This blank page was inserted to preserve pagination.

MAC-TR-62
ESL-R-395

EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation

User's Guide

An appendix for the Ph.D. thesis
"On-Line Solution of Elliptic
Boundary-Value Problems" [1]

Coyt C. Tillman, Jr.

Department of Mechanical Engineering
Massachusetts Institute of Technology

June, 1969

Project MAC and Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

This empty page was substituted for a blank page in the original document.

Abstract

This appendix for the author's forthcoming thesis, "On-Line Solution of Elliptic Boundary-Value Problems," is a user's guide for EPS. EPS solves two-dimensional boundary-value problems for elliptic systems of second-order partial differential equations. It also has general-purpose capabilities which permit the on-line definition and execution of arbitrary numerical procedures.

The guide is concerned primarily with the use of EPS for solving elliptic boundary-value problems. Linear problems of this type that have no complications such as free surfaces or undefined parameters can be solved on a one-pass basis. Nonlinearities and other complications can be accommodated by iteration. Solutions are obtained by a finite-difference method which permits the use of irregular lattices, hence the crowding of nodes in sensitive regions.

EPS operates on the IBM 7094 computer of the M.I.T. Compatible Time-Sharing System (CTSS), and exploits to an unusual degree the potential for interactive problem solving that CTSS affords. Input commands resemble statements in various algebraic compiler languages, and can be combined and abbreviated by means of macros. Improper input and other error conditions are handled so as to minimize user inconvenience. Common syntax errors, for example, are corrected automatically by the machine. Output is available in either numerical or graphical form.

Acknowledgements

EPS was developed in conjunction with the M.I.T. Computer-Aided Design Project. It is written largely in the AED-0 language developed by the Project and uses the ESL display equipment for three-dimensional surface displays. Many important features now available in AED, e.g., the free-storage package, input-output facilities, and the graphics package, were still under development when work on EPS began. Hence it was necessary to develop independently several subsystems which today would be provided by AED itself. Program design of EPS was greatly influenced by methods conceived by the AED group, and, as an early, comprehensive, computer-aided design system, EPS serves to demonstrate the utility of the AED tools and techniques.

This work has been supported by the Air Force Manufacturing Technology Laboratory, RTD, Wright-Patterson Air Force Base, under Contracts AF-33(657)-10954 and F33615-67-C-1530, M.I.T. DSR Projects 79442 and 70429, Electronic Systems Laboratory Computer-Aided Design Project.

Support has also been provided by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

Formative stages of the work were performed while the author was an IBM research assistant at the M.I.T. Computation Center.

Preface

This user's guide for EPS will be included as an appendix in the thesis "On-Line Solution of Elliptic Boundary-Value Problems" [1]. However, for the convenience of current and potential EPS users, it is being made available in advance of the entire thesis as a separate, self-contained document. As this document necessarily specifies the capabilities and important features of EPS, it may be of interest not only to actual users, but also to others with interests in such areas as man-machine communication, problem-oriented language design, and numerical analysis.

The guide has sixteen major sections, the last being a summary of the preceeding fifteen. For purposes of orientation, it may be helpful to read the summary first (pages 168-177).

Aside from the summary, many users will find that only the first seven sections are essential. Sections 1 and 2 discuss program access and basic operating conventions, while Sections 3 through 7 discuss the steps required for solving two-dimensional boundary-value problems for linear elliptic systems of second-order partial differential equations. Users with problems of this class that are initially well-defined, i.e., that have no free surfaces or parameters needing optimization, should in principle be able to ignore the rest of the guide, although operation of the system may be simplified by using features discussed later on.

On the other hand, users with problems that have nonlinearities or other complications, as well as those who wish to investigate various special extensions, e.g., the generation of flow nets or the use of EPS as a general-purpose language, will probably find the rest of the guide essential, particularly Sections 8, 9, and 10.

All users may need to refer occasionally to Sections 12 through 15 in order to overcome various possible difficulties.

Contents

Abstract		iii
Acknowledgements		iv
Preface		v
List of figures		ix
List of tables		ix
Introduction		xi
Section 1.	Access	1
Section 2.	Basic operating conventions	3
Section 3.	Lattice specification	7
3.1.	Admissible configurations	7
3.2.	Practical limitations and recommendations	11
3.3.	Computer input: basic approach	15
3.4.	Specifying lattice structure in i,j-space	16
3.5.	Specifying a mapping algorithm	17
Section 4.	Equation specification	22
4.1.	Admissible forms	22
4.2.	Specification of parameters	23
4.3.	Examples	27
Section 5.	Specification of boundary conditions	30
5.1.	Admissible forms	30
5.2.	Assigning segment numbers	31
5.3.	Specification of parameters	33
5.4.	Example	36
5.5.	Supplementary remarks	37
Section 6.	Getting a solution	41
6.1.	Generation of finite-difference equations	41
6.2.	Solution of finite-difference equations	41
Section 7.	Output	46
7.1.	Printed output	46
7.2.	Graphical output	50

Section 8. Data modification 52
8.1. Contour modification 53
8.2. Parameter modification 55
8.3. Modification of segment numbers 56

Section 9. Macros 57
9.1. Introduction 57
9.2. When macros can be used 60
9.3. Macro definitions 61
9.4. Special forms 66
9.5. Separators, arguments, and imperfect calls 69
9.6. Macro deletion and respecification 74
9.7. Additional examples 75

Section 10. Iterative problem solving 81
10.1. Introduction 81
10.2. Nonlinear equations 82
10.3. Iterative lattice revision 91
10.4. Over- and underrelaxation of iterated parameters 109

Section 11. Disk input and output 113
11.1. Disk input 113
11.2. Disk output 116

Section 12. Input verification 118
12.1. Determining active identifiers 118
12.2. Determining individual definitions 119

Section 13. Program interruption and run termination 123
13.1. Interrupts 123
13.2. Run termination 125

Section 14. Error messages 127

Section 15. Command descriptions 130
15.1. Terminology 130
15.2. Descriptions 134

Section 16. Summary 168

References 178

List of figures

1. Conventional lattices based on regular, orthogonal grids 8
2. Geometrically irregular lattices 9
3. Lattice for a multiply-connected domain 15
4. Lattice with linearly interpolated nodal distribution 19
5. Numbering of lattice sectors 27
6. Sample lattice and segment numbers 33
7. Sample lattice and boundary conditions for Laplace's equation 36
8. Typical displays produced by means of the plot-command 51
9. Iterative strategy for nonlinear equations, etc. 81
10. Schematic of trapezoidal dam 93
11. Lattice for seepage region of dam with interpolated nodal distribution 95
12. Graphical displays illustrating iterative generation of a flow net 104

List of tables

1. Summary of basic command forms 173
2. Summary of built-in functions 175
3. Summary of operators 176

This empty page was substituted for a blank page in the original document.

Introduction

When the development of EPS first began in the fall of 1964, it was intended that this system be a fully automatic, numerical assistant which would accept "textbook problem descriptions" of a particular class and produce reliable answers without demanding an intimate knowledge of computer methods from its users. Specifically, it was intended that EPS solve two-dimensional boundary-value problems for linear elliptic systems of second-order partial differential equations,[†] and it was hoped that users would be able to obtain solutions to particular problems by supplying only essential information describing governing equations, boundary geometry, and boundary conditions.

In a sense, EPS has both overshot and fallen short of its original goals. It has overshot these goals insofar as it is able today to treat a class of problems which is much broader than, although not so rigorously definable as, the class for which it was initially intended. On the other hand, EPS is not the "fully automatic, numerical assistant" once envisioned. With the benefit of hindsight, it is possible to say that both of these results were inevitable. The class of problems which EPS was meant to solve is itself broad enough to indicate the use of

[†]EPS is an acronym for "Equilibrium Problem Solver," an expression suggested by the fact that in mathematical physics boundary-value problems generally arise when continuum models are employed in the analysis of physical systems at equilibrium.

numerical methods which, for maximum flexibility, are best implemented in a semi-automatic form. Thus, in particular, it is necessary in EPS for the user to define a discrete model for his problem, in the form of a finite-difference lattice, in addition to the various parameters which are mathematically essential. At the same time, however, the many variables left open by the class of problems for which EPS was intended necessitated certain general-purpose, algebraic capabilities which make possible many important extensions. For example, it is possible to have EPS compute revised problem or lattice parameters using results from previous solutions. Consequently, iterative procedures can be effected which permit the treatment of boundary-value problems with nonlinearities, free surfaces, or undefined parameters. Further, the general-purpose features of the system can be exploited in an independent manner, and enable the user to specify and have executed algorithms which have nothing to do with the solution of boundary-value problems.

While in principle the various capabilities of EPS derive solely from specific computational features, in practice the utility and useability of the system stem as well from the fact that it was designed for interactive operation in an on-line computing environment. This arrangement is especially convenient, perhaps essential, in sophisticated applications, where the user may need to make numerous decisions in order to obtain solutions. Moreover, as the beginning user will quickly appreciate, the task of learning proper operating procedures is greatly facilitated when the results of a particular action can be determined immediately. In the design of EPS, considerable effort has been made to insure that input errors are detected quickly and reported

in an instructive manner, and, in general, there is no danger of destroying the system or losing past results because of erroneous input. Therefore, the user should not only feel free to experiment, but should realize that experimentation may well be the most expedient way of determining whether a specific input form or solution strategy is valid.

This empty page was substituted for a blank page in the original document.

1. Access

EPS was implemented for operation on the IBM 7094 computer of the M.I.T. Compatible Time-Sharing System (CTSS). It is assumed that the reader has access to CTSS and, further, that he is familiar with basic CTSS operating procedures and nomenclature. (The beginning user may find the "Time-Sharing Primer" in [2], Section AA.2, to be helpful in this regard.) Meeting these requirments, he may log in[†] at a computer console and proceed as indicated below.

There are currently two versions of EPS. One is maintained in a disk file called "eps saved" and is intended for users who have no desire, either present or anticipated, for graphical output. The other is maintained in a file called "eps-p saved" and is intended for users who do wish graphical output and who therefore have either logged in at an ESL graphic display console[§] or anticipate continuing their computations from such a console at a later time. To establish a link to either file, type[¶*]

```
link name1 saved m1416 cmf104
```

where $name_1$ is the appropriate first file name, i.e., "eps" or "eps-p".

[†] See [2], secs. AA.2 and AH.1.01.

[§] See [2], secs. AC.0 and AH.2.06, and Thornhill, et al [3].

[¶] EPS is available through the CTSS public file directory. Cf. [2], secs. AD.4 and AH.3.05.

[*] This guide uses lower-case letters for computer input and upper-case letters for computer output, as is the normal convention on most CTSS consoles. Characters intended literally are always shown in sans-serif type, while variable information is shown in italics or other easily distinguished type styles.

The computer will respond with lines of the form

W *T*
R ΔT_1+ΔT_2

where T, ΔT_1 and ΔT_2 represent timing information, as explained in [2]. After this, type[†§]

resume $name_1$

to initiate a run. The computer will respond by printing

W *T*
THIS VERSION OF EPS WAS LAST REVISED ON *DATE*.

PROCEED:

With this, submission of input can begin.

[†]Cf. [2], sec. AH.7.03.

[§]If the user is stationed at and planning to use an ESL display, he may need to load the PDP-7 display monitor before resuming EPS. See posted instructions or ESL staff for current procedures.

2. Basic operating conventions

Like CTSS, EPS performs the various operations of which it is capable in response to explicit commands from the user. Commands can be supplied directly from the console or they can be supplied from a disk file prepared in advance. However, it is recommended that only console input be used initially, if not in general, as it lends itself to faster understanding and poses fewer complications when mistakes are made. Therefore, most of this guide, including the present section, assumes that input is being typed in directly. Disk input, which in any event is basically simulated console input, is discussed further in Section 11, as is a mechanism for obtaining disk output.

In contrast to CTSS, the basic unit of input to EPS is not necessarily a single line of type representing a single command. Instead, any number of lines may be typed, and these may represent any number of commands. (No correspondence between physical lines and individual commands is required: one command may follow another on the same line, or a command may be continued from one line to the next, without special punctuation or continuation marks.) Thus the general procedure for using EPS is as follows: After receiving a "PROCEED:", the user types one or more commands on as many lines as are necessary and convenient, the maximum line length being 83 characters. After typing the last command and before giving a final carriage return, he types a "$" to indicate that processing can commence. The computer executes the commands in order of their specification, then prints another "PROCEED:" to indicate that more input can be submitted. This procedure is illus-

trated by the following sequence, which happens to show the definition of a finite-difference lattice for a semi-annular domain:

```
PROCEED:
append 0,0, 8,0, 8,20, 0,20 to zeta
close zeta$
DEFINITION OF NEW CURVE 'ZETA' HAS BEEN COMPLETED.
'ZETA' HAS BEEN CLOSED.

PROCEED:
define x=radius*cos(theta), y=radius*sin(theta),
    radius=ri+(ro-ri)*i/8, theta=pi*j/20
set ri=2.2, ro=4.7, pi=4*atan(1), i=4, j=0
print x,y $
   3.4500000E+00    0.00000000000

PROCEED:
  etc.
```

It is absolutely essential that the user type a "$" and a carriage return on completing each unit of input; otherwise, the computer will wait for more input and accomplish nothing. Eventually, of course, the user will wish to stop interacting with EPS and go on to other matters. To do this, he can type

```
quit$
```

whereupon control will be returned to the CTSS supervisor. For further details, see Section 13.

As the examples above indicate, commands for EPS are basically like statements in various algebraic compiler languages. However, certain common restrictions have been eliminated in the interest of more natural and flexible communication. The following features should be noted, in particular:

(1) An identifier, e.g., the name of an algebraic parameter, may be any string of up to 83 letters and digits that begins with a letter and is not a reserved word.[†]

(2) A number may be written in integer, decimal, or scientific (E-) format, regardless of the type of data it represents. Thus the following are equivalent: 30, 30.0, 3.0e+01, 3e1, .3e2, 300e-1.

(3) Command arguments that represent numerical values may be written as algebraic expressions in terms of numbers, parameter names, built-in functions, and standard algebraic, relational, and Boolean operators.[§] Further, because Boolean values are represented internally as numerical 1's and 0's, and because no parameter mode distinctions are observed, it is permissible to mix operations of different types at will. For example, the expression

```
(s grt 5 and s les 7.9)*t
```

has the value "1*t → t" when "s" lies between 5 and 7.9, and the value "0*t → 0" otherwise.

(4) Explicit storage declarations for subscripted parameters, etc., are unnecessary, as storage is allocated and reallocated automatically as problem information is introduced and deleted.

[†]Reserved words are the command words, multiliteral operators, and built-in function names shown in sans-serif type in Tables 1-3 of sec. 16, plus "if", "insert", and "bcd". The symbol "bcd" is used to introduce literals, which, though intended primarily for specifying output formats, may also be used as identifiers. See secs. 7.1 and 15.1.

[§]See sec. 15.1 and Tables 2 and 3, sec. 16.

Other conventions are more or less standard. For example, blanks are optional between any input element and any printed delimiter, namely, any one of the following:

```
+ - / * ( ) = : , $
```

On the other hand, at least one blank must be introduced between adjacent words and numbers. Tabs and carriage returns are considered to be logically equivalent to blanks; therefore, it is not possible to split up a word or number, typing part on one line and the rest on the next.

Typing errors in individual lines of EPS input can be corrected by the same methods used to correct command lines to CTSS, namely, by typing a question mark (?) or at-sign (@) to cause all preceeding characters in a given line to be ignored, or by typing one or more quotation marks (") or number signs (#) to cause one or more immediately preceeding characters to be ignored.† Further, the processing of input, once it has been typed, can be curtailed, if not prevented entirely, by use of the program interrupt button (see Section 13), while erroneous data can always be corrected by typing new commands which in effect override old ones (see Section 8). In any event, as indicated earlier, errors which are not detected by the user are generally detected and agreeably reported by the computer; in fact, many common syntax errors are not only detected but also corrected automatically (see Section 14).

---

†Beginning users should beware that, because EPS uses the CTSS 6-bit character set, certain symbols available on some consoles, e.g., the [ ] % ¢ ! etc., are ignored. Thus typing `sin(t%")` is equivalent to typing `sin()`, not `sin(t)`. Cf. [2], sec. AC.2.

3. Lattice specification

Input required by EPS for the solution of a two-dimensional boundary-value problem can be divided into three catagories:

(1) information describing the finite-difference lattice by which the domain of interest is to be modeled;

(2) information describing the governing partial differential equation(s) for the domain; and

(3) information describing the auxiliary conditions to be imposed along the contour(s) which bound the domain.

Since EPS does not actually assemble such information until a solution is explicitly requested, the order in which it is presented to the computer is virtually arbitrary. However, for the sake of concreteness, it is assumed here that the user will specify his input in the order indicated above. Therefore we first consider the selection and specification of a finite-difference lattice.

3.1. Admissible configurations. In the analysis of two-dimensional boundary-value problems by conventional finite-difference methods it is common to employ lattices based on orthogonal grids of equally spaced, straight lines. In some cases the analysis is carried out by writing and solving difference equations for those points which lie within or on the boundary of the domain of interest and which represent intersections of the grid lines with one another or with the boundary (see Figure 1a). The admission of boundary points which do not

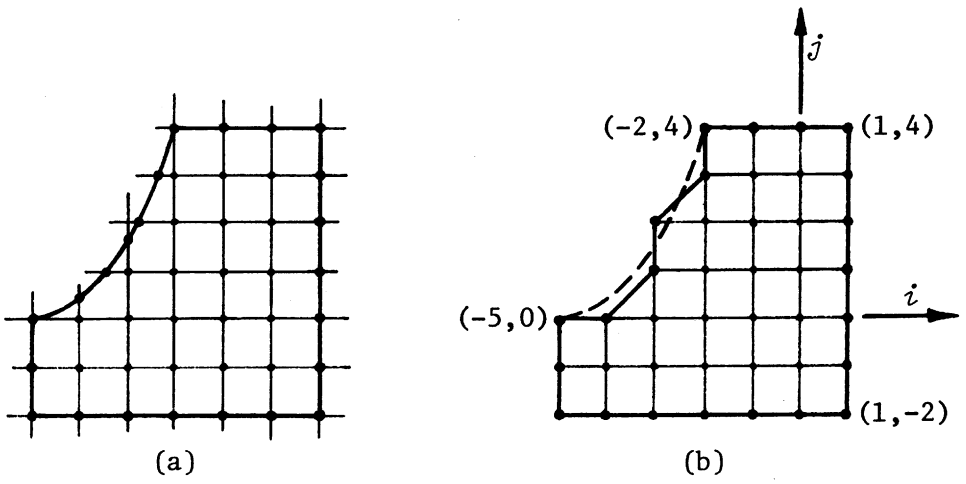


Fig. 1. *Conventional lattices based on regular, orthogonal grids. Labeling conventions employed with EPS prohibit lattice* (a), *but admit lattice* (b).

coincide with the nodes of the grid, i.e., the points at which the grid lines intersect one another, permits accurate modeling of boundary geometry, but introduces the possibility of irregularities in lattice topology which complicate automatic analysis. For this reason, many computer programs require that domain boundaries be represented (in general, approximated) by polygons whose sides join neighboring or diagonally opposed nodes (see Figure 1b). In fact, a requirement of this type is imposed by EPS, so that the lattice of Figure 1b is admissible to EPS while that of Figure 1a is not. Fortunately, however, Figure 1b is not representative of the entire class of lattices which EPS will accept, for EPS does not demand that grid lines be orthogonal, equally spaced, or straight. (It assumes that nodes are connected with straight lines, but allows these lines to vary in length and direction from node to node.) Consequently, the domain of Figure 1 could also be modeled with any of the lattices shown in Figure 2.

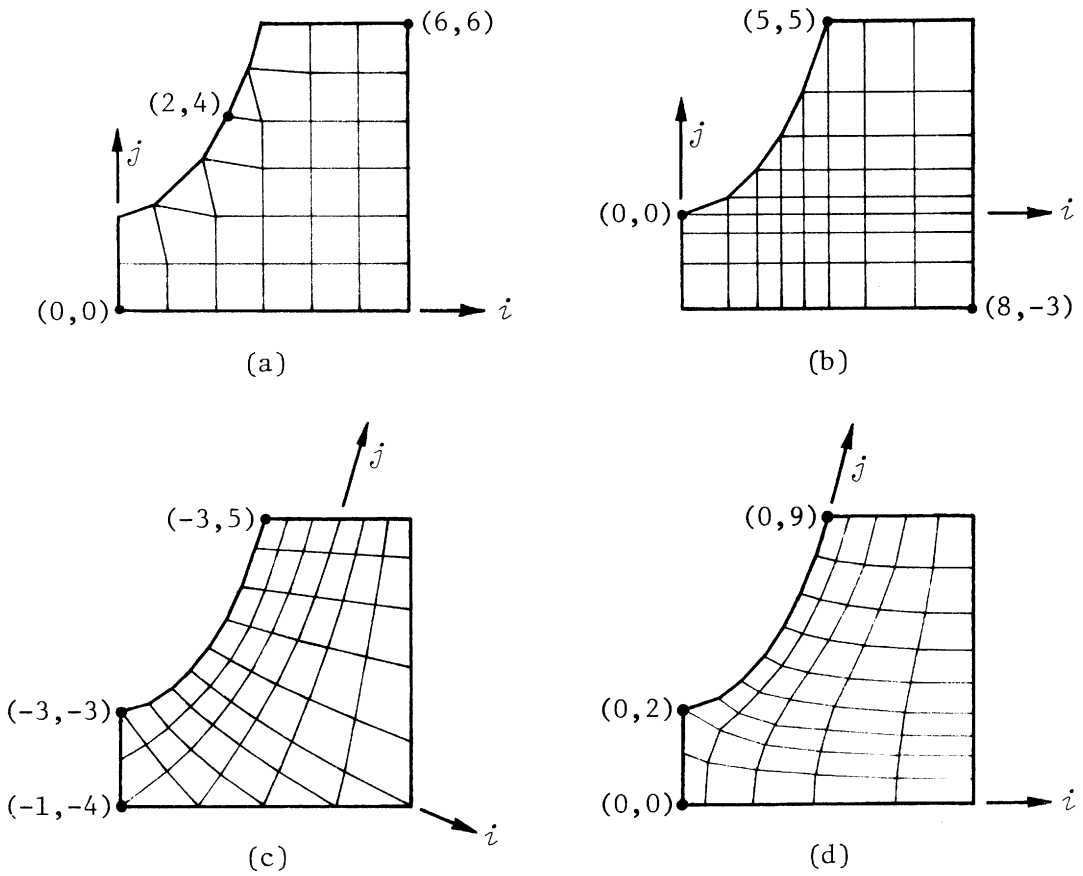


Fig. 2. *Geometrically irregular lattices, all admissible to EPS, with possible grid coordinate labelings.*

It is possible to label all lattice points in Figure 1b, and also all points in the lattices of Figure 2, in a discrete manner, assigning each node a unique pair of integers, or grid coordinates, such that the immediate neighbors, listed in clockwise order, of any node (i,j) not on a lattice boundary are $(i+1,j)$, $(i,j-1)$, $(i-1,j)$, and $(i,j+1)$. (Note that such a labeling is impossible for Figure 1a because of the special points along the curved portion of the boundary.) With this labeling convention, grid lines can be defined formally as paths through a lattice connecting nodes of constant i or constant j.

Clearly, if such a labeling is possible at all, then it can be carried out in any number of ways. However, once an i- or j-direction and the grid coordinates for any single node have been selected, the coordinates of all other nodes are fixed, and can be determined by counting.

It can be stated categorically that any finite lattice which can be labeled with discrete grid coordinates in the manner described above is, in principle, admissible to EPS. Conversely, lattices that cannot be so labeled are not admissible. In order for an acceptible labeling to be possible, a lattice must satisfy the following requirements:

(1) All lattice points must represent the intersections, or nodes, of two families of grid lines (lines of constant i and lines of constant j), arranged in such a way that each node not on the lattice boundary is connected to four and only four neighbors. This requirement specifically prohibits grid lines which terminate at interior nodes, a possibility sometimes admitted to allow for abrupt changes in lattice spacing.

(2) Neighboring grid lines of the same family, i.e., lines of constant i (or j) with i- (or j-) values differing by ±1, may not coincide with or cross over one another. However, in principle at least, several lines of the same family may converge to a single point (and then diverge away from that point, if the point is not on the lattice boundary), producing a set of superimposed nodes. (See Section 3.2 for practical recommendations.)

(3) The lattice boundary must consist of one or more closed polygons, the sides of which either coincide with grid lines or form grid diagonals, i.e., lines connecting nodes whose grid coordinates satisfy a relation of the form

$$j = \begin{Bmatrix}\text{integer} \\ \text{constant}\end{Bmatrix} \pm i$$

Note that in general any admissible lattice will appear as a net of quadrilateral and triangular elements, the latter arising (if at all) either along boundary segments which represent grid diagonals or in places where quadrilateral element degenerate into triangles because of the use of superimposed nodes.

3.2 Practical limitations and recommendations. In principle, the number of admissible lattices for any given problem is unlimited, not only because the number of grid lines and nodes is variable, but also because there are frequently several choices as to which basic form a lattice should assume. In practice, of course, there are various limiting considerations, the most important of which are discussed below.

(1) For the following reasons, it is generally impractical, if not impossible, to employ lattices with more than a few hundred nodes:

(a) The generation of finite-difference equations for a system of n second-order partial differential equations and a lattice with m nodes requires about $n^2m/10$ seconds of IBM 7094 computer time. Time requirements

for the solution of these equations and for the generation of output vary considerably, but taken together are typically of the same order.

(b) The number of words of core storage W required to treat a particular problem varies with

- m total number of nodes
- b number of boundary nodes
- n number of partial differential equations
- n_k number of dependent variables in kth equation
- d number of distinct sets of difference equation coefficients (see below)
- p number of storage words needed to represent algebraic parameters and macros used in problem definition (typically 500 to 2000 words)

and is given approximately by

$$W \simeq mn + 8b + \left(1 + 9\sum_{k=1}^{n} n_k\right)d + p$$

Clearly W must not exceed the amount of storage actually available, namely, 20,500 words with "eps saved" and 18,600 words with "eps-p saved". For regular lattices and differential systems with constant coefficients, it is likely that only a few distinct sets of difference equation coefficients will be stored (d = 1 or 2 or 3 ...), in which case several thousand nodes may be treated. However, for irregular lattices and/or systems with variable coefficients, it may be necessary to store a separate set of coefficients for each node ($d = m$), bringing the maximum number of nodes down to several hundred.

(2) In order to obtain high accuracy with practicable lattices, it is frequently necessary to crowd nodes in regions where

a solution is known, either by analysis or previous numerical experimentation, to have "large" third- or higher-order derivatives. In the neighborhood of a singularity, for example, the nodal density may need to be several orders of magnitude greater than in that of well-behaved regions.[†]

(3) In constructing lattices with variable nodal spacing, it is recommended that abrupt changes in spacing be avoided.[§] This argues for the use of a functional mechanism for arriving at nodal distributions. Such a mechanism may be specified quite easily with the EPS command language, as is shown in Section 3.5.

(4) In general, nodes should be arranged so that quadrilateral lattice elements are as nearly square as possible. Grid lines which intersect at angles less than 45° and lattice elements with aspect ratios greater than 5:1 should be avoided in most applications, especially in sensitive regions.

(5) In problems involving material interfaces or other conditions leading to differential equations with discontinuous coefficients, it is generally more accurate and convenient to arrange grid lines so that they coincide with lines of discontinuity. Similarly, nodes should be placed at all boundary points where boundary conditions change abruptly.

[†]See Parmelee [4], pp. 75ff.

[§]See Varga [5], p. 191.

(6) Other matters being equal, it is advisable to avoid lattice configurations which, because of the presence of diagonal boundary segments or superimposed nodes, contain triangular elements. Superimposed nodes should specifically be avoided at interior points of a domain, and should be used at boundary points only when boundary values, as opposed to gradient or more complex conditions, are prescribed for all dependent variables.

The above remarks are necessarily qualitative, for answers to specific questions, e.g., "How much accuracy can I expect from a lattice like 'this'?", are available only for a few special classes of lattices, and then only for simple differential systems. In practical applications it is generally necessary to determine solution accuracy empirically, for example, by treating for comparison a problem variation having a known solution or by modeling the actual problem with two or more lattices, each having a different number of nodes. In the latter approach, it may be possible to estimate errors roughly by noting the degree to which the numerical solution changes from lattice to lattice. In fact, if lattice modifications are handled in a systematic manner, the basic form of the lattice being held constant as more grid lines are introduced, it may be possible to obtain a greatly improved solution by extrapolation.†

---

†See Crandall [6], pp. 171-173 and pp. 269-271, and Parmelee [4], pp. 65ff.

3.3. Computer input: basic approach. As Section 3.1 may suggest, the process of selecting a lattice for EPS generally involves at least an implicit assignment of grid coordinates, it being convenient, in general, to introduce grid coordinates in order to test a lattice for admissibility. Grid coordinates are, in any event, essential to the procedure by which lattice information is communicated to the

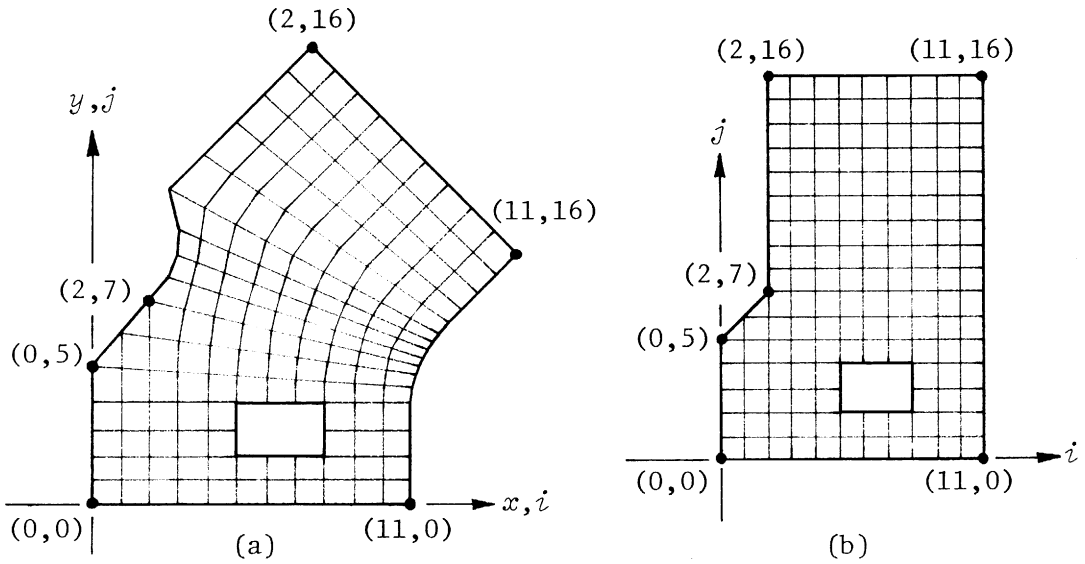


Fig. 3. *Lattice for a multiply-connected domain in* (a) *plane of independent variables* x *and* y, *and* (b) i,j*-plane.*

computer, and should be assigned in a manner which is natural and convenient to the problem at hand.† In essence, the grid coordinates form a discrete, curvilinear reference system, making it possible to view any lattice as a mapping from the i,j-plane of a regular, orthogonal grid bounded by horizontal, vertical, and ±45° lines (see Figure 3). The computer input required to define a lattice closely parallels this

†It is permissible to use both positive and negative integers, provided magnitudes are kept less than 2^{27} (134,217,728).

conceptualization, for it is necessary to provide the machine with a description of the lattice as it appears "in i,j-space" and with a mapping algorithm by means of which i,j-values can be converted into corresponding values for a problem's actual independent variables. The specific commands needed for these operations are discussed below.

3.4. Specifying lattice structure in i,j-space. Lattice structure in i,j-space is specified by defining the path of each bounding contour (only one in singly-connected domains) in terms of its i,j-coordinates. Specifically, it is necessary to type a command sequence of the form

append i_1,j_1, i_2,j_2, ..., i_N,j_N to *curvename*
close *curvename*

for each contour *curvename*, where grid coordinates i_1,j_1, i_2,j_2, ..., i_N,j_N represent the contour's vertices in the i,j-plane, listed in either clockwise or counterclockwise order.† For example, for the lattice of Figure 3, the user could type

```
append 0,0, 0,5, 2,7, 2,16, 11,16, 11,0 to alpha
close alpha
append 5,2, 5,4, 8,4, 8,2 to beta
close beta $
```

The computer would respond by printing

†Note that vertices in the i,j-plane may or may not coincide with boundary discontinuities in the plane of the independent variables.

```
DEFINITION OF NEW CURVE 'ALPHA' HAS BEEN COMPLETED.
'ALPHA' HAS BEEN CLOSED.
DEFINITION OF NEW CURVE 'BETA' HAS BEEN COMPLETED.
'BETA' HAS BEEN CLOSED.
```

The use of "alpha" and "beta" as contour identifiers is, of course, completely arbitrary; in general, contours are named like algebraic parameters (see Section 2), it being essential only that each contour be given a unique identifier.

3.5. Specifying a mapping algorithm. With EPS, the symbols "x" and "y" are always used to denote a problem's independent variables, even when these variables do not represent Cartesian space coordinates (see Section 4), and the procedure for specifying a mapping algorithm always involves at least a command of the form

```
define x=f, y=g
```

where f and g are expressions which either explicitly or implicitly depend on grid coordinates "i" and "j". In the simplest cases, i,j-dependence can be shown explicitly, so that a single command of the form shown above is all that is necessary to establish the required mapping. For example, to define a regular, orthogonal lattice where the nodes are spaced 0.1 unit apart in the x-direction and 0.15 unit apart in the y-direction, and where the node with grid coordinates (0,0) corresponds to (x,y) = (0.3,-0.9), the user could type

```
define x=i/10+.3, y=.15*(j-6) $
```

In other cases it is convenient or necessary to show i,j-dependence in a more implicit manner, through the use of auxiliary parameters. For instance, a lattice representing the discrete counterpart of a cylindrical grid with spacing $\Delta r = 0.5$ and $\Delta\theta = 15°$ could be defined as follows:

```
define x=radius*cos(theta), y=radius*sin(theta),
   radius=.5*i, theta=j*pi/12
set pi=4*atan(1) $
```

Note that the define-command is used to indicate functional relationships while the set-command is used to specify constants.†

Both of the examples above represent very simple situations, thus are not completely germaine to many practical applications, where it is often necessary to introduce lattice irregularities in order to achieve accurate results. Irregularities may, of course, be of varying complexity, a particularly simple case being one in which the grid lines are straight and parallel to the x- and y-axes, but unevenly spaced (see Figure 2b). In instances where the grid-line spacing cannot be expressed algebraically in terms of "i" or "j", such configurations can be defined by associating with "x" and "y" vectors which give the desired spacing, as in the following sequence:

```
define x=xx(i), y=yy(j)
set xx(0)=.53, xx(1)=.77, xx(2)=1.12, ...,
    yy(0)=.20, yy(1)=.42, yy(2)=.75, ... $
```

†Under normal circumstances the computer prints no explicit response in connection with define- and set-commands, but indicates completion of these commands simply by printing another "PROCEED:". This is also true of several other commands and can be assumed to be the case in general when the text makes no mention of a response.

Similarly, it is possible to associate "x" and "y" with matrices so that the coordinates of each node can be specified individually:

```
define x=xx(i,j), y=yy(i,j)
set xx(0,0)=5.2, yy(0,0)=3.8, xx(1,0)=6.3, yy(1,0)=3.6,
    xx(2,0)=7.3, yy(2,0)=3.5, ... $
```

The typing involved in specifying the components of vectors and matrices can be reduced considerably in comparison to that shown above through use of the EPS macro facility (see especially Section 9.4). However, representing "x" and "y" with matrices is demanding at best. The user tempted to use such a procedure is urged therefore to note that it is frequently possible to have the computer generate internal node coordinates automatically by interpolating between values specified along the

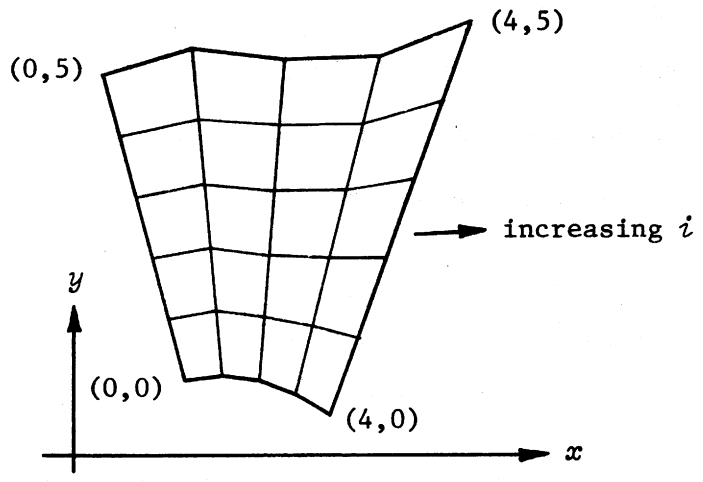


Fig. 4. *Lattice with linearly interpolated nodal distribution.*

lattice boundary. For example, consider linear interpolation, as illustrated by the lattice of Figure 4. This lattice could be specified by a sequence such as

```
define x=x0(i)+j*(x5(i)-x0(i))/5,
       y=y0(i)+j*(y5(i)-y0(i))/5
set x0(0)=1.3, y0(0)=1.0, x0(1)=1.8, y0(1)=1.0,
    x0(2)=2.4, y0(2)=0.9, ... $
```

this being the discrete counterpart of a two-dimensional ruled sheet transformation, applicable to domains which have two or more non-adjacent straight sides. A more general interpolation scheme, applicable to arbitrary domains, can easily be inferred from the surface equations discussed by Coons [7].

As a final point it should be noted that when a lattice assumes distinctly different forms in different regions, it is possible and sometimes convenient to introduce branches in the mapping algorithm so that mechanisms corresponding to the different regions can be specified independently. In general this is accomplished by using relational and Boolean operators (see Table 3, Section 16) to form expressions which take on different discrete values for different ranges of "i" and "j". Suppose, for example, that an orthogonal lattice is desired in which

$$x = .5\ i, \quad y = \begin{cases} .25\ j & \text{for } j \le 8, \\ .5\ (j\text{-}8) + 2 & \text{for } 8 \le j \le 20, \\ .75\ (j\text{-}20) + 8 & \text{for } j \ge 20. \end{cases}$$

This lattice can be defined in terms of the relational operator "grt", which in an expression of the form "α grt β" produces the value 1 when $\alpha > \beta$ and the value 0 otherwise. Specifically, we can type

```
define x=.5*i, y=yy((j grt 8)+(j grt 20)), yy(0)=
   .25*j, yy(1)=.5*(j-8)+2, yy(2)=.75*(j-20)+8 $
```

This example, chosen for convenience, is not meant to encourage the use of lattices which change character abruptly (see recommendations of Section 3.2). The technique it demonstrates is perhaps most useful in situations where parts of a lattice can be defined in a simple, algebraic manner, while other parts require tabular definitions. This is the case, for example, in the lattice of Figure 2a.

4. Equation specification

4.1. Admissible forms. The difference equation generator employed by EPS models linear systems of partial differential equations given in terms of dependent variables u_1, u_2, ..., u_n and independent variables x and y as follows:

$$\sum_{\ell=1}^{n} \left(\frac{\partial}{\partial x} \left(a_{k\ell} \frac{\partial u_\ell}{\partial x} + b_{k\ell} \frac{\partial u_\ell}{\partial y} + c_{k\ell} u_\ell \right) + \frac{\partial}{\partial y} \left(d_{k\ell} \frac{\partial u_\ell}{\partial x} + e_{k\ell} \frac{\partial u_\ell}{\partial y} + f_{k\ell} u_\ell \right) + g_{k\ell} u_\ell \right) = h_k \qquad [k = 1,2,\ldots,n] \tag{1}$$

Here the $a_{k\ell}$, $b_{k\ell}$, ..., $g_{k\ell}$ and h_k represent constants or functions of x and y which, along with the number n, must be specified by the user to indicate the particular system of interest. Note, for example, that Eq. (1) reduces to Laplace's equation in Cartesian coordinates,

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = 0, \tag{2}$$

when

$$\begin{gathered} u_1 \rightarrow \Phi, \\ n = 1, \\ a_{11} = e_{11} = 1, \\ b_{11} = c_{11} = d_{11} = f_{11} = g_{11} = h_1 = 0. \end{gathered} \tag{3}$$

It should be emphasized that the independent variables x and y in Eq. (1) may denote position in any right-handed coordinate system. Thus Eq. (1) corresponds to Laplace's equation in cylindrical coordinates,

$$\frac{\partial}{\partial r}\left(r\,\frac{\partial \Phi}{\partial r}\right) + \frac{1}{r}\,\frac{\partial^2 \Phi}{\partial \theta^2} = 0, \tag{4}$$

when

$$\begin{gathered} x \to r, \quad y \to \theta, \quad u_1 \to \Phi, \\ n = 1, \\ a_{11} = x, \quad e_{11} = 1/x, \\ b_{11} = c_{11} = d_{11} = f_{11} = g_{11} = h_1 = 0. \end{gathered} \tag{5}$$

Such a particularization is admissible, of course, since the $a_{k\ell}$, $b_{k\ell}$, etc. may vary with position; as a result, the user is free to employ whatever coordinate system best suits his problem. More importantly, he may deal with physical media whose properties change from point to point.

It is important to note that while the EPS difference equation generator models linear differential systems, solutions to nonlinear systems may frequently be obtained without difficulty. In essence this is done by having the computer produce and solve a sequence of linear problems whose spatially varying coefficients represent successively improving approximations to the solution-dependent coefficients for the nonlinear problem of interest. In this section we are, in fact, concerned primarily with the solution of linear equations, it being desired at this point only to indicate the feasibility and general character of the procedures needed for nonlinear applications. Further details and examples of such applications are given in Section 10.

4.2. Specification of parameters. The quantities n, $a_{k\ell}$, $b_{k\ell}$, etc. which indicate the particularization of Eq. (1) of interest are,

like the lattice parameters "x" and "y", specified through use of the set- and define-commands. As mentioned earlier, "set" is used for the specification of constants while "define" is used to specify functional relationships, e.g., to show how equation coefficients change with position. Before specific examples are discussed, the following remarks should be noted.

(1) When $n > 1$, equations should be ordered so that in the k^{th} equation u_k is the "dominating" dependent variable. Specifically, it is mandatory that u_k appear in the k^{th} equation, i.e., that at least one of the a_{kk}, b_{kk}, ..., g_{kk} be non-zero, and it is desirable that the following inequality be satisfied:

$$|a_{kk} + e_{kk}| \geq |a_{k\ell} + e_{k\ell}| \qquad [k,\ell = 1,2,\ldots,n]$$

(2) As Section 3.5 implies, EPS uses standard algebraic compiler notation for subscripted variables. Thus a_{23} is typed "a(2,3)" and h_2 as "h(2)".

(3) When all components of a subscripted variable are identical, subscript notation may be dropped and all components specified at once. For example, if coefficient $c_{k\ell} = 0$ for all k and ℓ, one may simply "set c = 0". This means, in particular, that when $n = 1$, subscripts may be omitted for all equation parameters.

(4) Spatially varying equation parameters can be specified in terms of grid coordinates "i" and "j" and/or in terms of any quanti-

ties which depend on "i" and "j", e.g., "x" and "y". Moreover, the functional behavior of a parameter can be given in algebraic, tabular, or mixed algebraic and tabular form. In any case, it is important to recognize that a discretely varying quantity such as "x" will be modeled as a continuous variable only when it operated on by the built-in interpolation function "fit", as in "fit(x)". We consider two common situations:

(a) *Algebraic expressions in x and y.* When one or more equation parameters are known algebraically in terms of the independent variables x and y, auxiliary variables representing "fit(x)" and "fit(y)" should be defined and used in place of x and y when the parameters in question are specified. For example, if $e_{22} = 3.5 + x^{.8}$ and $h_3 = x + \sqrt{y}$, type

```
define fx=fit(x), fy=fit(y), e(2,2)=
  3.5+fx power .8, h(3)=fx+sqrt(fy)
```

(b) *Tabular representations.* When an equation parameter $\alpha = \alpha(z)$, where z is a function available only in discrete form, say as a matrix "z(i,j)" with components which give z's values at the lattice nodes, then an auxiliary variable representing "fit(z(i,j))" should be defined and used in the specification of α:

```
define fz=fit(z(i,j)), α=α(fz)
```

Here "z(i,j)" could represent information determined by experimental measurement and specified component-by-component to the computer, or it could represent discrete information calculated by the computer, as

is the case in the solution of nonlinear equations (see Section 10.2).

(5) Discontinuous equation information is specified by using the relational and Boolean operators in a manner similar to that introduced in the discussion of lattice definition procedures in Section 3.5. Suppose, for example, that $h_2 = 5$ for $x < 7.2$ and $h_2 = 0$ for $x \geq 7.2$. This behavior could be specified in terms of the relational operator "les" (read "less than") by typing

```
define fx=fit(x), h(2)=hh(fx les 7.2)
set hh(0)=0, hh(1)=5
```

or, alternatively, by typing

```
define fx=fit(x), h(2)=5*(fx les 7.2)
```

As it happens, specifying discontinuous parameters in terms of "x" and "y" results in a certain amount of smoothing when lines of discontinuity do not coincide with grid lines. Moreover, for arbitrarily curved lines of discontinuity it is difficult to express discontinuous behavior in the manner shown above. These disadvantages may be avoided if, as recommended earlier, grid lines are made to coincide with lines of discontinuity. When this is the case, it is possible to express discontinuous parameters in terms of "i", "j", and a built-in (argumentless) function "octant", which, when difference equations are being formed, indicates lattice sector identity. Specifically, "octant" takes on integral values between 0 and

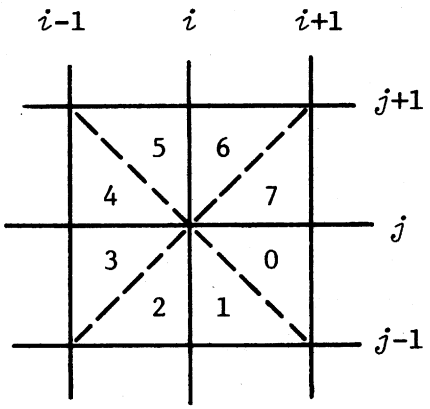


Fig. 5. *Numbering of lattice sectors.*

7, as shown in Figure 5, and may be used to differentiate between one side of a grid line and the other. For example, if h_2 = 5 for lattice elements below j = 20 (in i,j-space), and h_2 = 0 for elements above j = 20, one can type

```
define h(2)=hh((j les 20)+(j les 2 )),
   hh(1)=5*(octant les 4)
set hh(0)=0, hh(2)=5
```

This notation is tedious, but quite flexible and easily mastered with a little practice.

4.3. Examples. Shown below are input sequences corresponding to four special cases of governing system (1). Additional examples are given in Section 10.2.

A. Laplace's equation in Cartesian coordinates. The single command

```
set a=e=n=1, b=c=d=f=g=h=0 $
```

can be used to specify Laplace's equation in Cartesian coordinates. (See Eqs. (2) and (3).)

B. Laplace's equation in cylindrical coordinates. The command sequence

```
set n=1, b=c=d=f=g=h=0
define fr=fit(x), a=fr, e=1/fr $
```

can be used to specify Laplace's equation in cylindrical coordinates. (N.B., the independent variables are r and θ, not r and z; see Eqs. (4) and (5).)

C. Plate equation in Cartesian coordinates. The deflection $w = w(x,y)$ of a flat plate with variable stiffness $s = s(x,y)$ subjected to a pressure field $p = p(x,y)$, where x and y are Cartesian coordinates, satisfies

$$\text{div } (s \text{ grad } \nabla^2 w) = p$$

for sufficiently small p/s. This fourth-order system may be rewritten in the form of Eq. (1) on introduction of an auxiliary variable $\Phi = \nabla^2 w$, as follows:

$$\text{div } (s \text{ grad } \Phi) = p$$

$$\Phi - \nabla^2 w = 0$$

With Φ and w associated with u_1 and u_2, respectively, the command sequence to prescribe this system becomes

```
set n=2, a(1,2)=a(2,1)=b=c=d=e(1,2)=e(2,1)=f=g(1,1)=
    g(1,2)=g(2,2)=h(2)=0, a(2,2)=e(2,2)=-1, g(2,1)=1
```

```
define fx=fit(x), fy=fit(y), a(1,1)=e(1,1)= s(fx,fy),
   h(1)= p(fx,fy) $
```

(Note that *s(fx,fy)* and *p(fx,fy)* represent expressions to be filled in by the user.)

D. Plane stress equations expressed in terms of displacement variables, Cartesian coordinates. The Cartesian displacements $u = u(x,y)$ and $v = v(x,y)$ for a plane elastic continuum with constant material properties E and ν satisfy

$$\begin{aligned} &\frac{\partial}{\partial x}\left(\frac{E}{1-\nu^2} \left(\frac{\partial u}{\partial x} + \nu \frac{\partial v}{\partial y} \right)\right) + \frac{\partial}{\partial y}\left(\frac{E}{2(1+\nu)} \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)\right) = 0 \\ &\frac{\partial}{\partial x}\left(\frac{E}{2(1+\nu)} \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)\right) + \frac{\partial}{\partial y}\left(\frac{E}{1-\nu^2} \left(\nu \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)\right) = 0 \end{aligned} \tag{6}$$

when no stresses normal to the x,y-plane and no body forces are present. With u and v associated with u_1 and u_2, respectively, the command to specify this system becomes

```
set n=2, c=f=g=h=a(1,2)=a(2,1)=b(1,1)=b(2,2)=d(1,1)=
    d(2,2)=e(1,2)=e(2,1)=0, a(1,1)=e(2,2)= E/(1-ν*ν),
    b(1,2)=d(2,1)= ν*a(1,1), a(2,2)=b(2,1)=e(1,1)=
    d(1,2)= E/(2*(1+ν)) $
```

(Here E and ν represent numerical values to be filled in by the user.)

5. Specification of boundary conditions

5.1. Admissible forms. The EPS difference equation generator admits boundary conditions for governing system (1) which designate values for, or linear relationships between, the dependent variables u_ℓ and certain first-order functions F_k [$k = 1,2,\ldots,n$] of the u_ℓ. The F_k may not be elected by the user, but are restricted to a predetermined form which is natural and convenient for most applications (see Section 5.5). Specifically, the F_k are given in terms of the coefficients $a_{k\ell}$, $b_{k\ell}$, ..., $f_{k\ell}$ of governing system (1) and the local boundary slopes $\partial x/\partial s$ and $\partial y/\partial s$, arc length s being positive when the boundary is traced so that the domain lies to the right, as follows:

$$F_k = \sum_{\ell=1}^{n} \left(- \left(a_{k\ell} \frac{\partial u_\ell}{\partial x} + b_{k\ell} \frac{\partial u_\ell}{\partial y} + c_{k\ell}\, u_\ell\right) \frac{\partial y}{\partial s} + \left(d_{k\ell} \frac{\partial u_\ell}{\partial x} + e_{k\ell} \frac{\partial u_\ell}{\partial y} + f_{k\ell}\, u_\ell\right) \frac{\partial x}{\partial s} \right) \qquad [k = 1,2,\ldots,n] \tag{7}$$

Note that for Laplace's equation in Cartesian coordinates (see Eqs. (2) and (3)), definition (7) reduces to

$$F_1 = - \frac{\partial u_1}{\partial x} \frac{\partial y}{\partial s} + \frac{\partial u_1}{\partial y} \frac{\partial x}{\partial s} = \frac{\partial u_1}{\partial n} \rightarrow \frac{\partial \Phi}{\partial n}$$

where n represents the direction of the local outward normal. Thus, for this special case, boundary conditions must assume the form

$$\alpha(x,y) \frac{\partial \Phi}{\partial n} + \beta(x,y)\, \Phi = \gamma(x,y) \tag{8}$$

where either $\alpha(x,y)$ or $\beta(x,y)$ may become zero. More generally, for each boundary segment m, boundary conditions for governing system (1) must take the form

$$\sum_{\ell=1}^{n} (p_{k\ell m} F_\ell + q_{k\ell m} u_\ell) = r_{km} \qquad [k = 1,2,\ldots,n] \tag{9}$$

where the $p_{k\ell m}$, $q_{k\ell m}$, and r_{km} are constants or functions of position to be specified by the user.† The specification of boundary conditions therefore entails two distinct operations:

(1) assignment of unique segment numbers to portions of the boundary having different types of boundary conditions;

(2) specification of parameters $p_{k\ell m}$, $q_{k\ell m}$, and r_{km} for each segment number m to indicate the desired condition.

The specific input conventions by which these operations are effected are discussed below.

5.2. Assigning segment numbers. The boundary of the domain should be divided into segments, these corresponding to arcs over which the boundary conditions assume constant form, i.e., continuous behavior in terms of the p's and q's of Eq. (9). Each segment should then be assigned a number, say m, and identified to the computer in terms of

†Other forms, including nonlinear relationships between the F_ℓ and u_ℓ, can be accomodated by iteration. However, as explained in sec. 5.5, other forms seldom arise in physical applications, even when the governing equations are nonlinear.

the grid coordinates i_1,j_1 and i_3,j_3 of its end points plus the coordinates i_2,j_2 of some intermediate point, as follows:

impose m along i_1,j_1, i_2,j_2, i_3,j_3

The following remarks should be noted:

(1) Segment numbers may be arbitrary integers, but, to conserve memory, should be assigned in sequential fashion. Generally it is convenient to use the numbers 0,1,2,... (The number 0 is automatically assigned to segments not explicitly mentioned in impose-commands.)

(2) A given segment number may be used more than once if the same boundary condition applies to more than one portion of the boundary.

(3) The i_k,j_k may refer to arbitrary boundary nodes; they need not refer to the contour vertices discussed in Section 3.4.

(4) Intermediate coordinates i_2,j_2 are omitted in the case of a segment which joins neighboring boundary nodes.

These points are illustrated by the following command sequence, which could be used to assign the segment numbers shown in Figure 6.

```
impose 1 along 0,0, 1,0, 9,0
impose 2 along 0,0, 0,1, 6,7
impose 3 along 6,7, 7,7
impose 2 along 9,0, 9,3, 7,7
impose 4 along 3,3, 3,5, 3,3
```

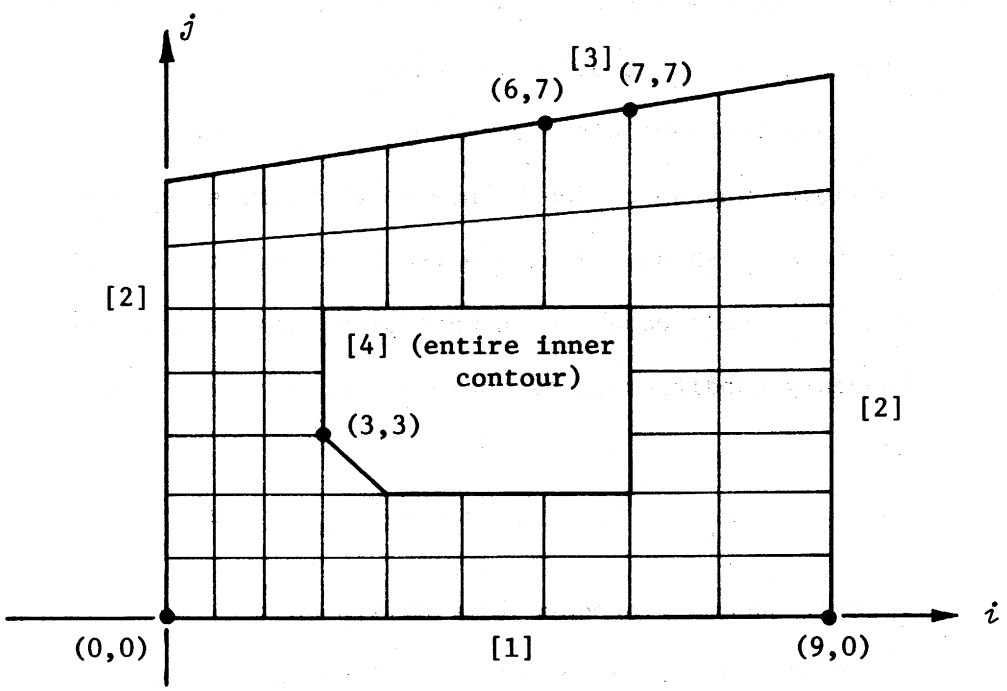


Fig. 6. *Sample lattice and segment numbers (in brackets). Ends of segments are indicated by large dots* ●.

5.3. Specification of parameters. The parameters $p_{k\ell m}$, $q_{k\ell m}$, and r_{km} of Eq. (9) are specified in exactly the same manner as the parameters $a_{k\ell}$, $b_{k\ell}$, ..., h_k of governing system (1) and, in particular, are subject to the same input conventions as described in points (2)-(5) of Section 4.2. Thus, for example, to specify $q_{123} = 1 + e^x$, type

```
define fx=fit(x), q(1,2,3)=1+exp(fx)
```

and so forth. Users with straightforward applications, especially problems with only one dependent variable ($n = 1$), may feel free at this point to skip to Section 5.4. For more advanced applications, the more detailed remarks below may be relevant.

(1) Because of the manner in which EPS models boundary conditions, greatest accuracy is achieved when the coefficients p_{klm} are constant along each boundary segment m. Thus, if possible, any space dependence which might naturally arise in the p_{klm} should be eliminated by division. For example, with Laplace's equation in Cartesian coordinates, where the boundary condition is as shown in Eq. (8) with $\alpha(x,y) \neq 0$, use

$$p_{11m} = 1, \quad q_{11m} = \beta/\alpha, \quad r_{1m} = \gamma/\alpha.$$

Clearly such a procedure is always possible for $n = 1$, and it is also possible for $n > 1$ so long as boundary conditions do not involve linear combinations of the F_k.

(2) When $n > 1$, equations that express boundary conditions should be ordered, if possible, so that for each boundary segment m and each equation k at least one of the coefficients p_{kkm} and q_{kkm} is non-zero. That is, if possible, the k^{th} equation should involve either F_k or u_k, or both. If for any m such an ordering is impossible, make certain that the boundary conditions are well posed, i.e., that they are self-consistent and sufficient to determine all of the u_k uniquely, then proceed with an ordering which maximizes the number of equations with non-zero q_{kkm}.

(3) At a node B which separates boundary segments with different segment numbers m_1 and m_2, the coefficients p_{klm} for each

equation k must satisfy at least one of the following conditions:

$$(p_{k\ell m_1})_B = 0 \qquad [\ell = 1,2,\ldots,n]$$

or

$$(p_{k\ell m_2})_B = 0 \qquad [\ell = 1,2,\ldots,n]$$

or

$$(p_{k\ell m_1})_B = (p_{k\ell m_2})_B \qquad [\ell = 1,2,\ldots,n]$$

Thus, for example, in boundary-value problems for Laplace's equation, boundary conditions may switch abruptly from Neumann conditions ($p_{11m_1} \neq 0$, $q_{11m_1} = 0$) to Dirichlet conditions ($p_{11m_2} = 0$, $q_{11m_2} \neq 0$), but not from "Neumann conditions to Neumann conditions" unless $p_{11m_1} = p_{11m_2}$ at the point of discontinuity. Again, for $n = 1$ there need be no conflict; in fact, the restriction cited here is satisfied automatically if boundary conditions are normalized so that the p_{11m} assume only values of unity or zero. However, when $n > 1$, conflicts may arise in rare cases. (If not anticipated by the user, they will be detected and reported by the computer at the time an attempt is made to generate difference equations.) In such cases, it is necessary to force the computer to employ the same segment number, say m_1, on both sides of the node in conflict. This is done by using a special form of the impose-command

impose m_1 at i_B,j_B

which effectively extends segment m_1 half-way to B's neighbor on the "m_2-side." In most instances, this effect may be counteracted by proper arrangement of nodes, but, at worst, it merely introduces an additional discretization error.

(4) Boundary condition parameters may be expressed in terms of boundary slopes $\partial x/\partial s$ and $\partial y/\partial s$ through use of the built-in (argumentless) functions "xs" and "ys", respectively. Thus if $r_{12} = (\partial x/\partial s)/y$, type

```
define fy=fit(y), r(1,2)=xs/fy
```

5.4. Example. Consider the solution of Laplace's equation in terms of the lattice and boundary conditions illustrated in Figure 7.

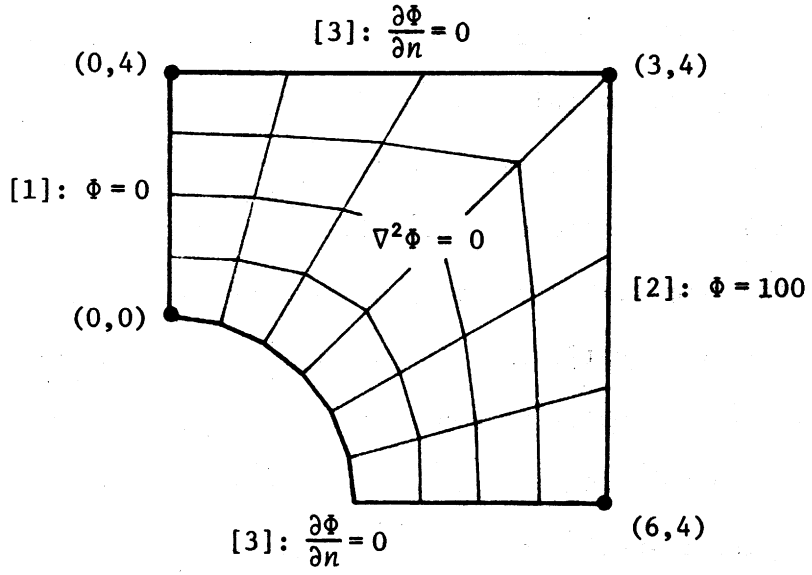


Fig. 7. *Sample lattice and boundary conditions for Laplace's equation.*

The boundary conditions shown here can be specified as follows:

```
impose 1 along 0,0, 0,2, 0,4
impose 2 along 3,4, 4,4, 6,4
impose 3 along 0,4, 2,4, 3,4
impose 3 along 0,0, 5,0, 6,4
set p(1,1,1)=p(1,1,2)=q(1,1,3)=r(1,1)=r(1,3)=0,
    q(1,1,1)=q(1,1,2)=p(1,1,3)=1, r(1,2)=100    $
```

Significant reductions in the amount of typing required to specify boundary conditions can be realized by means of appropriate macros. See Sections 9.1 and 9.7.

5.5. Supplementary remarks. For users with applications involving governing expressions other than Laplace's equation, an explanation regarding the fixed form of the functions F_k shown in Eq. (7) may be desirable. For this purpose, it is helpful to consider the special case in which the independent variables x and y represent Cartesian coordinates. This makes it possible to express the local unit outward normal to the boundary in terms of boundary slopes $\partial x/\partial s$ and $\partial y/\partial s$ and unit vectors $\bar{i}$ and $\bar{j}$, corresponding to the x- and y-direction, respectively, as follows

$$\bar{n} = -\bar{i}\,\frac{\partial y}{\partial s} + \bar{j}\,\frac{\partial x}{\partial s}$$

and suggests that the F_k be written as vector dot products

$$F_k = \bar{F}_k \cdot \bar{n} \qquad (10)$$

where the vectors $\bar{F}_k$ are defined by

$$\bar{F}_k = \bar{i}\left(\sum_{\ell=1}^{n}\left(a_{k\ell}\frac{\partial u_\ell}{\partial x} + b_{k\ell}\frac{\partial u_\ell}{\partial y} + c_{k\ell}\,u_\ell\right)\right) + \bar{j}\left(\sum_{\ell=1}^{n}\left(d_{k\ell}\frac{\partial u_\ell}{\partial x} + e_{k\ell}\frac{\partial u_\ell}{\partial y} + f_{k\ell}\,u_\ell\right)\right) \qquad [k = 1,2,\ldots,n] \tag{11}$$

Definition (11), in turn, suggests a more succinct statement of governing system (1), viz.,

$$\operatorname{div}\bar{F}_k + \sum_{\ell=1}^{n} g_{k\ell}\,u_\ell = h_k \qquad [k = 1,2,\ldots,n] \tag{1'}$$

This will be seen to resemble closely expressions commonly used in the representation of physical laws of continuity and equilibrium. Thus, in physical applications the vectors $\bar{F}_k$ are associated with mass flow vectors, current density vectors, stress dyadics, and so forth, while the related functions F_k represent boundary fluxes, currents, mechanical tractions, and the like. From this we may draw two important conclusions:

(1) the F_k are, at least in physical problems with Cartesian coordinates, exactly the quantities with which we normally prefer to deal, and

(2) the physical significance of the F_k can frequently be deduced without close scruntiny of the formal definition given in Eq. (7).

In fact, it will be found that these conclusions are valid, with minor qualifications, for any orthogonal curvilinear coordinate system.

Moreover, they are generally valid regardless of whether the governing system being treated is linear or nonlinear. Nonlinearities in physical problems normally arise in the constitutive laws which relate vectors $\bar{F}_k$ (or their counterparts in non-Cartesian reference frames) to the dependent variables u_ℓ, not in the equations which govern the actual behavior of the $\bar{F}_k$.

A final example, which illustrates point (2) above, can be given in terms of the plane stress equations introduced in Section 4.3 (Example *D*, Eq. (6)). These equations merely state the equilibrium conditions

$$\begin{aligned} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} &= 0 \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= 0 \end{aligned} \tag{12}$$

for two-dimensional Hookean elasticity. Comparing Eq. (12) to Eqs. (10) and (1'), we can deduce immediately

$$F_1 = (\bar{i}\, \sigma_{xx} + \bar{j}\, \tau_{yx}) \cdot \bar{n} = \text{boundary traction in } x\text{-direction}$$

$$F_2 = (\bar{i}\, \tau_{xy} + \bar{j}\, \sigma_{yy}) \cdot \bar{n} = \text{boundary traction in } y\text{-direction}$$

Accordingly, we see that the general form for boundary conditions for problems with two dependent variables, i.e.,

$$p_{11m} F_1 + p_{12m} F_2 + q_{11m} u_1 + q_{12m} u_2 = r_{1m}$$

$$p_{21m} F_1 + p_{22m} F_2 + q_{21m} u_1 + q_{22m} u_2 = r_{2m}$$

becomes, in the case of plane elasticity problems, a pair of linear equations relating the boundary tractions and displacements. These equations may be specialized so as to designate simple traction and displacement conditions, or may be used in a more general fashion to treat boundaries on elastic foundations, and so forth.

6. Getting a solution

6.1. Generation of finite-difference equations. Once a lattice and differential system have been specified as indicated in Sections 3, 4, and 5, or revised as indicated in Sections 8 and 10, finite-difference equations can be obtained simply by typing

```
form $
```

Assuming there are no detectable inconsistencies in the user's data and difference equations are being obtained for the first time, this command will result in an immediate response of the form

```
POINT TALLY IS N.
SPACE FOR SOLUTION MATRIX HAS BEEN ALLOTTED AND ZEROED.
```

where N is the total number of nodes in the lattice. When equations are being regenerated after a problem revision, the response may be abbreviated or omitted entirely. In any event, a pause will ensue which may, in fact, be imperceptible or last for several minutes, depending on problem complexity and computer load. Eventually a PROCEED-message will be printed, at which point it may be assumed that all difference equations have been formed.

6.2. Solution of finite-difference equations. EPS solves difference equations generated by means of the form-command by successive relaxation,† using a relaxation factor "omega", a tolerance parameter

†See, for example, Young [8], sec. 11.5.

"delta", and a maximum iteration index "limit" specified by the user. Relaxation factor "omega" has the usual significance, thus may be assigned a value between 0 and 1 for underrelaxation or a value greater than 1 for overrelaxation, according to the situation at hand (see below). Relaxation, once initiated, continues until either

(1) the absolute changes introduced in all components of the solution matrix during a given sweep of the lattice come to be less than or equal to "delta", or

(2) the number of sweeps reaches "limit".

After this, it is possible to request further relaxation, perhaps with different values for "omega", "delta", and "limit", or to proceed immediately with requests for output. It is also possible to intersperse requests for output with requests for further relaxation in order, for example, to monitor the convergence of individual components of the solution

Parameters "omega", "delta", and "limit" are specified by means of the set-command, as in

```
set omega=1.6, delta=2e-5, limit=200
```

while relaxation is initiated or resumed by typing

```
relax $
```

The relax-command always results, after one or the other of the termination conditions has been satisfied, in a response of the form

RELAXATION TERMINATED AFTER *N* PASSES. MAX SOLN CHANGE: *C*.

where N is the number of sweeps of the lattice actually performed and C is the $\max_{k,i,j} |\Delta u(k,i,j)|$ for the Nth sweep.[†] Note that either N will equal "limit" or C will be less than or equal to "delta".

Selection of "omega". In general, there is for any set of difference equations a maximum relaxation factor ω_{max} beyond which successive relaxation diverges. In many common applications ω_{max} is 2, while convergence is most rapid for some optimum relaxation factor between 1 and 2. Thus 1.5 is often a reasonable first choice for "omega". For lattices with many nodes (say more than 300) or in cases where a number of related problems are being treated, it may be worthwhile to attempt to estimate the optimum relaxation factor by experiment. Normally this can be done by setting "limit" to some relatively small number and observing the rate of decrease of the "MAX SOLN CHANGE" for different values of "omega".[§] Such a procedure may also be required in certain cases where divergence occurs for values of "omega" in the normal range. It has been found, for example, that ω_{max} is less than 2 for certain plate problems with clamped-edge boundary conditions (see Section 4.3, Example *C*) and considerably less than 1 for certain nonlinear systems arising in hydrodynamics.

[†]Here, as in actual computer input, components of the solution matrix are denoted by "u(k,i,j)", meaning the current approximation for dependent variable u_k at node (i,j).

[§]Convergence rates are often dramatically dependent on the relaxation factor. However, for lattices with no more than a few hundred nodes, the actual computation time required for relaxation with any "reasonable" factor is typically much less than that required for other operations involved in obtaining a solution, e.g., the generation of difference equations. In such cases there is little to be gained on a percentage basis by seeking the optimum factor.

Selection of "delta". It is emphasized that tolerance parameter "delta" can be used only to control errors in the discrete solution $u(k,i,j)$ that are associated with incomplete relaxation; its value has no bearing whatever on the discretization errors incurred by replacing a continuous domain with a discrete model. Moreover, "delta" is only indirectly related to relaxation error, its strict function being to specify a maximum allowable value for the "MAX SOLN CHANGE" computed during a given sweep of the lattice. Normally the maximum change occurs in a component of the solution matrix which itself has a relatively large absolute value. Thus assigning "delta" some value d means loosely that the relaxation should be terminated if and when the largest components of the solution matrix are changing by amounts no greater than d. It does not mean, however, that when the relaxation is terminated on this basis, the largest components will be "good to $\pm d$", for convergence to the true discrete solution may occur in steps which are much smaller than the actual error. For this reason, "delta" should generally be given a value which is one or two orders of magnitude less than the relaxation error considered tolerable in the largest solution components.[†] Note that convergence is normally fairly uniform with

---

[†]This is a rule of thumb which should be appropriate for most applications. However, more conservative, i.e., smaller, values for "delta" may be advisable if convergence is abnormally slow, as when difference equations are ill-conditioned and/or it is necessary to resort to small relaxation factors, e.g., less than 0.5. The ratio of relaxation step size to actual error is, of course, the critical variable in all cases. This can be estimated, if doubts arise, by observing the rate of decrease of the "MAX SOLN CHANGE" as the relaxation progresses. For example, if 80 sweeps are required to reduce this number by an order of magnitude, it is reasonable to assume that the reported solution changes are about 1/8 the size of the actual errors due to incomplete relaxation.

respect to all components of the solution, i.e., fractional rates of change in the smaller solution components can be expected to be about the same as those in the larger components.

Selection of "limit". Parameter "limit" has two intended functions: (1) to prevent a nonconvergent relaxation from continuing indefinitely, and (2) to permit close monitoring of the relaxation, as is necessary in estimating optimum values for "omega", etc. Here it suffices to say that fully relaxed solutions, i.e., solutions for which further improvement is impossible because of the effects of computational round-off error,† can be obtained in typical applications with a number of sweeps which is about the same as the number of nodes in the lattice. As a rule, then, "limit" should not be given a value much larger than the number of nodes, although this may mean in some cases that additional relax-commands are needed to reach full convergence.

---

†If full convergence is sought, it will be observed that, after decreasing steadily for a time, fractional solution changes (and hence reported values for the "MAX SOLN CHANGE") ultimately reach a lower limit beyond which further relaxation is fruitless. In some cases this limit is about the same as the round-off error size intrinsic to the computer, i.e., about 10^{-8} on the IBM 7094. However, it is not uncommon for the limit to be larger, e.g., 10^{-5} or 10^{-6}, especially with lattices of several hundred nodes.

7. Output

EPS provides both printed and graphical output representing the following quantities:

(1) solution components u(k,i,j);

(2) numerically evaluated first and second derivatives ux(k,i,j), uy(k,i,j), uxx(k,i,j), uxy(k,i,j), uyx(k,i,j), and uyy(k,i,j);

(3) numerically evaluated boundary fluxes flux(k,i,j) corresponding to the F_k of Eq. (7); and

(4) arbitrary functions of any of the above as well as of other quantities, not specifically related to a finite-difference solution, which the user has defined.

The particular commands by which output is obtained are described in the two sections which follow.

7.1. Printed output. Small quantities of printed output are obtained by using commands of the form

print e_1, e_2, ..., e_N

where the e_k are algebraic expressions whose values are to be printed in a standard scientific (E-) format. For example, to determine current numerical approximations for u_2 and $(\partial u_3/\partial y)/(\partial u_3/\partial x)$ at node (7,9), type

```
print u(2,7,9), uy(3,7,9)/ux(3,7,9) $
```

The computer will respond with a line of the form

±X.XXXXXXXE±XX ±X.XXXXXXXE±XX

assuming that the indicated quantities are defined (see Sections 14 and 15).

Larger quantities of output, e.g., tables representing solution properties over a wide range of i,j-values, can be obtained by embedding a print-command in one or more "do-loops". In general, an input sequence of the form[†]

do(s for v=v_1 step δv until v_2)

can be used to cause repetition of any command sequence s while some parameter v is incremented by δv from v_1 to v_2 (inclusively). Thus to cause printing of u_2, $\partial u_2/\partial x$ and $\partial u_2/\partial y$ along the grid line j = 5, for i = 1,2,...,20, type

```
do(print u(2,i,5), ux(2,i,5), uy(2,i,5) for i=1
   step 1 until 20)$
```

This will result in the printing of 20 lines of output, each containing three numbers in the same format as shown in the preceeding paragraph. Similarly, to cause printing of u_1 and u_2 along the grid lines j = 2,4, 6,8,...,20, for i = 8,9,...,11, type

---

[†]This input form does not represent a regular EPS command, but is transformed automatically into an appropriate sequence of regular commands in accordance with a macro definition which has been prespecified in both versions of EPS. See sec. 9.3.

```
do(do(print u(1,i,j), u(2,i,j) for i=8 step 1 until 11)
   for j=2 step 2 until 20) $
```

In the last two examples, no output of indices "i" and "j" is requested, thus none is produced. For small tables of output this may be appropriate, but in other cases it is convenient to include "looping indices" in lists of items to be printed. As a completely independent matter, it is recommended that when complicated functions are being printed, they be defined beforehand as auxiliary parameters, so the auxiliary parameter names can be used in actual print-requests.† For example, if in solving a plane stress problem (see Eqs. (6) and (12)) numerical values for the stress components σ_{xx}, σ_{yy}, and τ_{xy} are desired along the grid line i = 3, for j = 30,29,28,...,5, type

```
define sx= E*(ux(1,i,j)+ν*uy(2,i,j))/(1-ν*ν),
       sy= E*(uy(2,i,j)+ν*ux(1,i,j))/(1-ν*ν),
       tau= E*(ux(2,i,j)+uy(1,i,j))/(2*(1+ν))
set i=3
do(print j,sx,sy,tau for j=30 step -1 until 5)$
```

Parameters "sx", "sy", and "tau" could be used in any further requests without being redefined.

---

†The generation of large tables of printed output can consume considerable computation time, even as much as 0.5 second per line when complicated functions are involved. Using auxiliary parameter names in print-requests, rather than whole expressions, is one way of reducing output time requirements. Further savings can be realized by specifying and using looping macros which are less general, therefore more efficient, than the do-macro already provided. However, getting printed output with EPS is more expensive at best than, say, the user accustomed to batch processing with FORTRAN might expect. Discretion and selectivity are therefore advisable.

While the standard format used by EPS for printed output is capable of representing with full accuracy any number that EPS can produce, it is by no means an ideal format in all situations. For this reason, a special form of the print-command is recognized which allows the user to override the standard format with one of his own design. Specifically, to indicate that output is to be printed according to a MAD-language format specifier f, type

print e_1, e_2, ..., e_N using bcd $\lambda f \lambda$

where λ is any single character not present in f.[†§] An example is provided by the sequence

```
set i=j=7
print x, y, u(2,7,7), flux(2,7,7) using bcd /4f9.4*/ $
```

which would cause to be printed a line of the form

X.XXXX X.XXXX X.XXXX X.XXXX

assuming all results are positive and less than 10. The standard output

---

[†]MAD format conventions, described in [9], sec. 2.15, are virtually identical to those of FORTRAN. N.B., with EPS, output lines may contain no more than 83 characters, format variables are not allowed, and carriage control characters are meaningless (the first characters of output lines are printed literally).

[§]Expressions of the form "bcd $\lambda s \lambda$", called literals, must not be split up over more than one physical input line. The computer interprets the character λ, which, by definition, is the second character following reserved symbol "bcd", as a logical quotation mark, and treats string s as a single (logical) word. Literals provide an escape mechanism which can be used not only in conjunction with format specifiers, but any time it is desired to have treated as a single word a character string which would normally be illegal or, at any rate, not interpreted as a single word. See sec. 15.1.

format allows for a maximum of five numbers per line, overflow being carried to additional lines when print-commands specifying more than five expressions are employed. Obviously an overriding format, specifying smaller field widths and more numbers per line, can be introduced if the user desires.

7.2. Graphical output. When used in conjunction with the ESL display equipment, EPS (as represented by the disk file "eps-p saved") will accept a command of the form

```
plot z
```

where z is an expression giving the elevation at node (i,j) of a surface to be represented in graphical form. For example, to obtain a display of the finite-difference solution for dependent variable $u_1 = u_1(x,y)$, type

```
plot u(1,i,j)$
```

Similarly, to see $[(\partial u_2/\partial x)^2+(\partial u_2/\partial y)^2]^{.5}$, type

```
plot sqrt(ux(2,i,j)power 2+uy(2,i,j)power 2)$
```

Surfaces are represented as orthographic projections of the three-dimensional nets produced by moving the nodes of a finite-difference lattice out of the x,y-plane by amounts proportional to argument z (see Figure 8). Projection planes can be varied continuously, i.e., a surface can be made to appear to rotate in space, by means of the plexiglass globe adjacent to the display unit. Rotating the globe

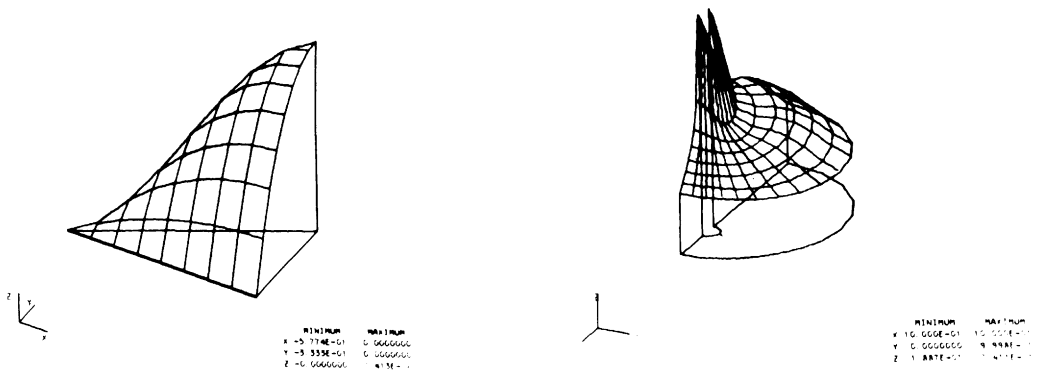


Fig. 8. *Typical displays (photographically reversed) produced by means of the plot-command*

about one or more of its three axes causes the surface to rotate in a corresponding manner. Releasing the globe stops the rotation.†

The user will find that the first display he requests appears with a "top-view" orientation, i.e., it shows the surface as seen by looking down the z-axis. Each subsequent display, on the other hand, appears with the same orientation as that in which the previous display was left unless an explicit command, namely,

```
plot nil$
```

is given to cause all rotation angles to be reset to their initial values. The above command represents one of several special forms available for altering the normal operation of the display mechanism. Other forms can be used, for example, to suppress certain picture elements, inhibit automatic scaling, etc. For details, see Section 15.

---

†In addition to the globe, the user will find near the display unit a control panel containing three large knobs. These knobs can be used to center the display or to change its magnification, although it is normally unnecessary to make such adjustments.

8. Data modification

Previous sections of this document introduce five commands used to specify problem input. These commands can be grouped conveniently as follows:

(1) the contour description commands, "append" and "close", used to define lattice structure in i,j-space,

(2) the parameter description commands, "set" and "define", used to specify various lattice, equation, boundary condition and relaxation parameters, and

(3) the segment identification command, "impose", used to number boundary arcs having different types of boundary conditions.

The present section describes how data associated with each of these command groups can be modified, either for purposes of error correction or to introduce problem variations.

Two general comments should be noted at the outset. First, it should be recognized that with EPS data can be modified in a very specific manner. The system always maintains intact information which is not explicitly involved in a modification. Thus it is seldom necessary to "start over from the beginning" when introducing a correction or problem variation; it is only required that information which is actually changing be respecified. Second, it should be noted that while data can be modified at any time, the effects of a modification on finite-difference equations and solution components can be realized only as a result of

explicit user requests. As indicated earlier, EPS does not actually assemble data to produce a solution until expressly directed to do so. This means that the user can make corrections as he sees fit during the course of a problem description without unnecessary loss of computer time, but it also means that he must remember to be explicit after introducing a problem variation lest he be misled into thinking that he has a new solution when he has not. A misimpression of this sort is made possible by the fact that "current" solutions are deleted by EPS only when major problem revisions, namely, changes to the lattice structure in i,j-space or to the number of differential equations n, are requested. Thus, to consider a problem variation, introduce all changes of interest, then type form- and relax-commands, as described in Section 6, to obtain a new solution. If no major changes are introduced, the relaxation routine will use the previous solution as its starting point; it will also use previous values for parameters "omega" "delta" and "limit" unless new ones are specified.

8.1. Modification of lattice structure in i,j-space. Data relating to lattice structure in i,j-space can be modified in three ways:

(1) New contours can be defined, e.g., to introduce holes in an existing domain, by typing additional append- and close-commands, as discussed in Section 3.4.

(2) An existing contour *curvename* can be deleted in its entirety, e.g., to eliminate a hole or facilitate some other

gross lattice revision, by typing

delete *curvename*

The computer's response to such a command has the form

'*CURVENAME*' HAS BEEN DELETED.

(3) An arc of contour *curvename*, beginning at i_1,j_1, passing through some intermediate point i_2,j_2, and ending at i_3,j_3, can be deleted by typing

delete i_1,j_1, i_2,j_2, i_3,j_3 from *curvename*

Then a replacement arc can be specified and *curvename* reclosed by typing additional append- and close-commands. For example, a new grid line, i=8, can be introduced in a lattice bounded by a single contour "alpha" with vertices 0,0, 7,0, 7,7 and 0,7 by means of the sequence

```
delete 7,7, 7,2, 7,0 from alpha
append 7,0, 8,0, 8,7 to alpha
close alpha $
```

The computer's response in this case would be

```
DELETION FROM 'ALPHA' HAS BEEN COMPLETED.
ADDITION TO 'ALPHA' HAS BEEN COMPLETED.
'ALPHA' HAS BEEN CLOSED.
```

The user should note that

(a) none of the i_k,j_k of the delete-command need necessarily coincide with nodes which are contour vertices

in the i,j-plane;

(b) when the append-command is used to append a new arc onto an existing curve, it is necessary that the new arc and the existing curve have at least one end point in common; and

(c) any boundary segment numbers associated with deleted arcs must be respecified after replacement arcs have been introduced. (Replacement arcs are automatically assigned a segment number of zero.)

8.2. Parameter modification. The modification of quantities such as

(1) lattice parameters x and y,

(2) equation parameter n and the a(k,l), b(k,l), ..., g(k,l) and h(k),

(3) boundary condition parameters p(k,l,m), q(k,l,m) and r(k,m), and

(4) relaxation parameters omega, delta and limit,

as well as any auxiliary parameters which the user may have introduced, can be accomplished in a natural and obvious manner, namely, by reusing the set- and define-commands. For example, if it is decided that coefficient a(2,2) should assume the constant value 2.5, type

```
set a(2,2)=2.5$
```

regardless of how a(2,2) was specified in the past. The older definition will be overwritten without any explicit computer response. Users with advanced applications or individuals wishing to use EPS as an online programming language should note the following points:

(1) Information relating to parameter type, such as the number and range of subscripts (DIMENSION information, in the FORTRAN sense) or the distinction between constants and variables, is updated automatically whenever a parameter is respecified, and it is permissible not only to change a variable into a constant or vice-versa, but also to change a scalar into an array or vice-versa.

(2) Parameters cannot be deleted entirely in current versions of EPS, but significant amounts of storage can be recovered by using set-commands to change unneeded variable or subscripted parameters into scalar constants. Obviously, such steps are essential only in applications where storage limitations are a problem.

8.3. Modification of segment numbers. Like parameters, segment numbers can be modified simply by reusing the command by which they are specified in the first place. Thus to assign a new segment number, type an appropriate impose-command, as described in Section 5.2. The old segment number (or numbers) will be overwritten without explicit response.

9. Macros

9.1. Introduction. In order to make possible the abbreviation of frequently used input sequences, EPS provides a macro facility similar to those of many machine language assemblers and some high-level compilers. With EPS, a macro, or input substitution rule, can be specified in terms of an identifier *macroname*, a set of formal parameters $\{p_k\}$, a set of separators $\{s_k\}$, and a definition $d(p_1;p_2;\ldots;p_N)$ as follows:†

let *macroname*$(p_1\ s_1\ p_2\ \cdots\ s_{N-1}\ p_N)$ mean
begin
$d(p_1;p_2;\ldots;p_N)$
end

(The notation $d(p_1;p_2;\ldots;p_N)$ is not meant to imply any particular input form, but merely represents an arbitrary input sequence in which the formal parameters (may) appear.) The computer responds to such a command by printing

INDICATED SUBSTITUTION FOR '*MACRONAME*' WILL BE MADE.

and, in processing subsequent input, replaces each call of the form

macroname$(a_1\ s_1\ a_2\ \cdots\ s_{N-1}\ a_N)$

where the a_k are actual parameters, or arguments, by an instance of the macro definition

†This is the normal form of the let-command. Two minor variations, discussed in sec. 9.4, are also permitted.

$$d(a_1;a_2;\ldots;a_N)$$

in which each occurrence of each p_k is replaced by a literal copy of the corresponding a_k.

Before going into further details, we consider a specific example, namely, a macro to facilitate the specification of boundary values for problems with one dependent variable (see Section 5). Such a macro can be specified in terms of an auxiliary parameter "m" as follows:

```
let prescribe(value along arc) mean
    begin
    set m=m+1, p(1,1,m)=0, q(1,1,m)=1
    define r(1,m)=value
    impose m along arc
    end
```

Then, for example, typing a call such as

```
prescribe(sqrt(fx) along 0,0, 2,0, 9,0)
```

becomes completely equivalent to typing the command sequence

```
set m=m+1, p(1,1,m)=0, q(1,1,m)=1
define r(1,m)=sqrt(fx)
impose m along 0,0, 2,0, 9,0
```

(It is necessary, of course, to assign "m" an initial value before making the first call.) Comparing this example with the general form of the preceeding paragraph, we see that the words "value" and "arc" are used as formal parameter names and the word "along" as a separa-

tor.[†] In general, formal parameters and separators may be arbitrary elements,[§] other than parentheses and dollar signs. Thus, highly readable constructions, such as the one given here, can be easily achieved.[¶]

As the reader will recognize, macros are basically functions which operate in a somewhat broader domain than the functions of ordinary algebra. In fact, the only essential difference between macros, as implemented in EPS, and ordinary functions, stems from the fact that macro arguments are substituted literally, without binding or pre-evaluation. Thus, for example, the macro specified by

```
let f(x,y) mean begin x/y end
```

is not totally equivalent to the function given by

$$f(x,y) = x/y,$$

since a call for the macro made in terms of arguments "a+b" and "c+d" is interpreted according to

```
f(a+b,c+d)  →  a+b/c+d
```

whereas the corresponding function value is

$$f(a+b,c+d) = (a+b)/(c+d).$$

Accordingly, certain precautions are necessary when dealing with macros

---

[†]The fact that "along" also appears in the macro definition has no significance.

[§]Words, numbers, and printed delimiters. See sec. 15.1.

[¶]For other considerations, see sec. 9.5.

having algebraic significance. For example, to insure that macro "f" behaves like an ordinary function in all contexts, it is necessary to use the more explicit definition given by

```
let f(x,y) mean begin ((x)/(y)) end
```

In point of fact, it is generally recommended that macros not be used to represent single algebraic functions,† but instead be used to represent complete commands or command sequences, as in the macro "prescribe" shown earlier. In such cases, unnatural precautions are seldom necessary, for arguments tend to be complete expressions, lists of expressions, or whole commands, rather than single variables to be substituted into larger expressions.

9.2. When macros can be used. Like all EPS commands, the let-command can be employed whenever the user chooses, and thus macros can be specified as they are needed. Moreover, once specified, a macro can be called at any point, so long as the resulting substitution is appropriate at that point. It is the user's responsibility to specify and call macros in such a way that well-formed and contextually valid substitutions are produced. EPS, for its part, is blindly obedient to the user's instructions, and can easily be induced to make substitutions which are improper. In general, no real damage or computational loss

†Algebraic functions are best represented as auxiliary variables, specified through use of the define-command. As currently implemented, this command does not admit to the explicit use of formal parameters, but, when used in conjunction with the set-command, does provide an efficient mechanism for evaluating formulae with varying parameters.

is incurred in such cases, but the user should understand that, since macro calls are literally replaced by their corresponding substitutions before command processing is initiated, error messages resulting from incorrect usage are normally expressed in terms of information which has been substituted for macro calls, rather than in terms of the calls themselves. Consequently, it is possible to receive error messages which, at the surface, appear totally unrelated to actual console input. A simple example is provided by the I/O sequence which follows:

```
PROCEED:
let pf(z) mean begin prnit z using bcd -8f9.4*- end$
INDICATED SUBSTITUTION FOR 'PF' WILL BE MADE.

PROCEED:
pf( ux(2,5,7),uy(2,5,7))$
PRNIT IS NOT A COMMAND.  TYPE 'LIST COMMANDS$' FOR
A REMINDER.
```

Here the input line "pf(...)$" has resulted in a somewhat oblique error message stemming from the earlier mistyping of the word "print". In this case the error message can be easily understood, but, as the adventurous or careless user will discover, more puzzling diagnostics are possible.

9.3. Macro definitions. As the last example suggests, no formal checking of the definition portion of a let-command is performed until an actual call for the definition is introduced. Consequently, there are no explicit restrictions as to what a macro definition may contain, although it is implicit in the manner in which input is analyzed that

a definition may not contain a terminating "$", nor may it contain the sequence delimiters "begin" and "end" except in matched pairs. To illustrate the varied forms which macro definitions may assume, it is useful to consider several specific possibilities.

Definitions with missing formal parameters. It is permissible to introduce macro definitions in which some (or all) of the macro's formal parameters are unrepresented, as in

```
let compute(x where y) mean begin print x end
```

It is understood that in processing subsequent calls, actual arguments corresponding to the missing parameters are to be ignored. Thus, given the definition above, the call

```
compute( m2+m3 where m2 and m3 are in ft.-lbs. )
```

is equivalent to

```
print m2+m3
```

It should be noted, in particular, that macro definitions may be totally degenerate. Thus the command

```
let comment(x) mean begin end
```

specifies a general mechanism for introducing explanatory remarks:

```
comment(this entire call would be ignored)
```

Macros which call other macros. It is permissible to define a macro α which calls another macro β which calls another macro γ, and so on, without restriction.[†] For example, a user might type

```
let pf(z) mean begin print z using bcd -8f9.4*- end
let psc(t) mean begin pf(sin(t),cos(t)) end $
```

Afterwards, he would find the call "psc(theta)" completely equivalent to the command

```
print sin(theta),cos(theta) using bcd -8f9.4*-
```

It should be noted that macro calls which appear within a macro definition are not processed when the definition is initially specified. Consequently, the order of specification of a system of macros is immaterial. Moreover, the effective meaning of a macro α which calls a macro β can be modified at any time by respecifying β (see Section 9.6).

Macros with conditional phrases. It is possible to specify that an input phrase *p* be processed if and only if an expression *e* has a non-zero value by typing

```
if e insert begin p end
```

Such a sequence, called a conditional phrase or if-phrase, is permitted

---

[†]As a safeguard against infinite looping, EPS interrupts input processing whenever more than twenty macro calls are encountered and expanded without producing a single executable command. In such situations, the user is requested to supply explicit input indicating whether or not he wishes further substitutions to be performed.

anywhere, but can be used to particular advantage in a macro definition, as is illustrated by the following command:

```
let zero(f,df,x,xo,tol) mean
    begin
    set x=xo
    print x,f
    if abs(f) grt tol insert
       begin
       zero(f,df,x,x-(f)/(df),tol)
       end
    end
```

This command specifies a recursive macro which uses the tangent (or Newton-Raphson) method† to find a zero for an expression "f" with independent variable "x" and first derivative "df". Called with a starting value of "xo" for the independent variable, the macro produces a sequence of new values by calling itself repeatedly until the absolute value of "f" comes to be less than or equal to "tol".

It should be noted that conditional phrases need not represent entire commands or command sequences, but may represent parts of commands as well. Consequently, the occurrence of an "if" in EPS input is not taken by the computer as a signal to terminate the command (if any) which preceeds it. Moreover, the computer must necessarily evaluate the expression on which an if-phrase depends before it can initiate processing of the command which immediately preceeds that phrase. Therefore, in defining recursive macros it is necessary to avoid con-

†See, for example, Hildebrand [10], sec. 10.8.

structions such as

```
set k=k+2
if k leq kmax insert begin phrase end
```

where the command immediately preceeding the if-phrase affects the way the if-phrase is interpreted. This can always be done by inserting a stop-command immediately before the if-phrase, as in

```
set k=k+2
stop
if k leq kmax insert begin phrase end
```

The stop-command has no explicit computational purpose, but insures that the set-command is executed before the conditional phrase is encountered.

We conclude this section with a final example, namely, the command that was used to prespecify the EPS do-macro introduced in Section 7.1:

```
let do(s for v=v1 step dv until v2) mean
    begin
    stop
    if (dv)*((v2)-(v1)) geq 0 insert
       begin
       set v=v1   s
       do(s for v=v+dv step dv until v2)
       end
    end
```

The reader should note again that this macro can be used to cause repetition of any command sequence "s" (including a sequence containing

other calls for the do-macro). He should also note that the parameters "v1", "dv", and "v2" represent arbitrary algebraic expressions and that the step size "dv" may be either positive or negative and either integral or fractional. Less general versions of this macro, e.g., versions which assume a fixed step size or prohibit arbitrary expressions for "v1" and "v2", are obviously possible, and may be expected, in general, to function more efficiently.

9.4. Special forms. The preceeding sections deal exclusively with macros specified by means of commands of the form

let *macroname*(p_1 s_1 p_2 ... s_{N-1} p_N) mean
begin
$d(p_1;p_2;\ldots;p_N)$
end

Actually, two variations of this form are permitted. One variation, used to specify parameterless macros, is given quite simply by

let *macroname* mean begin d end

This command causes the fixed definition d to be substituted for subsequent occurrences of the identifier *macroname*, and, while useful in many situations (see Section 9.7, Example *A*), requires no detailed explanation.

The second variation is used to specify self-repeating macros, and is given in terms of the constituents of a normal let-command, plus a "trailing separator" s_N, as follows:

```
let macroname(p1 s1 p2 ... sN-1 pN sN) mean
    begin
    d(p1;p2;...;pN)
    end
```

The presence of the trailing separator is taken to indicate that subsequent calls may contain more than one set of arguments, each set being separated from the next by an s_N, as in

```
macroname(a1 s1 a2 ... sN-1 aN sN
          b1 s1 b2 ... sN-1 bN sN
                .........
          z1 s1 z2 ... sN-1 zN)
```

The computer replaces each such call by a sequence of substitutions, one for each set of arguments, obtaining

```
d(a1;a2;...;aN)  d(b1;b2;...;bN)  ...  d(z1;z2;...;zN)
```

A concrete example is provided by the command

```
let pu(a,b,) mean
    begin
    set i=a, j=b   print x, y, u(1,a,b)
    end
```

This defines a macro which, for example, could be called with

```
pu( 1,1, 2,4, 5,9)
```

to cause execution of the command sequence

```
set i=1, j=1   print x, y, u(1,1,1)
set i=2, j=4   print x, y, u(1,2,4)
set i=5, j=9   print x, y, u(1,5,9)
```

Note that no difficulty arises from the fact that the same element, a comma, is used for both intermediate and trailing separators.

As the last example may suggest, self-repeating macros are extremely useful in situations where repetition of a given command sequence is desired for sets of arguments which cannot be generated algorithmically. One such situation of general interest concerns the specification of numerical data in the form of an array. Suppose, for example, that the components of a vector "z" represent experimental values which must be specified manually and individually. Without macros such an operation can be quite tedious, for it requires the explicit naming of each component, as in

```
set z(1)=2.51, z(2)=3.74, z(3)=4.97, z(4)=6.02,
    z(5)=7.76, z(6)=9.93, ...
```

However, by typing

```
let fillz(x,) mean begin set z(k)=x, k=k+1 end
```

one can eliminate the need to name each component, and write instead

```
set k=1  fillz(2.51, 3.74, 4.97, 6.02, 7.76, 9.93, ...)
```

More generally, one can specify a macro for filling any vector "v" with a list of values "x", starting with component "v(ko)", by typing

```
let fillvector(v from ko with x) mean
    begin
    let fillv(y,) mean begin set v(k)=y, k=k+1 end
    set k=ko   fillv(x)
    end
```

Similarly, one can specify a macro for filling an "n" column matrix "w" with values "x", assuming these are listed row-wise starting with component "w(1,1)", by typing

```
let fillmatrix(w thru n with x) mean
    begin
    let fillm(y,) mean
        begin
        set if j grt n insert begin i=i+1, j=1, end
           w(i,j)=y, j=j+1
        end
    set i=j=1   fillm(x)
    end
```

The reader should note that these last two macros have definitions which themselves specify macros. Nesting of macro definitions is, in fact, permitted to any depth.

9.5. Separators, arguments, and imperfect calls. In several of the preceeding examples, argument separators have been chosen in such a way as to produce calls which read almost like English sentences. This is often a worthwhile practice, as it makes the order and significance of arguments easier to remember. However, it is important to recognize that the primary function of a separator is to provide the computer with

an unambiguous means of distinguishing one argument from the next. To do this, separators must be chosen with forethought, in accordance with the following restriction: a separator s_k should not be any element which might occur in the preceeding argument a_k, except possibly within matching left and right parentheses.† Thus, for example, if it is intended that a_k be a list of algebraic expressions separated by commas, then s_k should not itself be a comma. On the other hand, if a_k is expected to be a single algebraic expression, then s_k could be a comma, since any commas which might occur in a_k would necessarily be "protected" by parentheses, as in "a(2,2)/fy".

The protective nature of parentheses in macro arguments means that arguments themselves are subject to a restriction, namely, the requirement that they not contain unbalanced parentheses. In principle, this is not a serious restriction, as it merely dictates that arguments be syntactically complete algebraic expressions, commands, macro calls, etc. In practice, however, difficulties can arise, for it is easy to introduce mismatched parentheses inadvertently when typing complicated expressions. The important point here is that mismatched parentheses cannot be detected as such by the EPS macro processor, for, at the level of the macro processor, calls with mismatched parentheses are indistinguishable from certain imperfect calls, which are admitted deliberately. As a result, mismatched parentheses simply lead to improper argument substitutions, which, in turn, generally yield ill-formed

†Macro calls are analyzed by means of a left-to-right scan. Therefore, it is not strictly necessary, when selecting a given separator, to consider the possible forms of the argument which follows it.

or otherwise invalid commands. These are normally reported without serious consequences, although, as indicated in Section 9.2, they may be reported in terms which are somewhat obscure.

Macro calls with argument lists which are incomplete by virtue of missing outer parentheses or missing arguments and separators are called imperfect calls, and are admitted by current versions of EPS in order to enable the user to take short cuts in certain restricted but common situations. Three cases that may be of general interest are discussed below.

(1) It is permissible to omit the opening left parenthesis of a macro call's argument list provided (a) the first argument a_1 does not itself begin with a left parenthesis and (b) the call is not itself an argument to another call. Thus, for example, no difficulty is incurred by typing

seq_1 do print z(k) for k=1 step 1 until 8) seq_2

provided seq_1 and seq_2 do not represent parts of another macro call in which the call "do print ..." is an argument.

(2) It is permissible to omit the closing right parenthesis of an argument list provided this list is the very last thing in an input sequence and, therefore, is followed immediately by a "$". (In effect, the computer fills in as many right parentheses as are needed to balance left parentheses occurring earlier in the sequence.) This case and case (1) are

illustrated concurrently in the sequence which follows:

```
set n=3, b=c=d=f=h=0
do do(set a(k,l)=e(k,l)=g(k,l)=0 for l=1 step 1
  until n) for k=1 step 1 until n)
do set a(k,k)=e(k,k)=g(k,k+1)=1 for k=1 step 1
  until n $
```

Note, in particular, that it has been possible to omit <u>both</u> outer parentheses in the final call for macro "do". This sequence, incidentally, specifies governing equation parameters corresponding to the system $\{ \nabla^2 u_1 + u_2 = 0, \ \nabla^2 u_2 + u_3 = 0, \ \nabla^2 u_3 = 0 \}$, which is equivalent to the sixth-order equation $\nabla^6 u_1 = 0$ (see Section 4).

(3) It is permissible, in general, to omit any or all of the arguments in a macro call and also to omit separators which would otherwise be left trailing at the <u>right end</u> of an argument list as a result of the omission of arguments. The computer uses formal parameter names in lieu of all missing arguments. Thus, by using parameter names which themselves have significance, one can specify macros with optional arguments. For example, a looping macro analogous to the FORTRAN DO-statement is specified by the command

```
let dof(s for i=0,n,1) mean
    begin
    do(s for i=0 step 1 until n)
    end
```

Possible calls and corresponding substitutions for this macro are illustrated below:

```
dof(seq for t=v,w,z) → do(seq for t=v step z until w)
dof(seq for m=3,9)   → do(seq for m=3 step 1 until 9)
dof(seq for k=5)     → do(seq for k=5 step 1 until n)
dof(seq for j)       → do(seq for j=0 step 1 until n)
dof(seq)             → do(seq for i=0 step 1 until n)
```

It must be emphasized that separators cannot be omitted when they occur at intermediate points in an argument list. For example, a call for macro "dof" which omits only the argument corresponding to parameter "n" must have the form

$$\text{dof}(seq \text{ for } v{=}v_1,,\delta v)$$

Similarly, a call which omits only the arguments corresponding to "i" and "n" must have the form

$$\text{dof}(seq \text{ for}{=}i_1,,\delta i)$$

For maximum convenience, macros should be specified so that optional arguments occur in order of increasing likelihood of omission.

The discussion of imperfect calls given here is not complete, but can be easily supplemented by direct experimentation. To facilitate experimentation, EPS provides a special output command, normally written in terms of a sequence p containing macro calls, as follows:

expand begin *p* end

The computer responds to this command by printing its interpretation of the sequence *p*. For example, after the specification of macro "prescribe" shown in Section 9.1, the computer would respond to

```
expand begin prescribe fx+fy along 5,5, 5,6) end $
```

by printing

```
SET M=M+1, P(1,1,M)=0, Q(1,1,M)=1 DEFINE R(1,M)=FX
+FY IMPOSE M ALONG 5,5, 5,6
```

Note that it is not assumed that sequence *p* represents an executable command sequence, nor is any attempt made to execute *p* if it turns out to be an executable command sequence. Consequently, the expand-command cannot be employed to determine the significance of calls for recursive macros, such as the do-macro, whose proper operation hinges on actual command execution. It may, on the other hand, be employed to cause printing of arbitrary text strings and, therefore, is useful not only for testing (non-recursive) macros, but also for printing non-numerical output, e.g., table headings and messages of confirmation (see Section 9.7, Example *C*).

9.6. Macro deletion and respecification. It is possible to delete one or more macros by typing a command of the form

remove begin *macroname*$_1$, *macroname*$_2$, ... end

For each identifier listed (and actually found to name a macro), the computer will respond with a message of the form

MACRO '*MACRONAME*' NOW INACTIVE.

Subsequent uses of all identifiers will then be taken literally. Since the number of storage registers required to represent a macro is roughly twice the number of elements in the let-command used to specify the macro initially (the do-macro requires about 100 words of storage), it is worthwhile to delete macros, thereby recovering storage for other purposes, whenever they are no longer needed.

A macro can be respecified at any time simply by introducing a new let-command. When this is done, the computer simulates an appropriate remove-command before processing the new macro description. Therefore, it prints a "MACRO INACTIVE" message before printing the usual "INDICATED SUBSTITUTION ... WILL BE MADE." As may be evident, no more than one macro with a given identifier is permitted. Therefore, any let-command specifying a *macroname* which is already in use necessarily forces the removal of the current description, even if the new description involves a different number of formal parameters or a different set of separators.

9.7. Additional examples. In this final section, three additional examples, related specifically to the solution of boundary-value problems, are discussed. (See also Sections 10 and 11.)

A. Specification of boundary conditions. The usefulness of macros

in specifying boundary conditions has already been indicated in terms of the macro "prescribe" shown in Section 9.1. This macro, however, is concerned with a very simple class of problems and boundary conditions. A more complicated situation and more imaginative use of macros is illustrated by the command sequence below, which specifies a system of macros for use in the solution of elasticity problems (see Section 4.3, Example *D*, and Section 5.5).

```
let make(a=b,c=d along e) mean
    begin
    set m=m+1, p(1,2,m)=p(2,1,m)=q(1,2,m)=q(2,1,m)=0,
        a=b, c=d
    impose m along e
    end
let dx mean begin p(1,1,m)=0, q(1,1,m)=1, r(1,m) end
let dy mean begin p(2,2,m)=0, q(2,2,m)=1, r(2,m) end
let tx mean begin p(1,1,m)=1, q(1,1,m)=0, r(1,m) end
let ty mean begin p(2,2,m)=1, q(2,2,m)=0, r(2,m) end
```

This sequence enables the user to specify boundary conditions in terms of the parameterless macros "dx", "dy", "tx", and "ty", which correspond, respectively, to x-displacement, y-displacement, x-traction, and y-traction conditions. For example, to indicate that the boundary arc 5,-2, 6,-2, 23,-2 is to have no x-displacement and is to be subjected to a y-traction of one unit, the user could type

```
make(dx=0, ty=1 along 5,-2, 6,-2, 23,-2)
```

Obviously it is possible to formulate similar systems of macros to facilitate the specification of boundary conditions in any application area.

B. Numerical differentiation. As indicated in Section 7, EPS provides built-in functions "ux", "uyy", etc. for calculating approximate values for first and second derivatives $\partial u_k/\partial x$, $\partial^2 u_k/\partial y^2$, etc. based on current states of solution matrix "u" and lattice parameters "x" and "y". At times, however, the user may find it necessary to supplement these built-in functions with procedures of his own which, for example, calculate derivatives of the u_k with respect to variables other than x and y, or which calculate derivatives of quantities other than the u_k. Here a procedure is considered which calculates approximate slopes for an arbitrary function f in the directions of the grid lines of a lattice. It is assumed that values of f for adjacent nodes of the lattice are well-behaved with respect to changes in arc length s measured along the grid lines, justifying the representation of $f(s)$ in the neighborhood of any three consecutive nodes with a second-order polynomial.† By choosing coefficients so that the polynomial matches given values f_k [k = 1,2,3] for arguments s_k corresponding to the three nodes and then differentiating, one obtains the following approximation for the slope of f at intermediate node s_2:

$$\left(\frac{df}{ds}\right)_{s_2} \simeq \frac{(s_2 - s_1)^2(f_3 - f_2) + (s_3 - s_2)^2(f_2 - f_1)}{(s_2 - s_1)(s_3 - s_2)(s_3 - s_1)}$$

This approximation has been incorporated in the following command sequence, which specifies a macro that causes printing of the approxi-

†In general this assumption demands, among other things, that the straight line segments which constitute the grid lines change direction slowly, i.e., the grid lines must be reasonable approximations to smooth curves.

mate slope in the "k"-direction, where "i" or "j" must be substituted for "k", of any quantity "f" at any intermediate node "i2,j2":

```
let slope(f,k,i2,j2) mean
    begin
    set i=i2, j=j2, f2=f, x2=x, y2=y,
        k=k-1, f1=f, ds21=dsfcn,
        k=k+2, f3=f, ds32=dsfcn
    print dfds
    end
define dsfcn=sqrt((x-x2)power 2+(y-y2)power 2),
    dfds=(ds21*ds21*(f3-f2)+ds32*ds32*(f2-f1))/
        (ds21*ds32*(ds21+ds32))
```

As a result of this sequence, the approximate slope of solution component u_2 in the i-direction at node (5,6) could be obtained by typing

```
slope(u(2,i,6),i,5,6) $
```

Note that the first argument in this call could also be written "u(2,i,j)", but not "u(2,5,6)", the latter being constant for all "i". To avoid confusion and minimize typing when several calls are anticipated, it is advisable to define as auxiliary parameters functions to be used as first arguments. For example, type

```
define psi=u(2,i,j)  slope(psi,i,5,6) $
```

All subsequent calls for derivatives of u_2 could then be made in terms of "psi".

C. Numerical integration. In many problems, integral properties

of a solution or its derivatives are of interest. EPS provides no built-in mechanism for computing integrals, so the user must supply his own. For most purposes, integrals can be adequately approximated by use of the trapezoidal rule, which is formulated in terms of an integrand $f(z)$ and a set of arbitrarily spaced sample points z_m, z_{m+1}, ..., z_n as follows:

$$\int_{z_m}^{z_n} f(z)dz \simeq \sum_{\ell=m+1}^{n} \frac{1}{2}[f(z_\ell) + f(z_{\ell-1})](z_\ell - z_{\ell-1})$$

Shown below is a command sequence implementing this formulation for integrations along grid lines of constant "k" between nodes "k=ko, l=m" and "k=ko,l=n".

```
define newsig=sig+.5*(f1+f0)*(z1-z0)
let sum(f,z for k=ko, l=m,n) mean
   begin
   set k=ko, l=m, f0=f, z0=z, sig=0, dl=1 sign(n-m)
   do( set f1=f, z1=z, sig=newsig, f0=f1, z0=z1
       for l=m+1 step dl until n)
   expand begin integral of f wrt z along k=ko,
       l=m,n is end
   print sig
   end
```

With macro "sum" one could, for example, obtain the approximate value of $\int_C (\partial u_1/\partial x)dy$ for a path C corresponding to the grid line i=5, for j=0 through 20, by typing

```
sum(ux(1,5,j),y for i=5, j=0,20) $
```

This call would result in a computer response of the form

```
INTEGRAL OF UX(1,5,J) WRT Y ALONG I=5, J=0,20 IS
  ±X.XXXXXXXE±XX
```

As in Example *B*, it is advisable to use auxiliary parameters for argument "f" if multiple calls are anticipated. Various generalizations (e.g., macros to perform area integrations) and refinements (more accurate integration procedures) are obviously possible.

10. Iterative problem solving

10.1. Introduction. This section discusses iterative techniques for solving boundary-value problems for nonlinear partial differential equations, for treating free-surface problems, and for constructing potential flow nets. In all cases the basic strategy is as illustrated in Figure 9. First, a finite-difference lattice and differential system approximating the situation of interest are specified and an initial solution is obtained by means of the procedures described in Sections 3 through 6. Then equation or lattice parameters are revised on the basis of the initial solution to establish a more accurate representation of the true situation, and a second solution is obtained. This process is repeated until convergence as defined by some suitable criterion is achieved.

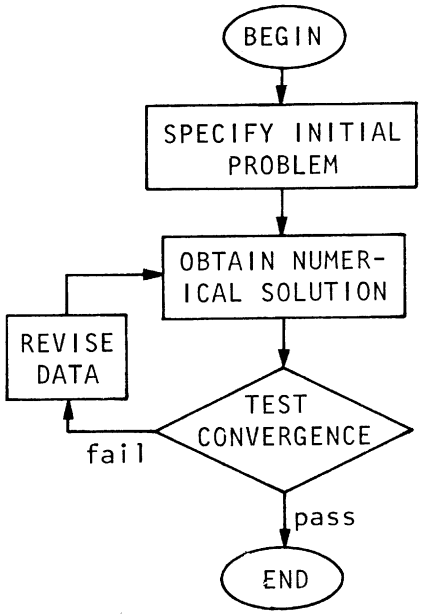


Fig. 9. *Iterative strategy for nonlinear equations, etc.*

Detailed procedures for several specific cases are given in Sections 10.2 and 10.3. However, it is emphasized that the basic strategy of Figure 9 is applicable to many situations not even implicitly considered. For example, engineering design problems requiring the optimization of part shape or other physical properties are obviously amena-

ble to the same approach. At the same time, however, the procedures actually considered may not be suitable for every possible application. Convergence may be prohibitively slow in some cases, while outright divergence may be unavoidable in others. (Techniques which can sometimes be used to accelerate convergence or increase numerical stability are discussed in Section 10.4.) In general, nevertheless, these procedures are remarkably dependable, and provide what is in many instances the only available means for realizing a solution.

10.2. Nonlinear equations. The form of governing system (1), Section 4.1, coincides with that of various nonlinear systems when coefficients $a_{k\ell}$, $b_{k\ell}$, ..., $g_{k\ell}$ are considered to be functions not only of the independent variables x and y, but also of the dependent variables u_1, u_2, ..., u_n and their derivatives. The EPS difference equation generator does not explicitly admit such coefficients, but it has been found that solutions can frequently be obtained by iteration in cases where no derivatives of order greater than one are involved, i.e., where

$$a_{k\ell} = a_{k\ell}(x,y,u_1,\ldots,u_n,\frac{\partial u_1}{\partial x},\ldots,\frac{\partial u_n}{\partial x},\frac{\partial u_1}{\partial y},\ldots,\frac{\partial u_n}{\partial y}),$$

$$b_{k\ell} = b_{k\ell}(x,y,u_1,\ldots,u_n,\frac{\partial u_1}{\partial x},\ldots,\frac{\partial u_n}{\partial x},\frac{\partial u_1}{\partial y},\ldots,\frac{\partial u_n}{\partial y}),$$

etc.

What this entails, basically, is the solution of a sequence of linear problems, each of which has spatially varying coefficients evaluated

by using the previous solution of the sequence. To be specific, estimates $u_k^0 = u_k^0(x,y)$ $[k = 1,2,\ldots,n]$ for the behavior of the dependent variables are used to formulate an initial problem with coefficients

$$a_{k\ell} = a_{k\ell}(x,y,u_1^0,\ldots,u_n^0,\frac{\partial u_1^0}{\partial x},\ldots,\frac{\partial u_n^0}{\partial x},\frac{\partial u_1^0}{\partial y},\ldots,\frac{\partial u_n^0}{\partial y}) = a_{k\ell}^0(x,y),$$

etc.

The solution components $u_k^1 = u_k^1(x,y)$ obtained by solving this initial problem are then used to formulate a second problem with coefficients

$$a_{k\ell} = a_{k\ell}(x,y,u_1^1,\ldots,u_n^1,\frac{\partial u_1^1}{\partial x},\ldots,\frac{\partial u_n^1}{\partial x},\frac{\partial u_1^1}{\partial y},\ldots,\frac{\partial u_n}{\partial y}) = a_{k\ell}^1(x,y),$$

etc.,

and so on, until finally the difference between one solution and the next falls below some acceptable tolerance. Convergence is not always guaranteed, of course, but in many practical applications it in fact proceeds at such a rate that only a few iterations are necessary.

In terms of computer input, it is generally convenient to use the following procedure:

(1) Specify coefficients $a_{k\ell}$, $b_{k\ell}$, etc. in terms of auxiliary parameters representing their various arguments, e.g., if $a_{21} = a_{21}(x,u_1,\partial u_1/\partial y) = 8x + u_1 - |\partial u_1/\partial y|$, type

```
define a(2,1)=8*fx+fu1-abs(fuy1)
```

(2) Specify auxiliary parameters introduced in step (1) using the fit-function discussed in Section 4.2, making certain

that reasonable initial estimates for all pertinent dependent variables and derivatives are included. For example, if it is decided that $u_1^0 = xy$, hence $\partial u_1^0/\partial y = x$, type[†]

```
define fx=fit(x), fy=fit(y).
   ful=fx*fy, fuyl=fx
```

(3) After obtaining a first solution, redefine the auxiliary parameters that represent dependent variables in terms of appropriate components of the solution matrix, e.g.,

```
define ful=fit(u(1,i,j))
```

Similarly, redefine parameters representing derivatives of the dependent variables, in this case introducing auxiliary matrices into which values for the derivatives produced by means of the built-in functions "ux" and "uy" can be stored prior to each succeeding iteration, e.g.,

```
define fuyl=fit(dyl(i,j))
```

(Coefficient parameters cannot be defined directly in terms of functions "ux" and "uy" because these functions cannot be referenced during the actual generation of finite-difference equations.)

(4) Specify pertinent components of auxiliary matrices intro-

---

[†]This example demonstrates the general case, but in practice it is normally convenient and numerically just as satisfactory to use constant values for the u_k^0 in formulating an initial problem. Generally it is appropriate to begin with some limiting case, e.g., $u_k^0 = 0$, corresponding to a standard linearization for the problem of interest.

duced in step (3), if any, e.g., by typing input sequences such as[†]

```
do( do( set dy1(i,j)=uy(1,i,j)
        for i=0 step 1 until imax)
        for j=0 step 1 until jmax)
```

(5) Type form- and relax-commands to obtain a new solution.

(6) Repeat steps (4) and (5) as needed until satisfactory convergence is achieved.

Note that the typing involved in repeating steps (4) and (5), as well as in carrying out any computations needed to test for convergence, can be reduced considerably by means of appropriate macros.

To illustrate the procedure outlined above, three specific examples are considered.

A. Heat conduction in a material with temperature-dependent conductivity. The steady-state temperature θ in a material with conductivity $k(\theta) = (\theta/\theta_o)^s k_o$, where θ_o, s, and k_o are constants, satisfies the following nonlinear equation:

[†]It may also be desirable at this point to revise boundary condition or other parameters. Normally the effects of nonlinearities show up gradually as the inhomogeneities or forcing functions in a differential system are increased from zero. Thus numerical instabilities can sometimes be avoided if the quantities h_k in governing system (1) and the quantities r_{km} in boundary equation (9) are initially assigned relatively small values, then gradually increased as the iteration progresses. Similarly, multipliers of nonlinear terms in governing equations can sometimes be varied in order to control instability. See Example *C*.

$$\text{div}\,[(\theta/\theta_o)^s k_o \,\text{grad}\,\theta] = 0.^{\dagger}$$

When expressed in two-dimensional form and in terms of Cartesian coordinates, this equation corresponds to governing system (1) with

$$u_1 \rightarrow \theta,$$
$$n = 1,$$
$$a_{11} = e_{11} = (u_1/\theta_o)^s k_o,$$
$$b_{11} = c_{11} = d_{11} = f_{11} = g_{11} = h_1 = 0.$$

Accordingly, governing equation parameters are specified by means of a command sequence such as the following:

```
set n=1, b=c=d=f=g=h=0
define a=e=ko*(t/θo)power s
```

Auxiliary parameter "t" is then given some appropriate initial definition and a first solution for the desired lattice and boundary conditions is obtained. After this, "t" is redefined in terms of solution matrix "u", i.e., the command

```
define t=fit(u(1,i,j))
```

is given, and form- and relax-commands are issued repeatedly until suitable convergence is achieved. Convergence rates of about one

---

†This equation can be linearized (converted to Laplace's equation) by means of a variable transformation. However, since linear boundary conditions relating heat flux and temperature become nonlinear when expressed in terms of the transformed variable (see Ames [11], pp. 21-23), it is often just as convenient to leave the equation in its original form. In any event, the procedure considered here is obviously applicable in other situations where linearization by variable transformation is not possible.

decimal digit per iteration have been observed in several test cases for various values for exponent s between -1 and 3.

B. Isentropic compressible flow. The continuity equation for the steady, irrotational, isentropic flow of a perfect gas can be written in terms of a velocity potential Φ and specific heat ratio k as follows:[†]

$$\text{div}\left((1-|\text{grad }\Phi|^2)^{1/(k-1)}\text{grad }\Phi\right) = 0.$$

When Cartesian coordinates are employed, the two-dimensional form of this equation corresponds to governing system (1) with

$$\begin{gathered} u_1 \to \Phi, \\ n = 1, \\ a_{11} = e_{11} = (1-|\text{grad } u_1|^2)^{1/(k-1)}, \\ b_{11} = c_{11} = d_{11} = f_{11} = g_{11} = h_1 = 0. \end{gathered}$$

Thus equation parameters are specified by means of a sequence such as

```
set n=1, b=c=d=f=g=h=0, k1=1/(k-1)
define a=e=(1-vv)power k1
```

where auxiliary parameter "vv", corresponding to the quantity $|\text{grad }\Phi|^2$, is normally set to zero initially in order to obtain a first solution representing ordinary potential flow. Afterwards it is convenient to

---

[†]This equation is obtained by noting that density ρ and velocity $\bar{v}$ for isentropic flow are related by $\rho/\rho_o = (1-|\bar{v}/v_{max}|^2)^{1/(k-1)}$, where ρ_o is the stagnation density and v_{max} the velocity magnitude at zero enthalpy, and by defining potential Φ so that grad $\Phi = \bar{v}/v_{max}$. See Shapiro [12], Arts. 4.3 and 4.4.

redefine "vv" in terms of a single matrix, as in

```
define vv=fit(vvm(i,j))
```

where the components of "vvm" are to be computed in accordance with

$$\text{vvm(i,j)} \leftarrow \left(|\text{grad } \Phi|^2 = (\partial\Phi/\partial x)^2+(\partial\Phi/\partial y)^2 \right)_{\text{at (i,j)}}$$

(Note that this is more efficient than having two matrices, one for $\partial\Phi/\partial x$ and one for $\partial\Phi/\partial y$, though it represents a slight departure from the procedure defined earlier.) Thus, for example, with a lattice which is rectangular in i,j-space, say with vertices (0,0), (20,0), (20,15), and (0,15), a second solution is obtained by typing

```
let revise mean
    begin
    do( do( set vvm(i,j)=vvfcn
        for i=0 step 1 until 20)
        for j=0 step 1 until 15)
    end
define vvfcn=ux(1,i,j)power 2+uy(1,i,j)power 2
revise form relax$
```

After this, succeeding solutions are obtained simply by typing "revise form relax$" repeatedly, or, better yet, by introducing and calling a recursive macro which automatically causes this sequence to be re-processed until some appropriate convergence condition is satisfied. For example, the macro defined below causes iterations to be repeated until fractional changes in "vvm(icrit,jcrit)" fall below "tol", where presumably "(icrit,jcrit)" is a node of particular concern.

```
let repeat(tol) mean
    begin
    set vvc=vvm(icrit,jcrit)
    revise  stop
    if abs((vvm(icrit,jcrit)-vvc)/vvm(icrit,jcrit))
      geq tol insert
       begin
       form  relax  repeat(tol)
       end
    end
```

Convergence rates of one decimal digit every two or three iterations have been observed in several test cases where boundary conditions were chosen so as to produce Mach numbers of unity at one or two points in the flow. Interestingly, there appear to be no problems with numerical instability even when Mach numbers exceed unity by small amounts, though solutions may not be physically tenable in such cases.

C. Viscous incompressible flow. The dynamical equations for steady, two-dimensional, viscous, incompressible flow can be written in terms of dimensionless Cartesian coordinates x and y, a dimensionless scalar vorticity function ω, dimensionless stream function Ψ, and Reynolds number R as follows:[†]

$$\frac{\partial^2\omega}{\partial x^2} + \frac{\partial^2\omega}{\partial y^2} - R\cdot\left(\frac{\partial\Psi}{\partial y}\frac{\partial\omega}{\partial x} - \frac{\partial\Psi}{\partial x}\frac{\partial\omega}{\partial y} \right) = 0$$

$$2\,\omega + \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2} = 0$$

[†]See, for example, Schlichting [13], pp. 58f.

It has been found that solutions to these equations for flows around obstacles can be obtained for modest Reynolds numbers, say, values less than 500, based on the characteristic dimension of an obstacle. However, to avoid numerical instabilities, it is generally necessary to proceed in stages, starting with the limiting case $R = 0$ and gradually introducing larger values, e.g., $R = 50, 100, 150, \ldots$, until a solution for the value of interest is obtained. Note, first, that the nonlinear terms of the first equation can be rewritten in the form

$$\frac{\partial}{\partial x}\left(-R\ \omega\ \frac{\partial \Psi}{\partial y} \right) + \frac{\partial}{\partial y}\left(R\ \omega\ \frac{\partial \Psi}{\partial x} \right)$$

Thus a correspondence with governing system (1) can be established in which

$$u_1 \rightarrow \omega, \quad u_2 \rightarrow \Psi,$$
$$n = 2,$$
$$a_{11} = a_{22} = e_{11} = e_{22} = 1, \quad g_{21} = 2,$$
$$b_{12} = -R\ u_1, \quad d_{12} = R\ u_1,$$

and in which all other coefficients are zero. Equation parameters corresponding to this formulation are specified by typing

```
set n=2, a(1,2)=a(2,1)=b(1,1)=b(2,1)=b(2,2)=c=
    d(1,1)=d(2,1)=d(2,2)=e(1,2)=e(2,1)=f=g(1,1)=
    g(1,2)=g(2,2)=h=0, a(1,1)=a(2,2)=e(1,1)=
    e(2,2)=1, g(2,1)=2
define b(1,2)=-rn*w, d(1,2)=rn*w
```

where auxiliary parameters "rn" and "w", corresponding to R and ω, respectively, are ordinarily set to zero before difference equations

are requested for the first time. After an initial solution is obtained, "w" is redefined by typing

```
define w=fit(u(1,i,j))
```

and succeeding iterations are performed by assigning "rn" a small non-zero value, typing form- and relax-commands until partial convergence is achieved, reassigning "rn" a slightly larger value, typing more form- and relax-commands, and so on. It has been observed that as larger values for "rn" are introduced, iterations display a growing tendency to wander and diverge. To counteract this tendency, smaller values for relaxation factor "omega" and techniques described in Section 10.4 can be introduced. Eventually, however, a Reynolds number is reached for which it is impossible to maintain numerical stability and an acceptable convergence rate simultaneously.

10.3. Iterative lattice revision. In the analysis of seepage through porous media, elasto-plastic material behavior, fluid jets, and other phenomena, it is common to encounter boundary-value problems with free surfaces, i.e., with boundary segments which are initially undefined. Generally it is possible to associate with a free surface more than one set of boundary conditions. Hence the shape of the surface can usually be determined by first assuming some initial configuration and obtaining a solution satisfying one set of boundary conditions, then using the discrepancy between that solution and the remaining set of boundary conditions to revise the configuration, and so on.

With EPS it is easy to carry out such a process if changes in

boundary geometry from one iteration to the next can be made without modifying lattice structure in i,j-space. Depending on the circumstances, this can usually be accomplished by means of one or both of the following techniques:

(1) The mapping between x,y- and i,j-coordinates can be expressed in terms of interpolation formulae giving the locations of interior nodes as functions of the locations of boundary nodes (see Section 3.5). Then iterations can be performed simply by revising auxiliary parameters which specify the locations of nodes lying on the free surface. Interpolation formulae obviously must be devised so that grid lines cannot assume illegal configurations as the free-surface nodes are repositioned.

(2) The mapping between x,y- and i,j-coordinates can be given in matrix form and each node adjusted independently to establish a lattice whose nodes lie along characteristic function contours. For example, in potential flow problems nodes can be made to lie along lines of constant potential and constant stream function, so that the lattice becomes a flow net.

The second technique is sometimes useful even when no free surface is present, since the lattice it produces may have intrinsic utility and may also accord more accurate solutions. (Experiments indicate that this is generally true in the case of potential flow problems.)

Because the strategy for revising lattice parameters varies from application to application, general input procedures corresponding to

the above techniques cannot be presented. Instead, procedures appropriate to a specific but representative problem are described. Consider, in particular, steady seepage through a porous dam with trapezoidal cross section and horizontal, impervious base (see Figure 10). In this situation, the free surface is the interface between the region of seepage and the dry portion of the dam (curve AB in the figure). A

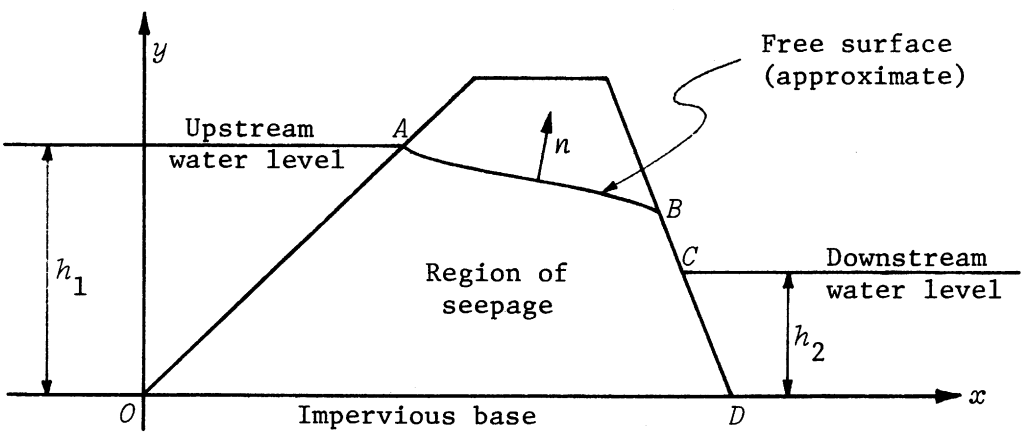


Fig. 10. *Schematic of trapezoidal dam.*

boundary-value problem for the region of seepage can be formulated in terms of the piezometric head h, which under suitable assumptions plays the same role in seepage problems as the velocity potential in potential flow problems. In particular, if it is assumed that Darcy's law is satisfied and, further, that the permeability of the dam is the same in all directions and at all points, then one arrives at the following formulation (coordinates x and y, local normal n, and dimensions h_1 and h_2 are as shown in Figure 10):†

†See, for example, Polubarinova-Kochina [14], pp. 31-35.

Governing equation:
$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0. \tag{13}$$

Boundary condition along

free surface AB: $\frac{\partial h}{\partial n} = 0, \quad h = y;$

exposed discharge face BC: $h = y;$

submerged discharge face CD: $h = h_2;$

impervious base OD: $\frac{\partial h}{\partial n} = 0;$

entrance face OA: $h = h_1.$ (14)

For each of the above expressions, an equivalent expression can be given in terms of a stream function Ψ, related to h by the Cauchy-Riemann equations

$$\frac{\partial \Psi}{\partial x} = -\frac{\partial h}{\partial y} \quad \text{and} \quad \frac{\partial \Psi}{\partial y} = \frac{\partial h}{\partial x}.$$

Thus it is possible to arrive at the following alternate formulation:

Governing equation:
$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = 0. \tag{15}$$

Boundary condition along

free surface AB: $\Psi = \Psi_A = \text{constant}, \quad \frac{\partial \Psi}{\partial n} = \frac{\partial y}{\partial s};$

exposed discharge face BC: $\frac{\partial \Psi}{\partial n} = \frac{\partial y}{\partial s};$

submerged discharge face CD: $\frac{\partial \Psi}{\partial n} = 0;$

impervious base OD: $\Psi = 0;$

entrance face OA: $\frac{\partial \Psi}{\partial n} = 0.$ (16)

Here s represents arc length along the boundary in the clockwise direction. Note that along the free surface, which is fixed only at entrance point A, two boundary conditions are stipulated in each formulation.

Input procedures corresponding to the two iterative techniques outlined earlier and appropriate to the seepage problem just described are now considered. The first procedure, based on the use of interpolation formulae and aimed solely at the determination of the free surface and the head distribution $h = h(x,y)$, is best suited to dams with steep discharge faces. The second procedure, where all nodes are revised independently to produce a flow net (hence solutions for both h and Ψ), is applicable to a wider range of configurations. Both procedures are outgrowths of an investigation carried out in collaboration with Lo [15], whose thesis may be consulted for further details and specific examples.

A. Solution based on use of interpolation formulae. For dams with steep discharge faces it has been found to be convenient and appropriate to use lattices of the type shown in Figure 11. Here each grid

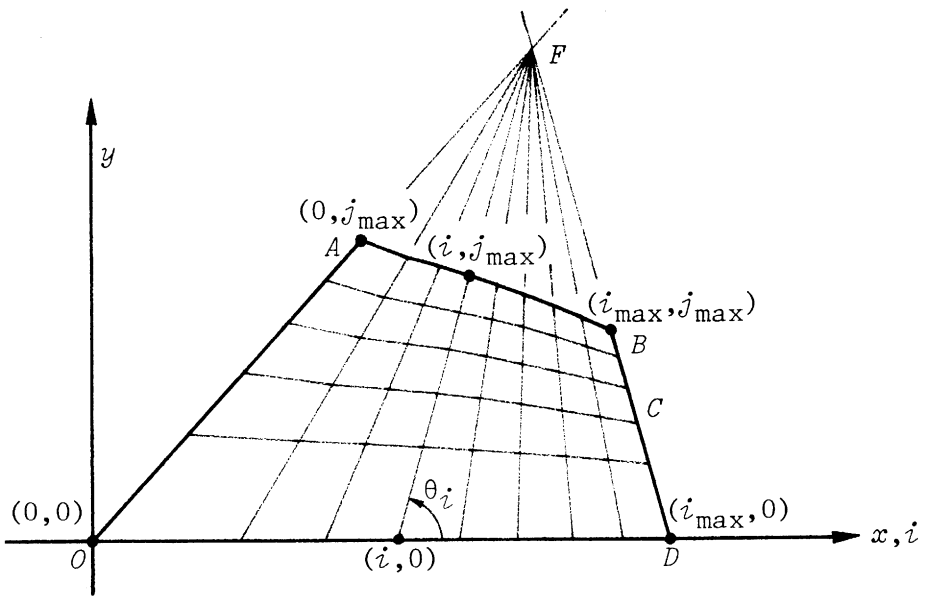


Fig. 11. *Lattice for seepage region of dam with interpolated nodal distribution.*

line of constant i coincides with a straight line connecting a point on the base of the dam with the intersection point F formed by extending the entrance and discharge faces. For each such line, nodes are distributed between a node $(i,0)$ on the base and a node (i,j_{max}) on the free surface in a preestablished manner, so that as the free-surface node is moved up and down, the other nodes move by proportional amounts.

Iterations are initiated by estimating the elevation $y_{i,j_{max}}$ of each free-surface node and obtaining a numerical solution for $h = h(x,y)$, based on Eqs. (13) and (14), which satisfies the boundary condition $\partial h/\partial n = 0$ along the free surface. Differences between this numerical solution and the estimated free-surface elevations--i.e., discrepancies between the solution and the boundary condition $h = y$--are then used to compute new elevations, etc. It has been found that for a free-surface node where the discrepancy is

$$\Delta h_{i,j_{max}} = h_{i,j_{max}} - y_{i,j_{max}},$$

a new elevation $y'_{i,j_{max}}$ can be computed using

$$y'_{i,j_{max}} = y_{i,j_{max}} + \Delta h_{i,j_{max}} \sin \theta_i, \tag{17}$$

where θ_i is the angle which grid line i makes with the base of the dam (see Figure 11). When this expression was employed in various tests, the $y_{i,j_{max}}$ converged at rates of one decimal digit every four or five iterations.

In order to carry out the above procedure on the computer, the user can proceed as follows:

(1) Specify lattice structure in i,j-space in the usual manner (see Section 3.4), and define a mapping algorithm in terms of coordinates "xf" and "yf" of intersection point F, length "xd" of the base of the dam, vector components "yfree(i)" corresponding to the $y_{i,j_{max}}$, and interpolation functions "fi" and "fj" as follows:

```
define x=xbase+(xf-xbase)*y/yf, xbase=fi*xd,
       y=fj*yfree(i)
```

Note that it is permissible to express "x" in terms of "y" or vice versa, but not to use implicit formulations in which "x" and "y" both reference one another.

(2) Specify parameters "xf", "yf", and "xd" in accordance with the known (fixed) geometrical features of the dam, and define "fi" and "fj" so as to produce the desired nodal distribution. As results are generally more accurate when nodes are crowded near exit point B of the free surface, it may be appropriate to type

define fi=(i/i_{max})power pi, fj=(j/j_{max})power pj

where pi and pj are positive numbers less than unity. (In Figure 11, $pi \simeq pj \simeq 0.65$.)

(3) Specify initial estimates for the elevations "yfree(i)" of the free-surface nodes. (In many cases it is satisfactory to use values corresponding to a straight line connecting the known entrance point A to some assumed exit point B.)

(4) Specify the governing equation and boundary conditions shown in Eqs. (13) and (14), using, in particular, the condition $\partial h/\partial n = 0$ along the free surface, by typing

```
set a=e=n=1, b=c=d=f=g=h=0
impose 1 along 0,0, 0,1, 0,jmax
impose 2 along imax,0, imax,1, imax,jmax
impose 0 at imax,jmax
set p(1,1,0)=q(1,1,1)=q(1,1,2)=1,
    q(1,1,0)=p(1,1,1)=p(1,1,2)=0
define r(1,2)=rr(fy grt h2), rr(1)=fy=fit(y)
set rr(0)=h2, r(1,0)=0, r(1,1)=h1
```

Note that no impose-commands are given for the boundary arcs corresponding to base *OD* and free surface *AB*, hence the segment number 0 is assigned automatically by the computer. However, it is necessary to assign segment number 0 explicitly at node (i_{max},j_{max}), i.e., exit point *B*, to insure that the condition $\partial h/\partial n = 0$, not the discharge face condition $h = y$, is used. The computer elects "boundary value conditions" over "boundary slope conditions" when there is a choice, unless explicitly directed to do otherwise.

(5) Obtain an initial solution by using the form- and relax-commands as indicated in Section 6.

(6) Specify a function for calculating new free-surface elevations in terms of parameters "s(i)" representing the sin θ_i of Eq. (17) by typing

```
define newy=yfree(i)+(u(1,i,jmax)-yfree(i))*s(i)
```

Then specify the "s(i)" by typing

```
do(set s(i)=yf/sqrt(yf power 2+(xf-xbase)power 2)
   for i=1 step 1 until i_max)
```

(7) Revise lattice geometry by typing

```
do(set yfree(i)=newy for i=1 step 1 until i_max)
```

and obtain a new solution by typing form- and relax-commands.

(8) Repeat step (7) until adequate convergence is obtained.

Note that the macro specified below could be called in lieu of steps (7) and (8) in order to cause iterations to be repeated until elevation changes at exit point B fall below "dy":

```
let adjust(dy) mean
    begin
    set i=i_max  stop
    if abs(newy-yfree(i)) geq dy insert
       begin
       do(set yfree(i)=newy for i=1 step 1 until i_max)
       form  relax  adjust(dy)
       end
    end
```

Generally convergence is slowest at exit point B, and, moreover, elevation errors are usually greatest in the vicinity of point B once convergence has been attained. However, surprisingly accurate results, e.g., maximum elevation errors of less than three percent of upstream water level h_1, can be obtained with fewer than 100 nodes if the nodes are

appropriately distributed and if the discharge face is relatively steep, say with $90° \leq \theta_{i_{max}} \leq 135°$.[†]

B. Solution based on the generation of a flow net. In the procedure described above, the only objectives were the determination of the free surface and the head distribution $h = h(x,y)$. Considered here is a procedure which generates an entire flow net, i.e., it produces a lattice whose grid lines approximate contours of constant h and constant stream function Ψ. Note that the free surface itself is a contour of constant Ψ (see Eq. (16)). An interesting feature of the present procedure is that the free surface is determined automatically, that is, the free-surface nodes are adjusted in exactly the same manner as nodes lying in the interior of the lattice. Further, since the true contours of constant h and Ψ are orthogonal, quadrilateral lattice elements are produced which are virtually rectangular, a fact which is partially responsible for this procedure's increased accuracy.

In brief, the procedure is as follows. An initial lattice having the topology of the desired flow net is specified and numerical solutions for h and Ψ are obtained which satisfy the boundary conditions $h = y$ and $\partial\Psi/\partial n = \partial y/\partial s$ along the estimated free surface. (For convenience, h and Ψ are treated as distinct dependent variables and solutions for both quantities, based on Eqs. (13)-(16), are obtained simultaneously. Obviously, either quantity could be determined from the other by integrating the Cauchy-Riemann equations; however, with EPS this requires

---

[†]See Lo [15], p. 4-33.

more input and does not necessarily save computer time.) Since the initial lattice will presumably not have the proper geometry, neither the additional free-surface conditions $\partial h/\partial n = 0$ and Ψ = constant nor the requirement that (other) grid lines be lines of constant h and Ψ will be satisfied. Instead, there will be differences Δh and $\Delta\Psi$ at most nodes between the desired and realized values of h and Ψ. These differences and numerical values for the derivatives $\partial h/\partial x$, $\partial\Psi/\partial y$, etc. are used to revise lattice geometry, then new solutions for h and Ψ are obtained, and so on, until all requirements are satisfied within tolerable limits.

A workable technique for revising lattice geometry involves the calculation of pairs of <u>full corrections</u> Δx and Δy for all nodes, then the replacement of the nodes' current coordinates (x,y) by new coordinates (x',y') given by

$$x' = x + \mu\Delta x, \quad y' = y + \mu\Delta y. \tag{18}$$

Here μ is a relaxation factor which for best results is typically given a positive value less than unity at the outset, then gradually increased to unity as iterations progress. Full corrections Δx and Δy are determined in the following manner:

(1) For interior and free-surface nodes, where x and y can be adjusted independently, the truncated Taylor expansions

$$\Delta h \simeq \frac{\partial h}{\partial x}\Delta x + \frac{\partial h}{\partial y}\Delta y$$

$$\Delta\Psi \simeq \frac{\partial\Psi}{\partial x}\Delta x + \frac{\partial\Psi}{\partial y}\Delta y = -\frac{\partial h}{\partial y}\Delta x + \frac{\partial h}{\partial x}\Delta y$$

are inverted to give

$$\Delta x = \frac{1}{J}\left(\frac{\partial h}{\partial x}\Delta h - \frac{\partial h}{\partial y}\Delta\Psi \right)$$
$$\Delta y = \frac{1}{J}\left(\frac{\partial h}{\partial y}\Delta h + \frac{\partial h}{\partial x}\Delta\Psi \right) \qquad (19)$$

where

$$J = \left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2 .$$

(The Cauchy-Riemann conditions are invoked in the expansion for $\Delta\Psi$ to minimize the number of derivative calculations.)

(2) For boundary nodes constrained to remain on prescribed arcs, say where x and y are given parametrically in terms of some variable t, i.e., where

$$x = x(t), \quad y = y(t),$$

either Δh or $\Delta\Psi$ is implicitly taken to be zero and full corrections are calculated using

$$\Delta x = x(t+\Delta t) - x(t), \quad \Delta y = y(t+\Delta t) - y(t),$$

where

$$\Delta t = \left(\frac{\partial h}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial h}{\partial y}\frac{\partial y}{\partial t} \right)^{-1} \Delta h \quad \text{if } \Delta\Psi = 0, \qquad (20)$$

$$\Delta t = \left(\frac{\partial \Psi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial y}{\partial t} \right)^{-1} \Delta\Psi \quad \text{if } \Delta h = 0. \qquad (21)$$

For example, along a fixed boundary arc required by boundary conditions to be a line of constant h (e.g., entrance face OA of the dam), nodes are adjusted so as to assume desired stream function values by using Eq. (21).

The justification for Eqs. (18)-(21) is perhaps clear for situations

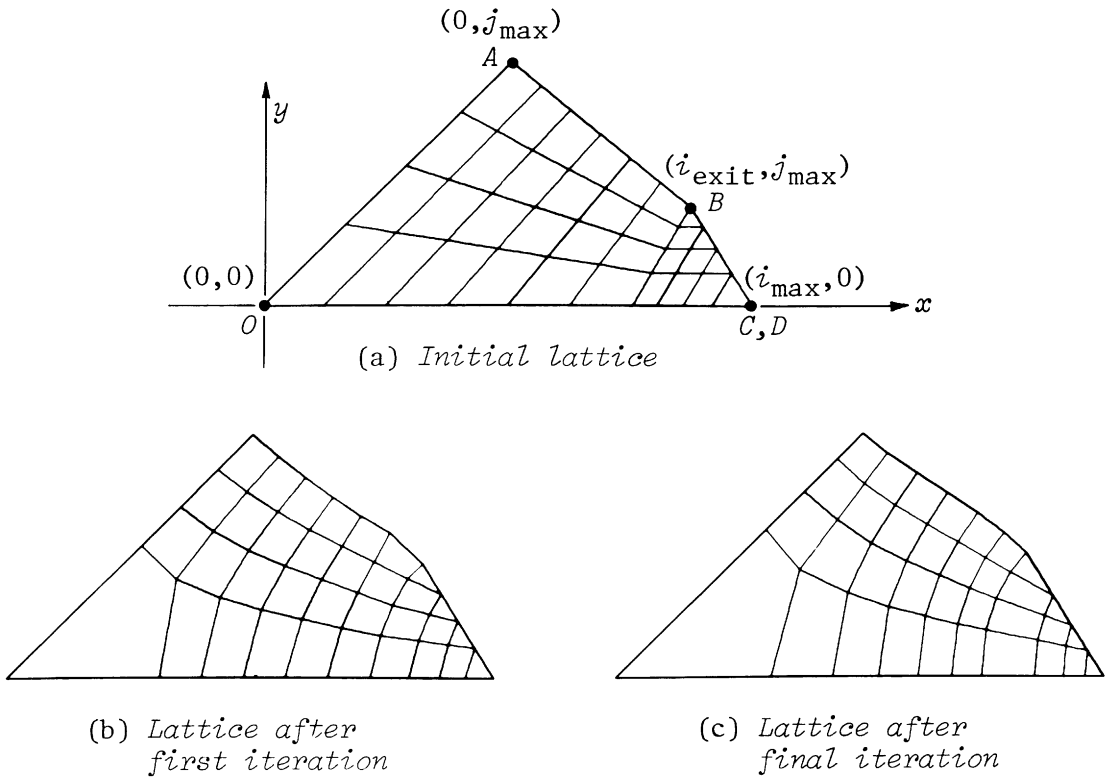
having no free surface, hence entailing no change in the (true) distributions $h = h(x,y)$ and $\Psi = \Psi(x,y)$ from iteration to iteration. In such cases the procedure shown here is nothing more than a Newton-Raphson iteration with a relaxation factor. A primary reason for the relaxation factor is, of course, to make it possible to prevent the free surface from shifting too abruptly and thus to insure that the current derivatives of h and Ψ provide a meaningful basis for extrapolation.

As a final point before turning to the subject of computer input, it should be noted that to avoid instabilities it may be necessary to use modified expressions for Δx and Δy at nodes where the derivative combinations appearing as denominators in Eqs. (19)-(21) tend to be small. This is the case, for example, along exposed discharge face BC of the dam when nodes are adjusted so as to assume prescribed stream function values. Here it is necessary to use

$$\Delta t = (t - t_C) \frac{\Delta\Psi}{(\Psi - \Psi_C)} \tag{22}$$

or some similar expression, rather than Eq. (21), which ordinarily would be used in such a situation. (In theory, the denominator of Eq. (21) is identically zero at exit point B.)

To describe actual computer input corresponding to the above procedure, it is convenient to consider the special (but practical) case of a dam with no tailwater, i.e., a configuration in which downstream water level h_2 is reduced to zero, leaving the entire discharge face exposed to the atmosphere. A typical solution for such a configuration is illustrated by Figure 12. Thirteen iterations were required to produce the final lattice of Figure 12c, at which point variations in the numerical



(a) *Initial lattice*

(b) *Lattice after first iteration*

(c) *Lattice after final iteration*

Fig. 12. *Graphical displays*† *illustrating the iterative generation of a flow net for a dam with no tailwater.*

solutions for h and Ψ along the grid lines of constant i and j, respectively, were less than 0.2 percent. The appearance of the final lattice differs from that of a conventional flow net principally because the nodes are connected by straight lines rather than smooth curves and, in addition, because the grid lines do not represent equal increments of h and Ψ. Here, in fact, the grid lines of constant j have Ψ-values which are distributed according to

$$\Psi = \left(\frac{j}{j_{max}}\right)^{.8} \Psi_A .$$

†Produced by "eps-p saved" on the ESL display console by typing "plot nil", then "plot 0$" (labels are added). See Section 7.2.

The lines of constant i represent equal increments of h only in the region between entrance face OA ($i = 0$) and the line of constant i whose upper end coincides with exit point B ($i = i_{exit}$). For $i \geq i_{exit}$, h-values are determined by the elevations at which the lines of constant j intersect discharge face BC (see boundary conditions, Eq. (14)).

To obtain a solution such as that of Figure 12, it is convenient to use the following procedure:

(1) Specify lattice structure in i,j-space (see Section 3.4), then define a mapping algorithm in terms of matrices "xx" and "yy" by typing

```
define x=xx(i,j), y=yy(i,j)
```

Note that it is easy to have the computer generate initial components for "xx" and "yy" (by means of appropriate do-loops) if, as in Figure 12a, the initial lattice is made up only of straight lines.

(2) Specify governing equations and boundary conditions corresponding to Eqs. (13)-(16) using the boundary conditions $h = y$ and $\partial\Psi/\partial n = \partial y/\partial s$ along the estimated free surface. (Note that these conditions also apply along discharge face BC.) For example, with $u_1 \rightarrow h$ and $u_2 \rightarrow \Psi$, type

```
set n=2, a(1,1)=a(2,2)=e(1,1)=e(2,2)=1,
    a(1,2)=a(2,1)=b=c=d=e(1,2)=e(2,1)=
    f=g=h=0
impose 1 along 0,0, 0,1, 0,jmax
impose 2 along 0,jmax, 1,jmax, imax,0
```

```
do( do( do( set p(k,l,m)=q(k,l,m)=0
    for k=1 step 1 until 2)
    for l=1 step 1 until 2)
    for m=0 step 1 until 2)
set p(1,1,0)=q(1,1,1)=q(1,1,2)=1,
    q(2,2,0)=p(2,2,1)=p(2,2,2)=1,
    r(1,0)=r(2,0)=r(2,1)=0, r(1,1)=h1
define r(1,2)=fy=fit(y), r(2,2)=ys
```

where, as before, h_1 is the upstream water level. (Note the use of the built-in function "ys", representing $\partial y/\partial s$.)

(3) Obtain initial solutions for h and Ψ by using form- and relax-commands as indicated in Section 6.

(4) Define parameters "ph" and "pp" giving desired values of h and Ψ, respectively, in terms of the computed value "u(2,0,j_{max})" of Ψ at entrance point A, two interpolation functions "fi" and "fj", and the anticipated new elevations "yy(i,i_{max}-i)+mu*dyy(i,i_{max}-i)" of nodes on discharge face BC by typing

```
define ph=phf(i geq iexit), pp=fj*u(2,0,jmax),
    phf(0)=h1+fi*(yy(iexit,jmax)+mu*dyy(iexit,jmax)
    -h1), phf(1)=yy(i,imax-i)+mu*dyy(i,imax-i)
```

Then specify "fi" and "fj" to indicate the desired contour distributions, e.g., type

```
define fi=i/iexit, fj=(j/jmax)power .8
```

to obtain the distribution used for the solution of Figure 12.

(5) Specify parameters "dh" and "dp" representing the differences

between desired and computed values of h and Ψ, respectively, by typing

```
define dh=ph-u(1,i,j), dp=pp-u(2,i,j)
```

Then specify full corrections "dx" and "dy", as given by Eq. (19), for the interior and free-surface nodes by typing

```
define dx=(hx*dh-hy*dp)/(hx*hx+hy*hy),
       dy=(hy*dh+hx*dp)/(hx*hx+hy*hy),
       hx=ux(1,i,j), hy=uy(1,i,j)
```

Similarly, specify full corrections (i) "dxoa" and "dyoa", based on Eq. (21), for nodes on entrance face *OA*, (ii) "dxbc" and "dybc", based on Eq. (22), for nodes on discharge face *BC*, and (iii) "dxoc", based on Eq. (20), for nodes on base *OC* by typing

```
set co=xx(0,jmax)/yy(0,jmax),
    cc=(xx(iexit,jmax)-xx(imax,0))/yy(iexit,jmax)
define dxoa=co*dyoa, dxoc=dh/hx,
    dyoa=dp/(co*ux(2,0,j)+uy(2,0,j)),
    dxbc=cc*dybc, dybc=dp*yy(i,j)/u(2,i,j)
```

(6) Specify a macro "save" for computing all non-zero components of two correction matrices "dxx" and "dyy" by typing

```
let save mean
    begin
    set i=0
    do( set dxx(0,j)=dxoa, dyy(0,j)=dyoa
        for j=1 step 1 until jmax-1)
    do( set i=imax-j, dxx(i,j)=dxbc, dyy(i,j)=dybc
        for j=1 step 1 until jmax)
```

```
set j=0
do( set dxx(i,0)=dxoc
    for i=1 step 1 until imax-1)
do( do( set dxx(i,j)=dx, dyy(i,j)=dy
    for i=1 step 1 until imax-j-1)
    for j=1 step 1 until jmax)
end
```

Then specify the components of "dxx" and "dyy" which are to be zero for all iterations by typing

```
set dxx(0,0)=dxx(imax,0)=
    dxx(0,jmax)=dyy(0,jmax)=0
do( set dyy(i,0)=0 for i=0 step 1 until imax)
```

(7) Specify functions corresponding to Eq. (18) for calculating revised x,y-coordinates by typing

```
define newx=xx(i,j)+mu*dxx(i,j)
       newy=yy(i,j)+mu*dyy(i,j)
```

Then, with a set-command, assign relaxation factor "mu" a starting value. Generally this value is not too critical as long as it is well below unity; 0.3 was used for the first few iterations of the solution shown in Figure 12.

(8) Revise lattice geometry by typing

```
save
do( do( set xx(i,j)=newx, yy(i,j)=newy
    for i=0 step 1 until imax-j)
    for j=0 step 1 until jmax)
```

and obtain new solutions for h and Ψ by typing form- and

relax-commands.

(9) Repeat step (8), increasing "mu" occasionally, until adequate convergence is obtained.

The following command specifies a recursive "adjust-macro" completely analogous to the one given previously with the procedure based on the use of interpolation formulae:

```
let adjust(tol) mean
    begin
    set i=i_exit, j=j_max  stop
    if abs(dybc) geq tol insert
       begin
       save
       do( do( set xx(i,j)=newx, yy(i,j)=newy
           for i=0 step 1 until i_max-j)
           for j=0 step 1 until j_max)
       form  relax  adjust(tol)
       end
    end
```

As with the previous procedure, convergence is generally slowest in the vicinity of exit point *B*. However, while errors are comparable in both procedures for a given number of nodes and for dams with steep discharge faces, they do not increase as rapidly with the present procedure when the slope of the discharge face is decreased. Further experiments are needed before more precise comparisons can be made.

10.4. Over- and underrelaxation of iterated parameters. In the procedure for generating flow nets given above, a relaxation factor μ

is needed to regulate the amounts by which the x,y-coordinates of the various nodes change from iteration to iteration. Such a factor may also be beneficial in other procedures, either to speed up convergence or simply to make convergence possible in cases where instability would otherwise be a problem.

In general, a procedure which does not involve a relaxation factor can be modified so as to include such a factor as follows. Suppose, first, that the procedure normally uses a value (or set of values) α_n for some iterated quantity (or set of quantities) α in order to produce the n^{th} solution. Then to introduce a relaxation factor μ, it is necessary only to substitute for α_n per se the weighted average

$$\bar{\alpha}_n = (1-\mu)\,\bar{\alpha}_{n-1} + \mu\,\alpha_n = \bar{\alpha}_{n-1} + \mu\,(\alpha_n - \bar{\alpha}_{n-1}).$$

The modified procedure is identical to the normal procedure when μ is one, generating in this case an increment $\Delta\alpha_n = (\alpha_n - \bar{\alpha}_{n-1}) = (\alpha_n - \alpha_{n-1})$ in preparation for the n^{th} solution. However, for any single iteration the modified procedure will generate a smaller increment for α than the normal procedure if μ is less than unity (underrelaxation). Conversely, the modified procedure will generate a larger increment than the normal procedure if μ is greater than unity (overrelaxation). Depending on the situation, either of these effects may be desirable. For example, if α converges monotonically with the normal procedure, it may be possible to accelerate convergence by using overrelaxation. On the other hand, if α normally behaves in an erratic manner and/or diverges, underrelaxation may yield better results. In any event, the actual mechanics of introducing a relaxation factor are quite straightforward. This can be illus-

trated by reconsidering two of the examples presented earlier.

A. Compressible flow. In the procedure for solving compressible flow problems (Example *B*, Section 10.2), a relaxation factor can easily be introduced at the point where the matrix components "vvm(i,j)", i.e., the current values of $|\text{grad } \Phi|^2$ at the various nodes, are recomputed. Specifically, it is necessary only to redefine the auxiliary parameter "vvfcn", used by macro "revise", in the following manner:

```
define vvfcn=(1-mu)*vvm(i,j)+mu*(ux(1,i,j)power 2
        +uy(1,i,j)power 2)
```

In test cases it has been observed that without a relaxation factor the "vvm(i,j)" tend to converge monotonically in much the same manner as in successive solutions obtained analytically by means of the Rayleigh-Janzen method. Therefore one would expect the optimum value of "mu" to be greater than unity.

B. Nonlinear heat conduction. In the procedure for solving conduction problems with temperature-dependent conductivities (Example *A*, Section 10.2), equation coefficients $a_{11} = e_{11} = (u_1/\theta_o)^s k_o$ are given directly in terms of the current solution matrix "u", which is overwritten to produce the "next" solution during each iteration. Since using a relaxation factor to calculate some iterated quantity $\bar{\alpha}_n$ requires the previous value $\bar{\alpha}_{n-1}$, it is necessary to modify the original procedure so that previous values for either the coefficients $a_{11} = e_{11}$ or the solution u_1 are saved. Choosing the former approach, the user could type

```
define a=e=fit(k(i,j)), kfcn=(1-mu)*k(i,j)+
                mu*k_o*(u(1,i,j)/θ_o)power s
```

then have matrix "k" refreshed before requesting each new solution by using auxiliary parameter "kfcn" and a macro analogous to the revise-macro cited above. Clearly the added processing implied by this modification must be weighed against the possible benefits of over- or under-relaxation.

As a final point, note that it is not necessary to use the same relaxation factor with all iterated parameters. In fact, in cases where iterations serve more than one purpose, the added control provided by multiple factors may be very desirable. While no experiments have been performed to confirm this point, it seems likely that the procedure for generating flow nets which was discussed in Section 10.3 could be improved by using two factors, one to regulate the rate of change of the free-surface nodes and a second to control the changes of all other nodes.

11. Disk input and output

EPS was designed strictly for on-line operation, but does provide a rudimentary mechanism for user-present disk I/O. With this mechanism it is possible to direct the computer (from a console) to take subsequent input from a disk file prepared previously by standard CTSS procedures. Similarly, it is possible to cause the computer to produce a disk file which will, in essence, be a transcript of all subsequent console interactions. Disk input is especially convenient in situations where several problems requiring common definitions are being treated, while disk output is useful when results generated by EPS are to be used as input to other programs. (Output files are written in a format which is compatible with standard CTSS editing and printing facilities, and generally can be made compatible with other application programs without difficulty.)

11.1. Disk input. Disk files which are to be used as EPS input must be written in line-marked format and must represent normal EPS commands and macro calls with CTSS 6-bit character code.† Such files can be created conveniently by means of the CTSS editing program, EDL, which is described thoroughly in [2], Sections AH.3.07 and AH.9.01. Two minor restrictions not specifically imposed by CTSS and EDL must be observed: (1) files must contain no lines with more than 83 characters, and (2) file names must not be reserved EPS vocabulary words (see Section 2). Once a suitable input file, say $name_1$ $name_2$, has been created, the user

†See [2], secs. AB.2 and AC.2.

can "resume eps" and, when it is appropriate, type

read *name*$_1$ *name*$_2$$

to cause the computer to begin reading the file. Note that file names which normally would not be interpreted as identifiers by EPS must be introduced as literals. For example, to read the file "*cases 5a,b", type "read bcd /*cases/ bcd /5a,b/$". See Section 15.1.

As with keyboard input, file reading proceeds until a line containing a "$" is encountered, whereupon the input preceeding the "$" is processed. Normally this cycle is repeated until either all of the file is read or an error is detected, at which time control is returned to the user. An example is provided by the following sequence:

```
edl
W 936.7
INPUT.
let tabulate(z for v=b,d,e) mean
    begin
    index
    expand begin v,z: end
    do(print v,z using format
       for v=b step d until e)
    index
    end $
let index mean begin expand begin end end
let format mean begin bcd +f4.2,8f9.5*+ end$
define sinh=(exp(x)-exp(-x))/2,
       cosh=(exp(x)+exp(-x))/2
tabulate(sinh,cosh for x=0,0.2,1)$

EDIT.
file alpha beta
*
R 4.050+2.037


resume eps
W 945.2
```

Creation of input file.

```
PROCEED:
read alpha beta$
FILE ALPHA   BETA HAS BEEN OPENED.
INDICATED SUBSTITUTION FOR 'TABULATE' WILL BE MADE.
INDICATED SUBSTITUTION FOR 'INDEX' WILL BE MADE.
INDICATED SUBSTITUTION FOR 'FORMAT' WILL BE MADE.

X,SINH,COSH:
0.00  0.00000  1.00000
0.20  0.20134  1.02007
0.40  0.41075  1.08107
0.60  0.63665  1.18547
0.80  0.88811  1.33743
1.00  1.17520  1.54308

END OF FILE ENCOUNTERED.  ALPHA   BETA HAS BEEN CLOSED.

PROCEED WITH CONSOLE INPUT:
define tanh=sinh/cosh
tabulate(tanh for x=.2,.2,1)$

X,TANH:
0.20  0.19738
0.40  0.37995
   ETC.
```

Response to input file.

Continuation of normal I/O.

From this example it will be seen that EPS responds to commands in a disk file in exactly the same manner as it does to commands supplied directly from the keyboard. Thus, for most purposes, disk files can be considered logically equivalent to (parameterless) macros. In fact, they offer two advantages over macros: (1) they do not consume core storage and (2), since they can be prepared with a text editor, they are easier to specify. Of course, they also have a disadvantage, namely, increased processing time, and generally should not be used to represent input sequences that are to be repeated often.

One further point should be mentioned. When an error is detected while input is being taken from a disk file, EPS prints whatever error message it would normally print, then returns to the user's console

for further instructions. It does not, however, close the input file in such a situation, but, in fact, will continue reading the file from where it left off if given the command

```
read switch$
```

Thus it is frequently possible to correct the source of an error, then proceed with the reading of an input file, without serious loss.[†]

11.2. Disk output. EPS does not provide disk output as an alternative to console output, but simply offers a special I/O mode in which all information printed at the user's console, including input typed by the user himself, is written concurrently into a disk file. To enter this special mode, the user types a command of the form

record *name*$_1$ *name*$_2$$

where *name*$_1$ *name*$_2$ identifies the file to be written.[§] Provided the user's track quota is not exhausted,[¶] the computer will respond by printing

---

[†]Actually, the command "read switch$" has more general utility. Disk files are allowed to be either active, i.e., in use, or inactive, and the command "read switch$" simply indicates that the present status of an input file is to be reversed. Thus it is possible not only to reactivate an input file which has been deactivated because of an error, but also, by placing a "read switch$" in an input file, to have the file deactivate itself in order to allow for user interaction, e.g., specification of parameters needed for further processing. Of course, after such an interaction, the user must issue a "read switch$" command himself in order to return control to the disk file.

[§]If no *name*$_1$ *name*$_2$ exists, one will be started. Otherwise, output will be appended to the existing *name*$_1$ *name*$_2$. Hence, it is possible to accumulate output from several sessions in the same file.

[¶]See [2], sec. AH.1.04.

FILE *NAME*$_1$ *NAME*$_2$ HAS BEEN OPENED.

After this, all information typed by the user's console printer will be copied onto the disk, except as noted below.

(1) Input cancelled by means of the CTSS "erase" characters (" and #) and "kill" characters (? and @) never reaches EPS and consequently is not recorded (see Section 2).

(2) The recording of all I/O can be discontinued by typing

```
record switch$
```

and restarted by typing the same command again. Thus the recording of interactions which are not of interest can be suppressed at will (see footnote on preceeding page).

Files created by use of the record-command are line-marked and written in CTSS 6-bit character code. Therefore, they can be printed by means of the CTSS PRINT-command† and edited with EDL.

---

†See [2], sec. AH.5.03.

12. Input verification

In most applications it becomes desirable from time to time to have the computer report its current status with respect to a particular class of input or with respect to a given item within a particular class. In extended or interrupted sessions, for example, it may be helpful simply to be reminded of what macro identifiers have been introduced in past input. Or, as a precaution against typing and transmission errors, it may be useful to have the computer report back complicated curve descriptions or parameter definitions. Operations such as these can be carried out by using the EPS list- and review-commands, as discussed below.

12.1. Determining active identifiers. It is possible to obtain listings of all active identifiers in each of one or more classes c_1, c_2, ..., c_N, where the possible classes are "curves", "variables", "macros" and "commands", by typing

list c_1, c_2, ..., c_N

For each c_k for which active identifiers exist, the computer will print a message of the form

C_k CURRENTLY ACTIVE: '$NAME_1$, $NAME_2$, ..., $NAME_M$'.

Thus, for example, to determine what curve and variable identifiers are currently in use, type

list curves, variables $

A possible response would be

```
CURVES CURRENTLY ACTIVE:  'ALPHA, BETA'.

VARIABLES CURRENTLY ACTIVE:  'LIMIT, DELTA, OMEGA,
R, Q, P, H, G, F, E, D, C, B, A, Y, X, J, I, U, UX,
UY, FLUX, UXX, UYY, UYX, UXY, FIT, XS, YS, OCTANT'.
```

(The word "variables" is something of a misnomer, for its use causes constant parameters and built-in functions, as well as variable parameters, to be listed.) Identifiers that have been introduced by the user are always listed in reverse chronological order, the first identifier listed being the one most recently introduced, and so forth.

12.2. Determining individual definitions. It is possible to obtain output giving current definitions for one or more curves, variables, or macros by typing a command of the form

$$\text{review} \left\{ \begin{matrix} \text{curve} \\ \text{variable} \\ \text{macro} \end{matrix} \right\} \text{begin } d_1,\ d_2,\ \ldots,\ d_N \text{ end}$$

(The braces { } are not meant literally, but merely signify that a choice is available.) The d_k may be single identifiers or they may be identifiers followed by certain qualifiers, depending on what class of information is being reviewed and on what output is desired. In particular, when curves are being reviewed, the d_k may assume two forms:

curvename or *curvename* along i_1,j_1, i_2,j_2, i_3,j_3

The second, or qualified, form is used when output is to be restricted to the portion of *curvename* passing from i_1,j_1 through i_2,j_2 to i_3,j_3.

Similarly, when variables are being reviewed, the d_k may take the form

$$paramname \quad \text{or} \quad paramname(s_1,s_2,\ldots,s_M)$$

where the s_k represent subscripts and may, in general, be arbitrary algebraic expressions. It should be noted that the number of subscripts M need not be as great as the actual number of subscripts associated with a given parameter. In fact, by omitting one or more subscripts, the user can obtain output indicating the current dimensions of a subscripted quantity (see example below). Finally, when macros are being reviewed, each d_k must be a single identifier; argument lists, for example, are not permitted here as they are with the expand-command discussed in Section 9.5.

Output generated by means of the review-command is largely self-explanatory, as is illustrated by the following sequence:

```
PROCEED:
let mtx(w=a,b,c,d,) mean
    begin
    set w(1,1)=a, w(1,2)=b, w(2,1)=c, w(2,2)=d
    end$
INDICATED SUBSTITUTION FOR 'MTX' WILL BE MADE.

PROCEED:
review macro begin mtx end$

MTX(W=A,B,C,D,) MEANS SET W(1,1)=A, W(1,2)=B, W(2,1)=C,
W(2,2)=D

PROCEED:
set n=2, b=c=d=f=h(2)=0
mtx(a=1,0,0,-1, e=1,0,0,-1, g=0,0,1,0)
define fx=fit(x), h(1)=po*(3.75+fx*fx)/s
review variable begin a,b,e,g,h(1),h(2) end$

A=(A(1,1)...A(2,2))
B=0
E=(E(1,1)...E(2,2))
```

```
G=(G(1,1)...G(2,2))
H(1)=PO*(3.75+FX*FX)/S
H(2)=0

PROCEED:
review variable begin a(1,1),a(2,2),e(1),e(2),po,fx,s end$

A(1,1)=1
A(2,2)=-1
E(1)=(E(1,1)...E(1,2))
E(2)=(E(2,1)...E(2,2))
'PO' NOT SPECIFIED.
FX=FIT(X)
'S' NOT SPECIFIED.

PROCEED:
append 0,0, 3,0, 3,3, 0,3 to ca    close ca
append 1,1, 2,1, 2,2, 1,2 to cb    close cb
impose 4 along 0,0, 2,0, 3,0
impose 5 along 0,0, 0,2, 3,0
impose 6 along 1,1, 2,2, 1,1
impose 7 at 3,0 $
DEFINITION OF NEW CURVE 'CA' HAS BEEN COMPLETED.
'CA' HAS BEEN CLOSED.
DEFINITION OF NEW CURVE 'CB' HAS BEEN COMPLETED.
'CB' HAS BEEN CLOSED.

PROCEED:
review curve begin ca along 0,1, 0,0, 3,2, cb end$

GRID COORDINATES AND LOADING INDICES OF 'CA':
 (     0,     1)     (     5,     5)
 (     0,     0)     (     5,     4)
 (     1,     0)     (     4,     4)
 (     2,     0)     (     4,     4)
 (     3,     0)     (     7,     7)
 (     3,     1)     (     5,     5)
 (     3,     2)     (     5,     5)

GRID COORDINATES AND LOADING INDICES OF 'CB':
 (     1,     1)     (     6,     6)
 (     2,     1)     (     6,     6)
 (     2,     2)     (     6,     6)
 (     1,     2)     (     6,     6)
```

It will be seen that curve reviews give <u>two</u> segment numbers (i.e., "LOADING INDICES") for each boundary node. The first segment number corresponds to the boundary arc immediately preceeding the node, and

the second to the arc immediately following the node, where direction of travel is implied by the order in which the nodes are listed. The curves specified in this example would not be specified in practice, since they correspond to a lattice in which all nodes are boundary nodes! However, the equation parameters have practical significance: they correspond to the plate equation with constant bending stiffness "s" and variable pressure "po*(3.75+fx*fx)" (see Section 4.3, Example *C*).

13. Program interruption and run termination

Occasionally the user will wish to prevent or curtail the processing of a unit of input, either because he recognizes that he has made an error or simply because he changes his mind. Further, he will wish at some point to quit EPS completely and have control passed back to the CTSS supervisor. These operations can be accomplished by means of the program interrupt button and the EPS quit-command, as described below.

13.1. Interrupts. Normal processing can be discontinued at any time by pressing once the program interrupt button (labeled "INT", "BREAK", "RESET", or "INTERRUPT", depending on console type). This action causes the computer to ignore any commands supplied since the last "PROCEED:" but not yet executed, and with certain minor exceptions results in the response

```
INT. 0
PROCEED:
```

After this, normal operation of the program can be continued. It is emphasized that interrupts have no ability to "undo" commands once they have been executed. Thus, in particular, data specified in previous interactions is neither lost nor disturbed in any way.

The user should note that if the interrupt is pressed during the specification of input, i.e., before a "$" and final carriage return have been given, no commands supplied since the last "PROCEED:" will have been executed. Thus in this situation the interrupt is anal-

ogous to the CTSS "kill" characters, ? and @ (see Section 2), except that it causes all preceeding lines of the unit of input, as well as the current line, to be ignored. (Also, of course, it causes output to be produced, including a new "PROCEED:". The latter must be awaited before typing can be resumed.)

On the other hand, if the interrupt is pressed after a "$" and final carriage return have been given, any number of commands, including none, may in general have been executed. In most cases, output produced before the interrupt will indicate how far processing has gone; if not, appropriate print- and review-commands may be used to ascertain this information (see Sections 7 and 12).

To avoid confusion, the user should be warned that interrupts often do not "take effect" immediately. Typically, the machine will require several seconds, say 5 to 15, to respond with "INT. 0", then possibly several more to produce a "PROCEED:". Moreover, because EPS normally completes the command it is currently executing (if any) before actually obeying an interrupt, it is possible to receive a small amount of regular output after the line "INT. 0" and before the new "PROCEED:".[†] Therefore, patience is necessary; attempting to hurry the machine by pressing the interrupt repeatedly will either delay matters further or, as indicated in the next section, produce a completely different effect.

[†]EPS allows itself to be interrupted in "mid-command" only when it is generating or solving finite-difference equations. See descriptions of the form- and relax-commands in sec. 15 for further details.

13.2. Run termination. Control can be returned to the CTSS supervisor at any time by pressing the interrupt button twice, or, at times when EPS is in input status, by typing the command

```
quit $
```

It is necessary to use the interrupt button if immediate termination is desired at a time when an extended calculation is in progress. However, for reasons given later, it is recommended that the quit-command be used whenever possible. In either case, the computer will ultimately respond with a line of the form

R ΔT_1+ΔT_2

where ΔT_1 and ΔT_2 give computation time and overhead, respectively, in seconds. After this, commands must be directed to CTSS. For completeness, three common situations are considered.

(1) The user may realize immediately after quiting EPS that he has not finished his computations. To continue, he can type

```
start
```

whereupon the computer will resume execution of EPS at the point where it left off.† Note that this procedure can be used deliberately to determine how much computer time is being used.

(2) The user may anticipate continuing his computations at a

†Cf. [2], sec. AH.7.03.

later time, in which case he can type

save *name*$_1$

to create a disk file "*name*$_1$ saved" to be resumed in the future.[†§] In applications requiring several iterative stages, it may be wise to create "saved" files at various points along the way in order to avoid having to recalculate earlier results if later strategies fail.

(3) The user may be satisfied with his results, certain that he is finished, and ready to leave his console, in which case he can log out as per [2], Section AH.1.02.

It should be noted that output files opened by means of the record-command (Section 11.2) are closed when and only when a quit-command is received; such files may be lost if the interrupt button is used to terminate a run. Further, if EPS is restarted (or saved and resumed) after being terminated by means of the interrupt button, it may enter input status without printing the usual "PROCEED:". To determine whether or not the computer is, in fact, awaiting input in this and other situations, type a "$" and carriage return. This has no computation effect, but insures that the computer will print a "PROCEED:" when it next needs input.

[†]Cf. [2], sec. AH.3.03.

[§]Disk files created in this manner may be from approximately 30 to 78 records in length, depending on which version of EPS is being used and on how much information has accumulated in core memory as a result of previous interactions.

14. Error messages

Whenever continued processing of an input sequence is impossible because of bad form, invalid or insufficient data, or the detection of an illegal machine condition, the computer prints an error message indicating specifically what difficulty has been encountered. Depending on what stage processing has reached and on the particular difficulty that has arisen, error messages may either demand a special response from the user or simply state that processing has been cut short and ask, by way of a "PROCEED:" message, for more input. In either case, messages are in principle completely self-explanatory, except in cases of input conversion errors, which are reported in terms of a numerical error code *N* and a line image *L* as follows:

ERROR *N* DETECTED IN SCANNING:
L

RETYPE THIS LINE AND ANY OTHERS WHICH FOLLOW:

Usually messages of this type are incurred as a result of poor console technique (see Section 2), as may be suggested by the list of possible codes given below.

N	ERROR CONDITION
1	Memory overflow (very unlikely in this context)
2	Incomplete or possibly truncated word or number (usually a line with more than 83 characters)
3	Number greater than 1.701412E+38
4	Illegal character sequence, e.g., "34beta"

Such messages are rare once the user has gained a little experience.

Much more likely, in general, are error messages resulting from the inadvertent omission of elements in algebraic expressions. Fortunately, the computer is more generous in such cases. All algebraic input to EPS is checked for syntax errors, i.e., missing (or excessive) parentheses, operators, and operands, and "corrected" when necessary by means of a simple but surprisingly effective strategy. "Corrected" expressions are reported with messages of the form

```
INCORRECT INPUT STRING MODIFIED TO READ 'EXPR'.
IS THIS WHAT YOU MEAN...
```

in response to which the user is expected to type "yes" or "no". If he types "yes", the computer will resume its normal operation, so presumably no loss will be suffered. If he types "no", the computer will execute a fatal error sequence, which means it will discard the command containing the erroneous expression, as well as any subsequent commands waiting to be processed. This action may or may not be explicitly acknowledged in further error messages, but in all cases it will necessarily take place. Note that it is perfectly legitimate to answer "yes", even when an improper correction has been made, in order to avoid retyping valid input which would otherwise be discarded. Afterwards improperly corrected information can be respecified, as shown in Section 8.

Fatal error sequences are executed in general when commands are found to contain non-correctable syntax errors, and also when they are given in terms of invalid data. However, non-fatal error sequences are normally executed when commands cannot be completed because of insuf-

ficient data, inadequate memory, or any other conflict potentially resolvable through subsequent user interaction. In such situations, input sequences of more than one element which have not been completely processed are not discarded, but instead are saved in a special push-down list called the command stack. Processing of the latest, i.e., top-most, input sequence on the command stack can be reinitiated at any time by means of the command "retry", while deletion of the top-most sequence can be effected by means of the command "ignore". In practice this arrangement is both natural and convenient, as the following example may illustrate:

```
PROCEED:
define radius=i*sqrt(c2+sqrt(c2*c2+15))/10
    c2=cos(2*t), t=j*pi/20
set i=10, j=0    print t,radius
set j=10      print t,radius$
PROCESSING HALTED ON CALL FOR UNDEFINED QUANTITY
'PI', WHICH WAS REFERENCED BY 'T'.
REDUCTION OF 'T,RADIUS' CANNOT BE COMPLETED.

COMMAND STRING 'PRINT T,RADIUS SET...' HAS BEEN
SAVED.  TYPE 'RETRY' TO REINITIATE PROCESSING,
'IGNORE' TO HAVE STRING DELETED.

PROCEED:
set pi=4*atan(1)    retry$
   0.00000000000   2.2360683E+00
   1.5707963E+00   1.7320509E+00
```

Clearly, command sequences which cause non-fatal errors will accumulate indefinitely in the command stack unless they are successfully retried or deleted by means of an "ignore". To avoid unnecessary storage demands, the latter command should be typed immediately if it is recognized that a given sequence will not be retried.

15. Command descriptions

In this section, descriptions of all of the EPS commands are presented in outline form. These descriptions take a broader view of the system than does the material of preceeding sections. In addition, they introduce a certain amount of new information, e.g., optional input forms, brief explanations of how the commands are actually processed, and listings of the conditions under which error sequences are executed. Thus this section should be of particular interest to experienced users who wish to pursue new applications. Furthermore, it brings together in a form convenient for quick reference information which may be useful to any user who encounters difficulties while using the computer.

15.1. Terminology. For each command, descriptive material is entered in a fixed order under the following headings:

PURPOSE	NORMAL RESPONSE
REFERENCES	SIDE EFFECTS
FORMAT	INTERRUPTS
EXAMPLES	NON-FATAL ERRORS
NORMAL OPERATION	FATAL ERRORS

Entries are omitted when inappropriate, e.g., if a command normally produces no response, the NORMAL RESPONSE entry is omitted.

The material is largely self-explanatory, except possibly for some of the notation and terminology used in certain FORMAT entries. For most commands, the FORMAT entries define admissible input forms in two steps. First, with a notation which is a slight extension of that employed in preceeding sections, basic structure is defined by an expres-

sion in which:

(1) information intended literally is shown in sans-serif type;

(2) command variables, or arguments, are shown in italics or other easily distinguished type styles;

(3) alternatives are listed vertically between braces {}; and

(4) information which may be omitted under certain conditions is enclosed in square brackets [].

Thus, for example, the FORMAT entry for the delete-command begins with the expression

$$\text{delete} \left\{ \begin{array}{c} [\textit{curvename}] \\ i_1,j_1,\ [i_2,j_2,]\ i_3,j_3\ [\text{from } \textit{curvename}] \end{array} \right\}$$

This indicates that a delete-command may assume the following basic forms:

delete
delete *curvename*
delete $i_1,j_1,\ i_3,j_3$
delete $i_1,j_1,\ i_3,j_3$ from *curvename*
delete $i_1,j_1,\ i_2,j_2,\ i_3,j_3$
delete $i_1,j_1,\ i_2,j_2,\ i_3,j_3$ from *curvename*

The appropriateness of any given form depends, of course, on the action desired by the user, as well as on conditions explained in the NORMAL OPERATION entry.

After basic structure has been defined, admissible forms of arguments, where allowed, are specified verbally. Thus in the case of the delete-command, the FORMAT entry concludes with the phrase

where *curvename* is an identifier and the i_k, j_k are algebraic expressions ($1 \leq k \leq 3$).

The meanings of terms such as "identifier" and "algebraic expression" may at this point be intuitively clear, but for completeness are given informally below, along with various related definitions. Starred items define terms actually used in command descriptions.

letter: one of the 26 letters of the alphabet, a single quotation mark ('), a form feed, or a period.

digit: one of the 10 decimal digits.

printed delimiter: one of the following: + - / * ( ) = : , $

character: a letter, digit, printed delimiter, blank, or tab.

normal word: a string of up to 83 letters and digits beginning with a letter. (Restriction: if the string begins with a period, it must consist of only the period, or else the period must be followed by a letter.)

literal: a string s of up to 77 characters introduced by means of an expression of the form "bcd $\lambda s \lambda$", where λ is any character not present in s and where the space between "bcd" and the first λ must be a single blank, not a tab or carriage return.

*word: a normal word or literal.

*integer: a string of one or more digits.

number with decimal point: an integer followed by a decimal point (i.e., period), a decimal point followed by an integer, or an integer followed by a decimal point followed by an integer.

number with scale factor: an integer or number with decimal point

followed by an "e" followed by a sign (+ or -) followed by an integer. The sign may be omitted when "+" is intended.

number: an integer, number with decimal point, or number with scale factor.

*input element: a word, number, or printed delimiter.

*input sequence: a sequence containing any number of input elements, including none. (Blanks, tabs, and/or carriage returns must be used to separate adjacent words and numbers, but are not considered part of the sequence per se.)

reserved command word: any word shown in sans-serif type in the summary of commands given in Table 1, Section 16.

function name: any word shown in sans-serif type in the summary of built-in functions given in Table 2, Section 16.

multiliteral operator: any word shown in sans-serif type in the summary of operators given in Table 3, Section 16.

reserved word: a reserved command word, function name, or multiliteral operator; also any one of the following: if insert bcd

*identifier: any word which is not a reserved word.

*parameter designator: an identifier, or, if the parameter is subscripted, an identifier followed by an expression list enclosed in parentheses.

function call: a function name, or, if the function requires arguments, a function name followed by an expression list enclosed in parentheses (see Table 2, Section 16).

unary operator: any input element designated as a unary operator in Table 3, Section 16.

diadic operator: any input element designated as a diadic operator in Table 3, Section 16.

*algebraic expression: a number, parameter designator, or function call, or, proceeding recursively, a sequence of the form

$$o_1\ e \qquad \text{or} \qquad e\ o_2\ e' \qquad \text{or} \qquad (e)$$

where o_1 is a unary operator, o_2 is a diadic operator, and each of the e's is itself a legitimate algebraic expression. For operator precedence information, see Table 3, Section 16.

expression list: a single algebraic expression or a sequence of algebraic expressions, each separated from the next by a comma.

15.2. Descriptions. Command descriptions follow. Underlined section numbers listed as REFERENCES indicate sources of background information and/or examples which are particularly relevant. The headings FATAL ERRORS and NON-FATAL ERRORS refer specifically to conditions under which fatal sequences and non-fatal error sequences, respectively, are initiated (see Section 14).

append-command

PURPOSE: To define or extend a bounding arc.

REFERENCES: Section 3, especially 3.4, and Section 8.1.

FORMAT: append i_1,j_1, i_2,j_2, ..., i_N,j_N [to *curvename*]

where the i_k,j_k are algebraic expressions ($1 \leq k \leq N$, $N \geq 1$) and where *curvename* is an identifier.

EXAMPLES:
```
append -5,0, 5,0, 5,9, -5,9 to zeta
append 0,0, imax,0, imax+jmax,jmax, 0,jmax
```

NORMAL OPERATION: The i_k,j_k are evaluated, rounded to nearest integers, and interpreted as the grid coordinates of successive vertices (in i,j-space) of a bounding arc. A list of all nodes on the arc, including nodes on the grid lines and grid diagonals formed by connecting the vertices, is constructed. If the phrase "to *curvename*" is specified but identifier *curvename* is not presently active, the node list is taken as the definition of a new arc. If "to *curvename*" is specified and found to refer to an existing, unclosed arc with an end point which coincides with either i_1,j_1 or i_N,j_N, then the existing arc is extended in accordance with the new node list. If "to *curvename*" is omitted, a new arc with a computer-generated name is defined.

NORMAL RESPONSE: Depending on whether a new definition or an extension is introduced, either

DEFINITION OF NEW CURVE '*CURVENAME*' HAS BEEN COMPLETED.

or

ADDITION TO '*CURVENAME*' HAS BEEN COMPLETED.

SIDE EFFECTS: Deletion of the difference equation coefficients and solution matrix generated as a result of any previous form-command and of the data base generated as a result of any previous plot-command.

NON-FATAL ERRORS: Memory overflow and references by the i_k,j_k to unde-

fined quantities.

FATAL ERRORS: Non-correctable syntax errors (e.g., only one pair of i,j-coordinates) and illegal data (e.g., successive vertices which cannot be connected by a single grid line or grid diagonal).

close-command

PURPOSE: To cause the end points of a bounding arc to be connected so as to form a closed contour.

REFERENCES: Section 3, especially 3.4, and Section 8.1.

FORMAT: close [*curvename*]

where *curvename* is an identifier.

EXAMPLES: close zeta
close

NORMAL OPERATION: The end points of contour *curvename*, if *curvename* is specified, or of the only contour which exists (presuming only one exists), if *curvename* is omitted, are inspected. If these points are different and can be joined by a grid line or grid diagonal, then an appropriate node list is generated and inserted between the existing end points. If the existing end points are identical, they are simply coalesced.

NORMAL RESPONSE: '*CURVENAME*' HAS BEEN CLOSED.

NON-FATAL ERRORS: Memory overflow and attempts to close contours whose end points cannot be connected by a single grid line or grid diagonal.

FATAL ERRORS: Omission of the argument *curvename* when more than one contour exists and specification of a *curvename* which does not exist.

define-command

PURPOSE: To specify functional relationships between algebraic parameters.

REFERENCES: Sections 3.5, 4.2, 4.3, 5.3, 7.1, 8.2, 10.2, and 10.3.

FORMAT: define $p_{11} = p_{12} = \ldots = p_{1M_1} = e_1$, $p_{21} = p_{22} = \ldots = p_{2M_2} = e_2$, $\ldots$, $p_{N1} = p_{N2} = \ldots = p_{NM_N} = e_N$

where the $p_{k\ell}$ are parameter designators and the e_k are algebraic expressions ($1 \leq k \leq N$, $N \geq 1$; $1 \leq \ell \leq M_k$, $M_k \geq 1$).

EXAMPLE: `define a=e=2+fx*fx, fx=fit(x), r(2,m)=fx/3.47`

NORMAL OPERATION: Argument groups of the form

$$p_{k1} = p_{k2} = \ldots = p_{kM_k} = e_k \qquad \text{(e.g., "a=e=2+fx*fx")}$$

are rewritten so as to become

$$p_{kM_k} = e_k,\ p_{kM_k-1} = p_{kM_k},\ \ldots,\ p_{k1} = p_{k2} \qquad \text{(e.g., "e=2+fx*fx, a=e")}$$

Thus, in effect, the original command is made to assume the form

$$\text{define } p_1 = e_1,\ p_2 = e_2,\ \ldots,\ p_L = e_L$$

For each assignment $p_k = e_k$, a subroutine for calculating e_k is compiled and the data base arranged so that in the future this subroutine is executed each time a value for p_k is required. If p_k has subscripts, these are evaluated and e_k is associated with the particular component that is indicated. Thus if "m" is 3 when the command of the example is specified, "fx/3.47" is associated with the single matrix component "r(2,3)".

SIDE EFFECTS: Previous definitions are deleted and subscript ranges are modified as needed.

NON-FATAL ERRORS: Memory overflow and references in subscripts of the p's to undefined quantities.

FATAL ERRORS: Non-correctable syntax errors.

delete-command

PURPOSE: To delete all or part of a bounding contour.

REFERENCES: Sections 3 and 8.1.

FORMAT: delete $\left\{\begin{array}{c} [curvename] \\ i_1,j_1,\ [i_2,j_2,]\ i_3,j_3\ [\text{from } curvename] \end{array}\right\}$

where *curvename* is an identifier and the i_k,j_k are algebraic expressions ($1 \leq k \leq 3$).

EXAMPLES:
```
delete
delete zeta
delete 0,5, 0,6 from curve2
delete 0,jmax, 2,jmax, imax,jmax
```

NORMAL OPERATION: If no argument is specified and only one contour exists, that entire contour is deleted. If the single argument *curvename* is specified, the entire contour *curvename* is deleted. If a phrase of the form "i_1,j_1, i_2,j_2, i_3,j_3 from *curvename*" is specified, the i_k,j_k are evaluated and rounded to nearest integers, then the portion of *curvename* running from node i_1,j_1 through i_2,j_2 to i_3,j_3 is deleted. The i_k,j_k may specify an arbitrary arc if the contour in question is closed; however, if the contour is unclosed, the specified arc and the unclosed contour must have at least one end point in common--i.e., it is not permissible to delete an intermediate arc from an unclosed contour, breaking the contour into two pieces. The phrase "i_2,j_2," can be omitted at any time if the contour is unclosed, but can be omitted when the contour is closed only if end points i_1,j_1 and i_3,j_3 are neighboring nodes (in which case the single line connecting these nodes is deleted). The phrase "from *curvename*" can be omitted whenever only one contour containing the specified arc exists--i.e., almost always, in practice.

NORMAL RESPONSE: Depending on whether an entire contour or a portion of a contour is deleted, either

'*CURVENAME*' HAS BEEN DELETED.

or

DELETION FROM '*CURVENAME*' HAS BEEN COMPLETED.

SIDE EFFECTS: Deletion of the difference equation coefficients and solution matrix generated as a result of any previous form-command, of the data base generated as a result of any previous plot-command, and of the segment numbers associated with all deleted arcs.

NON-FATAL ERRORS: References among the i_k, j_k to undefined quantities.

FATAL ERRORS: Omission of arguments when more than one contour exists and specification of invalid or ambiguous data.

expand-command

PURPOSE: To test macro definitions and cause printing of arbitrary text strings (e.g., table headings, etc.).

REFERENCE: Section 9, especially 9.5.

FORMAT: expand begin *p* end

where *p* is any input sequence which does not contain a "$" or unmatched begin-end pairs.

EXAMPLE: expand begin this is a text string. end

NORMAL OPERATION: The sequence *p* is passed through the EPS macro processor so that any macro calls it might contain are transformed into corresponding substitutions.

NORMAL RESPONSE: The sequence resulting from the expansion of *p*, if any, is printed. If *p* is the null sequence or "expands to the null sequence" (see comment-macro, Section 9.2), a blank line is printed.

FATAL ERRORS: Memory overflow and references to undefined quantities in the determining expressions of if-phrases (see Section 9.3).

form-command

PURPOSE: To generate finite-difference equations.

REFERENCES: Section 6, especially 6.1, and Section 8.

FORMAT: form

NORMAL OPERATION: Existing difference equation coefficients, if any, are discarded and a tally-command is simulated. If no solution matrix exists (necessarily the case if the form-command is being used for the first time or if an append- or delete-command has been given since the form-command was last used), one is created and set to zero. Similarly, if a solution matrix does exist, but the number of dependent variables "n" has been changed, the old matrix is discarded and a new one is created and set to zero. Otherwise, the existing matrix is left intact. Then (new) difference equation coefficients for the current lattice and differential system are generated. Coefficients corresponding to all "n" partial differential equations or boundary conditions are produced at first one node, then another. In terms of the lattice configuration in i,j-space, processing is from left to right along each grid line of constant j, starting with the line of smallest j, say j_{min}, and proceeding to $j_{min}+1$, $j_{min}+2$, etc.

NORMAL RESPONSE: The response, if any, resulting from the simulated tally-command, then, if a (new) solution matrix is created, the line

SPACE FOR SOLUTION MATRIX HAS BEEN ALLOTTED AND ZEROED.

INTERRUPTS: The user can give a single interrupt signal (see Section 13.1) either to curtail processing entirely, as when he discovers a mistake, or simply to determine how far along processing has come. The computer will respond by printing

GENERATION OF DIFFERENCE EQUATIONS HALTED AT GRID POINT (I,J). DO YOU WISH EQUATION GENERATION TO CONTINUE...

after which the user must type "yes" or "no".

NON-FATAL ERRORS: Illegal conditions recognized by the tally-command, memory overflows, and missing or unacceptable data (e.g., lattice parameters "x" and "y" which describe a lattice with illegal geometry).

ignore-command

PURPOSE: To discard a command sequence saved as a result of a non-fatal error.

REFERENCE: Section 14.

FORMAT: ignore

NORMAL OPERATION: The most recently saved command sequence is discarded and the one preceeding that, if any, is readied for possible retrial.

FATAL ERRORS: ignore-commands given when no saved command sequences exist to be ignored.

impose-command

PURPOSE: To associate a segment number, hence a given set of boundary condition parameters, with a bounding arc.

REFERENCES: Section 5, especially 5.2, and Section 8.3.

FORMAT: impose m $\begin{Bmatrix}\text{along } i_1,j_1,\ [i_2,j_2,]\ i_3,j_3 \\ \text{at } i_1,j_1\end{Bmatrix}$ [of *curvename*]

where m and the i_k,j_k $(1 \le k \le 3)$ are algebraic expressions and *curvename* is an identifier.

EXAMPLES: impose 3 along 0,0, 2,0, imax+jmax,jmax
impose newm at 5,9 of zeta

NORMAL OPERATION: The value of expression m is obtained, rounded to the nearest integer, and interpreted as a segment number to be associated with the arc specified by the remaining portion of the command. If the remaining portion has the form "along i_1,j_1, i_2,j_2, i_3,j_3 of *curvename*", the i_k,j_k are evaluated and rounded to nearest integers, and m is associated with the portion of *curvename* running from i_1,j_1 through i_2,j_2 to i_3,j_3. The phrase "i_2,j_2," can be omitted at any time if the contour in question is unclosed; however, if the contour is closed, it can be omitted only when i_1,j_1 and i_3,j_3 are neighboring boundary nodes (in which case m is applied only along the line connecting the two neighbors). If the remaining portion has the form "at i_1,j_1 of *curvename*", i_1 and j_1 are evaluated and rounded to nearest integers, and m is associated with the arc which includes i_1,j_1 and extends half-way to each of its neighbors (see Section 5.3, remark (3)). In all cases, the phrase "of *curvename*" can be omitted whenever the arc being specified exists on only one bounding contour--i.e., the phrase can be omitted almost always in practice.

SIDE EFFECTS: Existing segment numbers are overwritten without comment.

NON-FATAL ERRORS: References among the i_k,j_k to undefined quantities.

FATAL ERRORS: Non-correctable syntax errors and invalid or ambiguous data.

let-command

PURPOSE: To specify a macro.

REFERENCE: Section 9.

FORMAT: let *macroname*[(p_1 s_1 p_2 ... s_{N-1} p_N [s_N])] mean
begin
$d(p_1;p_2;\ldots;p_N)$
end

where *macroname* is an identifier, the p_k and s_k are arbitrary input elements other than dollar signs or parentheses, and $d(p_1;p_2;\ldots;p_N)$ is an arbitrary sequence containing, in particular, any number of occurrences of each p_k, but not containing dollar signs or mismatched begin-end pairs ($1 \leq k \leq N$, $N \geq 1$).

EXAMPLES:
```
let ff mean begin bcd /7f11.6*/ end
let pr(f:x) mean begin print x using f end
let dump(j from i1 to i2,) mean
    begin
    do(pr(ff:u(1,i,j),u(2,i,j)) for i=i1 step 1 until i2)
    end
```

NORMAL OPERATION: The data base is modified so that during the processing of subsequent input the specified transformation will be performed. For transformation conventions, see Sections 9.1, 9.4, and 9.5.

NORMAL RESPONSE: If *macroname* was already active, a line of the form

MACRO '*MACRONAME*' NOW INACTIVE.

Then, in any event, a line of the form

INDICATED SUBSTITUTION FOR '*MACRONAME*' WILL BE MADE.

NON-FATAL ERRORS: Memory overflow.

FATAL ERRORS: Non-correctable syntax errors.

list-command

PURPOSE: To cause listings of active identifiers to be printed.

REFERENCE: Section 12.1.

FORMAT: list c_1, c_2, ..., c_N

where each of the c_k is a class specifier, namely, one of the four words "curves", "variables", "macros", or "commands" ($1 \leq k \leq N$, $N \geq 1$).

EXAMPLES: list macros, variables, curves
list commands

NORMAL OPERATION: The active identifiers, if any, for each c_k, starting with c_1, are listed in the format shown below. Identifiers introduced by the user are listed in reverse chronological order.

NORMAL RESPONSE: For each c_k, either

C_k CURRENTLY ACTIVE: '$NAME_1$, $NAME_2$, ..., $NAME_M$'.

or

NO C_k ACTIVE.

depending on the current status of the indicated class.

FATAL ERRORS: Syntax errors, e.g., misspelled class specifiers.

plot-command

PURPOSE: To obtain or modify a graphical display.

REFERENCES: Sections 1 and 7.2.

FORMAT: plot $\left\{\begin{array}{l} \left\{\begin{array}{l} z \\ \texttt{again} \end{array}\right\} \text{ [sans } op_1, op_2, \ldots, op_N\text{]} \\ \texttt{nil} \\ \texttt{off} \\ \texttt{discard} \\ \texttt{movie } n \\ \texttt{tolerance } n \end{array}\right.$

where z is an algebraic expression, n is a positive integer, and each of the op_k ($1 \leq k \leq N$, $N \geq 1$) is an option designator, namely, one of the following:

ilines (or jlines)	causes lines connecting points of constant i (or j) to be omitted, except where needed to define surface or base-plane boundaries.
spikes	causes vertical lines normally drawn at boundary discontinuities from base plane to surface to be omitted
base	causes outline of domain normally shown in base plane, plus spikes noted above, to be omitted
axes	causes indicator showing x-, y-, and z-directions (normally drawn at bottom left of screen) to be omitted
data	causes numerical data indicating maximum and minimum values of x, y, and z (normally plotted at bottom right of screen) to be omitted
difxyz	causes same scale factor to be used for z-direction as is used for x- and y-directions (normally z-coordinates are scaled independently, since their ranges may differ greatly from x,y-ranges)
zscale (or xyscal)	causes new display to be constructed using same z- (or x,y-) scale factor as was used in last display
xtrans (or ytrans) (or ztrans)	causes new display to be constructed using same x (or y) (or z) translation constants as was used in last display
slave	causes displays to be plotted only on user's terminal, leaving second terminal free for another user

EXAMPLES:
```
plot u(2,i,j)
plot atan(uy(1,i,j)/ux(1,i,j)) sans slave
plot again sans ilines, base
plot off
plot tolerance 8
```

NORMAL OPERATION: If "plot *z* [sans ...]" is used, a command sequence of the form

```
tally
define bcd '(z)' = z
```

is simulated and a data base established by evaluating "(z)" and lattice parameters "x" and "y" at all nodes. This data base is used to construct a (new) surface display satisfying the option list supplied by the phrase "sans op_1, op_2, ..., op_N", if the latter is included. Subsequent displays of the same surface which satisfy differenct option lists can be generated from the same data base (hence at little additional computation expense) by using commands of the form "plot again [sans ...]". Other commands have the following effects:

(1) The command "plot nil" blanks the display tube and resets rotation angles, but leaves the user "signed on" to the display system.

(2) The command "plot off" has the same effects as "plot nil", but signs the user off.

(3) The command "plot discard" causes the data base created as a result of the last "plot *z*"-command to be discarded, thereby freeing memory for other purposes.

(4) The command "plot movie *n*" causes the synchronized movie camera to expose *n* frames of film, both immediately and on each subsequent use of push-button 4 (see Thornhill, et al [3], p. 2.4).

(5) The command "plot tolerance *n*" causes a tolerance parameter used in constructing displays to be set to *n*. In general, increasing this parameter reduces picture quality, but increases the number of grid lines which can be displayed. For complicated surfaces based on lattices of many nodes, it is sometimes necessary to increase the tolerance parameter beyond its normal value of four in order to obtain a complete display.

NON-FATAL ERRORS: Illegal conditions recognized by the tally-command, memory overflow, references to undefined quantities, and various illegal conditions recognized by the display system.

FATAL ERRORS: Non-correctable syntax errors and commands of the form "plot again [sans ...]" when no data base is active.

print-command

PURPOSE: To produce printed output.

REFERENCE: Section 7.1.

FORMAT: print e_1, e_2, ..., e_N [using bcd $\lambda f \lambda$]

where the e_k are algebraic expressions ($1 \leq k \leq N$, $N \geq 1$), f is a MAD-language format specifier, and λ is any single character that does not appear in f.

EXAMPLES:
```
print x, y, ux(1,-3,5), uy(2,7,9)
print atan(y/x), 8*sxx-3.67 using bcd /2f9.4*/
```

NORMAL OPERATION: The e_k are evaluated and printed as indicated below.

NORMAL RESPONSE: If the phrase "using bcd $\lambda f \lambda$" is included, values are printed in accordance with format specifier f. Otherwise, they are printed in accordance with the format specifier "5(1pe16.7)*". See Section 7.1 for format restrictions and examples.

NON-FATAL ERRORS: Memory overflow and references to undefined parameters.

FATAL ERRORS: Non-correctable syntax errors, including illegal format specifiers.

quit-command

PURPOSE: To return control to the CTSS supervisor.

REFERENCES: Section 2, 11, and 13.2.

FORMAT: quit

NORMAL OPERATION: Any disk file open as a result of a previous read- or record-command is closed and control is returned to CTSS via the A-core entry DORMNT.† If control is passed back to EPS, e.g., by means of a CTSS start-command, processing of any input remaining in the command sequence of which the quit-command was a part is continued in the normal manner. Thus by typing "quit α$", then a CTSS save-command, it is possible to create a "saved" version of EPS which will process the input sequence α immediately on being resumed.

NORMAL RESPONSE: For any disk file that is closed, a message of the form

FILE $NAME_1$ $NAME_2$ HAS BEEN CLOSED.

Then the usual CTSS time-usage message.

†See [2], sec. AG.6.01.

read-command

PURPOSE: To initiate the reading of an input file or to change an input file's status from active to inactive or vice versa.

REFERENCE: Section 11.1.

FORMAT: read $\begin{Bmatrix} name_1\ name_2 \\ \text{switch} \end{Bmatrix}$

where $name_1$ and $name_2$ are identifiers, each with six or fewer characters.

EXAMPLES: read alpha beta
read switch

NORMAL OPERATION: If the first form, i.e., "read $name_1$ $name_2$" is used, any input file already open as a result of a previous read-command is closed, then $name_1$ $name_2$ is opened and placed in active status. As a result, after the processing of the command sequence containing "read $name_1$ $name_2$" is completed, subsequent input is taken from $name_1$ $name_2$. The reading of this file continues until

(1) another command of the form "read $name_1$ $name_2$" is encountered, in which case the procedure described above is repeated;

(2) a quit-command is encountered, in which case the file is closed and control is returned to CTSS;

(3) an end-of-file is encountered, in which case the file is closed and the user is requested to supply further input from the console;

(4) a "read switch" is encountered, in which case the file is placed in inactive status and, after processing of the command sequence containing "read switch" is completed, the user is requested to supply further input from the console; or

(5) an error is detected, in which case the file is placed in inactive status and the user is immediately requested to supply further input from the console.

If the reading of the file is interrupted because of condition (4) or (5), a "read switch" supplied from the console at any later time will

cause reading to be continued from where it left off.

NORMAL RESPONSE: For the command "read $name_1$ $name_2$" the line(s)

```
[FILE NAME'1 NAME'2 HAS BEEN CLOSED.]
 FILE NAME1 NAME2 HAS BEEN OPENED.
```

where the line in brackets is printed only if some input file $name'_1$ $name'_2$ was already open. For the command "read switch", either

```
READ FILE HAS BEEN DEACTIVATED.
```

or

```
READ FILE HAS BEEN REACTIVATED.
```

whichever is appropriate.

NON-FATAL ERROR: Memory overflow.

FATAL ERRORS: Invalid file names (including file names not in the user's directory), input files with improper format (see Section 11.1), and "read switch" commands when no input file is open.

record-command

PURPOSE: To initiate the copying of console I/O into an output file or to change such a file's status from active to inactive or vice versa.

REFERENCE: Section 11.2.

FORMAT: record $\begin{Bmatrix} name_1\ name_2 \\ \text{switch} \end{Bmatrix}$

where $name_1$ and $name_2$ are identifiers, each with six or fewer characters.

EXAMPLES: record gamma delta
record switch

NORMAL OPERATION: If the command "record $name_1$ $name_2$" is used, any output file already open as a result of a previous record-command is closed, then $name_1$ $name_2$ is opened and placed in active status. As a result, subsequent user input and program output are copied into the file $name_1$ $name_2$, generating a transcript of all console I/O. (N.B., if $name_1$ $name_2$ refers to an existing file, information is appended to that file, not written over information already there.) The writing of a transcript continues until a quit-command or a "record switch" is given, the latter serving, in general, either to make an active output file inactive or to make an inactive file active.

NORMAL RESPONSE: For the command "record $name_1$ $name_2$", the line(s)

[FILE $NAME'_1$ $NAME'_2$ HAS BEEN CLOSED.]
FILE $NAME_1$ $NAME_2$ HAS BEEN OPENED.

where the line in brackets is printed only if an output file $name'_1$ $name'_2$ was already open. For the command "record switch", either

RECORD FILE HAS BEEN DEACTIVATED.

or

RECORD FILE HAS BEEN REACTIVATED.

whichever is appropriate.

NON-FATAL ERROR: Memory overflow.

FATAL ERRORS: Invalid file names, track quota overflow, and "record switch" requests when no output file is open.

relax-command

PURPOSE: To initiate or continue the successive relaxation of finite-difference equations.

REFERENCES: Section 6, especially 6.2, and Section 8.

FORMAT: relax

NORMAL OPERATION: If no difference equations exist, a form-command is simulated. Otherwise, it is assumed that the existing equations are to be maintained. Current values for relaxation factor "omega", tolerance parameter "delta", and maximum iteration index "limit" are used to update various internal parameters. Then relaxation is begun, using the current solution matrix as a starting point. During each iterative pass, corrections to the various components of the solution matrix are introduced in the same order as that in which difference equations are generated by the form-command. Thus all "n" dependent variables are treated at any given node before another node is considered, and, in terms of the lattice configuration in i,j-space, nodes are processed from left-to-right, bottom-to-top. Iterations are continued until one of the two termination conditions associated with parameters "delta" and "limit" is satisfied, unless an arithmetic overflow is detected (generally a result of numerical instability) or the user gives a single interrupt signal (see Section 13.1).

NORMAL RESPONSE: A line of the form

RELAXATION TERMINATED AFTER N PASSES. MAX SOLN CHANGE C.

where N is the number of iterative passes completed and C is the maximum absolute change introduced in any component of the solution matrix during the N^{th} pass.

NON-FATAL ERRORS: Illegal conditions detected as a result of simulated form-commands and arithmetic overflows.

remove-command

PURPOSE: To delete macro definitions.

REFERENCE: Section 9, especially 9.6.

FORMAT: remove begin $macroname_1$, $macroname_2$, ..., $macroname_N$ end

where the $macroname_k$ are identifiers ($1 \leq k \leq N$, $N \geq 1$).

EXAMPLES: remove begin prescribe end
remove begin pr, dump end

NORMAL OPERATION: Each $macroname_k$ found actually to name a macro causes the deletion of that macro.

NORMAL RESPONSE: For each macro actually deleted, a message of the form

MACRO '*MACRONAME*' NOW INACTIVE.

retry-command

PURPOSE: To reinitiate processing of a command sequence saved as a result of a non-fatal error.

REFERENCE: Section 14.

FORMAT: retry

NORMAL OPERATION: The most recently saved command sequence is taken from the command stack and retried.

FATAL ERRORS: retry-commands given when no saved command sequences exists to be retried.

review-command

PURPOSE: To determine current contour, parameter, or macro definitions.

REFERENCE: Section 12.2.

FORMAT: review $\left\{\begin{matrix}\text{curve}\\\text{variable}\\\text{macro}\end{matrix}\right\}$ begin d_1, d_2, ..., d_N end

where: (1) if the command "review curve ..." is used, each d_k is of the form

curvename [along i_1,j_1, i_2,j_2, i_3,j_3]

where *curvename* is an identifier and the i_ℓ,j_ℓ are algebraic expressions ($1 \le \ell \le 3$); (2) if the command "review variable ..." is used, each d_k is of the form

paramname[(s_1, s_2, ..., s_M)]

where *paramname* is an identifier and the s_ℓ are algebraic expressions ($1 \le \ell \le M$, $M \ge 1$); or (3) if the command "review macro ..." is used, each d_k is of the form

macroname

where *macroname* is an identifier ($1 \le k \le N$, $N \ge 1$).

EXAMPLES:
```
review curve begin zeta along 0,0, imax,0, imax,jmax end
review variable begin x, y, a(2,2), h end
review macro begin pr, dump end
```

NORMAL OPERATION AND RESPONSE: In all cases, the d_k are processed from left to right.

If the command "review curve ..." is used, lists of grid coordinates and segment numbers are printed for each d_k. When a d_k has the simple form *curvename*, all nodes of contour *curvename* are listed. When a d_k has the qualified form "*curvename* along i_1,j_1, i_2,j_2, i_3,j_3", only the nodes of *curvename* lying on the arc running from i_1,j_1 through i_2,j_2 to i_3,j_3 are listed.

If the command "review variable ..." is used, a definition, value, or subscript range is printed for each d_k. For d_k which refer to unsubscripted quantities or specific components of subscripted quantities, actual definitions (for quantities specified by define-commands) or values (for quantities specified by set-commands) are printed. For d_k which refer to subscripted quantities, but do not include a full set of subscripts, current subscript ranges are printed.

If the command "review macro ..." is used, the prototype call and corresponding definition for each d_k, i.e., each *macroname*, is printed.

See Section 12.2 for examples.

NON-FATAL ERRORS: References to undefined quantities among the grid coordinates i_ℓ,j_ℓ of a "review curve ..." command or among the subscripts s_ℓ of a "review variable ...." command.

FATAL ERRORS: Non-correctable syntax errors.

rotate-command

PURPOSE: To cause surface displays to be pre-rotated by discrete amounts before being transmitted to the display system.†

REFERENCE: Section 7.2.

FORMAT: rotate e_1 about a_1, e_2 about a_2, ..., e_N about a_N

where the e_k are algebraic expressions and each a_k is an axis specifier, namely, one of the symbols "x", "y", or "z" ($1 \leq k \leq N$, $N \geq 1$).

EXAMPLES:
```
rotate 180 about x
rotate phi about z, 2*phi about y
```

NORMAL OPERATION: The phrases "e_k about a_k" are processed from left to right. Each e_k is evaluated and interpreted as an angle (in degrees) about the a_k-axis through which surface displays are to be pre-rotated. The rotation matrix associated with this operation is reset to the identity matrix when a command such as "rotate 0 about x" is received.

NORMAL RESPONSE: The current display, if any, is replotted with the specified pre-rotations. Subsequent plot-commands produce new displays with the same pre-rotations until another rotate-command is received.

NON-FATAL ERRORS: References among the e_k to undefined quantities and illegal conditions recognized by the display system.

FATAL ERRORS: Non-correctable syntax errors.

---

†Real-time rotations (controlled by the plexiglass globe) are, in effect, added to the pre-rotations introduced by the rotate-command. This command is not needed often, but in certain applications is useful for obtaining specific views. For example, by typing "plot nil rotate 90 about x plot u(2,i,j)", the user will obtain immediately the view of surface "u(2,i,j)" seen by looking down the y-axis. Picture quality is generally superior when rotated views are obtained in this manner, for round-off errors introduced by the real-time hardware are avoided.

set-command

PURPOSE: To specify algebraic parameters that are to be treated as constants.

REFERENCES: Sections 3.5, 4.2, 4.3, 5.3, 5.4, 6.2, 8.2, 10.2, and 10.3.

FORMAT: set $p_{11} = p_{12} = \ldots = p_{1M_1} = e_1$, $p_{21} = p_{22} = \ldots = p_{2M_2} = e_2$, $\ldots$, $p_{N1} = p_{N2} = \ldots = p_{NM_N} = e_N$

where the $p_{k\ell}$ are parameter designators and the e_k are algebraic expressions ($1 \le k \le N$, $N \ge 1$; $1 \le \ell \le M_k$, $M_k \ge 1$).

EXAMPLE: `set maxphi=pi=4*atan(1), z(imax)=pi/3`

NORMAL OPERATION: Argument groups containing more than one "=" are rewritten as shown in the description of the define-command, producing a revised command of the form

set $p_1 = e_1$, $p_2 = e_2$, $\ldots$, $p_L = e_L$

For each assignment $p_k = e_k$ of the revised command, the value of expression e_k is computed and the data base arranged so that in the future this value is fetched each time the value of p_k is required. Subscripted parameters are, again, treated as with the define-command. Thus if the command of the example is given when "imax" is 32, the quantity "z(32)" is assigned the value "pi/3". Note, since processing is from left to right, that "pi" is known when the evaluation of "pi/3" is initiated.

SIDE EFFECTS: Previous definitions are deleted and subscript ranges are modified as needed.

NON-FATAL ERRORS: Memory overflow and references to undefined quantities.

FATAL ERRORS: Non-correctable syntax errors.

stop-command

PURPOSE: Inserted before an if-phrase, to insure that any preceeding command is executed before the if-phrase is interpreted.

REFERENCES: Sections 9.3, 10.2, and 10.3.

FORMAT: stop

NORMAL OPERATION: No action--control is returned to the EPS supervisor immediately.

tally-command[†]

PURPOSE: To close any bounding arcs not yet closed and to ready the portion of the data base which represents lattice structure in i,j-space for the generation of finite-difference equations or for the construction of graphical displays.

REFERENCE: Section 3.

FORMAT: tally

NORMAL OPERATION: If a tally-command has been executed previously, either as a result of explicit user input or as a result of an earlier form- or plot-command, and if no append- or delete-command has been executed since that time, no action is taken. Otherwise, the following operations are performed. First, for each bounding arc *curvename* not yet closed, the command

close *curvename*

is simulated. Then the node list for each contour is reordered, if necessary, so that the nodes are listed in clockwise (counterclockwise) order if the contour represents an external (internal) boundary. Finally, a check is made to insure that, in i,j-space, no contour crosses over itself or over any other contour. At the same time, certain bookkeeping operations, including the determination of the total number of nodes in the lattice, are performed.

NORMAL RESPONSE: None if no action is taken. Otherwise, a line report-

---

[†]Explicit use of the tally-command is unnecessary if input procedures described elsewhere are observed, for the command is simulated automatically whenever the operations it performs become essential to further processing, i.e., whenever a form- or plot-command is executed. Note, however, that it is possible to specify lattice structure in i,j-space by typing the relevant append-command(s), then, in lieu of the usual close-command(s), a single tally-command. This not only may save a small amount of typing, but also indicates immediately whether or not a lattice has acceptable topology.

ing the number of nodes *N* in the lattice as follows:

POINT TALLY IS *N*.

NON-FATAL ERRORS: Use of the command when no bounding contours are defined, when close-commands are needed but cannot be successfully executed, and when contours exist which cross over themselves or other contours.

16. Summary

To orient the beginning user and to provide the person who has used EPS in the past with a brief review, important input forms and notational conventions are outlined below. Items (3) through (9) constitute a check list which can be followed explicitly when solving linear boundary-value problems with no complicating factors such as free surfaces or undefined parameters.

(1) To establish a link to EPS, type

link *name*$_1$ saved m1416 cmf104

where *name*$_1$ is "eps-p" if the ESL display is to be used for graphical output, or simply "eps" if otherwise. See Section 1.

(2) To begin a run, type

resume *name*$_1$

and wait for the message "PROCEED:". In supplying subsequent input, remember that it is necessary to signal the completion of each unit of input explicitly by typing a "$" before giving a final carriage return. See Section 2.

(3) To define lattice structure in i,j-space, type a sequence of the form

append i_1,j_1, i_2,j_2, ..., i_N,j_N to *curvename*
close *curvename*

for each bounding contour *curvename*, where the i_k,j_k give the contour's vertices in the i,j-plane in either clockwise or counterclockwise order. See Section 3, especially 3.3 and 3.4.

(4) To specify a mapping algorithm, type

define x=f, y=g

where f and g either explicitly or implicitly (by referencing auxiliary parameters) indicate how nodal values "x" and "y" for the independent variables depend on grid coordinates "i" and "j". If auxiliary parameters are introduced, specify functional dependencies by means of additional define-commands, constants by means of appropriate set-commands. See Section 3, especially 3.5.

(5) To specify governing partial differential equations, type set- and/or define-commands designating parameters n, a(k,l), b(k,l), ..., g(k,l), h(k) corresponding to n, $a_{k\ell}$, $b_{k\ell}$, ..., $g_{k\ell}$, h_k in

$$\sum_{\ell=1}^{n} \left(\frac{\partial}{\partial x} \left(a_{k\ell} \frac{\partial u_\ell}{\partial x} + b_{k\ell} \frac{\partial u_\ell}{\partial y} + c_{k\ell} u_\ell \right) + \frac{\partial}{\partial y} \left(d_{k\ell} \frac{\partial u_\ell}{\partial x} + e_{k\ell} \frac{\partial u_\ell}{\partial y} + f_{k\ell} u_\ell \right) + g_{k\ell} u_\ell \right) = h_k \qquad [k = 1,2,\ldots,n] \tag{1}$$

Be certain that the fit-function is applied as shown in Section 4.2 when equation parameters vary spatially in a continuous manner.

(6) To number boundary segments corresponding to different boundary conditions, type commands of the form

impose m along i_1,j_1, i_2,j_2, i_3,j_3

where m is a segment number to be associated with the arc running from node i_1,j_1 through i_2,j_2 to i_3,j_3. See Section 5.2.

(7) To specify boundary conditions for a given segment m, type set- and/or define-commands for parameters p(k,l,m), q(k,l,m), and r(k,m) corresponding to $p_{k\ell m}$, $q_{k\ell m}$, and r_{km} in

$$\sum_{\ell=1}^{n} (p_{k\ell m} F_\ell + q_{k\ell m} u_\ell) = r_{km} \qquad [k = 1,2,\ldots,n] \tag{9}$$

where the F_ℓ are first-order functions of the dependent variables u_ℓ given in terms of the coefficients of governing system (1) as follows:

$$\begin{aligned} F_k = \sum_{\ell=1}^{n} \Big(& - \left(a_{k\ell} \frac{\partial u_\ell}{\partial x} + b_{k\ell} \frac{\partial u_\ell}{\partial y} + c_{k\ell} u_\ell\right) \frac{\partial y}{\partial \delta} \\ & + \left(d_{k\ell} \frac{\partial u_\ell}{\partial x} + e_{k\ell} \frac{\partial u_\ell}{\partial y} + f_{k\ell} u_\ell\right) \frac{\partial x}{\partial \delta} \end{aligned} \qquad [k = 1,2,\ldots,n] \tag{7}$$

Minor restrictions on the $p_{k\ell m}$ and other details are given in Section 5.3.

(8) To request the generation of finite-difference equations after data has been introduced as shown in items (3) through (7), or modified as indicated in item (12), type

form

(9) To obtain a solution for the finite-difference equations, type a set-command giving values for relaxation factor "omega", tolerance parameter "delta", and maximum iteration index "limit". Then initiate successive relaxation by typing

relax

If necessary, relaxation may be continued by means of additional relax-commands. For information regarding the selection of values for "omega", etc., see Section 6.2.

(10) To obtain printed output, type sequences such as

print e_1, e_2, ..., e_N

```
do(print e1, e2, ..., eN
   for i=i1 step δi until i2)

do(do(print e1, e2, ..., eN
   for i=i1 step δi until i2)
   for j=j1 step δj until j2)
```

where the e_k are algebraic expressions which may designate arbitrary algebraic combinations of the solution properties u(k,i,j), ux(k,i,j), etc., as defined below in Table 2. See Section 7.1.

(11) To obtain graphical output in the form of a surface display, type

```
plot z
```

where z is an algebraic expression. See Sections 7.2 and 15.2.

(12) To modify lattice structure in i,j-space, type delete-, append-, and close-commands as explained in Section 8.1. To modify parameter definitions, simply type more set- and/or define-commands. Similarly, to modify boundary segment numbers, type additional impose-commands.

(13) To solve problems with nonlinearities or other features demanding iterative use of the system, combine items (8), (9), and (12), as explained in Section 10.

(14) To avoid excessive typing, define appropriate macros by means of commands such as

```
let macroname(p1 s1 p2 ... sN-1 pN) mean
    begin
    d(p1;p2;...;pN)
    end
```

where the p_k are formal parameters, the s_k are separators, and

$d(p_1;p_2;\ldots;p_N)$ is a substitution to be made when calls of the indicated form are encountered. See Section 9.

(15) To use disk input or to cause console I/O to be copied into a disk file, type read- and record-commands, as explained in Section 11.

(16) To determine what identifiers are in use or to verify that definitions are correct, type list- and review-commands, as explained in Section 12.

(17) To interrupt the machine in order to prevent or curtail the processing of erroneous input, press the interrupt button once. To terminate a run, type

```
quit
```

or press the interrupt button twice. See Section 13.

Error messages resulting from improper input, etc., are generally self-explanatory, and in many cases indicate appropriate corrective action. However, if questions arise, consult Sections 14 and 15.

The tables which follow summarize most of the important elements of the EPS input language.

Table 1. Summary of basic command forms[†]

append i_1,j_1, i_2,j_2, ..., i_N,j_N [to *curvename*]

close [*curvename*]

define $p_{11} = p_{12} = \ldots = p_{1M_1} = e_1$, $p_{21} = p_{22} = \ldots = p_{2M_2} = e_2$, ..., $p_{N1} = p_{N2} = \ldots = p_{NM_N} = e_N$

expand begin p end

form

ignore

impose m { along i_1,j_1, [i_2,j_2,] i_3,j_3 / at i_1,j_1 } [of *curvename*]

let *macroname*[(p_1 s_1 p_2 ... s_{N-1} p_N [s_N])] mean
begin
$d(p_1;p_2;\ldots;p_N)$
end

list e_1, e_2, ..., e_N

plot { { z / again } [sans op_1, op_2, ..., op_N] / nil / off / discard / movie n / tolerance n }

print e_1, e_2, ..., e_N [using bcd $\lambda f\lambda$]

[†]See sec. 15 for detailed descriptions.

Table 1 (continued)

quit

read $\begin{Bmatrix} name_1\ name_2 \\ \text{switch} \end{Bmatrix}$

record $\begin{Bmatrix} name_1\ name_2 \\ \text{switch} \end{Bmatrix}$

relax

remove begin $macroname_1$, $macroname_2$, ..., $macroname_N$ end

retry

review $\begin{Bmatrix} \text{curve} \\ \text{variable} \\ \text{macro} \end{Bmatrix}$ begin d_1, d_2, ..., d_N end

rotate e_1 about a_1, e_2 about a_2, ..., e_N about a_N

set $p_{11} = p_{12} = \ldots = p_{1M_1} = e_1$, $p_{21} = p_{22} = \ldots = p_{2M_2} = e_2$, ..., $p_{N1} = p_{N2} = \ldots = p_{NM_N} = e_N$

stop

tally

Table 2. Summary of built-in functions

Call	Value	Restrictions
xs	local boundary slope $\partial x/\partial s$	use only in definitions of boundary condition parameters p(k,l,m), q(k,l,m) and r(k,m) (see Section 5.3)
ys	local boundary slope $\partial y/\partial s$	
octant	lattice sector number	use only in definitions of boundary condition parameters and governing equation parameters a(k,l), b(k,l), etc. (see Section 4.2)
fit(z)	interpolated or internodal value of z needed by difference equation generator when z is to be treated as a continuous variable	none--produces the nodal value of z for the current values of i and j when difference equations are not being generated (see Section 4.2)
u(k,i,j)	current approximation for u_k at node (i,j)	none (see Section 10 for proper use when solving nonlinear equations)
ux(k,i,j)	current approximation for $\partial u_k/\partial x$ at node (i,j)	use only in conjunction with requests for output or with commands which cause problem data to be revised; do not use directly in definitions of boundary condition or governing equation parameters (see Sections 7 and 10)
uy(k,i,j)	current approximation for $\partial u_k/\partial y$ at node (i,j)	
uxx(k,i,j)	current approximation for $\partial^2 u_k/\partial x^2$ at node (i,j)	
uxy(k,i,j)	current approximation for $\partial^2 u_k/\partial x \partial y$ at node (i,j)	
uyx(k,i,j)	current approximation for $\partial^2 u_k/\partial y \partial x$ at node (i,j)	
uyy(k,i,j)	current approximation for $\partial^2 u_k/\partial y^2$ at node (i,j)	
flux(k,i,j)	current approximation for F_k at node (i,j)	

Table 3. Summary of operators[†]

Operator	Type	Precedence	Usage	Interpretation
sin	unary	10	sin *a*	sine of *a*
cos			cos *a*	cosine of *a*
tan			tan *a*	tangent of *a*
				a in radians
atan			atan *a*	arc tangent of *a*, result in radians
log			log *a*	natural logarithm of *a* ($a > 0$)
exp			exp *a*	natural exponential of *a*, i.e., e^a
sqrt			sqrt *a*	square root of *a* ($a \geq 0$)
abs			abs *a*	absolute value of *a*
+			+*a*	plus *a*
-			-*a*	minus *a*
sign	diadic	9	*a* sign *b*	*a* with the sign of *b*
power	diadic	8	*a* power *b*	*a* to the power *b* (*a* ≥ 0 if *b* non-integral)
*	diadic	7	*a***b*	*a* multiplied by *b*
/			*a*/*b*	*a* divided by *b*
+	diadic	6	*a*+*b*	*a* plus *b*
-			*a*-*b*	*a* minus *b*
grt	diadic	5	*a* grt *b*	1 if $a > b$, 0 otherwise
geq			*a* geq *b*	1 if $a \geq b$, 0 otherwise
les			*a* les *b*	1 if $a < b$, 0 otherwise
leq			*a* leq *b*	1 if $a \leq b$, 0 otherwise
eql			*a* eql *b*	1 if $a = b$, 0 otherwise
neq			*a* neq *b*	1 if $a \neq b$, 0 otherwise

[†]In expressions involving multiple operations, operators of greater precedence are executed first unless parentheses indicate otherwise. Adjacent diadic operations of equal precedence are executed from left to right. Thus *a*/*b***c* is equivalent to (*a*/*b*)**c*. Sequences of unary opera-

Table 3 (continued)

Operator	Type	Precedence	Usage	Interpretation
not	unary	4	not *a*	1 if *a* = 0, 0 otherwise
and	diadic	3	*a* and *b*	1 if *a* ≠ 0 and *b* ≠ 0, 0 otherwise
or	diadic	2	*a* or *b*	1 if either *a* ≠ 0 or *b* ≠ 0, or both, 0 otherwise
exor			*a* exor *b*	1 if either *a* ≠ 0 or *b* ≠ 0, but not both, 0 otherwise
eqv	diadic	1	*a* eqv *b*	1 if *a* = 0 and *b* = 0 or if *a* ≠ 0 and *b* ≠ 0, 0 otherwise

---

tors are, of course, executed from right to left, i.e., "exp sqrt abs *a*" can only mean "exp(sqrt(abs *a*))".

Note, incidentally, that "exp", "sqrt", etc., are literally operators in EPS, not built-in functions as in most compiler languages. Operands of such operators need not be enclosed in parentheses unless parentheses are needed to indicate order of operation. Thus "sine of *a*" can either be written "sin *a*", as in mathematics, or "sin(*a*)", as in other languages; on the other hand, "sine of the quantity *a*+*b*" must be written "sin(*a*+*b*)".

References

1. Tillman, C. C. "On-Line Solution of Elliptic Boundary-Value Problems." Ph.D. thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1969 (incomplete at present).

2. Crisman, P. A. (ed.). *The Compatible Time-Sharing System: A Programmer's Guide*. Second Edition (updated through 1966). The M.I.T. Press, Cambridge, Mass., 1965.

3. Thornhill, D. E., et al. "An Integrated Hardware-Software System for Computer Graphics in Time-Sharing." *M.I.T. Electronic Systems Laboratory Tech. Rept.* ESL-R-356 (also *M.I.T. Project MAC Tech. Rept.* MAC-TR-56), December, 1968.

4. Parmelee, R. P. "Three-Dimensional Stress Analysis for Computer-Aided Design." Ph.D. thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1966.

5. Varga, R. S. *Matrix Iterative Analysis*. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1962.

6. Crandall, S. H. *Engineering Analysis: A Survey of Numerical Procedures*. McGraw-Hill Book Company, Inc., New York, 1956.

7. Coons, S. A. "Surfaces for Computer-Aided Design of Space Forms." *M.I.T. Project MAC Tech. Rept.* MAC-TR-41, June, 1967.

8. Young, D. M. "The Numerical Solution of Elliptic and Parabolic Differential Equations," in *Survey of Numerical Analysis* (J. Todd, ed.). McGraw-Hill Book Company, Inc., New York, 1962.

9. Arden, B. W., et al. "The Michigan Algorithm Decoder (The MAD Manual)." University of Michigan, 1966.

10. Hildebrand, F. B. *Introduction to Numerical Analysis*. McGraw-Hill Book Company, Inc., New York, 1956.

11. Ames, W. F. *Nonlinear Partial Differential Equations in Engineering*. Academic Press, New York, 1965.

12. Shapiro, A. H. *The Dynamics and Thermodynamics of Compressible Fluid Flow*. Vol. I. Ronald Press, New York, 1953.

13. Schlichting, H. *Boundary Layer Theory* (J. Kestin, tr.). McGraw-Hill Book Company, Inc., New York, 1960.

14. Polubarinova-Kochina, P. Ya. *Theory of Ground Water Movement* (J. DeWiest, tr.). Princeton University Press, 1962.

15. Lo, R. C.-Y. "Steady Seepage with Free Surface." Ph.D. thesis, Division of Engineering and Applied Physics, Harvard University, 1969.

This empty page was substituted for a blank page in the original document.

CS-TR Scanning Project
Document Control Form

Date : 2/2/96

Report # LCS-TR-62

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 194 (201-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP:	(1-194) UNN'ED TITLE & BLANK PAGES, III-IX, UNN'ED BLANK, XI-XIII, UNN'ED BLANK, 1-179, UNN'ED BLANK,
(195-201) SCANCONTROL, COVER, FUNDING AGENT, DOD, TRGT'S (7)	

Scanning Agent Signoff:
Date Received: 2/2/96 Date Scanned: 2/13/96 Date Returned: 2/15/96

Scanning Agent Signature: Michael W. Cook

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Massachusetts Institute of Technology
Project MAC and Electronic Systems Laboratory

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE

EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation--User's Guide

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)* Appendix for forthcoming Ph.D. thesis "On-Line Solution of Elliptic Boundary-Value Problems," Department of Mechanical Engineering

5. AUTHOR(S) *(Last name, first name, initial)*

Tillman, Coyt C., Jr.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June, 1969	194	15

8. CONTRACT OR GRANT NO.

Office of Naval Research Contract Nonr-4102(01), Project Nos. NR-048-189 and RR 003-09-01, and Air Force Contracts AF-33(657)-10954 and F33615-67-C-1530

9a. ORIGINATOR'S REPORT NUMBER(S)
MAC-TR-62
ESL-R-395

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

None

12. SPONSORING MILITARY ACTIVITY

Advanced Research Projects Agency
3D-200 Pentagon
Washington, D.C. 20301

Air Force Manufacturing Technology Laboratory, RTD
Wright-Patterson AFB

13. ABSTRACT A user's guide for EPS is presented. EPS solves two-dimensional boundary-value problems for elliptic systems of second-order partial differential equations. It also has general-purpose capabilities which permit the on-line definition and execution of arbitrary numerical procedures.

The guide is primarily concerned with using EPS to solve boundary-value problems. Linear problems of this type that have no free surfaces or undefined parameters can be solved on a one-pass basis. Nonlinearities and other complications can be handled by iteration. A finite-difference method is employed which permits the use of irregular lattices, hence the crowding of nodes in sensitive regions.

EPS operates on the IBM 7094 computer of the M.I.T. Compatible Time-Sharing System (CTSS), and exploits to an unusual degree the potential for interactive problem solving that CTSS affords. Input commands resemble statements in various algebraic compiler languages, and can be combined and abbreviated by means of macros. Improper input and other error conditions are handled so as to minimize user inconvenience. Common syntax errors, for example, are corrected automatically by the machine. Output is available in either numerical or graphical form.

14. KEY WORDS

Computer-aided design
Computer graphics
Elliptic boundary-value problems
Elliptic partial differential equations
Finite-difference analysis
Interactive problem solving
Multiple-access computers
Numerical analysis
On-line computers
Time-shared computers

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED

Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 2/18/1996

M.I.T. Libraries
Document Services