Computer Program Descriptions

Scalar Finite-Element Program Package for Two-Dimensional Field Problems

PURPOSE: This program produces assembled finite-element matrices for solving scalar two-dimensional Laplace's, Poisson's, or Helmholtz's equations.

LANGUAGE: Fortran IV, *G* level. Source deck length, including comments and documentation, approximately 4750 cards.

AUTHORS: A. Konrad and P. Silvester, Department of Electrical Engineering, McGill University, Montreal 110, P. Q., Canada.

AVAILABILITY: ASIS-NAPS Document No. NAPS-01604. Copies of the source decks may be obtained from the second author, on 9-track IBM magnetic tape or in card form, within two years of publication. Prepayment of U. S. $60 for tape or U. S. $100 for cards is requested.

Manuscript received October 14, 1970, revised July 21, 1971.
For program listing, order document NAPS-01604 from ASIS National Auxiliary Publications Service, c/o CCM Information Corporation, 909 Third Avenue, New York, N. Y. 10022; remitting $4.00 per microfiche or $12.70 per photocopy.

DESCRIPTION: The entire program package contains two basic subroutines, READIN and ASSEMB, which generate finite-element matrices; plus a host of matrix manipulation and data handling subroutines, all called by a common main program. The two central subroutines are furnished in two versions: one providing elements up to fourth order, the other up to sixth order. The fourth-order element program is, in principle, similar to an earlier waveguide program [1], [2] and uses the same elements [3], but differs in the flexibility allowed in input and output arrangement. The enlarged version uses new element matrices generated by means of a Formac program similar in principle to that described in [3].

The listing is quite extensively commented so that users can readily adapt the package to special needs if so desired. Indeed, some users may prefer to employ locally available routines for all functions except those performed by READIN and ASSEMB; others may wish to stay with the security of an already debugged package. The latter has been written to accommodate a very large variety of problems, and will probably satisfy many users without any alteration.

READIN, the first subroutine called by the main program, serves two functions: 1) it reads data cards, prints out all input data, extends the input data set, and generates logical codes for guiding the

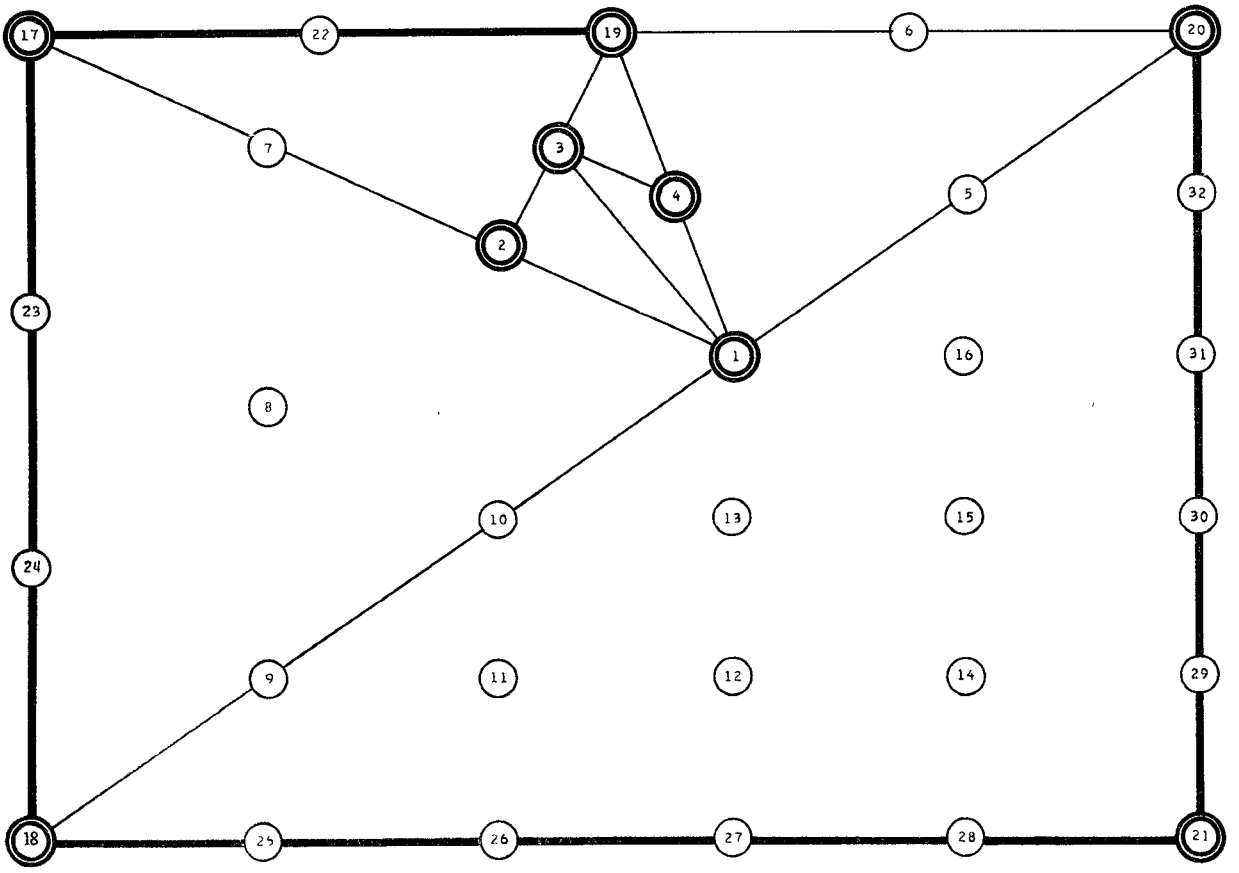

Fig. 1. Map of approximate point disposition in the elements specified by the input data. The heavily circled points are those appearing in the input point list (see Fig. 2), but renumbered; the remainder have been supplied by subroutine READIN. The light lines denote element boundaries. Note that various element orders have been used.

**** ----------SAMPLE PROBLEM---------- ****

SCALE = 1.00 HORIZONTAL

SCALE = 1.00 VERTICAL

*** INPUT POINT LIST ***

POINT NO.	HORIZONTAL COORDINATE	VERTICAL COORDINATE
1.	-16.00000	0.0
2.	-16.00000	-16.00000
3.	3.20000	-6.40000
4.	-3.20000	-4.26666
5.	-1.60000	-2.13333
6.	0.0	0.0
7.	16.00000	0.0
8.	16.00000	-16.00000
9.	1.60000	-3.20000

*** INPUT ELEMENT LIST ***

TRIANGLE NO.	ORDER N	VERTICES: A	B	C	CONSTRAINTS A_B	B_C	A_C	MATERIAL PROPERTY (PERMITTIVITY)
1.	1	3	4	5	0	0	0	1.00000
2.	1	3	5	9	0	0	0	1.00000
3.	1	5	6	9	0	0	0	1.00000
4.	2	3	6	7	0	0	0	1.00000
5.	2	1	6	4	1	0	0	1.00000
6.	3	1	2	3	1	0	0	1.00000
7.	5	2	8	7	1	1	0	1.00000

*** ASSEMBLED POINT LIST ***

POINT NO.	HORIZONTAL COORDINATE	VERTICAL COORDINATE	POINT NO.	HORIZONTAL COORDINATE	VERTICAL COORDINATE	POINT NO.	HORIZONTAL COORDINATE	VERTICAL COORDINATE
1.	3.20000	-6.40000	12.	3.20000	-12.80000	23.	-16.00000	-5.33333
2.	-3.20000	-4.26666	13.	3.20000	-9.60000	24.	-16.00000	-10.66667
3.	-1.60000	-2.13333	14.	9.60000	-12.80000	25.	-9.60000	-16.00000
4.	1.60000	-3.20000	15.	9.60000	-9.60000	26.	-3.20000	-16.00000
5.	9.60000	-3.20000	16.	9.60000	-6.40000	27.	3.20000	-16.00000
6.	8.00000	0.0	17.	-16.00000	0.0	28.	9.60000	-16.00000
7.	-9.60000	-2.13333	18.	-16.00000	-16.00000	29.	16.00000	-12.80000
8.	-9.60000	-7.46666	19.	0.0	0.0	30.	16.00000	-9.60000
9.	-9.60000	-12.80000	20.	16.00000	0.0	31.	16.00000	-6.40000
10.	-3.20000	-9.60000	21.	16.00000	-16.00000	32.	16.00000	-3.20000
11.	-3.20000	-12.80000	22.	-8.00000	0.0			

**** ----------SAMPLE PROBLEM---------- ****

****
* *
* EIGENVALUE = 0.187075E 00 *
* *
****

*** LIST OF POINTS WITH FREE POTENTIALS ***

POINT NO.	FREE POTENTIAL	POINT NO.	FREE POTENTIAL	POINT NO.	FREE POTENTIAL
1.	0.347866E 00	7.	0.339793E-01	13.	0.280359E 00
2.	0.165919E 00	8.	0.891267E-01	14.	0.120003E 00
3.	0.110519E 00	9.	0.466054E-01	15.	0.221251E 00
4.	0.240717E 00	10.	0.180403E 00	16.	0.290905E 00
5.	0.324126E 00	11.	0.109832E 00		
6.	0.365176E 00	12.	0.157206E 00		

Fig. 2. Partial printout for the sample problem. From top to bottom: input point list, specifying the vertices; input element list, specifying the seven elements in Fig. 1 in terms of their vertices; full list of renumbered points appearing in Fig. 1; output giving the lowest TM mode of the guide shown. Other sections of the printout give problem statistics, full point lists for all elements, solution values for the constrained boundary nodes, etc.

finite-element assembly; and 2) it checks for data errors. The input data formats are arranged so as to reduce keypunching to an absolute minimum, thus simplifying program use as well as reducing likelihood of blunders; data not explicitly supplied are generated by READIN. For example, a sixth-order element involves 28 nodal points. Only the three triangle vertex locations, however, are supplied by the input, while the remaining twenty-five are generated by the program, and added to the input point list. Similarly, data which may repeat from element to element (e.g., material properties or order of element) may be omitted from punching, and will be inserted by READIN.

If a data error is found, READIN instructs the main program to call READIN again. In this way, long runs of data sets may be processed without undue concern about punching errors; identifiably mistaken data sets are simply read, but not processed. If no error exists, subroutine ASSEMB is called. Here matrices S and T are con-

structed from the element describing matrices (stored as block data) for each triangle and are assembled into one large S and T matrix, respectively.

Once the element matrices have been assembled, the problem is ready for solution, either using a Gaussian elimination routine, or else by means of an eigenvalue problem package, depending on the problem. Appropriate routines are included in the program. With the exception of READIN and MAP, all subroutines are input–output free, the final processing of results and their printing being handled by the main program. Subroutine MAP sketches on the line printer the approximate locations of the generated nodal points, so as to permit rapid freehand sketching and checking of output.

Example

A brief illustration of program use is provided by solving for the TM modes in a vaned (thin-ridged) waveguide similar to that in [2]. Fig. 1 shows half the cross section of such a guide, as drawn by the subroutine MAP from the input data. The input data proper for this problem consist of nine cards giving the coordinates of the nine triangle vertices (the points doubly circled in Fig. 1) and seven cards each of which defines one triangular finite element by stating its three vertex numbers, the degree of polynomial approximation to be used in that triangle, and boundary constraints (if any). A few other cards, which perform housekeeping functions (e.g., furnishing the title SAMPLE PROBLEM), are also included with the input. These cards are processed by subroutine READIN. For illustrative purposes, the present problem includes a variety of elements (one fifth-, one third-, two second-, and three first-order); for all elements except first order, additional points are generated (the singly circled points in Fig. 1), and all points are renumbered for processing convenience. All input data and all data generated are printed out, as is a set of problem statistics (array sizes actually used, error codes, etc.). The latter are invaluable, should a set of data be abandoned because of an error condition.

In the authors' opinion, the average user cannot be expected to have any acquaintance with the internal operation of the individual subprograms. Every effort has been made, therefore, to identify, and to communicate to the user, any reason for malfunctioning—whether arising from hardware considerations (such as excess matrix size) or blunders (e.g., specifying a triangle with two identical vertices).

Part of the input data, as reproduced by READIN, appear in Fig. 2, followed by the listing of all points (including both original input and the newly generated ones) as renumbered. In the input element list, it may be noted that boundary conditions are given for each triangle edge; the character 0 is interpreted as a continuity requirement between elements or as a homogeneous Neumann boundary condition at edges; the character 1 is interpreted as a homogeneous Dirichlet condition. (Other options are also provided.)

The matrix assembly and equation solution follow; these generate no output. Once solutions have been obtained, they are printed out as listings of eigenvalues and point values of the corresponding eigenfunctions. Part of the output, illustrative of the arrangement, appears in Fig. 2. It will be noted that values are listed only for unconstrained points. Where Dirichlet constraints are imposed, the values are the same for all modes, and are therefore only printed out once at the beginning of the output listing.

Timings

The assembly of element matrices is executed rapidly, so that the major portion of computing time (about 70–95 percent, depending on the problem) is expended on the matrix algebra of actual equation solving. Timing estimates, therefore, may be made by prospective users by adding a small overhead cost to the known timings for solving matrix equations. The matrix algebra routines included in this package use Gaussian elimination for simultaneous equation solving, and the Householder transformation method for the eigenvalue problem; however, other subprograms may be substituted without difficulty. The storage requirements are roughly 100 kbytes for the smaller (fourth-order) program, and nearly 200 kbytes for the larger one. These figures are of course dependent on the maximum matrix orders contemplated, and may be modified by the user.

The sample problem above required solving an eigenvalue problem of order 16; the execution phase (excluding compilation and input–output operations) cost 76¢ in central processing unit charges.

References

[1] P. Silvester, "A general high-order finite-element waveguide analysis program," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-17, Apr. 1969, pp. 204–210.
[2] ——, "High-order finite element waveguide analysis," *IEEE Trans. Microwave Theory Tech.* (Comp. Prog. Desc.), vol. MTT-17, Aug. 1969, p. 651.
[3] ——, "High-order polynomial triangular finite elements for potential problems," *Int. J. Eng. Sci.*, vol. 7, 1969, pp. 849–861.