
x_1 = 1.0 ;    x_2 = 4.0 ;    x_3 = 9.0 ;
```

That is, both produce the same five variables and values. The integer variable `x_dim` is automatically created to record the number of subscripted variables created. Note that the base variable `x` (without underscore or integer) is also created and assigned the value of the first data entry. The same variables and assignments could also be produced explicitly using variable substitution, for example,

ASSIGN Command

```
x = 1.0 ;
do i = 1, 3 ;
    x_'i' = i ** 2 ;
enddo ;
```

5. The following example demonstrates type declaration and assignment for character variables.

```
Alfa = "Design # 4993" ; ! Quotes implicitly declare character.
CHARACTER Beta ; ! Type explicitly declares character.
Beta = Alfa ; ! Quotes are not required on Alfa.
```

Note that Alfa and Beta must both be declared type character before the ASSIGN command.

6. In the following example, concatenation of character strings is used to produce a file name for a CALL command.

```
CHARACTER FILENAME, PATH, FULLPATH ;
FILENAME = "Input. mgc" ;
PATH = "C:\Inputdir\" ;
FULLPATH = " 'PATH' 'FILENAME' " ;
CALL FULLPATH ;
```

7. The ASSIGN command can also be used to display the present value of a previously assigned variable. For example, the command

```
x_3 ;
```

will display the value, nine (see Example 4).

INTEGER Command

Function: Declares a variable to be type integer.

Syntax:
INTEGER variable_name ... ;

Arguments:
variable_name - name of a variable.

Description:

The INTEGER command explicitly declares a variable to be type integer. In an implicit declaration (see ASSIGN command), an integer variable must begin with a letter, i – n. Hence, the real purpose of the INTEGER command is to allow variables that begin with other letters to be type integer.

Any number of integer variables may be declared in a single command. If explicit declaration is used, it must precede the variable assignment and use. Once declared (implicitly or explicitly), the variable type can never be changed, but new values may be assigned to the variable, as convenient. The integer variable can be used in place of an integer constant wherever one is required as a data entry.

See Also:

ASSIGN, Ch. 6
REAL, Ch. 6
CHARACTER, Ch. 6

Examples:

In this example, the variable X is declared to be type integer. In the absence of this explicit declaration, the variable would be real, and the data entry would be converted to a real value (1.0).

```
INTEGER X ;  
X = 1 ;
```

REAL Command

Function: Declares a variable to be type real.

Syntax:

REAL variable_name ... ;

Arguments:

variable_name - name of a variable.

Description:

The REAL command explicitly declares a variable to be type real. In an implicit declaration (see ASSIGN command), a real variable must begin with a letter, a – h or o – z. Hence, the real purpose of the REAL command is to allow variables that begin with other letters to be type real.

Any number of real variables may be declared in a single command. If explicit declaration is used, it must precede the variable assignment and use. Once declared (implicitly or explicitly), the variable type can never be changed, but new values may be assigned to the variable, as convenient. The real variable can be used in place of a real constant wherever one is required as a data entry.

See Also:

ASSIGN, Ch. 6

INTEGER, Ch. 6

CHARACTER, Ch. 6

Examples:

In this example, the variable I is explicitly declared to be type real. In the absence of this declaration, the variable type would be integer, and the real constant (1.345) would be truncated to an integer constant (1).

```
REAL    I    ;  
I  =    1.345 ;
```


CHARACTER Command

Function: Declares a variable to be type character.

Syntax:

CHARACTER variable_name ... ;

Arguments:

variable_name - name of a variable.

Description:

The CHARACTER command explicitly declares a variable to be type character. In an implicit declaration (see ASSIGN command), a character variable is declared by double quotes enclosing the value. Hence, the real purpose of the CHARACTER command is to allow character variables to be assigned other character variables. In this case, the double quotes are not entered.

Any number of character variables may be declared in a single command. If explicit declaration is used, it must precede the variable assignment and use. Once declared (implicitly or explicitly), the variable type can never be changed, but new values may be assigned to the variable, as convenient. The character variable can be used in place of a character constant wherever one is required as a data entry.

See Also:

ASSIGN, Ch. 6
INTEGER, Ch. 6
REAL, Ch. 6

Examples:

This example illustrates some simple character variable operations.

```
CHARACTER alfa, beta ; ! alfa explicitly declared character.  
alfa = " This is a string. " ; ! Quotes are still necessary.  
beta = alfa ; ! both are character variables, no quotes.
```

FUNCTION Command

Function: Creates functions for use by other commands.

Syntax:

```
FUNCTION function_name(x,...) = expression, ... ;  
FUNCTION function_name DATA npairs x,y ... ;
```

Arguments:

function_name	-	function name. = arbitrary, user-defined.
x	-	first independent variable, etc. = arbitrary, user-defined.
expression	-	arithmetic expression in the FORTRAN style containing constants, variables, intrinsic functions, user-specified functions, the mathematical operators add (+), subtract (-), multiply (*), divide (/), and exponentiate (**), and the Boolean operators .GT., .LT., .LE., .GE., .EQ., and .NE.
npairs	-	number of data pairs (integer).
x,y	-	independent and dependent values of data pair (real).

Description:

The FUNCTION command is used to create a function and give it a unique function_name. Then the function can be used in other commands simply by entering the function_name. Only the function_name, and not the parentheses or independent variables, is entered. (The function_name must be unique, and a previously-created function will be replaced by a new function if they have the same function_name.)

The two syntactical forms allow either expressions or tabular data. An expression may contain up to ten independent variables. They must be enclosed by parentheses and separated by commas within the function_name. Expressions are entered as strings in FORTRAN style. The equals sign shown in the command syntax must be entered.

As the syntax indicates, it is possible to associate more than one expression with a single function_name. In this event, MCL creates a vector (multi-value) function. This capability can be used very effectively with the pseudo-array (multi-variable) capability described in the ASSIGN command (also see Examples, below).

Mathematical and Boolean operators have the same relative priority as in FORTRAN. The Boolean operators, .GT., .LT., .LE., .GE., .EQ., and .NE. can be used to construct logical functions (see Examples, below). The result of a logical expression using Boolean

FUNCTION Command

operators is 1 if the expression is true and 0 if the expression is false. Two pre-defined system variables, SYS\$TRUE (=1.0) and SYS\$FALSE (=0.0), exist to facilitate Boolean operations.

The expression may contain various other existing functions, which include:

- user-specified functions (previously created in FUNCTION commands),
- intrinsic mathematical functions (see Tables),
- grid mesh and index functions (see Tables), and
- feedback-loop functions (see Tables).

Any user-specified function (one previously created in a FUNCTION command) can be included in another FUNCTION command.

The Tables list all of the available in-line functions. The intrinsic mathematical functions include all of the standard trigonometric, transcendental, hyperbolic, numeric, and cylindrical Bessel functions. The grid mesh functions include coordinates as a function of indices and indices as functions of coordinates.

The Tables also list two feedback-loop functions, OBS\$name and LASTKEY. The LASTKEY function is the ASCII code (integer value) of the last key stroke on the keyboard. This allows user feedback, i.e., the ability to control features of the simulation while it is in progress. Time-history data (see OBSERVE command) from the simulation itself may be entered as a function, thus providing a simulation feedback loop. Observer values are referenced as OBS\$name, where "name" can be set in the OBSERVE command. The value of OBS\$name when the function is computed is the value of the observer at the end of the previous time step. Examples using both types of feedback loops are given below.

For the DATA option, a set of (x,y) data pairs is entered in order of increasing x. This produces a function of a single independent variable, or $y = \text{name}(x)$. When the function is evaluated for a value of x which falls between data values, y will be computed via linear interpolation. When it is evaluated for a value outside the data range, the nearest end-point value is used; i.e., extrapolation is never performed.

Restrictions:

1. A function must be defined before it is referenced by other commands.
2. A function may not reference itself.
3. A function whose name duplicates an intrinsic function will replace the intrinsic function.
4. Function names must not duplicate variable names.
5. The number of data pairs per function is limited to 1000.

FUNCTION Command

See Also:

ASSIGN, Ch. 6
OBSERVE, Ch. 22

Examples:

1. The following commands create two functions. The first function, named ALFA, is a sine wave function of retarded time. The second function, named BETA, includes the first, modifying it with the addition of some noise.

```
C = 2.998E8 ;  
FUNCTION ALFA(X,T) = 10. * SIN (4E9 * (X/C+T) ) ;  
FUNCTION BETA(X,T) = ALFA(X,T) * (1 + GAUSSIAN(0,.05)) ;
```

2. A command syntax contains the symbol, af(t,x), indicating a requirement for two independent variables. However, the desired input is linear in t and independent of x. This can be entered as an expression or even by using the DATA option.

```
FUNCTION PULSE(T,X) = T / 1.0E-8 ;  
FUNCTION PULSE DATA 2 0.0,0.0 1.0E-8,1.0 ;
```

Within the range ($0 < t < 1e-8$), these functions will give the same result. MCL ignores the unused independent variable from the first command. In the second command, MCL interprets the data as a function of the first independent variable (in this case, t) and performs linear interpolation between the data points. (The data option cannot be used with more than one independent variable.)

3. A vector (multi-value) function is created by entering multiple expressions in the same FUNCTION command. In the following example, a vector function having different linear rates is created and the values evaluated at $t = 1.0$ are assigned to a (multi-variable) pseudo-array.

```
FUNCTION Pulse(t) = 1*t, 2*t, 3*t ;  
X = Pulse(1.0) ;
```

These two commands create the variables and values equivalent to those produced in the following commands:

FUNCTION Command

```
INTEGER  X_DIM  ;      X_DIM = 3  ;      X  =  1.0  ;  
X_1  =  1.0  ;      X_2  =  2.0  ;      X_3  =  3.0  ;
```

4. The LASTKEY function provides user feedback-loop capability. The following commands enable beam emission to be controlled interactively from the PC console by pressing the B-key (ASCII code 66). The function expression uses a Boolean logic operator, and the value is either 0 (false) or 1 (true). The beam is turned on when the B-key is pressed and turned off when any other key is pressed.

```
FUNCTION  BEAM_ON(T)  =  LASTKEY.EQ.66  ;  
FUNCTION  CURRENT_J(T)  =  BEAM_ON(T)  ;  
EMISSION  KEYED_BEAM  ELECTRON  1  1  BEAM  CURRENT_J  ;
```

5. The OBS\$name function provides simulation feedback loop capability. We wish to use the dynamic voltage measured at Port_B to modify the incoming voltage applied at Port_A using a PORT command. The following commands could be employed.

```
OBSERVE  FIELD  E2  Port_B  NAME_SUFFIX  VOUT;  
FUNCTION  VINC(T)  =  CONST - OBS$VOUT  ;  
PORT Port_A  POSITIVE  TM  1  INCOMING  VINC  1  VOLTAGE;
```

FUNCTION Command

Trigonometric Functions.

Name	Function	Argument	Definition
SIN(x)	real	real	Sine (x), argument in radians
COS(x)	real	real	Cosine (x), argument in radians
TAN(x)	real	real	Tangent (x), argument in radians
ASIN(x)	real	real	Arcsine (x), result in radians
ACOS(x)	real	real	Arccosine (x), result in radians
ATAN(x)	real	real	Arctan (x), result in radians
ATAN2(x,y)	real	real, real	Arctan(x/y), result in radians

Transcendental Functions.

Name	Function	Argument	Definition
SQRT(x)	real	real	Square Root: $(x)^{(1/2)}$
ELLIPTIC1(x)	real	real	Complete elliptic integral of first kind: K(x)
ELLIPTIC2(x)	real	real	Complete elliptic integral of second kind: E(x)
EXP(x)	real	real	Exponential: e^x
LOG(x)	real	real	Natural Logarithm: $\ln(x)$
LOG10(x)	real	real	Common Logarithm: $\log_{10}(x)$

Hyperbolic Functions.

Name	Function	Argument	Definition
SINH(x)	real	real	Hyperbolic Sine: $\sinh(x)$
COSH(x)	real	real	Hyperbolic Cosine: $\cosh(x)$
TANH(x)	real	real	Hyperbolic Tangent: $\tanh(x)$

Numeric Functions.

Name	Function	Argument	Definition
ABS(x)	real	real	Absolute Value: $ x $
INT(x)	integer	real	Truncated Integer: $x = I + r, r < 1$
MAX(x,y)	real	real, real	Largest Value of x or y
MIN(x,y)	real	real, real	Smallest Value of x or y
MOD(x,y)	real	real, real	Remaindering: $x - \text{INT}(x/y) * y$
NINT(x)	integer	real	Nearest Int: $\text{INT}(x+.5), x > 0; \text{INT}(x-.5), x < 0$
SIGN(x)	real	real	Sign of x; -1 (x<0), +1 (x>0)
GAUSSIAN(x,y)	real	real, real	Gaussian Random No: $\exp(-(r-x)^2/2y)$
RANDOM	real	none	Uniform Random No; $0 < r < 1$
STEP(x,y)	real	real, real	Step Function: 1 (x>y), 1/2 (x=y), 0 (x<y)
THETA(x)	real	real	Step Function: 1 (x>0), 0 otherwise

FUNCTION Command

Cylindrical Bessel Functions

Name	Function	Argument	Definition
BESSELJ0(x)	real	real	J0(a)
BESSELJ1(x)	real	real	J1(a)
BESSELY0(x)	real	real	Y0(a)
BESSELY1(x)	real	real	Y1(a)
BESSELI0(x)	real	real	I0(a)
BESSELI1(x)	real	real	I1(a)
BESSELK0(x)	real	real	K0(a)
BESSELK1(x)	real	real	K1(a)
BESSELJN(n,x)	real	integer, real	J _n (x)
BESSELJP(n,x)	real	integer, real	J' _n (x)
BESSELYN(n,x)	real	integer, real	Y _n (x)
BESSELYP(n,x)	real	integer, real	Y' _n (x)
BESSELJNZ(n,x)	real	integer, real	Zeros of Bessel Function, J _n (x)
BESSELJPZ(n,x)	real	integer, real	Zeros of Derivative, J' _n (x)
BESSELYNZ(n,x)	real	integer, real	Zeros of Bessel Function, Y _n (x)
BESSELYPZ(n,x)	real	integer, real	Zeros of Derivative, Y' _n (x)

Grid Mesh Functions

Name	Function	Argument	Definition
X1FG(i)	real	integer	Nearest x1 full-grid point vs. grid index
X2FG(i)	real	integer	Nearest x2 full-grid point vs. grid index
X3FG(i)	real	integer	Nearest x3 full-grid point vs. grid index
X1HG(i)	real	integer	Nearest x1 half-grid point vs. grid index
X2HG(i)	real	integer	Nearest x2 half-grid point vs. grid index
X3HG(i)	real	integer	Nearest x3 half-grid point vs. grid index
X1GR(x)	real	real	X1 coordinate vs. real (continuous) grid index
X2GR(x)	real	real	X2 coordinate vs. real (continuous) grid index
X3GR(x)	real	real	X3 coordinate vs. real (continuous) grid index

FUNCTION Command

Grid Index Functions

Name	Function	Argument	Definition
I1FG(x)	integer	real	Grid index vs. nearest x1 full-grid point
I2FG(x)	integer	real	Grid index vs. nearest x2 full-grid point
I3FG(x)	integer	real	Grid index vs. nearest x3 full-grid point
I1HG(x)	integer	real	Grid index vs. nearest x1 half-grid point
I2HG(x)	integer	real	Grid index vs. nearest x2 half-grid point
I3HG(x)	integer	real	Grid index vs. nearest x3 half-grid point
I1CL(x)	integer	real	Grid index of cell containing x in x1
I2CL(x)	integer	real	Grid index of cell containing x in x2
I3CL(x)	integer	real	Grid index of cell containing x in x3

Feedback Loop Functions

Name	Function	Argument	Definition
LASTKEY	integer	none	ASCII code of last key stroke by user
OBS\$name	real	none	Time-history data from simulation

7. CONTROL STATEMENTS

This Chapter covers the following commands:

DO / ENDDO
IF / ELSEIF / ELSE / ENDIF
CALL / RETURN
\$namelist\$

You can use these commands to control the flow of commands to be processed.

The DO / ENDDO commands provides the capability to loop repeatedly over the same block of commands. They function in a manner similar to their FORTRAN counterparts. However, the counter variable and the increment can be real as well as integer, and the counter increment can be negative as well as positive. The counter can be reset within the loop to provide an exit.

The logical IF commands allow decision making and branching within the input file. Again, these commands follow the FORTRAN convention. Any number of ELSEIF commands can be used within the IF / ENDIF pair; however, only one ELSE command is allowed.

The CALL / RETURN construct provides a pseudo-subroutine capability that allows a single, large input file to be broken up into smaller, more convenient files. This allows better organization of the input data by function. It also avoids unnecessary duplication of input data, since any data which repeats can simply be placed in a separate file and called repeatedly. The input data contained in one file is available to all the other files (there is no hidden input data).

Finally, the \$namelist\$ command provides a facility to process existing namelist files without altering them. This allows input data produced for other applications to be read directly.

This page is intentionally left blank.

DO / ENDDO Commands

Function: Repeats execution of a block of commands.

Syntax:

```
DO variable = expression, expression [ , expression ] ;  
    [ block ]  
ENDDO ;
```

Arguments:

variable	-	do-loop counter variable of type integer or real.
expression	-	scalar expression of type integer or real. The first expression is the initial value of the variable, the second is the final value, and the third (optional) is the increment (default = 1).
block	-	a sequence of executable commands.

Description:

The DO / ENDDO construct allows repetition of a blocked set of commands. The block must be placed between the DO and ENDDO commands. The variable is the do-loop counter, which may be real or integer. Its initial, final, and increment values are evaluated from the three expressions. If the third expression is not entered, then the increment is given a default value of unity. The block is repeated until the variable passes the final value (the increment can be negative and the variable decreasing).

DO / ENDDO constructs can be nested. One do-loop can lie within another as long as the range of the inner loop lies completely within the range of the outer DO loop. In general, each of the nested loops should have a unique variable.

The value of the do-loop counter variable can be changed within the loop, which provides a useful means for exiting the loop. This is typically associated with a logical IF test on some variable calculated within the loop. To cause an exit, simply set the variable to a value beyond the final value.

Restrictions:

Do-loop nesting is limited to a depth of four.

DO / ENDDO Commands

Examples:

1. Do-loops can be used for many different purposes. This example demonstrates the creation and unique labeling of ten repeating area “islands” which might be used to construct ridges in a waveguide. Given the island width w , spacing d , and height h , we have

```

Xa = - d ;
Ya = 0 ;
Yb = Ya + h ;

Nislands = 10 ;
DO I = 1, Nislands ;
    Xa = Xa + d ;
    Xb = Xa + w ;
    AREA Island'I' RECTANGULAR Xa,Ya Xb,Yb ;
ENDDO ;

```

The resulting areas are named “Island1, Island2, ... Island10.”

2. This example inverts a transcendental equation using Newton’s method. It uses do-loop variable reassignment to terminate the loop when convergence is achieved.

```

! Problem: solve f(x) = x exp(x) = 10 for x
FUNCTION f(x) = x * exp (x) ;
y = 10. ; x_guess = 1.0 ; dx = 0.001 * x_guess ;
tolerance = 0.0001 * x_guess ; ! four decimal points accuracy.
DO iter = 1, 1000 ;
    x_solution = x_guess -
        ( f (x_guess) - y ) * dx / (f(x_guess + dx) - f(x_guess)) ;
    IF ( ABS (x_solution - x_guess) .LT. tolerance ) THEN ;
        iter = 1001 ;
    ENDIF ;
    x_guess = x_solution ;
ENDDO ;
! Returns with x_solution = 1.7455

```

IF / ELSEIF / ELSE /ENDIF Commands

Function: Provides conditional execution of commands.

Syntax:

```
IF (logical_expression) THEN ;
    [ block ]
[ ELSEIF (logical_expression) THEN ;
    block ]
...
[ ELSE ;
    block ]
ENDIF ;
```

Arguments:

logical_expression	- logical expression.
block	- sequence of executable commands.

Description:

The IF command works together with ELSE, ELSEIF, and ENDIF commands to provide conditional execution of other commands. As the syntax indicates, the ELSE and ELSEIF branches are optional. Multiple ELSEIF branches can be entered between the IF and ENDIF commands, but no more than one ELSE branch.

In writing a logical expression, mathematical and Boolean operators have the same relative priority as in FORTRAN. The Boolean operators, .GT., .LT., .LE., .GE., .EQ., and .NE. can be used to construct logical expressions. The result of a logical expression is 1 if the expression is true and 0 if the expression is false. Two pre-defined system variables, SYS\$TRUE (=1.0) and SYS\$FALSE (=0.0), exist to facilitate Boolean operations.

Examples:

In the following example, the number of time steps, Kmax, is calculated from variables Kcycle and Ncycles, which have been entered previously in the input data. (The variable Kcycle is the number of time steps in an RF cycle, and Ncycles is the number of RF cycles.)

```
IF ( Ncycles .EQ. 0 ) THEN ;
    Kmax = 1 ;
ELSE IF ( Ncycles .GT. 0 ) THEN ;
    Kmax = Ncycles * Kcycle ;
```

IF / ELSEIF / ELSE /ENDIF Commands

ENDIF ;

Notice that a negative value of Ncycles will result in Kmax being undefined, unless it is assigned elsewhere.

CALL / RETURN Commands

Function: Opens and exits a new command file.

Syntax:

(in the first file)

CALL new_file ;

(at the end of new_file)

RETURN ;

Arguments:

new_file - name of a second command file.

Description:

The CALL / RETURN commands provide pseudo-subroutine capability. It allows a single input file to be broken up into a number of smaller files, which can be organized according to function. For example, it may be desirable to separate design parameter data from simulation commands. Also, commands which would be repeated many times in an input file are clearly a candidate for this treatment.

A CALL command in one file transfers control to "new_file". (All of the constants, variables etc., from either file are accessed by the other, so there is no list of arguments to be transferred between files.) New_file can contain any commands which could be used in the original file. It can also contain logical IF commands which lead to one or more RETURN commands. However, the last command in new_file must be a RETURN command. When any RETURN command is encountered, new_file will be closed, and commands after the CALL in the original file will be processed.

CALL commands may be nested up to five levels deep. Thus, new_file may also contain CALL commands to other files, etc.

Restrictions:

1. CALL commands cannot be nested more than five levels deep.
2. Any file which is accessed with a CALL command must be terminated with a RETURN command.
3. CALL commands are not allowed within do-loops.

See Also:

\$namelist\$, Ch. 7

CALL / RETURN Commands**Examples:**

In developing a template for simulation of a generic device, one might wish to isolate the actual input data for a specific design from the generic simulation (algorithm, output, etc.) commands. Thus, at the point in the template where design data is required, the command

```
CALL design.mgc ;
```

could be inserted. This file, design.mgc, would then contain the design-specific data (and be terminated with a RETURN command). The advantage of this approach is that the same simulation methodology can be applied to many different designs which are isolated and identified in individual files.

\$namelist\$ Command

Function: Processes FORTRAN-style namelists.

Syntax:

```
$group_name  
    [ block ]  
$[end]
```

Arguments:

group_name	-	symbolic name of namelist.
block	-	block of variable = value assignments.
end	-	terminator on some namelists (ignored by MCL).

Description:

The \$namelist\$ command is provided to facilitate data entry from FORTRAN-style namelists. (Use of this command for any other purpose is discouraged.) It allows a namelist file to be read simply by using the CALL command to access the file. It should not be necessary to alter the namelist file itself.

The namelist file may contain more than one namelist. In each namelist, the block between \$group_name and \$end should consist of the usual FORTRAN-style assignments, i.e., variable = value. Assignments should be separated by commas. Array and character assignments are permitted.

MCL initially treats each namelist as a single MCL command. The command begins and ends with the namelist delimiter (\$). (The DELIMITER command can be used to change the default delimiter (\$) to accommodate an unusual namelist convention.) Everything to the right of the second delimiter is ignored by MCL. Hence the “end” in \$end is ignored. Thus, it is not necessary to add a RETURN command to a namelist file, even though it is accessed by a CALL command.)

MCL accommodates multiple occurrences of the same namelist in a file by using a pseudo-array capability for group_name variables. It will also create a variable named group_name_INSTANCES to record the number of such occurrences. It will treat any namelist arrays which it encounters in a similar manner. (The user is encouraged to become familiar with the variable-naming convention in order to effectively use the namelist data in other MCL commands. See Examples, below).

See Also:

CALL, Ch. 7
DELIMITER, Ch. 8

\$namelist\$ Command**Examples:**

A popular 1-D helix-TWT program requires input from a namelist file. This file, named HELIX.NML, includes the following lines:

```
$helix  name="input helix", radius=1.,  
pitch=.300,.315  
$end  
$helix  name="output helix",  
radius=1.,  
pitch=.315,.295,.285  
$end
```

The MCL input file contains a single command to read this namelist file:

```
CALL HELIX.NML ;
```

The processing of this namelist file will produce variables and values equivalent to those that would be obtained by executing the following commands:

```
INTEGER HELIX_INSTANCES ;  
HELIX_INSTANCES = 2 ;  
HELIX_1.NAME = "input helix" ;  
HELIX_1.RADIUS = 1. ;  
INTEGER HELIX_1.PITCH_DIM ;  
HELIX_1.PITCH_DIM = 2 ;  
HELIX_1.PITCH = .300 ;  
HELIX_1.PITCH_1 = .300 ;  
HELIX_1.PITCH_2 = .315 ;  
HELIX_2.NAME = "output helix" ;  
HELIX_2.RADIUS = 1. ;  
INTEGER HELIX_2.PITCH_DIM ;  
HELIX_2.PITCH_DIM = 3 ;
```

\$namelist\$ Command

```
HELIX_2.PITCH = .315 ;  
HELIX_2.PITCH_1 = .315 ;  
HELIX_2.PITCH_2 = .295 ;  
HELIX_2.PITCH_3 = .285 ;
```

This page is intentionally left blank.

8. I/O UTILITIES

This Chapter covers the following commands:

BLOCK
COMMENT / C / Z / !
DELIMITER
ECHO / NOECHO
JOURNAL

You can use these commands to modify the input file and to control disposition of the processed commands.

The **BLOCK** commands allow the transfer of blocks of text from the input file to any other file. It can provide substantial automation for simulations which involve more than one code. For example, the input file for a **MAGIC** simulation can automatically create the input file required for post-processing with **POSTER**. It can even submit the **POSTER** run automatically. The **BLOCK** commands can be used to wrapper codes.

MCL supports extensive commenting capability with the **COMMENT** commands. Combined with the use of (self-documenting) variables, input files can easily be fully documented. In addition, you can use this capability to disable commands temporarily without actually removing them from the input file.

You can use the **DELIMITER** command to redefine any of the delimiter characters. For example, if an existing namelist file contains a non-standard delimiter, you can simply redefine the **MCL** delimiter, whereas changing the file itself might render it unreadable in its original application. Another convenient use of this command is to redefine the command delimiter (default ;) as a carriage return. This would potentially allow actual **FORTRAN** to be processed.

The **ECHO / NOECHO** commands provide control over what goes into the output (log) file, to prevent the huge output files which would result from processing commands within a do-loop, for example. The **JOURNAL** command can be used to record commands entered interactively. Thus, a file can be developed interactively and saved for subsequent runs in the batch mode.

This page is intentionally left blank.

BLOCK / ENDBLOCK Commands

Function: Copies blocks of text from the input file to some other file.

Syntax:

```
BLOCK COMMENT ;  
    [ block ]  
ENDBLOCK ;
```

or

```
BLOCK WRITE file_name [ write_mode [ carriage_control ] ] ;  
    [ block ]  
ENDBLOCK ;
```

Arguments:

file_name	- name of the other file.
write_mode	- file write mode. = NEW (default), OVERWRITE, or APPEND.
carriage_control	- carriage control for file_name. = FORTRAN (default) or LIST.
block	- arbitrary text.

Description:

The primary use of the BLOCK / ENDBLOCK command is to copy a block of text from the input file to some other file. It provides a convenient means of creating other input files for post-processing, etc. Text is copied verbatim, with one exception: any designated variable substitution is performed before the text is copied (see Examples, below).

Following the BLOCK keyword, you must select either COMMENT or WRITE. The COMMENT keyword copies only to the output (log) file, while the WRITE keyword copies to any user-specified file_name. In the latter case, write_mode specifies what will happen if the file already exists. If you enter NEW (the default), the system tries to open a new file. It will not overwrite or destroy an existing file; instead, the BLOCK command returns with an error message. Entering OVERWRITE will destroy an existing file. Entering APPEND adds to the existing file and is useful for constructing files a bit at a time.

The carriage_control provides carriage control in file_name. If you enter FORTRAN, a blank character will be placed at the beginning of each line (required for certain FORTRAN programs), whereas LIST will cause each line to start in column one. Any variable substitution (designated by appropriate delimiters, see ASSIGN command) will be performed before the text is copied. The block of text itself is delimited by the ENDBLOCK command.

The BLOCK commands can provide substantial automation. For example, an input file for one code, e.g., MAGIC, can also write an input file for post-processing in another code, e.g., POSTER. It can even submit the post-processing job. Variable substitution allows

BLOCK / ENDBLOCK Commands

important data, e.g., a beam voltage or an antenna current, to be passed from the MAGIC run to the subsequent post-processing run. Ultimately, the combination of the BLOCK command, variable substitution, and the COMMAND command give MCL very powerful wrapping capability, not just for the MAGIC family of codes, but for any set of codes.

See Also:

ASSIGN, Ch. 6

COMMAND, Ch. 9

Examples:

The following example writes an input file for one of the POISSON magnet design codes. It also writes a DOS batch file to run the POISSON code. Finally, it submits a DOS command to run the batch file. (The variables, TITLE, IRUN, DX, RMAX, and ZMAX have been previously defined.)

```
! === write AUTOMESH input file ===
BLOCK WRITE "'IRUN'.DAT" OVERWRITE ;
'TITLE'
$REG NREG=1, DX='DX', XMAX='RMAX', YMAX='ZMAX',
      NPOINT=5, ITRI=2 $
$PO X=0.0 , Y=0.0 $
$PO X=0.0 , Y='ZMAX' $
$PO X='RMAX' , Y='ZMAX' $
$PO X='RMAX' , Y=0.0 $
$PO X=0.0 , Y=0.0 $
ENDBLOCK ;
! === DOS batch file to run AUTOMESH ===
BLOCK WRITE "'IRUN'.BAT" OVERWRITE ;
AUTOMESH < 'IRUN'.DAT >> AUTOMESH.LOG
COPY OUTAUT 'IRUN'.AUT
COPY TAPE73 'IRUN'.T73
DEL OUTAUT
DEL TAPE73
ENDBLOCK ;
! === Run batch file ===
COMMAND "'IRUN'.BAT" ;
```


COMMENT / C / Z / ! Commands

Function: Creates comments in input and/or output files.

Syntax:

COMMENT "comment" ;

C [command] ;

Z [command] ;

[command] ! [comment]

Arguments:

comment - user-defined character string.
command - command name plus data entries.

Description:

MCL allows comments to be entered in the input file in a variety of ways. Some, but not all, are echoed in the output (log) file.

The COMMENT command is intended to enter general comments in the input file which are echoed in the output file. The presence of a COMMENT command cannot affect the behavior of the simulation. The comment must be enclosed by double quotes. Thus, any character except double quotes is permitted within the comment. For example, a semicolon is permitted within the comment. (Also, see BLOCK COMMENT command, which does not require double quotes.)

The C command is typically used to preserve another command in the input file without executing it. Because the original command is now ignored, the simulation results may change. The C command can extend over many lines and is terminated by the command delimiter. It is typically created simply by adding the character C to the beginning of an existing command.

The Z command is also used to preserve a command in the input file without executing it. Thus, it too can affect the behavior of the simulation. It can extend over many lines and is terminated by the command delimiter. (The only difference between the C and Z commands is that the Z writes the message "Command Ignored" in the output file, whereas C writes nothing.)

The ! command is typically used to enter short comments following another command on a single line of the input file. The comment is not echoed in the output file. Thus, ! is similar to C. They differ in that C may extend over many lines (before the command delimiter), while ! applies only to the remainder of a single line (and has no delimiter). Note

COMMENT / C / Z / ! Commands

that an executable command may extend over many lines, and any or all of the lines may contain a ! comment.

See Also:

ASSIGN, Ch. 6

BLOCK COMMENT, Ch. 8

Examples:

The commands below depict various uses of the comment commands.

```
COMMENT "The C command below preserves the ASSIGN command in
the input file, but no information will be written to the
output file. (By contrast, Z would write, Command Ignored.)
In either case, the command will be ignored (not processed).
Note that, as written, the command below extends over two
lines." ;
```

```
C ALFA
    = 100;
```

```
! This comment extends only to the end of the line.
! Every new line needs new ! command. No delimiter. No
! echo in output.
! Can comment out another command if it is all on one line.
! An example is: TITLE "This is a title" ;
! (This TITLE command will not be processed.)
! However, the ASSIGN command below will most definitely be
! processed.
```

```
        ALFA = 100. ! This comment might explain why ALFA
! is set to 100.
```

```
! Note that a command with ! comments may be spread over
! many lines, e.g.,
```

```
        FUNCTION Big_wave(x,t) = ! This line has the
! function name
!           SIN (k*x - w*t) ;      ! This line has the
! expression
```

```
! The following COMMENT command illustrates variable substitution
A = 5. ;
COMMENT " The value of A is 'A' " ;
! The comment will read " The value of A is 5.00e+00 "
```

DELIMITER Command

Function: Redefines delimiters.

Syntax:

DELIMITER [type] character ;

Arguments:

type	-	type of delimiter. = COMMAND (default type), default character is semicolon (;) = SUBSTITUTION, default character is single quote (') = STRING, default character is double quote (") = NAMELIST, default character is dollar sign (\$) = FORMAT, default character is colon (:)
character	-	ASCII character. = RETURN, or select from character set.

Description:

The DELIMITER command is used to change MCL delimiters. The new delimiter will take effect on the following command. Delimiters may be changed as often as desired.

The type specifies which delimiter to change. A delimiter can be changed to virtually any other single character. If more than one character is entered, then the first character is used, and the remaining characters are ignored. The single exception to this rule occurs for the COMMAND delimiter, which may be set to a carriage return as described below.

The default value for the COMMAND delimiter is a semicolon (;). You can change this to a carriage return by entering the character constant, RETURN. With this delimiter, command continuation is provided with a (&) at the end of each line. (Alternatively, a long mathematical expression can be entered without continuation characters by enclosing the expression in double quotes.) The semicolon delimiter for commands remains active in addition to any new delimiter specified.

Restrictions:

Delimiters should not be set to a comma or to a period.

DELIMITER Command**Examples:**

1. The COMMAND delimiter can be set to a slash (/) to allow use of older MCL input decks, written when the default delimiter was a slash.

```
DELIMITER  COMMAND  /  ;
C  Then the following old commands don't need to be  changed. /
TITLE "TEST CASE 1" /
...
```

```
C  Reset the delimiter again  /
DELIMITER  RETURN  ;
C  This allows entering a command without using
a delimiter; however, the character &
must precede continuation lines  ;
```

2. The following commands illustrate how to create and use character variables to redefine the symbol substitution delimiter and the character string delimiter. First, character variables are assigned, and then the default SUBSTITUTION delimiter is replaced with the character %, and the default STRING delimiter is replaced with the character &. Next, the character variable SINGLEQUOTE is defined to be the character of the same name, as is the character variable DOUBLEQUOTE. Finally, the delimiters are returned to their original default values.

```
CHARACTER  SINGLEQUOTE  ;
CHARACTER  DOUBLEQUOTE  ;
DELIMITER  SUBSTITUTION  %  ;
DELIMITER  STRING  &  ;
SINGLEQUOTE  =  &'&  ;
DOUBLEQUOTE  =  &"&  ;
DELIMITER  STRING  %DOUBLEQUOTE%  ;
DELIMITER  SUBSTITUTION  %SINGLEQUOTE%  ;
```

3. The FORMAT delimiter is used for variable substitution with a specific format. The example below uses both the SUBSTITUTION and FORMAT delimiters.

```
BEAM_VOLT = 130.354Kilovolts ;
VOLTS = BEAM_VOLT/1.kilovolt ;
ASTRING = "Beam voltage is 'VOLTS:F4.0' kilovolts." ;
```

DELIMITER Command

The character variable ASTRING will have the value "Beam voltage is 130. kilovolts." Another example is given below in which a file name is created which includes a run number padded with zeros.

```
IRUN = 2 ;  
AFILE = "RUN'IRUN:I2.2'" ;
```

In this case, the character variable AFILE will have the value "RUN02".

ECHO / NOECHO Commands

Function: Turns on/off the echoing of processed commands to output file.

Syntax:

```
ECHO ;  
NOECHO ;
```

Arguments:

None

Description:

The ECHO command enables echoing output of processed commands to the output file. The NOECHO command disables echoing of the processed commands to the output file.

Examples:

This input demonstrates the use of ECHO and NOECHO. Within the do-loop, NOECHO and ECHO are used to suppress output from certain commands as unnecessary. For the commands between the NOECHO and the ECHO, output of the processed commands will only occur if there is an error in a command.

```
DO I = 1, 100 ;  
  NOECHO ;      !      Turn off output.  
    first block of commands ! Output is suppressed.  
  ECHO ;      !      Turn output back on.  
    second block of commands ! Output is on.  
ENDDO ;
```

JOURNAL Command

Function: Controls journal facility.

Syntax:
JOURNAL option ["astring"] ;

Arguments:

option	-	option to turn journal on and off, and to review contents of journal. = ON, OFF, REWIND, or REVIEW.
string	-	optional. Valid for REVIEW only. It selects lines from the journal with string in it.

Description:

A journal file can be used to record all of the valid commands entered during an interactive session. They will be recorded in the order entered, including comments. Then the journal file may be edited to rapidly create a new input file. This is useful when the program is used interactively to create the initial input. The file is always created with the name JOURNAL.DAT. The ability to close and reopen the journal file will depend on the operating system.

This page is intentionally left blank.

9. EXECUTION CONTROL

This Chapter covers the following commands:

START
STOP
TERMINATE
MESSAGE
PAUSE
RECORD
RESET
RESTART
COMMAND
KEYBOARD

You can use these commands to control execution of the input file.

The most commonly used commands are **START**, which initiates the simulation after all input data has been entered, and **STOP**, which exits from the application.

You can use the **TERMINATE** command to specify the circumstances under which you want the simulation to stop. The options range from none to errors of decreasing severity. The **MESSAGE** command will print any errors encountered, and offers the same specification of options. The **PAUSE** command will cause a pause to occur on the specified level of errors, but is more useful in diagnosing a code problem than for an input error.

The **RECORD**, **RESET**, and **RESTART** commands provide the basic means of continuing a simulation. Continuing is necessary when the simulation does not develop as quickly as planned or to get additional output. It is also a useful safety precaution for very long runs. **RECORD** writes data from the original simulation, **RESET** clears data from the original output specifications, and **RESTART** initiates the continued simulation.

The **COMMAND** command allows you to send a command to the operating system while your MCL application is still running. It can be used to start other applications from your MCL application. The **KEYBOARD** commands also allow you to interact with the simulation during execution.

This page is intentionally left blank.

START Command

Function: Interrupts the processing of input and initiates the simulation.

Syntax:
START ;

Arguments:
None.

Description:

The START command initiates execution of the simulation as defined by the previous input commands. If there are errors in any of the prior commands, the simulation will be terminated.

STOP Command

Function: Stops execution.

Syntax:
STOP ;

Arguments:
None.

Description:

The STOP command stops execution immediately after processing this command. Any commands that follow the STOP command are ignored.

TERMINATE Command

Function: Terminates execution on errors of various severity.

Syntax:

TERMINATE severity ;

Argument:

severity - Error severity level.
 = WARNING, terminate on abort, error, or warning.
 = ERROR, terminate on abort or error.
 = ABORT, terminate on abort (default).
 = NONE, do not terminate.

Description:

This command allows the user to control termination on errors of different levels of severity. Specifying severity causes termination for all errors of that and all greater severities. Termination occurs as soon as the error condition is encountered. No further input processing takes place.

The error conditions are listed above in order of increasing severity. The approximate meaning of the severity levels is as follows. The NONE level will attempt to ignore all errors and continue the simulation. ABORT conditions are generally due to code errors. ERROR conditions are usually caused by user input errors that result in failure. WARNING conditions are caused by user input that is legal, but in violation of the intended usage.

Restrictions:

Due to the evolutionary nature of the code, not all errors are yet categorised by severity level. These errors will not be captured by this command.

See Also:

MESSAGE, Ch. 9

PAUSE, Ch. 9

TERMINATE Command**Examples:**

If a variable is not defined prior to being used, this will generate an error message but may not cause termination. Setting the TERMINATE command to terminate on ERROR will prevent a job from continuing until the variable is corrected. In the following example, the variable VOLTMAX has been misspelled. Thus, this variable is undefined in the function VOLTME. In such cases, it may be better to ensure that termination occurs. However, note that the input data may contain more errors which will not be detected due to the termination.

```
TERMINATE ERROR ;  
DEFINE VOLTMAX 50E3 ;  
DEFINE WOMECA 10.7E10 ;  
FUNCTION VOLTME(T) = VLTMAX * SIN ( WOMECA*T ) ;
```

MESSAGE Command

Function: Controls error message printing.

Syntax:

```
MESSAGE severity ;  
MESSAGE MAXIMUM aborts errors warnings ;
```

Arguments:

severity	-	error severity level. = WARNING, print aborts, errors, and warnings messages (default). = ERROR, print abort and error messages. = ABORT, print abort error messages. = NONE, do not print error messages.
warnings	-	maximum number of message allowed (default=5000).
errors	-	maximum number of message allowed (default=500).
aborts	-	maximum number of message allowed (default=50).

Description:

This command controls the print messages for errors. (The error severity levels are described in the TERMINATE command.) This command is provided primarily to allow warning messages to be turned off. The option to turn off ABORT and ERROR level messages should be used with caution.

The MAXIMUM option allows you to change the number of messages that are printed before the code automatically terminates execution.

Restrictions:

Due to the evolutionary nature of the code, not all errors are yet categorised by severity level. These errors will not be captured by this command.

See Also:

TERMINATE, Ch. 9
PAUSE, Ch. 9

PAUSE Command

Function: Pauses on errors of various severities.

Syntax:

PAUSE option ;
PAUSE timer ;

Arguments:

option - pause option (alpha)
 = NONE, do not pause on errors (default).
 = ABORT, pause on abort (code) errors.
 = ERROR, pause on abort or user errors.
 = WARNING, pause on abort, user, or warning errors.

Description:

This command allows the user to tell the code to pause on different levels of errors. The levels of errors are described in the TERMINATE command. The option to pause the code is only useful when debugging code. Hence this command should only be used when working with MRC personnel to examine problems with running the code.

Restrictions:

Due to the evolutionary nature of the code, not all errors are yet categorized by severity. Such errors will not trigger a pause no matter what is set for the pause option.

See Also:

MESSAGE, Ch. 9
TERMINATE, Ch. 9

RECORD Command

Function: Records the simulation to disk for later continuation.

Syntax:

RECORD timer file_name period;

Arguments:

timer	-	timer name, defined in TIMER commands.
file_name	-	name of restart file.
period	-	retention period (days).

Description:

The RECORD command writes the entire state of the simulation to file_name, at the times specified by the timer. After the simulation finishes, it can be restarted from file_name using the RESTART command.

Periodically recording restart files during a long simulation protects the investment of CPU time from being lost in a system crash. Each new restart file overwrites the previous file, so the resulting file contains the most recent simulation state. One may also record a restart file on the last time step of the simulation, so that the simulation may be continued at a later time if desired. On Cray COS systems, the retention period specifies the number of days the resulting file will exist before being automatically deleted. On other systems, the data entry is accepted but ignored.

See Also:

RESTART, Ch. 9

Examples:

```
C These commands will write a restart file named
  RESTART every 100 time steps, and will preserve
  it for ten days. ;
RECORD TIMES RESTART 10 ;
TIMER TIMES PERIODIC 100 9999 100 ;
```

RESET Command

Function: Clears data from previously entered commands.

Syntax:
RESET command ;

Arguments:

command	-	command (name).
		= CONTOUR.
		= LOOKBACK.
		= OBSERVE.
		= PERSPECTIVE.
		= PHASESPACE.
		= RANGE.
		= SPECIAL.
		= TRAJECTORY.
		= VECTOR.
		= VELOCITY.

Description:

The RESET command clears the current parameters associated with the command option. This command is useful when resuming a simulation with a RESTART command. For example, the previously activated graphics output commands can be cleared and new ones entered.

Restrictions:

When used with OBSERVE, the number of new commands must be identical to the number of old commands. The same physical measurements must be specified. The only parameters that should be altered are the output controls and the time and frequency windows.

See Also:

RESTART, Ch. 9

Examples:

The following example illustrates the use of the RESET command to alter the output graphics selected after resuming a simulation with restart. Note that, in the original simulation the observe variables are simply recorded and no FFT processing is requested. The particle phase-space plots use the default axial limits. After resuming the simulation, the user resets the OBSERVE and PHASESPACE commands. New OBSERVE commands are entered which change the FFT option and specify both a time window and a frequency window for the FFT's. A new PHASESPACE command is entered, specifying a particular spatial region.

RESET Command

```
C Original simulation. ;
OBSERVE FIELD E1  Line_A ;
OBSERVE FIELD B3  Point_B ;
PHASESPACE FOR-PHASE AXES X1 X2 ;

C Resume simulation with RESTART. ;
RESET OBSERVE ;
OBSERVE FIELD E1  Line_A
    FFT 3
    WINDOW TIME 5.E-8  10.E-8
    WINDOW FREQUENCY 0 100E6 ;
OBSERVE FIELD B3  Point_B
    FFT 3
    WINDOW TIME 5.E-8  10.E-8
    WINDOW FREQUENCY 0 100E6 ;
RESET PHASESPACE ;
PHASESPACE FOR-PHASE AXES X1 X2
    AXIS X  0.1 0.2
    AXIS Y  0.2 0.3 ;
```

RESTART Command

Function: Reads in a previously recorded simulation for continuation.

Syntax:
RESTART file_name ;

Arguments:
file_name - input file.

Description:

The RESTART command reads the specified file immediately at the time that the command is processed. Thus, any commands entered before the RESTART command will be overwritten. After the RESTART command, other commands (such as the RESET command) may be entered to modify the original simulation. The simulation is started, as for any simulation, with the START command. For example, if continuation is desired, but for a greater time period, a new DURATION command must be entered after the RESTART command.

Restrictions:

1. Certain commands should not be entered after a RESTART command, because they will be ignored or will damage the fidelity of the simulation. For example, commands that change the physical configuration should not be entered. Similarly, changing any algorithm may damage the simulation. Most commands that request output may be entered if additions to the output are desired.

2. The deletion of output is not possible. Commands should be added carefully. It is not possible to retroactively request information about time steps preceding the time step at which the continuation is being restarted. Thus, OBSERVE commands may be added, but they should only request observation of time steps after the continuation of the simulation.

See Also:

RECORD, Ch. 9
DURATION, Ch. 11

COMMAND Command

Function: Issues a command to the operating system.

Syntax:

```
COMMAND "system command"  
    [ { MINIMIZE, MAXIMIZE } ]  
    [ CONTINUE ] ;
```

Arguments:

system command - operating system command in double quotes.

Description:

The COMMAND command sends a command to the operating system for immediate processing while the MCL application is still running. This allows other applications to be started from the MCL application.

If none of the optional parameters are entered, the MCL application waits while the new application executes and then it continues. In a Windows environment, MINIMIZE (MAXIMIZE) will minimize (maximize) the new application. CONTINUE will cause the present application to continue independent of the new application.

Examples:

A template contains some design variables which the user needs to set before they are processed. These variables are in a file named DESIGN.MGC. The following commands bring up the file in the DOS editor, forcing the user to edit and save the file before it is processed by the CALL command.

```
COMMAND "edit design.mgc" ; ! Forces user to edit file.  
CALL design.mgc ; ! Processes the file
```

KEYBOARD Commands

Function: Allows control key interaction with simulation during execution.

Syntax: N. A.

Arguments: N. A.

Description:

You may interact with a simulation during execution by pressing certain designated control keys. This allows examination of intermediate results while the simulation is still in progress.

There are two graphical output modes: screen and metafile. The system is always in one mode or the other, and the mode selected determines whether graphical output appears on the monitor (screen mode) or whether it is set to the PS graphics metafile (metafile mode). You can change the mode at will by pressing the f6 and Alt-f6 control keys or by using commands (Part 2: OUTPUT SCREEN and OUTPUT METAFILE). The designated control keys and their functions are as follows:

- Esc — terminates simulation being executed.
- f1 — creates new record in REC file (see below) and continues
- f2 — creates new REC file record and RST file and terminates
- f3 — turns PAUSE ON for graphics (screen mode only)
- f4 — turns PAUSE OFF for graphics (screen mode only)
- f5 — writes all time history plots up to current time step
- f6 — turns screen mode on and metafile mode off
- Alt-f6 — turns metafile mode on and screen mode off
- f7 — writes all time history plots and terminates simulation
- f8 — writes all phase-space plots
- f9 — writes all contour plots
- f10 — writes all perspective plots
- f11 — writes all range plots
- f12 — writes all vector plots

Pressing the f1 or f2 keys is done in anticipation of eventually continuing the simulation. The resulting REC file will be named MGCnnnnn.REC, where nnnnn is the current time step index, e.g., MGC00124.REC. Similarly, the RST file will be named MGCnnnnn.RST. In addition to the designated control keys, you can use an intrinsic function named LASTKEY in commands to provide customized interactive capability (FUNCTION, Ch. 6).

KEYBOARD Commands

Restrictions:

1. KEYBOARD commands are available only for the PC version of MAGIC.
2. Selection of metafile mode does not include selection of driver or file name.
3. Plots which require an accumulation of data over several time steps will not be displayed unless the interrupt time matches that of the timer.

See Also:

FUNCTION (LASTKEY), Ch. 6
OUTPUT SCREEN, Ch. 20
OUTPUT METAFILE, Ch. 20
TIMER, Ch. 11

This page is intentionally left blank.

10. OBJECTS

This Section covers the following commands:

POINT
LINE
AREA
SYSTEM
LIST
DELETE

You can use these commands to create spatial objects in your choice of coordinate system, to delete previously created objects, and to list objects for inspection.

There are three types of spatial objects: points, lines, and areas. In addition, the types may include different shape options. For example, lines may be either straight, conformal (with a coordinate system axis), circular or elliptical. Areas may be either rectangular, conformal, or functional. The conformal option is especially useful to create curved lines and areas in non-cartesian coordinate systems. If the coordinate system used to perform the simulation is not convenient to enter a spatial object, then you can specify another coordinate system just to enter the spatial data.

When you create a spatial object, you must give it an `object_name`. This allows the object to be referred to in other commands. For example, you can use an `AREA` command to create an object and then use a `DIELECTRIC` command (see Ch. 14) to specify its permittivity. Generally, the `object_name` should be a unique alphanumeric label which has meaning to you. If two or more objects are given the same `object_name`, they then form a group of objects which will be treated in the same manner by any subsequent commands which refer to that `object_name`.

There are other useful options. For example, you can delete spatial objects which have already been created, and you can list the existing objects.

This page intentionally left blank

POINT Command

Function: Specifies a point or points.

Syntax:
POINT object_name point,... ;

Arguments:

object_name - name of spatial object, user-defined.
point - spatial coordinates x1, x2 (m or rad).

Description:

The POINT command is used to specify the locations of points in space. (The keywords, POINT and POINTS, are interchangeable — that is, they will produce the same result from the same data.)

The arguments are defined as follows. The object_name is used to provide a unique label to a point or a group of points. (All points in a group will be treated in the same manner by subsequent commands which refer to this object_name.) Points may be given the same object_name simply by including them all in a single command or by issuing individual commands using the same object_name. In any subsequent operations on the group, the points will be treated in the order in which they are entered.

The argument, point, requires two real coordinates to completely define the point in the specified coordinate system. More than one point can be entered in a single command. The coordinate system used to defines the axes (see SYSTEM, Ch. 10) can be changed at will for convenience in entering spatial data. If no coordinate system is specified, then the default system will be in effect.

Restrictions:

In any subsequent operations on points in a group, the points will be treated in the order in which they are entered.

See Also:

LINE, Ch. 10
AREA, Ch. 10
SYSTEM, Ch. 10
DELETE, Ch. 10
LIST, Ch. 10
MARK, Ch. 11

POINT Command**Examples:**

In the examples which follow, it is assumed that the cartesian coordinate system is active unless otherwise noted.

(1) To enter a spatial point named ORIGIN at the origin of a cartesian coordinate system, the command is

```
POINT ORIGIN 0,0 ;
```

(2) To enter the two end-points defining the axis of symmetry (running from 0 to Zmax in z) in a two-dimensional cylindrical (z,r) coordinate system, some possible commands are

```
SYSTEM CYLINDRICAL ;  
POINTS AXIS 0,0 Zmax,0 ;
```

The two points will belong to the spatial object group named AXIS. Note that the group does not include a line object, but does include two points.

(3) To make magnetic field measurements on a group of points (a, b, ...) in cylindrical (z,r) coordinates, some possible commands are

```
SYSTEM CYLINDRICAL ;  
POINTS B_dot Za,Ra Zb,Rb Zc,Rc ... ;  
OBSERVE FIELD B3 B_dot ;
```

This illustrates the use of the object_name, B_dot, in performing subsequent operations (time-history plots) on the entire group of points.

LINE Command

Function: Specifies a line or lines.

Syntax:

```
LINE object_name STRAIGHT start_point end_point,... ;
LINE object_name CONFORMAL start_point end_point,... ;
LINE object_name CIRCULAR center_point start_point end_point ;
LINE object_name ELLIPTICAL center_point x_radius y_radius
                        start_angle end_angle ;
```

Arguments:

object_name	- name of spatial object, user-defined.
start_point	- end-point coordinates (m or rad).
end_point	- end-point coordinates (m or rad).
center_point	- coordinates (m or rad).
x_radius	- radius in x-axis (m).
y_radius	- radius in y-axis (m).
start_angle	- end-point angle (rad).
end_angle	- end-point angle (rad).

Description:

The LINE command is used to specify the locations of lines in space. (The keywords, LINE and LINES, are interchangeable — that is, they will produce the same result from the same data.)

The coordinate system used to define lines (see SYSTEM, Ch. 10) can be changed at will for convenience in entering the spatial data. If no coordinate system is specified, then the default system will be in effect.

The arguments are defined as follows. The object_name is used to provide a unique label to a line or a group of lines. (All lines in a group will be treated in the same manner by subsequent commands which refer to this object_name.) Lines may be given the same object_name simply by including them all in a single command or by issuing individual commands using the same object_name. In any subsequent operations on lines in a group, the lines will be treated in the order in which they are entered.

There are four shape options: STRAIGHT, CONFORMAL, CIRCULAR, and ELLIPTICAL. The simplest option is STRAIGHT, which requires only the specification of

LINE Command

two end_points. This will produce a single straight (linear) line, irrespective of the coordinate system (which serves only to define the end-points). Note that if more than two points are supplied, then more lines will be added in a “connect-the-dots” manner. Thus, three end_points will produce two lines, etc., and the lines will be joined end-to-end, continuously.

For many applications, it is desirable for lines to “bend” to conform with the local coordinates. This is easily accomplished using the CONFORMAL option. If more than two end-points are supplied, then the “continuation” effect described above is obtained. For example, a pie-shaped outline can easily be traced out in polar coordinates using this option.

The CIRCULAR option will produce a circular arc, irrespective of the coordinate system specified. The points required are, first, the center_point, which locates the center of the circle, followed by the arc end-point locations. The convention is that the arc is always drawn counter-clockwise, from the start_point to the end_point. This option requires that the user maintain the correct radius to within a reasonable approximation.

The ELLIPTICAL option will produce an elliptical arc, irrespective of the coordinate system specified. Next, the magnitudes of the axes, x_radius and y_radius, are entered. (Note that the orientation of the ellipse is limited to coincide with the major axes.) Finally, two angles specify the end-points of the arc. The convention is that the arc is always drawn counter-clockwise, from the start_angle to the end_angle. (The angles are measured counter-clockwise from the x-axis.)

Restrictions:

In any subsequent operations on a group, lines will be treated in the order in which they are entered.

See Also:

POINT, Ch. 10
AREA, Ch. 10
SYSTEM, Ch. 10
LIST, Ch. 10
DELETE, Ch. 10

Examples:

In the examples which follow, it is assumed that the cartesian coordinate system is active unless otherwise noted.

LINE Command

(1) To enter a line representing the axis of symmetry (running from 0 to $Z_{\max}$ in z) in a two-dimensional cylindrical (z,r) coordinate system, some possible commands are

```
SYSTEM CYLINDRICAL ;
LINE AXIS STRAIGHT 0,0 Zmax,0 ;
```

The line will belong to the spatial object group named AXIS. Note that the group does not include points, but does include one line.

(2) We can create the outline of a box of width, W , and height, H , with a single LINE command using the continuation convention.

```
LINE BOX STRAIGHT 0,0 W,0 W,H 0,H 0,0 ;
```

The result will be four lines with common end-points. The four lines will belong to a group called BOX. Note that this is not an AREA.

(3) This example uses DO/ENDDO commands to create ten lines at uniformly-spaced axial locations between a cathode at radius, R_c , and an anode at radius, R_a . (Using variable substitution, the individual lines are given the object_names, LINE1, LINE2, ..., LINE10.)

```
Z = Z0 ;
DO I 1, N ;
  Z = Z + dZ ;
  LINE VOLT'i' STRAIGHT Z,Rc Z,Ra ;
ENDDO ;
```

These lines could be used to measure cathode-anode voltages along a coaxial cable, for example.

(4) Here are two ways to create a circular arc of radius, R , subtending a quarter of a circle between 0 and $\pi/2$ degrees. The first way uses polar coordinates and the CONFORMAL option.

```
SYSTEM POLAR ;
LINE AB CONFORMAL R,0 R,pi/2 ;
```

The second way uses cartesian coordinates and the CIRCULAR option.

```
SYSTEM CARTESIAN ;
LINE AB CIRCULAR 0,0 R,0 0,R ;
SYSTEM POLAR ;
```

The two results should be identical when viewed in the same (POLAR) coordinate system.

AREA Command

Function: Specifies an area.

Syntax:

```
AREA object_name RECTANGULAR corner_point, corner_point;  
AREA object_name CONFORMAL corner_point, corner_point;  
AREA object_name POLYGONAL corner_point,... ;  
AREA object_name FUNCTIONAL f(x1,x2);
```

Arguments:

object_name - name of spatial object, user-defined.
corner_point - corner-point coordinates x1, x2 (m or rad).
f(x1,x2) - function of spatial coordinates, defined
in FUNCTION command.

Description:

The AREA command is used to specify the locations of areas in space. (The keywords, AREA and AREAS, are interchangeable — that is, they will produce the same result from the same data.)

The coordinate system used to define areas (see SYSTEM) can be changed at will for convenience in entering the spatial data. If no coordinate system is specified, then the default system will be in effect.

The arguments are defined as follows. The object_name is used to provide a unique label to an area or a group of areas. (All areas in a group will be treated in the same manner by subsequent commands which refer to this object_name.) Areas may be given the same object_name simply by issuing individual commands using the same object_name. In any subsequent operations on areas in a group, the areas will be treated in the order in which they are entered.

There are four shape options: RECTANGULAR, CONFORMAL, POLYGONAL, and FUNCTIONAL. The simplest option is RECTANGULAR, which requires only the specification of two opposing corner_points. This will produce a single (planar) rectangle, irrespective of the coordinate system (which serves only to define the corner_points).

For many applications, it is desirable for an area to “bend” so that the outline conforms with the local coordinates. This is easily accomplished using the CONFORMAL option. This option works in exactly the same way as the RECTANGULAR option, except that the shape

AREA Command

will be distorted to match the specified coordinate system. It is ideal for entering a pie-shaped area in polar or spherical coordinates, for example.

The POLYGONAL option can be used to create an area bounded by any number (≥ 3) of straight lines. Enter the corner_points defining the lines in order. Lines are not allowed to intersect (cross). The perimeter must be continuous: the first and last end-point must be the same to provide closure.

The FUNCTIONAL option can be used to create an area of arbitrary shape. To utilize it, you enter the name of a function which you have previously specified in a FUNCTION command. The shape of the area is determined according to the sign of the function, which can be evaluated through the spatial coordinates. The area exists wherever the sign is negative, and does not exist wherever the function is positive or vanishing (zero). Repeating, or periodic, structures are particularly simple to create with this option.

Restrictions:

In subsequent operations on a group of objects, areas will be treated in the order in which they are entered.

See Also:

POINT, Ch. 10
LINE, Ch. 10
SYSTEM, Ch. 10
LIST, Ch. 10
DELETE, Ch. 10

Examples:

In the examples which follow, it is assumed that the cartesian coordinate system is active unless otherwise noted.

(1) To create a rectangular area of width, W, and height, H, with one corner at the origin, use the following command:

```
AREA BOX RECTANGULAR 0,0 W,H ;
```

An equivalent alternative (in cartesian coordinates) is the following command:

```
AREA BOX CONFORMAL 0,0 W,H ;
```

AREA Command

(2) To create an annulus between R_a and R_b which subtends zero to $\pi/2$ in polar coordinates, use the following command:

```
SYSTEM POLAR ;  
Pi = 3.14158 ;  
AREA Annulus CONFORMAL Ra,0 Rb,Pi ;
```

(3) This example uses the FUNCTIONAL option to create a circular area of radius, R , about the origin in cartesian coordinates.

```
FUNCTION alfa (x,y) = + x*x + y*y - R*R ;  
AREA beta FUNCTIONAL alfa ;
```

The function is called alfa and is clearly negative only within a circle of radius, R . The created area will be named "beta." Equivalently, this same area could be created with the following commands:

```
Twopi = 2.0 * Pi ;  
SYSTEM POLAR ;  
AREA beta CONFORMAL 0,0 R,Twopi ;  
SYSTEM CARTESIAN ;
```

SYSTEM Command

Function: Specifies the coordinate system.

Syntax:
SYSTEM system ;

Arguments:

system	- coordinate system.
	= CARTESIAN, cartesian (x,y).
	= CYLINDRICAL, cylindrical (z,r).
	= POLAR, polar (r, ϕ)
	= SPHERICAL, spherical (r, θ ,).

Description:

The four coordinate systems described above are available. The default coordinate system is cartesian. The coordinate system can be changed at will to create new spatial objects in the input file. Once a coordinate system is specified, it (or the default) remains active until it is changed. The spatial grid will be set from and the simulation performed in the last coordinate system which is specified.

Restrictions:

1. The cylindrical (r, θ ,z) and spherical (r, θ , ϕ) coordinate systems cannot extend down to zero radius.
2. Axial symmetry boundary conditions are not automatically provided by the choice of coordinate system, but must be specified explicitly by the user.

See Also:

POINT, Ch. 10
LINE, Ch. 10
AREA, Ch. 10
AUTOGRID, Ch. 11
GRID, Ch. 11
SYMMETRY, Ch. 12

References:

B. Goplen, R. Worl, and R. E. Clark, "Simulations of the PBFA-II Voltage Adder," Mission Research Corporation Report, MRC/WDC-R-091, November 1984.

SYSTEM Command**Examples:**

In this example, the (default) cartesian system is assumed to be active. We will create some points in cartesian coordinates and then switch to a Polar system to actually create the grid and perform the rest of the simulation.

```
POINTS  bunch_of_points  Xa,Ya  Xb,Yb  Xc,Yc,  ...  ;  
SYSTEM  POLAR  ;
```

LIST Command

Function: Lists previously entered points, lines, areas, or markers.

Syntax:

LIST [object_name] ;

Arguments:

object_name - name of spatial object.
= user-defined in POINT, LINE, or AREA commands

Description:

The LIST command is used to list specified objects and associated markers from an input file. Typically, this file would be read in or created in the interactive mode. Use of a specific object_name allows restricting the list to specific objects or groups of objects. Otherwise, all objects and markers will be listed.

Restrictions:

See Also:

POINTS, Ch. 10
LINES, Ch. 10
AREAS, Ch. 10
MARK, Ch. 11

Examples:

- (1) We shall create two points, labeled a and b and then list all and create c.

```
POINT  a    Xa,Ya  ;  
POINT  b    Xb,Yb  ;  
LIST    ;  
POINT  c    Xc,Yc  ;
```

The listing will contain the two points labeled a and b, but not c.

- (2) We next mark point c and list all marker locations.

```
MARK    c    ;  
LIST    c    ;
```

Marker locations will be listed for point c, but not for points a and b, which were not marked.

DELETE Command

Function: Deletes previously entered objects or markers.

Syntax:
DELETE { OBJECT, MARK } object_name ;

Arguments:

object_name - name of existing spatial object.
= user-defined in POINT, LINE, or AREA commands.

Description:

Note — the DELETE command is presently inoperable.

The DELETE command is used to remove specified objects or markers from an input file. Typically, this file would be read in or created in the interactive mode. Note that DELETE MARK deletes only the markers, and not the object itself. DELETE OBJECT will delete both the object and any markers associated with it. Use of a specific object_name allows specific objects or markers to be deleted.

Restrictions:

See Also:

POINTS, Ch. 10
LINES, Ch. 10
AREAS, Ch. 10
MARK, Ch. 11

Examples:

(1) We shall create two points, labeled a and b and then delete b and create c.

```
POINT  a   Xa,Ya  ;  
POINT  b   Xb,Yb  ;  
DELETE OBJECT  b  ;  
POINT  c   Xc,Yc  ;
```

The final result is two points, labeled a and c.

11. GRIDS

This Section covers the following commands:

DURATION
TIMER
MARK
AUTOGRID
GRID ORIGIN
GRID EXPLICIT
GRID UNIFORM
GRID QUADRATIC
GRID PADE
GRID SIN / COS

You can use these commands to create grids in time and space. The **DURATION** command is used to specify the time span covered by the simulation. The **TIMER** command is used to specify trigger times for use by other commands. For example, you can use **TIMER** to specify when certain measurements are to be made.

To construct a spatial grid, you have two choices: the **AUTOGRID** command (automatic), and the **GRID** commands (manual). The **AUTOGRID** command is used in conjunction with **MARK** commands to generate the grid automatically. The **MARK** command sets markers and cell sizes at key locations on spatial objects (see Ch. 10). The **AUTOGRID** command uses this data to generate the grid. If you use the **MARK** and **AUTOGRID** commands, no other commands are necessary to construct a spatial grid.

You can also construct a spatial grid manually using **GRID** commands. The **GRID ORIGIN** command fixes the value of the coordinate at the origin. As an option, you may also enter the cell size at the origin. You use the other **GRID** commands to construct consecutive sections of the grid. The different options (**EXPLICIT**, **UNIFORM**, **QUADRATIC**, etc.) allow different functional variations for the cell size within each section. You can use any number and combination of **GRID** commands. The first command builds the first section, beginning from the origin. Each subsequent **GRID** command extends the grid by adding a new section of grid that is appended to the previous section.

The spatial grid index extends over the range

$$2 \leq i \leq ix_{\max}$$

where $ix_{\max}$ is chosen to meet the simulation requirements (subject to code limitations). To facilitate grid extension, the code automatically generates and updates a system variable, **ISYS\$InMX** ($n = 1$ and 2), containing the current number of defined grid points. This

variable may be referenced in other commands. Other system variables generated by the **AUTOGRID** and **GRID** commands include **SY\$XnMN** and **SY\$XnMX**, the initial and final full-grid points.

DURATION Command

Function: Specifies the time span for a time-dependent simulation.

Syntax:
DURATION time_span ;

Arguments:
time_span - simulation time span (sec).

Description:

The DURATION command is used to specify the time_span for a time-dependent simulation.

Two system variables are produced when this command is entered. The variable, ISYS\$KMAX, represents the total number of electromagnetic time steps in time_span. The variable, SYS\$RUNTIME, is equal to the time_span. Both of these system variables can be used in subsequent commands.

See Also:

MAXWELL, Ch. 17

TIMER Command

Function: Defines a time trigger.

Syntax:

```
TIMER timer_name PERIODIC { INTEGER, REAL }
    start_time [ stop_time [ time_increment ] ]
    [ INTEGRATE time_interval ] ;
```

(or)

```
TIMER timer_name DISCRETE { INTEGER, REAL }
    trigger_time1 [ trigger_time2, ... ]
    [ INTEGRATE time_interval ] ;
```

Arguments:

timer_name	- timer name. = arbitrary, user-defined.
start_time	- do-loop start time or time index.
stop_time	- do-loop stop time or time index (default = infinity).
time_increment	- do-loop time or time index increment (default = MAX(1,start_time)).
trigger_time1, ...	- discrete values of time or time indices.
time_interval	- time or time index interval for integration.

Description:

Many commands require instruction as to when they are to be exercised, or “triggered.” A simple example is the RANGE command, which produces a plot only when it is exercised. The TIMER command defines a time trigger, which is just a sequence of times. Once it is defined, a time trigger can be applied to any command requiring one (such as RANGE) simply by entering the timer_name.

There are two options, PERIODIC to enter a periodic trigger, and DISCRETE to enter an irregular trigger. With either one, you can specify REAL to enter times in seconds or INTEGER to enter time indices. (Time indices are used with the time_step (TIME_STEP, Ch. 17) to compute times in seconds.)

During the simulation, the PERIODIC option causes a do-loop to calculate trigger_times. A data entry is always required for start_time (which must be positive), but the data entries for stop_time and time_increment are optional. If data is not entered, then stop_time is set to infinity, and time_increment is set to unity or start_time, whichever is greater.

TIMER Command

The DISCRETE option allows the trigger_times to be entered explicitly. This option is useful when the trigger_times are few or irregularly spaced. The trigger_times must be entered in increasing order.

Many commands which require timers make physical measurements of fields, currents, etc. In such applications, the INTEGRATE option in TIMER will cause integration to be performed over a time_interval before the trigger_time. When the measurement is initiated (at trigger_time - time_interval), it will be made continuously (every time step) for the period of time_interval. Upon completion (trigger_time), the measurements cease, and the average value is calculated and output (see Examples, below).

Restrictions:

1. A single TIMER command with the DISCRETE option allows up to ten trigger_times to be entered.
2. In using the INTEGRATION option, you cannot allow measurements initiated by two trigger_times to overlap. In other words, the time_interval must not exceed the difference between the trigger_times.

See Also:

TIME_STEP, Ch. 17
RANGE, Ch. 23, and
other commands which require timers.

Examples:

1. The following example requests LINPRINT output of fields E1, B2, and B3 every 20 time steps starting on time index 10 and continuing through time index 90. (Printout will occur on time indices 10, 30, 50, 70, and 90. Printout will not occur on time index 100.)

```
TIMER   P_timer   PERIODIC   INTEGER 10, 100, 20 ;
LINPRINT   P_timer   E1 ...
LINPRINT   P_timer   B2 ...
LINPRINT   P_timer   B3 ...
```

2. The following example illustrates use of the INTEGRATE option. The measurement involves time-averaged gain as a function of axial position in an RF amplifier. This complex diagnostic is obtained using only three commands:

```
FUNCTION Gain(P_Poynting) =
    10.0 * LOG10 (P_Poynting / P_input ) ;
```

TIMER Command

```
TIMER Gain_timer PERIODIC
      INTEGER K_plot, Infinity, K_plot
      INTEGRATE K_period ;
RANGE Gain_timer
      1 POYNTING P1 Input_port Infinity
      TRANSFORM Gain ;
```

P_input is the input power, K_plot is the time_increment (time between plots), and K_period is the time_interval for integration (number of time steps in an RF period). Note that time_interval must be less than, or equal to, time_increment to prevent the integration measurements from overlapping and producing an error.

MARK Command

Function: Marks locations on spatial objects for automatic gridding.

Syntax:

```
MARK object_name
      [ { X1, X2 } [ MINIMUM ] [ MIDPOINT ] [ MAXIMUM ]
      [ SIZE cell_size ] ] ;
```

Arguments:

object_name - name of spatial object, defined in POINT, LINE, or AREA command.
cell_size - cell size at marked location (m or rad).

Description:

Spatial objects may be created either before or after the spatial grid is generated. However, there is an advantage in defining the objects first, because this information can be used to generate the grid automatically. To do this, you use the MARK command to mark key locations on a spatial object. The AUTOGRID command will then generate grid lines which conform to the object at the marked locations. Everywhere else, the (unmarked) spatial objects will be slightly distorted and/or shifted to conform to the grid. Thus, the MARK command can be used to preserve the exact size and shape of critical geometric features, such as a gap width, a conducting surface, a voltage integral, etc. Any type of spatial object (points, lines, and areas) may be marked using this command.

The arguments and options are as follows. The object_name must be previously defined in a POINT, LINE, or AREA command. If you simply mark the spatial object (simple form, with no options), the result will be four marked locations, i.e., the MINIMUM and MAXIMUM in both coordinates (X1 and X2). However, you can refine this by using the optional arguments. First, select the coordinate (X1 or X2). Next, you may identify critical locations, with the choices being MINIMUM, MIDPOINT, and MAXIMUM (or just MIN, MID, and MAX — MINIMUM and MAXIMUM refer to the extrema of the object, and MIDPOINT is their average value). You may choose none (blank) or any or all of these. If you do not specify, only the extrema will be marked. Finally, you may also specify the cell_size, which will be applied to all locations marked with that command.

When the AUTOGRID command is processed, all marked locations will be ranked in ascending order, and any duplications will be eliminated. Thus, there is no penalty associated with redundant specifications, and the MARK commands can be structured for convenience. If no MARK commands have been entered, AUTOGRID will search the entire list of spatial objects to determine the outer bounds of the simulation, and will treat these as marked locations. Any MARK commands entered after the AUTOGRID command will be ignored.

Marked locations are shown by arrowheads in the DISPLAY command.

MARK Command**Restrictions:**

1. A spatial object must be created before it can be marked.
2. The MARK commands should precede the (grid generation) AUTOGRID command. Any subsequent MARK commands will be ignored.

See Also:

POINT, Ch. 10
LINE, Ch. 10
AREA, Ch. 10
SYSTEM, Ch. 10
DELETE, Ch. 10
LIST, Ch. 10
AUTOGRID, Ch. 11
DISPLAY, Ch. 24

Examples:

1. To mark a point named "a" in both coordinates, the command is simply

```
MARK  a  ;
```

This would preserve this location in the grid to facilitate a precise field measurement, for example. (Note that this simple form of the MARK command always produces four marked locations; two of them are redundant for a point.)

2. To mark the minimum x2 location of an object named "anode," the command is

```
MARK  anode  X2  MINIMUM  ;
```

3. If we wanted to mark the minimum and maximum in x1 and to specify a cell_size of 0.0005 meters at both locations, the command could read

```
MARK  anode  X1  MIN  MAX  SIZE  0.0005  ;
```

or just

```
MARK  anode  X1  SIZE  0.0005  ;
```

4. To delete all marked locations on the anode, use the command

```
DELETE  MARK  anode  ;
```

MARK Command

5. To list all of the marked locations, enter the command

`LIST ;`

From the Examples above, only the point “a” marked locations would be listed, since those for the “anode” were deleted in Example 4.

AUTOGRID Command

Function: Generates the spatial grid automatically from spatial markers.

Syntax:
AUTOGRID [{ X1, X2 } [total_cells]] ;

Arguments:

total_cells - approximate number of cells desired (default = 100).

Description:

The AUTOGRID command is used to generate the spatial grid automatically. It makes use of previously-defined spatial markers (see MARK, Ch. 11). If the AUTOGRID command and MARK commands are used, no other gridding commands are needed.

There are few options in AUTOGRID, since most of the gridding logic is controlled by the spatial markers. You may choose to grid one coordinate automatically (e.g., AUTOGRID X1) and the other (X2) manually with GRID commands. Finally, if the spatial marker data is sparse, you may wish to enter the desired number of total_cells in that coordinate.

The AUTOGRID algorithm always places grid lines precisely on all of the markers. Between two adjacent markers, the cell size always obeys a Pade equation (see GRID PADE, Ch. 11). In general, the cell sizes will vary. Under certain circumstances, a uniform grid can result.

The algorithm logic depends upon the data that is available to it. To grid a region between two markers, the Pade prescription requires exactly three parameters, e.g., the distance between markers and the cell size at each marker. The algorithm calculates the distances between markers, and it uses the following approach to obtain cell sizes.

First, if the MARK commands have specified cell sizes at all of the markers, no more is required. If an interior marker cell size is missing, it will be calculated from available cell sizes by linear interpolation. If an exterior marker cell size is missing, it will be set equal to the nearest cell size (no extrapolation). If no cell sizes have been entered, then the total_cells and the total distance will be used to calculate the marker cell sizes.

Once all marker cell sizes have been calculated, they are adjusted to ensure an integer number of cells between markers. Finally, the grid itself is computed from Pade equations.

AUTOGRID Command

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product, i.e., the total number of cells.

See Also:

MARK, Ch. 11

GRID ORIGIN Command

Function: Creates an origin in the specified coordinate axis.

Syntax:

```
GRID ORIGIN { X1, X2 }  
            axis_origin [ cell_size [ grid_index ] ] ;
```

Arguments:

axis_origin	-	coordinate value in meters or radians (default = 0.0).
cell_size	-	cell size at axis_origin in meters or radians.
grid_index	-	optional grid index at axis_origin, integer (default = 2).

Description:

The GRID ORIGIN command is used to specify the origin in one of the coordinate axes. It should be entered before any of the GRID options are used to build sections of the grid.

If the axis_origin is not specified in a GRID command, the lowest value spatial marker is used. If there are no spatial markers, a value of zero will be used. The cell_size at the axis_origin can also be specified. If so, it may be used in building the first section of grid. The optional specification of grid_index provides control over the simulation position in index space; normally, it is not needed.

See Also:

GRID EXPLICIT, Ch. 11
GRID UNIFORM, Ch. 11
GRID QUADRATIC, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in cylindrical coordinates requires a grid between the inner and outer radii (R_inner and R_outer), but there is no reason to grid the region between the center axis and R_inner, since it is not used. Therefore, we can shift the radial axis origin to R_inner with the command

```
GRID ORIGIN X2 R_inner ;
```

GRID EXPLICIT Command

Function: Creates an arbitrarily varying grid in a region.

Syntax:

```
GRID EXPLICIT { X1, X2 } CELLS ncells  
[ SIZE cell_size, ... ]  
[ GRID full_grid, ... ] ;
```

Arguments:

ncells	-	number of cells in the region.
cell_size	-	cell size in meters or radians.
full_grid	-	grid values in meters or radians.

Description:

The EXPLICIT option allows an explicit sequence of data to be entered to create an arbitrary grid. Ncells must be specified. Then two options are available to specify the cell-by-cell data. The SIZE option allows a sequence of cell_size values to be entered. They are used internally to construct the grid. The GRID option allows full_grid values to be entered.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product, i.e., the total number of cells.

See Also:

GRID ORIGIN, Ch. 11
GRID UNIFORM, Ch. 11
GRID QUADRATIC, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

GRID UNIFORM Command

Function: Creates a uniform grid in a region.

Syntax:

```
GRID UNIFORM { X1, X2 }  
    [ DISTANCE region_size ]  
    [ FIRST { MATCH, cell_size } ]  
    [ CELLS ncells ] ;
```

Arguments:

region_size	-	length of the region in meters or radians.
cell_size	-	cell size of region in meters or radians.
ncells	-	number of cells in the region.

Description:

The UNIFORM option results in equally-spaced full-grid points given by

$$x_i^f = x_1^f + (i-1) \text{ cell\_size}$$

where cell_size can be entered directly or will be calculated from other parameters. Although a uniform grid can also be achieved using all of the other options, the UNIFORM option is the simplest way, and it reduces round-off errors in grid spacing.

The UNIFORM algorithm requires only two parameters to specify the grid completely. As indicated in the syntax, there are three options, and you must select two. If the parameters are either under- or over-specified, an error condition will result. The options are DISTANCE, FIRST, and CELLS. (The FIRST option refers to the first cell. If MATCH is entered, the algorithm will attempt to obtain the cell_size from the ORIGIN command or from the last cell in the preceding region. Otherwise, cell_size should be entered.)

If DISTANCE is one of the options specified, cell_size will be normalized to provide an exact integral number of cells, thus ensuring that grid lines fall precisely on the ends of the region_size.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product, i.e., the total number of cells.

GRID UNIFORM Command**See Also:**

GRID ORIGIN, Ch. 11
GRID EXPLICIT, Ch. 11
GRID QUADRATIC, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in cylindrical coordinates requires a grid between the inner and outer radii (R_{inner} and R_{outer}). We can shift the radial origin to R_{inner} and divide the gap into 20 uniform cells with the commands

```
GRID ORIGIN X2 R_inner ;  
gap = R_outer - R_inner ;  
Ncells = 20 ;  
dR = gap / Ncells ;  
GRID X2 UNIFORM DISTANCE gap FIRST dR ;
```

Note that the DISTANCE and FIRST options are used to satisfy the requirement for two parameters.

GRID QUADRATIC Command

Function: Creates a quadratic grid in a region.

Syntax:

```
GRID QUADRATIC { X1, X2 }
    [ DISTANCE region_size ]
    [ FIRST { MATCH, cell_size } ]
    [ LAST cell_size ]
    [ CELLS ncells ] ;
```

Arguments:

region_size - length of the region in meters or radians.
 cell_size - cell size in meters or radians.
 ncells - number of cells in the region.

Description:

The FUNCTIONAL option results in a quadratically-spaced grid over a region. The i th full-grid point x_i^f in a particular region is given by the formula,

$$x_i^f = x_1^f + (i - 1) \frac{(I^2 \delta x_1^f - d)}{I(I - 1)} + (i - 1)^2 \frac{(d - I \delta x_1^f)}{I(I - 1)}, \quad 1 \leq i \leq I + 1,$$

where d = the region_size, I = ncells, and δx_1^f = cell_size for the last cell. This means that the cell_size is allowed to increase or decrease uniformly by a constant amount

$$\delta x_{i+1}^f = \delta x_i^f \pm \epsilon,$$

where

$$\epsilon = \frac{2(d - I \delta x_1^f)}{I(I - 1)}.$$

The QUADRATIC algorithm requires only three parameters to specify the grid completely. As indicated in the syntax, there are four options, and you must select three. If the parameters are either under- or over-specified, an error condition will result. The options are DISTANCE, FIRST, LAST, and CELLS. (The FIRST option refers to the first cell. If MATCH is entered, the algorithm will attempt to obtain the cell_size from the ORIGIN command or from the last cell in the preceding region. Otherwise, cell_size should be entered.) If DISTANCE is one of the options specified, cell_size will be normalized to

GRID QUADRATIC Command

provide an exact integral number of cells, thus ensuring that grid lines fall precisely on the ends of the region_size.

There is a constraint on these parameters. The equation

$$d < I^2 \delta x < (2I - 1) d$$

must be satisfied for the grid to be monotonic. This constraint can be violated only by choosing extremely non-uniform parameters. An error message will result.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product, i.e., the total number of cells.
3. The grid must be monotonic: the parameter constraint described above must be satisfied.

See Also:

GRID ORIGIN, Ch. 11
GRID EXPLICIT, Ch. 11
GRID UNIFORM, Ch. 11
GRID PADE, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in cylindrical coordinates requires a grid between the inner and outer radii (R_inner and R_outer). We can shift the radial origin to R_inner and divide the gap into 20 quadratically varying cells, where the last cell is three times as large as the first cell, with the commands

```
GRID ORIGIN X2 R_inner ;
gap = R_outer - R_inner ;
Ncells = 20 ;
dR_ave = gap / Ncells ;
dR_first = 0.5 * dR_ave ;           ! dR_first +
    dR_last = 2 dR_ave.
GRID QUADRATIC X2
    DISTANCE gap FIRST dR_first CELLS Ncells ;
```

GRID QUADRATIC Command

Note that the DISTANCE, FIRST, and CELLS options are used to satisfy the requirement for three parameters.

GRID PADE Command

Function: Creates a Pade function grid in a region.

Syntax:

```
GRID PADE { X1, X2 }
          [ DISTANCE region_size ]
          [ FIRST { MATCH, cell_size } ]
          [ LAST cell_size ]
          [ CELLS ncells ] ;
```

Arguments:

region_size - length of the region in meters or radians.
 cell_size - cell size in meters or radians.
 ncells - number of cells in the region.

Description:

The PADE option results in a Pade function-spaced grid over a region. The i th full-grid point x_i^f in a particular region is given by the formula

$$x_i = (a + b i) / (c + d i) .$$

The use of the Pade functional prescription is recommended for a region requiring rapid variation in the cell_size.

The PADE algorithm requires only three parameters to specify the grid completely. As indicated in the syntax, there are four options, and you must select three. If the parameters are either under- or over-specified, an error condition will result. The options are DISTANCE, FIRST, LAST, and CELLS. (The FIRST option refers to the first cell. If MATCH is entered, the algorithm will attempt to obtain the cell_size from the ORIGIN command or from the last cell in the preceding region. Otherwise, cell_size should be entered.)

If DISTANCE is one of the options specified, cell_size will be normalized to provide an exact integral number of cells, thus ensuring that grid lines fall precisely on the ends of the region_size. (In the Pade prescription, the cell_size refers to the cell half-grid, whereas in all other options, it refers to full-grid.)

Restrictions:

- There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product, i.e., the total number of cells.

GRID PADE Command

See Also:

GRID ORIGIN, Ch. 11
GRID EXPLICIT, Ch. 11
GRID UNIFORM, Ch. 11
GRID QUADRATIC, Ch. 11
GRID SIN / COS, Ch. 11

Examples:

A simulation of a coaxial cable in cylindrical coordinates requires a grid between the inner and outer radii (R_{inner} and R_{outer}). We can shift the radial origin to R_{inner} and divide the gap into 20 Pade function cells, where the last cell is three times as large as the first cell, with the commands

```
GRID ORIGIN X2 R_inner ;
gap = R_outer - R_inner ;
Ncells = 20 ;
dR_ave = gap / Ncells ;
dR_first = 0.5 * dR_ave ;           ! dR_first
      + dR_last = 2 dR_ave.
GRID PADE X2
      DISTANCE gap FIRST dR_first CELLS Ncells ;
```

Note that the DISTANCE, FIRST, and CELLS options are used to satisfy the requirement for three parameters.

GRID SIN / COS Command

Function: Creates a trigonometrically varying grid in a region.

Syntax:

```
GRID { SIN, COS } { X1, X2 }
      [ DISTANCE region_size ]
      [ FIRST angle ]
      [ LAST angle ]
      [ CELLS ncells ] ;
```

Arguments:

region_size - effective radius of the region in meters.
 angle - trigonometric angle in the region.
 ncells - number of cells in the region.

Description:

The SIN / COS option results in a trigonometrically varying grid over a region. It is typically applied as a matched pair to both coordinates simultaneously. The advantage of this option is that a line passing through the corners of the two-dimensional cells can be made to resemble a circular or elliptical surface very closely. The arc lengths in each cell will be nearly equal. The full-grid point are given by

$$x_i = R [\sin (A_i + i \, dA) - \sin (A_i)]$$

$$y_i = R [\cos (A_i) - \cos (A_i + i \, dA)]$$

where $dA = (Af - Ai) / ncells$

and R is the region_size, Ai is the FIRST angle, and af is the LAST angle.

The SIN / COS algorithm requires exactly four parameters to specify the grid completely. As indicated in the syntax, there are four choices, so all are required. The region will be appended on to the existing grid in both coordinates; however, it will not accept cell-size data from either coordinate or from the origin.

Restrictions:

1. There is an upper limit to the number of spatial grid points in each coordinate.
2. There is a separate upper limit to the product, i.e., the total number of cells.

GRID SIN / COS Command

See Also:

GRID ORIGIN, Ch. 11
GRID EXPLICIT, Ch. 11
GRID UNIFORM, Ch. 11
GRID QUADRATIC, Ch. 11
GRID PADE, Ch. 11

12. OUTER BOUNDARIES

This Chapter covers the following commands:

SYMMETRY
PORT
OUTGOING
FREESPACE
MATCH
IMPORT

You can use these commands (along with conducting areas) to define the outer boundaries of the simulation. These are the only commands which can be used as outer boundaries. To use them effectively requires knowledge of a few simple conventions:

1. In general, outer boundaries are applied on lines. However, the **FREESPACE** command can be applied only on an area, and the downstream edge of this area is part of the outer boundary perimeter. (Similarly, the physical property, **CONDUCTOR**, applies only to an area.)
2. The outer boundary perimeter must completely enclose the simulation area. The perimeter can have any shape and be made up of any combination of outer boundaries and conductors. But it must not contain any "holes." Holes can cause serious errors which may not be readily observable. Rigid adherence to the closed perimeter convention is the best protection.
3. Some outer boundaries require a sense, or direction, as indicated by the syntax arguments, { **POSITIVE**, **NEGATIVE** }. The sense is always from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter **POSITIVE**. Otherwise, enter **NEGATIVE**.

The **SYMMETRY** commands provide axial, mirror, and periodic symmetry boundaries. You must apply all such boundaries explicitly, since none are assumed. Axial symmetry is required in cylindrical coordinates if, and only if, the simulation area extends to zero radius. Mirror symmetry is used to cut simulation costs in half, and periodic symmetry is used to study a limited region of a repetitive or re-entrant device.

Other boundaries allow outgoing waves to exit the simulation and, simultaneously, incoming waves to enter. The **PORT** command provides a good, general-purpose boundary and is used extensively. Its main drawback is the need to specify the phase velocities precisely. It is not a broad-band boundary, and it typically does not work well with multiple frequency, or broad spectrum waves. The **OUTGOING** command can also be used for outgoing and incoming waves. Its advantages over **PORT** are that phase velocities need not be specified, and that it tolerates a broader spectrum. However, it is more sensitive to instability.

The FREESPACE command only treats outgoing waves. It applies a special (non-physical) resistivity over an area (rather than a line). Its advantage is that it is very broad band and will handle a spectrum of phase velocities simultaneously. Its disadvantages are sensitivity to space charge and a possible need to tune the resistivity parameters.

There are also special-purpose boundaries. If external impedances (outside the perimeter) are important, you can use the MATCH command to match a transmission line to the simulation. (Transmission lines are discussed in the next Chapter.) The spatial line at which the match is made is considered part of the simulation perimeter. The IMPORT command can be used to introduce particles and fields from another simulation (EXPORT, Ch. 25).

SYMMETRY Command

Function: Applies outer boundaries for axial, mirror, and periodic symmetry.

Syntax:

```
SYMMETRY AXIAL line_name ;  
SYMMETRY MIRROR line_name ;  
SYMMETRY PERIODIC line_name line_name ;
```

Arguments:

line_name - name of a spatial line, defined in LINE command.

Description:

The SYMMETRY commands apply symmetry conditions on conformal lines which are part of the simulation perimeter. The symmetry boundaries can be applied in various combinations. They affect particle transformations as well as the electromagnetic fields.

The options available are AXIAL, MIRROR, and PERIODIC. AXIAL is needed if (and only if) the coordinate system is non-cartesian and contains the region of zero radius (or $\theta = 0$ or π in spherical coordinates). MIRROR, as the name suggests, replicates bilateral symmetry, which means simply that whatever occurs in the simulation space also occurs simultaneously in the virtual space opposite the boundary. PERIODIC is used to effect a repetitive structure or to close polar coordinates when the full range of polar angle (2π) is required. Note that PERIODIC requires two distinct spatial lines.

Restrictions:

1. The spatial line where the boundary condition is applied must be conformal.
2. The symmetry boundaries must be compatible with the models used in the simulation to retain physical fidelity. For example, the axial boundary must be used only at zero radius, the mirror and periodic boundaries should only be used on flat (never curved) surfaces, the periodic boundaries must appear on opposite sides, and so on.
3. The algorithm logic assumes conventional use of these boundaries, e.g., mirror symmetry near the edge (not the middle) of the simulation, axial symmetry at zero radius, etc. If you attempt unconventional uses, please be aware that the algorithm logic may thwart your intent.
4. Axial symmetry is not currently available in polar coordinates. Hence, polar coordinate simulations may not contain the origin. Axial symmetry is available in spherical coordinates, which may not contain the origin.

SYMMETRY Command

5. SYMMETRY AXIAL results in a modified Courant condition, in which the first radial cell appears to be smaller than it actually is. The effective size is

$$\sqrt{2/3} \delta r \equiv 0.82 \delta r$$

References:

B. Goplen, R. S. Coats, and J. R. Freeman, "Three-Dimensional Simulation of the PROTO-II Convolution," Mission Research Corporation Report, MRC/WDC-R-055, presented at the 4th IEEE Pulsed Power Conference, June 6-8, 1983.

Examples:

A simulation of a six-vane, relativistic magnetron (A6) is performed using the polar (r,θ) coordinate system. The SYMMETRY command is used to establish periodic boundaries which allow particles and fields to be re-entrant from one edge of the simulation perimeter to the other. The figure on the following page shows a trajectory plot at time 0.2 ns. The periodic boundaries are shown with a thick dashed line. Note that Figure 12-1 is drawn in a cartesian coordinate system. Consequently, the periodic boundaries appear to overlap.

```
topi = 2 * 3.14159 ;  
LINE Line_at_0 CONFORMAL R_cathode,0 R_anode,0 ;  
LINE Line_at_2pi CONFORMAL R_cathode,topi R_anode,topi ;  
SYMMETRY PERIODIC Line_at_0 Line_at _2pi ;
```


SYMMETRY Command

MAGIC VERSION: JUNE 1989 DATE: 07/20/89
SIMULATION: COUPLED MAGNETRON TEST

TRAJECTORY PLOT OF IONS (ISPE = 2)
FROM TIME 1.992E-10 SEC TO 2.000E-10 SEC

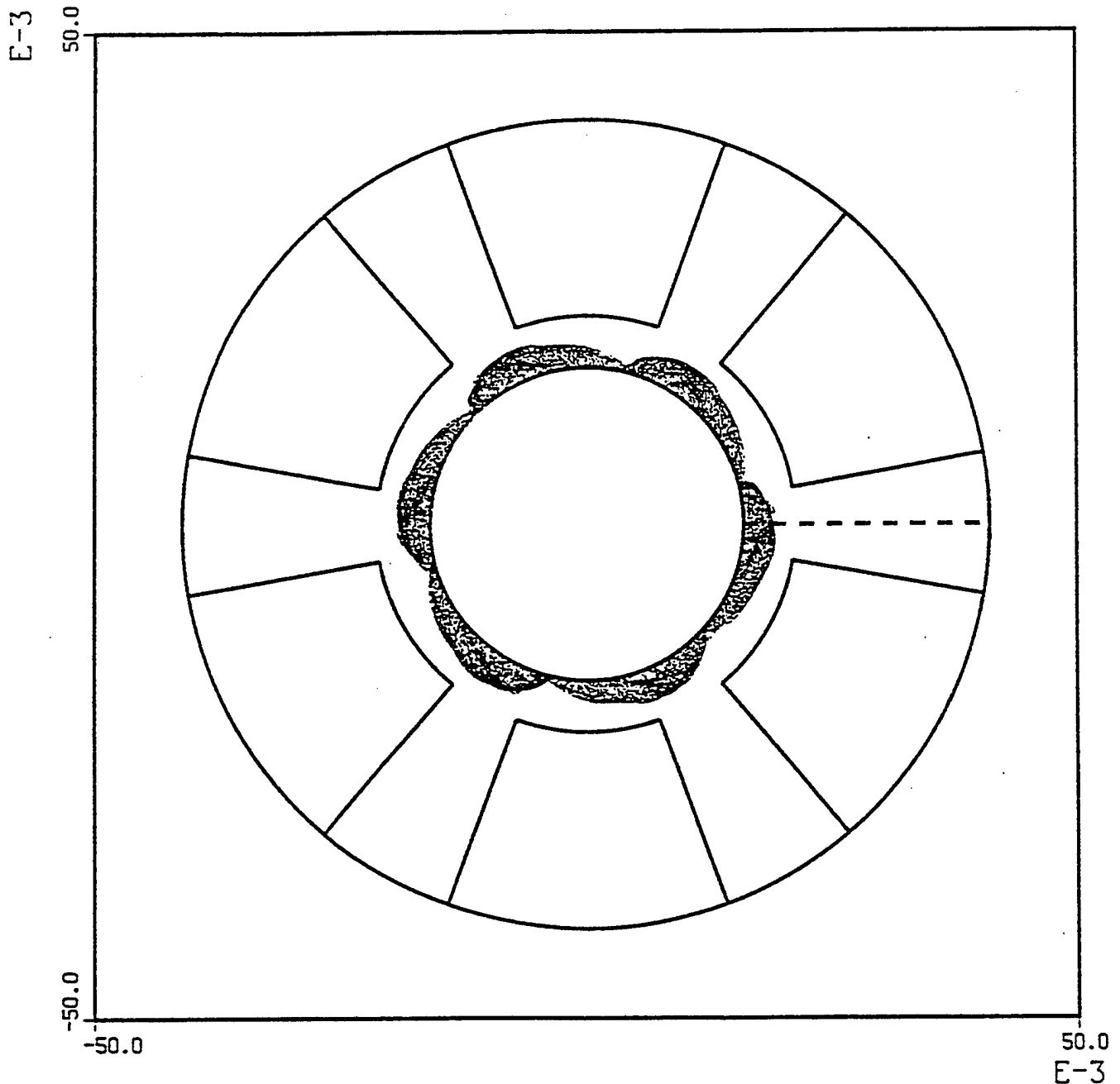


Figure 12-1. Re-entrance in the A6 magnetron.

PORT Command

Function: Applies an outer boundary for outgoing (and incoming) waves.

Syntax:

```
PORT line_name { POSITIVE, NEGATIVE }
    [ TE phase_velocity [ INCOMING f(t) g(x) ] ]
    [ TM phase_velocity [ INCOMING f(t) g(x)
        [ VOLTAGE [ CIRCUIT time_constant ] ] ] ]
    [ EXPANSION scale ] ;
```

Arguments:

- line_name - name of a spatial line, defined in LINE command.
- phase_velocity - relative phase velocity, v/c (unitless).
- f(t) - temporal function for incoming wave (V or V/m), defined in FUNCTION command.
- g(x) - spatial-profile function for incoming wave (arbitrary units), defined in FUNCTION command.
- time_constant - circuit time constant (sec).
- scale - geometric scaling factor (unitless, default is unity)

Description:

The PORT command specifies a boundary which allows outgoing waves and particles to escape and incoming waves to enter. You can apply it on any spatial line which is a conformal part of the simulation perimeter. The typical simulation produces outgoing waves due to scattering from geometric irregularities and plasma interactions within the simulation perimeter. You may also define arbitrary incoming waves. Thus, incoming waves, outgoing waves, and charged particles may be incident upon the boundary.

First, enter line_name to specify where the boundary is applied. The line must be conformal with one coordinate and part of the simulation perimeter. You must also enter the sense of the line, which is always from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter POSITIVE. Otherwise, you must enter NEGATIVE.

The algorithm uses the specified phase_velocity to allow outgoing fields to exit and to separate the incoming and outgoing fields. You should specify a phase_velocity for modes (TE and/or TM) present in the simulation. The default of unity is suitable only for TEM waves in vacuum (TEM is a special case of TM). The performance of this algorithm depends critically on

PORT Command

matching the actual phase velocities. For example, the reflection coefficient for an outgoing wave with phase velocity, v , is

$$\rho = \left(\frac{v - v_{port}}{v + v_{port}} \right)^2$$

which gives perfect transmission only if the velocities match. If multiple phase velocities are present in a single electromagnetic mode, inaccuracy is unavoidable.

The **INCOMING** keyword provides the option to specify incoming waves, i.e., waves incident on the boundary from outside the perimeter. In this case, the total field in the boundary is the sum of the incoming and outgoing fields, which we write in the form

$$E(x,t) = E_i(x,t) + E_o(x,t) = f(t)g(x) + E_o(x,t)$$

For the incoming wave, you must specify the temporal function, $f(t)$, and the spatial profile function, $g(x)$, where x is the coordinate transverse to the direction of propagation (parallel to the boundary). (For the TM mode, if you enter **VOLTAGE**, $g(x)$ will be re-normalized so its integral over the boundary gives unity. Then $f(t)$ will carry units of volts.)

In general, discontinuities in f and its first derivative should be avoided. For example, to launch a sinusoidal wave, we recommend multiplying the sine wave by an envelope which goes to unity in a few cycles (see Examples, below). The spatial profile function, $g(x)$, for a TEM wave is usually simple, e.g., unity for cartesian simulations and $1/r$ for axial propagation in cylindrical simulations. For non-TEM waves, $g(x)$ is generally more complex; e.g., Bessel functions for cylindrical waveguide modes.

The **PORT** command can easily be used to produce the so-called “port approximation,” which can be useful in linear beam and other applications. If you set the `phase_velocity` to zero, the scattered wave will be forced to vanish at the boundary. Then, the total field in the boundary will be just the incoming field, $f(t) g(x)$. In employing this approximation, it is good practice to obtain the spatial profile, $g(x)$, from a separate simulation using the same geometry (assuming it is unknown). It may also be sensible to modulate the temporal function, $f(t)$, as described above.

The **CIRCUIT** option can be used only with **TM ... VOLTAGE** to provide a feedback loop. It causes the total field to equal the specified incoming field. As the equation above shows, they differ by the outgoing component. There are two ways to cause them to be equal. One is to set the `phase_velocity` to zero, which forces the outgoing field at the boundary to zero, thereby trapping it within the simulation with possible deleterious effects. The other way, using **CIRCUIT**, filters the difference between the total and incoming waves and adds it to the incoming wave, so that the sum of the circuit, incoming, and outgoing waves approaches the value specified for the

PORT Command

incoming wave. The `time_constant` in the circuit filter should be a few time steps for fast response. The `CIRCUIT` capability is similar to that provided by the feedback loop (Part 2: `FUNCTION`).

The `EXPANSION` option is used in special cases in which the cross-sectional area in the direction of propagation grows (or shrinks), e.g., radial propagation in a conical coax. In most applications, e.g., axial propagation in a cylindrical coax, the area does not change with distance. However, if the area does change, the fields must change also. Scale multiplies fields at the boundary (x) to produce the correct magnitude one cell inside the perimeter ($x-dx$). For example, for a radially outgoing boundary in polar coordinates, scale would be set to $R/(R-dR)$, since the fields inside the boundary are compressed. This accounts for changing area but not changing impedance. Thus, it works well for radial propagation in spherical coordinates (a conical coax) but not in cylindrical or polar, where continuous scattering occurs. We recommend avoiding such applications if possible. Sometimes, a “dog-leg” section can be added so that the PORT boundary is applied in a region of constant impedance. If this is not possible, scale can sometimes be varied in an ad hoc manner to achieve acceptable results.

Restrictions:

1. The spatial line where the boundary condition is applied must be conformal.
2. The electromagnetic time step must be less than the cell size (in the direction of propagation) divided by phase_velocity times c for Courant stability, $\delta t \leq \delta x / \beta c$.
3. The value of $f(t)$ and its first derivative at $t = 0$ should vanish. (Before $t = 0$, the effective value of $f(t)$ is zero, regardless of the function's value.)
4. The spatial profile function, $g(x)$, must be correct for the geometry and the specified incoming wave. Artificial scattering will occur at the boundary if $g(x)$ is incorrect.
5. Avoid close proximity to structural singularities, e.g., corners, which produce fringe fields in all directions.
6. Avoid close proximity to particles, which can distort phase velocities and cause artificial field reflections.
7. Avoid applications involving impedance change. The pre-eminent example is the radially outgoing wave in cylindrical coordinates. We would recommend building an elbow and letting the wave out axially. If this is not possible, then use scale, but test the results!

PORT Command**See Also:**

MAXWELL, Ch. 17
OUTGOING, Ch. 12
FUNCTION, Ch. 6

References:

B. Goplen, "Boundary Conditions for MAGIC," Mission Research Corporation Report, MRC/WDC-R-019, 23rd Annual Meeting, APS Division of Plasma Physics, 12-16 October, 1981.

B. Goplen, R. S. Coats, and J. R. Freeman "Three-Dimensional Simulation of the PROTO-II Convolution," Mission Research Corporation Report, MRC/WDC-R-055, 4th IEEE Pulsed Power Conference, June 6-8, 1983.

Examples:

In this example, the incoming wave is a sinusoidally varying, one-volt, TEM wave, applied to the "inlet" at the left end of a coaxial cable with radii, R_cathode and R_anode.

```
LINE inlet STRAIGHT 0,R_cathode 0,R_anode ;  
FUNCTION g(r) = 1 / r ;  
FUNCTION f(t) = SIN (omega*t ) * ( 1.0 - EXP( - omega*t/5 ) ;  
PORT inlet POSITIVE TM 1.0 INCOMING f g VOLTAGE ;
```

OUTGOING Command

Function: Applies an outer boundary for outgoing (and incoming) waves.

Syntax:

```
OUTGOING line_name { POSITIVE, NEGATIVE }  
    { TE, TM, ALL }  
    [ INCOMING f(t) g(x) ]  
    [ ORDER norder ]  
    [ COEFFICIENT alpha_1,beta_1, ... ] ;
```

Arguments:

line_name	-	name of spatial line, defined in LINE command.
f(t)	-	temporal function for incoming wave (V or V/m), defined in FUNCTION command.
g(x)	-	spatial-profile function for incoming wave (arbitrary units), defined in FUNCTION command.
order	-	approximation order (default = 3).
alpha_n	-	nth alpha coefficient.
beta_n	-	nth beta coefficient.

Description:

The OUTGOING command specifies an outlet boundary condition which absorbs any waves which are propagating outward from the simulation through the boundary. It is a generalization of the PORT command and operates in a similar manner, except that OUTGOING boundaries can absorb waves of unknown phase velocity, and indeed, different phase velocities simultaneously.

First, enter line_name to specify where the boundary is applied. Line_name must be conformal with one coordinate and part of the simulation perimeter. You must also enter the sense of the line, which is always from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter POSITIVE. Otherwise, you must enter NEGATIVE.

It is anticipated that the OUTGOING boundary will eventually have a robust capability to launch an incoming wave with the INCOMING option. A capability is currently included which may be satisfactory for some applications. However, users are cautioned to use this option with skepticism, since it is not fully debugged. Note that if the INCOMING option is used, then either TM or TE (but not ALL) must be specified for the mode. It is possible to have two OUTGOING boundaries, one TE and one TM, at the same line_name.

Particles may exit through an OUTGOING boundary. The physical validity of this process is still under investigation; however, it is believed that the OUTGOING boundary provides a satisfactory treatment in this situation.

OUTGOING Command

This boundary condition is an exceptional absorber at near speed-of-light phase velocities (normal incidence). With the ORDER option, the order controls the effectiveness of the boundary at near grazing incidence, with a higher order, resulting in better absorption at shallower angles of incidence. For example, the default order (3) results in 2% field amplitude reflection at 20 degrees incidence, 6% at 10 degrees, and 10% at 5 degrees, while at order 5, the reflection is only 0.3% at 20 degrees, 2% at 10 degrees, and 5% at 5 degrees. In general, the default order should be quite satisfactory for most applications; however, if a simulation is known to possess a significant amount of its RF power in shallow incidence waves, then a higher order can be used. With machine single precision (32 bits), there is no benefit in using an order greater than about 9.

The COEFFICIENT option allows the user to specify non-standard coefficients for the boundary, and it is generally reserved for study of the properties of the boundary itself. Under normal circumstances, this option should not be used.

Restrictions:

1. The spatial line where the boundary condition is applied must be conformal with one of the spatial coordinates.
2. The maximum number of OUTGOING boundaries is 25.
3. The INCOMING option cannot be used with the mode set to ALL.
4. The OUTGOING boundary may touch other spatial objects (outer boundaries or material properties) only at its endpoints. It cannot cross or overlap another spatial object which has material properties (e.g., CONDUCTOR, POLARIZER, etc.). In such applications, two separate OUTGOING boundaries must be applied.

See Also:

MAXWELL, Ch. 17
PORT, Ch. 12
FUNCTION, Ch. 6

References:

"Free-Space Boundary Conditions for the Time Dependent Wave Equations," E. L. Lindman, *Journal of Computational Physics*, 18 (1975), pg. 66.

"Higher Order Absorbing Boundary Conditions for the Finite-Difference Time-Domain Method," P. A. Tirkas, C. A. Balanis, and R. A. Renaut, *IEEE Transactions on Antennas and Propagation*, 40 (1992), pg. 1215.

OUTGOING Command

Examples:

This is an example of an OUTGOING boundary used to absorb the wave propagating along a helix TWT. Note that two separate spatial applications (above and below the helix) are required. (See Restrictions, above.)

```
SYSTEM CYLINDRICAL ;  
LINE axis STRAIGHT 0,0 L,0 ;  
SYMMETRY AXIAL axis ;  
LINE helix STRAIGHT 0,Rh L,Rh ;  
POLARIZER HELIX 45. helix ;  
AREA anode RECT 0,Ra L,Ra+dR ;  
CONDUCTOR anode ;  
LINE below STRAIGHT L,0 L,Rh ;  
LINE above STRAIGHT L,Rh L,Ra ;  
OUTGOING below NEGATIVE ALL ;  
OUTGOING above NEGATIVE ALL ;
```


FREESPACE Command

Function: Applies an outer boundary for outgoing waves.

Syntax:

```
FREESPACE area_name { POSITIVE, NEGATIVE } { X1, X2 }
           { TE, TM, ALL, E1, E2, ..., B3 }
           [ CONDUCTIVITY f(x) ] ;
```

Arguments:

area_name - name of spatial area, defined in AREA command.
 f(x) - name of conductivity function (mhos/m), defined in a FUNCTION command.

Description:

The FREESPACE command allows outgoing waves to escape from the simulation. It is similar in purpose to the PORT and OUTGOING commands. However, whereas PORT and OUTGOING are applied along a line (boundary) in the simulation, FREESPACE is applied over a finite area. The downstream edge of this area is part of the simulation perimeter.

First, enter area_name, which defines the area of application. You must also enter the sense, which is always from outside the perimeter to inside, along a specified axis. If this is in the positive direction (increasing coordinate), enter POSITIVE. Otherwise, enter NEGATIVE. The axis (X1 or X2) specifies the axis along which propagation occurs.

The FREESPACE algorithm applies a spatially-varying conductivity to both electric and magnetic components in the field equations. This reduces the phase shift between components as both are degraded, minimizing reflection. You must specify the field components to which conductivity will be applied. In addition to the electromagnetic modes (TE, TM, or ALL), you may also enter individual field components (E1, E2, ..., B3). In this case, a separate FREESPACE command must be given for each desired component and care must be exercised to avoid duplication (e.g., issuing commands for TM and E1 fields). One reason to allow multiple FREESPACE commands on the same spatial region is to be able to use different conductivity functions on different field components. For example, a special treatment of the longitudinal electric field may be used to relax space-charge fields if particles penetrate the boundary area.

The default conductivity function increases quadratically with distance in the direction of the outgoing wave, or

$$f(x_n) = \sigma_m x_n^2,$$

where the spatial coordinate, x_n , is normalized to a total length of unity over the distance specified by the spatial area. The proper "zero" location, orientation, and spatial scaling are accounted for automatically by the code, so that the minimum value of conductivity (zero) will

FREESPACE Command

be applied at the upstream edge of the FREESPACE area, and the maximum conductivity (σ_m) will be applied at the downstream edge, which is part of the simulation perimeter. (Fields actually in the perimeter will vanish.)

The default value for the maximum conductivity, σ_m , is determined from the equation,

$$\sigma_m = \frac{4\pi c \epsilon_0}{\lambda} = 4\pi \epsilon_0 f = 2\epsilon_0 \omega,$$

where λ is estimated from the distance. (It is assumed that the model will be employed over a spatial distance equal to about half the wavelength.)

The FREESPACE algorithm has a very broad bandwidth, and results are usually insensitive to the detail of the conductivity function. However, you can override the default function and enter your own function by using the CONDUCTIVITY option.

Restrictions:

1. The maximum number of FREESPACE commands is twelve.
2. The boundary should encompass at least half a wavelength in distance and at least ten (but no more than 40) cells in the spatial grid.

See Also:

FUNCTION, Ch. 6
PORT, Ch. 12
OUTGOING, Ch. 12

Examples:

We wish to absorb the outgoing wave at the outlet end of a coaxial cable which is subjected to an incident voltage pulse at the inlet. For the sake of illustration, the default (quadratic) conductivity will be replaced by a linear conductivity having a maximum value of 10^{-3} mhos/m. This is illustrated in the following commands:

```
FUNCTION SIGMA(X) = 1.0E-3 * X ;
FREESPACE Area_free NEGATIVE X1
      TM CONDUCTIVITY SIGMA ;
```

In this example, the freespace boundary is applied over Area_free. Note that the normalized spatial variable in the conductivity function will vary from zero to one over the axial length of this area, irrespective of its actual physical length.

MATCH Command

Function: Matches an outer boundary to a transmission line.

Syntax:

```
MATCH line_name { POSITIVE, NEGATIVE } point_name  
      transmission_line point { POSITIVE, NEGATIVE } ;
```

Arguments:

line_name	-	name of spatial line for the voltage match, defined in LINE command.
point_name	-	NULL, or the name of a spatial point for the current match, defined in POINT command.
transmission_line	-	name of a transmission line, defined in TRAMLINE command.
point	-	transmission line point for voltage match.

Description:

The MATCH command connects a transmission line to the simulation. Specifically, the transmission line current and voltage are coupled with the TM fields (E1, E2, B3) of the simulation.

The spatial convention for the linear match is illustrated in Figure 12-2. First, you enter a spatial line describing the integral path for voltage. You must also enter the direction from outside the perimeter to inside. If this is in the positive direction (increasing coordinate), enter POSITIVE. Otherwise, enter NEGATIVE. The point_name determines the type of match between the transmission line current and the magnetic field. If you enter NULL, an average of the magnetic field will automatically be performed over a path adjacent to the voltage integral. Alternatively, you may use the magnetic field at a single point by entering a point_name. You must also identify the transmission line and specify its end point and its direction from outside to inside, either POSITIVE or NEGATIVE. The cell size of the transmission line and the perpendicular cell size of the simulation should be the same at the match.

Restrictions:

1. All transmission lines must be specified using TRAMLINE commands.
2. Primary and secondary cell sizes must be identical at the match point (in the direction of propagation).
3. The maximum number of MATCH commands is ten.

MATCH Command**See Also:****TRAMLINE, Ch. 13****JOIN, Ch. 13****GRID, Ch. 11****Examples:**

Consider two coaxial, six-vane magnetrons coupled by a transmission line which runs from a vane slot on the first magnetron to a vane slot of the second. The following commands are used to define the transmission line to match it to the simulation. The transmission line, **CONNECT**, is defined using the impedance option. The impedance is set to 5.4 ohms, and the relative dielectric constant is set to unity. The line is 1 meter in length. One **MATCH** command connects the transmission line to the first magnetron, and the other **MATCH** command connects the transmission line to the second magnetron.

```
TRAMLINE CONNECT IMPEDANCE 5.4 1.0
      FUNCTION 101 1 0.0 100 1.068E-3 10.68E-2 ;
MATCH Line_ab NEGATIVE Point_c CONNECT 0.0 POSITIVE ;
MATCH Line_ef NEGATIVE Point_g CONNECT 1.0 NEGATIVE ;
```

It is possible to simulate two complete (2π) magnetrons simultaneously using an extended azimuthal grid ($0-4\pi$) and two sets of periodic symmetry boundary conditions. The resulting magnetrons can be completely independent, or they can be coupled as indicated above.

MATCH Command

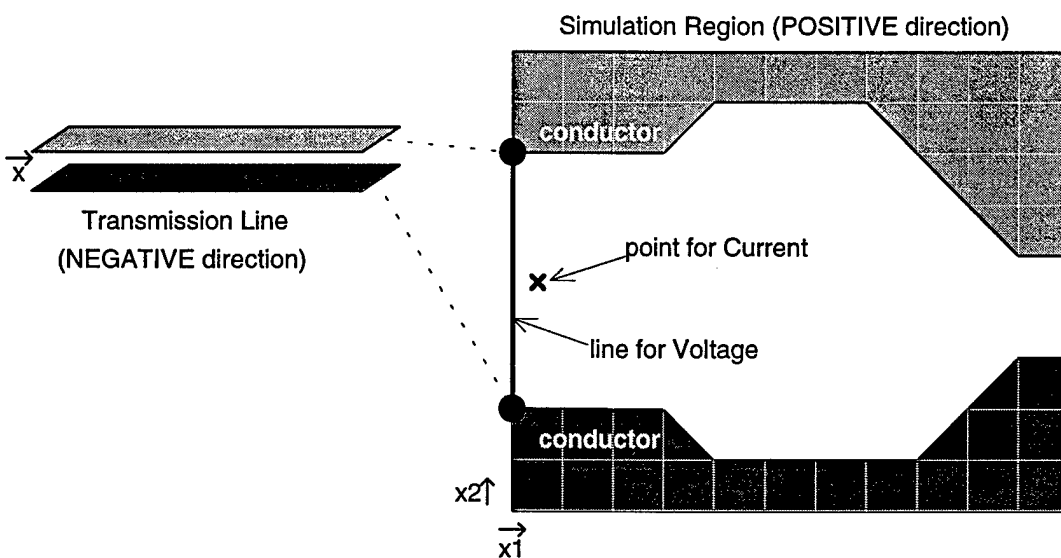


Figure 12-2. Matching to a transmission line.

IMPORT Command

Function: Applies an outer boundary for imported particles and fields.

Syntax:

```
IMPORT line_name { POSITIVE, NEGATIVE }  
    file_prefix file_format [ field ] [ field ] species  
    [ SCALE scale(t) ]  
    [ VELOCITY phase_velocity ]  
    [ OFFSET axial_offset ]  
    [ IGNORE-Z ]  
    [ REWIND-AT-EOF cycles ]  
    [ RECORD-AT-EOF ]  
    [ TERMINATE-AT-EOF ] ;
```

Arguments:

line_name	-	name of spatial line, defined in LINE command.
file_prefix	-	prefix of file from which data is read.
file_format	-	file format, ASCII or BINARY.
field	-	field to import (see table in EXPORT command).
species	-	species to import (see SPECIES command).
scale	-	field and current scaling factor, real or defined in FUNCTION command (default = unity).
phase_velocity	-	relative phase-velocity, v/c (unitless, default = 0).
axial_offset	-	axial position offset for import plane (m).
cycles	-	number of times to rewind (integer).

Description:

The IMPORT command defines an outer boundary for incoming fields and particles which were produced by a previous simulation. Outgoing waves may also be allowed to escape.

The import boundary is applied to line_name, which must be conformal with one of the axes and part of the simulation perimeter. The sense is always from outside the perimeter to inside. (This is also the direction of the incoming particles and wave.) If this is in the direction of increasing coordinate, enter POSITIVE; otherwise, enter NEGATIVE.

The file_prefix label is applied to two files: one for particles and one for fields. The precise file specification is platform-dependent. For example, on the VAX, file_prefix

IMPORT Command

specifies `file_prefixGRD` and `file_prefixPAR`, whereas on UNIX-based platforms, the `file_prefix` must include a version number; e.g., `EXPO.01` specifies `EXPOGRD.01` and `EXPOPAR.01`. The `file_format` is either ASCII or binary. The `field` and `species` select which fields and species of particles are to be imported. Up to two field components may be specified.

The keyword, `SCALE`, and `scale` may be used to introduce an additional temporal scaling of the imported field amplitude and particle current. This can be used to introduce a full scale result more gently into the empty simulation. The default for `scale` is unity.

The keyword, `VELOCITY`, may be used to set the value of `phase_velocity`. The `IMPORT` command is similar the `PORT` command in that outgoing waves may be allowed to escape. If a non-zero `phase_velocity` is entered, outgoing waves may escape, and the total field will be the sum of the incoming (imported) and outgoing fields. If zero is entered, the boundary fields will be forced to the import values, and any outgoing waves will be reflected.

Normally, an `EXPORT` file from a `MAGIC` simulation will write particle coordinates relative to the export boundary. However, to accommodate arbitrary placement of the boundary, the `OFFSET` option allows specification of an `axial_offset`, which will be added to the particle coordinates as they are imported.

The `IGNORE-Z` keyword causes the particle coordinates to be overwritten with the those of the `IMPORT` boundary. This provides a more forceful means of importing particles for which the recorded axial coordinate does not match the simulation. However, the longitudinal spacing between particles is lost when this option is used. The `OFFSET` keyword may be used in conjunction with `IGNORE-Z` to start all particles at a distance, `axial_offset`, inside the import boundary. In this case, note that `axial_offset` must have the correct sign and a fraction of one cell width.

The keywords, `REWIND-AT-EOF`, `RECORD-AT-EOF`, and `TERMINATE-AT-EOF`, control the action taken when an end-of-file is encountered. `REWIND-AT-EOF` is used to repeat the file cycle times. `RECORD-AT-EOF` is used to force generation of a `RESTART` file synchronized with the EOF. `TERMINATE-AT-EOF` is used to terminate the simulation when the import data has been exhausted.

Restrictions:

1. Only one `IMPORT` command is allowed in a simulation.
2. The import boundary geometry should be consistent with the export geometry. The spatial grids and time steps do not have to match, although it is not a bad idea to do so.

IMPORT Command**See Also:****EXPORT, Ch. 25****DUMP, Ch. 25****PORT, Ch. 12****Examples:**

In the following case, an **IMPORT** command is used to read both TM and TE components of the electric field as well as particles moving in the positive z-direction through a spatial line named "inlet." The coordinate system is cylindrical.

```
IMPORT inlet POSITIVE
EXP'icase' ASCII "ERHO(RHO)" "EPHI(RHO)" ELECTRON
OFFSET deltaz IGNORE-Z ;
```


13. TRANSMISSION LINES

This Chapter covers the following commands:

TRAMLINE
JOIN
LOOKBACK
VOLTAGE

You can use these commands to create one-dimensional transmission lines, to join them together, to allow outgoing waves to escape, and to allow incoming waves to enter.

The TRAMLINE command is used to create a transmission line. There are various options available to specify the properties of the line, which may be functions of distance; but variations in time are not allowed, nor are plasma processes. The spatial grid options duplicate the multi-dimensional options (GRID, Ch. 11).

You can use the JOIN command to connect different transmission lines together to form linear, series, and parallel circuits. The LOOKBACK command is used to allow outgoing waves to escape at a specified boundary of a transmission line. The VOLTAGE command is used to allow a specified incoming wave to enter (and outgoing waves to escape) through a specified boundary.

Although transmission lines can be modeled separately, their most important use is to extend the two-dimensional simulation into a plasma-free region of space. (The MATCH command, which connects them, is described in Chapter 12.) The solution algorithm and time step for the transmission line will automatically match those of the two-dimensional simulation.

This page is intentionally left blank.

TRAMLINE Command

Function: Specifies transmission-line parameters.

Syntax:

```
TRAMLINE transmission_line

{ RADII r_outer(x) r_inner(x) [resistance(x)] ,
  IMPEDANCE impedance(x) permittivity(x) [resistance(x)] ,
  CAPACITANCE capacitance(x) inductance(x) [resistance(x)] }

{ EXPLICIT CELLS ncells
  [ SIZE cell_size, ... ]
  [ GRID full_grid, ... ] ,
  UNIFORM
  [ DISTANCE region_size ]
  [ FIRST { MATCH, cell_size } ]
  [ CELLS ncells ] ,
  QUADRATIC
  [ DISTANCE region_size ]
  [ FIRST { MATCH, cell_size } ]
  [ LAST cell_size ]
  [ CELLS ncells ] }

[ INITIALIZE voltage(x) current(x) ] ;
```

Arguments:

transmission_line	-	transmission line (alpha).
r_outer	-	outer radius of vacuum coax (m), constant or defined in FUNCTION command.
r_inner	-	inner radius of vacuum coax (m), constant or defined in FUNCTION command.
resistance	-	resistance per unit length (ohms/m), constant or defined in FUNCTION command.
impedance	-	impedance for arbitrary geometry (ohms), constant or defined in FUNCTION command.
permittivity	-	relative dielectric constant (unitless) constant or defined in FUNCTION command.
capacitance	-	capacitance per unit length (f/m), constant or defined in FUNCTION command.
inductance	-	inductance per unit length (h/m), constant or defined in FUNCTION command.

TRAMLINE Command

ncells	-	number of cells in the region.
cell_size	-	cell size in meters.
region_size	-	length of the region in meters.
full_grid	-	grid values in meters or radians.
voltage	-	initial voltage distribution in volts, constant or defined in FUNCTION command.
current	-	initial current distribution in Amperes, constant or defined in FUNCTION command.

Description:

The TRAMLINE command specifies parameters for transmission line models. The assumption is that regions so represented are devoid of plasma, so that impedance properties may be specified precisely. It is possible to match transmission lines to the two-dimensional simulation. In addition, it is possible to join transmission lines together to form linear, parallel, and series circuits. Boundary conditions may be applied to transmission lines using the LOOKBACK and VOLTAGE commands.

The TRAMLINE command specifies parameters defining the transmission line length, spatial resolution, and basic physical properties. The transmission_line is an arbitrary name which is used to reference the transmission line in other commands.

The first set of options specify the impedance. Choose one of these. Any required functions are functions of position along the transmission line measured in meters. For the RADII option, the arguments are r_outer and r_inner of a vacuum coaxial cable. For the IMPEDANCE option, the arguments are impedance and permittivity. For the CAPACITANCE option, the arguments are capacitance and inductance (per unit length). If resistance is required, it can also be entered under any of the options.

The second set of options is used to specify the spatial grid. Choose one of these. The algorithms are identical with those for the two-dimensional simulation (Part 2: GRID), and will not be discussed here. The entire grid must be specified using only one option. It is not possible to append grid onto a transmission line. All lines have a spatial origin of zero.

The final option provides the capability for non-zero initialization of the transmission line. The arguments are voltage and current functions.

The transmission line model calculates several variables of interest. These variables can be recorded by invoking the OBSERVE and RANGE commands. The following table lists the keywords and physical definitions associated with each variable.

TRAMLINE Command

Keyword	Definition	Notes
CURRENT	current	
POWER	power	
VOLTAGE	voltage	
CAPACITANCE	capacitance/length	1
INDUCTANCE	inductance/length	1
IMPEDANCE	impedance	1
ENERGY-CAP	capacitive energy	2,3
ENERGY-IND	inductive energy	2,3
ENERGY-TOT	total energy	2,4

1. Only accessible with the RANGE command.
2. For OBSERVE, requires a finite range, not a point.
3. For RANGE, returns energy/length.
4. Not accessible with the RANGE command.

Restrictions:

1. The maximum number of transmission lines that can be defined is ten.
2. The maximum number of total grid points that can be defined is 1000.

See Also:

SYSTEM, Ch. 10
GRID, Ch. 11
MATCH, Ch. 12
LOOKBACK, Ch. 13
VOLTAGE, Ch. 13
MAXWELL, Ch. 17
OBSERVE, Ch. 22
RANGE, Ch. 23

Examples:

Consider a coaxial, six-vane magnetron with a transmission line connected to the bottom of one vane slot. The impedance of the transmission line is matched to the characteristic impedance at the bottom of the vane slot. The end of transmission line LOAD is terminated with a VOLTAGE

TRAMLINE Command

command. In this example, no reflection from the end of the transmission line is desired; therefore the voltage function is set to zero applied voltage.

```
TRAMLINE LOAD IMPEDANCE 4.034 1.0
      UNIFORM      DISTANCE 1.0 CELLS 11 ;
MATCH vane_slot NEGATIVE NULL LOAD 0.0 0.05 ;
FUNCTION REFLECTED(X)= 0.0 ;
VOLTAGE LOAD 1.0 NEGATIVE REFLECTED ;
```

JOIN Command

Function: Joins two transmission lines together.

Syntax:

```
JOIN primary_line point { POSITIVE, NEGATIVE, SERIES, PARALLEL }  
      secondary_line point { POSITIVE, NEGATIVE } [CROSSOVER];
```

Arguments:

primary_line	-	name of primary transmission line, defined in TRAMLINE command.
secondary_line	-	name of secondary transmission line, defined in TRAMLINE command.
point	-	spatial coordinate (m).

Description:

The JOIN command is used to connect two transmission lines. The two lines can either be joined at their ends, or the end of one line (the secondary line) can be joined to an interior point of another line (the primary line) to produce parallel or series circuits.

Both transmission lines should have the same cell size at their joining points. The two transmission lines can have different properties at the joint, however. For example, JOIN can be used with two transmission lines of different capacitance and inductance to model impedance mismatch effects.

When the joining point is the end of the transmission line (always the case for the secondary line), the user must specify whether the line lies in the POSITIVE or NEGATIVE direction from the endpoint along the transmission line coordinates. When the joining point is an interior point (primary line only), the user must specify whether the circuit connection is made in SERIES or in PARALLEL. In general, series joints tend to form a voltage divider, while parallel joints tend to form a current divider. Note that if impedance mismatch is to be avoided at series and parallel joints, it is necessary to have a step discontinuity in the impedance function of the primary transmission line. Figure 13-1(a) illustrates the eight possible circuit configurations for end-to-end, series, and parallel joints. For illustration simplicity, the transmission lines are depicted as being of the parallel-plate variety.

Eight additional circuit configurations are possible by reversing the voltage polarity of the secondary line with respect to the primary line, as shown in Figure 13-1(b). This is illustrated as a crossover between the top and bottom plates of parallel-plate transmission lines immediately before the joint. These additional eight circuit configurations are invoked with the CROSSOVER option.

JOIN Command

Restrictions:

1. All transmission lines must be specified using TRAMLINE commands.
2. For linear and parallel circuits, the line cell sizes must be identical at the joint.
3. The maximum number of JOIN commands is ten.

See Also:

MATCH, Ch. 12
TRAMLINE, Ch. 13

Reference:

B. Goplen, J. Brandenburg, and T. Fitzpatrick, "Transmission Line Matching in MAGIC," Mission Research Corporation Report, MRC/WDC-R-102, September 1985.

JOIN Command

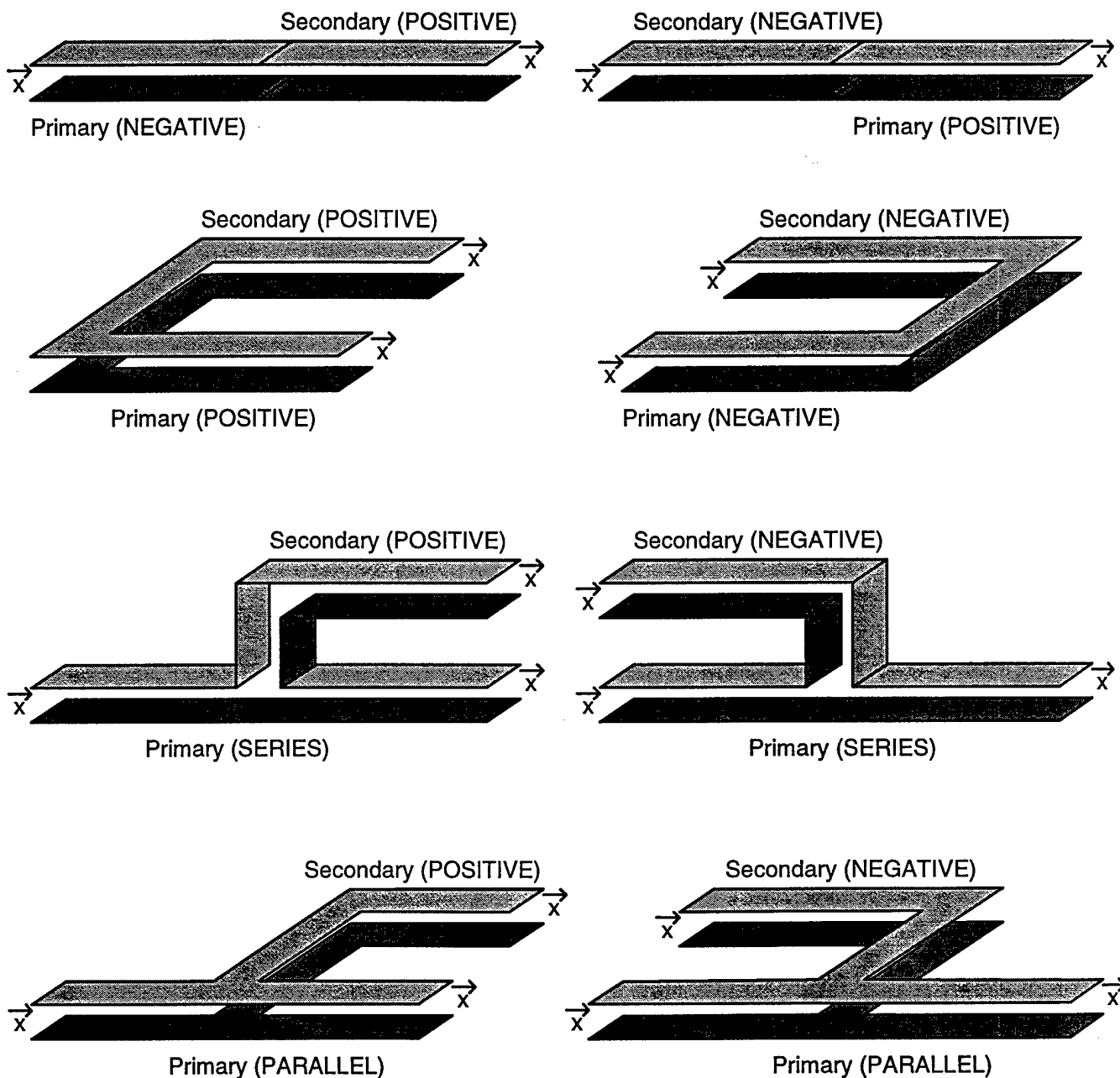


Figure 13-1(a). Default circuit configurations of JOIN with two parallel-plate transmission lines. The sign of the voltage (bottom-plate to top-plate) is preserved between primary and secondary transmission lines.

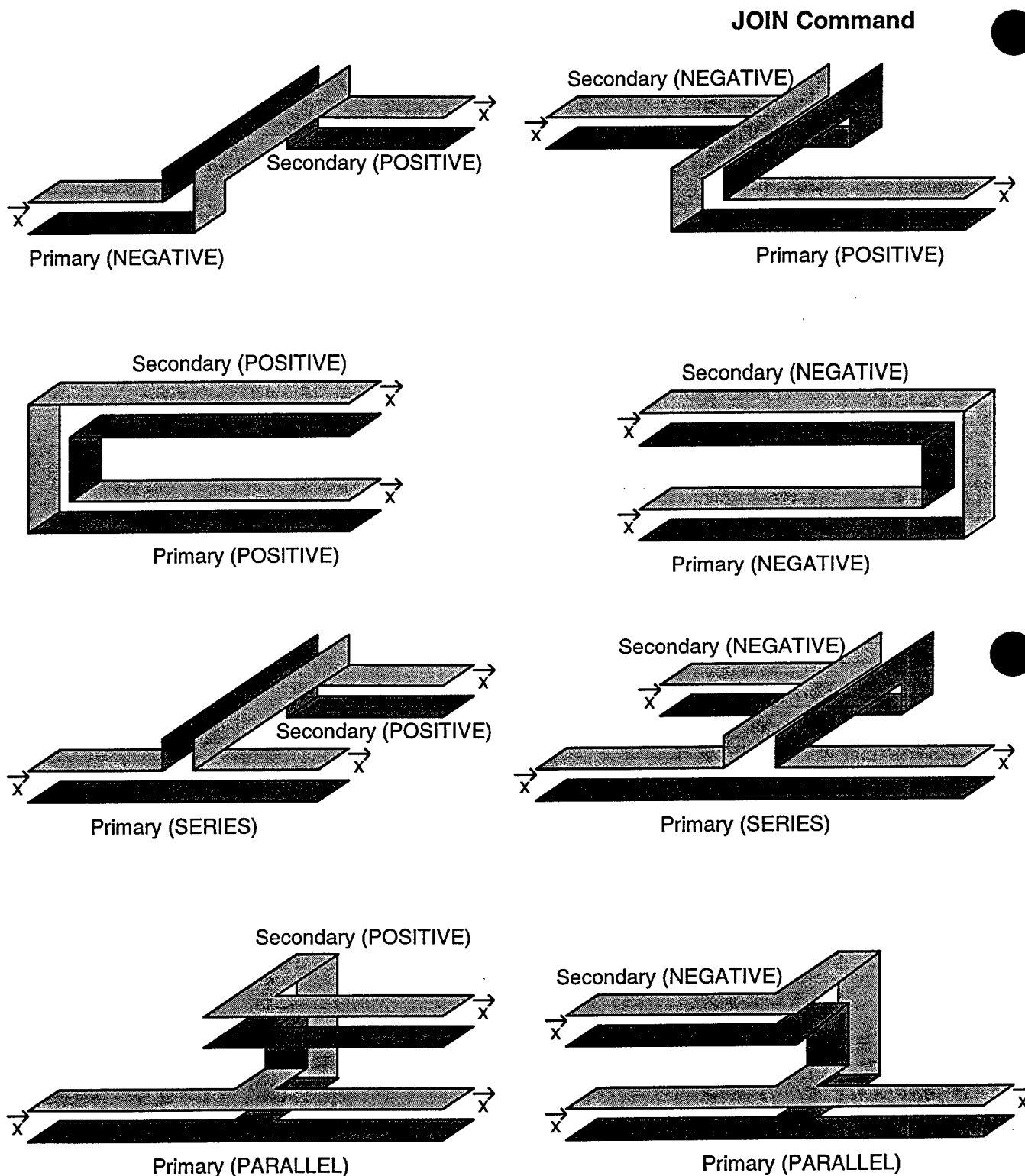


Figure 13-1(b). CROSSOVER circuit configurations for JOIN with two parallel-plate transmission lines. The sign of the voltage (bottom-plate to top-plate) is reversed between primary and secondary transmission lines.

LOOKBACK Command

Function: Specifies a boundary in a transmission line for outgoing waves.

Syntax:

```
LOOKBACK transmission_line  
           point {POSITIVE, NEGATIVE } ;
```

Arguments:

transmission_line	-	name of line, defined in TRAMLINE command.
point	-	spatial coordinate (m).

Description:

The LOOKBACK command specifies an outlet boundary condition which absorbs any outgoing waves from transmission_line. The sense is always from outside the line to inside. If this is in the direction of increasing coordinate, enter POSITIVE. Otherwise, enter NEGATIVE. The location of the boundary is defined by point.

See Also:**VOLTAGE, Ch. 13****Reference:**

B. Goplen, "Boundary Conditions for MAGIC," Mission Research Corporation Report, MRC/WDC-R-019, presented at the Twenty-Third Annual Meeting APS Division of Plasma Physics, 12-16 October 1981.

Restrictions:

1. The total number of LOOKBACK commands in a simulation is limited to five.
2. A FIELDS command must precede any LOOKBACK commands.
3. A Courant criterion limits the time step.

VOLTAGE Command

Function: Specifies a boundary in a transmission line for outgoing (and incoming) waves.

Syntax:

```
VOLTAGE transmission_line  
point { POSITIVE, NEGATIVE } f(t) ;
```

Arguments:

transmission_line	-	name of line, defined in TRAMLINE command.
point	-	spatial coordinate (m).
f(t)	-	voltage function, defined in FUNCTION command

Description:

The VOLTAGE command specifies a voltage function, $f(t)$, which is incoming to the transmission_line. The sense is always from outside the line to inside. If this is in the direction of increasing coordinate, enter POSITIVE; otherwise, enter NEGATIVE. The location of the boundary is defined by point.

See Also:

FUNCTION, Ch. 6
LOOKBACK, Ch. 13

Restrictions:

1. The total number of VOLTAGE commands in a simulation is limited to five.
2. A FIELDS command must precede any VOLTAGE commands.
3. A Courant criterion limits the time step.

14. BULK PROPERTIES

This chapter covers the following commands:

AIR-CHEMISTRY
CONDUCTANCE
CONDUCTOR
DIELECTRIC
MATERIAL

You can use these commands to assign bulk material properties to areas. (Areas are the only spatial objects which can have a bulk property associated with them.) Bulk properties affect both electromagnetic fields and charged particles. In general, particles which enter any area which has been assigned a bulk property will be destroyed (absorbed on the surface). The sole exception is air-chemistry, which relies upon charged particles to ionize the ambient gas.

There are four commands which provide bulk property effects. These are AIR-CHEMISTRY, CONDUCTANCE, CONDUCTOR, and DIELECTRIC. (The remaining command, MATERIAL, provides a list of conventional materials and their properties which can be used by a variety of other commands.)

The AIR-CHEMISTRY command is used to create a gas in the 0.05 – 1 atmosphere pressure regime. When energetic charged particles ionize the gas, the ionized particle species are treated as fluids, which reduce the ambient electric fields through a resistivity effect. Therefore, although the air-chemistry model is complex, the bulk property involved is simply a resistivity which varies through time and the specified spatial area.

Resistivity is also the bulk property provided by the CONDUCTANCE command; however, in this case, the resistivity is assumed to be known in advance and must be specified as a function of time and space. The CONDUCTOR command provides the extreme case of zero resistivity, or perfect conductivity. It is commonly used to represent metallic components. All electromagnetic fields within the specified area will vanish identically. Finally, the DIELECTRIC command provides a way to modify the relative permittivity within an area. Where not otherwise specified, the vacuum permittivity of unity will be in effect.

This page intentionally left blank

AIR-CHEMISTRY Command

Function: Applies an air conductivity model.

Syntax:

```
AIR-CHEMISTRY { CENTERED, EXPONENTIAL }
               { LL, LON, RAD, PET, USER }
               { LL, LON, RAD, PET, USER }
               { B, LON, RAD, PET, USER }
               pressure grate kstability
               [GAS { AIR, N2 }]
               [HUMIDITY vpress]
               [MOBILITY xmue(E,p,x1,x2) xmui(p)]
               [AVALANCHE alpha(E,p,x1,x2)]
               [ATTACHMENT beta(E,p,x1,x2)
                alpha_e(p) alpha_i(p)] ;
```

or

```
AIR-CHEMISTRY IONIZE qion(t) bava(t) batt(t) area_name ;
```

Arguments:

pressure	-	air pressure (atm).
grate	-	ionization rate coefficient (ion pair/C-m).
kstability	-	time step interval for stability test.
vpress	-	percent water vapor pressure, default is 0%.
xmue	-	name of electron mobility function, (m ² /volt-sec).
xmui	-	name of ion mobility function, (m ² /volt-sec).
alpha	-	name of avalanche coefficient function, (1/sec).
beta	-	name of attachment coefficient function, (1/sec).
alpha_e	-	name of recombination coefficient function, (m ³ /sec).
alpha_i	-	name of neutralization coefficient function, (m ³ /sec).
qion	-	name of ionization rate function.
bava	-	name of avalanche rate function.
batt	-	name of attachment rate function.
area_name	-	name of spatial area, define in AREA command.

Description:

The first form of the AIR-CHEMISTRY command specifies the details of the algorithms, models, and parameters of the air-conductivity model, where a gas with finite conductivity is assumed to fill the specified simulation space entirely. The air-conductivity

AIR-CHEMISTRY Command

results from gas ionization leading to background densities of electrons (n_e), positive ions (n_p), and negative ions (n_n). The densities of these background species evolve according to:

$$\begin{aligned} Dn_e/Dt &= \text{qrate} * |J| + (\text{alpha} - \text{beta}) * n_e - \text{alpha}_e * n_e * n_p \\ Dn_n/Dt &= \text{beta} * n_e - \text{alpha}_i * n_n * n_p \\ n_p &= n_e + n_n \end{aligned}$$

where:

- qrate = ionization rate coefficient
- |J| = absolute value of current density
- alpha = avalanche coefficient
- beta = electron attachment coefficient
- alpha_e = electron-ion recombination coefficient
- alpha_i = positive-negative ion neutralization coefficient.

The air-conductivity is obtained from the equation:

$$\sigma = e * (x_{mue} * n_e + x_{mui} * n_p + x_{mui} * n_n)$$

where:

- e = charge of the electron
- xmue = background electron mobility
- xmui = background ion mobility

The number densities for all ionized species (electrons, negative ions, and positive ions) are calculated by either a CENTERED-difference or an EXPONENTIAL-difference algorithm.

The next three arguments control, in the following order:

- a) the avalanche model,
- b) the attachment model, and
- c) the conductivity model.

The avalanche coefficient, alpha, and the electron and ion attachment coefficient, beta, alpha_e, and alpha_i, are presently available in the original Longmire-Longley model (LL), or in three new models, including a Longmire-Longley model (LON), a Radasky model (RAD), and a Pettus model (PET), or as arbitrary user provided functions of electric field, gas pressure, and position (USER). The new models have additional options, including HUMIDITY, e.g., vapor pressure, effects in all three models and the choice of GAS, either AIR or nitrogen, N2, in the Radasky model. The electron and ion mobilities, xmue and xmui, are available in the original Baum model (B), or Baum-like accompanying models to the three new models, or as arbitrary USER provided functions of electric field, gas pressure, and position. Specifying USER for any of the three arguments requires that the corresponding MOBILITY, AVALANCHE, or ATTACHMENT option be used.

AIR-CHEMISTRY Command

All air chemistry quantities are functions of the electric field and air pressure. The electric field is computed dynamically; however, the air pressure must be entered by the user. The ionization rate, $\text{grate} \cdot |J|$, is computed from the absolute current density $|J|$ in this model of air conductivity. The constant of proportionality, grate , must be specified by the user. The current density term, J , is computed self consistently by the code from the particle motions. The stability of the algorithm is checked at intervals given by kstability .

The air-chemistry module calculates a number of fields which vary in time and two-dimensional space. These air-chemistry fields can be output in the same manner as the electromagnetic fields. The definitions of the air-chemistry fields are given in the following table.

Keyword	Symbol	Definition
QION	Q	ionization rate
BAVA	α	avalanche coefficient
BATT	β	electron attachment coefficient
XMBE	μ_e	electron mobility
RHOE	n_e	electron number density
RHON	n_n	negative ion number density
RHOP	n_p	positive ion number density
SIGMAC	σ	conductivity

In the second form of the AIR-CHEMISTRY command option, the IONIZE keyword specifies a simplified, electrons-only, model. The three functions represent the zeroth, linear, and quadratic coefficients in a differential equation for conductivity given by

$$D\sigma/Dt = qion \cdot |Q0| + bava \cdot \sigma - batt \cdot \sigma^2$$

which contains the electron charge density, $Q0$, in the ionization term instead of the current.

Restrictions:

1. A stability criterion exists for the centered-difference algorithm which limits the allowable time step for the species integration.

AIR-CHEMISTRY Command

2. The ionization rate is computed from the net current density; cases involving large current-density cancellations (e.g., counterflow) will not be modeled accurately.

3. The air chemistry algorithm is applicable in the pressure regime, $0.05 \text{ atm} \leq P \leq 1 \text{ atm}$.

See Also:

EMISSION ... BEAM, Ch. 16

TIME_STEP, Ch. 17

OBSERVE, Ch. 22

References:

K. Wahlstrand, R. Worl, K. Nguyen, and B. Goplen, "AURORA Drift Tube Simulations," Mission Research Corporation Report, MRC/WDC-R-163, April 1988.

B. Goplen, "An Air Chemistry Algorithm for SOS," Mission Research Corporation Report, MRC/WDC-R-043, September 1983.

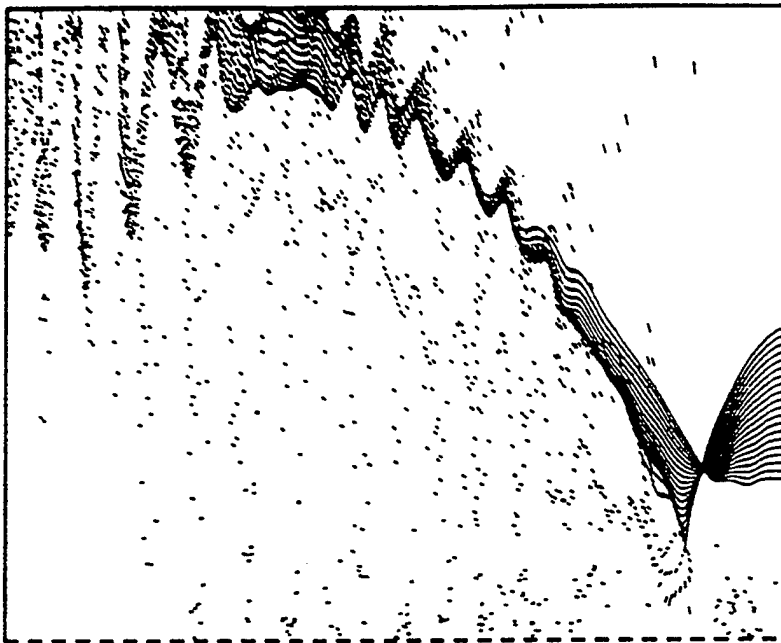
Examples:

The use of the AIR-CHEMISTRY command is illustrated using a drift tube as an example. The purpose of the drift tube is to sharpen the 45 nsec rise time of the 7 MeV, hollow electron beam as it exits the diode. The following commands include an air-chemistry simulation at a pressure of one atmosphere using exponential differencing.

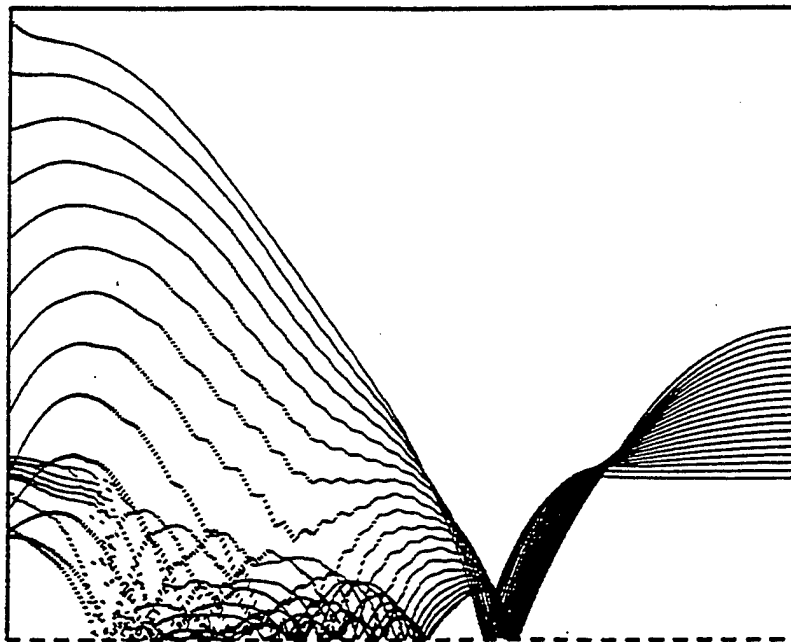
```
AIR-CHEMISTRY EXPONENTIAL LL LL B 1.0 5.E22 1 ;  
FUNCTION JPLUS DATA 4  
      0.0      0.0  
      45E-9    1E6  
      85E-9    1E6  
      165E-9   0.0 ;  
EMISSION BEAM JPLUS 4.5E9  
EMIT BEAM Annulus_Area ;
```

Note that the electromagnetic field algorithm will automatically account for the fluid conductivity if the air-chemistry module is activated. Figure 14-1 illustrates beam propagation both in vacuum (without air-chemistry effects) and in one atmosphere of air. The vacuum case exhibits Coulomb repulsion and the onset of plasma oscillations. The one atmosphere case exhibits crossovers due to overfocusing and the onset of thermalization.

AIR-CHEMISTRY Command



(a) Beam propagation in vacuum.



(b) Beam propagation in 1 atm air.

Figure 14-1. Air-chemistry effects in Aurora drift tube.

CONDUCTANCE Command

Function: Applies finite conductivity property.

Syntax:

CONDUCTANCE area_name sigma(x1,x2) [SCALE scale_factor(t)] ;

Arguments:

area_name	-	name of spatial area, defined in AREA command.
scale_factor(t)	-	conductivity scale factor, temporal function defined in a FUNCTION command.
sigma(x1,x2)	-	conductivity (mhos/m), constant or spatial function defined in a FUNCTION command.

Description:

The CONDUCTANCE command specifies conductivity (mhos/m) in a two-dimensional region. The conductance, sigma, may be entered as a constant or as a function of the spatial coordinates. The SCALE option multiplies the specified conductance by a time-dependent scale_factor. Conductivity will be applied only within the specified spatial area.

Note that, if conductance is employed over large areas of the simulation, the damping time scale will go approximately as

$$\tau = \epsilon/\sigma$$

The conductance field can be output in the same manner as the electromagnetic fields, using the field name SIGMA (static option) or SIGMAC (dynamic option).

Restrictions:

1. The maximum number of conductance regions that can be specified is ten.
2. Conductance regions must not overlap.

See Also:

FUNCTION, Ch. 6
MAXWELL, Ch. 17

CONDUCTOR Command

Function: Applies perfect (infinite) conductivity property.

Syntax:
CONDUCTOR area_name ;

Arguments:

area_name - name of spatial area, defined in AREA command.

Description:

The CONDUCTOR command applies perfect (infinite) conductivity in the area designated by area_name.

CONDUCTOR materials are the most basic and most common of the bulk materials. All electromagnetic fields within perfectly conducting materials vanish. All particles which enter the object are destroyed on the surface. Furthermore, objects with this material property, and only this property, may emit particles as well.

CONDUCTOR materials are always resolved into conformal segments and cell-diagonal segments. For nonconformal areas, the shape is approximated in a partially smoothed stair-step fashion. Under some circumstances this may be unsatisfactory, and the SHIM command can be used to improve the smoothness of the perfectly conducting material.

See Also:

AREA, Ch. 10
EMIT, Ch. 16
SHIM, Ch. 15

Examples:

The plasma-edge cathode consists of an outer area named "cavity" and an inner area named "island." These areas can be made perfectly conducting with the following commands.

```
CONDUCTOR    cavity ;  
CONDUCTOR    island ;
```

The device in operation is shown in Figure 14-2.

CONDUCTOR Command

MAGIC VERSION: SEPTEMBER 1988 DATE: 11/30/88
SIMULATION: TEXAS TECH PLASMA EDGE CATHODE

TRAJECTORY PLOT OF ELECTRONS (ISPE - 2)
FROM TIME 1.200E-09 SEC TO 1.202E-09 SEC

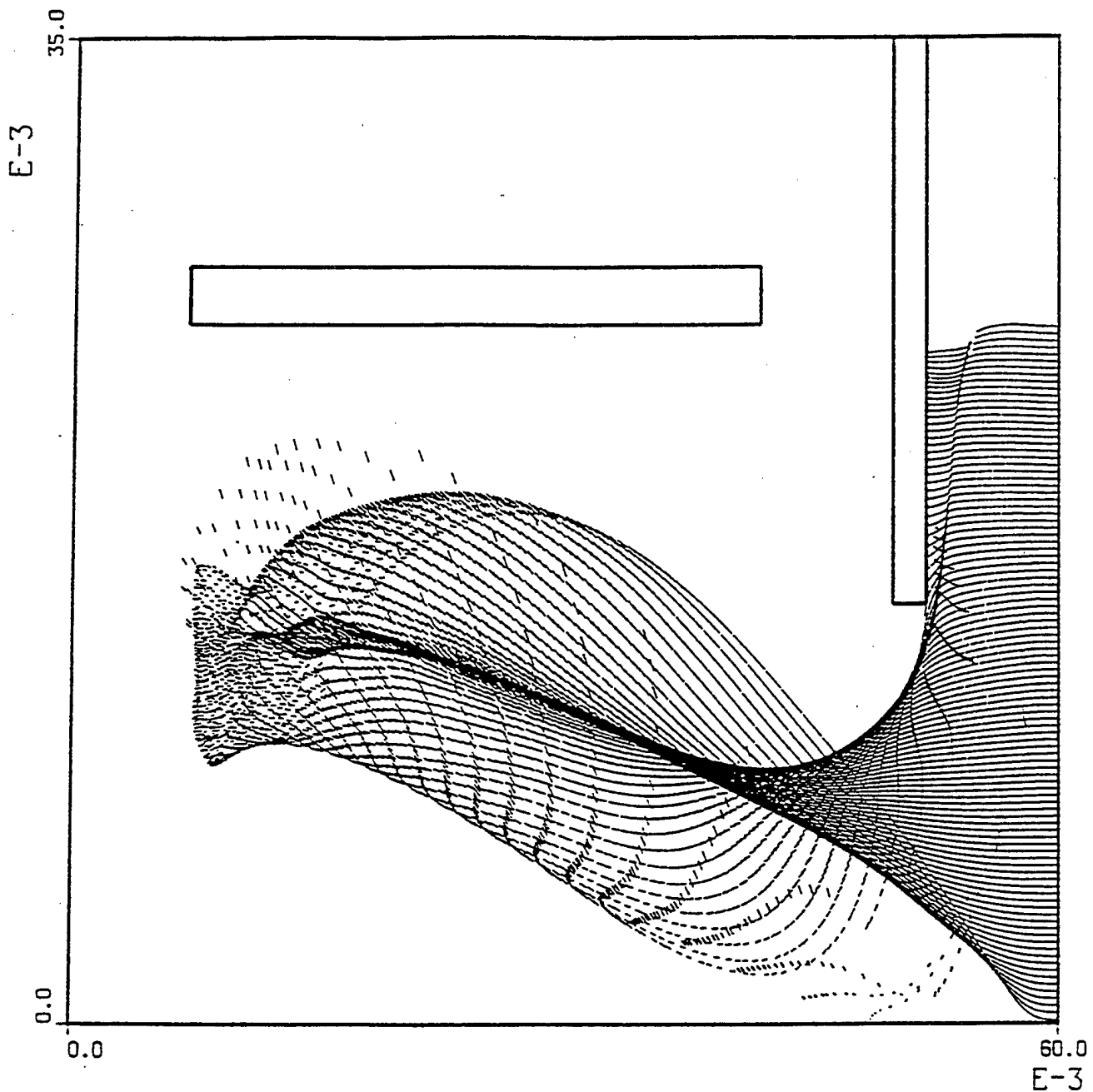


Figure 14-2. Electron trajectories in the plasma edge cathode.

DIELECTRIC Command

Function: Applies dielectric permittivity property.

Syntax:

```
DIELECTRIC area_name permittivity(x1,x2)
      [{ EPS1, EPS2, EPS3 }] ;
```

Arguments:

area_name - name of spatial area, defined in AREA command.
permittivity- relative permittivity (eps/eps0), constant or spatial function defined in a FUNCTION command.

Defaults:

The default permittivity distribution is isotropic; that is, a single scalar permittivity affects all field components equally. To create an anisotropic distribution, you must specify each component of permittivity (EPS1, EPS2, EPS3) independently using separate DIELECTRIC commands.

Description:

The DIELECTRIC command applies dielectric permittivity on an area designated by area_name. The permittivity (here meaning relative permittivity, also known as the dielectric constant) may be entered either as a constant or as a scalar function of the spatial coordinates. The permittivity function will be applied everywhere within the area. To enter an isotropic dielectric, a single command is sufficient, and the permittivity component is not entered. By default, the permittivity entered will be applied to all three components. To make the dielectric anisotropic, enter separate commands to specify each permittivity component (EPS1, EPS2, EPS3) independently. As many as three commands may be required, depending on the application.

The vacuum value of permittivity is used everywhere else (outside of area_name) in the simulation. A particle which strikes area_name is destroyed, and its charge is deposited permanently on the surface. If lossy dielectrics are required, conductance can also be applied to area_name (CONDUCTANCE, Ch. 14). If area_name contains or contacts an outer boundary (e.g., PORT, Ch. 12), then any phase velocity information required in the outer boundary command must account for the dielectric property. (Failure to consider this may result in artificial scattering from the outer boundary.)

The displacement fields (D1, D2, D3) can be output in the same manner as any other two-dimensional field. The relative permittivity components can also be output using the field names, EPS1, EPS2, and EPS3.

DIELECTRIC Command**Restrictions:**

1. Only an area object can have a dielectric property.
2. Anisotropic dielectrics require one or more DIELECTRIC commands to be entered.
3. Use of a DIELECTRIC command may alter the phase velocity and thus affect input requirements for outer boundary commands.
4. The maximum number of DIELECTRIC commands that can be entered is twenty.

See Also:**PORT, Ch. 12****CONDUCTANCE, Ch. 14****Examples:**

(1) In the following example, a coaxial cable has a isotropic dielectric inserted in the gap (area_name is "Gap_Area") between the inner and outer conductors. An isotropic dielectric with relative permittivity of unity can be applied using the command,

```
DIELECTRIC Gap_Area 1 ;
```

This will produce the same electromagnetic effect as vacuum: however, any particle entering "Gap_Area" will be destroyed on the perimeter.

(2) To create an anisotropic dielectric with a relative permittivity of four in the azimuthal component (and unity for the radial and axial components), the command,

```
DIELECTRIC Gap_Area 4 EPS3 ;
```

is sufficient.

(3) In this example, the permittivity is assumed isotropic with a dielectric constant of four, and a Poisson solution is obtained for the scalar potential and electrostatic fields. Figure 14-3 shows vectors of the electrostatic fields (E1ST, E2ST) and contours of the scalar potential (PHST) in the "Gap_Area." The commands are

```
DIELECTRIC Gap_Area 4 ;  
POISSON ... ;  
CONTOUR ONCE FIELD PHST Gap_Area ;  
VECTOR ONCE FIELD E1ST,E2ST NUMBER 24 24 ;
```


DIELECTRIC Command

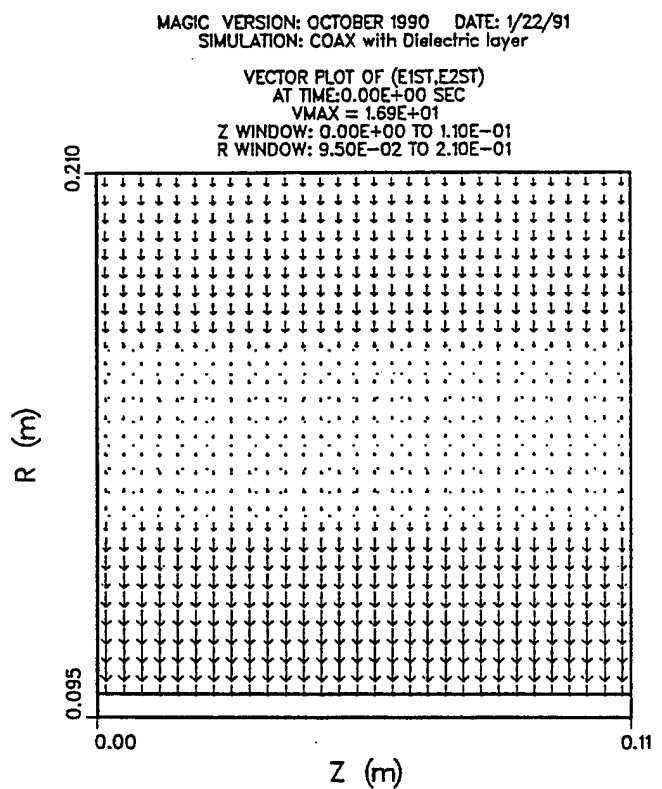
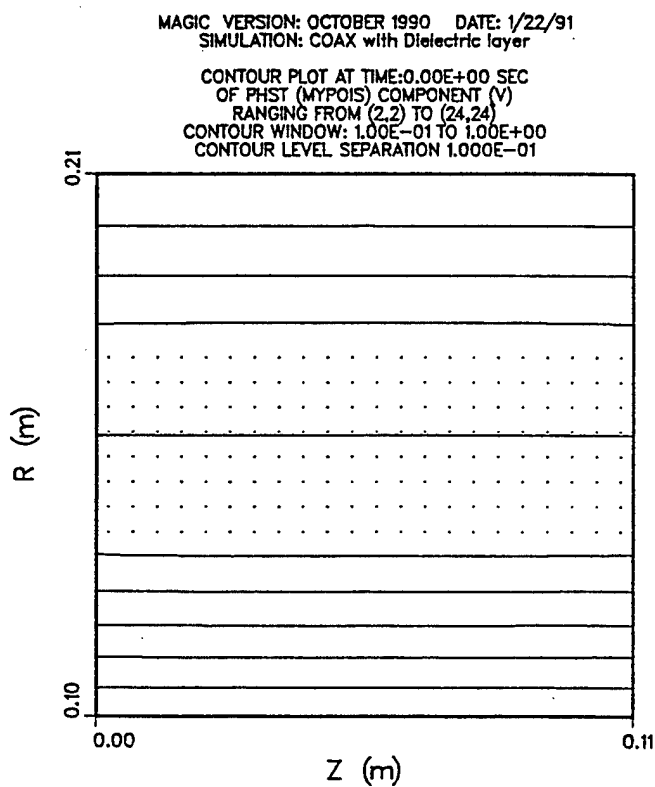


Figure 14-3. Scalar potential and electrostatic fields in dielectric.

MATERIAL Command

Function: Specifies material properties.

Syntax:

```
MATERIAL name  
[MASS num mass density]  
[RESISTIVITY rho(x1,x2)]  
[DIELECTRIC epsilon(x1,x2)] ;
```

Arguments:

name	-	material name, user-defined.
num	-	atomic number.
mass	-	atomic mass (amu).
density	-	density (kg/m**3)
rho	-	name of resistivity function (ohm-meter), defined in FUNCTION command.
epsilon	-	name of relative dielectric function defined in FUNCTION command.

Description:

The MATERIAL command associates a set of physical properties with an arbitrary name. Other commands, such as FOIL, which require values for these properties may reference the name specified in the MATERIAL command. The MASS, DIELECTRIC, and RESISTIVITY options specify particular material properties.

MASS is used to enter mass properties of a material. Num specifies the atomic number, mass specifies the atomic mass, and density specifies the mass density of the material.

RESISTIVITY is used to enter the electrical resistivity ρ of a material. DIELECTRIC is used to enter the relative dielectric constant ϵ_r of the material. There are currently no commands which use these data items.

Restrictions:

The maximum number of materials that can be specified is twenty.

See Also:

FOIL, Ch. 15

Example:

```
MATERIAL GOLD MASS 79 197 2.5E4 ;
```

15. UNIQUE GEOMETRY

This chapter covers the following commands:

FOIL
POLARIZER
SHIM
STRUT

You can use these commands to apply unique geometry properties on lines. (Lines are the only spatial object which can have a unique geometry property associated with them.) While bulk properties affect both electromagnetic fields and charged particles, unique geometry properties generally affect only the electromagnetic fields. Particles which cross the line are unaffected, except through the fields (i.e., there is no absorption). The sole exception to this rule is the foil model, in which charged particles can be scattered and/or absorbed.

The four commands listed above invoke different geometry effects. Their unifying feature is that they all represent physical effects which occur on a spatial scale too small to be resolved by any realistic spatial grid. As a class, they have been described as “sub-grid” models. It is because of this small spatial scale that these models are applied only to lines (which have zero width) and not to areas.

The FOIL command describes a thin foil. It is treated as a perfect conductor electromagnetically, and charged particles which penetrate the foil can undergo multiple scattering with resulting energy loss and even absorption.

The POLARIZER command approximates a three-dimensional structure, specifically, an infinite array of wires. For, example, in two-dimensional cylindrical coordinates, it can effectively represent a three-dimensional helical structure. In non-trivial applications, the model couples transverse magnetic and transverse electric field components, which are generally independent in a purely electromagnetic, two-dimensional simulation. For conformal orientations, the model can block either field component while letting the other component pass.

The STRUT command provides a capability similar to that of the POLARIZER command, but also allows the inductive properties of the wires to be included. However, it is presently limited to conformal orientations.

The SHIM command produces a very fine-scale geometric variation on the surface of a conductor. (The scale of the variation must be smaller than the surrounding spatial grid.) This model can be used to represent a very gradual slope or a rippled effect on a surface.

This page is intentionally left blank.

FOIL Command

Function: Applies a thin foil geometry.

Syntax:
FOIL material thickness line_name [nscatter] ;

Arguments:

material	-	name of material, defined in a MATERIAL command.
thickness	-	thickness of foil (meters).
line_name	-	name of conformal spatial line, defined in LINE command.
nscatter	-	number of scattering events per foil transmit.

Description:

The FOIL command is used to specify a thin, conducting surface along the specified line. The transverse electric fields on such surfaces vanish; however, particles which penetrate the foil may either be destroyed or scattered. The scattering parameters will be calculated from the material properties of the foil and the thickness of the foil. Scattering requires three-dimensional kinematics.

The material from which the foil is made is entered using the MATERIAL command. Nscatter, is used to set the number of numerical scattering events that occur in a transit of the foil. For thin foils, this number may be set to unity, which is the default. For thicker foils, a number greater than unity may be used. In this case, the total energy degradation and total momentum rotation is evaluated as if the particle traversed nscatter foils each of length, thickness/nscatter. A new momentum vector is evaluated for each scattering event and is used as the incident vector for the subsequent scattering.

A foil may be referenced by other commands and is named by its material appendix suffix FOIL; e.g., a foil material TIN will have the name TINFOIL.

Restrictions:

1. The line must be conformal.
2. The maximum number of foils is twenty.
3. Three-dimensional kinematics is required.

FOIL Command

See Also:

MATERIAL, Ch. 14

Examples:

The following example illustrates the FOIL and MATERIAL commands. The MATERIAL command is used to enter the mass properties of platinum in the material table. Two thin conformal foils are specified using the FOIL command. A trajectory plot of the electrons is requested. Figure 15-1 shows the scattering trajectories of the particles passing through each foil.

```
NUM = 78 ;  
fMASS = 195 ;  
DENS = 2.14E4 ;  
MATERIAL PLATINUM MASS NUM MASS DENS ;  
THICKNESS = 2.E-7 ;  
FOIL PLATINUM THICKNESS FOIL1_LINE ;  
FOIL PLATINUM THICKNESS FOIL2_LINE ;  
TIMER TRAJ_TIMER DISCRETE 1 ;  
TRAJECTORY KMAX TRAJ_TIMER KMAX ALL TRAJ_AREA ;
```

FOIL Command

MAGIC VERSION: OCTOBER 1990 DATE: 1/22/91
SIMULATION: FOIL TEST

TRAJECTORY PLOT OF ALL SPECIES
FROM TIME 0.000E+00 SEC TO 1.700E-10 SEC

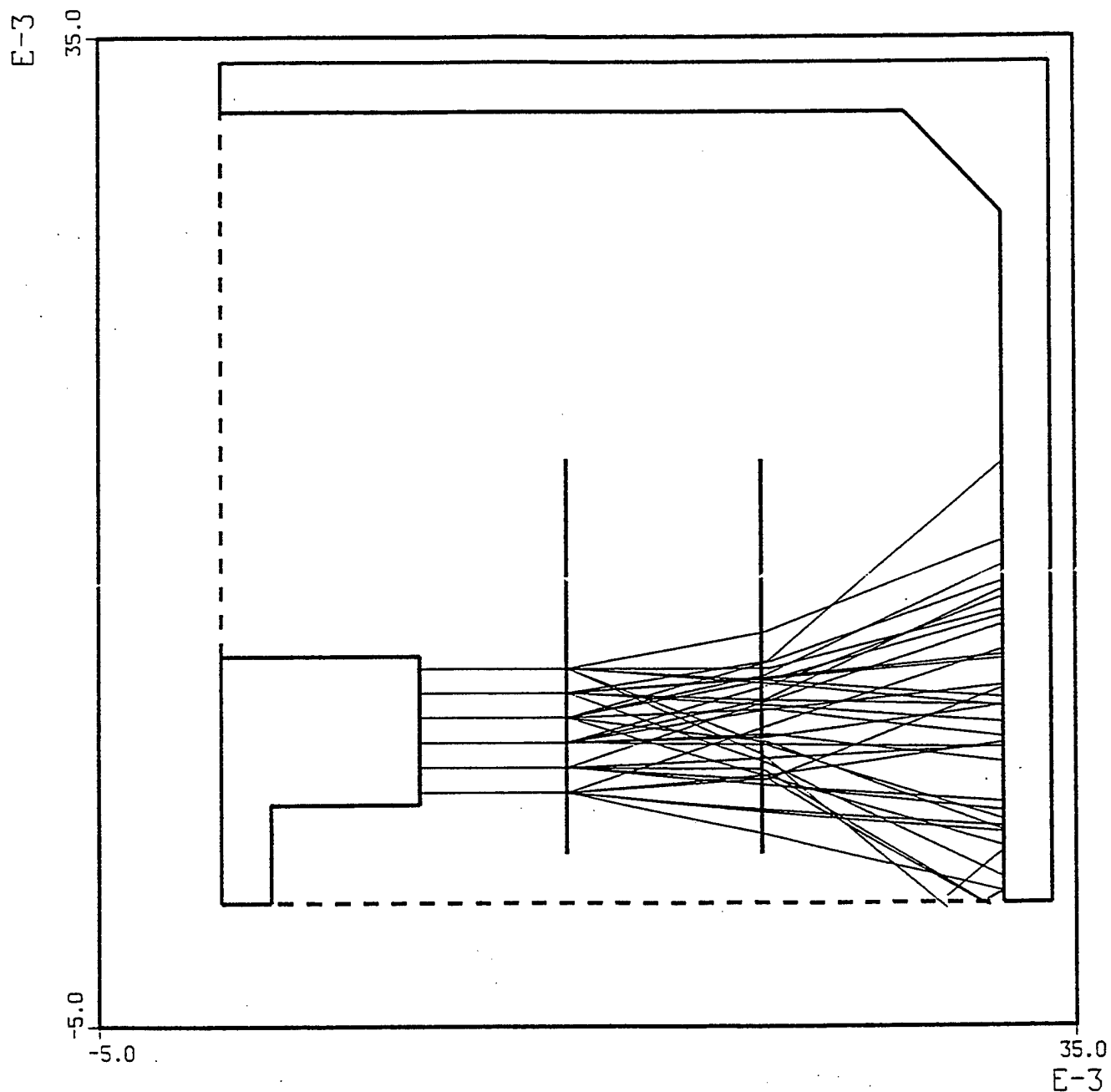


Figure 15-1. Trajectory plots of electrons scattered by foil.

POLARIZER Command

Function: Applies a polarizer geometry.

Syntax:

POLARIZER line_name pitch_angle(x1,x2) [ialg] ;

Arguments:

pitch_angle - pitch angle (degrees), constant or spatial function defined in a FUNCTION command.
ialg - algorithm (integer).
line_name - name of conformal line, defined in LINE command.

Description:

The POLARIZER command provides a capability to model certain three-dimensional conducting structures in a two-dimensional simulation. In effect, it couples TE and TM electromagnetic fields to reflect the correct conditions for an array of "wires" at an angle to the axis of symmetry. A perfect application is a helical geometry in cylindrical coordinates. In this case, the helix sheath approximation described by Pierce (see References) provides an excellent theoretical discussion. There are other, non-helical applications as well, including wave-splitters, antenna shields, and waveguide screens.

The polarizer is applied along a single, conformal line specified by the line_name. The pitch_angle (in degrees) describes the orientation of the wires in the array, relative to the X3 (ignorable) axis, measured in the positive direction of the X1 axis or the X2 axis. A functional specification allows the pitch angle to vary with the relevant spatial coordinate.

The treatment of the polarizer endpoints can be controlled via ialg. In general, the endpoints of the polarizer may simply float in space, or they may be terminated on other boundaries, such as a conductor, an antenna, or an outgoing model. The centered algorithm, (ialg=0), does not impose the polarizer conditions at the endpoints, so that the conditions of any terminating boundary will still be met despite the presence of the polarizer.

Two additional algorithms are also available. These apply the polarizer condition at one end, either at the right/upper endpoint (ialg=+1), or at the left/lower endpoint (ialg=-1). These two additional algorithms have the important property that they are exactly energy-conserving for all wavelengths, whereas the centered algorithm can result in a small degree of damping on the shortest wavelengths. In situations where this damping may affect physical results, such as for a coarsely gridded helix, the energy-conserving algorithms should be used; however, for well resolved wavelengths, there is no appreciable difference between algorithms.

POLARIZER Command

Particles pass through a polarizer boundary unscathed. Of course, particle trajectories may be affected by strong electromagnetic fields in the vicinity of the boundary.

Restrictions:

1. A maximum of 50 polarizers are allowed.
2. The line must be conformal.
3. The polarizer is intended to be an internal object, and is not suitable for an exterior boundary condition.

See Also:

OUTGOING, Ch. 12
CONDUCTOR, Ch. 14
MODE, Ch. 17
DRIVER, Ch. 19

References:

J. R. Pierce, *Traveling Wave Tubes*, Van Nostrand Company, Princeton, NJ, 1950, Appendix 2.

D. Smithe, G. Warren, and B. Goplen, "A Helix Polarization Model for 2-D Particle-in-Cell Simulation of Helix Traveling Wave Tubes," APS Div. of Plasma Physics, Seattle, WA., November 1992.

B. Goplen, D. Smithe, K. Nguyen, M. Kodis, and N. Vanderplaats, "MAGIC Simulations and Experimental Measurements from the Emission Gated Amplifier I & II Experiments," 1992 IEDM, San Francisco, CA, December 1992.

Examples:

In the following example (cylindrical coordinates), a model of a helix is created between axial coordinates, ZI and ZF, at a radius coordinate, RI. The pitch angle (PSI_DEGREES) is measured from the azimuthal coordinate (ϕ) toward the axial coordinate (z). The polarizer is given the name "HELIX."

```
LINE HELIX_LINE CONFORMAL ZI RI ZF RI ;  
POLARIZER HELIX PSI_DEGREES ;
```

SHIM Command

Function: Applies a thin, perfectly conducting shim geometry.

Syntax:

```
SHIM CONDUCTOR area_name  
      THICKNESS shim_thickness(x) WINDOW area_name ;
```

Arguments:

area_name	- name of spatial object, defined in AREA command.
shim_thickness	- shim thickness, constant or defined in FUNCTION command.

Description:

The SHIM command is used to add a small, perfectly-conducting layer, or shim, to a conformal surface of a perfectly conducting object. The thickness of the layer must be smaller than the spatial cell size. Thus, this command provides a means to model a fine-scale variation in a conducting surface without resolving the variation with the spatial grid.

The shim is applied to CONDUCTOR area_name. This spatial object must be defined in an AREA command. Furthermore, the spatial object must also be perfectly conducting (CONDUCTOR, Ch. 14). The shim_thickness must be a fraction of the cell size and may be constant or a function of one of the spatial coordinates. The precise location of the shim is provided by WINDOW area_name, which must be immediately adjacent to one conformal surface of the conducting object. In other words, it describes where on the object the shim is to be applied. The WINDOW area should just touch the CONDUCTOR area on the conformal surface where the shim is to be applied.

The conformal surface where the shim is applied can have a length as short as one spatial cell or it can be much longer. It is also possible to taper or to modulate such an extended surface by means of the shim_thickness function. However, the shim_thickness is not allowed to exceed the height of the cell adjacent to the surface. A Courant violation may occur if this constraint is not observed. Care should also be taken to avoid using the shim in the immediate presence of certain types of outer boundaries. A good practice is to terminate the thickness function at both ends with zero thickness and to stay well removed from port and outgoing wave boundaries.

Restrictions:

1. The maximum number of SHIM boundaries that can be specified is five (5).
2. The shim model can be applied only to conformal surfaces of perfectly conducting area objects.

SHIM Command

3. The WINDOW area should touch the CONDUCTOR area, but they should not overlap.
4. Nonuniform spacing in the direction normal to the conducting surface is not allowed (the first two cells must be of equal size).
5. The thickness of the shim must not exceed the height of the cell adjacent to the conducting surface.

See Also:

AREA, Ch. 10
CONDUCTOR, Ch. 14
FUNCTION, Ch. 16
TIME_STEP, Ch. 17
MAXWELL, Ch. 17

Examples:

In the following test case, we define a coaxial cable with large radii and known impedance. The gap between the cathode and anode is modeled with two radial cells. An incident, semi-infinite wave is introduced and propagates through the cable without reflection. Next, a shim is introduced on the anode (case shown below). The shim_thickness varies sinusoidally from zero at the inlet and outlet to half the gap size at the midpoint. (Note that the function vanishes at the end points.). Since the midpoint impedance is effectively halved, the effect on the reflected wave is measurable.

```
dR = Router - Rinner ; Rmid = Rinner + 0.5*dR ;
AREA anode CONFORMAL 0,Router L,Router_plus ;
AREA cathode CONFORMAL 0,Rinner_minus L,Rinner ;
AREA shim_application 0, Rmid L,Router ;
FUNCTION thickness(z)=0.5 * dR * sin (1pi*z/L) ;
CONDUCTOR anode ; CONDUCTOR cathode ;
SHIM CONDUCTOR anode
      THICKNESS thickness WINDOW shim_application ;
```

STRUT Command

Function: Applies a strut (inductive circuit element) geometry.

Syntax:

```
STRUT line_name strut_number strut_diam(x1,x2)
      [RESISTIVE_MATERIAL material frequency]
      [SURFACE_RESISTIVITY sresist(x1,x2)]
      [FORCE_INDUCTANCE induct(x1,x2)]
      [FORCE_RESISTANCE resist(x1,x2)]
      [SUBCYCLES ncycles] ;
```

Arguments:

- line_name - name of conformal line, defined in a LINE command.
- strut_number - number of struts in third dimension (real).
 = number of struts/meter (cartesian).
 = number of struts around axis (cylindrical).
 = number of struts/meter (polar).
 = number of struts around axis (polar).
- strut_diam - strut diameter (meters), constant or spatial function defined in a FUNCTION command.
- material - rod material.
 = SILVER, COPPER, ALUMINUM, BRASS, or SOLDER.
- frequency - frequency for the material's surface resistivity (hertz).
- sresist - name of strut surface resistivity function (ohms), defined in a FUNCTION command.
- induct - name of forced inductance function (henrys/meter), defined in a FUNCTION command.
- resist - name of forced resistance function (ohms/meter), defined in a FUNCTION command.
- ncycles - number of subcycles, default is 1 (integer).

Description:

The STRUT command provides a capability to model certain three-dimensional conducting elements in a two-dimensional simulation. Such elements are generally thin conducting rods or "struts," which permit current to flow along them, often connecting one conductor to another. The struts typically are widely spaced in the simulation's ignorable direction, allowing electromagnetic wave energy at all but very low frequencies to pass through; thus, it cannot be represented as a solid conductor, since this would effectively block the passage of all electromagnetic energy.

STRUT Command

In the simulation, the strut must lie on a conformal line designated by `line_name`. You must also specify the `strut_number` in the ignorable (third) dimension.

The principal physical attribute of such a strut is the generation of strong magnetic fields near its surface, where current flows, leading to a behavior essentially identical to an inductive circuit element. The inductance is determined primarily by the diameter of the rod, which is specified by the function, `strut_diam`, in the command line. In cases where a circular cross section is not sufficient, or when precise control of the inductance is required, the user has the option of directly specifying the inductance per length, in henrys/meter, with the `FORCE_INDUCTANCE` option. If this option is used, the `strut_diam` value is not used to calculate inductance.

An additional physical attribute of a strut is a potential resistive component. By default, the resistance of a strut is zero. There are three means provided to assign resistance to a strut. The most simple means is with the `RESISTIVE_MATERIAL` option, which accepts one of five materials and a frequency, in order to compute the surface resistivity (or "resistance per square," $[= 1/(\text{conductivity} * \text{skin-depth})]$). A second option, `SURFACE_RESISTIVITY`, allows the user to specify a value of surface resistivity, `resist`, for a material which may not be present in the above list. The resistance of the rod is then calculated from the surface resistivity and the diameter of the strut. As with the inductance, the user also has the option of specifying the resistance per length directly, in ohms/meter, with the function argument, `resist`, of the `FORCE_RESISTANCE` option.

It is known that the strut model causes an additional courant-limit constraint on the time step and may additionally require subcycling of the strut model. In general, subcycling will be necessary when the time step exceeds the centered-difference courant limit, as is possible with the time-biased and hi-q MAXWELL options; the number of subcycles, `ncycles`, as specified in the `SUBCYCLES` option is expected to go as the ratio of the time step to the centered-difference limit. The reduction of the maximum allowable time step is usually very small in 2-D, only becoming noticeable for very small inductances, e.g., when the rod diameter approaches the length of the rod, or the distance between rods. The exact strut-affected courant limit for the time step is currently unknown, so the user may need to use trial and error techniques to find a suitably small time step which does not result in a courant instability.

STRUT variables which can be output with the `OBSERVE` command are `CURRENT`, `CHARGE`, `INDUCTANCE`, `RESISTANCE`, `STORED_ENERGY`, and `OHMIC_LOSS`. All pertinent variables can also be printed using the `DIAGNOSE` command with the subroutine name `STRUT`.

STRUT Command**Restrictions:**

1. A maximum of ten struts are allowed.
2. The line must be conformal to the grid.
3. The line may not contact an area which has a dielectric permittivity property.
4. The number of subcycles is the same for all struts and is taken to be the value of ncycles from the last strut command to use the SUBCYCLES option.

See Also:

CONDUCTOR, Ch. 14
DIELECTRIC, Ch. 14
MATERIAL, Ch. 14
TIME_STEP, Ch. 17
DIAGNOSE, Ch. 21
OBSERVE, Ch. 22

Example:

The following example illustrates a wide gap cavity with six, 1.5-millimeter diameter struts across the gap. The cavity is intended for use with a very intense beam propagating close to the drift tube wall. Without the space charge and return current on the drift tube wall the beam would space-charge limit and fail to propagate. The strut's inductive property causes it to appear as a metal conductor at low frequencies while simultaneously being transparent to high frequencies. Thus, the strut carries the steady-state beam return current and image charge, so that the beam can propagate across the gap. However it also passes the high-frequency cavity oscillation, allowing the cavity to interact with beam bunching.

```

SYSTEM CYLINDRICAL ;
AREA TUBE_AND_CAVITY POLYGON 0., RTUBE
      ZBGN_GAP, RTUBE          ZBGN_GAP, RIN_CAVITY
      ZBGN_CAVITY, RIN_CAVITY  ZBGN_CAVITY, ROUT_CAVITY
      ZEND_CAVITY, ROUT_CAVITY ZEND_CAVITY, RIN_CAVITY
      ZEND_GAP, RIN_CAVITY     ZEND_GAP, RTUBE
      ZEND, RTUBE ;
LINE GAP CONFORMAL ZBGN_GAP, RTUBE ZEND_GAP, RTUBE ;
STRUT GAP 6 1.5_MM ;

```

16. EMISSION PROCESSES

This chapter covers the following commands:

EMISSION [options]
EMISSION BEAM
EMISSION EXPLOSIVE
EMISSION GYRO
EMISSION HIGH_FIELD
EMISSION PHOTOELECTRIC
EMISSION THERMIONIC
EMIT
EXCLUDE
PHOTON

You can use these commands to describe the emission of charged particles from the surfaces of an object.

Most of the parameters required for the emission models are specified using default values. You can change any of these values using the **EMISSION [options]** command. You can use the rest of the **EMISSION** commands to create customized models of the following emission processes:

BEAM	- an arbitrary, prescribed beam.
EXPLOSIVE	- field-extraction from a surface plasma.
GYRO	- a prescribed beam for gyro devices.
HIGH_FIELD	- high-field (Fowler-Nordheim) emission.
PHOTOELECTRIC	- photoelectric emission.
THERMIONIC	- thermionic emission.

The first three are “artificial” emission processes (as opposed to fundamental processes which involve the work function). In **BEAM** emission, the beam properties are assumed to be known a priori and are simply prescribed, without regard for the underlying fundamental processes. For example, it can be used to model a beam which enters a cavity, without including the processes which create the beam in the first place. In **EXPLOSIVE** emission, one or more species is extracted from an assumed surface plasma, a phenomenon typically encountered in pulsed-power applications. The underlying processes of high-field emission and joule heating which produce the plasma, are considered only phenomenologically. The surface plasma is assumed, but not actually modeled with particles. Instead, particles are created with charge sufficient to cause the normal electric field at the surface to vanish. This zero, or near-zero, surface field is a distinguishing feature of **EXPLOSIVE** emission. **GYRO** emission is really a special case of **BEAM** emission in which the controls have been adapted specifically for gyro (rotating beamlet) applications.

The last three represent fundamental emission processes, in which energy is supplied to overcome a work function. In `HIGH_FIELD` emission, the energy is supplied by the ambient electric field, and results in quantum-mechanical tunneling. The electron yield thus varies with the electric field on the surface, according to the Fowler-Nordheim equation. In `PHOTOELECTRIC` emission, the energy to overcome the work function is supplied by an incident photon. The energy spectrum of the emitted electrons depends on the incident photon spectrum, and a transport code is typically used to compute the electron energy spectrum, which is required input for this emission model. The spatial distribution of the photon flux is specified with a `PHOTON` command, used only in conjunction with `PHOTOELECTRIC` emission. In `THERMIONIC` emission, the energy to overcome the work function is thermal, and the electron yield varies with the surface temperature, according to the Richardson-Dushman equation.

Once an emission model has been created, you can enable emission on a particular object by using the `EMIT` command. Note that the `EMIT` command can be used only with area objects (`AREA`, Ch.10), and not from lines or points, and that the objects must be perfectly conducting (`CONDUCTOR`, Ch. 14). If you don't want the entire surface of the object to emit, you can restrict the actual area of emission with `EXCLUDE` commands. Or, you can define the structure as multiple objects using multiple `AREA` commands, and then `EMIT` only from the desired object.

The following table presents some general guidelines on the selection of emission algorithms. Alternatives are best used only with a good understanding of the algorithm and the underlying physics. For example, `EXPLOSIVE` emission is recommended for Child-Langmuir applications, since the current density is critically dependent upon the surface dynamics (normal electric field and charge density). An attempt to use `BEAM` emission in such applications will either under-emit, producing a non-zero surface field, or over-emit, causing low-velocity particles to pile up at the surface which prevents resolution. It is unlikely that the proper Child-Langmuir dynamics will be obtained in either event.

General conditions for algorithm selection.

EMISSION process	BEAM	EXPLOSIVE	GYRO	HIGH_ FIELD	PHOTO- ELECTRIC	THERMIONIC
magnetic insulation		X				
Child Langmuir		X				
diodes		X				
field-emitter arrays				X		
prescribed beams	X					
prescribed gyro-beams			X			
beam formation		X			X	X
thermionic cathodes						X

This page is intentionally left blank.

EMISSION [options] Command

Function: Specifies options which may be used in any emission process command.

Syntax:

EMISSION process arguments

```
[ MODEL model_name ]
[ SPECIES species_name ]
[ NUMBER creation_rate(t,x1,x2) ]
[ TIMING step_multiple ]
[ SURFACE_SPACING { RANDOM, UNIFORM } ]
[ OUTWARD_SPACING { RANDOM, FIXED }
  displacement(t,x1,x2) ] ;
```

Arguments:

process	-	emission process (BEAM, EXPLOSIVE, GYRO, HIGH_FIELD, PHOTOELECTRIC, and THERMIONIC).
arguments	-	arguments for specific emission processes.
model_name	-	name of emission model, user-defined.
species_name	-	particle species name (ELECTRON, PROTON, or defined in SPECIES command).
creation_rate	-	particle creation rate (particles/cell/time step), integer or defined in FUNCTION command.
step_multiple	-	LORENTZ time-step multiple (integer).
displacement	-	maximum displacement outward from the surface (m), constant or function defined in FUNCTION command.

Defaults:

The following option default values are set. If they are adequate for your requirements, no changes are needed. If you wish to use different values, simply enter the desired keywords and new values when you enter the EMISSION command.

model_name	-	set to the name of the process (e. g., BEAM, etc.).
species_name	-	ELECTRON.
creation_rate	-	3 particles / cell / emission time step.
step_multiple	-	1 Lorentz time_step / emission time step.
displacement	-	initial_velocity x step_multiple x SYS\$DTIME.

The default values for both spacing options (SURFACE_SPACING and OUTWARD_SPACING) are set to RANDOM. (For GYRO emission, the SURFACE_SPACING default is UNIFORM.)

EMISSION [options] Command**Description:**

The various emission processes (BEAM, EXPLOSIVE, etc.) have some common control options, all of which are set to default values. These common features are described here, while those features unique to individual emission processes are described in the commands which immediately follow.

The emission `model_name` allows you to specify a unique, alphanumeric name for each emission model that you create. This name will be referred to in `EMIT` and `EXCLUDE` commands. The default `model_name` is the same as the process (e.g., BEAM, EXPLOSIVE, etc.). However, if you create more than one model using the same process, then you must give each model an individual, unique `model_name`.

The `species_name` can be used to specify `ELECTRON` or `PROTON`, which are permanently stored in the internal species table. If another choice is desired, the desired particle parameters must be specified in a `SPECIES` command. The default `species_name` is `ELECTRON`.

The `creation_rate` is the number of discrete particles which will be created per cell at each emission time step. Note that this has nothing to do with the amount of charge created, but is strictly a statistical parameter. The data entry can be an integer constant, or it can be a function which allows greater control of particle statistics in space and time. The default value is three.

The emission time step is expressed as an integer `step_multiple` of the `LORENTZ` (not `MAXWELL`) time step. The default value is unity.

The `SURFACE_SPACING` option allows you to specify precisely how particles will be spaced going along the surface. Its main effect is on the way trajectories appear in plots. The default value is `RANDOM`, which provides a physical appearance. As an alternative, the selection of `UNIFORM` generally produces continuous striations, which may aid in interpreting phase-space output without adversely affecting the physics.

The `OUTWARD_SPACING` option allows you to specify precisely how particles will be spaced going outward (away from the surface). Usually, it is best not to position particles precisely on the surface, but to give them some small, outward displacement. (This option can have an important effect on the solution.) Two choices are available for distributing particles within this displacement. The default is `RANDOM`, which spaces particles randomly within the displacement distance. This choice generally provides superior charge continuity, making maximum statistical value of the particles. The alternative is `FIXED`, which positions all particles precisely at the outermost extent of the displacement. The default value for displacement equals the product of the particle velocity, the Lorentz time step, and the emission `step_multiple`.

EMISSION [options] Command**Restrictions:**

Up to ten EMISSION commands of all types may be used in a simulation.

See Also:

FUNCTION, Ch. 6
EMISSION BEAM, Ch. 16
EMISSION EXPLOSIVE, Ch. 16
EMISSION GYRO, Ch. 16
EMISSION HIGH_FIELD, Ch. 16
EMISSION PHOTOELECTRIC, Ch. 16
EMISSION THERMIONIC, Ch. 16
EMIT, Ch. 16
EXCLUDE, Ch. 16
LORENTZ, Ch. 18
SPECIES, Ch. 18

EMISSION BEAM Command

Function: Specifies a beam emission model.

Syntax:

```
EMISSION BEAM current_density(t,x1,x2) beam_voltage(t,x1,x2)
[ THERMALIZATION fraction(t,x1,x2) ]
[ COSINES x1cos(t,x1,x2) x2cos(t,x1,x2) x3cos(t,x1,x2) ]
[options] ;
```

Arguments:

current_density - incident current density (A/m**2), real constant or function defined in a FUNCTION command.

beam_voltage - beam voltage (eV), real constant or function defined in a FUNCTION command.

fraction - thermalization fraction, real constant or function defined in a FUNCTION command.

x1cos, etc. - directional cosines, real constant or function defined in a FUNCTION command.

options - see EMISSION [options] command.

Defaults:

None, except for the option defaults. The command must be entered for this model to be used.

Description:

The incident current_density for beam emission is described by the equation,

$$\frac{d^2 q}{dA dt} = J(t, x_1, x_2)$$

which allows the current density (A/m²) to depend on time, t, and the spatial coordinates, x1 and x2. The initial kinetic energy and momentum for each particle are computed relativistically from the beam_voltage,

$$T = q V(t, x_1, x_2)$$

which is also allowed to be a function of time and the two spatial coordinates. In the absence of a transformation with the COSINES option, the momentum will be directed outward, normal to the emitting surface.

EMISSION BEAM Command

For obvious reasons, no defaults are provided for these two functions. Therefore, both `current_density` and `beam_voltage` must be entered in an EMISSION BEAM command before this model can be used.

The options provide the possibility of thermalizing the beam or giving it a direction (which is not directed normally outward) from the surface. The THERMALIZATION option allows the specified fraction of beam energy to be distributed in the transverse coordinates. The energy in each coordinate is determined randomly and the dominant (normal) component is re-normalized to preserve the specified `beam_voltage`. The COSINES option causes the beam to propagate in a direction not normal to the emitting surface. The directional cosines are applied to the momentum computed from the `beam_voltage` to obtain the non-normal components. Preservation of the specified `beam_voltage` is not checked; therefore, care should be taken that the directional cosines supplied are a unitary transformation.

Restrictions:

Up to ten EMISSION commands of all types may be used in a simulation.

See Also:

FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16
EXCLUDE, Ch. 16

References:

B. Goplen and R. Worl, "Numerical Simulations of a Monotron Oscillator Cavity Traveling Wave Tube," Mission Research Corporation Report, MRC/WDC-R-098, July 1985.

F. Friedlander, A. Karp, B. Gaiser, J. Gaiser, and B. Goplen, "Transient Analysis of Beam Interaction with Antisymmetric Mode in Truncated Periodic Structure Using 3-Dimensional Computer Code SOS," Trans. IEEE, ED-2, Vacuum Electron Devices, November 1986.

B. Goplen, R. Worl, W. M. Bollen, and J. Pasour, "Vircator Modulation Study," Mission Research Corporation Report, MRC/WDC-R-139, November 1987.

Examples:

This example illustrates the use of beam-emission to simulate a gated emission gun. The X2259A klystron amplifier is an axisymmetric power tube which uses a gated emission Pierce gun to supply the electron beam. In this example, a spherical cathode surface is represented using a non-uniform spatial grid. The EMISSION BEAM command is used with a bunch shape function

EMISSION BEAM Command

to simulate the emission properties of the gun. The Pierce gun is controlled with an external RF circuit driven at the desired bunching frequency. Here, the function, SHAPE, is used to modulate the electron beam, which is emitted only during the positive half of the RF cycle. Note that inclusion of the step function in SHAPE limits the beam to a single bunch.

```

!   Define Class B bunch shape function
    FREQ = 419.005E6 ;           TOPI = 6.283185307 ;
    OMEGA = TOPI*FREQ ;
    PEAKJ = 0.968E4 ;           ! units are A/m**2
    PERIOD = 1.0/FREQ ;
    FUNCTION SHAPE(T)=PEAKJ*MAX(SIN(OMEGA*T),0.0)
                                *STEP(T,PERIOD) ;

!
!   Define beam voltage
    V = 71 ;                     ! corresponds to 5x106 m/sec

!
!   Specify the beam-emission algorithm
    EMISSION BEAM SHAPE V ;

!
!   Enable emission from the cathode
    EMIT BEAM CATHODE ;

```

Figure 16-1 illustrates creation of the electron bunch using these commands. Once the electrons are created, they are accelerated by an applied voltage pulse applied at the anode-cathode gap (dashed line). The spherical cathode and focusing electrodes provide the basic beam profile. Figure 16-1 also illustrates most of the resonator cavity used for extracting RF from the bunched beam.

EMISSION BEAM Command

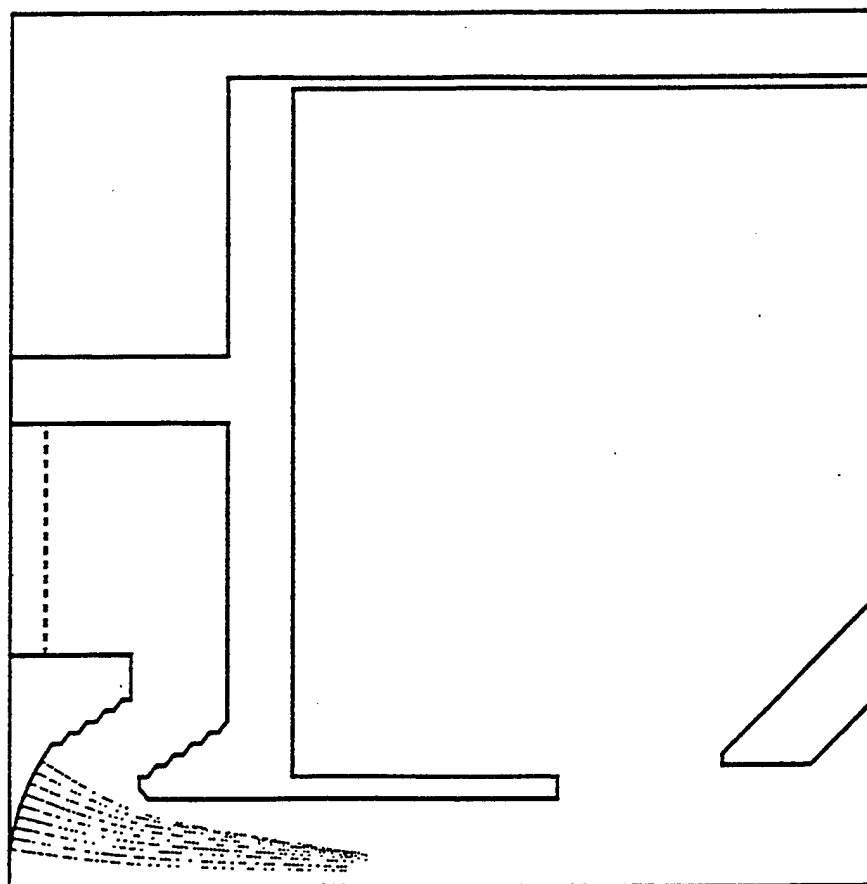


Figure 16-1. Class B beam from the X2259A klystron.

EMISSION EXPLOSIVE Command

Function: Specifies an explosive-emission process.

Syntax:

```
EMISSION  EXPLOSIVE
[ THRESHOLD threshold_field(t,x1,x2) ]
[ RESIDUAL  residual_field(t-tb,x1,x2) ]
[ PLASMA   formation_fraction(t-tb,x1,x2) ]
[ VELOCITY initial_velocity(t,x1,x2)]
[ MINIMUM_CHARGE particle_charge(t-tb,x1,x2) ]
[ MAXIMUM_CURRENT_DENSITY current_density(t-tb,x1,x2) ]
[ options ] ;
```

Arguments:

threshold_field	- breakdown field threshold (V/m), constant or defined in FUNCTION command.
residual_field	- electric field residual (V/m), constant or defined in FUNCTION command.
formation_fraction	- plasma formation fraction ($0 < f < 1$), constant or defined in FUNCTION command.
initial_velocity	- particle initial velocity (m/sec), constant or defined in FUNCTION command.
particle_charge	- minimum particle charge (Coul), constant or defined in FUNCTION command.
current_density	- maximum current density (A/m^2), constant or defined in FUNCTION command.
options	- see EMISSION [options] command.

Defaults:

In addition to the option defaults, the following additional defaults are employed.

threshold_field	- 2.3×10^7 V/m.
residual_field	- 0 V/m.
formation_fraction	- $f = t/\tau$ ($t < \tau$) or 1 ($t > \tau$), with $\tau = 5 \times 10^{-9}$ sec.
initial_velocity	- 5×10^6 m/sec.
particle_charge	- 0 Coul.
current_density	- ∞ A/m^2 .

If all of the default values are adequate for your requirements, simply enter EMISSION EXPLOSIVE. If values other than the defaults are desired, include the relevant keywords and new values.

EMISSION EXPLOSIVE Command**Description:**

Explosive emission results from plasma formation on a material surface. Almost any surface exhibits microscopic protrusions, or “whiskers.” When exposed to large voltages, electric field enhancement at the whiskers can cause significant high-field emission (quantum-mechanical tunneling overcoming the work function). Subsequently, the whisker may dissipate due to Joule heating, resulting in the formation of a plasma on the material surface. This surface plasma will typically “emit” under the influence of the ambient electric field, with the species extracted from the plasma being determined by the sign of the field.

Our model largely ignores the physical details of the plasma formation process, relying instead on a phenomenological description. However, the particle emission itself is based upon Child’s law of physics, specifically, the normal electric field vanishing at the plasma surface.

In our phenomenological treatment of plasma formation, breakdown can occur only if the normally directed field at the half cell, E_c , exceeds some specified breakdown field_threshold, or

$$|E_c| > E_{\text{threshold}}$$

This test is performed continuously for every surface cell on the emitting object. If the field at a particular cell exceeds the threshold, then that cell is said to “break down.” (A single, non-emitting cell between two emitting cells is also allowed to break down, even if the threshold is not exceeded.) The time of breakdown, t_b , is recorded for each cell that breaks down. Subsequently, every cell has its own history and is treated independently.

Once initiated by breakdown, plasma formation in a cell is assumed irreversible. However, the cell does not become fully effective instantly; instead, plasma formation is assumed to occur gradually, so that the effectiveness of the cell increases from zero to unity according to a user-specified formation_fraction. (The default formation_fraction is linear with a 5 nsec risetime.)

It is important to realize that we do not actually create a surface plasma using particles (an approach which leads to poor results in many applications). Instead, we use our phenomenological model and Gauss’s law to calculate the charge which would be drawn away from the surface. Thus, a cell which has broken down may “emit” charged particles due to the influence of the ambient electric field. If the field is of one sign, electrons will be emitted; if the other sign, then protons (or positively charged ions) will be emitted. (Note that each species must be enabled in separate EMISSION and EMIT commands.) The calculation is based upon application of Gauss’s law to the half cell, allowing for a small residual field at the surface. The result is

$$\frac{dq}{dA} = \epsilon_0 f(t - t_b)(E_c - E_r) - \rho \, dx$$

EMISSION EXPLOSIVE Command

where f is the plasma formation_fraction (note the dependence on t_b), and p is the existing charge density at the surface (which accounts for incoming as well as outgoing particles of all species).

Additional restrictions may be placed on the created particles in the form of maximum current_density and minimum particle_charge, according to

$$\left| \frac{dq}{dA dt} \right| < J_{\text{maximum}}$$

and

$$|dq/nl| > Q_{\text{minimum}}$$

All of the explosive emission arguments allow either a constant or a function to be entered. Functions can depend on time, t , in seconds and the spatial coordinates, $x1$ and $x2$, in meters. Note that only the threshold_field is a function of absolute, or simulation, time. The other four arguments (if functions) are always referenced to the time of cell breakdown, which generally varies with location on the surface.

Restrictions:

1. Up to ten EMISSION commands of all types may be used in a simulation.
2. The initial_velocity and initial_distance SPACING option should never both be set to zero, since this will cause particles to become trapped on the surface.

See Also:

FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16
EXCLUDE, Ch. 16

References:

R. B. Miller, An Introduction to the Physics of Intense Charged Particle Beams, Plenum Press, 1982.

B. Goplen, R. E. Clark, and S. J. Flint, "Geometrical Effects in Magnetically Insulated Power Transmission Lines," Mission Research Corporation Report, MRC/WDC-R-001, April 1979.

EMISSION EXPLOSIVE Command

D. B. Seidel, B. C. Goplen, and J. P. Van Devender, "Simulation of Power Flow in Magnetically Insulated Convolutes for Pulsed Modular Accelerators," presented at the 1990 Fourteenth Pulse Power Modulator Symposium, June 3-5, 1980.

B. Goplen and R. Worl, "Pulsed Power Simulation Problems in MAGIC," Mission Research Corporation Report, MRC/WDC-R-124, February 1987.

B. Goplen, R. Worl, J. McDonald, and R. Clark, "A Diagonal Emission Algorithm in MAGIC," Mission Research Corporation Report, MRC/WDC-R-152, December 1987.

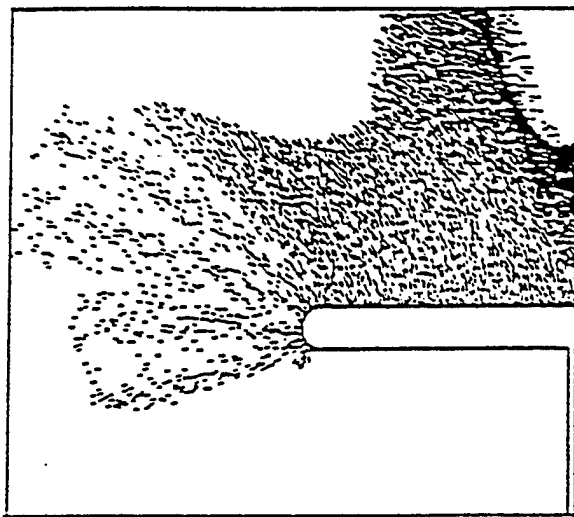
Examples:

This example illustrates explosive emission from the Aurora cathode. It uses the default values for all parameters. Thus, the only commands required are,

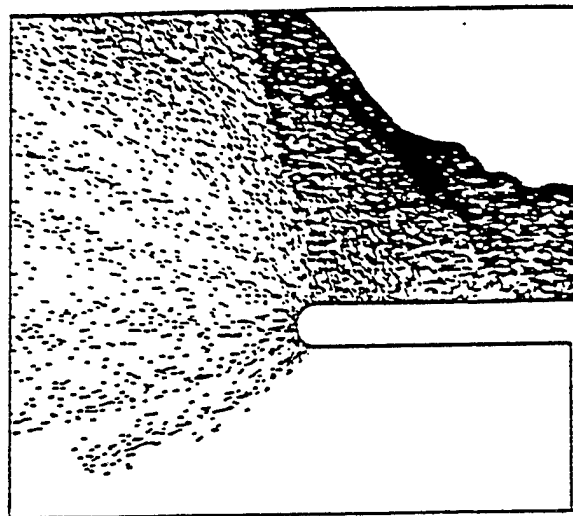
```
EMISSION EXPLOSIVE ;  
EMIT EXPLOSIVE CATHODE ;
```

Figure 16-2 illustrates four stages of beam development in the Aurora diode.

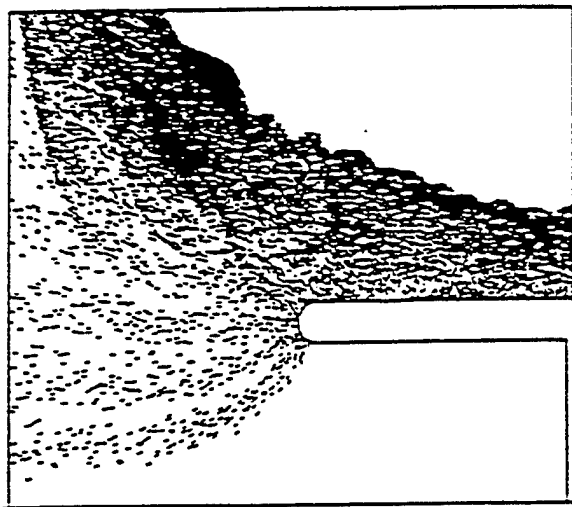
EMISSION EXPLOSIVE Command



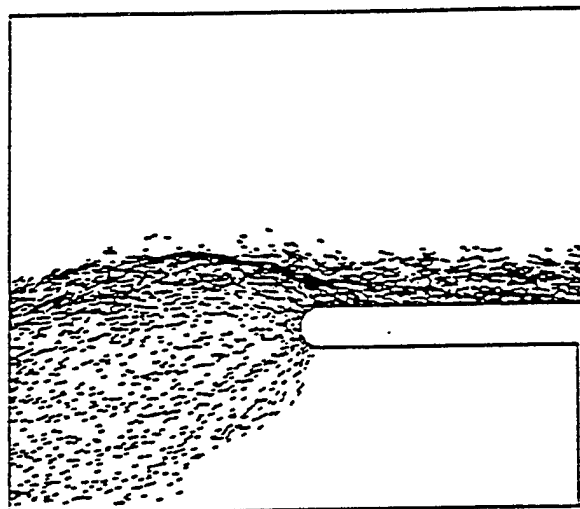
(a) trajectories at 8 nsec



(b) trajectories at 9 nsec



(c) trajectories at 10 nsec



(d) trajectories at 15 nsec

Figure 16-2. Field emission from the Aurora diode.

EMISSION GYRO Command

Function: Specifies a gyro-emission process.

Syntax:

```
EMISSION  GYRO
          beam_current(t)  magnetic_field
          p_longitudinal(t) p_transverse(t) x_center(t)
          [ options ] ;
```

Arguments:

beam_current - total beam current (A), constant or defined in FUNCTION command.

magnetic_field - magnetic field at emitting surface (T).

p_longitudinal - longitudinal momentum (m/sec), constant or defined in FUNCTION command.

p_transverse - transverse momentum (m/sec), constant or defined in FUNCTION command.

x_center - location of guiding center (m), constant or defined in FUNCTION command.

options - see EMISSION [options] command.

Defaults:

None, except for the option defaults. The command must be entered for this model to be used.

Description:

The gyro-emission algorithm produces a beam of particles gyrating about a guiding center axis parallel to the externally applied magnetic field. Particles are emitted from a circle on the surface, which must be conformal with a spatial coordinate. The guiding center axis must be normal to the surface. The radius of the circle will be determined from the specified field and momentum components. Three components of momentum are assigned to each particle such that its guiding center travels along the same axis, regardless of where it was emitted.

The gyro-emission arguments begin with the incident beam_current. You must also specify the magnetic_field on the emitting surface, and this should be consistent with the externally supplied magnetic field (PRESET, Ch. 19). The initial momentum is specified as two components of the relativistic momentum, gamma times velocity, in units of m/sec. The p_longitudinal component is parallel to the guiding center axis and the p_transverse component is perpendicular to the axis.

EMISSION GYRO Command

The function, `x_center`, specifies the guiding center coordinate in the coordinate perpendicular to the axis. Normally, the guiding center axis will be aligned with X1; in this case, `x_center` will be the distance in X2 axis at which emission is centered. Note that four of the data entries are allowed to be functions of time, and that even random fluctuations (using the intrinsic `RANDOM` function) can be introduced by this means.

Restrictions:

1. Up to ten `EMISSION` commands of all types may be used in a simulation.
2. Gyro-emission is presently limited to the cartesian (x,y) and cylindrical (z,r) coordinate systems.
3. The `magnetic_field` specified should be consistent with any external magnetic fields (`PRESET`, Ch. 19).

See Also:

FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16
EXCLUDE, Ch. 16

Examples:

The cyclotron auto-resonant maser (CARM) is an amplifier in which the gain mechanism is the resonance interaction between a relativistic electron beam and an electromagnetic wave propagating in a waveguide and immersed in a strong axial magnetic field. The CARM interaction depends strongly on beam energy, axial velocity, waveguide dimension, magnetic field strength, and the electromagnetic mode. The following example illustrates the use of the `EMISSION GYRO` command in a simulation of CARM interaction in a cylindrical geometry. The electron beam has a kinetic energy of 766 keV, the waveguide mode is a TE10 at 103 GHz, and the gain at saturation was 26 dB. The following commands were employed and Figure 16-3 illustrates the resulting electron trajectories.

```
EMISSION GYRO 2500 2.024 6.2E8 3.0E8 0.002735 ;
```


EMISSION GYRO Command

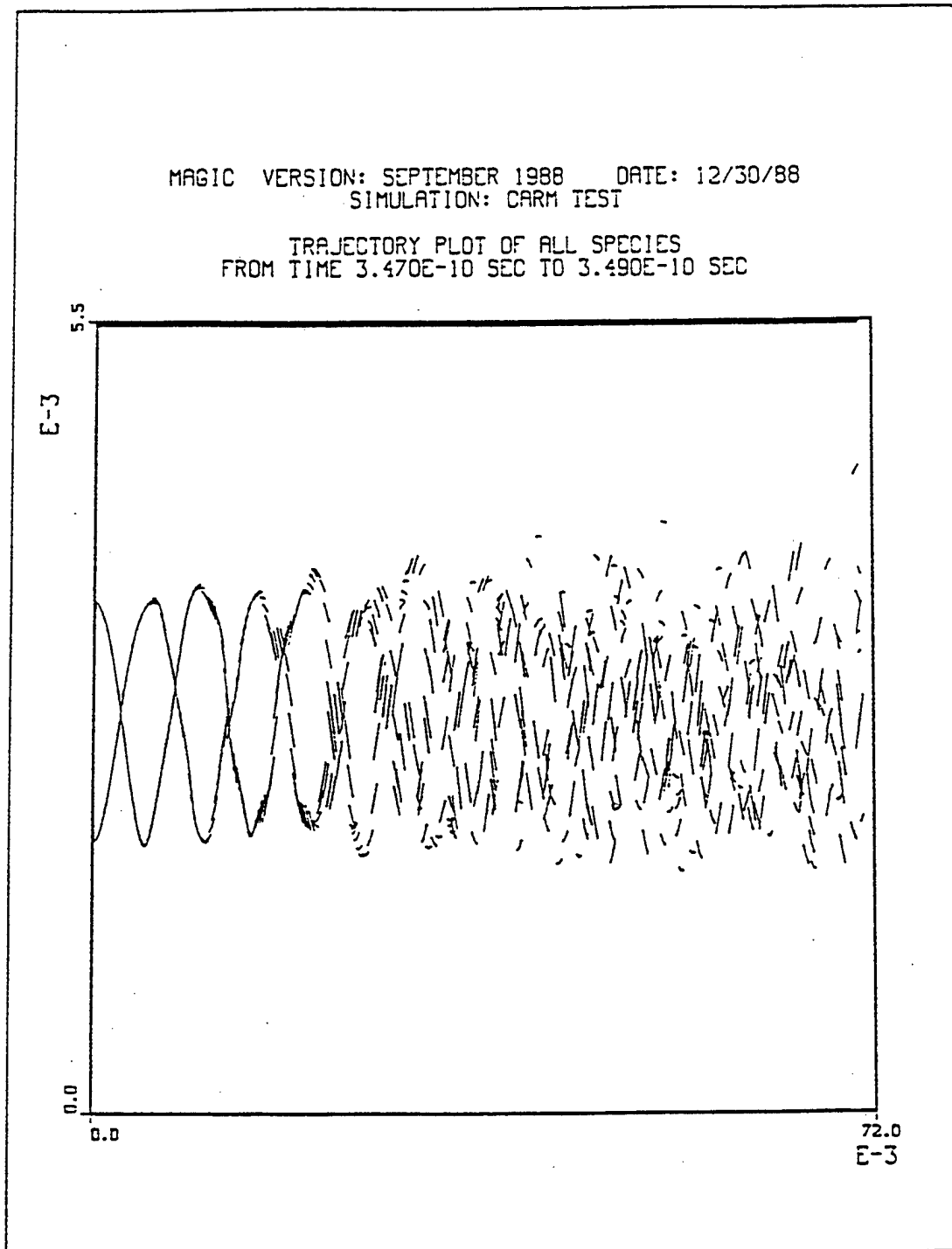


Figure 16-3. Trajectory plot of CARM electrons.

EMISSION HIGH_FIELD Command

Function: Specifies a high-field emission process.

Syntax:

```
EMISSION HIGH_FIELD a b phi(t,x1,x2)
[ options ] ;
```

Arguments:

a	-	Fowler-Nordheim constant, A (A/m).
b	-	Fowler-Nordheim constant, B (m ⁻¹ V ^{-1/2}).
phi	-	work function (eV), constant or defined in FUNCTION command.
options	-	see EMISSION [options] command.

Defaults:

None, except for the option defaults. The command must be entered for this model to be used.

Description:

The basic HIGH_FIELD emission process is described by the Fowler-Nordheim equation,

$$\frac{d^2 q}{dA dt} = \frac{A E_s}{\phi t(y)^2} \exp\left(\frac{-Bv(y)\phi^{3/2}}{E_s}\right),$$

where A and B are the Fowler-Nordheim constants. The work function, ϕ , may be either a constant or a function with allowed dependence on time, t, and the spatial coordinates, x1 and x2.

The remaining variables are computed internally. The normal electric field at the surface, E_s , is computed from the application of Gauss's law to the half-cell immediately above the emitting surface, or

$$E_s = (E_c A_c - q/\epsilon_0) / A_s ,$$

where E_c is the electric field at the half-grid, A_c and A_s are the cell areas at half-grid and surface, respectively, and q is the existing charge in the half-cell.

EMISSION HIGH_FIELD Command

The functions $t(y)$ and $v(y)$ are approximated by

$$t(y)^2 = 1.1$$

$$v(y) = 0.95 - y^2$$

$$y = 3.79 \times 10^5 E_s^{1/2} / \phi,$$

where y is the Schottky lowering of the work function barrier.

Restrictions:

Up to ten EMISSION commands of all types may be used in a simulation.

See Also:

FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16
EXCLUDE, Ch. 16

Examples:

This example illustrates Fowler-Nordheim emission from a field emitter array (FEA). The geometry is a Lincoln Lab wedge field emitter triode shown in Figure 16-4. The tip radius was assumed to be 100 Angstroms, which was modeled using nonuniform spacing in cartesian coordinates. The gate and anode voltages were taken to be 40 and 100 volts, respectively.

The emission model for this example creates three particles per cell on every time step. The particles have an initial energy of zero and are initially positioned 10^{-11} m from the surface and centered (uniform) transversely. The work function was assumed be five eV. This model, named STD, is specified in the following commands:

```
delta = 1.0E-9 ;
A = 1.5414E-6 ; B = 6.8308E+9 ; PHI = 5.0 ;
EMISSION HIGH_FIELD A B PHI
      OUTWARD_SPACING UNIFORM delta
      MODEL STD ;
      EMIT STD WEDGE ;
```

Electron trajectories from the wedge are visible in Figure 16-4.

EMISSION HIGH_FIELD Command

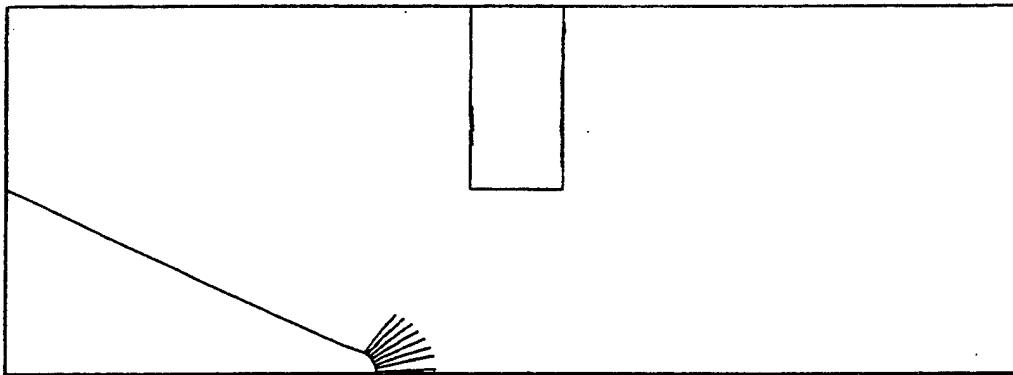


Figure 16-4. Trajectories of Lincoln Lab Wedge Field Emitter. The fine grid (femtosecond time step) and low-velocity particles provide an ideal application for the quasi-static electromagnetic algorithm.

EMISSION PHOTOELECTRIC Command

Function: Specifies a photoelectric emission process.

Syntax:

```
EMISSION PHOTOELECTRIC photon_name f(e) yield photon_energy
[options] ;
```

Arguments:

photon_name	-	photon source model, defined in PHOTON command.
f(e)	-	electron energy function (electron/kev), defined in FUNCTION command (type DATA only).
yield	-	conversion yield (electrons/photon).
photon_energy	-	average photon energy (keV).
options	-	see EMISSION [options] command.

Defaults:

None, except for the option defaults. The command must be entered for this model to be used.

Description:

The photoelectric emission process in MAGIC is described by the equations,

$$\frac{d^5 q}{dA dt dE \sin \theta d\theta} = \frac{1}{\pi} \eta s(t) f(E) \cos \theta$$

where

$$\eta = -41.87 \eta' \chi(r) / \epsilon .$$

Here, E represents the electron energy (keV), θ is the polar angle with respect to the surface normal, η is the conversion yield (electrons/photon), $\chi(r)$ is the fluence (cal/cm^2), ϵ is the average photon energy (keV), $s(t)$ is the photon pulse time history (sec^{-1}), and $f(E)$ is the electron energy distribution (electrons/keV).

Both the photon time history, $s(t)$, and the fluence, $\chi(r)$, must be specified in a PHOTON command. The photon source model must reference the name used in the PHOTON command. The electron energy function, $f(E)$, must be specified using type DATA in a FUNCTION command. The data will consist of pairs representing groups of electron energy (keV) vs relative group normalization. Only relative normalization is required for the energy distribution; the correct

EMISSION PHOTOELECTRIC Command

normalization will be automatically supplied. The angular distribution, $\sin\theta \cos\theta$, is built into the algorithm using a ten-group, equal probability representation. The remaining arguments are the conversion yield (electrons/photon), and the average photon_energy (kev), representing η and ϵ respectively.

Restrictions:

1. Up to ten EMISSION commands of all types may be used in a simulation.
2. The EMISSION PHOTOELECTRIC command can only be used with the species type, ELECTRON.

See Also:

FUNCTION, Ch. 6
 EMISSION [options], Ch. 16
 EMIT, Ch. 16
 EXCLUDE, Ch. 16
 PHOTON, Ch. 16

References:

A. McIlwain and B. Goplen, "Representation of Photoemission from Blackbody Spectra," Science Applications, Inc. Report, SAI-75-516-AQ, January 1976.

B. Goplen, J. Brandenburg, and R. Worl, "Canonical SGEMP Simulation Problems," Mission Research Corporation Report, MRC/WDC-R-109, also, AFWL-TN-86-57, March 1986.

Examples:

We consider emission from silicon caused by a symmetric, triangular, 10 nsec FWHM pulse of five keV blackbody originating at coordinates (0,0) and propagating as a plane wave in the positive x direction. The yield for silicon is 10^3 electrons/photon, and the average photon energy is 13.6 keV. The electron energy distribution consists of five equally probable groups with midpoint energies of 1, 2, 4, 7, and 10 keV.

The model specifies three particles per cell and a creation frequency that matches the electromagnetic time step. Creation is fixed at a distance of 10^{-5} m from the surface but is random transversely. This model, named STD, is specified in the following commands:

```
C PHOTO EMISSION EXAMPLE ;
!
! Define photon time history
FUNCTION S DATA 3 0.0,0.0 1.0E-8,1.0E+8
```

EMISSION PHOTOELECTRIC Command

```
                2.0E-8,0.0 ;
!
! Define electron energy distribution
FUNCTION F DATA 5  1.0,0.2  2.0,0.2  4.0,0.2
                  7.0,0.2  10.0,0.2 ;
!
! Define photon source
PHOTON SOURCE 1.0 S 0.0 PLANE-X1 0.0 0.0 ;
!
! Specify the photoelectric emission model
CONV = 1.0E-3 ;
EAVE = 13.6 ;
EMISSION PHOTOELECTRIC SOURCE F CONV EAVE
        OUTWARD_SPACING RANDOM 1E-5;
!
! Emit from the spatial object labeled silicon
EMIT PHOTOELECTRIC SILICON ;
```

EMISSION THERMIONIC Command

Function: Specifies a thermionic emission process.

Syntax:

```
EMISSION THERMIONIC
  work_function(t,x1,x2) temperature(t,x1,x2)
  [ options ] ;
```

Arguments:

- work_function - work function (eV), constant or defined in FUNCTION command.
- temperature - temperature (deg K), constant or defined in FUNCTION command.
- options - see EMISSION [options] command.

Defaults:

None, except for the option defaults. The command must be entered for this model to be used.

Description:

The basic thermal emission process in MAGIC is described by the Richardson-Dushman equation,

$$\frac{d^2q}{dAdt} = A_0 T^2 \exp\left(\frac{-\phi}{kT}\right)$$

where k is the Boltzmann constant and A_0 is the Dushman parameter ($1.204 \times 10^6 \text{ A/m}^2 \text{ Kelvin}^2$). The work function, ϕ , may be either a constant or a function with allowed dependence on time, t , and the spatial coordinates, $x1$ and $x2$. The same is true of the temperature.

Restrictions:

Up to ten EMISSION commands of all types may be used in a simulation.

See Also:

FUNCTION, Ch. 6
EMISSION [options], Ch. 16
EMIT, Ch. 16
EXCLUDE, Ch. 16

EMIT Command

Function: Enables emission from the perimeter of a spatial area.

Syntax:
EMIT model_name area_name ;

Arguments:
model_name - name of emission model, defined in EMISSION command.
area_name - name of spatial area, defined in AREA command.

Description:

The EMIT command enables particle creation everywhere on the perimeter of an area object. An EMIT command must be used for particle emission to occur. Also, emission can occur only from area objects (not points or lines), and the specified area must be perfectly conducting (CONDUCTOR, Ch. 14).

Only two arguments are required. The first is the model_name specified in an EMISSION command, and the second is the area_name specified in an AREA command. Note that emission will be enabled over the entire surface of the specified object. If creation is to be restricted from certain spatial regions, these must be specified in EXCLUDE commands.

Restrictions:

1. Up to thirty EMIT commands may be used in a simulation.
2. Only AREA spatial objects can emit.
3. The spatial object must be specified as perfectly conducting in a CONDUCTOR command.
4. Non-emitting regions of emitting surfaces must be designated with EXCLUDE commands.

See Also:

AREA, Ch. 10
CONDUCTOR, Ch. 14
EMISSION [options], Ch. 16
EXCLUDE, Ch. 16

EMIT Command**Examples:**

To cause a beam-emission model named STD to be applied to a spatial object named KATHODE, the command is

```
EMIT STD KATHODE ;
```

EXCLUDE Command

Function: Excludes emission from specified spatial areas.

Syntax:
`EXCLUDE { model_name, ALL } area_name ;`

Arguments:
`model_name` - name of emission model, defined in EMISSION command.
`area_name` - name of spatial area, defined in AREA command.

Description:

The EXCLUDE command is used to eliminate particle emission from a particular emission model within a specified spatial region. It does not prohibit particles from otherwise entering or existing in such regions; it simply prohibits emission.

The emission model `model_name` must be specified. This provides the capability to exclude emission from one model while allowing it from another. If ALL is entered instead, no particle creation will be allowed in the specified region.

Restrictions:

1. The number of EXCLUDE commands in a simulation is limited to ninety.
2. The emission `model_name` must be duplicated in one of the emission commands unless ALL is used.
3. EXCLUDE functions only on a cell-by-cell basis and will eliminate (or allow) particle creation within an entire cell.

See Also:

EMISSION [options], Ch. 16
EMIT, Ch. 16

Example:

To prohibit particle creation from the beam-emission model, STD, over the spatial region defined by

$$0 \leq x_1 \leq 0.05 \text{ and } 0.8 \leq x_2 \leq 1.0,$$

EXCLUDE Command

one should use the command,

```
AREA NO_EMIT CONFORMAL 0,0.8 0.05,1.0 ;  
EXCLUDE STD NO_EMIT ;
```

PHOTON Command

Function: Specifies photon source in time and space.

Syntax:

```
PHOTON photon_name chi(r) s(t) advance
      { PLANE-X1, PLANE-X2, CYLINDRICAL } point_name ;
```

Arguments:

photon_name	-	name of photon source, user-defined.
chi(r)	-	spatial function, user-defined in FUNCTION command.
s(t)	-	temporal function, user-defined in FUNCTION command.
advance	-	retarded time advancement (m).
point_name	-	name of photon source point, defined in a POINT command.

Description:

The PHOTON command allows the user to specify one or more photon sources to drive photoelectric emission processes. Each model so specified must be given a unique, user-defined, alphanumeric name. The photon_name will be referred to by the EMISSION PHOTOELECTRIC command, which specifies the photoelectric emission model.

The photoelectric-emission process is described by the equation,

$$\frac{d^5 q}{dA dt dE \sin \theta d\theta} = \frac{1}{\pi} \eta s(t) f(E) \cos \theta,$$

where

$$\eta = -41.87(\eta' / \xi) \chi(r).$$

The PHOTON command specifies the spatial function $\chi(r)$, and the temporal function, $s(t)$. The time used in the temporal function is retarded, i.e., reduced by the spatial distance (divided by the speed of light). For both functions, the spatial distance is computed from the photon source point to the point of local emission. The geometrical nature of the photon output must be specified as PLANE-X1 or PLANE-X2 for plane waves or CYLINDRICAL wave in x1 and x2. The retarded time advancement should be used initially to position the wave front near the emitting structure.

PHOTON Command

Restrictions:

1. The number of PHOTON commands in a simulation is limited to five.
2. The temporal function must be normalized to unity.

See Also:

FUNCTION, Ch. 6

EMISSION PHOTOELECTRIC, Ch. 16

Examples:

Consider photoelectric emission caused by a symmetric, triangular, photon pulse of 10 nsec half width, originating at coordinate (0, 0), and propagating as a plane wave in the positive x_1 direction. (This photon source model is arbitrarily given the name, SOURCE.) The conversion yield is 10^{-3} electrons/photon and the average photon energy is 13.6 keV. The resulting electron energy function consists of five equally probable groups at energies of 1, 2, 4, 7, and 10 keV. (This electron energy function is arbitrarily given the name, EFN.)

Particle creation is fixed on, and random over, the emitting cell surface. There are three particles per cell created on every time step. This model is specified in the commands,

```
FUNCTION EFN DATA
    5 1.0 0.2 2.0 0.2 4.0 0.2 7.0 0.2 10.0 0.2;
EMISSION PHOTOELECTRIC SOURCE EFN 1.0E-3 13.6
    SPACING 0.0 RANDOM ;
FUNCTION TEMPORAL DATA
    3 0.0 0.0 1.0E-8 1.0E8 2.0E-8 0.0 ; .
POINT SOURCE_POINT 0., 0. ;
PHOTON SOURCE 1. TEMPORAL 0.0 PLANE-X1 0.0 0.0 ;
```

17. ELECTROMAGNETIC FIELDS

This Chapter covers the following commands:

TIME_STEP
MODE
MAXWELL CENTERED
MAXWELL QUASI_STATIC
MAXWELL HIGH_Q
MAXWELL BIASED

The MAXWELL algorithms solve Maxwell's equations to obtain electromagnetic fields (E1, E2, E3, B1, B2, B3) which vary in time and space. (Other algorithms calculate electrostatic fields and eigenfunctions (POISSON and EIGENMODE, Ch. 19), but only the MAXWELL algorithms calculate time-varying fields.) You can use the MAXWELL commands to select an electromagnetic algorithm and/or parameters, the MODE command to select one of the electromagnetic modes, and the TIME_STEP command to set the time step.

You may not need to use any of these commands. The default electromagnetic solution uses the following parameters:

time step - 80% of centered-difference Courant criterion
mode - BOTH (both TE and TM) modes
algorithm - BIASED (time-biased)

This combination provides a very conservative, robust solution for use with relativistic particles. It is consistent with the particle algorithm defaults (Ch. 18). If you are unsure of your simulation requirements, this is the recommended configuration.

On the other hand, you may elect to use the other electromagnetic algorithms to speed up the simulation, to improve fidelity by matching the algorithm to the physics, or even to suppress certain physical effects to gain insight. Algorithm selection is especially important, and the following table indicates conditions which might favor one algorithm over another. For example, if particles are either absent or non-relativistic, then the CENTERED algorithm generally provides superior speed.

General conditions for algorithm selection.

MAXWELL algorithm	CENTERED	QUASI_ STATIC	HIGH_Q	BIASED
speed required	X			
long time scales		X		
no particles	X			
very slow particles		X		
slow particles	X			
relativistic particles			X	X
cavities and particles			X	
high particle noise				X

A brief explanation of the principal factors which impact the electromagnetic field solution follows.

1. Field definition — The electromagnetic fields (E1, E2, E3, B1, B2, B3) as well as the charge and current densities (Q0, J1, J2, J3) are defined at discrete locations in time and space. Figure 17-1 illustrates the spatial definition in the unit cell. The cell shown is square; however, in practice it will be elongated (GRID, Ch. 11), and the sides will be curved (or non-parallel) in polar and spherical coordinates (SYSTEM, Ch.10). The fields are also defined at discrete values of time (TIME_STEP, Ch. 17), with the electric and magnetic fields separated by half a time step.

2. Time step — All MAXWELL algorithms use a time step to advance the fields in time. The size of the time step has virtually no effect on accuracy, so it is generally advisable to use the largest possible time step to minimize expense. However, a Courant stability criterion limits the size of the time step, and each algorithm has a different limit. Rapid, catastrophic failure results from exceeding this limit. Also, outer boundaries (PORT, Ch. 12) and particle dynamics (LORENTZ, Ch. 18) may effectively limit the size of the electromagnetic time step.

3. Spatial grid — Accuracy in the electromagnetic solution is determined primarily by spatial resolution, i.e., the cell size (GRID and AUTOGRID, Ch. 11) relative to the wavelength. The classic example is that calculated eigenvalues are always down-shifted from their physical values. In addition, there is an upper limit to the frequency which can be supported by the grid. Any wavelength shorter than about six cells will degenerate into noise and be lost from the solution.

4. Damping — Two of the MAXWELL algorithms (HIGH_Q and BIASED) contain damping designed to reduce high-frequency fields (particularly noise from relativistic particles). However, some damping occurs at all frequencies, including those of physical interest. In other words, each algorithm has an intrinsic, frequency-dependent Q. If the algorithm damping exceeds the actual physical damping, incorrect fields may result with adverse effects on results for saturation, energy balance, efficiency, etc.

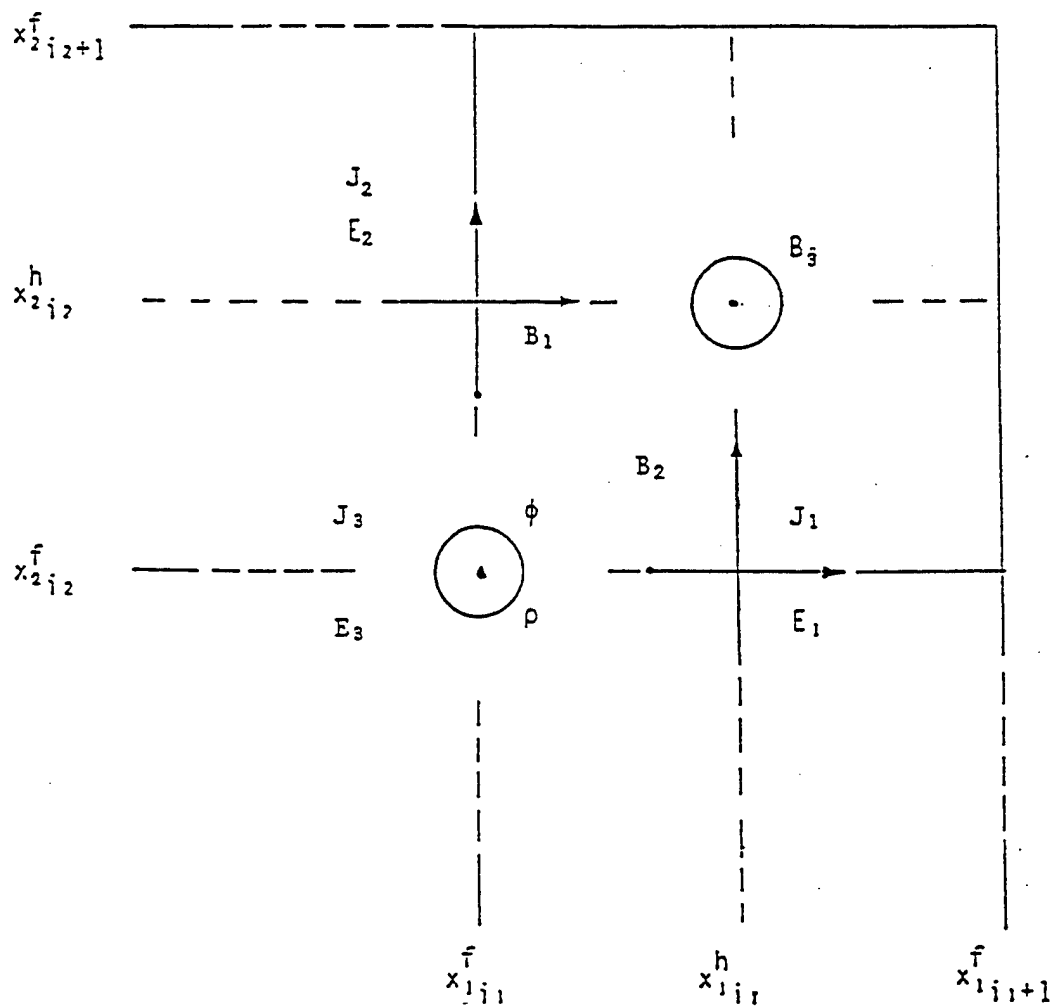


Figure 17-1. Spatial definition of fields.

5. Modes — By default, both the transverse electric (TE) mode and the transverse magnetic (TM) mode are calculated (TEM is a special case of TM). The TE fields (B_1 , B_2 , E_3) and the TM fields (E_1 , E_2 , B_3) may coexist independently, or they may be coupled through particle dynamics and/or unique structures (POLARIZER, Ch. 15). For simulations which require only the TM mode (a common occurrence), eliminating the TE mode will double the speed. (Pure TE mode simulations, which omit all space-charge effects, are also possible, but rarely seen.)

6. Particles — All the MAXWELL algorithms may be used with particles (Ch. 18) to produce self-consistent simulations which fully account for field-particle interactions. All account for space charge exactly, i.e., they satisfy Gauss's law at the cell level. The MAXWELL algorithms differ in that they accommodate different velocity regimes, ranging from very low to highly relativistic. Plasma simulation can be especially difficult when the Debye length is unresolved spatially or the plasma frequency is unresolved temporally.

7. Materials, etc. — All MAXWELL algorithms are compatible with the full range of outer boundaries (Ch. 12) and material properties (Ch. 14 and 15). However, the number of possible combinations is infinite, and diligence (i.e., testing) is always advised.

TIME_STEP Command

Function: Specifies the electromagnetic time step.

Syntax:
TIME_STEP time_step ;

Arguments:

time_step - electromagnetic algorithm time step (sec).

Defaults:

If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used.

Description:

You can use the TIME_STEP command to specify the time_step used in the electromagnetic algorithm. If you do not specify the time_step, a default value equal to exactly 80% of the centered-difference Courant criterion will be used, irrespective of the choice of algorithm (note that the default algorithm is time-biased).

The centered-difference Courant ratio squared stability criterion is given by $\chi < 1$, where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^2 \frac{1}{(\delta x_i)^2}.$$

is the Courant ratio squared, and δx_i is the cell size in meters. Once the spatial grid is completed, it is automatically searched to find the most restrictive cell. Then the default time_step is calculated using $\chi = 0.8$. This value is also entered into the system variable, SYSS\$DTIME. This system variable is accessible in the input, and can then be used to alter the time_step (See Examples, below).

The centered-difference criterion is used as the default because it is safe. It is conservative for all electromagnetic algorithms and coordinate systems. It is safe with all outer boundaries which use a time step. It is also safe when used in the particle kinematics. However, there may be circumstances in which you want to alter the time step. For example, you may wish to take advantage of the larger time_step possible with other algorithms, or you may wish to make a minor adjustment of the time_step for periodicity (see Examples, below). In some cases, it may be necessary to significantly reduce the time_step to accommodate particle resolution requirements (LORENTZ, Ch. 18).

TIME_STEP Command**Restrictions:**

The centered-difference criterion is used to calculate the default time_step, irrespective of the algorithm actually used.

See Also:

AUTOGRID, Ch. 11
GRID, Ch. 11
MAXWELL algorithms, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Examples:

1. Suppose that the default (time-biased) algorithm is satisfactory, but you want to make use of a more aggressive time_step to speed up the calculation. By inspection of the time-biased Courant criterion, you recognize that you can double the time_step. (Note that other factors, such as particle kinematics, etc., might preclude this choice.) Make use of the system variable, SYSSDTIME, which is available once the spatial grid is complete. To double the time_step, the commands are

```
dt = 2.0 * SYSSDTIME ;  
TIME_STEP dt ;
```

After these commands are processed, SYSSDTIME will contain the value of dt and not the original (default) value.

2. Suppose that the magnitude of the default time_step is satisfactory, but that you want to adjust it very slightly so that an exact (integer) number of time_steps equals some specified period. (This is useful in certain cavity-tuning procedures). You can achieve the desired effect with the following commands, using 10 GHz as an example.

```
frequency = 10 GHz ;  
period = 1.0 / frequency ;  
I = period / SYSSDTIME ;  
dt = period / I ;  
TIME_STEP dt ;
```

Note that dt differs only slightly from the default time_step.

MODE Command

Function: Specifies the electromagnetic mode.

Syntax:
MODE { TE, TM, BOTH } ;

Arguments:

none.

Defaults:

The default mode is BOTH. You can use the MODE command to select TE or TM, thus suppressing the other mode.

Description:

You can use the MODE command to select the electromagnetic mode. The modes are transverse electric (TE) with fields B1, B2, and E3, and transverse magnetic (TM) with fields E1, E2, and B3. The transverse electromagnetic (TEM) mode is not a separate mode, but is actually a special case of TM.

Note that the majority of two-dimensional simulations produce only a transverse magnetic (TM) mode. Certain rare cases produce only a transverse electric (TE) mode. If both modes are produced, they may co-exist independently or they may be coupled through particle kinematics and/or unique structures such as the polarizer (POLARIZER, Ch. 15).

If you can suppress one mode, the electromagnetic solution speed will double. To ascertain whether both modes are required, you must consider the interaction between the field and current density components in the Maxwell and Lorentz equations. If you are unsure, use the default and inspect the results. If all the fields in either mode are zero, then that mode is not required. Even if a mode exists, you may decide to suppress it for speed if it is not relevant to the dynamics. Also, you may wish to suppress a mode to study particular physical effects. For example, suppressing the TM mode eliminates space-charge.

See Also:

MAXWELL algorithms, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

MODE Command**Examples:**

You can suppress the TE mode with the following command.

```
MODE TM ;
```

MAXWELL CENTERED Command

Function: Specifies centered-difference electromagnetic algorithm.

Syntax:
MAXWELL CENTERED ;

Arguments:

none.

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is BIASED. You can change this with one of the MAXWELL commands. The default mode is BOTH. You can use the MODE command to select TE or TM, thus suppressing the other mode. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

Description:

The MAXWELL CENTERED command specifies a centered-difference solution of the fully time-dependent Maxwell's equations. This algorithm is the simplest of the time-dependent field algorithms. It is suitable for use without particles or with non-relativistic particles.

The centered-difference Courant criterion is given by $\chi < 1$, where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^2 \frac{1}{(\delta x_i)^2}$$

is the Courant ratio squared and δx_i is the cell size in meters. In certain cases, SYMMETRY, Ch. 12, the criterion may be even more restrictive; however, $\chi < 0.80$ is generally safe. Despite the relatively restrictive time step, the centered-difference algorithm is the fastest in terms of covering a given time span.

Another characteristic of the centered-difference algorithm is that it provides no damping at any frequency (the algorithm Q is infinite). Thus, while excellent for purely electromagnetic simulations, center-difference is very susceptible to the high-frequency noise typically produced by relativistic particles. In extreme form, this susceptibility appears as field aliasing (alternating signs in the cell fields). Therefore, we recommend use of the centered-difference algorithm only for purely electromagnetic simulations or with non-relativistic particles.

MAXWELL CENTERED Command**Restrictions:**

1. This algorithm has an upper limit to the time step used.
2. It should be used only in particle-free or non-relativistic simulations.

See Also:

SYMMETRY AXIAL, Ch. 12
MODE, Ch. 17
TIME_STEP, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Examples:

An algorithm designed for speed in Cartesian coordinates with no particles might use centered-difference with a slight increase in the default time step while suppressing the TE mode.

```
MAXWELL  CENTERED ;      ! switch from BIASED to CENTERED
dt = SYS$DTIME * 1.2 ; ! default is 80% of centered max
TIME_STEP  dt  ;      ! dt is now 96% of centered max
MODE  TM  ;          ! suppress the TE mode
```


MAXWELL QUASI_STATIC Command

Function: Specifies quasi-static electromagnetic algorithm.

Syntax:
MAXWELL QUASI_STATIC beta ;

Arguments:

beta - speed-of-light fraction, v/c ($0 < \text{beta} < 1$).

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is BIASED. You can change this with one of the MAXWELL commands. The default mode is BOTH. You can use the MODE command to select TE or TM, thus suppressing the other mode. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

Description:

The MAXWELL QUASI_STATIC command specifies a quasi-static solution of the fully time-dependent Maxwell's equations. The quasi-static algorithm is the same as centered-difference, but uses a speed-of-light artificially reduced by the factor, beta. This allows the time step to be made larger by a factor of $1/\text{beta}$, and this increase makes the algorithm suitable for use with very low-velocity particles.

The quasi-static algorithm is useful in a restricted class of problems in which steady-state or very low frequency behavior is being investigated. In general, this would apply to simulations of either very low-velocity particles ($v \ll c$) or very long-wavelength radiation (wavelength much larger than simulation dimensions). Such simulations are normally plagued by the need for excessively small time steps because of the Courant restriction in an electromagnetic simulation. The quasi-static algorithm introduces a factor in Faraday's Law which relaxes the Courant condition by slowing the speed of light propagation. Ampere's Law is not altered, so both the electrostatic and magnetostatic physical limits are preserved. However, the transients to the quasi-static limits are much slower in the simulation than in reality, and hence only quasi-static results are meaningful when this algorithm is employed.

The Courant stability criterion is given by $\beta \chi < 1$, where β is beta and

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^2 \frac{1}{(\delta x_i)^2}.$$

MAXWELL QUASI_STATIC Command

is the centered-difference Courant ratio squared. Certain geometries are even more restrictive (SYMMETRY AXIAL, Ch. 12). Since useful values of beta are less than unity, the allowable time_step is larger than the centered-difference time_step by the reciprocal of beta. The damping property of the quasi-static algorithm is the same as that for centered difference (the algorithm Q is infinite); however, the total absence of damping is not a problem for low-velocity particle applications.

To use the quasi-static algorithm, you set beta to the desired fraction ($0 < \beta < 1$) of the speed-of-light (Note that $\beta = 1$ recovers the centered-difference algorithm). The time step may then be increased beyond the usual Courant limit by the reciprocal of this fraction. Suggested values of beta are 3 to 10 times the particle beta (v/c), or 10-to-30 times the ratio of the simulation dimensions to the wavelength of the radiation.

Restrictions:

1. This algorithm has an upper limit to the time step used.
2. It should be used only in simulations involving no particles or very low velocity particles.

See Also:

SYMMETRY AXIAL, Ch. 12
 MODE, Ch. 17
 TIME_STEP, Ch. 17
 CONTINUITY, Ch. 18
 LORENTZ, Ch. 18

Examples:

An algorithm designed for very low-velocity particles might use quasi-static with a ten-fold increase in the default time step while suppressing the TE mode.

```
beta = 0.1           ! beta is v/c
MAXWELL QUASI_STATIC beta ; ! switch to QUASI_STATIC
dt = SYS$DTIME/beta ; ! default is 80% of centered max
TIME_STEP dt ;       ! dt is now 800% of centered max
MODE TM ;            ! suppress the TE mode
```

MAXWELL HIGH_Q Command

Function: Specifies high-Q electromagnetic algorithm.

Syntax:
MAXWELL HIGH_Q [gamma [courant_ratio]] ;

Arguments:

gamma - filter factor ($0 < \text{gamma} < 1$).
courant_ratio - centered-difference Courant ratio squared (unitless).

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is BIASED. You can change this with one of the MAXWELL commands. The default mode is BOTH. You can use the MODE command to select TE or TM, thus suppressing the other mode. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

The HIGH_Q algorithm provides defaults (which can be over-ridden) for gamma and courant_ratio, as described below.

Description:

The MAXWELL HIGH_Q command specifies a high-Q solution of the fully time-dependent Maxwell's equations. This algorithm artificially damps electromagnetic fields. It damps all frequencies in the same manner as the time-biased algorithm, but damps less at physical low frequencies, hence the name, "high-Q." It is especially suitable for cases involving relativistic particles and cavities.

The Courant stability criterion is given by

$$\chi < \left[\frac{2 - 3\gamma^2 + 2\gamma^3}{2(1 - 3\gamma^2 + 2\gamma^3)} \right]^{1/2}, \quad 0 \leq \gamma \leq 1/2$$

and

$$\chi < (3\gamma)^{1/2}, \quad 1/2 \leq \gamma \leq 1$$

MAXWELL HIGH_Q Command

where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^2 \frac{1}{(\delta x_i)^2}.$$

is the centered-difference Courant ratio squared. Certain geometries are slightly more restrictive (SYMMETRY AXIAL, Ch. 12). In no case can the time_step exceed $\sqrt{3}$ times the centered-difference stability criterion. Also, it may not be possible to take advantage of the larger time_step due to constraints on outer boundaries (PORT, Ch. 12) and particle dynamics (LORENTZ, Ch. 18).

The damping function is given by

$$F(\beta^2) = (1 - \gamma \beta^2)^2 (1 + 2\gamma \beta^2),$$

where β is the frequency, normalized to the highest value that can be supported by the spatial grid. In comparison with the time-biased algorithm, high-Q produces much less damping at lower frequencies, as shown in the following figure. However, note that some non-physical damping occurs at all non-zero frequencies. In other words, the algorithm itself has a Q which is frequency-dependent and must be accounted for in simulations involving saturation or energy balance.

You may adjust the degree of damping with the filter factor, gamma. A value of zero is equivalent to centered-difference, while a value of one causes maximum damping. The default and recommended value is 0.85. The high-Q algorithm derives the courant_ratio from the grid, but this value may be over-ridden by the user in rare cases, for example, when the cell with smallest grid is outside all simulation boundaries.

References:

MugShots, Vol. 1, No. 2 (May 1992).

Restrictions:

1. This algorithm has an upper limit to the time step used.
2. It is most appropriately used in relativistic particle applications where damping at low and intermediate frequencies would be undesirable, such as high-Q cavities.
3. Be aware that some damping occurs (the algorithm Q is finite) at all frequencies. This damping is completely unrelated to physical loss mechanisms such as resistivity or outgoing waves. In effect, the reciprocals of all Qs (including the algorithm Q) add to produce the effective Q.

MAXWELL HIGH_Q Command**See Also:**

SYMMETRY AXIAL, Ch. 12
MODE, Ch. 17
TIME_STEP, Ch. 17
CONTINUITY, Ch. 18
LORENTZ, Ch. 18

Examples:

An algorithm designed for relativistic particles might use high-Q with a modest increase in the default time step while suppressing the TE mode.

```
MAXWELL  HIGH_Q ;           ! switch from BIASED to HIGH_Q
dt = SYS$DTIME * 1.5 ;      ! default is 80% of centered
TIME_STEP dt ;              ! dt is 120% of centered max
MODE  TM ;                  ! suppress the TE mode
```

Note that a time step which exceeds the centered-difference maximum may lead to trouble with the particle algorithms (LORENTZ, Ch. 18) and outer boundaries (PORT, Ch. 12).

MAXWELL HIGH_Q Command

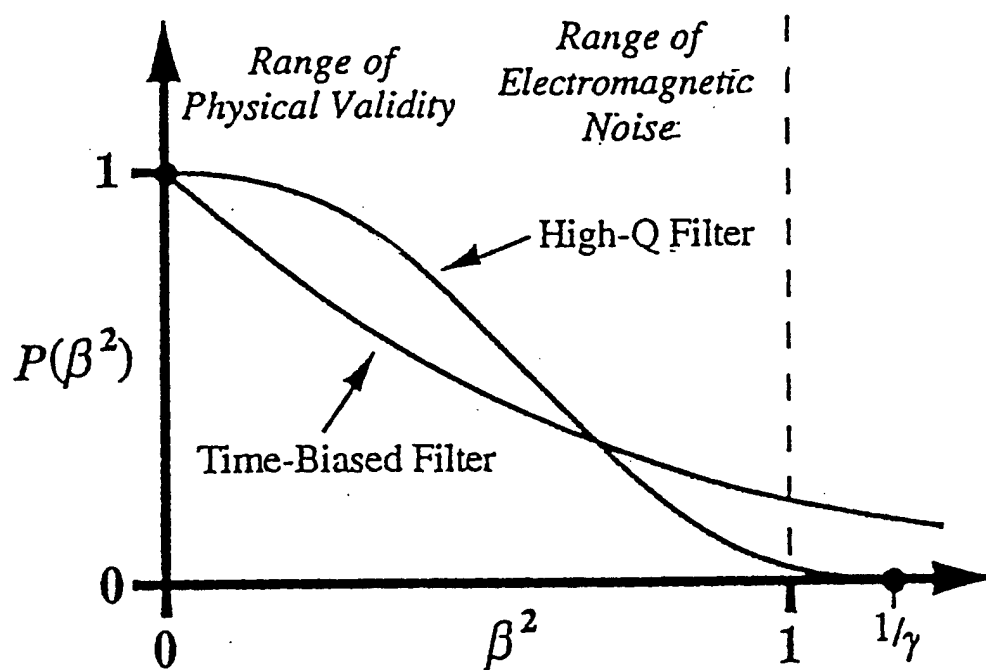


Figure 17-2. Comparison of time-biased and high-Q noise filters.

MAXWELL BIASED Command

Function: Specifies time-biased electromagnetic algorithm.

Syntax:

MAXWELL BIASED [alpha1 alpha2 alpha3 [iterations [coefficient, ...]]] ;

Arguments:

- alpha1 - forward-bias fraction ($t+3/2dt$).
- alpha2 - center-bias fraction ($t+1/2dt$).
- alpha3 - rearward-bias fraction ($t-1/2dt$).
- iterations - number of relaxation iterations (and coefficients).
- coefficient - relaxation coefficients (see Table).

Defaults:

Defaults are set for all electromagnetic field algorithm parameters. The default algorithm is **BIASED**. You can change this with one of the MAXWELL commands. The default mode is **BOTH**. You can use the MODE command to select TE or TM, thus suppressing the other mode. You can use the TIME_STEP command to specify the time_step. If you do not specify the time_step, a default value equal to 80% of the centered-difference Courant criterion will be used, irrespective of the MAXWELL algorithm selected.

The BIASED algorithm provides defaults (which can be over-ridden) for all parameters, as described below.

Description:

The MAXWELL BIASED command specifies a time-biased, iterative solution of the fully time-dependent Maxwell's equations. The time-biased algorithm is an implicit scheme designed to damp high-frequency noise arising from relativistic particles, poor particle statistics, or certain numerical instabilities.

The basic idea involves iterating between electric and magnetic field calculations during a single time step. Monotonically decreasing relaxation coefficients are applied to corrections in the electric field solution. Because of the iterations, this algorithm is computationally more expensive than the centered-difference algorithm, even though a larger time step can be used. The bias fractions represent contributions from magnetic field differences at three points in time: $t+3/2dt$, $t+1/2dt$, and $t-1/2dt$. These fractions are subject to the constraints,

MAXWELL BIASED Command

$$\begin{aligned}\alpha_1 + \alpha_2 + \alpha_3 &= 1 \\ \alpha_1 &> \alpha_3 \\ \alpha_2^2 - 4\alpha_1\alpha_3 &\geq 0.\end{aligned}$$

The relaxation coefficients are typically in the range zero to one, with the first coefficient always equal to one. The following table lists various sets of temporal and relaxation coefficients, assuming $\alpha_3 = 0$.

The Courant stability criterion is given by $(\alpha_2^2 - 4\alpha_1\alpha_3)^{1/2} \chi < 1$, where

$$\chi^2 = c^2 \delta t^2 \sum_{i=1}^2 \frac{1}{(\delta x_i)^2}.$$

is the centered-difference Courant ratio squared. Some geometries are slightly more restrictive (SYMMETRY AXIAL, Ch. 12). Since the bias fraction factor is bounded, the allowable time_step is always greater than the centered-difference algorithm, and it can be substantially greater. Common choices using $\alpha_1 = \alpha_2 = 1/2$ (default), give double the centered-difference result. However, because the time-biased algorithm must iterate, it is always slower than centered difference. Also, it may not be possible to take advantage of the larger time_step due to constraints on outer boundaries (PORT, Ch. 12) and particle dynamics (LORENTZ, Ch. 18).

In the limit of sufficient iterations, the time-biased damping function is given by

$$F(\beta^2) = (1 + 4\alpha_1\chi^2\beta^2)^{-1},$$

where β is the frequency, normalized to the highest value that can be supported by the spatial grid. Thus, for reasonable choices of forward-bias fraction and Courant ratio, a high degree of damping of unwanted high frequencies associated with relativistic particles is achieved. However, note that some non-physical damping occurs at all non-zero frequencies. In other words, the algorithm itself has a Q, which is frequency-dependent and must be accounted for in simulations involving saturation or energy balance.

As indicated by the syntax, you may elect to let the code determine all of the optional values. In this case, the default values are $\alpha_1 = 1/2$, $\alpha_2 = 1/2$, $\alpha_3 = 0$, iterations = 3, and coefficients = 1.0, 0.2, 0.11111. If you want to supply only values for α_1 , α_2 , and α_3 , the code will automatically select the minimum allowed value for iterations, and it will generate the coefficients. You may elect to specify iterations as well, in which case the code will generate for the coefficients. Finally, you may enter all the optional values.

MAXWELL BIASED Command**References:**

B. B. Godfrey and B. Goplen, "Practical Evaluation of Time-Biased Electromagnetic Field Algorithms for Plasma Simulations," Mission Research Corporation Report, AMRC-N-146, presented at the Twenty-Second Annual Meeting APS Division of Plasma Physics, 10-14 November 1980.

Restrictions:

1. This algorithm has an upper limit to the time step used.
2. It is most appropriately used in relativistic particle applications to suppress high levels of numerical noise.
3. Be aware that some damping occurs (the algorithm Q is finite) at all frequencies. This damping is completely unrelated to physical loss mechanisms such as resistivity or outgoing waves. In effect, the reciprocals of all Qs (including the algorithm Q) add to produce the effective Q.

See Also:

SYMMETRY AXIAL, Ch. 12

MODE, Ch. 17

TIME_STEP, Ch. 17

CONTINUITY, Ch. 18

LORENTZ, Ch. 18

Examples:

An algorithm designed for relativistic particles might use time-biased with an aggressive increase in the default time step while suppressing the TE mode.

```
MAXWELL BIASED ;      ! switch from BIASED to BIASED!  
dt = SYS$DTIME * 2.0 ; ! default is 80% of centered max  
TIME_STEP dt ;       ! dt is now 160% of centered max  
MODE TM ;            ! suppress the TE mode
```

Note that a time step which exceeds the centered-difference maximum may lead to trouble with the particle algorithms (LORENTZ, Ch. 18) and outer boundaries (PORT, Ch. 12).

MAXWELL BIASED Command

Coefficients for the time-biased algorithm.

α_1	I	τ 's
0.125	2	1.0, 0.60494
	3	1.0, 0.75385, 0.60494
	4	1.0, 0.83944, 0.68410, 0.60494
0.25	2	1.0, 0.36000
	3	1.0, 0.52941, 0.36000
	4	1.0, 0.65759, 0.44305, 0.36000
0.50	3	1.0, 0.20000, 0.11111
	4	1.0, 0.29912, 0.15022, 0.11111
	5	1.0, 0.39559, 0.20000, 0.13383, 0.11111
	6	1.0, 0.48267, 0.25457, 0.16470, 0.12613, 0.11111
0.75	6	1.0, 0.13458, 0.05385, 0.03182, 0.02349, 0.02041
	7	1.0, 0.17381, 0.06984, 0.04000, 0.02803, 0.02260, 0.02041
	8	1.0, 0.21488, 0.08768, 0.04944, 0.03359, 0.02591, 0.02205, 0.02041
	9	1.0, 0.25676, 0.10712, 0.60001, 0.04000, 0.03000, 0.02459, 0.02169, 0.02041
	10	1.0, 0.29857, 0.12791, 0.07159, 0.04717, 0.03472, 0.02775, 0.02371, 0.02144, 0.02041
	11	1.0, 0.33964, 0.14980, 0.08410, 0.05504, 0.04000, 0.03142, 0.02624, 0.02308, 0.02125, 0.02041
	12	1.0, 0.37943, 0.17256, 0.09743, 0.06355, 0.04579, 0.03551, 0.02919, 0.02517, 0.02262, 0.02111, 0.02041
	16	1.0, 0.52021, 0.26799, 0.15728, 0.10308, 0.07336, 0.05556, 0.04418, 0.03654, 0.03125, 0.02750, 0.02481, 0.02291, 0.02161, 0.02080, 0.02041

18. CHARGED PARTICLES

This Chapter covers the following commands:

SPECIES
LORENTZ
CONTINUITY CONSERVED
CONTINUITY NOT_CONSERVED
CONTINUITY CORRECTED
CONTINUITY LOW_T

These commands specify the charged-particle algorithms. You can use the **SPECIES** command to create new particle species, if required (electrons and protons are already available). You can use the **LORENTZ** command to specify how often the particle coordinates (X1, X2, X3) and momenta (P1, P2, P3) are calculated and whether the calculation is relativistic and 2-D or 3-D. You can use the **CONTINUITY** commands to select the charge-continuity algorithm which calculates the charge-density field (QO) and the current-density fields (J1, J2, J3) from the particle motion. The charge-density field error (QERR) may also be calculated.

You may not need to use any of these commands. The default charged-particle algorithms use the following parameters:

species available - electrons and protons
Lorentz algorithm - 3-D relativistic using the electromagnetic time_step
continuity algorithm - CONSERVED (satisfies Gauss's law exactly)

This combination provides a very conservative, robust solution for simulations involving both nonrelativistic and highly relativistic particles. It is consistent with the electromagnetic algorithm defaults (Ch. 17). If you are unsure of your simulation requirements, this is the recommended configuration.

On the other hand, you may elect to use the other particle algorithms to speed up the simulation, to improve fidelity by a better match of the algorithm to the physics, or even to suppress certain physical effects to gain insight. The following table provides guidance in selecting the continuity algorithm.

General conditions for continuity algorithm selection.

CONTINUITY algorithm	CONSERVED	NOT_ CONSERVED	CORRECTED	LOW_T
speed required		X		
low noise required		X		X
moderate noise acceptable			X	
extreme noise acceptable	X			
exact conservation required	X			
approximate conservation acceptable			X	X
poor conservation acceptable		X		
beams and some plasmas	X		X	
low-temperature plasmas		X		X

A more detailed explanation of the principal factors which influence the simulation follows.

1. Particle definition — Our “particle” is an artificial entity which typically represents a large number of physical particles. The charge-to-mass ratio for all particles of a given species is fixed and is usually identical with a physical species (see SPECIES command). But artificial particles can represent any number of physical particles to meet requirements of the emission processes, which include statistical properties such as creation frequency, particle number, etc. (EMISSION, Ch. 16). It is possible that no two particles will be identical in charge (or mass). Once created by an emission process, particles move through space under the influence of Lorentz forces (LORENTZ, Ch. 18) and contribute to charge and current densities (CONTINUITY, Ch. 18), which are used in Maxwell’s equations (MAXWELL, Ch 17).

2. Time step — As with the electromagnetic field algorithms, there are temporal resolution requirements for particles. There is a time-step instability in the Lorentz and continuity algorithms which is analogous to the Courant instability in Maxwell’s equations. To prevent it, particles must not be allowed to transverse more than one cell in a particle time step. By default, the particle time step is equal to the electromagnetic time_step, the default for which is 80% of the centered-difference Courant criterion. If both defaults are used, the particle stability requirement is automatically satisfied. However, if you use a more aggressive electromagnetic time_step and/or a non-unity kinematics multiplier, the stability requirement could be violated for relativistic particles.

Other stability constraints which may be encountered include cyclotron frequency resolution, $\omega_c \delta t < 1$, an orbit resolution problem associated with high applied magnetic fields, and plasma frequency resolution, $\omega_p \delta t < 2$, a problem associated with high-density plasmas which can result in catastrophic instability.

3. Spatial grid — As with the electromagnetic field solution, accuracy in the particle orbits and current-density fields is determined primarily by the spatial resolution, i.e., the cell size (GRID and AUTOGRID, Ch. 11). For example, if field resolution requirements dictate some maximum cell-to-cell longitudinal field variation (say, 25%), then a space-charge-limited boundary layer might be represented well by 30 cells, barely by 10 cells, and not at all by 3. There are also stability constraints which may be encountered such as Debye length resolution, $\delta x \leq \lambda_D$. This mild instability is encountered in low-temperature plasmas and results in “self-heating.”

4. Local charge conservation — All of the continuity algorithms solve the equation

$$\nabla \cdot J + \partial_t \rho = 0.$$

to obtain current- and charge-density fields from the particle motion. However, the algorithms differ in the way that they solve this equation and in the degree to which they satisfy Gauss’s law

$$\nabla \cdot E - \rho / \epsilon = 0$$

locally, i.e., in each spatial cell. Unfortunately, the algorithm which provides exact charge conservation (conserved continuity) also produces the greatest particle noise. Nevertheless, because of the importance of maintaining Gauss’s law, this is the recommended and default algorithm.

5. Noise — We typically represent a huge number of small, physical particles with a small number of huge, artificial particles. The motion of these huge particles in the finite spatial grid creates numerical noise. The magnitude of this noise goes inversely as the square root of the number of particles, so simulation improvement by this means alone can be an expensive proposition. Instead, algorithms are designed to smooth fields. Some work indirectly by damping electromagnetic fields (high_Q and time-biased), while others work directly on the current-density fields (corrected continuity and low_T).

6. Statistics — The issue is how to adequately represent the required phase space with the fewest particles. The emission processes (EMISSION, Ch. 16) provide statistical controls which allow particular regions of phase space to be emphasized with more particles (of lower charge). For highly correlated phase space (e.g., a charged-particle beam), relatively few particles are required. In such cases, particle methods have great advantage. On the other hand, a thermal plasma might require huge numbers of particles to represent even the simplest physics, and in such cases, fluid methods might be advantageous.

7. Dimensions — By default, the Lorentz kinematics are relativistic and three-dimensional. This is consistent with the electromagnetic field defaults (time-biased with both electromagnetic modes). However, simulations which require only non-relativistic kinematics ($v/c < 0.2$) or two dimensions can offer considerable speed advantage. Determining the relativistic requirement is usually straightforward. The dimensionality required can always be found by analyzing field (including any external fields) and particle momentum components and their interplay between Maxwell's equations and the Lorentz equation. Note that even if the kinematics is 3-D, the component J_3 may vanish from symmetry, so that a TE mode does not necessarily result.

8. Materials, etc. — All particle algorithms are compatible with the full range of material properties and outer boundaries. In general, particles may be created and destroyed by bulk property materials (Ch. 14), while they pass through unique structures (Ch. 15) which have infinitesimal thickness. At outer boundaries (Ch. 12), particles may have their coordinates or momenta altered (at symmetries), or they may be destroyed.

SPECIES Command

Function: Creates new particle species.

Syntax:

```
SPECIES species_name  
      CHARGE charge_multiplier  
      MASS  mass_multiplier { ELECTRON, PROTON, AMU } ;
```

Arguments:

species_name	-	name of species, user-defined.
charge_multiplier	-	electronic charge unit multiplier (signed and unitless).
mass_multiplier	-	mass unit multiplier (unitless).

Defaults:

The ELECTRON and PROTON species have already been created and are ready for use. However, any other species which is required must be created explicitly.

Description:

The SPECIES command is used to create a new particle species.

An internal species table contains the characteristics of all particle species to be used in the simulation. The ELECTRON and PROTON species reside permanently in the species table. You can add other species, as required, to the table using SPECIES commands. The particle species may be based upon physical particles (e.g., ELECTRON and PROTON), but there is no requirement that they be. You are free to create arbitrary particle species to satisfy simulation requirements, which can include non-physical particles such as heavier electrons or lighter protons. You can also create species with identical physical properties but different names to distinguish them in the output. Once created in the table, the new species can be used in a variety of emission (EMISSION, Ch. 16) and initialization (POPULATE, Ch. 19) processes.

A particle species is completely specified by its charge-to-mass ratio. However, for ease of identification with common physical particles, the SPECIES syntax has several arguments. You must give each new species a unique species_name, which will be referred to in other commands. The electronic charge unit has a magnitude of e and positive sign. The CHARGE keyword is followed by the charge_multiplier, which is simply the number of units of electronic charge and carries the sign (+ or -). For convenience, several options are available to specify mass. The MASS keyword is followed by the mass_multiplier and the mass units, which may be ELECTRON, PROTON, or AMU. The AMU, or atomic mass unit (defined so that the mass of the carbon atom equals exactly 12 amu) is more suitable for larger atoms and isotopes.

SPECIES Command**Restrictions:**

The particle species table can contain up to fifteen (15) charged particle species.

See Also:

EMISSION processes , Ch. 16

POPULATE, Ch. 19

LORENTZ, Ch. 18

Examples:

1. A carbon isotope species can be quickly created with the command,

```
SPECIES carbon CHARGE +1 MASS 12 AMU ;
```

2. An electron species named "blue_electron" (to distinguish it from ELECTRON) can be created with the command,

```
SPECIES blue_electron CHARGE -1 MASS 1 ELECTRON ;
```

The "blue_electron" particles are identical with and behave in exactly the same manner as the "ELECTRON" particles. They simply have a different species_name which allows them to be readily distinguished in phase-space plots, etc.

3. An artificial, light-weight species named "proton_lite" can be created with the command,

```
SPECIES proton_lite CHARGE +1 MASS 0.1 PROTON ;
```

At one-tenth the mass of physical protons, "proton_lite" particles will be much more responsive to electromagnetic forces. This might enable simulations precluded by a conventional approach, such as proton backflow in an electron diode.

LORENTZ Command

Function: Specifies Lorentz kinematics algorithm.

Syntax:

```
LORENTZ
[ SPECIES { ALL, ELECTRON, PROTON, species_name } ]
[ TIMING step_multiple ]
[ { RELATIVISTIC, NON_RELATIVISTIC } ]
[ { TWO_D, THREE_D } ] ;
```

Arguments:

species_name - particle species, defined in SPECIES command.
step_multiple - multiple of electromagnetic time_step (integer).

Defaults:

The defaults are ALL species, step_multiple = 1, RELATIVISTIC, and THREE_D.

Description:

The LORENTZ command is used to specify the particle kinematics algorithm. The options may be selected independently for each species of particles, except for step_multiple, which must be the same for all species.

This algorithm adds fields from the Maxwell solution to any magnetostatic fields (PRESET, Ch. 19), and uses these to solve the Lorentz force equation to advance particle coordinates and momenta in time. The particle time step is allowed to be an integer step_multiple of the electromagnetic time_step (TIME_STEP, Ch. 17) to allow efficient simulation of low-velocity particles. However, the Lorentz algorithm has a stability criterion on the time step, and you must ensure that no particle is able to move more than one cell in a particle time step. If the default electromagnetic time_step and default step_multiple are used, stability is guaranteed; however, you may be able to improve efficiency with a more aggressive time_step and/or step_multiple.

You may also be able to improve efficiency by restricting the Lorentz solution to non-relativistic and/or two-dimension kinematics. Determining the relativistic requirement is usually straightforward, and you can safely use non-relativistic (and a larger step_multiple) for $v/c < 0.2$. The dimensionality is more complex, but it can always be determined by analyzing field and momentum components and the interplay between Maxwell's equations and the Lorentz equation. The nature of the emission process (e.g., photoemission) or the presence of external magnetic fields or coupled electromagnetic modes may mean that 3-D kinematics is required. This is also directly related to possible restrictions in the electromagnetic mode (MODE, Ch. 17). However, even if the kinematics is 3-D, the component J3 may vanish from symmetry, so that a TE mode is not necessarily produced.

LORENTZ Command

Some physical regimes not only preclude a non-unity `step_multiple` but actually require a smaller electromagnetic `time_step`. For example, resolution of circular orbits in a high magnetic field (See Examples) or of the plasma frequency at high density may call for a smaller particle time step than that given by the defaults. In such cases, the only recourse is to cut the electromagnetic `time_step` to accommodate the particle requirement.

The Lorentz algorithm calculates the particle coordinates (X1, X2, X3) and momenta (P1, P2, P3), which may be observed in various output commands (e.g., PHASESPACE, Ch. 24). The units of momentum are m/sec, which includes the relativistic factor, γ , but not the rest mass of the particle.

Restrictions:

1. The `step_multiple` (and electromagnetic `time_step`) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.
2. The `step_multiple` must be the same in all LORENTZ commands.
3. Any timers which involve particle output must take the `step_multiple` into account. For example, PHASESPACE plots will be blank unless they are produced at the actual time of the particle kinematics.

See Also:

MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17
PRESET, Ch. 19
PHASESPACE, Ch. 24

References:

B. Goplen, R. Worl, and J. Brandenburg, "Particle Subcycling in Pulsed Power Simulations," Mission Research Corporation Report, MRC/WDC-R-125, October 1987.

Examples:

1. To improve efficiency in simulating a low-velocity ($v/c < 0.2$) electron beam in a cavity, we consider the initial conditions: purely axial emission with no azimuthal momentum (e.g., thermal content), no external magnetic fields, no intrinsic mode coupling, and no outer boundaries. We conclude that we can restrict the Maxwell solution to the TM mode and the Lorentz solution to 2-D, non-relativistic kinematics. Further, we can double the electromagnetic `time_step` and double the `step_multiple` (this increases the particle time step by a factor of four). This is aggressive use of

LORENTZ Command

the algorithms, but should reduce simulation costs by a factor of at least four without sacrificing fidelity. The commands are

```
MODE  TM  ;  
dt   =  2.0 * SYS$DTIME  ;  
TIME_STEP  dt  ;  
LORENTZ  TIMING  2  NON_RELATIVISTIC  TWO_D  ;
```

2. Figure 18-1 illustrates the deleterious effect that an excessive time step can have on particle kinematics, specifically, electron orbits in a high magnetic field. The six gyro-orbits shown have the same magnetic field and electron velocity, but are simulated with succeeding greater time steps. Case (a) exhibits excellent resolution with 100 time steps, and case (b) might be acceptable. However, case (c), with four steps, no longer resembles a circular orbit. Thereafter, the orbit becomes linear (precessing) and eventually, in the large time-step limit, simply goes back and forth as shown in case (f).

LORENTZ Command

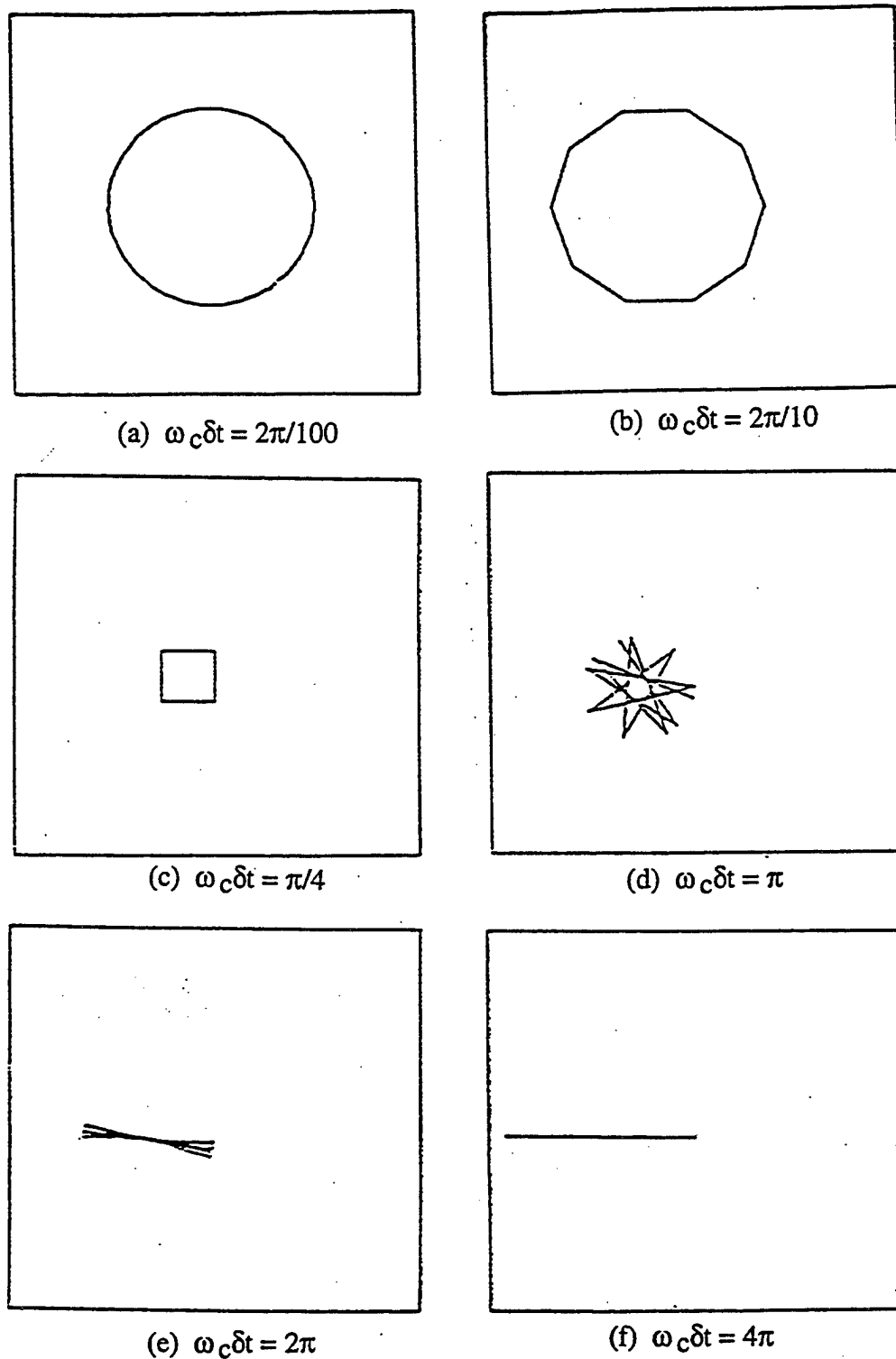


Figure 18-1. Effect of increasing Lorentz time step on gyro-orbits.

CONTINUITY CONSERVED Command

Function: Specifies conserved continuity algorithm.

Syntax:
CONTINUITY CONSERVED ;

Arguments:

None.

Defaults:

Defaults are set for all particle algorithm parameters. The default continuity algorithm is CONSERVED. You can change this with one of the CONTINUITY commands. The default Lorentz algorithm is three-dimensional and relativistic, and it uses the electromagnetic time_step (a step_multiplier of one). Electrons and protons are available as the default species.

Description:

The CONTINUITY CONSERVED command specifies a continuity algorithm which conserves charge (Gauss's law) exactly, but is high in numerical noise. It is suitable for most particle applications. An electromagnetic algorithm which provides damping, such as high_Q or time-biased, can be used to reduce noise if required (MAXWELL, Ch. 17).

This algorithm produces current-density fields which conserve Gauss's law exactly (within roundoff error). The algorithm can be derived from application of Gauss's law to conformal, rectilinear motion, with the particle orbit being a sum of such rectilinear motions and the order of the rectilinear motions being determined randomly. It requires no correction, since it achieves charge conservation directly. However, in PIC parlance, the algorithm uses "nearest-grid-point" charge allocation, and this is what produces the extreme numerical noise.

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). Since no error in Gauss's law is produced, the charge-density error field (QERR) is not calculated for this algorithm.

Restrictions:

1. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.

CONTINUITY CONSERVED Command

2. Only one CONTINUITY command can be given. (All particle species must use the same algorithm.) If more than one command is given, the last command will dominate.

3. If the CONSERVED algorithm is used with relativistic particles, then an electromagnetic field algorithm which includes damping, such as high-Q or time-biased, is recommended to reduce noise.

See Also:

LORENTZ, Ch. 18
MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17

CONTINUITY NOT_CONSERVED Command

Function: Specifies not-conserved continuity algorithm.

Syntax:
CONTINUITY NOT_CONSERVED ;

Arguments:

None.

Defaults:

Defaults are set for all particle algorithm parameters. The default continuity algorithm is CONSERVED. You can change this with one of the CONTINUITY commands. The default Lorentz algorithm is three-dimensional and relativistic, and it uses the electromagnetic time_step (a step_multiplier of one). Electrons and protons are available as the default species.

Description:

The CONTINUITY NOT_CONSERVED command specifies a continuity algorithm which is both fast and low in numerical noise. However, it does not conserve charge (Gauss's law) locally. It is suitable for brief, non-repetitive, transient phenomena, in which the errors (produced by the particles) in the electric fields are small in comparison with their magnitude.

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). The error in Gauss's law,

$$\rho_{err} = \rho - \epsilon \nabla \cdot E$$

is a scalar field named QERR which is accessible as output in the same manner as the other fields.

Restrictions:

1. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.
2. Only one CONTINUITY command can be given. (All particle species must use the same algorithm.) If more than one command is given, the last command will dominate.

CONTINUITY NOT_CONSERVED Command

3. The NOT_CONSERVED algorithm should be used only where violations of Gauss's law will not adversely affect the results.

See Also:

LORENTZ, Ch. 18

MAXWELL algorithms, Ch. 17

TIME_STEP, Ch. 17

MODE, Ch. 17

CONTINUITY CORRECTED Command

Function: Specifies corrected continuity algorithm.

Syntax:
CONTINUITY CORRECTED diffusion_coefficient ;

Arguments:
diffusion_coefficient - error diffusion coefficient (unitless).

Defaults:

Defaults are set for all particle algorithm parameters. The default continuity algorithm is CONSERVED. You can change this with one of the CONTINUITY commands. The default Lorentz algorithm is three-dimensional and relativistic, and it uses the electromagnetic time_step (a step_multiplier of one). Electrons and protons are available as the default species.

Description:

The CONTINUITY CORRECTED command specifies a continuity algorithm which is low in numerical noise and conserves charge (Gauss's law) approximately. It uses the same algorithm as NOT_CONSERVED, but then it applies a correction which allows it to conserve charge (Gauss's law) approximately. The correction restores charge conservation at long time scales, as controlled by the diffusion_coefficient. A larger coefficient restores conservation more quickly, but results in greater noise.

In this algorithm, the error in Gauss's law,

$$\rho_{err} = \rho - \epsilon \nabla \cdot E$$

is computed immediately following the electromagnetic field calculation. Then the subsequent current density is corrected by the addition of an error diffusion term,

$$\bar{J}' = \bar{J} + d \nabla \bar{\rho}_{err} ,$$

where $d = 1/4 c_d (\delta x)^2 / \delta t$ and c_d is the diffusion_coefficient, δx^2 is the minimum square grid spacing, and δt is the time step. (Use of this algorithm with a diffusion_coefficient of zero will produce the same result as the not_conserved algorithm, but at greater expense.) No benefit will accrue from using a diffusion_coefficient greater than 2. Useful values will lie between 0.1 and 1.

CONTINUITY CORRECTED Command

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). The error in Gauss's law is a scalar field named QERR which is accessible as output in the same manner as the other fields.

References:

B. Goplen, R. Worl, and L. Ludeking, "A New Current Density Algorithm in MAGIC," Mission Research Corporation Report, MRC/WDC-R-155, December 1987.

Restrictions:

1. The step_multiple (and electromagnetic time_step) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.
2. Only one CONTINUITY command can be given. (All particle species must use the same algorithm.) If more than one command is given, the last command will dominate.
3. The CORRECTED algorithm should be used only where local charge conservation is required but electromagnetic field damping is inadvisable.

See Also:

CONTINUITY NOT_CONSERVED, Ch. 18
LORENTZ, Ch. 18
MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17

Examples:

The following three commands are used in three separate simulations to illustrate the effect of non-conservation of charge. The model is a simple diode maintained at fixed voltage with an applied transverse magnetic field. Physically, this might represent boundary layer formation in a magnetic insulation simulation.

```
CONTINUITY NOT_CONSERVED ;  
CONTINUITY CORRECTED 2.0E3 ;  
CONTINUITY CONSERVED ;
```

Electron trajectory plots from these three simulations are shown together in Figure 18-2.

CONTINUITY CORRECTED Command

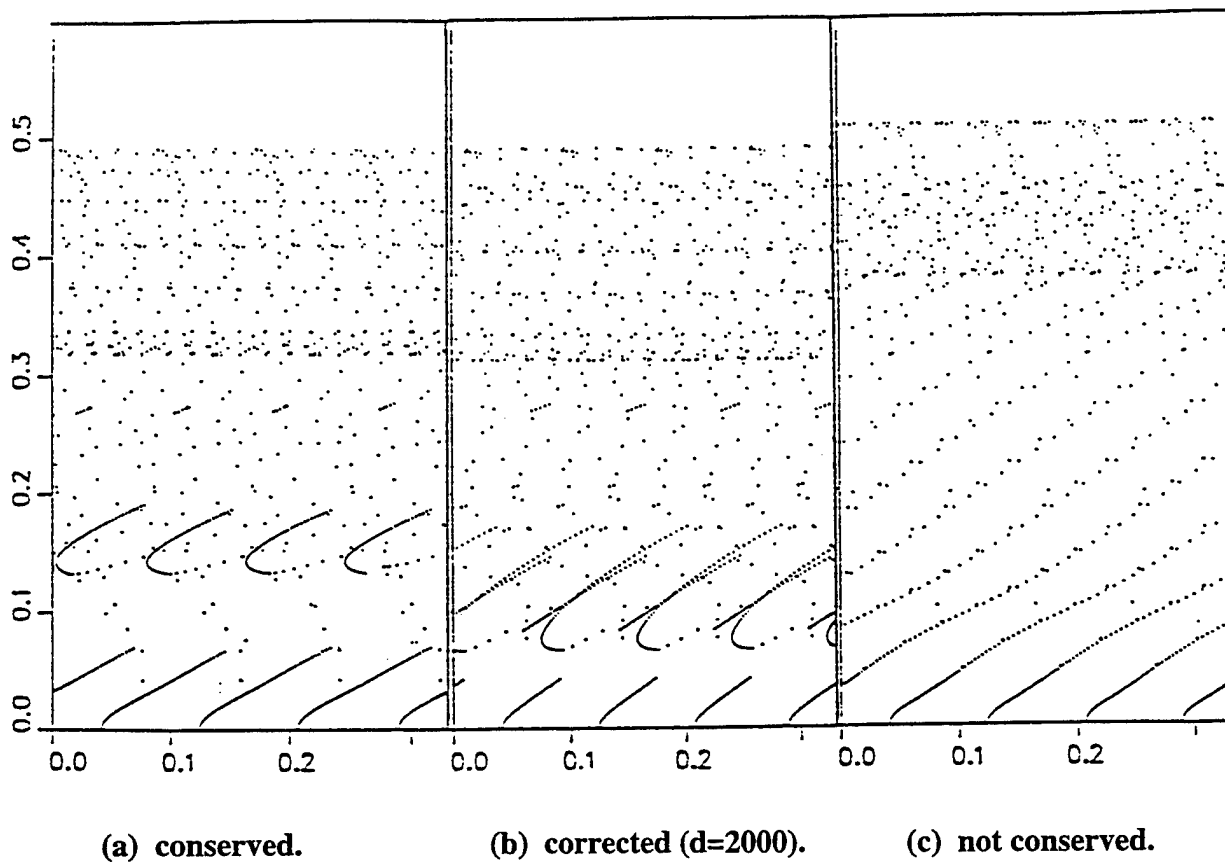


Figure 18-2. Charge conservation effects in magnetic insulation.

CONTINUITY LOW_T Command

Function: Specifies low-T continuity algorithm.

Syntax:
CONTINUITY LOW_T [debye_ratio] ;

Arguments:
debye_ratio - Debye length-to-grid spacing ratio (unitless).

Defaults:

Defaults are set for all particle algorithm parameters. The default continuity algorithm is CONSERVED. You can change this with one of the CONTINUITY commands. The default Lorentz algorithm is three-dimensional and relativistic, and uses the electromagnetic time_step (a step_multiplier of one). Electrons and protons are available as the default species.

Description:

The CONTINUITY LOW_T command specifies a continuity algorithm which conserves charge (Gauss's law) approximately and also reduces numerical noise. It is suitable for low-temperature plasma applications, where self-heating occurs due to failure of the spatial grid to adequately resolve the Debye length.

This algorithm makes use of the CONSERVED algorithm, which conserves charge (Gauss's law) exactly. To reduce the numerical noise associated with this algorithm, it then performs a temporal filter of the current-density fields. In the process, it loses the exact charge conservation feature. The algorithm has shown some ability to suppress the grid instability associated with particle self-heating when the debye length (=thermal speed/plasma frequency) of a group of particles is smaller than the grid spacing. This generally occurs when debye_ratio is near the ratio of debye length to the grid spacing.

This algorithm calculates the charge-density field (QO) and the current-density fields (J1, J2, J3). The charge-density error field (QERR) is not calculated.

CONTINUITY LOW_T Command**Restrictions:**

1. The `step_multiple` (and electromagnetic `time_step`) must be chosen to preclude a particle from moving more than one cell width in a single particle time step.
2. Only one `CONTINUITY` command can be given. (All particle species must use the same algorithm.) If more than one command is given, the last command will dominate.
3. The `LOW_T` algorithm should be used only in low-temperature plasma applications.

See Also:

CONTINUITY CONSERVED, Ch. 18
LORENTZ, Ch 18
MAXWELL algorithms, Ch. 17
TIME_STEP, Ch. 17
MODE, Ch. 17

This page is intentionally left blank.

19. OTHER ALGORITHMS

This Chapter covers the following commands:

POPULATE
PRESET
DRIVER
CIRCUIT
POISSON
EIGENMODE

The group of commands involves initial conditions, external sources, and electrostatic and eigenvalue fields. You can use the POPULATE command to create an initial charged-particle distribution and the PRESET command to initialize the electromagnetic and magnetostatic fields. You can use the DRIVER command to apply external current-density sources and the CIRCUIT command to apply external voltage sources between conductors. You can use the POISSON command to compute electrostatic fields and the EIGENMODE command to compute frequency eigenvalues and cyclic field distributions.

The following table provides guidance on the physical conditions which bear on the algorithm selection.

General conditions for algorithm selection.

CONDITION	POPULATE	PRESET	DRIVER	CIRCUIT	POISSON	EIGENMODE
External B		X				
External J			X			
External V				X	X	
Initial E, B		X				
Initial $\rho = 0$	X					
Initial $\rho \neq 0$	X	X			X	
Electrostatic $\rho = 0$					X	
Electrostatic $\rho \neq 0$	X				X	
Eigenvalues						X
Degenerate Modes		X				X

A brief discussion of the main considerations follows.

1. Initial charged-particle distributions — The POPULATE command can be used to create an initial charged-particle distribution within a specified spatial area. If the resulting charge distribution is non-neutral, then electromagnetic field initialization is required. Recall that the

default initial field vanishes, which implies that there is no net charge anywhere in space. Thus, if you create a charged particle but do not initialize the fields to account for it, the code implicitly creates a second particle of opposite charge to cancel the one created. In effect, this second charge has infinite mass and remains embedded in the spatial grid for the duration of the simulation.

2. Initial field distributions — The PRESET command can be used to initialize electromagnetic fields (E1, E2, E3, B1, B2, B3) as well as electrostatic fields (E1ST, E2ST, PHST) and magnetostatic fields (B1ST, B2ST, B3ST). As the simulation transient progresses, the electromagnetic fields will normally change from their initial values. The electrostatic and magnetostatic fields are presumed due to some external source, such as a permanent magnet, and these fields never change. In interpreting output, note that the dynamic electric fields (E1, E2) may be initialized to electrostatic results (E1ST, E2ST), but that the dynamic magnetic fields (B1, B2, B3) are never initialized to magnetostatic fields (B1ST, B2ST, B3ST). Instead, the magnetostatic and dynamic magnetic fields are added together when particle forces are required for the Lorentz equation. This provides flexibility in the specification of magnetostatic fields. However, care should be taken that initialized electric fields are self-consistent, since charge may be present in the initial state of the system.

3. External current-density sources — The DRIVER command can be used to specify external current-density sources which may be arbitrary functions of time and space. These sources will be applied directly to the electric field calculation and are not included in the current-density fields (J1, J2, J3), which are due solely to charged particles. Therefore, the DRIVER sources may not be observed, except through their effect on electromagnetic fields.

4. External voltage sources — The CIRCUIT command can be used to apply an external voltage source between conductors which are otherwise unconnected. This may be used to maintain a specified voltage between conductors, and it is also commonly used to represent the effect of a transverse electromagnetic (TEM) wave propagating in the direction of the ignorable coordinate (x3). (Simulation of a magnetron presents a practical example.) A TEM waveform is always given by solution of the Laplacian, which we can obtain using a POISSON command. Note that the number of solutions required typically equals the number of unconnected conductors, minus one.

5. Electrostatic field solutions — There are at least two circumstances in which it is necessary to use the POISSON command to compute the static potential (PHST) and electrostatic fields (E1ST, E2ST). One involves computing the initial electric field distribution for a non-neutral initial charge distribution. (The charges are created with a POPULATE command, the electrostatic fields are computed with the POISSON command, and the dynamic fields (E1 and E2) are initialized using a PRESET command.) The second circumstance involves determining the TEM waveform associated with propagation in the ignorable coordinate. In this case, there are no particles, but the various conductors will have specified potentials. The POISSON command is used simply to obtain a Laplacian solution. The voltages associated with these TEM fields are applied using the CIRCUIT command.

6. Eigenvalues and eigenfunctions — The EIGENMODE command can be used to obtain frequency eigenvalues and field eigenfunctions from a steady-state solution of Maxwell's time-

dependent equations. No particle distributions may be used. The PRESET command can be used to initialize eigenfunctions in searching for degenerate modes.

This page is intentionally left blank.

POPULATE Command

Function: Creates an initial charged-particle distribution.

Syntax:

```
POPULATE species area_name n_x1_sites n_x2_sites
      { FUNCTION CHARGE q(x1,x2)
        p1(x1,x2) p2(x1,x2) p3(x1,x2) ,
      FUNCTION DENSITY rho(x1,x2)
        p1(x1,x2) p2(x1,x2) p3(x1,x2) ,
      READ CHARGE file_name record_tag q_record
        p1_record p2_record p3_record ,
      READ DENSITY file_name record_tag rho_record
        p1_record p2_record p3_record }
```

Arguments:

species	- name of particle species, defined in SPECIES command.
area_name	- name of spatial area, defined in AREA command.
n_x1_sites	- number of uniformly-spaced sites in x1 (integer) .
n_x2_sites	- number of uniformly-spaced sites in x2 (integer).
q	- particle charge in Coulombs, spatial function or constant.
rho	- charge density in Coul/sq. m, spatial function or constant.
p1	- x1-momentum in m/sec, spatial function or constant.
p2	- x2-momentum in m/sec, spatial function or constant.
p3	- x3-momentum in m/sec, spatial function or constant.
file_name	- name of file containing particle data.
record_tag	- value (usually time) used to select record.
q_record	- name of record containing charge values.
rho_record	- name of record containing charge_density values.
p1_record	- name of record containing x1_momentum values.
p2_record	- name of record containing x2_momentum values.
p3_record	- name of record containing x3_momentum values.

Description:

The POPULATE command creates an initial particle distribution. A particle is completely specified by its species, its charge, its coordinates (X1, X2, X3) and its momenta (P1, P2, P3). Note that the definition of momentum includes the relativistic factor, γ , but not the particle rest mass. The units are meters (or radians) for the coordinates and m/sec for momenta.

You must first specify the species and the area_name in which particles will be created. There will be n_x1_sites_x1 for particle creation, uniformly distributed in x1 within area_name. Similarly, there will be n_x2_sites uniformly distributed in the other coordinate.

POPULATE Command

Thus, the total number of particles created within `area_name` will be `n_x1_sites` times `n_x2_sites`.

The particle charge and momentum can be specified using one of four options: **FUNCTION CHARGE**, **FUNCTION DENSITY**, **READ CHARGE**, and **READ DENSITY**.

The **FUNCTION** options offer a very effective means of creating particles. The **FUNCTION CHARGE** option allows you to specify spatial functions for particle charge and three momentum components. The **FUNCTION DENSITY** option is exactly the same, except that the first function represents charge density instead of particle charge. Wherever the charge (or charge-density) function vanishes, no particles will be created. Thus, complicated spatial distributions can be created by properly constructing the charge function, making use of intrinsic functions such as **STEP**, **MIN**, **MAX**, **RANDOM**, and **GAUSSIAN**.

The **READ** options read particle data from a file. In the **READ CHARGE** option, you specify the `file_name` and the `record_tag` followed by the names of records containing particle charge and the three momentum components. (The `record_tag` usually specifies the record time; if it is set to -1, then the first record will be used.) The **READ DENSITY** option is exactly the same, except that the first particle record contains density instead of charge data.

If the initial charge distribution is non-neutral, then field initialization using the **POISSON** and **PRESET** commands is required. Recall that the default initial field vanishes, which by Gauss's law implies that there is no net charge anywhere in space. Thus, if you create charged particles but do not initialize the fields to account for them, the code implicitly creates a second group of particles of opposite charge to cancel the ones created. In effect, particles in this second group have infinite mass and remain embedded in the spatial grid for the duration of the simulation.

See Also:

SPECIES, Ch. 18

POISSON, Ch. 19

PRESET, Ch. 19

Restrictions:

The maximum number of **POPULATE** commands is 50.

References:

B. Goplen, "Simulations of Self-Colliding Orbit Stability Against Negative Mass Observed in Migma IV Experiment," APS Division of Plasma Physics, Baltimore, MD, 3-7 November 1986.

POPULATE Command**Examples:**

The following commands create a fifty by fifty array of (2500) “electrons,” each with a charge of 10^{-8} Coulombs and uniformly distributed in an area named “diode.” The particle momenta are outward directed, increasing linearly from the origin.

```
FUNCTION  p1(x1,x2) = 1.e8 * x1 ;
FUNCTION  p2(x1,x2) = 1.e8 * x2 ;
FUNCTION  p3(x1,x2) = 0 ;
POPULATE ELECTRON diode 50 50
          FUNCTION CHARGE 1.E-8 p1 p2 p3 ;
```

PRESET Command

Function: Initializes specified electromagnetic, electrostatic, or magnetostatic field.

Syntax:

```
PRESET field
{ FUNCTION function_name(x1, x2) ,
  POISSON poisson_name ,
  READ file_name data_type file_field file_format }
[ MODIFY { ADD, REPLACE } ]
[ SCALE scale_factor(t) ] ;
```

Arguments:

field	- field component (E1, E2, etc.).
function_name	- spatial distribution, defined in FUNCTION command.
poisson_name	- name of Poisson solution, defined in POISSON command.
file_name	- name of the file containing field data.
data_type	- data type (CONTOUR, LINPRINT, PERSPECTIVE, or SURFACE).
file_field	- field component as recorded in the file.
file_format	- file format (ASCII or BINARY).
scale_factor	- field scaling factor, constant or function defined in FUNCTION command.

Defaults:

The default initialization value for all electromagnetic, electrostatic, and magnetostatic fields is zero. The default values for the initialization options are REPLACE and scale_factor = 1.

Description:

The PRESET command initializes a specified electromagnetic, electrostatic, or magnetostatic field.

The only fields which can be initialized are the electromagnetic fields (E1, E2, E3, B1, B2, B3), the electrostatic fields (E1ST, E2ST, PHST), and the magnetostatic fields (B1ST, B2ST, B3ST). As the simulation transient progresses, the electromagnetic fields will normally change from their initial values. Electrostatic fields never change. They are normally used to initialize (PRESET) electric fields or to modify them during the transient (CIRCUIT). The magnetostatic fields are presumed due to some external source, such as a permanent magnet, and these fields never change. However, the magnetostatic fields are never used to initialize the dynamic magnetic fields. Instead, the magnetostatic and dynamic magnetic fields are added together only when

PRESET Command

particle forces are required for the Lorentz equation. This distinction is important to remember in interpreting the field values.

There are no constraints on the distributions which can be used for the magnetostatic fields, although we strongly advise making them divergence-free. Similarly, the electromagnetic field distributions must be self-consistent, i.e., satisfy Maxwell's equations. In particular, Gauss's law must always be satisfied. This is simple in the absence of charge (the default fields vanish), but requires solution of Poisson's equation (POISSON) to initialize electric fields (PRESET) if a non-neutral charge distribution (POPULATE) is created.

There are three options, or methods, of initializing fields: FUNCTION, POISSON, and READ. The FUNCTION method requires that you define a function of the spatial coordinates. This function will be evaluated precisely at the field locations to set the initial values. This method can be used to set any field.

When particles are present, the POISSON method can be used to initialize the electric fields E1 and E2. (Note that space charge affects only the TM mode.) You must obtain a Poisson solution (the solution and the poisson_name is specified in the POISSON command), which calculates the scalar field PHST and the electrostatic fields E1ST and E2ST. The PRESET command simply transfers the E1ST and E2ST fields to E1 and E2. (E1ST and E2ST never change, while E1 and E2 will evolve from their initial values.)

Finally, the READ method reads field values from file_name, and interpolates these values to the spatial grid. The input file must use the DUMP database format. Then data_type specifies the type of output, with the choices being CONTOUR, LINPRINT, PERSPECTIVE, or SURFACE. The file_field is the field component recorded in the file, and the file_format is either ASCII or BINARY. Directions for preparing an input file for use with the READ option are documented separately (see References).

You can use the MODIFY and SCALE options to alter the fields being entered with PRESET. The REPLACE option can be used to patch together several contiguous spatial segments. For example, if data is available for three spatial areas, these may be read in and patched together. Use of REPLACE will cause the new field values to replace the previous values. If it is desired to create a superposition of fields, ADD should be used in place of REPLACE. With the ADD option, the fields are added to the existing fields established by earlier PRESET commands. The scale_factor can be used to alter the magnitude of the entered field. The scale_factor can be a function of time only when applied to magnetostatic fields.

PRESET Command**Restrictions:**

1. Each field component to be initialized requires a separate PRESET command.
2. In presetting dynamic fields (E1, E2, ...), care must be taken that the preset fields are self-consistent, since charge may be present in the initial state of the system and boundary conditions will be applied.
3. Caution must be used when initializing magnetostatic fields. Certain fields may not make sense geometrically in non-Cartesian coordinate systems, and furthermore, the use of a particular component can break an assumed symmetry chosen for the dynamic field solution. For example, an x1- or x2-component of magnetic field could drive the transverse electric (TE) mode due to particle motion in the x3 direction, and substantial error may result if the TE mode is not represented in the simulation.
4. In interpreting output, recall that dynamic magnetic fields do not include any magnetostatic fields. The latter are included only in the particle force calculation. The dynamic electric fields, by contrast, may have been initialized to electrostatic values or may have electrostatic components being introduced continuously.
5. If an initial particle distribution is used with default initial fields (zero), the algorithms will assume that fixed (immobile) charges are embedded in the grid to exactly neutralize the initial distribution. (The fixed charges remain after the particles begin to move.) The electric fields must be initialized to properly account for space charge.

See Also:

MAXWELL algorithms, Ch. 17
LORENTZ, Ch. 18
POISSON, Ch. 19

References:

L. Ludeking, "Importing Field Data to MAGIC," MRC/WDC-M-037M, January 1990.

Examples:

1. The following commands are used to initialize the B3 fields to values from the function, "bfield."

```
FUNCTION bfield(x,y) = 0.1 * (x*x + y*y) ;  
PRESET B3 FUNCTION bfield ;
```


PRESET Command

2. The following commands are used to initialize the B3ST fields to previously computed values. The data is read from file_name "BCOIL". The data_type is LINPRINT, and the file_field in the file is B3. The data was recorded in ASCII format.

```
PRESET B3ST READ BCOIL LINPRINT B3 ASCII ;
```

3. The following command is used to initialize the dynamic electric fields (E1 and E2) to values obtained from the Poisson solution, "Quick_Start." The Poisson solution is obtained for a charge distribution entered with the POPULATE command and with zero voltage on the "anode" and "cathode."

```
POISSON Quick_Start 2 cathode 0.0 anode 0.0 ;  
PRESET E1 POISSON Quick_Start ;  
PRESET E2 POISSON Quick_Start ;
```

DRIVER Command

Function: Applies an external current-density source to a specified area.

Syntax:

```
DRIVER { J1, J2 J3 } j(t,x1,x2) area_name ;
```

Arguments:

j - name of current-density function, defined in
 FUNCTION command.

area_name - name of application area, defined in AREA command.

Description:

The DRIVER command specifies a prescribed (analytic) current-density source to drive electromagnetic fields in a local spatial area. The current-density component is either J1, J2, or J3. The current density function, in units of A/m², must be specified using the FUNCTION command. The current density will be applied only within the specified area.

Note that current-density sources entered using this command are applied directly to the electric fields. They are not added to the current-density fields (J1, J2, and J3), which contain only currents associated with actual particle motion.

Restrictions:

The current-density sources entered with the DRIVER command are not added to the particle current-density fields, and may not be observed directly.

See Also:

MAXWELL algorithms, Ch. 17

Examples:

To apply the current density source,

$$J_3(x_1, t) = 100x_1 \cos(2\pi \cdot 10^9 t),$$

within an area labeled "source_area," one would use the commands,

```
FUNCTION source(x1,x2,t) = 100 * x1 * COS(6.28E9*t) ;
DRIVER J3 source source_area ;
```

CIRCUIT Command

Function: Specifies an external circuit to apply voltage between conductors.

Syntax:

```
CIRCUIT v(t) poisson_name
      [ MEASURE
        { MATRIX area_name, INTEGRAL lines [ line_name,... ] } ]
      [ MODEL
        { STATIC, RESISTIVE z, INDUCTIVE z l } ] ;
```

Arguments:

v(t)	-	voltage function, defined in FUNCTION command.
poisson_name	-	Poisson solution name, defined in POISSON command.
area_name	-	name of spatial area, defined in AREA command.
lines	-	number of voltage measurements (integer).
line_name	-	name of spatial line, defined in LINE command.
z	-	circuit impedance (ohms).
l	-	circuit inductance (henrys).

Defaults:

The defaults are MEASURE MATRIX, with the full simulation area used for area_name, and MODEL STATIC.

Description:

The CIRCUIT command provides a means of applying a time-varying voltage between a set of unconnected conductors. Physically, this can be used to represent a transverse electromagnetic (TEM) wave traveling in the third (symmetry) spatial dimension. In a magnetron simulation, for example, the dominant field and particle kinematics occur in the (r,θ) plane, but the voltage pulse between anode and cathode propagates in the axial (z) coordinate. This can be approximated using the CIRCUIT command. The electrostatic solution for the TEM wave in the third dimension is computed using the POISSON command, which must be used if the CIRCUIT command is given.

The temporal voltage function, v(t), represents the external voltage source which is applied to the circuit connecting two or more conductors. The TEM solution is identified in both commands by poisson_name, which is defined in the POISSON command.

The keyword MEASURE specifies the method of measuring the voltage between pairs of conductors. The MATRIX option specifies the capacitive matrix method to obtain voltages. The dot product of the dynamic electric field and electrostatic field solutions over area_name is taken, and voltages are computed dynamically on the basis of energy conservation by solving the coupled

CIRCUIT Command

capacitance equations. The INTEGRAL option provides line-integral measurements between two conductors to obtain a voltage. The argument lines specifies the number of voltage measurements, each consisting of a straight-line integral along each line_name. The INTEGRAL method is much faster than the MATRIX method, since it depends only on a few line integrals rather than complete volume integrals.

The keyword MODEL is used to specify one of three circuit models: STATIC, RESISTIVE, and INDUCTIVE. In the STATIC model (zero impedance), the conductor voltage is maintained precisely at the value specified by the external voltage source. In the RESISTIVE model, the source voltage is applied through the finite external impedance specified by z. In the INDUCTIVE model, the source voltage is applied through the finite external impedance and inductance specified by z and l, respectively. The default model is STATIC.

For applications in which the applied voltage across different pairs of conductors is varied independently, it is necessary to specify multiple Laplacian solutions and circuits. For example, in a triode consisting of a cathode, grid, and anode, the voltage on the grid may be varied with an RF signal, whereas the anode voltage might be held fixed. In this case, two Laplacian solutions and two circuits are required.

The circuit algorithm calculates variables which can be recorded by using OBSERVE commands. The following table lists the variable names and the physical symbol and definition associated with each.

Variable	Symbol	Definition
DCHARGE	δQ	differential charge added by circuit
CHARGE	Q	total charge added by circuit
CURRENT	I	current in circuit
DENERGY	δE	differential energy added by circuit
ENERGY	E	total energy added by circuit
POWER	P	circuit power added
VOLTAGE	V	applied circuit voltage
DVOLTAGE	δV	differential voltage added to system

Restrictions:

1. The maximum number of CIRCUIT commands is five.
2. All circuits must use the same measurement method.

CIRCUIT Command

See Also:

POISSON, Ch. 19
OBSERVE, Ch. 22

References:

L. Ludeking, B. Goplen, W. M. Bollen, "Gated Emission Simulations," Mission Research Corporation Report, MRC/WDC-R-119, August 1986.

L. Ludeking, B. Goplen, and N. Vanderplaats, "Simulation of Gated Emission Triodes," Bulletin of the American Physical Society, Vol. 31, p. 1600, November 1986.

CIRCUIT Command

Examples:

1. In this example, the circuit algorithm uses the function "Volts" for the external voltage source and applies the Laplacian solution with the name Laplace. The STATIC circuit model is used by default. Also by default, the voltage measurements are made using the MATRIX method with the area being the complete simulation area.

```
Vmax = 100.E3 ;
Trise = 10.E-9 ;
FUNCTION Volts(t) = Vmax * (1.0 - EXP(-t/Trise));
POISSON Laplace 2 Cathode 0.0 Anode 1.0 ;
CIRCUIT Volts LaPlace ;
```

2. This example shows commands used in a planar model of the Varian/EIMAC X2259A electron gun. The model consists of a planar cathode, a grid, and a planar anode. Since the voltage on the anode and grid are applied separately, two circuits are required. The first circuit uses the function VRF for a voltage source with the Laplacian solution POI-ONE. By default, the STATIC circuit model is used. The voltage measurements are made using the MATRIX method. The second circuit uses the function VCONST for a voltage source with the Laplacian solution POI-TWO. The STATIC circuit model is used, and the voltage measurements are made with the MATRIX method.

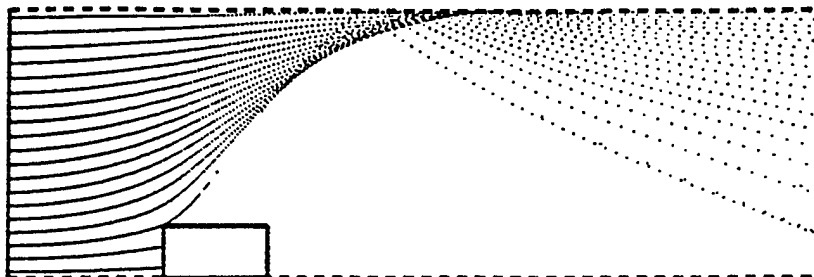
```
C Laplacian solution with grid at 1 volt. ;
POISSON POI-ONE 3 CATHODE 0.0 ANODE 0.0
  GRID 1.0 INITIALIZE FGRID
  TEST 1000 10 1.E-5
  ALGORITHM ADI 1 1 ;
C Laplacian solution with anode at 1 volt. ;
POISSON POI-TWO 3 CATHODE 0.0 ANODE 1.0
  GRID 0.0 INITIALIZE FANODE
  TEST 1000 10 1.E-5
  ALGORITHM ADI 1 1 ;
C Define circuit one, applied to grid. ;
FREQGRID = 420.58E6 ;
VBIAS = - 280.0 ;
VSWING = 310.0 ;
WFREQ = PI * 2. * FREQGRID ;
FUNCTION VRF(T) = VBIAS + VSWING * COS(WFREQ*T) ;
CIRCUIT VRF POI-ONE ;
C Define circuit two, applied to anode. ;
VANODE = 8.5E4 ;
FUNCTION VCONST(T) = VANODE ;
CIRCUIT VCONST POI-TWO ;
```

Figure 19-1 shows electron trajectories and the longitudinal and transverse phase space of the electrons near the cathode and grid wire.

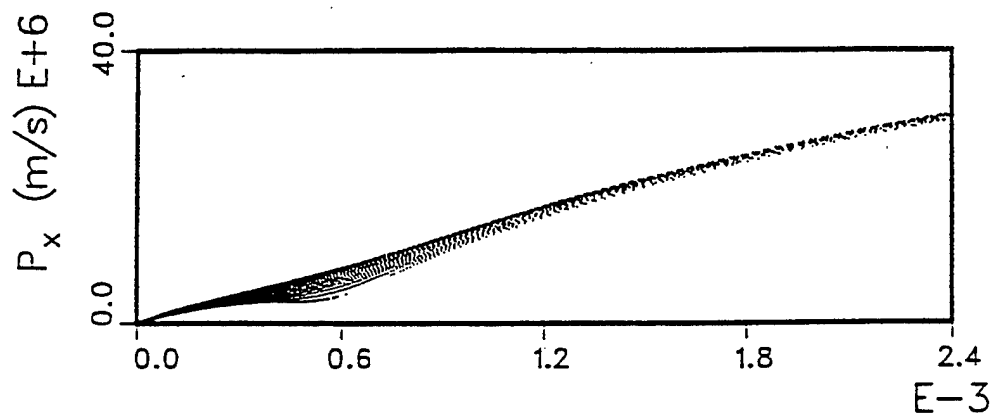
Phase Space at Time 3.75E-09 sec

CIRCUIT Command

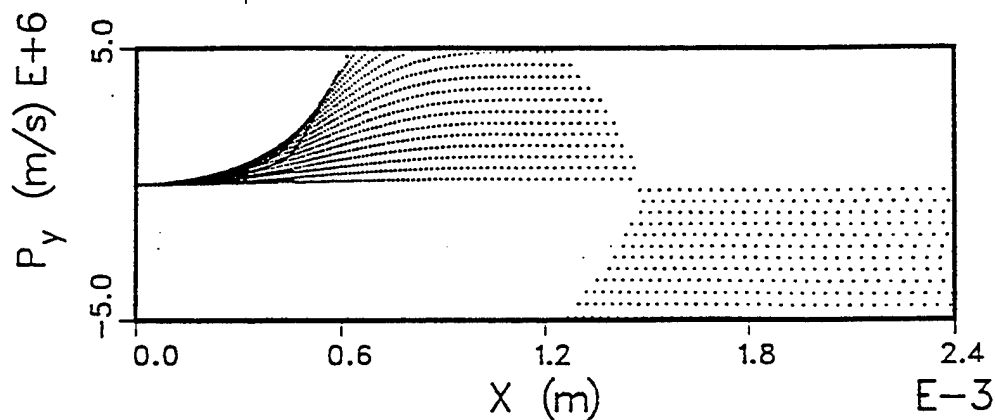
$V_a = 85 \text{ kV}$
 $\text{Grid RF} = 420.58 \text{ MHz}$
 $\text{Grid bias} = -280 \text{ V}$
 $\text{Grid swing} = 310 \text{ V}$
 $D_{ca} = 50 \text{ mm}, D_{cg} = .6096 \text{ mm}$



(a) Particle trajectories.



(b) Longitudinal phase space.



(c) Transverse phase space.

Figure 19-1. Varian/EIMAC X2259A electron gun.

POISSON Command

Function: Specifies an electrostatic field solution using a Poisson algorithm.

Syntax:

```
POISSON poisson_name conductors area_name,voltage ...
      [ ALGORITHM { SOR omega, SCA radius, ADI i1flag i2flag } ]
      [ TEST total_iterations test_iterations test_error ]
      [ INITIALIZE function(x1, x2) ]
      [ POPULATE scale_factor ] ;
```

Arguments:

poisson_name	- Poisson solution name.
conductors	- number of unique conductors.
area_name	- name of object, defined in AREA command.
voltage	- object potential (V).
omega	- over-relaxation coefficient (unitless).
radius	- spectral radius of the Jacobi matrix (unitless).
i1flag	- x1-coordinate ADI iteration flag (= 0, 1, or 2).
i2flag	- x2-coordinate ADI iteration flag (= 0, 1, or 2).
total_iterations	- maximum number of iterations (integer).
test_iterations	- iterations between convergence tests (integer).
test_error	- convergence criterion (unitless).
gradient	- initial potential function (NULL, READ, X1, X2 or defined in FUNCTION command.
scale_factor	- scale factor to apply to initial charge density.

Defaults:

The POISSON command must be given explicitly with specified voltages to obtain a Poisson solution (there is no default command). For the options, the defaults are gradient = NULL, scale_factor = 1, total_iterations = 10,000, test_iterations = 10, and ALGORITHM SOR with omega = 1.

Description:

The POISSON command is used to specify a solution of the generalized Poisson equation to obtain an electrostatic field distribution, and it will produce values for the scalar potential (PHST) as well as the electrostatic field (E1ST, E2ST). This solution may be used to construct an initial condition with space charge (POPULATE, Ch. 19) for a subsequent transient simulation or to represent a TEM wave propagating in the third (symmetry) coordinate (CIRCUIT, Ch. 19).

POISSON Command

You must specify the `poisson_name`, since it is possible to have more than one Poisson solution. This is followed by specifying the number of conductors followed by a list of object names and voltages.

The numerical solution is obtained by one of three methods: successive over-relaxation (SOR), SOR with Chebyshev acceleration (SCA), or alternating direction implicit (ADI). The SOR algorithm uses the parameter `omega` to improve the rate of convergence. When `omega` is between one and two, the algorithm provides over-relaxation. When `omega` is between zero and one, the algorithm is called under-relaxation. For some problems, under-relaxation may provide superior convergence properties.

The SCA algorithm is distinct from the SOR algorithm as it uses a variable over-relaxation coefficient `omega`. The spectral radius is used to calculate the values of `omega` on each iteration cycle. SCA reduces to SOR (`omega=1`) for a radius of 0. If the spectral radius is not known, it can be estimated from the expression

$$\rho = \frac{\cos(\pi / I1MX) + (dx/dy)^2 \cos(\pi / I2MX)}{1 + (dx/dy)^2}.$$

This equation assumes a rectangular `I1MX` x `I2MX` grid, and allows for $dx \neq dy$. This expression holds for homogeneous Dirichlet and Neumann boundary conditions. For periodic boundaries, replace π by 2π . For large values of `I1MX` and `I2MX`, ρ is usually close to unity, and this can be used as a starting estimate. Chebyshev acceleration always provides improved convergence behavior over the standard SOR algorithm.

Generally, the ADI algorithm provides the most rapid convergence. The iteration flags, `i1flag` and `i2flag`, determine which coordinate is implicit. For entries of 0 and 0, both directions are implicit. Alternatively, entries 1 or 2 describe operations on a coordinate that is not implicit in a given iteration; either a single pass or a double pass (first odd, then even) will be performed.

All three methods require iteration, and a convergence test is performed at `test_iteration` intervals. Failing convergence to achieve `test_error`, the maximum number, `total_iterations`, will be performed. These parameters may be altered from their default values using the `TEST` keyword.

The keyword `INITIALIZE` is used to select the type of initialization function to be used. The options here are `NULL`, for no initialization, or `X1` or `X2`, which specify the coordinate of maximum gradient. (Proper choice of arguments, `X1` or `X2`, will enhance convergence.) Alternatively, a function name can be specified, and in this case, the potential is initialized to a two-dimensional field specified by the function. If `READ` is specified, then the initialization values are read in through the use of the `PRESET PHST READ...` command (`PHST` is the name of the scalar potential).

POISSON Command

The keyword POPULATE is used to enter the scale_factor to be applied to the initial charge distribution (POPULATE, Ch. 19).

Restrictions:

1. The treatment of axial symmetry for the spherical coordinate system is approximate and suitable only for large radii.
2. The origin is not treated in the polar (r,θ) coordinate system.
3. Conductors specified as unique potential surfaces must be so defined in the CONDUCTOR command. No more than ten conductors can be referenced in the POISSON command.
4. The over-relaxation coefficient omega is convergent only for $0 \leq \omega \leq 2$. If $0 \leq \omega \leq 1$, then under-relaxation is being used. For some problems, under-relaxation provides superior convergence.
5. The Jacobi spectral radius is restricted to values of $0 \leq \text{radius} \leq 1$.
6. The convergence criterion, test_error, should be set to order 10^{-5} for single precision calculations on a 32-bit machine.

See Also:

AREA, Ch. 10
CIRCUIT, Ch. 19
POPULATE, Ch. 19
PRESET, Ch. 19

References:

B. Goplen, W. M. Bollen, J. McDonald, I. Coleman, and R. E. Clark, "Solution of the Generalized Poisson's Equation in MAGIC," Mission Research Corporation Report, MRC/WDC-R-049, February 1983.

POISSON Command**Examples:**

1. The following command instructs the code to obtain a Poisson solution labeled Pierce for two conductors, cathode and anode, which are set at 0 and 1 volts, respectively. The field is initialized with a gradient in the x1 coordinate. The remaining arguments use default values. Figure 19-2 displays the equipotential contours.

```
SYSTEM SPHERICAL ;  
POISSON pierce 2 cathode 0.0 anode 1.0 INITIALIZE X1 ;
```

POISSON Command

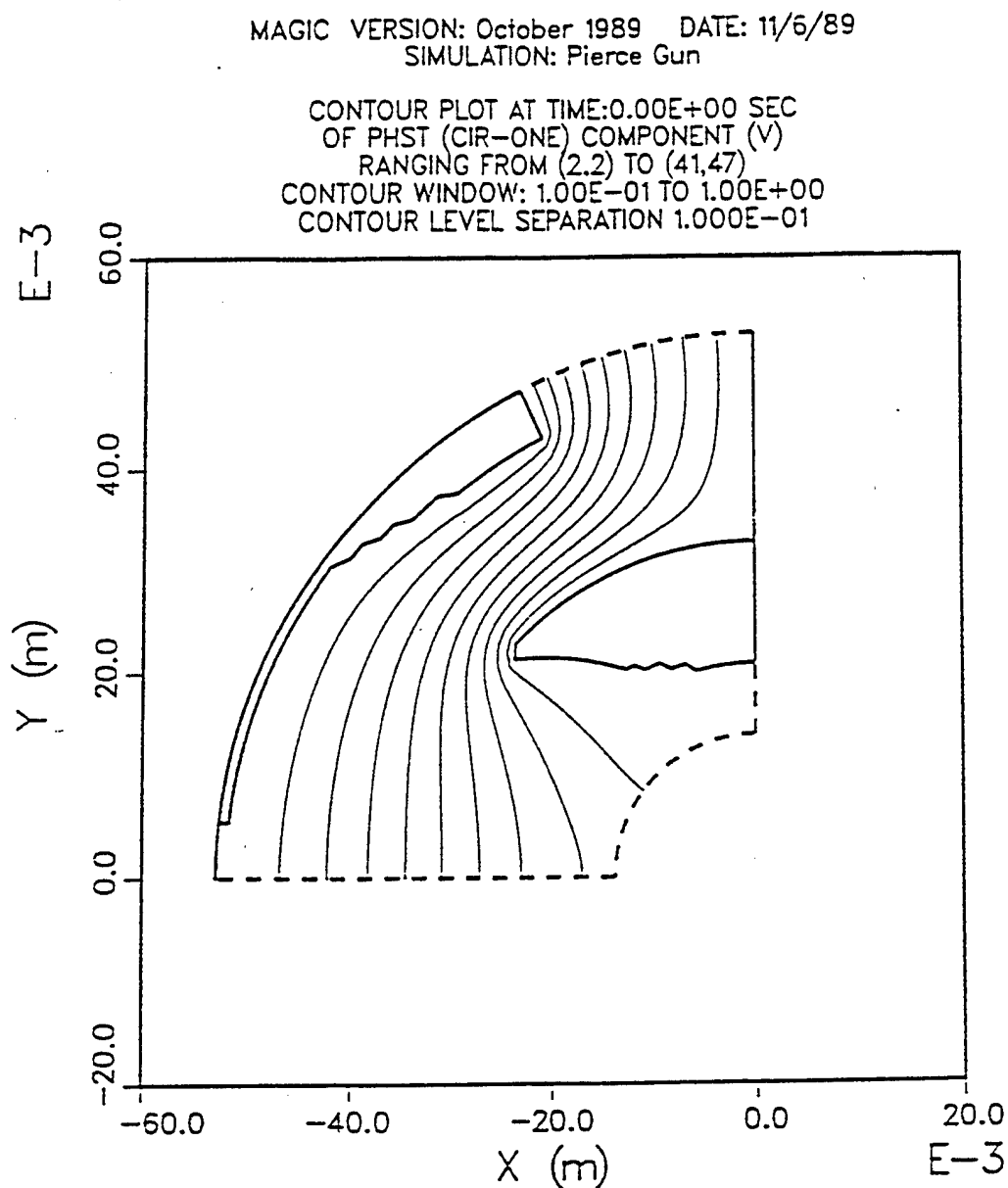


Figure 19-2. Potential contours for Pierce gun.

EIGENMODE Command

Function: Specifies eigenmode electromagnetic algorithm.

Syntax:

```
EIGENMODE
  [ SCAN_AT frequency ]
    [ SCAN_FROM frequency ]
    [ SCAN_TO frequency ]
    [ SCAN_LIST scan_number frequency, frequency, ... ]
  [ WINDOW df(f) ]
    [ ENERGY area_name ]
    [ SAFETY_FACTOR safety_factor ]
    [ ITERATIONS iterations ]
  [ MODE { TE, TM, BOTH } ] ;
```

Arguments:

frequency	- center frequency of eigenmode test (Hz).
scan_number	- number of discrete frequency scans (integer).
df(f)	- width of frequency window in Hz, constant or defined in FUNCTION command.
area_name	- name of spatial area, defined in AREA command.
safety_factor	- safety factor (unitless, default is 1.15).
iterations	- polynomial applications (integer, default is 30.)

Defaults:

The algorithm parameters are completely defined using the following defaults. The SCAN_AT frequency is zero (seeks the lowest mode). The value of df(f) is derived automatically from the simulation dimensions. The area_name is the entire simulation area, the safety_factor is 1.15, and the iterations is 30. The default mode is BOTH.

Description:

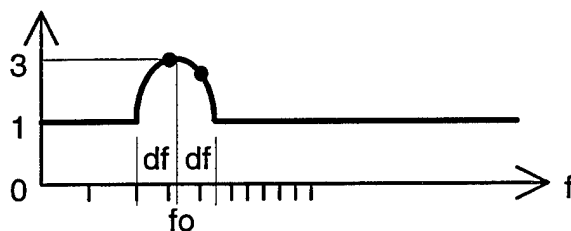
The EIGENMODE command specifies an eigenvalue solution of the fully time-dependent Maxwell's equations. It can be applied to any geometry consisting of non-lossy elements and models, e.g., conductors, symmetries, polarizers, and dielectrics. Incoming and outgoing wave boundaries and other sources or sinks are not permitted. The EIGENMODE algorithm will find one or several eigenmodes and report the corresponding frequencies.

Keep in mind several factors which can alter the frequencies in subtle ways, especially when comparing the results to analytical models. First, if the geometry is not precisely the same, e.g., due to dimensions snapped to the nearest grid, or stair-stepping, the frequency will be altered. The

EIGENMODE Command

single most important factor for good agreement is maintaining the same net volume within the simulation. Also, frequencies will always be downshifted slightly because of spatial finite-difference effects. This effect cannot be quantified exactly, but can be estimated as $f_{FD} = f_{true}[1 - 0.04(k\delta x)^2]$, where k and δx represent typical wave number and grid spacing within the mode. For well resolved wavelengths, e.g., $k\delta x \ll 1$, the effect is small.

The EIGENMODE algorithm applies an operator to a given field pattern which grows eigenmodes in the frequency range $f_0 - \delta f$ to $f_0 + \delta f$, where f_0 and δf are the user-specified frequency and frequency window data items, “freq” and “dfreq”. It grows modes as illustrated in the figure below, such that the center frequency, “ f_0 ”, grows fastest. Each time the operator is applied, the center frequency is grown roughly a factor of 3 above the modes not within the growth window. The operator is applied 30 times, which is sufficient to grow a mode within the frequency window from the noise level to near purity. In general, the computational time required to apply the operator increases as the width of the frequency window is made narrower.



If there are several eigenmodes within the frequency window, e.g., nearly degenerate modes, then the one closest to the center frequency will dominate. A situation where there are two modes within the frequency window is illustrated above. In this case, the algorithm will converge to the nearly central mode, but may contain some contamination from the nearby mode. Thus, a good rule of thumb for the algorithm is that the frequency window should be no larger than the typical spacing between modes. At the end of the run, a concise report will be written to the summary and log files telling the frequency of the converged mode and its confidence level. A poor confidence level is usually an indicator of contamination from a nearly degenerate mode. The near degeneracy can usually be resolved by repeated runs with smaller frequency windows. It is also possible that no modes occur within the search window.

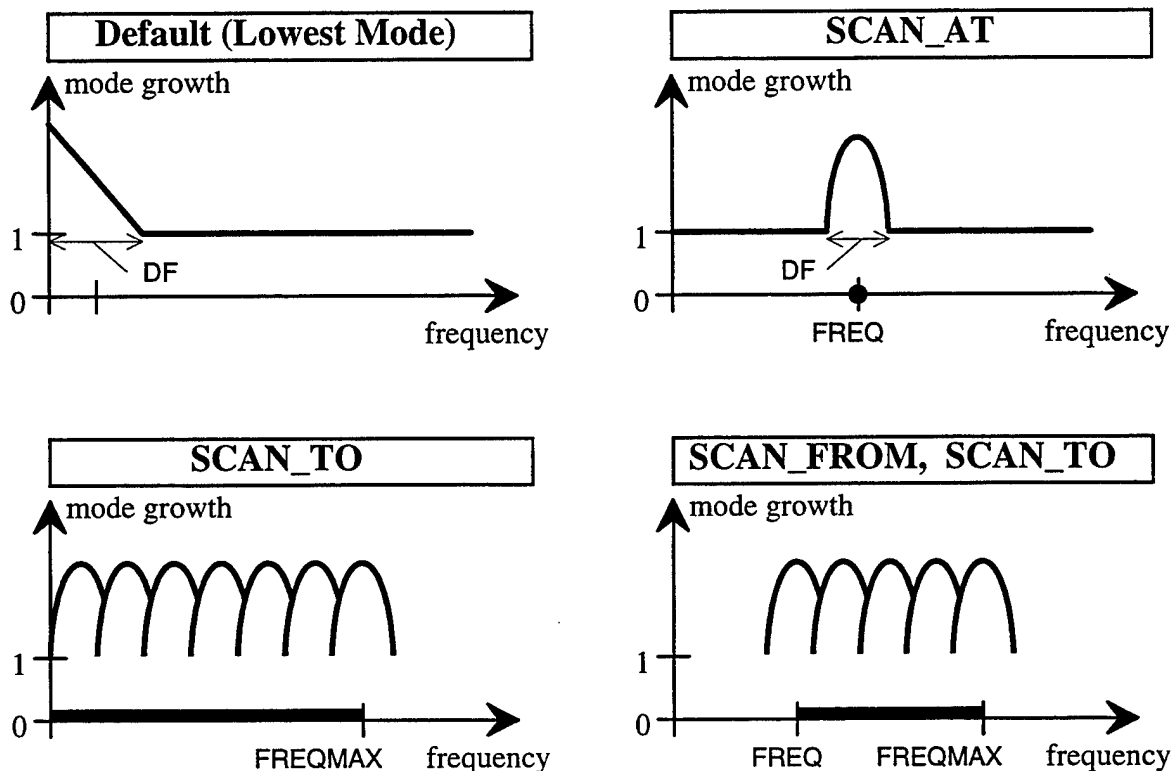
The algorithm contains no inherent preference for the location of the frequency window in the entire spectrum. However, the higher the mode number, the closer together the modes, so that typically $df(f)$ must be made smaller, with the result that more computational time is required. The algorithm will supply a default value for the frequency WINDOW option, based on the overall dimensions of the simulation. This should be satisfactory for most applications, so that typically the WINDOW option need not be used. The exceptions include the presence of a slow-wave structure in the simulation, or when most

EIGENMODE Command

of the volume of the simulation is metal or outside the region of interest. In such cases, the user will need to supply a more appropriate estimate.

There are typically four ways in which the algorithm can be used to search for eigenmodes. These are:

- 1) searching for the lowest mode of a system,
- 2) searching for a mode near some known frequency,
- 3) searching for all the modes up to some maximum frequency, and
- 4) searching for all the modes in some known frequency range.



By default, the SCAN_AT frequency is set to zero, which finds the lowest mode of a system. If the frequency WINDOW option needs to be used, $df(f)$ should be set to a conservatively high estimate of what you think the lowest mode is. For a relatively benign geometry, the lowest mode can be found simply by entering the command name, EIGENMODE, with no additional parameters.

The user can choose to search for a mode near some known frequency, instead of the lowest frequency, using the SCAN_AT option. The algorithm will return the single mode closest to this known frequency, or if no modes occur within the search window, it will indicate so in the Eigenmode Report.

EIGENMODE Command

Often times the user wishes to find all of the lowest modes. The `SCAN_TO` option provides this capability. It essentially repeats the `SCAN_AT` procedure multiple times, moving the search window further up the frequency spectrum each time. The search window is moved by an amount slightly less than the width of the search window, so that no gaps in the spectrum occur. After each search is performed, the resulting mode, if any, is written to the Eigenmode Report. Note that because of the overlap, it is possible for the same frequency to be reported twice in consecutive search windows. The effectiveness of the `SCAN_TO` option depends on using a suitably small frequency window. Typically, the scan should result in at least every third or fourth window having no eigenmode present. If the scan is reporting a frequency in every window, then the search window is too large and should be decreased. Generally speaking, it is much simpler to run the code for a longer time with a narrower window than it is to attempt to decipher what is happening when there are contaminating modes within the search window.

The user may wish to perform a scan on a restricted part of the spectrum. One common example is a finer detail scan, e.g., one with a smaller window, where two nearly degenerate modes are suspected. The restricted scan is accomplished by using the `SCAN_FROM` option together with the `SCAN_TO` option.

The algorithm will automatically start out with a random initial field pattern from which the selected range of eigenmodes can be grown. However, the option exists for the user to initialize the field pattern by using the `PRESET` command to set the electric fields, `E1`, `E2`, and `E3`. This might be desired in rare circumstances where, for example, there are degenerate modes, and the user wishes to exclude one such mode by starting with a field pattern in which that mode is absent.

The `ENERGY_REGION`, `SAFETY_FACTOR`, and `ITERATIONS` options control features which are normally under algorithm control; hence these options will rarely be used. The `ENERGY_REGION` dictates the region of the simulation used to calculate the energy quantities, which lie at the heart of determining the eigenmode frequency of a mode. The default is the entire simulation region. The `SAFETY_FACTOR` controls the estimate of the highest mode in the system, and it is closely related to the Courant limit. By default it is 1.15, e.g., 15% greater than Courant. The `ITERATIONS` option controls the number of times the eigenmode windowing operator is applied. The default is 30, which should result in a virtually pure mode when there is only one single mode within a search window. By increasing the number of iterations, it is possible to improve the convergence to the center-most mode when near-degeneracy occurs. However, in this situation, it is recommended that a smaller search window be used rather than increasing the number of iterations, since this ultimately provides greater accuracy in a shorter amount of time.

The `EIGENMODE` algorithm also creates three default Timers which allow the user to observe the field patterns as the algorithm converges to a solution. These timers are named `TSYS$EIGENMODE`, which triggers when an eigenmode has been found, `TSYS$EIGEN`, which

EIGENMODE Command

triggers after each of the 30 iterations, permitting the user to observe the progress of the algorithm, and `TSYS$FIRST`, which triggers when the algorithm begins a new search window. These timers can be used in `RANGE`, `CONTOUR`, and other output commands in the normal manner. Observing the convergence with `TSYS$EIGEN` can be both mesmerizing and useful, especially if there are closely-spaced, competing modes.

You can use the `MODE` option to select the electromagnetic mode, with the choices being `TE`, `TM`, or `ALL`.

Restrictions:

1. The eigenmode algorithm should be used only in purely electromagnetic simulations (no particles).
2. No outer boundaries (e.g., `PORT`) or other possible electromagnetic sources (e.g., `DRIVER`) or sinks (e.g., `CONDUCTANCE`) are allowed.
3. To be safe, all regions outside of the simulation region of interest should be made perfectly conducting. (Otherwise, convergence can be adversely affected.)

Examples:

The following example uses the `EIGENMODE` algorithm to find the mode closest to 1 GHz. It displays the `B3` field throughout the iterations to the screen and then pauses, displays, and saves the `B3` field after convergence. The result is shown in **Error! Reference source not found.**

```
FREQ0 = 1.0 GIGAHERTZ ;
DFREQ = 0.2 GIGAHERTZ ;
EIGENMODE SCAN_AT FREQ0 WINDOW DFREQ ;
DISPLAY REAL SYS$X1MN:SYS$X2MX SYS$X2MN:SYS$X2MX
CONTOUR SOLIDFILL TSYS$EIGEN FIELD B3 REAL
      SYS$X1MN:SYS$X1MX SYS$X2MN:SYS$X2MX NODUMP ;
CONTOUR SOLIDFILL TSYS$EIGENMODE FIELD B3 REAL
      SYS$X1MN:SYS$X2MX SYS$2MN:SYS$X2MX ;
GRAPHICS SCREEN ;
GRAPHICS COLOR ;
GRAPHICS PAUSE ;
TIMER INITIAL DISCRETE 1 ;
GRAPHICS PAUSEOFF INITIAL ;
GRAPHICS PAUSEON TSYS$EIGENMODE ;
GRAPHICS PAUSEOFF TSYS$EIGFIRST ;
DUMP FORMAT ASCII ;
DUMP NAME "MODE1GHZ." ;
DUMP TYPE SOLIDFILL ;
```

EIGENMODE Command

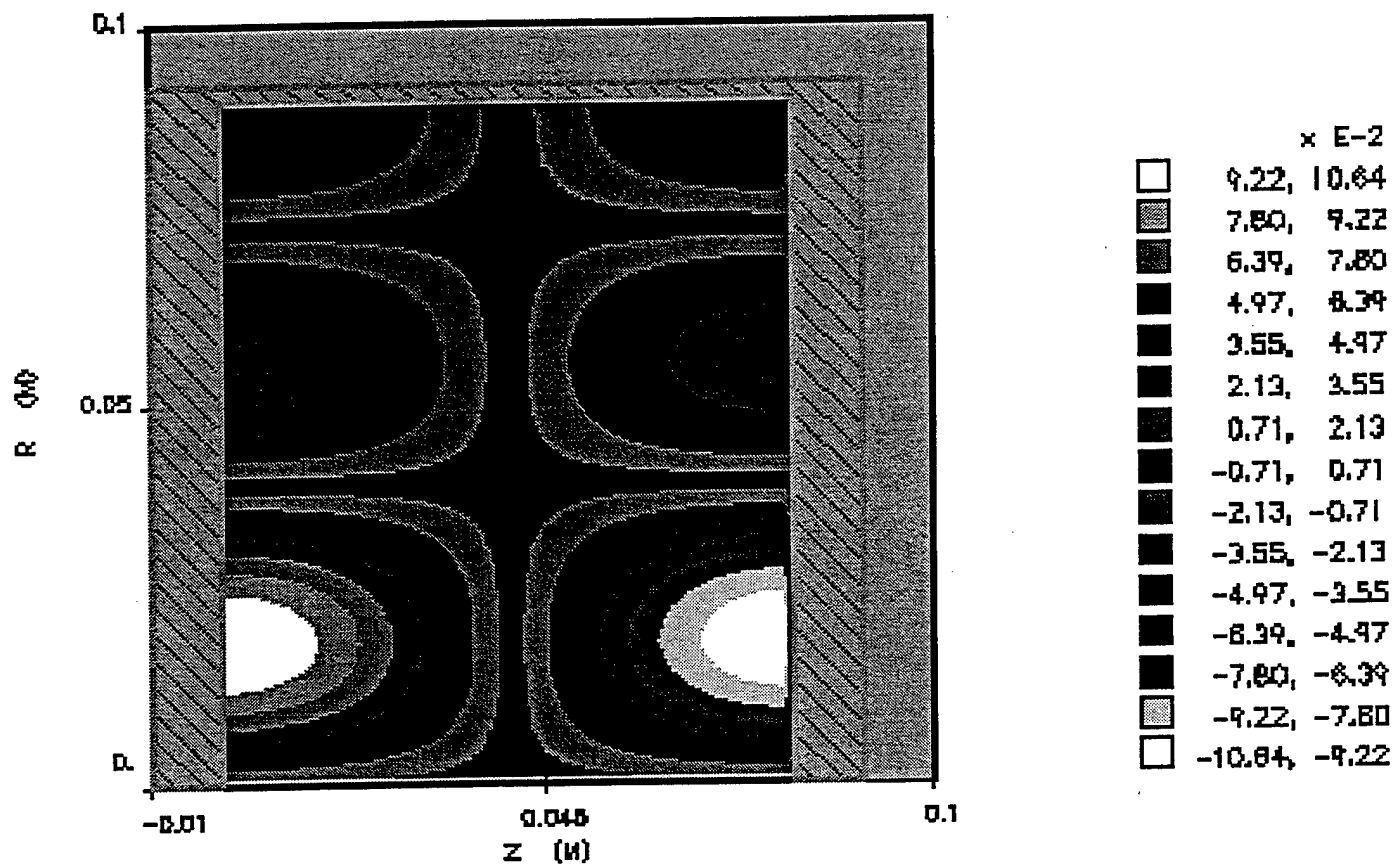


Figure 19-3. B3 field from eigenmode solution.

20. OUTPUT CONTROL

This Chapter covers the following commands:

HEADER
GRAPHICS

You can use these commands to control the content, style, and disposition of output.

The **HEADER** command is used to provide information which is reproduced in all output, such as your organization, your personal identification, and so on. The **GRAPHICS** command is used to dispose output to a video screen or a printer, to enable and disable color, to change the orientation from portrait to landscape, to shift the position of the plot on the screen, and to pause and continue plotting.

This page is intentionally left blank.

HEADER Command

Function: Provides background information which identifies simulation.

Syntax:

```
HEADER
{ ORGANIZATION "company",
  AUTHOR "name",
  DEVICE "model",
  RUN "identifier",
  REMARKS "remarks" } ;
```

Arguments:

company	-	company, organization, or group name.
name	-	name of individual.
device	-	name of device.
identifier	-	run identification.
remarks	-	remarks.

Description:

The HEADER command provides background information on the simulation which identifies it uniquely.

The options are as follows. The ORGANIZATION option allows you to enter the name of your company or organization. Your own name should be provided under the AUTHOR option. The DEVICE option allows specification of the generic name or model number of the device being simulated. The RUN option can be used to enter a unique identifier for an individual simulation. Finally, the REMARKS option can be used to record brief comments relevant to the device or study.

Information from the HEADER command is supplied to all modes of output. In plotted output, it is arranged in "blueprint" format on each plot. (Blanks will be substituted for all options not entered.)

Restrictions:

1. Entering each option requires a separate HEADER command.
2. Remarks should be brief to avoid truncation in the plotted output.

HEADER Command**Examples:**

Information from the following HEADER command will be provided in all output modes.

```
HEADER ORGANIZATION "Mission Research Corp.  
Newington VA" ;  
HEADER AUTHOR "David N. Smithe" ;  
HEADER DEVICE "TWT-L02" ;  
HEADER RUN "L02" ;
```

Figure 20-1 illustrates plotted output from this example.

HEADER Command

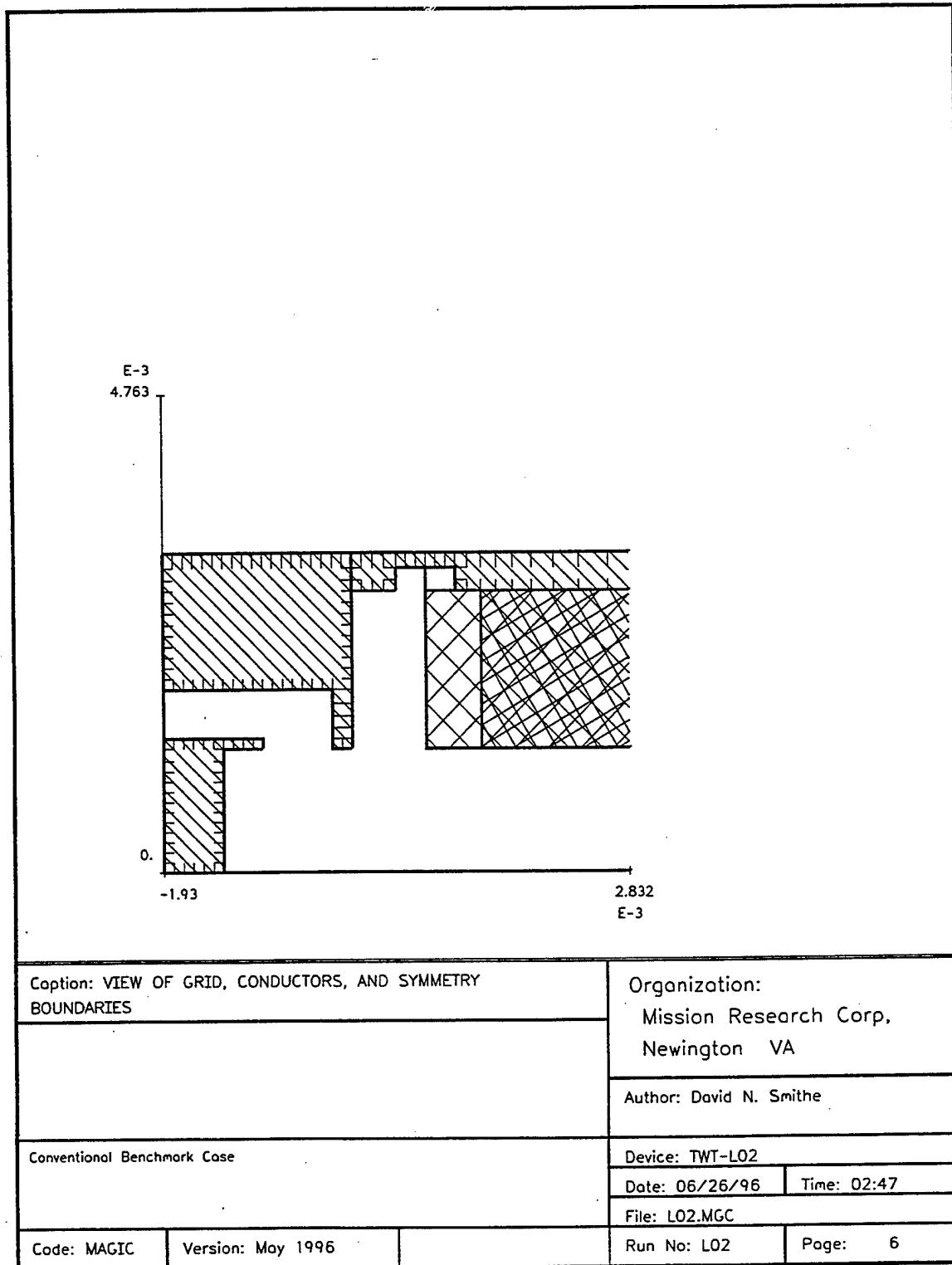


Figure 20-1. Header information in plotted output.

GRAPHICS Command

Function: Controls graphics disposition and device parameters.

Syntax:

```
GRAPHICS
{ PRINTER [ file_name [ device_driver ] ],
  SCREEN ,
  COLOR ,
  NOCOLOR ,
  ORIENTATION { LANDSCAPE, PORTRAIT } ,
  SHIFT x y ,
  PAUSE
  NOPAUSE
  PAUSEON timer ,
  PAUSEOFF timer } ;
```

Arguments:

file_name	-	name of file, user-defined (default = METAFILE).
device_driver	-	name of device driver (see PC installation files).
x,y	-	plot origin shift in inches.
timer	-	timer name, defined in TIMER command.

Description:

The GRAPHICS command provides control over the plotting device and process.

The output destination is either PRINTER or SCREEN. The default is a PostScript PRINTER and the default file_name is METAFILE.PS. However, you can change the file_name and also specify a device driver. You must use SCREEN for interactive processing, and the other OUTPUT options are operable only if SCREEN has been selected.

The COLOR/NOCOLOR options are used to turn color on and off. Color is beneficial in interpreting many forms of graphical output; however, this option may not function depending upon the type of printer graphics linked with the code.

The ORIENTATION option allows you to re-orient the plots from LANDSCAPE to PORTRAIT and vice versa. The SHIFT option allows you to re-position plots on the video screen. To move the plot up and to the right, enter positive numbers for x and y.

The PAUSE/NOPAUSE/PAUSEON/PAUSEOFF options are used to enable/disable a pause between plots. If pause is enabled, you must press the ENTER key on the keyboard after every plot to proceed. PAUSE should not be used with PRINTER graphics (by default, PAUSE is off).

GRAPHICS Command**Restrictions:**

Only one option may be specified with each GRAPHICS command.

Examples:

To plot on the video (screen) in color with a pause between plots, use the following commands.

```
GRAPHICS SCREEN ;  
GRAPHICS COLOR ;
```

This page is intentionally left blank.

21. PRINTED OUTPUT

This Chapter covers the following commands:

COURANT
DEBUG
DIAGNOSE
LINPRINT
PARAMETER
STATISTICS

You can use these commands to produce various types of printed output. These commands are designed primarily for printed output. However, many of the output commands designed for graphical output also have print options.

The **COURANT** command calculates the Courant stability calculation using the specified electromagnetic field algorithm, time step, and spatial grid. The **DEBUG** command is used to print internal data from specified algorithms for debugging purposes. The **DIAGNOSTIC** command is similar, except that it is used to print routine results such as the spatial grid and the Courant criterion. The **LINPRINT** command can be used to print tables of electromagnetic field values within a specified spatial area. The **PARAMETER** command can be used to assign variables which are automatically recorded in the Summary file. (Otherwise, the variable functions identically to one created with an **ASSIGN** command.) You can use the **STATISTICS** command to keep track of particle statistics during the simulation.

This page is intentionally left blank.

COURANT Command

Function: Calculates electromagnetic stability.

Syntax:
COURANT { SEARCH, point } ;

Arguments:
point - name of spatial point, defined in POINT command.

Description:

The PRINT COURANT command performs a Courant stability calculation. It can be printed by a DIAGNOSE command to the log file, and can be located by searching for the word "COURANT." The SEARCH option will search the entire spatial grid for the most restrictive case, while specifying point will give the result for that specific location only.

All of the electromagnetic field algorithms have an upper limit to the time step which depends upon the algorithm and the spatial grid. Use of a time step which exceeds this limit will cause numerical instability leading to catastrophic failure. The stability calculation is performed automatically at the beginning of the simulation. If a violation is detected, an error message will be printed and the simulation will be terminated.

Note that the Courant calculation is a straightforward evaluation of standard expressions which are Maxwell-algorithm-dependent (MAXWELL, Ch. 17). It does not account for factors such as an unusual spatial grid or a non-cartesian coordinate system. For example, the criterion for a cylindrical system which includes the axis may be smaller by a factor of as much as $\sqrt{2/3}$, or 0.82.

See Also:

DIAGNOSE, Ch. 21
TIME_STEP, Ch. 17
MAXWELL algorithms, Ch. 17

Restrictions:

The actual criterion may be smaller than the one calculated. However, a safety factor of 0.8 is believed to be sufficient to account for all known departures from the standard result.

COURANT Command**Examples:**

The following commands will calculate the Courant criterion and write the results to the log file.

```
COURANT SEARCH ;  
DIAGNOSE COURANT 1 0 0 ;
```

DEBUG Command

Function: Specified diagnostic output for debugging.

Syntax:

```
DEBUG ALL ;  
DEBUG ON name [istart istop [istep]] ;  
DEBUG OFF [name] ;  
DEBUG LIST ;  
DEBUG PAUSEON ;  
DEBUG PAUSEOFF ;
```

Arguments:

name - diagnostic name (alpha).
istart - beginning time index (integer).
istop - ending time index (integer).
istep - time index interval (integer).

Description:

The DEBUG command is used to turn on and off diagnostic output for debugging the simulation. The option, ALL, turns on all available diagnostics. The user is warned to exercise this option with caution. Option ON turns on the diagnostic of the specified name. Optional parameters set a time step range limit for the diagnostic to be on. If istart and istop are not specified, the diagnostic will be on until it is turned off. If the interval, istep, is not specified, then a value of unity is assumed by default. Option OFF turns off all diagnostics or just the specified diagnostic. The option, LIST, will provide a list of the available diagnostics at the terminal. The option, PAUSEON, will cause the program to pause and wait for the user to press the “enter” key after each diagnostic message.

See Also:

DIAGNOSE, Ch. 21

DIAGNOSE Command

Function: Specifies diagnostic output.

Syntax:
DIAGNOSE diagnostic kstep kstart kstop ;

Arguments:

diagnostic	-	diagnostic name (alpha).
kstep	-	time index step (integer).
kstart	-	starting time index (integer).
kstop	-	stopping time index (integer).

Description:

The DIAGNOSE command requests detailed printed output from various subroutines for diagnostic purposes. The desired output is specified by the diagnostic. Most diagnostics are given command names. The printed output is provided at times specified by the time-sequence indices, kstep, kstart, and kstop. This command functions like a Fortran do-loop except that the time step index, kstep, must be positive.

See Also:

DEBUG, Ch. 21

Examples:

To print out the spatial grid at the beginning of the simulation, one can use the command,

```
DIAGNOSE SPACING 1 0 0 ;
```

The Courant stability criterion can be printed out with the command,

```
DIAGNOSE COURANT 1 0 0 ;
```

Numerous other diagnostics are available if required.

LINPRINT Command

Function: Prints a field component table in scientific notation.

Syntax: LINPRINT timer field area orientation scale ;

Arguments:

timer	- timer name, defined in TIMER command.
field	- field component (E1, E2, ...).
area	- name of spatial area, defined in AREA command.
orientation	- spatial orientation. = X1, x1 vertical and x2 horizontal. = X2, x2 vertical and x1 horizontal.
scale	- scaling factor (unitless).

Description:

The LINPRINT command causes fields to be printed as a table of numbers in scientific notation. The table displays the data more precisely, though less compactly, than a plot. The values of field that are within the specified area are printed at times specified by the TIMER command.

The field component values are multiplied by the scaling factor, scale, and are printed out in scientific (exponential) notation to four significant digits. These values are arranged in a table with vertical and horizontal axes arranged according to the specified orientation. The values corresponding to a single grid index of the vertical axis are printed on a single line or, if more than ten values, a series of lines which wrap around. The grid index is printed to the left of these values.

Restrictions:

The total number of LINPRINT commands in a simulation is limited to ten.

See Also:

TIMER, Ch. 11

PARAMETER Command

Function: Defines which variables will be listed in the summary file.

Syntax:
PARAMETER variable = expression ;

Arguments:

variable - character variable.
expression - arithmetic or string expression.

Description:

The PARAMETER command defines variables in the exact same manner as the ASSIGN command. However, in addition to the variable definition, the PARAMETER command also displays the variable in the summary file.

The variable type naming conventions and the expression syntax for PARAMETER variables is identical to that for ASSIGN, and the use of PARAMETER variables with other commands is identical to that for ASSIGN commands.

See Also:

ASSIGN, Ch. 6

Examples:

The following example creates a ASSIGN variable called LARS and a PARAMETER variable called DARS and uses them to create a third variable called HARS.

```
DEFINE LARS = -1 ;      ! not listed in Summary File
PARAMETER DARS = 1. ;   ! listed in Summary File
HARS = LARS + 3*DARS ;   ! result = 2
```

STATISTICS Command

Function: Specifies times at which particle statistics are collected and printed.

Syntax:
STATISTICS timer ;

Arguments:
timer - timer name, defined in TIMER command.

Description:

The STATISTICS command causes particle statistics to be collected during the simulation. These statistics include the number created, the number destroyed, the number existing, etc. This information is printed in the log file at the specified intervals during the simulation, and totals and averages are printed in the summary at the end of the simulation.

Particle statistics are printed at simulation times specified by the timer. The particle statistics printed are:

- number existing at the last measurement,
- number created since the last measurement,
- number destroyed since the last measurement,
- number existing at present, and
- error in the number of particles.

The error reveals any variation between the actual count of existing particles and the difference between creation and destruction counts. If this number is anything but zero, then the simulation results are questionable.

In the summary at the end of the simulation, several additional statistics are printed out. These are:

- total number created during the simulation,
- total number destroyed during the simulation,
- highest number existing during the simulation, and
- average number existing during the simulation.

Note that the summary will print zeros for these statistics in the event that the STATISTICS command has been omitted.

This page is intentionally left blank.

22. TIME PLOTS

This Chapter covers the following commands:

OBSERVE options
OBSERVE SET_DEFAULT
OBSERVE AIRCHEM
OBSERVE CIRCUIT
OBSERVE ENERGY
OBSERVE FIELD
OBSERVE FIELDENERGY
OBSERVE FLUX
OBSERVE POYNTING
OBSERVE QMEA
OBSERVE STATISTICS
OBSERVE STRUT
OBSERVE TRAMLINE
ENERGY
FLUX

You can use the OBSERVE commands to plot the value of a simulation variable vs. time. There are many processing options which can be applied to the data, and these are described by the OBSERVE options command. Defaults are set for many of these options, which can be re-set using the OBSERVE SET_DEFAULT command. All of the remaining OBSERVE commands involve plotting a particular data_type, such as FIELD, ENERGY, etc.

The ENERGY and FLUX commands calculate energy and particle flux variables, respectively. They do not output results directly; however, the variables which they calculate may be accessed and plotted by using OBSERVE and RANGE (Ch. 23) commands.

This page is intentionally left blank.

OBSERVE [options] Command

Function: Specifies processing options for plotting simulation variable vs. time.

Syntax:

```
OBSERVE data_type arguments
      [ INTERVAL multiple ]
      [ NAMESUFFIX suffix ]
      [ TRANSFORM function(xobs,time,time_step)]
      [ { DIFFERENTIATE, INTEGRATE } ]
      [ FILTER fraction ]
      [ FFT index ]
      [ WINDOW TIME timelo timehi ]
      [ WINDOW FREQUENCY freqlo freqhi ]
      [ { PLOT, NOPLOT } ]
      [ { DUMP, NODUMP } ]
      [ { PRINT, NOPRINT } ] ;
```

Arguments:

data_type	-	the type of observer variable. = AIRCHEM, = CIRCUIT, = ENERGY, = FIELD, = FIELDENERGY, = FLUX, = POYNTING, = QMEA, = SET_DEFAULT, = STATISTICS, STRUT, or = TRAMLINE.
arguments	-	arguments for each data_type, see commands.
multiple	-	time-step interval between observations (integer).
suffix	-	name given to observe variable (alpha).
function	-	transform function, defined in FUNCTION command.
fraction	-	filter fraction (0<fraction<1).
index	-	Fourier-transform option (integer). = 1, data only, no FFT. = 2, FFT only, real and imaginary parts. = 3, FFT only, magnitude. = 4, data and FFT, real and imaginary parts. = 5, data and FFT, magnitude.
timelo, timehi	-	time boundaries of FFT integration (sec).
freqlo, freqhi	-	frequency boundaries of FFT plot (Hz).

OBSERVE [options] Command

Defaults:

The default values for the optional command arguments are listed below. Note that the default values can be reset repeatedly (OBSERVE SET_DEFAULT, Ch. 22).

Keyword	Argument	Default
INTERVAL	multiple	1
NAMESUFFIX	suffix	numerical order
TRANSFORM	function	not performed
INTEGRATE	-	not performed
DIFFERENTIATE	-	not performed
FILTER	fraction	1.0 => no filtering
FFT	index	1 => data only
WINDOW TIME	timelo, timehi	entire range
WINDOW FREQUENCY	freqlo, freqhi	entire range
PLOT	-	YES
DUMP	-	YES
PRINT	-	NO

OBSERVE [options] Command**Description:**

The OBSERVE commands provide the capability to record specific simulation variables as a function of time. Individual commands are structured for each data_type (e.g., OBSERVE FIELD to obtain electromagnetic field data), and each data_type has a different set of arguments. However, there are a number of common data processing operations (e.g., FFT) which can be applied in any OBSERVE command. The common options and operations are discussed here. Individual commands describe each of the data_types and their specific arguments.

The keyword INTERVAL is used to select the observation interval, in time steps. The frequency with which the code records simulation variables is specified by multiple, the number of time steps between measurements. A small value for multiple, e.g., 1, results in time history plots with closely spaced points and Fourier plots with high frequencies. For simulations with a great many time steps, e.g., 500,000 or more, it is advisable to increase multiple to reduce the size of the data record. Usually, multiple should not be increased to the point where the time scales of interest are not well resolved, though. (The present implementation applies the last-entered value of multiple to all time histories.)

Associated with each OBSERVE command is a system variable, OBS\$suffix, where suffix can be specified with the NAMESUFFIX option. This variable can be used in the same manner as a variable from a ASSIGN command. Its value is updated once every observed time step. Use of this variable in a FUNCTION command can provide feedback to an incoming wave (PORT, Ch. 12) a current-density source (DRIVER, Ch. 19) or a voltage source (CIRCUIT, Ch. 19). Use of this variable in another OBSERVE TRANSFORM function can lead to manipulations involving several observe variables, for example, getting impedance from voltage and current observations. Finally, use of this variable in a PARAMETER command expression after the simulation, i.e., between the START and STOP commands, can provide a way to spot-check simulation behavior from the summary file. If the NAMESUFFIX option is not invoked, then the default suffix is a number, 1, 2, 3, etc., corresponding to the order in which the OBSERVE commands are entered.

There are data processing operations which can be performed to modify observables. The keywords are TRANSFORM, DIFFERENTIATE, INTEGRATE, FILTER, and FFT. Use of one or more of these keywords causes the raw data to be modified in a particular fashion and sequence. Except for FFT, the modified data is passed to the OBS\$suffix variables. When several of these operations are used simultaneously on the same data, the user should be aware of the particular order in which they are performed:

- 1) TRANSFORM,
- 2) DIFFERENTIATE or INTEGRATE,
- 3) FILTER, and
- 4) FFT.

OBSERVE [options] Command

The TRANSFORM keyword is used to apply a transformation to the selected observable. The function has three explicit variable arguments: data (the observed data), time, and the time_step(between observations). Other variables and functions, including other observe variables, may be referenced by the function in the normal fashion. (See the FUNCTION command for details. Note that when other observe variables are referenced, their value corresponds to the previous time step, and not the current time step.) If a transform is performed, the name of the transformation function is used for the y-axis label.

The DIFFERENTIATE keyword performs a time derivative of the data. Similarly, the INTEGRATE keyword performs a time integration of the data. The DIFFERENTIATE and INTEGRATE keywords are mutually exclusive; only one or the other may be specified.

The FILTER keyword permits the user to apply an RC-type filter to the data. It results in a new signal, g_n , at time step n , given by the following formula:

$$g_n = f s_n + (1-f) g_{n-1},$$

where s_n is the original observed signal at time step n , and f is the filter fraction, a number between zero and one. It is assumed that $g_0=0$. This is a discrete approximation of the traditional RC-type filter given by:

$$g(t) = (f / \delta t) \int_0^t dt' s(t') e^{-(t-t')f/\delta t},$$

where δt is the time step times the interval multiple, and the filter fraction, f , corresponds to the value of $\delta t/RC$. As a general rule, filtering occurs over a time scale given approximately by $\delta t/f$. (Note that a change in the interval multiple will affect the filter, unless the filter fraction is also changed to account for it.)

The keyword FFT is used to select the Fourier transform option. The index specifies the combination of plots desired for any observe variable. The keyword WINDOW is used in conjunction with the FFT keyword to select the time interval, t_{lo} to t_{hi} , for the Fourier integration, and the frequency boundaries, f_{lo} and f_{hi} , for the plots of the transformed data.

The actual FFT process proceeds as follows. First the signal from t_{lo} to t_{hi} is linearly interpolated onto a new set of N uniformly spaced points, such that $N=2^M$ is the next largest exact power of two. The exact interpolation times are $t_j = t_{lo} + (j-1/2)Dt$, where $Dt = (t_{hi} - t_{lo})/N$. The FFT frequencies are $f_k = kDf$, where $Df = (t_{hi} - t_{lo})^{-1}$, and run from $k=0$ to $k=N/2$, e.g., $f_{max} = (2Dt)^{-1}$. Starting from the N interpolated signal values, S_j , the FFT is given by:

$$\tilde{S}_k = \sigma_k \Delta t \sum_{j=1,N} S_j e^{-i2\pi f_k t_j}, \text{ e.g., } \tilde{S}(f) = 2 \int_{t_{lo}}^{t_{hi}} dt S(t) e^{-i2\pi f t},$$

OBSERVE [options] Command

where $\sigma_k = 1$ for $k=0$ and $\sigma_k = 2$ otherwise. It is precisely the quantity $\tilde{S}_k$ which is displayed in the FFT plots. The inverse formula for this convention is:

$$S_j = \Delta f \sum_{k=0, N/2} \text{Re}\{\tilde{S}_k e^{i2\pi f_k t_j}\} \quad , \text{ e.g., } S(t) = \int_0^{f_{\max}} df \text{Re}\{\tilde{S}(f) e^{i2\pi f t}\} \quad .$$

The normal factor of 2π is missing because of the integration over frequency, f , instead of angular frequency, ω . The extra factor of two in the transform is compensated for by an integration over only positive frequencies in the inverse transform, a common manipulation where pure real signals are concerned.

Time history records can be stored and displayed by three different methods:

- 1) as a plot in a metafile or on screen (e.g., the default behavior),
- 2) as a data record in a DUMP file, or
- 3) as column data in the log file.

A plot of the data in a metafile or on screen is the normal result of an OBSERVE command. If a DUMP TYPE OBSERVE or DUMP TYPE ALL command is issued, then the data is also stored as a time history record in the GRD file, for later display or post-processing. Under certain circumstances, it may be desirable to have a particular set of observe data displayed on screen, but not stored, in order to save disk space, or perhaps stored but not displayed for some other reason. The NOPLOT and NODUMP options allow control of data display and storage on a per command basis. Complementary PLOT and DUMP options are also available, in case the data display and storage is controlled by variable substitution, or in case the defaults have been reset.

It is also possible to generate a columnar listing of the observe data in the log file after the last time step has executed. This option is normally turned off; however, it can be activated with the PRINT option, or deactivated with the NOPRINT option. (The present implementation applies the same PRINT option to all time histories. Therefore, the last entered PRINT or NOPRINT option governs printing to the log file.)

Normally time history data is stored and displayed only once after the last time step has been executed. On certain platforms, e.g., the PC, it is possible to display and store the current state of the observe data at any stage during the simulation. Depressing the F5 key on the keyboard will initiate immediate display and storage of the observe data. (See KEYBOARD, Ch. 9 for further details.)

OBSERVE [options] Command**Restrictions:**

After the simulation is complete, time-history data is loaded into the memory space used to hold field components. Consequently, the total number of time-history records must be less than one-fourth of this amount of storage.

See Also:

- ASSIGN, Ch. 6**
- FUNCTION, Ch. 6**
- KEYBOARD, Ch. 9**
- PORT, Ch. 12**
- DRIVER, Ch. 19**
- CIRCUIT, Ch. 19**
- DUMP, Ch. 25**
- OBSERVE SET_DEFAULT, Ch. 22**
- OBSERVE AIRCHEM, Ch. 22**
- OBSERVE CIRCUIT, Ch. 22**
- OBSERVE ENERGY, Ch. 22**
- OBSERVE FIELD, Ch. 22**
- OBSERVE FIELDENERGY, Ch. 22**
- OBSERVE FLUX, Ch. 22**
- OBSERVE POYNTING, Ch. 22**
- OBSERVE QMEA, Ch. 22**
- OBSERVE STATISTICS, Ch. 22**
- OBSERVE STRUT, Ch. 22**
- OBSERVE TRAMLINE, Ch. 22**

OBSERVE SET_DEFAULT Command

Function: Resets default options for all subsequent OBSERVE commands.

Syntax:
OBSERVE SET_DEFAULT [options] ;

Arguments:
options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE SET_DEFAULT, can be used to reset the OBSERVE option default values, thus saving the user from having to include non-default settings in each OBSERVE command explicitly. The new defaults apply to all subsequent OBSERVE commands until a new OBSERVE SET_DEFAULT command is issued.

The most common usage will be to set the default INTERVAL option for all observes. In addition to INTERVAL, all the other options can also have their defaults set, except for the TRANSFORM and NAMESUFFIX options.

See Also:

OBSERVE [options], Ch. 22

Examples:

In this example, the default observe interval is reset for all observes, and the filter option is reset for four of the six field observes, i.e., for B3, E3, B1, and B2.

```
OBSERVE SET_DEFAULT INTERVAL 10 ;
OBSERVE FIELD E1 REAL XOBSA,YOBSA XOBSB,YOBSB ;
OBSERVE FIELD B3 REAL XOBSA,YOBSA XOBSB,YOBSB FILTER 0.01 ;
OBSERVE SET_DEFAULT FILTER 0.01 ; ! ...filter by default
OBSERVE FIELD E3 REAL XOBSA,YOBSA XOBSB,YOBSB ;
OBSERVE FIELD B1 REAL XOBSA,YOBSA XOBSB,YOBSB ;
OBSERVE FIELD B2 REAL XOBSA,YOBSA XOBSB,YOBSB ;
OBSERVE SET_DEFAULT FILTER 1.0 ; ! ...back to no-filtering
OBSERVE FIELD E2 REAL XOBSA,YOBSA XOBSB,YOBSB ;
```

OBSERVE AIRCHEM Command

Function: Specifies air chemistry variable to be plotted vs. time.

Syntax:

```
OBSERVE AIRCHEM variable point  
[options] ;
```

Arguments:

- variable - air chemistry variable (see AIR-CHEMISTRY, Ch. 14).
= RHOE, RHON, RHOP, QION, BAVA, BATT, XMBE,
or SIGMAC.
- point - name of point, defined in POINT command.
- options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE AIRCHEM, is used to plot air-chemistry variables vs. time. The variable is one of the air-chemistry fields, and the point is the location in space.

See Also:

AIR-CHEMISTRY, Ch. 14
OBSERVE [options], Ch. 22

OBSERVE CIRCUIT Command

Function: Specifies circuit variable to be plotted vs. time.

Syntax:

```
OBSERVE CIRCUIT poisson_name variable
                    [options] ;
```

Arguments:

- poisson_name - Poisson solution name (alpha).
= arbitrary, user-defined in POISSON command.
- variable - circuit variable (see CIRCUIT, Ch. 12).
= CHARGE, CURRENT, VOLTAGE, ENERGY,
POWER, DCHARGE, DVOLTAGE, DENERGY.
- options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE CIRCUIT, is used to specify a circuit model variable to plot vs. time. The poisson_name specifies the circuit. (See CIRCUIT, Ch. 12 a list of the valid observable variables.)

See Also:

CIRCUIT, Ch. 12
POISSON, Ch. 19
OBSERVE [options], Ch. 22

Examples:

The following commands instruct the code to observe and record the circuit variables, CURRENT and VOLTAGE. At the end of the simulation, plots of the time histories of these variables are produced. The observe variables are recorded every time step, since itmalt is left at its default value.

```
OBSERVE CIRCUIT MYTEST CURRENT ;
OBSERVE CIRCUIT MYTEST VOLTAGE ;
```

OBSERVE ENERGY Command

Function: Specifies energy variable to be plotted vs. time.

Syntax:

OBSERVE ENERGY variable index integrate
[options] ;

Arguments:

- variable - energy balance variable (see ENERGY, Ch. 22).
= CREATED, DESTROYED, SURVIVING, GAINED,
VOLTAGE, LOOKBACK, MATCH, BOUNDARY,
ELECTRIC, MAGNETIC, EM, TOTAL, RESIDUAL,
and RATIO.
- index - species/surface/field component index (integer or alpha).
= species_name, defined in SPECIES command.
= ALL, sum over species.
= 0, sum energies over index.
> 0, choose indexed energy.
- integrate - time integration option (integer).
= 0, power (J/sec).
= 1, energy (J).
- options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE ENERGY, is used to specify an energy balance variable to be plotted vs. time.

The index identifies a particular species, boundary, or component when observing particle energy, boundary flux, or field energy, respectively. If the index is set to zero, the energies corresponding to all valid values of ienergy are summed. For some values of aenergy, such as EM and RESIDUAL, the index must be zero.

The value of integrate selects either a rate of change in the energy variable or an accumulated energy. Note that some energies are instantaneous, such as electric field energy, while others are accumulated over time, such as energy lost through outer boundaries. Rates of change are available for either type of variable.

OBSERVE ENERGY Command

See Also:

ENERGY, Ch. 22

OBSERVE [options], Ch. 22

Restrictions:

1. The ENERGY command must be used for this measurement to be made.
2. The timer used in the ENERGY command should be matched to the observe interval selected.

OBSERVE FIELD Command

Function: Specifies electromagnetic field variables to be plotted vs. time.

Syntax:

```
OBSERVE  FIELD field spatial_object
          [options] ;
```

Arguments:

- field** - field component (see MAXWELL, Ch. 17).
= B1, B2, B3, E1, E2, E3, D1, D2, D3, DIE1,
DIE2, DIE3, B1ST, B2ST, B3ST, B1AV,
B2AV, B3AV, E1AV, E2AV, E3AV, J1, J2, J3,
Q0, QERR, SIGMA, SIGMAC, E1ST, E2ST,
and PHST.
- spatial_object** - name of point or line, defined in POINT or LINE
command.
- options** - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE FIELD, is used to specify a field to be plotted vs. time. If the specified spatial_object is a point, the field at that location will be measured. If the spatial_object is a line, then an integral of the specified field will be taken along the line. This will produce a voltage measurement, as opposed to an electric field, for example.

See Also:

MAXWELL algorithms, Ch. 22
OBSERVE [options], Ch. 22

Examples:

The integrated E2 field is recorded and a time history plot is produced.

```
LINE  Line_AB  STRAIGHT  X,Ya  X,Yb  ;
OBSERVE  FIELD  E2  Line_AB  ;
```

OBSERVE FIELDENERGY Command

Function: Specifies electromagnetic field energy variable to plotted vs. time.

Syntax:

OBSERVE FIELDENERGY variable area
[options] ;

Arguments:

variable - energy balance variable.
= EM, ELECTRIC, MAGNETIC, TM, TE,
E1, E2, E3, B1, B2, or B3.
area - name of spatial area, defined in AREA command.
options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE FIELDENERGY, is used to specify an electromagnetic energy variable to be plotted vs. time.

The variable may take the value of EM, ELECTRIC, MAGNETIC, TM, TE, E1, E2, E3, B1, B2, or B3. The value of EM yields the total electromagnetic energy in both modes. The value of ELECTRIC yields the total electric field energy in both modes. The value of MAGNETIC yields the total magnetic field energy in both modes. The value TM yields the total electromagnetic energy in the TM mode (E1, E2, B3). The value TE yields the total electromagnetic energy in the TE mode (B1, B2, E3). The selections E1, E2, ... through B3 yield the energy of the respective field component.

The area is used to restrict the area of measurement.

See Also:

ENERGY, Ch. 22

OBSERVE [options], Ch. 22

Restrictions:

1. The ENERGY command must be used for this measurement to be made.
2. The timer used in the ENERGY command should be matched to the observe interval selected.

OBSERVE FLUX Command

Function: Specifies flux variable to be plotted vs. time.

Syntax:

```
OBSERVE FLUX flux_name variable
      [options] ;
```

Arguments:

flux_name - name of flux measurement, defined in FLUX command.
variable - flux variable (see FLUX, Ch. 22).
 = CURRENT-DENSITY, POWER-DENSITY,
 CHARGE-DENSITY, ENERGY-DENSITY,
 CURRENT, POWER, CHARGE, ENERGY,
 PARTICLE-CHARGE, PARTICLE-ENERGY,
 or MACRO-PARTICLES.
options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE FLUX, specifies a particle flux variable to be plotted vs. time. The flux_name, specifies the name of the flux measurement, as defined in a FLUX command. The variable specifies the precise measurement to be plotted. It is best to have the FLUX timer trigger on time steps which are integer multiples of the interval used in OBSERVE.

See Also:

FLUX, Ch. 22
OBSERVE [options], Ch. 22

Restrictions:

1. A FLUX command must be used for this measurement to be made.
2. The timer used in the FLUX command should be an integer multiple of the observe interval selected.

Examples:

The following example illustrates the use of OBSERVE in recording flux information. This example is taken from a simulation of a klystron amplifier. A flux surface was defined to lie on an inside segment of the klystron collector. The OBSERVE command was used to record the energy flux due to electron deposition on the flux surface named COLL4. Figure 22-1 illustrates the observe results for this case.

OBSERVE FLUX Command

```
TIMER FOR-FLUX PERIODIC INTEGER 0 99999 2 ;  
FLUX COLL4 POSITIVE FOR-FLUX ELECTRON ;  
OBSERVE FLUX COLL4 ENERGY INTERVAL 2 ;
```

OBSERVE FLUX Command

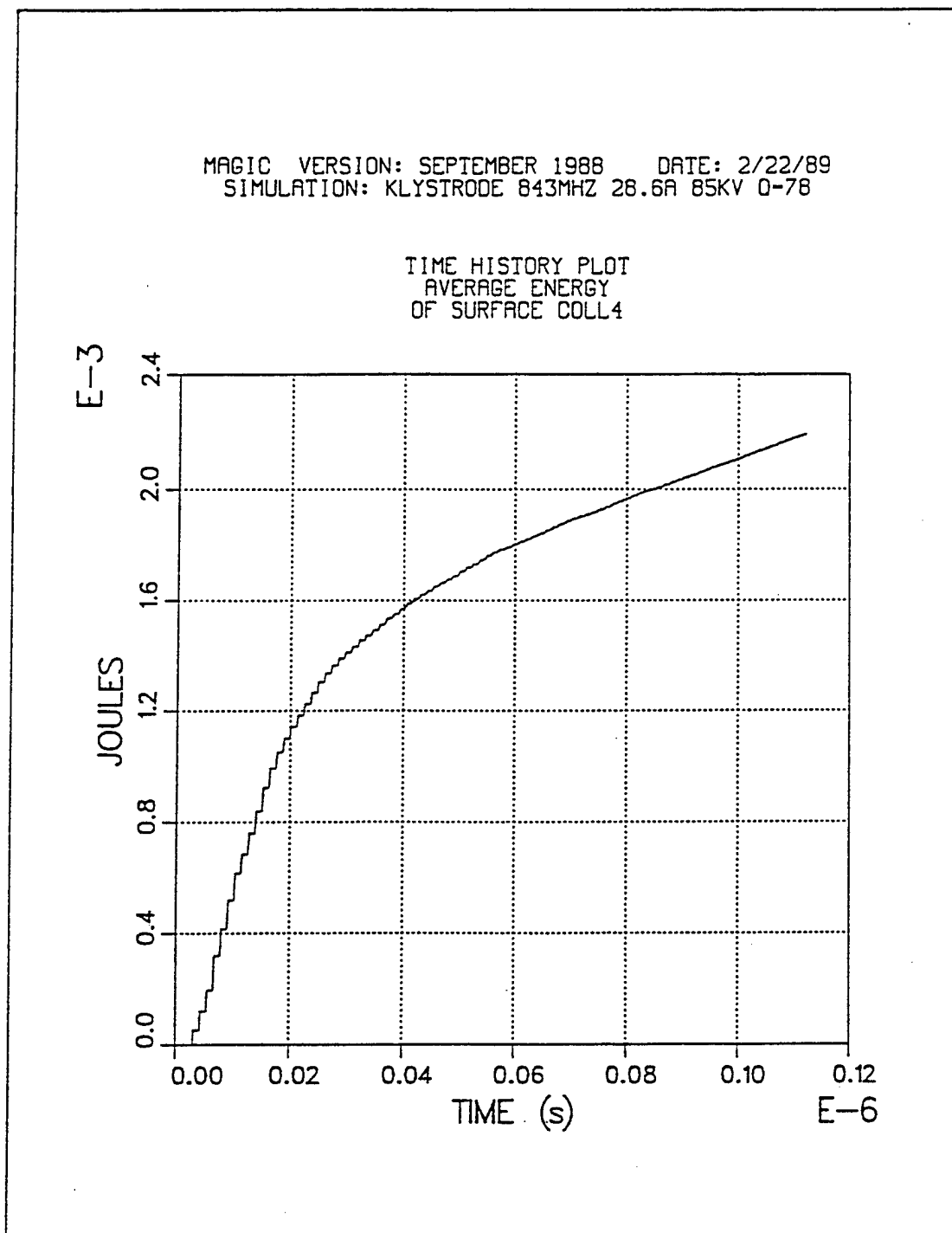


Figure 22-1. Time history of flux data.

OBSERVE POYNTING Command

Function: Specifies Poynting flux variable to be plotted vs. time.

Syntax:

```
OBSERVE POYNTING mode line_name { POSITIVE, NEGATIVE }  
[options] ;
```

Arguments:

mode	-	component of Poynting flux. = TM, (E1xB3 or E2xB3). = TE, (E3xB2 or E3xB1). = ALL (both TE and TM).
line_name	-	name of line for Poynting measurement, defined in LINE command.
options	-	processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE POYNTING command specifies a Poynting flux variable to be plotted vs. time.

The mode specifies the field components to be used in making the measurement. The line_name specifies where the measurement is made. Line_name must be conformal with one spatial coordinate, but it may be positioned anywhere in space. The energy flow is perpendicular to the line (it goes through it). Therefore, you must also enter the sense of the line. If the energy flows against the positive (increasing) coordinate, enter POSITIVE. Otherwise, enter NEGATIVE.

See Also:

OBSERVE [options], Ch. 22

OBSERVE QMEA Command

Function: Specifies charge-density variable to be plotted vs. time.

Syntax:

```
OBSERVE QMEA variable area_name  
[options] ;
```

Arguments:

- variable - charge-density variable (see CONTINUITY, Ch. 18).
= ERRSQR, sum of charge-density error squared.
= RHOSQR, sum of charge-density squared.
= RATIO, ratio of rhoerr/rho.
- area_name - name of area, define in AREA command.
- options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command, OBSERVE QMEA, is used to specify a charge-density variable to be plotted vs. time. The variable specifies the charge-density component. The area_name specifies the spatial area indices over which the measurement is made.

See Also:

CONTINUITY, Ch. 18
OBSERVE [options], Ch. 22

OBSERVE STATISTICS Command

Function: Specifies particle statistics variable to be plotted vs. time.

Syntax:

```
OBSERVE STATISTICS variable species  
[options] ;
```

Arguments:

- variable - particles statistical quantity (see STATISTICS, Ch. 21).
= CREATED, DESTROYED, STORED, or ERROR.
- species - name of particle species.
= ALL, ELECTRON, PROTON, or defined in SPECIES command.
- options - processing options (see OBSERVE [options], Ch. 22).

Description:

The command OBSERVE STATISTICS specifies a particle statistical variable to be plotted vs. time. The variables are CREATED, DESTROYED, STORED and ERROR. Specification of a species restricts the statistical calculation to that species.

See Also:

STATISTICS, Ch. 21
OBSERVE [options], Ch. 22

Restrictions:

A STATISTICS command must be used for this measurement to be made.

OBSERVE STRUT Command

Function: Specifies strut variable to be plotted vs. time.

Syntax: OBSERVE STRUT name variable spatial_object [options] ;

Arguments:

- name - strut name (alpha).
= arbitrary, user-defined in STRUT command.
- variable - strut variable (alpha).
See STRUT command for details,
= CURRENT, CHARGE, INDUCTANCE,
RESISTANCE,
STORED_ENERGY, or OHMIC LOSS.
- spatial_object - name of point or line, defined in POINT or LINE
command.
- options - processing options (see OBSERVE [options]).

Description:

The command, OBSERVE STRUT, is used to specify a strut variable to be plotted vs. time. The argument, name, specifies the strut name as defined in a STRUT command. The argument, asvar, specifies the strut variable to be observed. The spatial_object is used to specify the point or portion of the strut to observe. Spatial_object must lie along the specified strut. If the spatial_object is a point, the variable at that point will be recorded. If the spatial_object is a line, an average or integral along the line will be taken, which ever makes physical sense.

See Also:

STRUT, Ch. 15

OBSERVE [options], Ch. 22

OBSERVE TRAMLINE Command

Function: Specifies a transmission-line variable to be plotted vs. time.

Syntax:

```
OBSERVE TRAMLINE tline_name variable point
                    [options] ;
```

Arguments:

- tline_name - transmission-line name, defined in TRAMLINE command.
- variable - transmission-line variable (see TRAMLINE, Ch. 13).
= VOLTAGE, CURRENT, POWER, CAPACITANCE, INDUCTANCE, IMPEDANCE, ENERGY-CAP, ENERGY-IND, or ENERGY-TOT.
- point-name - name of spatial point on transmission line coordinate, defined in POINT command.
- options - processing options (see OBSERVE [options], Ch. 22).

Description:

The OBSERVE TRAMLINE command is used to specify a transmission-line variable to be plotted vs. time. The tline_name specifies the transmission line and the point_name specifies the coordinate location. name as defined in a TRAMLINE command. The variable specifies the transmission line variable to be observed.

See Also:

TRAMLINE, Ch. 13
OBSERVE [options], Ch. 22

ENERGY Command

Function: Specifies a field and particle energy measurement.

Syntax:
ENERGY timer area_name ;

Arguments:

timer	-	timer name (alpha).
area_name	-	name of spatial area, defined in AREA command.

Description:

The ENERGY command performs an energy calculation, one purpose of which is to see how well the total energy present in the simulation tracks energy sources and sinks. This is basically a test of energy conservation, though the calculation of total energy isolates individual contributions which can be viewed through the OBSERVE command. Also, a summary of these energies can be printed during the simulation at times specified by the DIAGNOSE command.

For electromagnetic fields, both total energy and lost energy are calculated. Only the spatial area specified by area_name indices is included in the calculation. Transmission lines and their match points are not. Total energy is calculated for each component in the electric and magnetic fields and then summed. The variables accessible to the OBSERVE command include ELECTRIC, MAGNETIC, and the individual components, E1, etc. These energies are calculated only at the times specified by the timer, due to the expense of the calculation.

Electromagnetic energy can flow out of the simulation through outer boundaries such as those defined by PORT and OUTGOING commands. The Poynting flux through each such boundary is integrated over time from the beginning of the simulation, resulting in the total energy lost through each boundary. These energies are designed by the arguments, PORT and OUTGOING and by integer arguments representing the boundary number.

For particles, kinetic energies of created particles are accumulated from the beginning of the simulation, as are kinetic energies of destroyed particles. The total kinetic energy of all particles in the simulation is calculated, but only at the times specified by the timer, due to the expense of the calculation. From these energies, the energy gained by particles is accumulated from the beginning of the simulation. Four components of particle energy are designated by the variables, CREATED, DESTROYED, SURVIVING, and GAINED. These energy variables are first calculated separately for different particle species and then summed over species, so information concerning a particular species may be analyzed.

At any given instant, the total energy of particles and fields should balance the energy gained by created particles plus the energy lost through boundaries or destroyed particles. In other words, the residual energy, defined as the sum of these energies, must be zero. Of course, numerical approximations and round off conspire to make it non-zero, so residual energy, relative

ENERGY Command

to total energy, is a measure of simulation integrity. These quantities, total energy, residual energy, and the ratio of residual to total are designated by the arguments, TOTAL, RESIDUAL, and RATIO, respectively.

All variables may be output using the OBSERVE ENERGY command. The following table lists the variables and the physical definition associated with each.

Variable	Physical Definition
CREATED	energy introduced by creation of particles
DESTROYED	energy lost by destruction of particles
SURVIVING	energy of existing particles
GAINED	energy given by fields to particles ($F \cdot v$)
EDOTJ	energy given to fields by particles ($E \cdot J$)
PORT	energy lost through port outer boundaries
OUTGOING	energy lost through outgoing outer boundaries
MATCH	energy lost through transmission line boundaries
BOUNDARY	energy lost through all boundaries
ELECTRIC	energy of electric fields
MAGNETIC	energy of magnetic fields
EM	energy of both electric and magnetic fields
TOTAL	energy of particles and electromagnetic fields
RESIDUAL	energy not accounted for
RATIO	residual-to-total ratio

See Also:

TIMER, Ch. 10

PORT, Ch. 12

OUTGOING, Ch. 12

DIAGNOSE, Ch. 21

OBSERVE ENERGY, Ch. 22

Examples:

In this example, the energy of a klystron resonator is calculated using the ENERGY command and recorded using the OBSERVE command. Notice that the energy algorithm is invoked every other time step, and recorded at the same rate. Figure 22-2 illustrates the growth of the energy in the resonator as a function of time.

```
TIMER FOR-ENERGY PERIODIC INTEGER 0 99999 2 ;
ENERGY FOR-ENERGY LIMITED_AREA ;
OBSERVE ENERGY EM 0 1 INTERVAL 2 ;
```

ENERGY Command

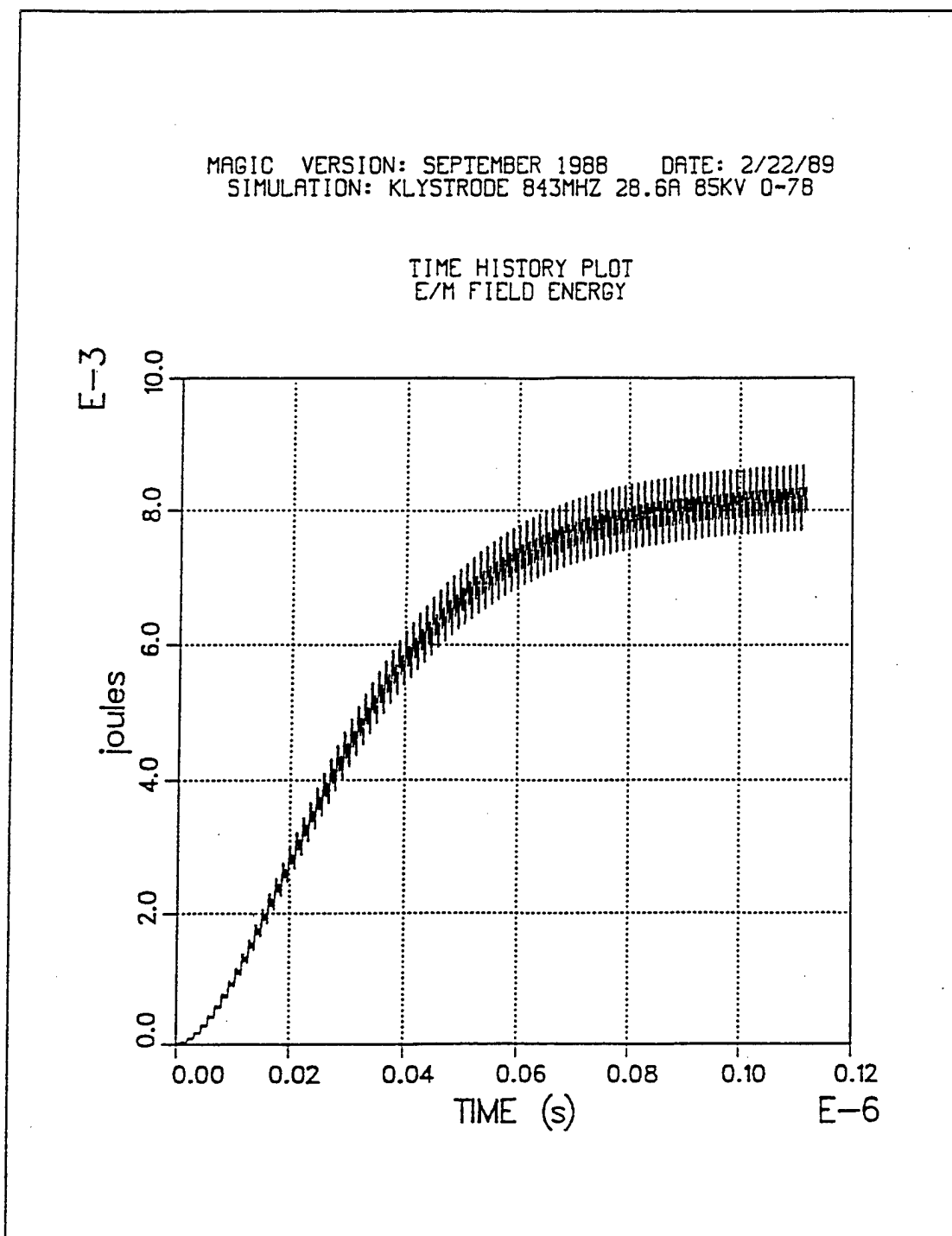


Figure 22-2. Time history of an energy variable.

FLUX Command

Function: Specifies particle flux measurement.

Syntax:

```
FLUX line_name { POSITIVE, NEGATIVE } timer species;
```

Arguments:

line_name	-	name of spatial line, defined in LINE command.
timer	-	timer name, defined in TIMER command.
species	-	name of particle species. = ALL, ELECTRON, PROTON, or defined in SPECIES command.

Description:

The flux command defines a spatial line at which particle flux is measured. The charge, energy, and number of particle crossing the line is accumulated on each time step.

The line_name specifies where the measurement is made. Line_name may be positioned anywhere in space. The particle motion is perpendicular to the line (it goes through it). In general, particles may cross the line going in either direction (e.g., counterflow). However, this measurement is one-way, not net. Therefore, you must also enter the sense of the motion that you wish to measure. If you want to measure particles with positive velocity, enter POSITIVE. Otherwise, enter NEGATIVE. In a single FLUX command, the flux may be due to particles crossing a surface either in the positive direction or the negative direction, but not both.

Plots of flux may be obtained using the OBSERVE and RANGE commands. OBSERVE plots the specified flux variable summed over all flux measurements of the same line_name vs. time. RANGE plots the flux variable vs. distance along line_name. Specifically, the x axis on range plots of flux quantities is a parameter, S, which is the distance from the beginning of the flux surface to each point measured along the curves of the flux surface. The flux variables which may be referenced in other commands, including OBSERVE and RANGE, are given in the following table.

The variables, CURRENT-DENSITY through ENERGY, are averaged over trigger times in timer. The appropriate accumulators for those quantities are zeroed at the beginning of each time interval and the averaging starts again. For the variables CHARGE through ENERGY-DENSITY, the flux is accumulated over the entire simulation, completely independent of the timer.

DUMP output of flux data is possible using DUMP with the FLUX option. The dump output for flux includes the particle momentum and position for each particle crossing each flux boundary. The particles dumped are grouped into DUMP records. Such a record contains information for all particles crossing a surface during a time interval specified by the timer.

FLUX Command

Flux Variables.

Keyword	Quantity	Units
CURRENT-DENSITY	current density	A/m ²
POWER-DENSITY	power density	watts/m ²
MACRO-PARTICLES	macro-particle density	no/sec-m ²
CURRENT	current	A
POWER	power	watts
ENERGY	energy	joules
CHARGE	charge	Coul
PARTICLE-CHARGE	macro-particle charge	Coul
PARTICLE-ENERGY	macro-particle energy	joules
CHARGE-DENSITY	charge density	C/m ²
ENERGY-DENSITY	energy density	joules/m ²

See Also:

OBSERVE FLUX, Ch. 22

RANGE FLUX, Ch. 22

DUMP, Ch. 25

FLUX Command**Examples:**

In this example, the FLUX command is used to measure the power density of an electron beam on a collector beginning after time step 900. The FLUX measurement COLSAT is defined to measure particles crossing the conductor surface COLLECTOR. Flux information is collected beginning with time step 900, and a RANGE plot of the power density on the flux surface is produced at time step 1000. The following commands were used.

```
LINE COLLECTOR Xa,Ya  Xb,Yb  Xc,Yc ;  
TIMER Time_start DISCRETE 18.36e-9 ;  
FLUX COLLECTOR POSITIVE Time_start ALL;  
TIMER Time_end DISCRETE 20.4e-9 ;  
RANGE Time_end 1 FLUX COLLECTOR POWER-DENSITY ;
```

Figure 22-3 shows a plot of the power density versus position along the surface at a time of 20.4 ns.

FLUX Command

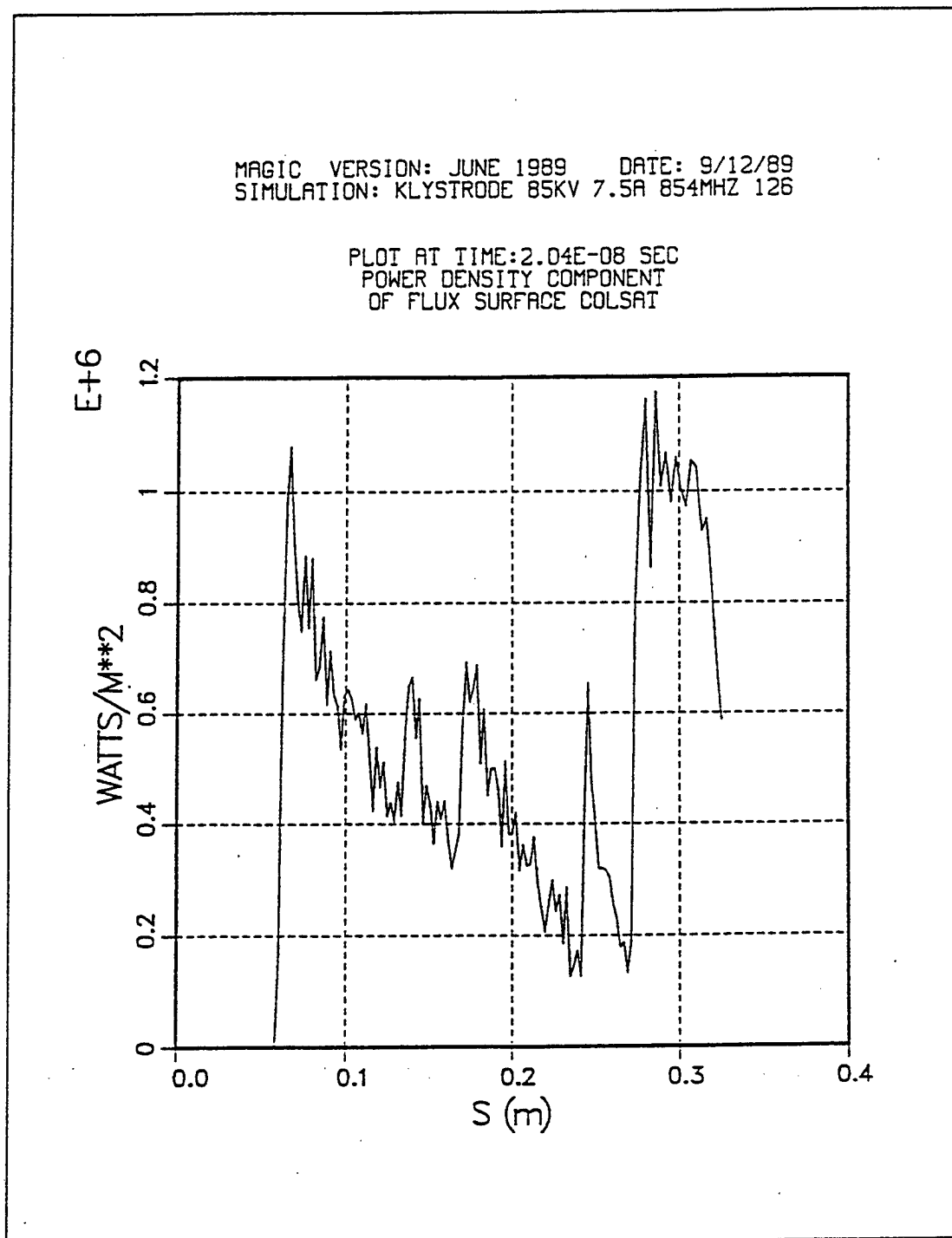


Figure 22-3. Plot of power density of flux surface.

23. 1-D PLOTS

This Chapter covers the following commands:

RANGE [options]
RANGE AIR_CHEMISTRY
RANGE FIELD
RANGE FLUX
RANGE PARTICLE
RANGE POYNTING
RANGE TRAMLINE

You can use the RANGE commands to plot the value of a simulation variable vs. a spatial dimension. Typically, the spatial dimension will coincide with one of the spatial coordinates (x1 or x2) but, in some cases, it may represent an arbitrary path in space (such as the inner surface of a particle collector).

There are many processing options which can be applied to the RANGE data, and these are described by the RANGE [options] command. Defaults are set for many of these options. All of the other RANGE commands involve plotting a particular data_type, such as FIELD, ENERGY, etc.

The ENERGY and FLUX commands described in Ch. 22 calculate energy and particle flux variables, respectively. They do not output results directly; however, the variables which they calculate may be accessed and plotted by both OBSERVE (Ch. 22) and RANGE commands.

This page is intentionally left blank.

RANGE ... [options] Command

Function: Plots simulation variable as a function of spatial position.

Syntax:

RANGE timer

data_type arguments

```
[ AXIS {X,Y} rmin rmax [ rstep ] ]
[ TRANSFORM function(x,y) ]
[ PRETRANSFORM function(x,y,t) ]
[ FFT { NONE, MAGNITUDE, COMPLEX } ]
[ { PLOT, NO PLOT } ]
[ { PRINT, NO PRINT } ]
[ { DUMP, NO DUMP } ] ;
```

Arguments:

timer	- timer name, defined in TIMER command.
data_type	- (see RANGE commands). = AIR_CHEMISTRY, FIELD, FLUX, PARTICLE, POYNTING, or TRAMLINE.
arguments	- different arguments for each data_type, see RANGE commands.
rmin,rmax	- axis limits (real).
rstep	- axis step size (real).
function	- data transform function, defined in FUNCTION command.

Description:

This RANGE command specifies a simulation variables to be plotted vs. a spatial distance at specified times during the simulation. The spatial dimension will normally be a conformal line, but may, in certain circumstances, contain multiple segments (or bends). For example, in a plot of flux data on a surface that has bends, the x axis will be the distance along the surface. The simulation variable may be a field, flux, particle, Poynting, or transmission-line variable. Those features unique to each data_type will be described in separate RANGE commands. However, these different data_types have many process controls in common, as described below.

The process options are common to all RANGE plots. For example, the timer is used to specify the times at which output is to occur.

The AXIS option allows refinement of the actual plot produced by providing control over the axes, both in extent and in spacing. The first argument, either X or Y, specifies the plot axis. The arguments, rmin and rmax, specify the plot extrema, and the argument, rstep, specifies the size of plot subdivisions. The X-axis will normally be a spatial coordinate along a conformal line. When

RANGE ... [options] Command

the range type is FLUX, the X-axis will be the distance along the surface, which may contain multiple segments and bends.

There are mathematical data processing operations which can be performed to modify range values. The keywords are TRANSFORM, PRETRANSFORM, and FFT. Use of one or more of these keywords causes the raw data to be modified in a particular fashion and sequence.

In addition, RANGE can make use of the INTEGRATE option for timers. The TIMER ... INTEGRATE option specifies a temporal averaging interval as the number of time steps prior to the time in which the output occurs. A common use of the INTEGRATE option is to average particle or Poynting data over a full period, to get cycle-averaged quantities.

The TRANSFORM option will be applied to the data after all the temporal averaging, if any, has been performed. It is also possible to apply a transformation to the raw data before it is temporally averaged, using the PRETRANSFORM option. When several of these operations are used simultaneously on the same data, the user should be aware of the particular order in which they are performed relative to each other. These operations are performed in the following first-to-last order:

- 1) PRETRANSFORM,
- 2) Temporal averaging indicated by TIMER ... INTEGRATE,
- 3) TRANSFORM, and
- 4) FFT.

The PRETRANSFORM option permits the user to supply a transformation function to operate on the range values before they are temporally averaged. Normally, this option would only be used if the timer indicated temporal averaging, e.g., TIMER ... INTEGRATE. The function pretrans(x,y,t) uses three arguments. The argument, x, corresponds to the spatial coordinate values. The argument, y, corresponds to the range values. And the argument, t, corresponds to time. A common use of the pretransformation is multiplication by a sinusoidally time varying function to extract harmonic information. If a pretransform is performed, the name of the transformation function is used for the y-axis label.

The TRANSFORM option permits the user to supply a transformation function to operate on the range values. The function transform(x,y) uses two arguments. The argument, x, corresponds to the spatial coordinate values. The argument, y, corresponds to the range values. If a transform is performed, the name of the transformation function is used for the y-axis label, overwriting the name from the pretransformation function, if necessary.

RANGE ... [options] Command

The FFT keyword is used to select the Fourier transform option. The argument `fftplot` specifies the combination of plots desired for any observe variable. A complex FFT is performed, e.g., given range data, $y(x)$, the FFT is given by:

$$\tilde{y}(\lambda) = 2 \int_{x_{\min}}^{x_{\max}} dx y(x) e^{-i2\pi\lambda x} .$$

The user can then select whether to view a single plot of the magnitude of $\tilde{y}$, or two plots showing its complex real and imaginary parts, by choosing either `MAGNITUDE` or `COMPLEX` for `fftplot`. Normally, a plot of the original range data is also produced. To omit the plot of the original data, the user can choose `MAGNITUDE_NODATA` and `COMPLEX_NODATA` for `fftplot`. There is no ability to control the axes of the FFT plots. For reference, the inverse formula of the FFT is:

$$y(x) = \int_0^{\lambda_{\max}} d\lambda \operatorname{Re}\{\tilde{y}(\lambda) e^{i2\pi\lambda x}\} .$$

The normal factor of 2π is missing because of the use of inverse wavelength, λ , instead of wavenumber. The extra factor of two in the transform is compensated by an integration over only positive inverse wavelengths in the inverse transform, a common manipulation where pure real data is concerned.

Range data records can be stored and displayed in three different manners:

- 1) as a plot in a metafile or on screen (e.g., the default behavior),
- 2) as a data record in a DUMP file, or
- 3) as column data in the log file.

A plot of the data in a metafile or on screen is the normal result of a `RANGE` command. If a `DUMP TYPE RANGE` or `DUMP TYPE ALL` command is issued, then the data is also stored as a 1-D data record in the “`grd`” dump file, for later display or postprocessing. Under certain circumstances, it may be desirable to have a particular set of range values displayed on screen, but not stored, in order to save disk space, or perhaps stored but not displayed for some other reason. The `NO PLOT` and `NODUMP` options allow control of data display and storage on a per command basis. Complementary `PLOT` and `DUMP` options are also available, in case the data display and storage is controlled by variable substitution.

It is also possible to generate a columnar listing of the range data in the log file. This option is normally turned off; however, it can be activated with the `PRINT` option, or deactivated with the `NO PRINT` option.

RANGE ... [options] Command

Normally time history data is stored and displayed at the times indicated by the specified timer. On certain platforms, e.g., the PC, it is possible to display and store the current state of the range values at any stage during the simulation. Depressing the F11 key on the keyboard will initiate immediate display (but not dumping) of any range values which are not being time-averaged. (See KEYBOARD, Ch. 9 for further details.)

Restrictions:

The total number of RANGE commands defined in a simulation is limited to fifty.

Restrictions:

The total number of RANGE commands defined in a simulation is limited to ten.

See Also:

TIMER, Ch. 11
TRAMLINE, Ch. 13
FLUX, Ch. 22
RANGE ... AIR_CHEMISTRY, Ch. 23
RANGE ... FIELD, Ch. 23
RANGE ... FLUX, Ch. 23
RANGE ... PARTICLE, Ch. 23
RANGE ... POYNTING, Ch. 23
RANGE ... TRAMLINE, Ch. 23

RANGE...AIR_CHEMISTRY Command

Function: Plots air chemistry variable as a function of spatial position.

Syntax:
RANGE timer AIR_CHEMISTRY variable line [options] ;

Arguments:

variable - air-chemistry variable.
See AIR-CHEMISTRY command for details,
= RHOE, RHON, RHOP, QION, BAVA, BATT,
XMBE, or SIGMAC.
line - conformal line defining spatial coordinate (integers).
options - standard plot options, e.g., AXIS, PRETRANSFORM,
TRANSFORM, FFT, NOPLOT, NODUMP, or PRINT.

Description:

The command RANGE ... AIR_CHEMISTRY instructs the code to plot air chemistry variables versus a spatial coordinate. An AIR_CHEMISTRY command must have been previously issued to use this command.

Restrictions:

The total number of RANGE commands defined in a simulation is limited to fifty.

See Also:

TIMER, Ch. 11
AIR_CHEMISTRY, Ch. 14
RANGE ... [options], Ch. 23

RANGE...FIELD Command

Function: Plots field variable as a function of spatial position.

Syntax:

```
RANGE timer FIELD field line_name [ TRANSVERSE ncells ]  
[ options ] ;
```

Arguments:

field	-	electromagnetic field (E1, ...).
line_name	-	name of conformal line defining spatial coordinate, defined in LINE command.
ncells	-	cells in transverse average of field (integer, default is one).
options	-	standard plot options, e.g., AXIS, PRETRANSFORM, TRANSFORM, FFT, NOPLOT, NODUMP, or PRINT.

Description:

For field data, selected with the FIELD keyword, field must be one of the defined electromagnetic fields. The line_name specifies the coordinate along which the field is plotted; it must be conformal with either the x1 or x2 axis. To smooth the data, the TRANSVERSE option can be used to specify that the field should be averaged over ncells in the direction transverse to the direction of line_name.

Restrictions:

The total number of RANGE commands defined in a simulation is limited to fifty.

See Also:

TIMER, Ch. 11

RANGE ... [options], Ch. 23

Examples:

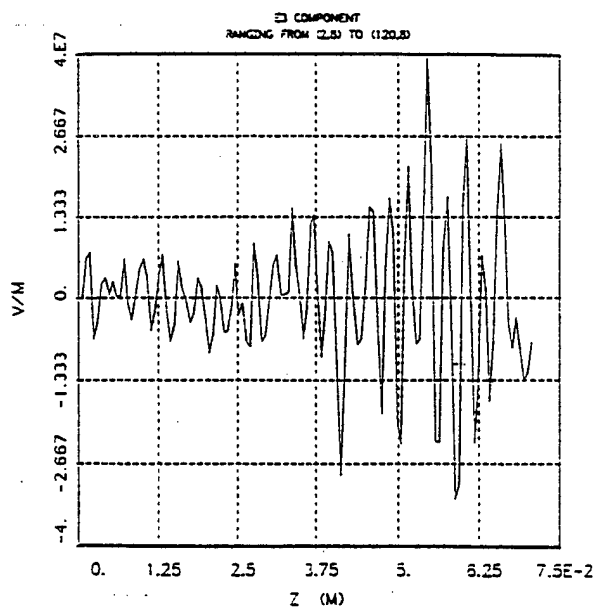
The following commands illustrate the use of various RANGE options in diagnosing a cylindrical CARM simulation. A gyro-magnetic beam with a current of 2.5 kA is introduced at the left end of a hollow, confining chamber. A confining magnetic field of 2.024 tesla is applied along the axial direction. A current-density driver with a frequency of 103.29 GHz is applied at the emission surface. The first RANGE command measures the magnitude of the azimuthal electric field at a fixed value of r versus z, and performs an FFT on the data. Figures 23.1a and 23.1b show the results at a simulation time of 0.29 ns. The next three RANGE commands measure the transverse averaged particle energy, power, and current versus z. A temporal average over one

RANGE...FIELD Command

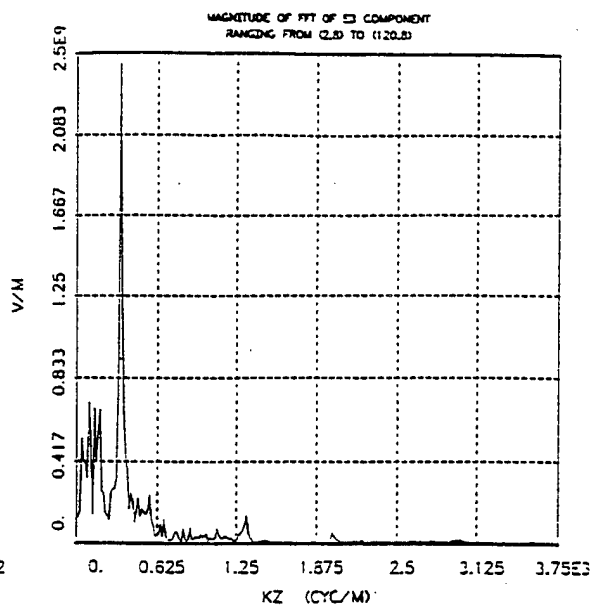
period of the drive frequency is applied by using a **TIMER** with an integration interval equal to the period. Figures 23.1c, 23.1d, and 23.1e show the plots resulting from these commands for a simulation time of 0.29 ns. Notice that the current measurement confirms the supplied current of 2.5 kA. The next **RANGE** command invokes a measurement of the transverse integrated **POYNTING** flux **P1** versus **z**. A temporal average of one RF period was applied to this data as well. Figure 23.1f shows the **POYNTING** flux power versus **z**. Finally, the last two **RANGE** commands draw plots of the particle **FLUX** measured near the end of the simulation. The flux surface is transverse to the beam direction. Figures 23.1g and 23.1h show the time averaged beam power versus **r** profile and the time averaged current-density versus **r** profile, respectively.

```
TIMER REPEAT PERIODIC INTEGER 150 9999 150 ;
TIMER REPEAT_AV PERIODIC INTEGER 150 9999 150
      INTERVAL KPERIOD;
TIMER REPEAT5 PERIODIC INTEGER KPERIOD 9999 KPERIOD ;
FLUX COLLECT REPEAT5 ELECTRON beam_axis POSITIVE ;
RANGE REPEAT 5 FIELD E3 beam_axis 12 ;
RANGE REPEAT_AV 1 PARTICLE ELECTRON X1 MEV ;
RANGE REPEAT_AV 1 PARTICLE ELECTRON X1 POWER ;
RANGE REPEAT_AV 1 PARTICLE ELECTRON X1 CURRENT ;
RANGE REPEAT_AV 1 POYNTING P1 beam_axis TRANSVERSE 6 ;
RANGE REPEAT 1 FLUX COLLECT POWER ;
RANGE REPEAT 1 FLUX COLLECT CURRENT-DENSITY ;
```

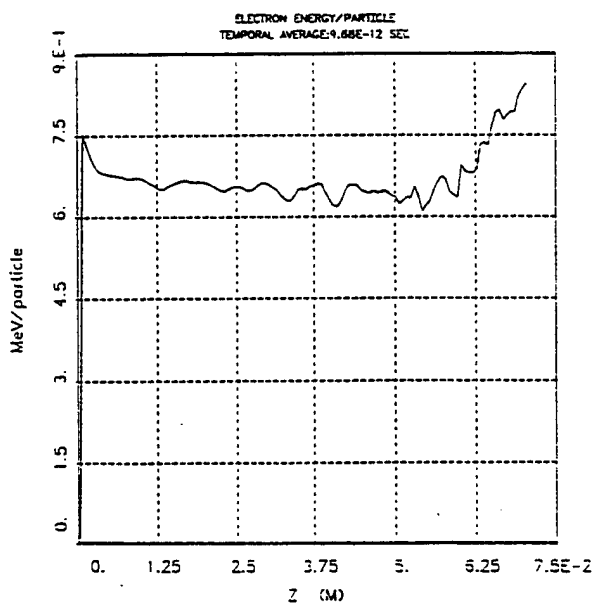
RANGE...FIELD Command



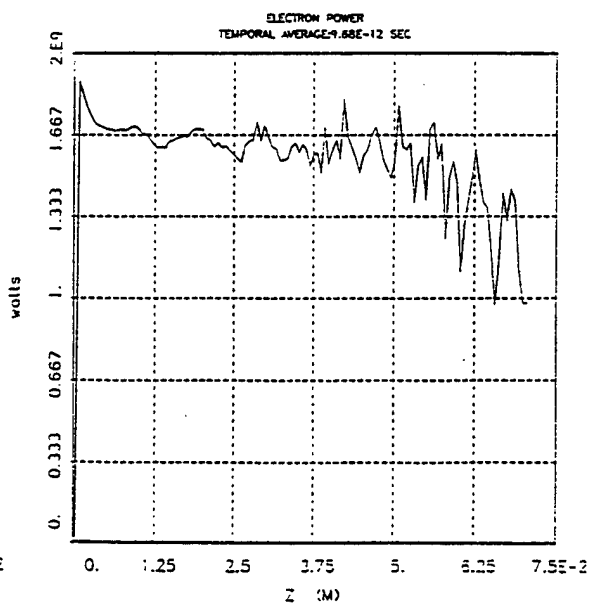
(a) Electric field.



(b) FFT of electric field.



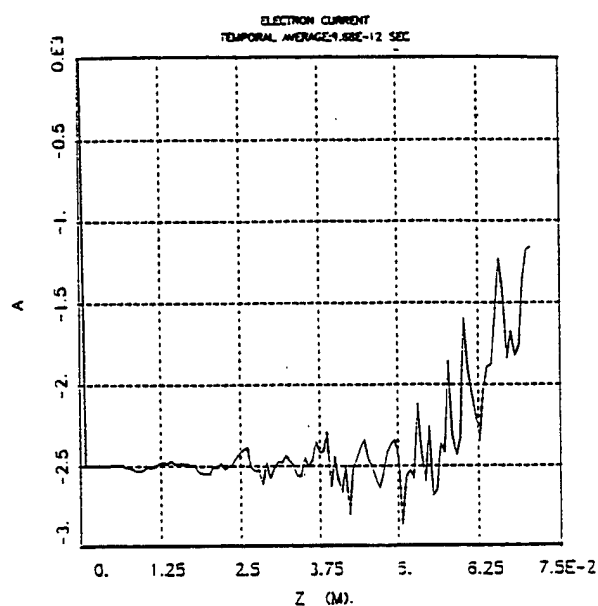
(c) Electron energy/particle.



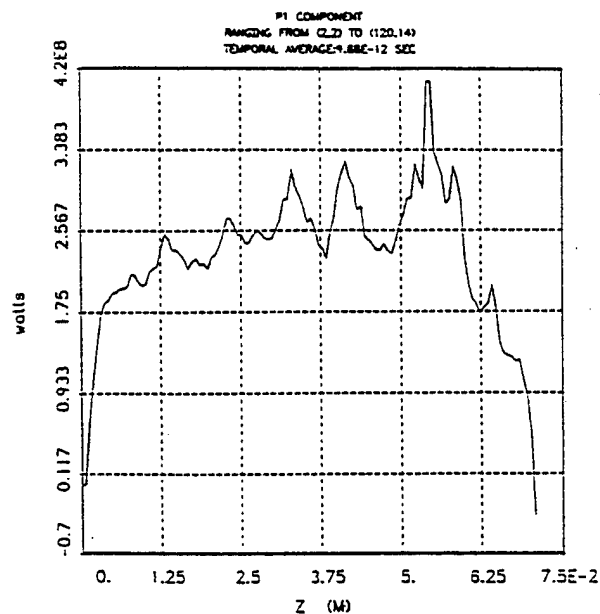
(d) Electron power.

Figure 23-1. Examples of range plots.

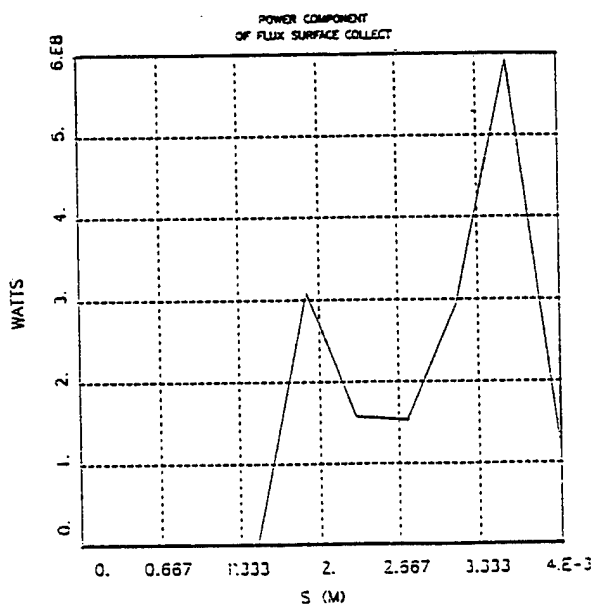
RANGE...FIELD Command



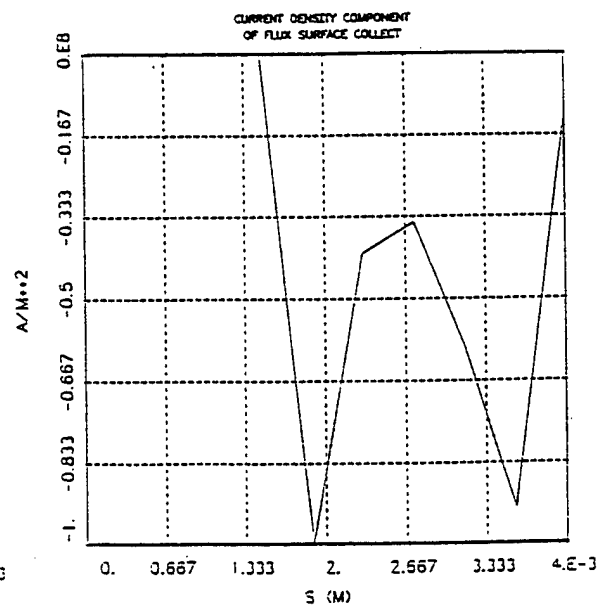
(e) Electron current.



(f) Poynting power flow.



(g) Radial beam power profile at fixed z.



(h) Radial current density at fixed z.

Figure 23-1 . Examples of range plots (continued).

RANGE...FLUX Command

Function: Plots flux variable as a function of spatial position.

Syntax:
RANGE timer FLUX flux_name variable [options] ;

Arguments:

flux	-	flux measurement name (defined in FLUX command).
variable	-	flux variable (see FLUX, Ch. 22).
options	-	standard plot options, e.g., AXIS, PRETRANSFORM, TRANSFORM, FFT, NOPLOT, NODUMP, or PRINT.

Description:

The RANGE ... FLUX command specifies a flux variable to be plotted against spatial position.. The flux_name must be previously defined in a FLUX command. If the surface contains bends (multiple segments) or is curved, then the linear distance will be measured along the surface (See FLUX command for a list of variables).

Restrictions:

The total number of RANGE commands defined in a simulation is limited to fifty.

See Also:

TIMER, Ch. 11

FLUX, Ch. 22

RANGE ... [options], Ch. 23

RANGE...PARTICLE Command

Function: Plots particle variable as a function of spatial position.

Syntax:

```
RANGE timer PARTICLE species { X1, X2 } variable
      [ EFFICIENCY function(x,e) ]
      [ WEIGHTING { FLUX, DENSITY } ]
      [ options ] ;
```

Arguments:

species	-	particle species name. = (SPECIES command for valid choices).
variable	-	particle flux variable (see table). = CURRENT (amps). = POWER (watts). = JOULE, energy/particle in joules. = EV, energy/particle in eV. = KEV, energy/particle in keV. = MEV, energy/particle in MeV.
function	-	function to convert energy to efficiency. = user-defined in FUNCTION command.
options	-	standard plot options, e.g., AXIS, PRETRANSFORM, TRANSFORM, FFT, NOPLOT, NODUMP, or PRINT.

Description:

The command, RANGE ... PARTICLE, measures the propagation of particles along a specified axis. The raw data measurement calculates the current and power distribution-per-unit-length as a function of distance.

The EFFICIENCY option specifies a function which can be used to convert energy/particle to efficiency. The function takes two arguments: x (the axis coordinate) and e (the value of the energy at x). For example, if the beam has 60 keV of energy prior to the output circuit and the collector, the conversion function could be defined as $f(x,e) = 1 - e/60$. If KEV is selected for the energy measurement, the plot will represent the fractional energy loss as a function of position.

The WEIGHTING option can be used to compute energy per particle by two methods. FLUX weighting (the default) derives energy per particle as a ratio of energy flux to power flux, whereas DENSITY weighting derives energy per particle as a ratio of energy density to charge density. The two methods may produce subtle differences when the beam is not steady-state or CW, when it contains counter-flowing particles, or when particle losses are occurring.

RANGE...PARTICLE Command

Restrictions:

The total number of RANGE commands defined in a simulation is limited to ten.

See Also:

TIMER, Ch. 11

SPECIES, Ch. 18

RANGE ... [options], Ch. 23

RANGE...POYNTING Command

Function: Plots Poynting variable as a function of spatial position.

Syntax:

```
RANGE timer POYNTING field line_name
      [ TRANSVERSE_INTEGRAL line_name ]
      [ SPLITTER
        [ FREQUENCY { DISCRETE fmin fmax df,
                      CONTINUOUS fmin fmax df,
                      LIST nf f1 f2 ... fnf } ]
        [ DIRECTIONAL { POSITIVE, NEGATIVE, TOTAL, ALL } ]
        [ PHASEVELOCITY vphase(x,freq) ]
        [ SECTIONS_AT ns x1 x2 ... xns [ SMOOTH ] ]
        [ RECORD_EVERY nsteps ] ]
      [ options ] ;
```

Arguments:

field	- Poynting flux field (watts/ sq. m). = S1, $(E_2 \times B_3 - E_3 \times B_2)$ = S2, $(E_3 \times B_1 - E_1 \times B_3)$ = S3, $(E_1 \times B_2 - E_2 \times B_1)$
line-name	- conformal spatial line, defined in LINE command.
fmin, fmax, df	- splitter frequency window and increment (Hz).
nf	- number of frequencies in list (integer).
f1 f2 ... fnf	- list of frequencies (Hz).
vphase	- phase velocity (m/sec), constant or defined in FUNCTION command.
ns	- number of section breaks for computed phase velocity (integer).
x1 x2 ... xns	- section breaks for computed phase velocity (unitless).
nsteps	- time step interval for field records in the scratch file (integer).
options	- standard range options, e.g., AXIS, PRETRANSFORM, TRANSFORM, FFT, NOPLOT, NODUMP, or PRINT.

Description:

The RANGE ... POYNTING command plots a Poynting field (S1, S2, S3) or transverse area integrals thereof. The line_name specifies the plot's x-axis spatial coordinate and must be conformal with either the x1 or x2 axis.

RANGE ... POYNTING Command

By default, no transverse integral is performed. However, you can use the `TRANSVERSE_INTEGRAL` option to integrate in the transverse coordinate over a path specified by `line_name`. The units of the field resulting from this integration are watts. Transverse integration works only for the S1 and S2 fields.

Additional analysis on the Poynting flux can be performed using the `SPLITTER` option. The `SPLITTER` splits the Poynting flux into its positive and negative directional components and into its different frequency components. It results in a separate range plot for each directional and each frequency component. For example, if `ALL` (`POSITIVE`, `NEGATIVE`, and `TOTAL`) directional components and five frequencies are requested, then 15 (=3x5) range plots will result. Currently, the `SPLITTER` analysis works only for the S1 and S2 fields, and requires transverse integration.

To perform its function properly, the `SPLITTER` requires Poynting data over at least one oscillation period of the smallest frequency of interest. This means that the `TIMER ... INTEGRATE` option should be used, with an integration interval of at least one period. Note that when the `FREQUENCY` option is being used, the smallest frequency of interest is typically the frequency increment, `df`, and not the minimum frequency, `fmin`.

The `SPLITTER` analysis requires either the `FREQUENCY` option, or the `DIRECTIONAL` option, or both.

The `FREQUENCY` option controls the split of the Poynting flux into different frequency components. It is based upon a Parseval's identity applied to the temporal average of the Poynting flux over the specified timer interval. The user must specify the frequencies of interest either as a `DISCRETE` or `CONTINUOUS` frequency window, `fmin` to `fmax`, with a uniform increment, `df`, or as a specific `LIST` of frequencies, in order from smallest to largest. A value of zero for `fmin` or `f1` is permitted.

By default, the temporal behavior of the fields is assumed to be non-repeating, and hence the frequency split is assumed to indicate power within a continuous spectrum. The resultant powers will therefore be displayed as such, e.g., in watts/Hz. This is the case with the `FREQUENCY CONTINUOUS` and `LIST` options. However, for the particular case of CW excitation, the spectrum is actually discrete, consisting of a fundamental CW frequency and its harmonics; and the power in each frequency can be computed absolutely, e.g., in watts, which is far more convenient to work with. The `FREQUENCY DISCRETE` option can be used to analyze a select part of a discrete spectrum. When this is done, it will be assumed that the frequency increment, `df`, is the fundamental CW frequency, and the timer interval will be checked to verify that it is at least $1/df$ in length (an error message will be generated if it is not).

The `DIRECTIONAL` option controls the directional split of the Poynting flux into the `POSITIVE` and `NEGATIVE` directed power. It is performed separately on each frequency component. The split is directionally pure, that is, the positive power is always positive, and the

RANGE ... POYNTING Command

negative power is always negative. However, RANGE will reverse the sign of the negative power, to facilitate reading of the data. The TOTAL, or unsplit, power may be of either sign. Specifying ALL for the DIRECTIONAL option will result in calculations of positive, negative, and total power.

To perform the directional split, an estimate of the phase velocity is required. The internal algorithm used to estimate the phase velocity, which is based upon a minimum standing-wave principal, is of unknown accuracy and robustness. Hence, if the phase velocity of the system can be established a priori, then it is advisable to use the PHASEVELOCITY option to supply this quantity directly. Note that the phase velocity is, in general, a function of frequency and position along the line used to specify the range data.

If the phase velocity is not known a priori, but is expected to vary with position, it is possible to perform the directional split independently in several sections along the range line, with independent estimates of the phase velocity in each section. The SECTIONS_AT option is used to specify the coordinates of the section breaks. The default is one section, e.g., no section breaks. Also by default, the phase velocity is assumed constant in a section. The SMOOTH option indicates that the phase velocity should instead be assumed to be smoothly varying from one section to another.

The SPLITTER analysis requires that the entire time history of the fields be stored to a scratch file on disk before the analysis can be performed, resulting in a fairly sizable scratch file. For this reason, the maximum amount of information in terms of directionality and frequency content should be derived from the fewest number of RANGE ... POYNTING ... SPLITTER commands possible. To decrease the disk load, the fields are not stored every time step; rather, they are stored on an interval determined from the need to resolve the maximum frequency of interest, as specified in the FREQUENCY option, e.g., approximately 30 field records in the shortest period of interest, or 30 records total if the FREQUENCY option is not used. If greater or lesser temporal resolution is called for, the RECORD_EVERY option can be used to directly specify the number of time steps between successive records.

Restrictions:

The total number of RANGE commands defined in a simulation is limited to fifty.

See Also:

RANGE ... [options], Ch. 23
TIMER, Ch. 11

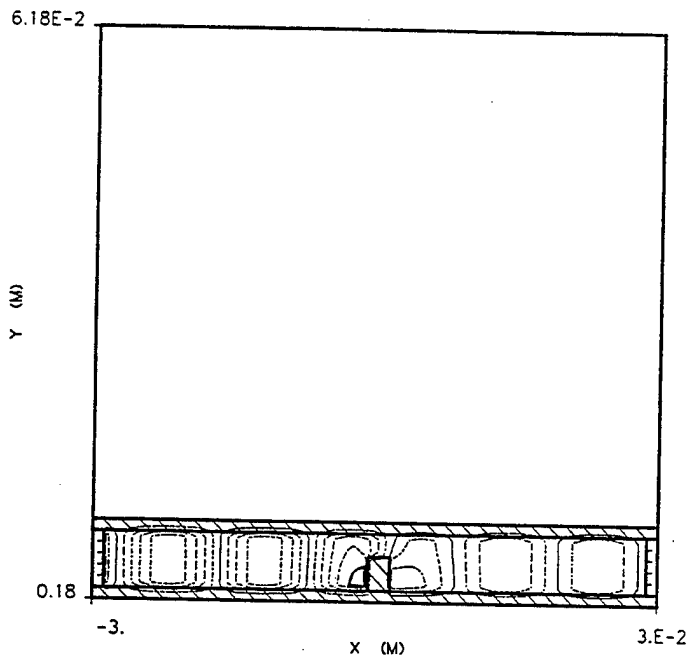
RANGE ... POYNTING Command**Examples:**

In this example, a POYNTING SPLITTER command is used to display the power flow in a parallel plate transmission line which has an obstruction halfway down its length. Figure 23-2a shows the geometry, and contours of B3. A 1.00 watt TEM incident wave enters from the PORT boundary on the left, and propagates toward the obstruction in the center. At the obstruction, approximately 62% of the power is reflected, while the remaining 38% is transmitted, and continues to propagate towards the PORT boundary on the right. The net power, Figure 23-2b, throughout the transmission line is therefore 0.38 watts, however, the splitter diagnostic, Figure 23-2c and d, properly shows that, to the left of the obstruction, the power consists of 1.00 watt forward power and 0.62 watt backward power. It also shows that the power to the right of the obstruction is all forward power.

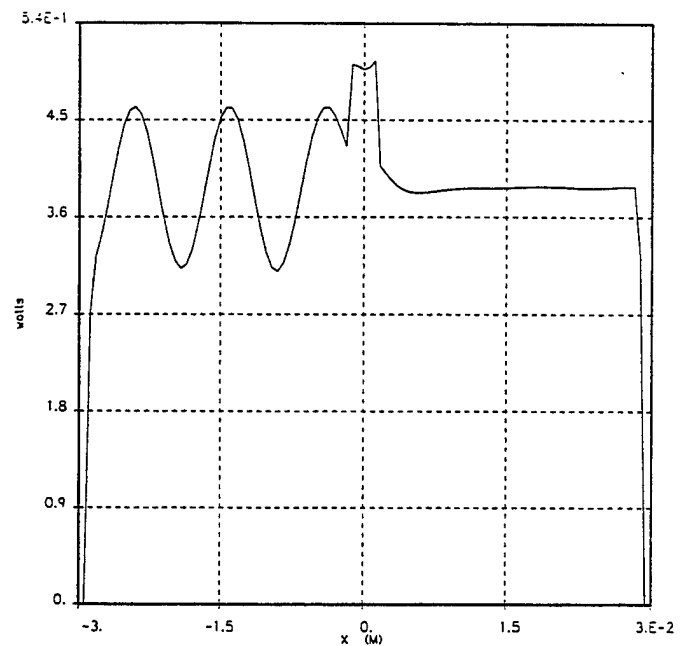
The TIMER command is also shown below because the SPLITTER option requires that the timer's INTEGRATE option be used. Here the time integration is performed over a single oscillation period. Also note that the TRANSVERSE_INTEGRAL option is required for SPLITTER.

```
TIMER CYCLE_SNAP PERIODIC NT,99999,999999 INTEGRATE T_PERIOD ;  
RANGE CYCLE_SNAP POYNTING S1 POYNTING_LINE  
      TRANSVERSE_INTEGRAL INTEGRATE_LINE  
      SPLITTER DIRECTIONAL ALL ;
```

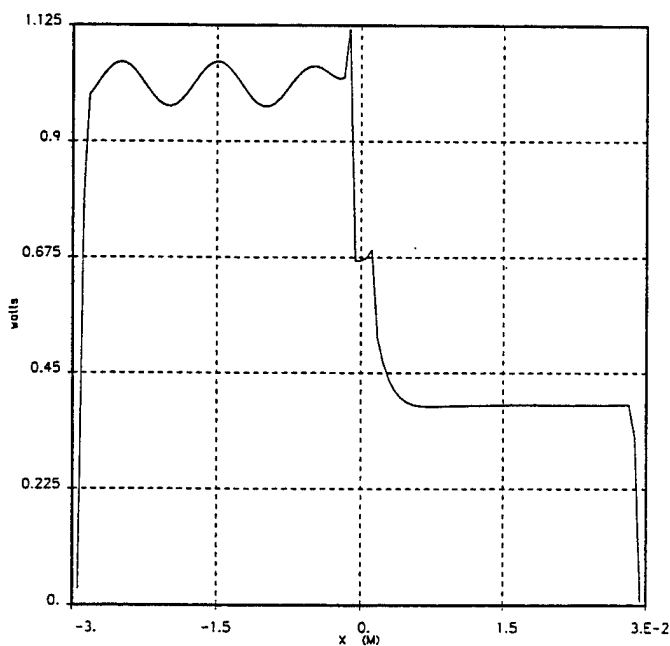
RANGE ... POYNTING Command



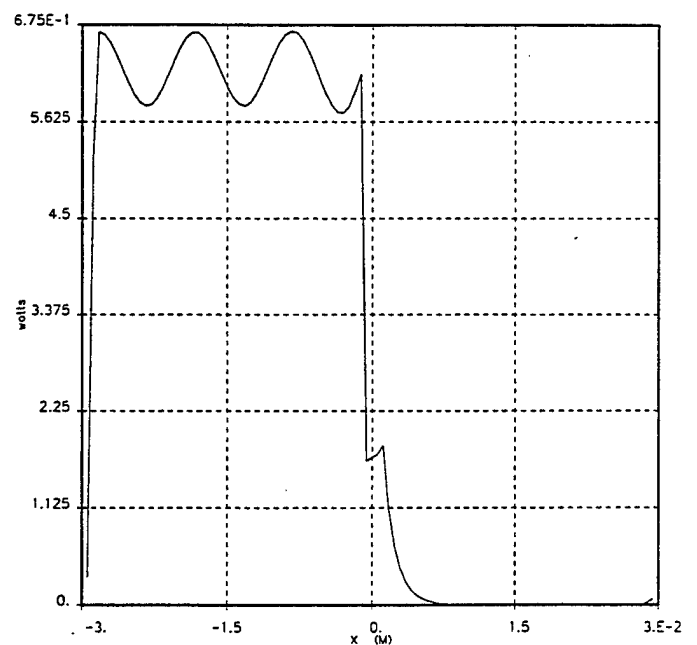
(a) B3 contours in the obstructed Waveguide



(b) Net poynting power



(c) Forward Poynting Power



(d) Backward Poynting Power

Figure 23-2. Example of the Poynting Splitter Diagnostic.

RANGE...TRAMLINE Command

Function: Plots transmission line variable as a function of spatial position.

Syntax: RANGE timer TRAMLINE tline_name variable line_name [options] ;

Arguments:

- tline_name - name of transmission line (defined in TRAMLINE command).
- variable - transmission line variable (see TRAMLINE command).
- line_name - name of spatial line (defined in LINE command).
- options - standard plot options, e.g., AXIS, PRETRANSFORM, TRANSFORM, FFT, NOPLOT, NODUMP, or PRINT.

Description:

The transmission line is specified by tline_name. The variable specifies the particular transmission line variable that can be observed as a function of position, and the range of spatial position is specified by line_name.

(See TRAMLINE command for a list of variables.)

Restrictions:

The total number of RANGE commands defined in a simulation is limited to fifty.

See Also:

TIMER, Ch. 11
TRAMLINE, Ch. 13
RANGE ... [options], Ch. 23

24. 2-D PLOTS

This Chapter covers the following commands:

DISPLAY
CONTOUR
VECTOR
PERSPECTIVE
PHASESPACE
TRAJECTORY
TAGGING

You can use the **DISPLAY** to plot the simulation spatial objects and/or spatial grid. Two-dimensional plots of any field over any two-dimensional portion of the simulation can be output. Such plots can be output as **CONTOUR** plots showing the "equipotentials," or they can be output as **VECTOR** plots representing the direction and size of the field at each location by the direction and size of the plotted vector. Also, three-dimensional **PERSPECTIVE** plots of any field over any two-dimensional portion of the simulation region can be created. Any or all of these plots can be output repeatedly at any set of selected times during the simulation, and for any plot, the same information can be output in the form of printed tables.

For particles, **PHASESPACE** plots of any canonically conjugate momenta can be created. **TRAJECTORY** plots showing the position of particles as a function of time can be created. Plotted particles may be restricted by **TAGGING**. The particle plots can be obtained repeatedly at any set of selected times during the simulation.

This page is intentionally left blank.

DISPLAY Command

Function: Displays selected simulation features.

Syntax:

```

DISPLAY [ area_name ]
        [ OBJECTS ]
          [ POINT ] [ LINE ] [ AREA ]
        [ SPATIAL_GRID ]
          [ MARK ] [ GRID ]
        [ OUTER_BOUNDARIES ]
          [ SYMMETRY ] [ PORT ] [ OUTGOING ]
          [ FREESPACE ] [ MATCH ] [ IMPORT ]
        [ MATERIAL_PROPERTIES ]
          [ AIR-CHEMISTRY ] [ CONDUCTOR ]
          [ CONDUCTANCE ] [ DIELECTRIC ]
        [ UNIQUE_GEOMETRIES ]
          [ FOIL ] [ POLARIZER ] [ SHIM ] [ STRUT ]
        [ OUTPUT ]
          [ RANGE ] [ OBSERVE ]
        [ MAXWELL ]
          [ OUTER_BOUNDARIES ] [ MATERIAL_PROPERTIES ]
          [ UNIQUE_GEOMETRIES ]
        [ PERIMETER ]
          [ CONDUCTOR ] [ OUTER_BOUNDARIES ] ;

```

Arguments:

area_name - name of spatial area, defined in AREA command.

Defaults:

If you simply enter a DISPLAY command with no other data entries, you will automatically display all of the features listed above. The default plot area is the entire simulation.

Description:

The DISPLAY command provides a visual interpretation of simulation features. You can enter any number of DISPLAY commands, and you can vary the content in each plot by selecting different options or by entering the commands at different locations in the command file. (Each DISPLAY command can plot only information available to it from preceding commands, but not from commands which follow it.) The default area is the entire simulation region. As an option, you may specify an area_name to enlarge this area or to zoom in on some selected region.

If you enter a simple DISPLAY command (with or without area_name) with no options, you will automatically display all of the features listed above in the command syntax. If you enter

DISPLAY Command

options, then only the selected features will be displayed. Also, for convenience, some group options contain subsets of other options. For example, if you enter OBJECTS, the plot will display all points, lines, and areas, whereas, if you enter POINT, then only the points will be displayed. These group options and some of the associated plotting conventions are summarized briefly below. (Note that several of the color conventions were unknown at the time of printing.)

The OBJECTS group includes POINT, LINE, and AREA. All spatial objects are plotted as thick, red lines. The + symbol is used for point objects.

The SPATIAL_GRID group includes MARK, with markers plotted as white arrowheads on the coordinate axes, and GRID, with grid lines plotted as thin, white lines (every tenth line is thick).

The OUTER_BOUNDARIES group contains all of the outer boundaries plotted as dashed lines, SYMMETRY in dark blue, PORT in yellow, etc.

The MATERIAL_PROPERTIES group includes all material properties plotted as cross-hatched lines on area objects, CONDUCTOR in gray, DIELECTRIC in green, CONDUCTANCE in purple, etc.

The UNIQUE_GEOMETRIES group includes all unique geometries plotted as thick, solid lines, POLARIZER in green, STRUT in purple, etc.

The OUTPUT group includes output which requires spatial definition, specifically, RANGE and OBSERVE. It can be used to verify the precise locations of measurements.

The MAXWELL group includes three other groups: OUTER_BOUNDARIES, MATERIAL_PROPERTIES, and UNIQUE_GEOMETRIES. The groups contain all of the features which can affect electromagnetic fields and particle kinematics.

The PERIMETER group includes CONDUCTOR and the OUTER_BOUNDARIES group. Visual inspection of this display can verify that the perimeter is contiguous.

Restrictions:

1. There is no restriction on the number of DISPLAY commands.
2. Each plot can include only data from commands which precede the DISPLAY command.
3. The polar (r,θ) or spherical (r,θ) coordinate system results are displayed in cartesian space rather than coordinate space.

DISPLAY Command**Examples:**

1. You can use the command order and/or option selection to specify the content of the DISPLAY plots. For example, the following commands produce two plots, one with just the area object and the second with the area object and the grid together.

```
AREA box ... ;  
DISPLAY ;  
AUTOGRID ... ;  
DISPLAY ;
```

However, these same results could be obtained using option selection, as follows.

```
AREA box ... ;  
AUTOGRID ... ;  
DISPLAY AREA ;  
DISPLAY AREA GRID ;
```

2. To help verify the integrity, you can display only those features which are part of the outer perimeter. Visual inspection can ensure that there are no “holes” in the outer perimeter, which must be contiguous for the simulation to be valid. The command is simply

```
DISPLAY PERIMETER ;
```

3. In the following example, the DISPLAY command is used to plot a Pierce electron gun, consisting of an emitting cathode, a focus electrode, and an accelerating anode. This simulation was done using the spherical coordinate system. In the DISPLAY command, physical cartesian coordinates must be specified. The x-axis limits are given as -0.055 m to 0.01 m. The y-axis limits are given as -0.01 m to 0.055 m. Figure 24-1 illustrates all of the simulation features.

```
AREA Box RECTANGULAR -0.055, 0.01 -0.01, 0.55 ;  
DISPLAY Box ;
```

DISPLAY Command

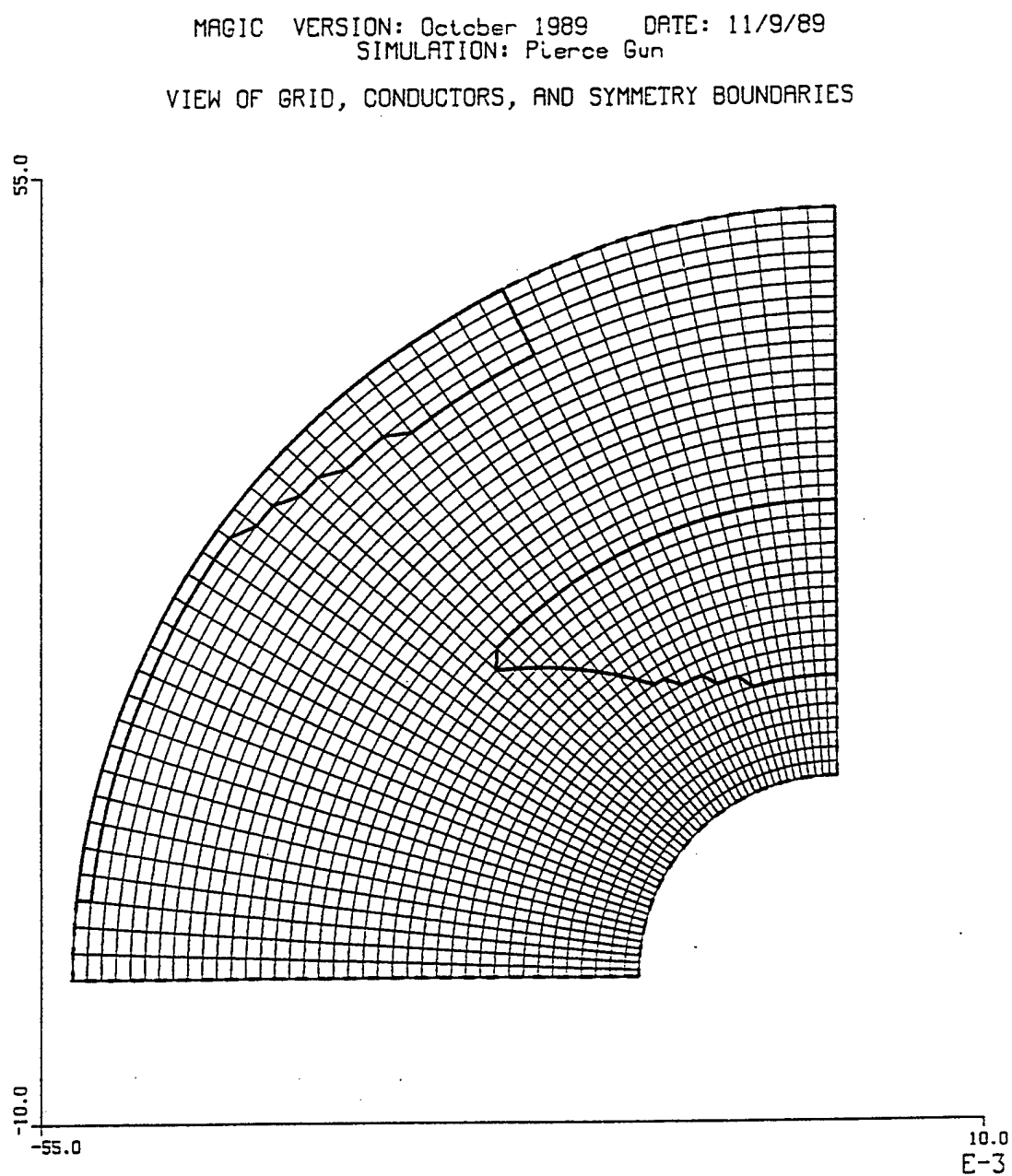


Figure 24-1. Display of Pierce gun.

CONTOUR Command

Function: Specifies contour plots of fields.

Syntax:

```
CONTOUR timer FIELD field area_name
      [ AXIS {X, Y, Z} rmin rmax [ rstep ] ]
      [ BOUNDARY { YES, NO } ]
      [ TRANSFORM f(x,y,z) ]
      [ INTEGRATE { X1ALIGN, X1ANTI, X2ALIGN, X2ANTI } ]
      [ SEPARATION level ]
      [ SYSTEM system ]
      [ YXSCALE yxscale ]
      [ { LEGEND, NOLEGEND } ]
      [ SOLIDFILL ]
      [ { DUMP, NODUMP } ]
      [ { PLOT, NOPLOT } ] ;
```

Arguments:

timer	- timer name, defined in TIMER command.
field	- field (E1, E2, ...).
area_name	- name of spatial area, defined in AREA command.
rmin,rmax	- axis limits (real).
rstep	- axis step size (real).
dlabel	- distance between contour labels (inches or alpha). = user-defined or DEFAULT, program defined.
clabel	- angle of curvature for contour labels (degrees or alpha). = user-defined or DEFAULT, program defined.
clevel	- contour level separation (arbitrary).
system	- coordinate system for output variables (CARTESIAN, CYLINDRICAL, POLAR, or SPHERICAL). = SPHERICAL.
yxscale	- y-to-x scaling factor (unitless).
f	- transform function, defined in FUNCTION command.

Defaults:

The following table lists the default values used for unspecified arguments.

CONTOUR Command

Keyword	Arguments	Default Value
AXIS	rmin,rmax,rstep	code calculated
BOUNDARY	-	YES
SEPARATION	clevel	code calculated
SYSTEM	system	same as simulation
YXSCALE	yxscale	1.
LEGEND	-	on
NOLEGEND	-	off
SOLIDFILL	-	off

Description:

The CONTOUR command instructs the code to plot contour curves of field components at various times during the simulation. Each contour curve consists of all points for which the field component has the same single value; as a result, these curves delineate areas of relatively high or low fields.

Contour plots of the field are made at time steps specified by the timer. The contours are confined to the two-dimensional region specified by area_name.

The keywords AXIS, BOUNDARY, TRANSFORM, INTEGRATE, SEPARATION, SYSTEM, YXSCALE, LEGEND, NOLEGEND, and SOLIDFILL are used to set optional control arguments. These can be entered in any order after the FIELD keyword and arguments are specified.

The keyword, AXIS, is used to establish the plot axes limits. The next argument specifies either X or Y for the spatial coordinates or Z for the field variable. The arguments rmin and rmax specify the axis extrema. The argument rstep specifies the step size used in labeling the plot axis. A value of zero for rstep causes only the endpoints of the axis to be labeled. Not entering a value has the same effect. Default limits for the axes depend on the spatial grid definition. The number of contour levels can be specified by using AXIS Z. The user then enters the maximum and minimum contour levels to be used and the step size between levels. Do not use more than 12 contour levels.

CONTOUR Command

The keyword, **BOUNDARY**, is used to invoke drawing of the simulation boundaries. This includes conductors and symmetries. The default is **YES**.

The **INTEGRATE** option results in contours of the integral of the field in one coordinate direction. The integral may be taken in the positive direction (**ALIGN**) or the negative direction (**ANTI**). This option is useful for illustrating quasi-static potentials when the electric field is integrated, or magnetic field lines which correspond to contours of the magnetic potential.

The **TRANSFORM** option results in contours of an arbitrary function of the field and coordinates, instead of the field itself. The first two function variables are the coordinates, and the third variable is the field.

The keyword, **SEPARATION**, can be used to set **clevel**, the difference between contour levels. In this case, the code determines the extrema and the number of levels to draw. If contour levels are not specified, the code will automatically determine the extrema and find ten levels.

The keyword, **SYSTEM**, is used to select an output coordinate system that is different than the simulation coordinate system. Specifically, the keyword **SYSTEM** is used when it is desired to convert polar coordinates (r,θ) or spherical coordinates (r,θ) to a cartesian coordinate system for contour plots. If not used, the contour plot uses r on the x-axis and θ on the y-axis for these coordinate systems.

The keyword, **YXSCALE**, allows the user to adjust **yxscale**, the y-to-x axis scaling factor. The default value for **yxscale** is unity. This option is useful when the aspect ratio of the simulation model is very large. Then the small dimension can be enlarged to improve visualization.

The keywords, **LEGEND** and **NOLEGEND**, respectively create or delete the axis labels. The keyword, **SOLIDFILL**, provides shading between contour lines.

Restrictions:

The total number of **CONTOUR** commands in a simulation is limited to twenty.

See Also:

TIMER, Ch. 11
GRAPHICS, Ch. 20

CONTOUR Command**Examples:**

Consider the case of a hollow cylindrical waveguide. The waveguide is excited from the left in a TE_{10} mode. The following CONTOUR command is used to create a (z,r) cross-sectional view of contours of E_θ , the azimuthal electric field. Notice in Figure 24-2 that positive valued contours are drawn with solid lines, and that negative valued contours are drawn with dotted lines.

```
TIMER CTIME PERIODIC INTEGER 200 99999 200 ;
CONTOUR CTIME
      FIELD E3  2 102 2 30
      AXIS Z -12 12 2
      AXIS X 0.0 70E-3 10E-3
      AXIS Y 0.0 14E-3 7E-3 ;
```


CONTOUR Command

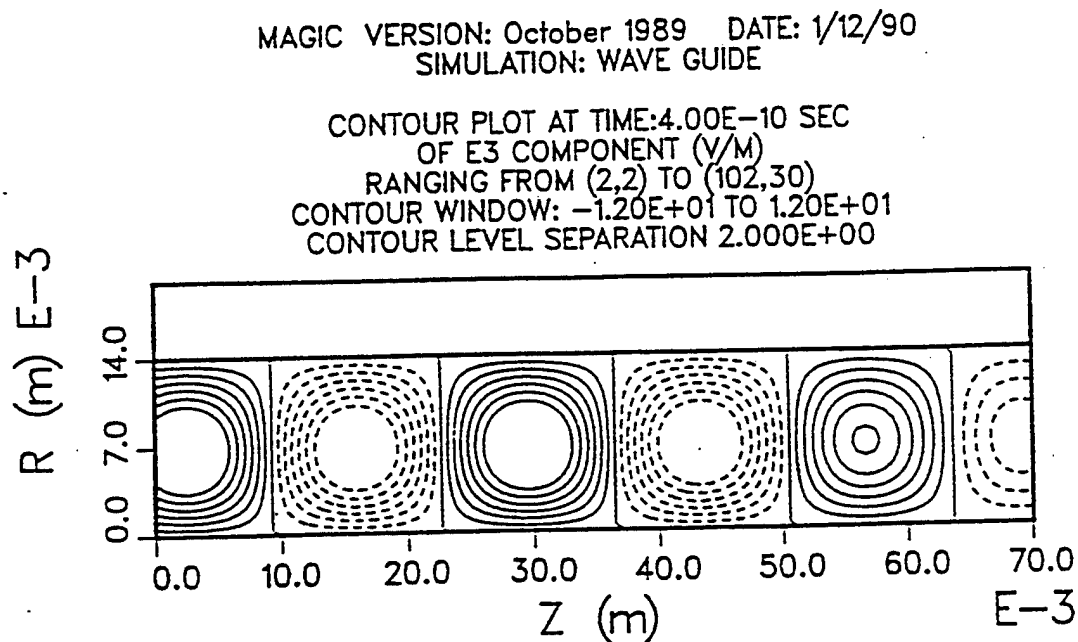


Figure 24-2. Contours of the azimuthal electric field at time 0.4 ns.

VECTOR Command

Function: Specifies vector plots of fields.

Syntax:

```
VECTOR timer FIELD fieldx fieldy
  [ BOUNDARY { YES, NO } ]
  [ AXIS { X, Y } rmin rmax [ rstep ] ]
  [ NORMALIZATION normalization ]
  [ NUMBER ivecx ivecyl ]
  [ SCALE { LINEAR, LOGARITHMIC } [ idecade ] ]
  [ WINDOW { x1, x2 } wmin wmax ] ;
```

Arguments:

timer	-	timer name, defined in TIMER command.
fieldx, afield	-	x and y field components (E1, E2, ...).
rmin, rmax	-	axis limits (real).
rstep	-	axis step size (real).
normalization	-	vector magnitude normalization factor (unitless).
ivecxl	-	number of vectors along horizontal axis (integer).
ivecyl	-	number of vectors along vertical axis (integer).
idecade	-	number of decades in logarithmic scaling (integer).
wmin, wmax	-	window variable extrema (real).

Defaults:

Keyword	Argument	Default Value
BOUNDARY	abound	YES
AXIS	rstep	0.0
NORMALIZATION	vnorm	code-calculated
NUMBER	ivecxl, ivecyl	30, 30
SCALE	ascale	LINEAR
WINDOW	wmin, wmax	spatial grid extrema

Description:

The VECTOR command instructs the code to plot vectors (with arrowheads) showing the magnitude and direction of a field as a function of position in space.

VECTOR Command

Vector plots are made at time steps specified through the timer. Two field components, `fieldx` and `fieldy`, provide arguments for the horizontal and vertical components of the plotted vectors; they should correspond to the `x1`- and `x2`-components of a vector field. Vectors are always converted to a cartesian frame prior to drawing the plot.

The keywords `BOUNDARY`, `AXIS`, `NORMALIZATION`, `NUMBER`, `SCALE`, and `WINDOW` are used to set optional parameters. These can be entered in any order after the `FIELD` keyword and parameters are specified.

The keyword, `BOUNDARY`, is used to invoke drawing of the simulation boundaries. This includes conductors and symmetries.

The keyword, `AXIS`, is used to establish the plot axes limits. Vector plots are output as vectors in a cartesian coordinate system. Locally orthogonal non-cartesian coordinates, such as spherical coordinates, are converted to a cartesian grid. Values are entered in meters. The arguments, `rmin` and `rmax`, specify the axis extrema. The argument, `rstep`, specifies the step size used in labeling the plot axis. A value of zero for `rstep` causes only the endpoints of the axis to be labeled. Not entering a value has the same effect. Default limits for the axes depend on the grid definition and the coordinate system. The default limits correspond to the entire simulation in both coordinates. Use of the default is not recommended since the aspect ratio of the vectors will vary with the grid aspect ratio. The keyword, `NORMALIZATION`, is used to impose a specific normalization on the vector field magnitudes. All vectors will be scaled by this quantity.

The keyword, `NUMBER`, is used to select the density of vectors on the plot in each axial direction. The argument, `ivecx`, is the number of vectors to be drawn along the plot `x`-axis, and `ivecy` is the number to be drawn along the plot `y`-axis. The region of interest is divided into `ivecx` times `ivecy` uniformly sized cells. A vector is plotted in the center of each of these cells using values interpolated from the field components. The keyword, `SCALE`, is used to select the scaling of the vectors. The vectors are scaled linearly or logarithmically. When scaled linearly, the length of the vector is proportional to the magnitude of the field at that point. When scaled logarithmically, the length depends on the logarithm of the field magnitude and the number of decades, `idecade`. The maximum magnitude produces the longest vector, while a magnitude that is below the minimum produces a vector of length zero. Particularly useful choices for `ascale` and `idecade` are `LOG` and `100`. Since most field magnitudes are only a few decades below maximum, all plotted vectors will have nearly the same maximum length, producing a vector plot displaying only direction. With either type of scaling, the horizontal and vertical components are scaled identically if the region displayed has an aspect ratio of unity, (i.e. the region is square). Otherwise, the horizontal and vertical scaling factors will have a ratio equal to the aspect ratio; that is, if the region appears compressed, the vectors will be equally compressed.

The keyword, `WINDOW`, is used to further limit the two-dimensional region for which the vectors are to be constructed. The windows correspond to limits in the coordinate system of the simulation. Thus for spherical coordinates, the `WINDOW X2` is entered in units of radians rather

VECTOR Command

than meters as is the WINDOW X1. The argument, avar, specifies the coordinate system variable, and the arguments, wmin and wmax, specify the window extrema. Only vectors inside the window extrema are plotted. When no window limits are imposed, the code uses the spatial grid extrema.

Restrictions:

1. The total number of VECTOR commands in a simulation is limited to twenty.
2. Fields in polar (r,θ) and spherical (r,θ) coordinates do not show vectors with a polar mapping, but instead are plotted with cartesian coordinates.
3. AXIS is required input for cylindrical (r,θ) and spherical (r,θ) coordinate systems.

See Also:

SYSTEM, Ch. 10
TIMER, Ch. 11
GRAPHIC, Ch. 20

Examples:

The following example is taken from a simulation of the Aurora diode. In this case, a vector plot of the current density was requested every 250 time steps. The region of interest was specified to be 0.0 to 0.75 m along the axis of symmetry and from each direction is selected to be 20, with linear scaling of the current density. Figure 24-3 shows the vector field at a time of 15 ns.

```
TIMER FOR-VTR    PERIODIC    INTEGER 250 999999 250 ;  
  VECTOR FOR-VTR  FIELD J1 J2  
    NUMBER 20 20  
    SCALE LINEAR  
    AXIS X 0.0 .75 .125  
    AXIS Y 0.0 .75 .125 ;
```

VECTOR Command

MAGIC VERSION: MARCH 1987 DATE: 8/4/88
SIMULATION: AURORA DIODE - SHORT PULSE - RUN 7

VECTOR PLOT OF (J1,J2)
AT TIME: 1.50E-08 SEC
VMAX = 2.11E+06

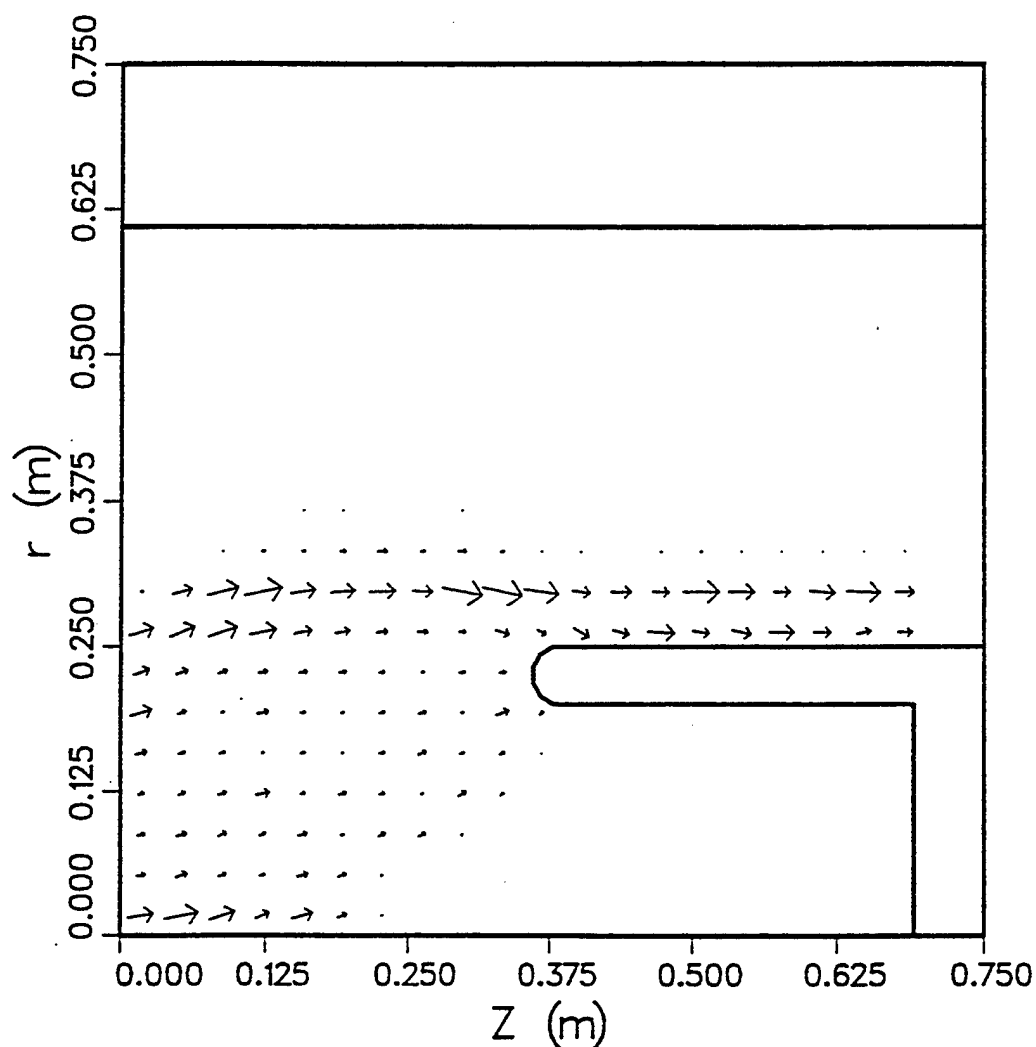


Figure 24-3. Current density in the Aurora diode.

PERSPECTIVE Command

Function: Specifies perspective plots of field components.

Syntax:

```
PERSPECTIVE timer FIELD field area_name i1step i2step
      [ AXIS {X, Y, Z} rmin rmax [ rstep ] ]
      [ EYEANGLE distance elevation heading scale ]
      [ WORKBOX xsize ysize zsize ]
      [ INTEGRATE {X1ALIGN, X1ANTI, X2ALIGN, X2ANTI} ]
      [ {DUMP, NODUMP} ]
      [ {PLOT, NOPLOT} ] ;
```

Arguments:

timer	- timer name, defined in TIMER command.
field	- field component (E1, E2,...).
area_name	- name of spatial area, defined in AREA command.
i1step	- i_1 plotting step size (integer).
i2step	- i_2 plotting step size (integer).
rmin,rmax	- axis limits (real).
rstep	- axis step size (real).
distance	- distance scaling factor (unitless).
elevation	- eyepoint elevation, polar angle theta (deg).
heading	- eyepoint heading, azimuthal angle phi (deg).
scale	- y-to-x scaling factor (unitless).
xsize	- x dimension of workbox (inches).
ysize	- y dimension of workbox (inches).
zsize	- z dimension of workbox (inches).

Defaults:

Keyword	Argument	Default Value
EYEANGLE	distance	1.0
	elevation	0.0
	heading	0.0
	scale	1.0
WORKBOX	xsize,ysize,zsize	code-calculated

PERSPECTIVE Command

Description:

The PERSPECTIVE command instructs the code to plot field components during the simulation. Each plot shows the field as a surface, the height of which corresponds to the value of the field as a function of position in the spatial grid. The entire surface is displayed with three-dimensional perspective. Thus, at a glance, one can see the value of a field component everywhere in the spatial grid.

Plots are made at time steps specified by the time sequence specified by the timer. Each plot displays the values of the field within the region of the spatial grid specified by the `area_name`. The plot shows each value as a point in space connected by lines to neighboring points. These lines make up a network of grid lines which appear as a surface in three-dimensional space. However, if the spatial grid is very fine, these grid lines will be quite dense, and the surface will appear black and formless. In this case, the grid step sizes, `i1step` and `i2step`, should be made larger than one; this will reduce the number of points used to construct the surface. The keywords `AXIS`, `EYEANGLE`, and `WORKBOX` are used to set optional parameters. These can be entered in any order after the other parameters are specified.

The keyword, `AXIS`, is used to establish the plot axes limits. The arguments, `rmin` and `rmax`, specify the axis extrema. The argument, `rstep`, specifies the step size used in labeling the plot axis. A value of zero for `rstep` causes only the endpoints of the axis to be labeled. Not entering a value has the same effect. Default limits for the axes depend on the spatial grid definition for the X and Y axes. The Z axis limits are obtained by finding the extrema of the selected field component.

The keyword, `EYEANGLE`, is used to select the eyepoint from which the plot origin is viewed. This sets the perspective orientation of the plot. The default condition is a perspective plot drawn with an elevation of 45 degrees and a heading of 45 degrees. The parameters entered with `EYEANGLE` are relative to these values. The `EYEANGLE` parameters are: distance, which is a distance scaling factor; elevation, which is the polar angle theta; heading, which is the azimuthal angle phi; and scale, which is the y-to-x axis scaling factor. The default values are 1, 0, 0, and 1. A distance factor of greater than unity decreases the perspective of the view. These definitions are illustrated in Figure 24-4.

The keyword, `WORKBOX`, is used to establish the size of three-dimensional workbox volume. The default volume is automatically calculated by the code. The workbox volume affects the number of tick marks applied to the axes. The entire volume is scaled to fit the plot area. Therefore, entering values of 6 6 3 for the workbox and 4 4 2 will produce plots identical, except for the number of axes tick marks.

PERSPECTIVE Command**Restrictions:**

1. The total number of PERSPECTIVE commands in a simulation is limited to twenty.
2. Plots in cylindrical (r,θ) and spherical (r,θ) coordinates do not show the spatial grid with a polar mapping, but instead with the same rectangular mapping used with cartesian coordinates.

PERSPECTIVE Command

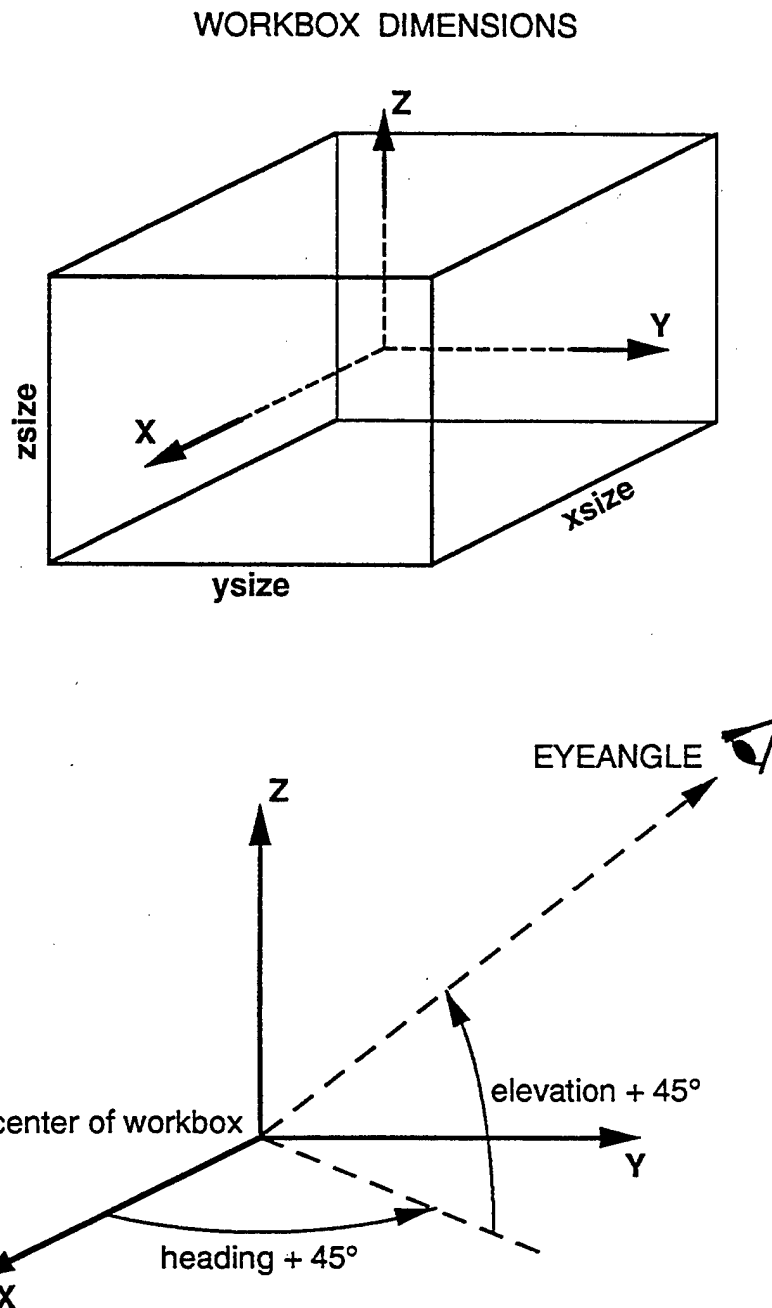


Figure 24-4. Definitions for perspective plot.

PERSPECTIVE Command**Examples:**

1. The following command creates plots at periodic intervals of 100 time steps, beginning at time step 100 and continuing until time step 1000. The perspective plots of the E1 field include data along the X1 coordinate from indices 2 to 40 in steps of 2, and along the X2 coordinate from indices 2 to 60 in steps of 3. Default values are used for the workbook and EYEANGLE. The code calculates the axis limits from the region of the spatial grid included and the extrema of the field components.

```
TIMER FORPER PERIODIC INTEGER 100 1000 100 ;
PERSPECTIVE FORPER FIELD E1 PLOT_AREA 2 3 ;
```

2. In the following command, one plot is created at time step 200 of the J1 field component. The X1 indices range from 20 to 40, and the X2 indices range from 30 to 50. The workbook is set to 8 inches by 6 inches by 3 inches. The EYEANGLE is set at a distance of 1, the default elevation of 45 degrees is used, the heading is rotated 90 degrees clockwise from the default value, and the Y-to-X axis scaling ratio is set to 1. The X-axis limits are set to .0 and .4 m with a step size of .1 m. The Y-axis limits are set to .0 and .3 m with a step size of .1 m. The Z-axis limits will be calculated from the field values.

```
TIMER ONCE DISCRETE INTEGER 200 ;
PERSPECTIVE ONCE
    FIELD J1 plot_area 1 1
    WORKBOX 8. 6. 3.
    AXIS Y .0 .3 .1
    AXIS X .0 .4 .1
    EYEANGLE 1. 0. 90. 1. ;
```

3. Consider the case of a hollow cylindrical waveguide. The waveguide is excited from the left in a TE_{10} mode. The three-dimensional perspective plot of E_θ , the azimuthal electric field in the waveguide shown in Figure 24-5, was created using the following commands. The simulation employed the cylinder (z,r) coordinate system.

```
TIMER CTIME PERIODIC INTEGER 200 99999 200 ;
PERSPECTIVE CTIME
    FIELD E3 PLOT_AREA 1 1
    WORKBOX 7 1.4 1.4
    AXIS Z -12 12 12
    AXIS X 0.0 70E-3 10E-3
    AXIS Y 0.0 14E-3 7E-3 ;
```

PERSPECTIVE Command

MAGIC VERSION: October 1989 DATE: 1/12/90
SIMULATION: WAVE GUIDE

PERSPECTIVE PLOT AT TIME: 4.00E-10 SEC
OF E3 COMPONENT
RANGING OVER (2,2) TO (102,30)

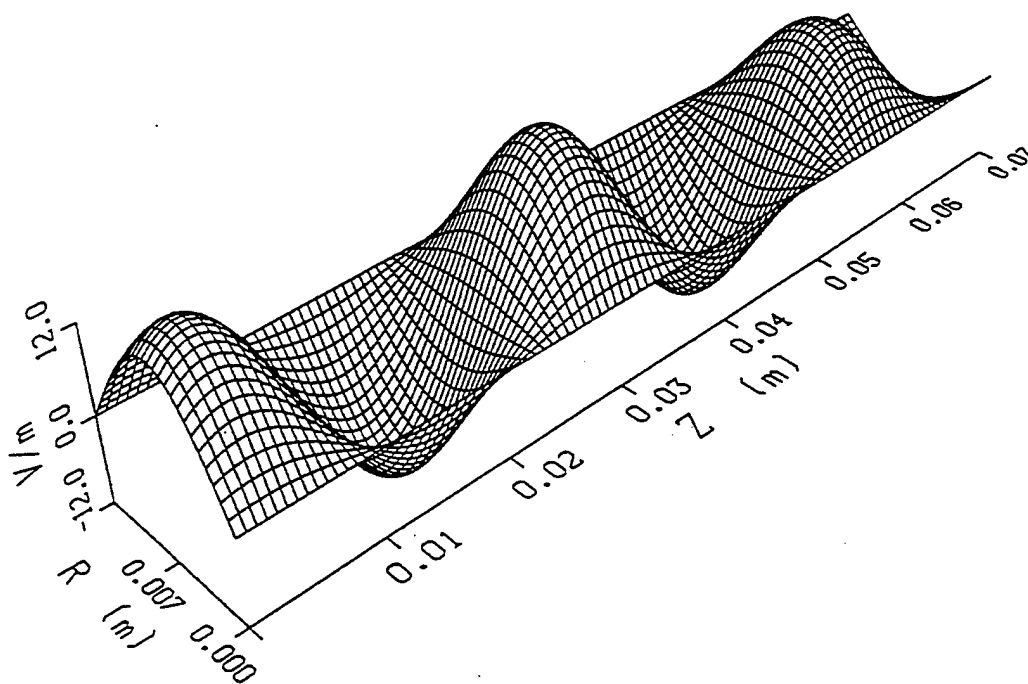


Figure 24-5. Perspective plot of the azimuthal electric field at time 0.4 ns.

PHASESPACE Command

Function: Plots particle phase space.

Syntax:

```
PHASESPACE timer AXES xaxis yaxis
  [ AXIS { X, Y } rmin rmax [ rstep ] ]
  [ WINDOW variable wmin wmax ]
  [ SPECIES species ]
  [ SELECT tag ]
  [ SYSTEM system ]
  [ BOUNDARY { YES, NO } ]
  [ { PLOT, NOPLOT } ]
  [ { DUMP, NODUMP } ] ;
```

Arguments:

timer	- timer name, defined in TIMER command.
xaxis	- x-axis variable. = X1, X2, P1, P2, P3, Q, EN, GA, or = afctx(x1,x2,p1,p2,p3,q,en), a user-defined function.
yaxis	- y-axis variable. = X1, X2, P1, P2, P3, Q, EN, GA, or = afcty(x1,x2,p1,p2,p3,q,en), a user-defined function.
rmin,rmax	- axis extrema (real).
rstep	- axis step size (real).
avar	- window variable (X1, X2, P1, P2, or P3).
wmin,wmax	- window variable extrema (real).
species	- particle species (alpha or integer). = ALL, ELECTRON, PROTON, or = user-defined in SPECIES command.
tag	- selects particle tagging for output. = ALL, all particles. = TAG, only tagged particles.
system	- coordinate system for output variables (CARTESIAN, CYLINDRICAL, POLAR, or SPHERICAL).

Defaults:

The default values for the optional arguments are listed.

PHASESPACE Command

Keyword	Argument	Default Value
AXIS	rmin, rmax, rstep	(see Description, below)
WINDOW	variable	(infinite window)
SPECIES	species	ALL
SELECT	tag	ALL
SYSTEM	system	(simulation coordinate system)
BOUNDARY		YES

Description:

The PHASESPACE command produces plots of particle phase space at times specified by the timer. The timer must specify times which take into account the kinematics step_multiple (LORENTZ, Ch. 18).

The xaxis and yaxis specify the phase-space variables. The standard variables that may be specified are X1, X2, P1, P2, or P3. These are the physical coordinates and momenta of the particles. (Figure 24-6 presents a classic phase-space result.) In addition, the arguments, Q, EN, and GA may be used to select the charge per macroparticle (coulombs), kinetic energy per physical particle (eV), and relativistic factor (unitless). Other variables can be selected for the axes by using a function name for the arguments, axaxis and ayaxis. The function implicitly refers to seven particle variables: x1, x2, p1, p2, p3, q, and en. The variables x1 and x2 denote the physical coordinates of the particle. The variables p1, p2, and p3 denote the momenta of the particle in m/s. The variable q denotes the particle charge and the variable en denotes the kinetic energy in eV.

The keywords, AXIS, WINDOW, SPECIES, SELECT, SYSTEM, and BOUNDARY, are used to set optional parameters. These can be entered in any order after the AXES keyword and parameters are specified.

The keyword, AXIS, is used to establish the plot axes limits. The arguments, rmin and rmax, specify the axis extrema. The argument, rstep, specifies the step size used in labeling the plot axis. A value of zero for rstep causes only the endpoints of the axis to be labeled. Not entering a value has the same effect.

The keyword, WINDOW, can be used to specify an acceptance window in any of these variables: x_1 , x_2 , p_1 , p_2 and p_3 . Only particles falling within the variable limits, wmin and wmax, will be plotted. Note that the window variable does not have to be one of the plotted variables; it simply offers a mechanism to discriminate plotted particles.

The keyword, SPECIES, is used to select the particle species to be included in the plot. The argument, ALL, will produce a plot with all species on one plot. Plots of individual

PHASESPACE Command

species may be created by specifying the particular species name, e.g., ELECTRON, PROTON, etc. The default for species is ALL.

The keyword, SYSTEM, is used to select an output coordinate system that is different than the simulation coordinate system. This causes conversion of the phase-space variables to the output coordinate system. The conversion is done prior to application of the phase-space window. The default is the same coordinate system on input and output.

The keyword, SELECT, is used to select tagged particles for the phase-space output. The argument, ALL, selects all the particles and the argument, TAG, selects only the tagged particles. The default is ALL.

The keyword, BOUNDARY, is used to invoke drawing of the simulation boundaries, including conductors and symmetries. This option is available only when the x- and y-axis variables are X1 and X2. The default is YES.

Restrictions:

1. The total number of PHASESPACE commands is limited to 20.
2. The BOUNDARY option is available only for X1 and X2.

See Also:

SYSTEM, Ch. 10
TIMER, Ch. 11
SPECIES, Ch. 18
LORENTZ, Ch. 18
TAGGING, Ch. 24

Examples:

1. The following commands produce phase-space plots at time step 1500 and every 500 time steps afterward. We request phase-space plots of the particle positions, x1 and x2, and of the momentum components, p1 and p2. The plot axial limits are established with the keywords AXIS X or AXIS Y. For the particle positions, we use the limits 0.0 to .25 m along the x-axis, and the limits .0675 to 0.0925 m along the y-axis. For the momentum-space plot, the axial limits are minus and plus a third the speed of light. No windowing of the particles is requested, and therefore all particles will be shown that fit onto the plot area. The commands are:

```
TIMER FOR-CNTR PERIODIC INTEGER 1500 99999 500 ;  
PHASESPACE FOR-CNTR  
    AXES  X1 X2
```

PHASESPACE Command

```
AXIS X 0.0      0.25
AXIS Y 0.0675 0.0925 ;
PHASESPACE FOR-CNTR
AXES  P1 P2
AXIS X -1.E8  1.E8
AXIS Y -1.E8  1.E8 ;
```

2. In this example, the first command selects a phase-space plot of the kinetic energy/electron in eV versus the x1 coordinate. The second set of commands converts the kinetic energy/electron in eV to kinetic energy/electron in MeV. The y-axis label will use the name of the function.

```
PHASESPACE FORPHAS AXES X1 EN ;
C2MEV = 1.0E-6 ;
FUNCTION ENERGY_per_el_MEV(X1,X2,P1,P2,P3,Q,EN)
= EN*C2MEV ;
PHASESPACE FORPHAS AXES X1 ENERGY_per_el_MEV ;
```

PHASESPACE Command

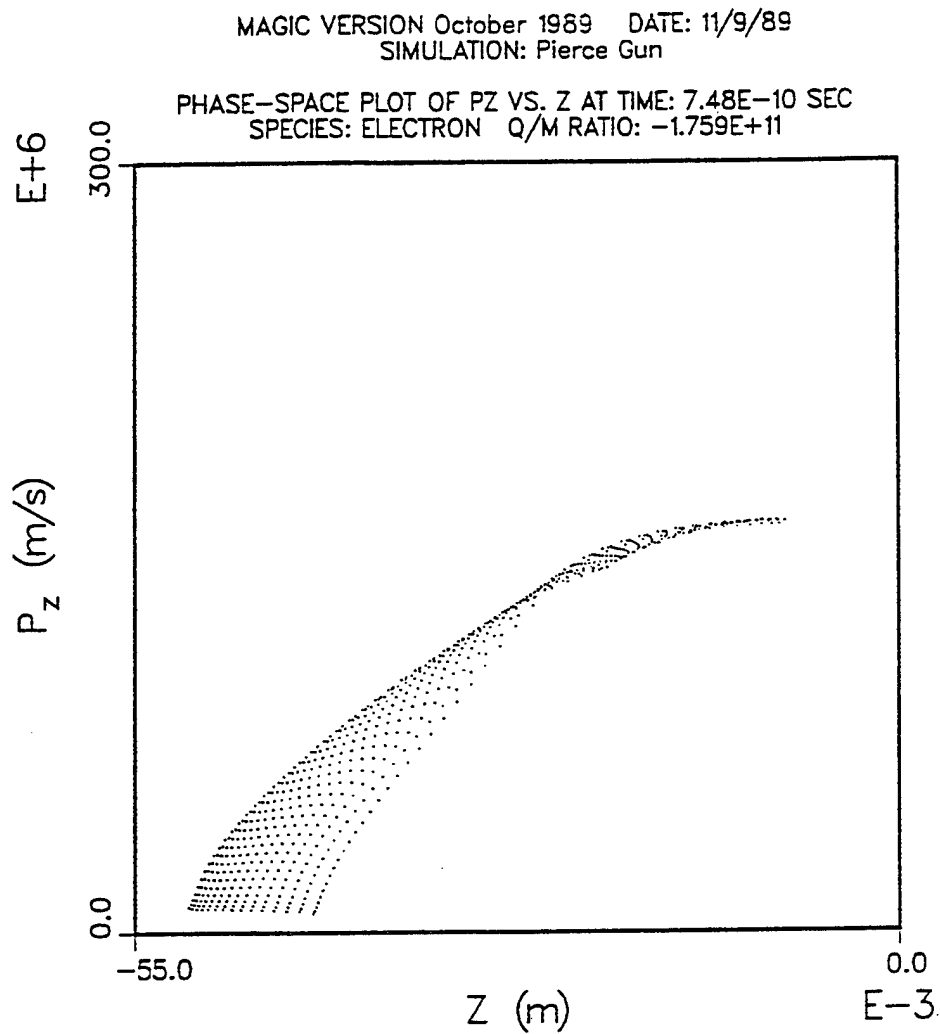


Figure 24-6. Phase space for a Pierce electron gun.

TRAJECTORY Command

Function: Specifies graphical output of particle trajectories.

Syntax:

TRAJECTORY timer species [area_name] ;

Arguments:

timer - timer name, defined in TIMER command.
species - particle species (ALL, ELECTRON, PROTON, or
user-defined in SPECIES command.
area_name - name of plot area, defined in AREA command.

Description:

The TRAJECTORY command will plot particle trajectory paths for a finite number of time steps. The plots are output at the timer trigger times; the duration is specified using the INTEGRATE option. The region of the simulation plotted is specified in cartesian coordinates; if the simulation uses polar coordinates, particle positions are transformed to cartesian before being masked by the boundaries of the plotted region. Unless aspecies is ALL, only one particle species will be displayed in a single plot. Also, only a fraction of these particles will be displayed if trajectory tagging has been specified in the TAGGING command.

Trajectory plots are usually stored in a scratch file and plotted out at the end of the simulation. If the simulation is long and the particles are many, the scratch file can grow extremely large. To avoid this, set kdump to an integer less than the total number of time steps, causing the scratch file to be dumped every kdump time steps. Dumping causes every trajectory plot in the scratch file to be plotted, including those currently in progress, and the scratch file to be erased. Thus, a trajectory with a long duration may be cut in half (i.e., plotted as two separate plots), so the choice of kdump should consider such plots. In any case, the scratch file is dumped at the end of the simulation. The timer index is shifted by one. If the timer triggers at itime, the actual starting time is $t = (itime-1)\delta t$.

Restrictions:

1. The total number of TRAJECTORY commands in a simulation is limited to ten.
2. The selection of a single time point (ksteps=0) may be undesirable when certain plotting hardware make visibility of single points difficult; then, PHASESPACE should be used instead.

TRAJECTORY Command**See Also:**

TIMER, Ch. 12
SPECIES, Ch. 18
GRAPHICS, Ch. 20
TAGGING, Ch. 24

Example:

This example comes from a test case involving magnetic insulation in a coaxial line. We request particle trajectory plots every 500 time steps, beginning at 1500 time steps. Each plot will cover a time span of one time step and a spatial range from 0.1 m to 0.2 m in x1 and 0.07 m to 0.08 m in x2. Figure 24-7 shows one of the plots created. The commands are

```
TIMER TRajs PERIODIC INTEGER 1500 99999 500 INTEGRATE 1 ;  
AREA Plot_area REC 0.1,0.7 0.2,0.8 ;  
TRAJECTORY TRajs ELECTRON Plot_area ;
```

TRAJECTORY Command

MAGIC VERSION: October 1989 DATE: 10/13/89
SIMULATION: MAGIC TEST NO. 1

TRAJECTORY PLOT OF ELECTRONS (ISPE = 1)
FROM TIME 6.247E-09 SEC TO 6.250E-09 SEC

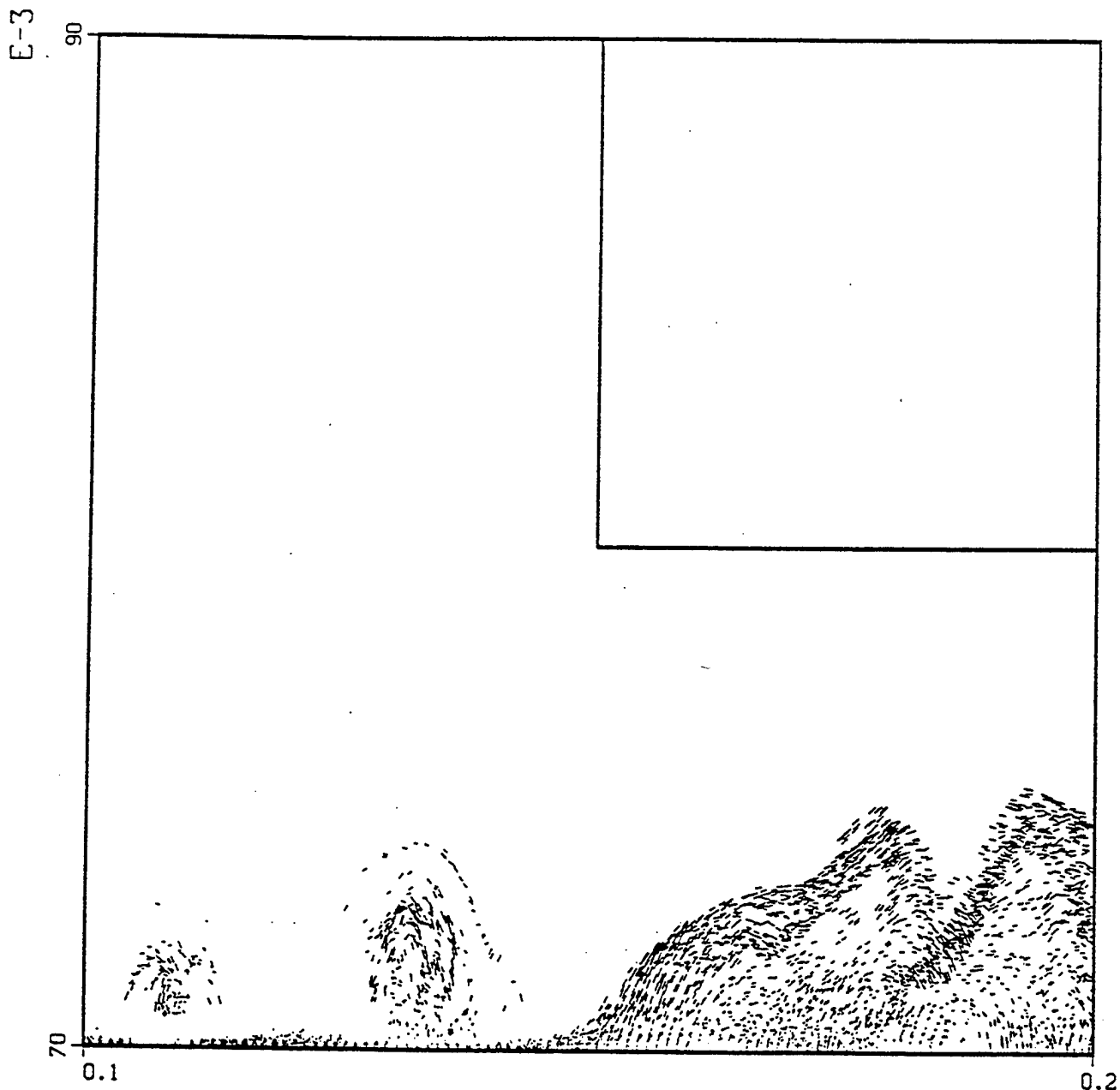


Figure 24-7. Trajectory plot of electrons.

TAGGING Command

Function: Reduces number of particles plotted.

Syntax:
TAGGING fraction ;

Arguments:
fraction - fraction of particles tagged ($0 < \text{fraction} \leq 1$).

Default:

The default tagging fraction is unity.

Description:

The TAGGING command specifies the fraction of particles chosen to appear in particle plots. By default, ftag is 1.0, and particle plots will show all particles in the simulation. However, this default may produce an extremely dense plot and an extremely large plot file; setting ftag to a fraction less than 1.0 limits the plots to showing only a randomly chosen fraction of the particles in the simulation. Particle tagging affects only the output commands PARTICLES, PHASESPACE and TRAJECTORY.

Restrictions:

1. Tagging affects particles of all species.
2. The TRAJECTORY command automatically selects only tagged particles.
3. The PARTICLES and PHASESPACE commands may record either the tagged particles or all the particles, as specified in the command.

See Also:

PHASESPACE, Ch. 24
TRAJECTORY, Ch. 24

Examples:

The following example illustrates the use of TAGGING in conjunction with DUMP and PHASESPACE to create a data file which can be post processed to make a movie. The TAGGING fraction is set to .25 to prevent the data file from becoming too large. The DUMP command is used to store the phase-space data in the data file. The TIMER command is used to specify the rate at which the phase-space data is dumped, and the PHASESPACE command is used to select the data. The commands are:

TAGGING Command

```
TAGGING 0.25 ;  
DUMP TYPE PHASESPACE NOPLOT ;  
TIMER FOR-MOVY PERIODIC INTEGER 1 99999 8 ;  
PHASESPACE FOR-MOVY AXES X1 X2 ;
```

This page is intentionally left blank.

25. DATA

This Chapter covers the following commands:

DUMP
EXPORT
MOVIE
PARTICLES

You can use the DUMP command to write output data to a file for post-processing. You can use the EXPORT command to write data for fields and particles which pass through an outer boundary. (This data can be read in subsequently as a outer boundary for another simulation (IMPORT, Ch. 12). The MOVIE and PARTICLES commands both write particle data to an output file. The distinction between them is that MOVIE is optimized for producing videos and motion pictures.

This page is intentionally left blank.

DUMP Command

Function: Specifies output for post-processing.

Syntax:

```
DUMP
{ PREFIX prefix ,
  SUFFIX suffix ,
  FORMAT { ASCII, BINARY } ,
  TYPE data_type [NOPLOT] } ;
```

Arguments:

prefix	-	ascii prefix to GRD, FLD, PAR and TOC file names.
suffix	-	ascii suffix to GRD, FLD, PAR and TOC file names.
data_type	-	type of data to dump (alpha).
		= SPATIAL, all geometric specifications.
		= CONTOUR, data from contour plots.
		= FLUX, particles crossing flux boundaries.
		= GRID, grid definition.
		= LINPRINT, data from linprint.
		= OBSERVE, data collected by observe.
		= PHASESPACE, data from phasespace plots.
		= RANGE, data from range plots.
		= PERSPECTIVE, data from perspective plots.
		= TRAJECTORY, data from trajectory plots.
		= VECTOR, data from vector plots.
		= ALL, all plots.

Defaults:

The defaults are prefix = input file prefix, suffix = NONE, and FORMAT BINARY. The data_type has no default and must be specified explicitly using the DUMP command.

Description:

This command enables the dumping of data in a format that can be read by MAGIC and POSTER for graphical post-processing and by MAGIC for other simulations.

In general, DUMP output data is sorted according to type and recorded in three files named FLD (field data), PAR (particle data), and GRD (grid data). (TOC is the “table-of-contents” file.) It is not necessary to know what data goes where, since other software reading the files will know which file contains each type of data. The three files are created when the simulation begins, and any of the files for which no data is recorded are deleted at the end of the simulation.

DUMP Command

The data in the files can be reviewed simply by running MAGIC or POSTER with the TOC file as the input file.

You can use the PREFIX and SUFFIX options to create unique names for the four files, as follows:

```
'prefix'FLD'suffix'  
'prefix'PAR'suffix'  
'prefix'GRD'suffix'  
'prefix'TOC'suffix'
```

The default prefix is the same as the input file prefix. If the mode is interactive, the default prefix is 'FILE,' and the default suffix is 'none.'

The FORMAT option specifies the data format. ASCII is useful for transferring data between computers. BINARY (the default format) is compact for saving disk space. The same format is used in all three data files.

The TYPE option specifies the type of plot data. The way DUMP works is to intercept data as it about to be plotted for various data types, such as CONTOUR, OBSERVE, PHASESPACE, and VECTOR. Only certain data is recorded. For example, the CONTOUR option causes the values of the field that is to be plotted to be dumped to a file. VECTOR causes the two fields to be dumped for the locations at which the vectors in the plots are to be drawn. PHASESPACE causes the attributes of the particles that are specified for plotting to be dumped. RANGE causes the set of (x,y) values of the curve to be dumped. OBSERVE causes the set (time,value) to be dumped for each observer. The option NOPLOT causes the plots not to be plotted in-line.

Other dump types, such as GRID, BOUNDARY, and FLUX, do not connect to plotting commands. The GRID type causes the grid definition to be dumped once at the beginning of the run. The BOUNDARY type causes all of the geometric specifications to be dumped from the following commands: CONDUCTOR, SYMMETRY, PORT, OBSERVE, DRIVER, CONDUCTANCE, and FREESPACE. In cases where the user does not specify a name for the geometric specification, the boundary is assigned a name according to the command which entered it and the number of the command in the input file. The FLUX type causes the attribute of each particle that crosses each FLUX boundary to be dumped.

The file names must conform to requirements of the operating system. On systems using the file extension (such as the PC), a period can be used as the last character of the prefix to make FLD, PAR, GRD, and TOC the file extensions (see Examples, below).

DUMP Command**Restrictions:**

The full file name constructed from the prefix and suffix must be a valid file name for the operating system in use.

See Also:

AUTOGRID, Ch. 11
GRID, Ch. 11
OBSERVE, Ch. 22
FLUX, Ch. 22
RANGE, Ch. 23
CONTOUR, Ch. 24
VECTOR, Ch. 24
PERSPECTIVE, Ch. 24
PHASESPACE, Ch. 24
TRAJECTORY, Ch. 24

Examples:

The following set of DUMP commands will cause files to have the names 201.GRD, 201.FLD, 201.PAR, and 201.TOC, which is useful if simulations are given unique numbers. The data will be in binary and the data requested by OBSERVE and PHASESPACE commands will be dumped. Also, PHASESPACE plots will not be shown on the screen or in the printer file.

```
DUMP PREFIX "201." ;  
DUMP FORMAT BINARY ;  
DUMP TYPE OBSERVE ;  
DUMP TYPE PHASESPACE NOPLOT ;
```

EXPORT Command

Function: Defines export surface and timing.

Syntax:

```
EXPORT line_name { POSITIVE, NEGATIVE }
      file_prefix file_format [field] [field] species [timer] ;
```

Arguments:

line_name	-	name of line defining surface, defined in LINE command.
file_prefix	-	prefix of file to record information.
file_format	-	file format (ASCII or BINARY).
field	-	fields to export (see table below).
species	-	species to export (ALL or defined in SPECIES command).
timer	-	timer indicating when to export particles, defined in TIMER command.

Description:

The EXPORT command defines a surface at which fields and particles are exported to a file. The line_name specifies the surface location. Particles are exported when they cross the surface in the indicated direction, either POSITIVE or NEGATIVE.

The file_prefix label applies to two files: one for particles and one for fields. The precise file specification is platform-dependent. For example, on the VAX, file_prefix specifies file_prefixGRD and file_prefixPAR, whereas on UNIX-based platforms, the file_prefix must include a version number; e.g., EXPO.01 specifies EXPOGRD.01 and EXPOPAR.01. The file_format is either ASCII or BINARY. The field and species select which fields and species of particles are to be exported.

The argument, field, specifies the electromagnetic field components to be recorded in the export surface plane. Up to two field components may be specified. If this argument is omitted, then no fields will be exported. Either one or two components may be specified. The field components, uniquely labeled for compatibility with other codes, are shown in the table which follows.

Particles are always recorded by the EXPORT command. The species selects which particles are to be exported as they cross the surface. The particles' coordinate data is transformed and recorded relative to the export surface.

EXPORT Command

Definition of Field Components	
Coordinate System	afield
Cartesian (x,y,z)	EX(X) EY(Y) EZ(X) EZ(Y)
Cylindrical (z, ρ , ϕ)	EZ(Z) ERHO(RHO) EPHI(Z) EPHI(RHO)
Polar (ρ , ϕ ,z)	ERHO(RHO) EPHI(PHI) EZ(RHO) EZ(PHI)
Spherical (r, θ , ϕ)	ER(R) ETHETA(THETA) EPHI(R) EPHI(THETA)

The timer determines the time steps on which the data is exported. If no timer is entered, the default behavior is to export on every time step. If specified, the timer typically will include a contiguous set of time steps. Export will turn on at the end of the first time step indicated in timer. Thus, if timer specifies all time steps from 1000 to 1100, data will actually be exported for steps 1001 through 1100. It is also possible to export data periodically, e. g., from 1000 to 1100 every 10 time steps. In this case, export turns on at the end of time step 1000, and particle data is collected for 10 time steps before being recorded. Thus, the first particle record contains data from steps 1001 to 1010, the next contains particles from 1011 to 1020, etc. If EXPORT is used with a non-unity LORENTZ step_multiplier, then the beginning time step should have modulus, one, and the ending time step should have modulus, zero, relative to the multiple.

Exported particle information is automatically recorded via FLUX commands. These commands are internally activated and are not defined by the user. However, information about exported particles may be obtained in the same manner as for any user-specified FLUX surface by referencing flux surface EXPORT in an OBSERVE or RANGE command.

EXPORT Command**Restrictions:**

Only one EXPORT command is allowed in a simulation.

See Also:

TIMER, Ch. 11
IMPORT, Ch. 12
LORENTZ, Ch. 18
SPECIES, Ch. 18
FLUX, Ch. 22
RANGE, Ch. 23
DUMP, Ch. 25

Examples:

In the following case, an EXPORT command is used to record both TM and TE components of the electric field as well as particles moving in the positive z-direction through a surface of constant z. The coordinate system is cylindrical.

```
TIMER Texport periodic INTEGER 1 100000 1 ;  
LINE EXIT_PLANE STRAIGHT 5mm, 0.0, 5mm, 3mm ;  
EXPORT EXIT_PLANE POSITIVE EXP'icase' ASCII  
    "ERHO(RHO)" "EPHI(RHO)" ALL Texport ;
```

MOVIE Command

Function: Selects particle information to record in database file.

Syntax:

```
MOVIE timer
    [ SPECIES species ]
    [ TAGGING tag ]
    [ WINDOW variable wmin wmax ]
    [ DATA data ] ;
```

Arguments:

timer	- timer name, defined in TIMER command.
species	- particle species name (ALL, ELECTRON, PROTON, or user-defined in SPECIES command.
tag	- particle output tagging. = NO, select all particles. = YES, select only particles specified in TAGGING command.
variable	- window variable (Q0, X1, X2, P1, P2, P3, or EN).
wmin,wmax	- window limits (real).
data	- variable selection string (Q0, X1X2EN, P1P2P3, etc.)

Description:

The MOVIE command is used to select particle information for recording in a DUMP file. Due to the volume of data normally associated with particle records, it is generally advisable to restrict recording to those features which are actually required, and the MOVIE command provides a convenient way to accomplish this. It is possible to record any or all of the following variables:

Q0	- particle charge (Coul)
X1, X2	- spatial coordinates (m or rad)
P1, P2, P3	- relativistic momenta (m/sec)
EN	- energy (J).

Particle information is written to the DUMP file at the simulation times specified by atimer. (See the DUMP command for details in invoking the recording of particle data.) After the simulation completes, this file can be read and analyzed with POSTER.

The keyword, SPECIES, is used to select a particle species. The default for this option is to record all particle species.

MOVIE Command

The keyword, TAGGING, is used to specify only tagged particles for recording in the dump file. Tagged particles can be selected only if tagging has been enabled in the TAGGING command; otherwise, all particles will be selected for recording. The default for this option is TAGGING NO, in which case, all particles are recorded.

The keyword, WINDOW, is used to provide a window in any of the particle variables to discriminate against recording. The variable specifies the particle variable and the arguments, wmin and wmax, specify the window extrema. Only particles inside the window limits are recorded. The default for any variable not specified in a WINDOW option is infinite extrema, i.e., no window (see Examples, below).

The keyword, DATA, specifies which particle variables will actually be recorded. For example, the specification, DATA X1P1, will cause only the X1 and P1 variables to be recorded. The default for the DATA option is that all variables will be recorded. This is equivalent to the specification, DATA Q0X1X2P1P2P3EN (see Examples, below).

The MOVIE command is very similar to the PARTICLES command. The distinction between them is that the MOVIE command allows the user to specify certain coordinates to be written, and so is even more discriminatory than PARTICLES. In addition, data produced with the MOVIE command is written in a compressed (integer) format which saves space but is less precise than the PARTICLES format.

Restrictions:

The total number of MOVIE commands is limited to ten.

See Also:

TIMER, Ch. 11
SPECIES, Ch. 18
TAGGING, Ch. 24
DUMP, Ch. 25
PARTICLES, Ch. 25

Examples:

The following commands,

```
TIMER EVERY PERIODIC INTEGER 1 999999 1 ;  
MOVIE EVERY SPECIES PROTON TAGGING YES  
WINDOW P1 0.0 3.0E10 DATA X1EN ;
```


MOVIE Command

will cause data to be recorded at every time step for all particles of the PROTON species, specifically, that fraction of such particles specified in a TAGGING command and further restricted to having positive momentum, P1. For these particles, only the variables X1 and EN will be recorded.

PARTICLES Command

Function: Selects particle information to record in database file.

Syntax:

```
PARTICLES timer
      [ SPECIES species ]
      [ TAGGING { YES, NO } ]
      [ WINDOW variable wmin wmax ] ;
```

Arguments:

timer - timer name, defined in TIMER command.
species - particle species name (ALL, ELECTRON, PROTON, or
 user-defined in SPECIES command).
variable - window variable (Q0, X1, X2, P1, P2, P3, or EN).
wmin,wmax - window limits (real).

Description:

The PARTICLES command is used to select particle information for recording in a DUMP file. Due to the volume of data normally associated with particle records, it is generally advisable to restrict recording to those features which are actually required, and the PARTICLES command provides a convenient way to accomplish this. For each selected particle, a record is made of all of the following variables:

Q0 - particle charge (Coul)
X1, X2 - spatial coordinates (m or rad)
P1, P2, P3 - relativistic momenta (m/sec).

Particle information is written to the DUMP file at the simulation times specified by atimer. (See the DUMP command for details in invoking the recording of particle data.) After the simulation completes, this file can be read and analyzed with POSTER.

The keyword, SPECIES, is used to select a particle species. The default for this option is to record all particle species.

The keyword, TAGGING, is used to specify only tagged particles for recording in the dump file. Tagged particles can be selected only if tagging has been enabled in the TAGGING command; otherwise, all particles will be selected for recording. The default for this option is TAGGING NO, in which case, all particles are recorded.

The keyword, WINDOW, is used to provide a window in any of the particle variables to discriminate against recording. The variable specifies the particle variable and the arguments, wmin and wmax, specify the window extrema. Only particles inside the window

PARTICLES Command

limits are recorded. The default for any variable not specified in a WINDOW option is infinite extrema, i.e., no window (see Examples, below).

The PARTICLES command is very similar to the MOVIE command. The distinction between them is that the MOVIE command allows the user to specify certain coordinates to be written, and so is even more discriminatory than PARTICLES. In addition, data produced with the MOVIE command is written in a compressed (integer) format which saves space but is less precise than the PARTICLES format.

Restrictions:

The total number of PARTICLES commands is limited to ten.

See Also:

TIMER, Ch. 11
SPECIES, Ch. 18
TAGGING, Ch. 24
DUMP, Ch. 25
MOVIE, Ch. 25

Examples:

The following commands,

```
TIMER EVERY PERIODIC INTEGER 1 999999 1 ;  
PARTICLES EVERY SPECIES PROTON TAGGING YES  
WINDOW P1 0.0 3.0E10 ;
```

will cause data to be recorded at every time step for particles of the PROTON species, specifically, that fraction of such particles specified in a TAGGING command and further restricted to having positive momentum, P1.

This page is intentionally left blank.