PROJECTIVE SOLUTION OF DIFFERENTIAL EQUATIONS

BY

ZOLTAN CSENDES, B.S.E.(MICH.), M. ENG.
DEPARTMENT OF ELECTRICAL ENGINEERING.

McGILL UNIVERSITY
MONTREAL, QUEBEC.

OCTOBER, 1972.

PROJECTIVE SOLUTION OF DIFFERENTIAL EQUATIONS

BY

ZOLTAN CSENDES, B.S.E. (MICH.), M. ENG.

A thesis submitted to the Faculty of Graduate Studies
and Research in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Department of Electrical Engineering
McGill University
Montreal, Quebec.

October, 1972.

PROJECTIVE SOLUTION OF DIFFERENTIAL EQUATIONS

BY

ZOLTAN CSENDES, B.S.E. (MICH.), M. ENG.

ABSTRACT

A projective method is developed for the solution of differential equations which uses different finite dimensional spaces to approximate the domain and the range spaces of the differential operator. It is shown that rectangular matrices are equivalent to analytic operators in these spaces. Using Lagrangean interpolation polynomials, simple numerical matrix equivalents are evaluated for the two operators $\frac{d}{dx}$ and $f(x)$. From these two types of matrices and a third projection matrix, the discretization of any differential equation is performed by ordinary matrix addition and multiplication. The result of such discretization is a rectangular matrix equation that is solved using the generalized inverse of the matrix to obtain an approximate general solution of the differential equation. Two computer programs using this procedure are presented, one of which solves arbitrary linear ordinary differential equations and the other for the solution of arbitrary linear two-dimensional partial differential equations. The input to these programs is a new computer language for the representation of differential equations called DECL, defined in this thesis. Several applications of the programs to engineering problems are demonstrated. In addition, numerical experiments are performed using the ordinary differential equation program in conjunction with Newton's method to solve nonlinear ordinary differential equations.

ACKNOWLEDGEMENT

I am indebted to my thesis advisor, Dr. P. Silvester, for the deep interest and numerous invaluable comments he contributed to this work. His enthusiasm, devotion to scientific inquiry and professional and personal generosity have all been inspiring. It has been a pleasure to learn from him.

I would also like to take this opportunity to acknowledge with sincere appreciation the many discussions I have had with my fellow electrical engineering students over the past two years. Special mention is due Peter Benedek and Bela Konrad with whom I shared office quarters and with whom we daily discussed our problems. In addition, I wish to thank Dr. A. Gopinath for his kind advice and the sincere interest he took in my work.

The financial assistance of the National Research Council of Canada is gratefully acknowledged.

TABLE OF CONTENTS

		Page
ABSTRACT		i
ACKNOWLEDGEMENTS		ii
TABLE OF CONTENTS		iii
CHAPTER I	INTRODUCTION	1
CHAPTER II	ELEMENTARY FACTORS	11
2.1	Differentiation Matrices	12
2.2.	Simple Operators	21
2.3	Compound Operators	24
CHAPTER III	PROJECTIVE APPROXIMATION	32
3.1	Galerkin's Method	33
3.2	Introducing Basis Functions	37
3.3	Relation to the Energy Norm Approximation	43
3.4	Elementary Matrices Revisited	46
3.5	Multi-Dimensional Regions	49
CHAPTER IV	DISCRETIZATION OF ARBITRARY DIFFERENTIAL EQUATIONS	60
4.1	Coordinate-Independent Factors	62
4.2	Coordinate-Dependent Factors	69
4.3	Two-Dimensional Equations	77
CHAPTER V	SOLUTION TECHNIQUES	85
5.1	Generalized Matrix Inversion	86
5.2	General Solutions	93
5.3	Particular Solutions	98
5.4	Solution in Multiple Regions	108
5.5	Eigenvalue Problems	119
5.6	Nonlinear Equations	127
CHAPTER VI	AUTOMATIC SOLUTION OF DIFFERENTIAL EQUATIONS	132
6.1	Simplified Input and Output Procedures	134
6.2	Ordinary Differential Equations	155
6.3	Nonlinear Ordinary Differential Equations	176

Page

6.4 Partial Differential Equations 189

CHAPTER VII CONCLUSIONS 214

REFERENCES 218

CHAPTER I

INTRODUCTION

In recent years, considerable progress has been made in developing direct numerical methods for the solution of differential equations. To a large extent, this advance is the result of improvements in computer systems, enabling computationally lengthy algorithms to be handled with comparative ease. However, in addition to the procedural refinements which naturally accompany increased usage, there has also been a major change in the type of numerical methods used. A decade ago, the most common method of solving partial differential equations in engineering applications was by means of finite difference formulae; today, for many engineering problems, use of the finite element method is standard.

The effect of the new procedures has been to enlarge the role of projective methods in the numerical solution of differential equations. These methods, of which the Rayleigh-Ritz method, the Galerkin method and the least-squares method occupy a dominant role, are thoroughly treated in standard books by Mikhlin [1], Mikhlin and Smolitskiy [2] and Kantorovich and Krylov [3]. The basic idea in projective methods is to express the solution of a boundary value problem as a sum of known functions with undetermined coefficients. A matrix equation for the unknowns is then derived by requiring the approximate solution to satisfy certain minimality conditions. Provided that the functions and the approximation criteria

are chosen judiciously, highly accurate approximate solutions result.

As originally conceived, classical projective methods were designed to solve a single boundary value problem in a single geometric region. In recent times, however, it has become apparent that the geometric limitations of these methods can be eased by defining new sets of trial functions which are non-zero only over small, standard subregions. The equations for the solution of a boundary value problem in a complicated region can then be easily determined by evaluating the conditions on a projective solution in each of the subregions under specified continuity conditions. The technique of obtaining solutions by this procedure is commonly called the finite element method.

The use of the finite element method was first promoted as a distinct and generally applicable procedure for the solution of structural mechanics problems by Turner, Clough, Martin and Topp [4] in 1956. Its further application in structural mechanics and plate bending analysis was developed in the early 1960's by Clough [5], by Adini and Clough [6] and by Gallagher [7]. In the mid 1960's Argyris made extensive studies of aeronautical structures using the finite element method [8]. At the same time, Zienkiewicz and Cheung [9,10] developed finite element methods for problems occurring in civil engineering. To a large extent, the early popularity of the finite element method in mechanical and civil engineering is due to the considerable work of two groups,

one in aeronautical structures in which the work of Argyris predominates and the other in civil engineering, led by Zienkiewicz.

The application of the finite element method to other engineering disciples quickly followed. In the late 1960's, Oden [11,12] developed finite element methods for nonlinear elasticity and thermomechanical problems. In electrical engineering, the first use of the finite element method was the analysis of electromagnetic wave propagation in homogeneous waveguides by Silvester [13] in 1968 and by Ahmed and Daly [14] in 1969. This was followed in 1970 by the nonlinear finite element analysis of electric machine problems by Silvester and Chari [15] and the vector finite element formulation of dielectric loaded waveguides by Csendes and Silvester [16] and by Daly [17].

Applied mathematicians became interested in the finite element method in 1968 with the publication of theoretical papers by Zlamal [18] and by Birkhoff, Schultz and Varga [19]. Further theoretical treatment from the point of view of error bounds for the finite element method was given by Babuska [20] and by Schultz [21].

In the past two to three years, the number of papers appearing on the finite element method has been increasing at an explosive rate. A large part of the engineering applications of these methods is contained in two recent books on the finite element method, one by Zienkiewicz [22] and the other by Oden [23]. However, there is already so much

literature on the subject that, on many topics, these books are limited to mere surveys of the material or, in some cases, to just citing bibliographical references of the original papers.

A similar situation also exists in the theoretical analysis of the finite element method, as evidenced by a recent survey paper by Zlamal [24]. This paper, along with one by Whiteman [25], provides an extensive bibliography of finite element methods. A more reasonably self-contained treatment of the mathematical implications of the finite element method, including rigorous convergence proofs of the method, may be found in a monograph by Varga [26].

Another development of importance to the finite element method, but not directly related to it, is the study of optimal finite difference schemes made by Babuska, Prager and Vitasek [27]. They prove that the consistent Galerkin approximation of a boundary value problem solves the problem of minimizing the norm of the error operator for finite difference methods. Since this is the criterion by which finite element formula are established, it implies that finite element solutions are in general more accurate than the corresponding finite difference solutions.

As originally conceived, the term - the finite element method- had a rather narrow definition. To many early workers, such as Argyris and Zienkiewicz, it meant using basis functions that had local support, were interpolatory at the edges of the elements and resulted in piecewise continuous solutions. However, as in all

successful areas in which many people work, with every new development, the scope of the finite element method has become enlarged. Silvester and Hsieh [28] have used the term when working with an exterior problem in an infinite region to define finite elements of infinite size;Brauchli and Oden [29] employ conjugate approximation functions in the finite element method which are non-zero over the entire solution region; Hazel and Wexler [30] have formulated finite element solutions in terms of non-interpolatory basis functions; and Reichert and Vogt [31] provide such a general definition of the finite element method that it includes almost any method in which pieces are connected together, specifically including the five point finite difference operator. (On the other hand, Babuska, Prager, and Vitasek [27], and more recently Miranker [32], have adopted such a general definition of the finite difference method that it includes virtually all finite element methods!)

In view of this altering usage and common confusion over notation, it is necessary to examine those properties which specifically characterize the finite element method and to define the method accordingly. First of all, a finite element method must be projective. This property implies that there is some kind of correspondence between the analytic space of approximating functions and a finite dimensional vector space. It

therefore distinguishes the finite element method from the more general definition of finite difference methods given by Babuska, Prager and Vitasek [27] in which no analytic function space is defined. Second, in a finite element method, one or more element shapes must be specified over which a standard set of basis functions is defined. Third, the finite element solution of a boundary value problem in a region that is a union of elements must be given as a combination of the solutions of the corresponding elemental boundary value problems. These three conditions correspond closely to the properties by which the finite element method was originally developed and yet provide a flexible definition well-suited to contemporary usage.

Within this general definition of finite element methods there coexist several widely different procedures of developing and using the method, its elements and their basis functions. Most of the early work [33] developed finite element procedures using linear approximation polynomials between the function values at the vertices of triangular or rectangular elements. With these simple approximating functions, the matrix elements of the discretized equation can be evaluated analytically for finite elements of a general shape. The resulting algebraic formulae are then used to generate the elements of a matrix equation for the solution in specific cases.

However, in quest of higher solution accuracies, more sophisticated finite elements have been introduced.

Among the large number of element shapes and basis functions that have been tried, two types of finite elements predominate. One is the isoparametric elements described by Zienkiewicz [22] in which the basis functions are Cartesian products of one - dimensional interpolation polynomials in curvilinear coordinates. The others are the high-order symmetric triangular interpolation polynomial elements discovered independently and with entirely different approaches by Silvester [34] and by Irons [35].

An undesirable byproduct of the use of high-order finite elements is an increase in the level of complexity in the procedures used to evaluate the matrix elements. With isoparametric elements, the technique of evaluating general algebraic formulae for the elements of finite element matrices has been abandoned entirely and all calculations are performed from first principles using numerical methods [22]. With the high-order triangular finite elements, it is possible to retain the efficient algebraic approach for generating matrix elements, but the required analytic calculations are lengthy and difficult. Thus, at the present time, the main shortcoming of projective methods is the complicated and expensive calculations needed to discretize each separate type of differential equation.

The primary concern of this thesis is to derive an efficient and accurate algebraic procedure for the discretization of arbitrary differential equations. In order to do so, it will be necessary to devise discrete matrix operators which have analogous combinatorial behavior to the elementary components

of analytic operators. Once such matrix operators are developed, the matrix equivalent of any differential equation can be determined by simply combining the elementary matrix operators algebraically.

The chief obstacle to forming matrix equivalents of the elementary components of analytic operators is the fact that classical projective methods treat differential equations and boundary conditions as an inseparable unit called a boundary value problem. In these formulations, it is not possible to develop an atomic approach to discretizing differential equations because the entire boundary value problem is fused together. Consequently, in this thesis, the extension of projective methods to solve differential equations which are independent of boundary conditions will be considered. It will be shown that the formulation obtained from this approach yields, for the first time by the direct application of a numerical method, general solutions of differential equations instead of particular ones.

There is, however, in addition to the theoretical limitations mentioned above, a major practical drawback in applying projective methods to solve many common engineering problems. The use of a projective method to solve a differential equation generally requires the development of an extremely complicated and expensive computer program. An indication of the extensive labor sometimes required is provided by the analysis of the Boeing model 747 aircraft wing-body intersection by finite element methods [36]. For this problem, approximately one hundred man-months of effort and about twenty-

eight hours of CDC 6600 computer time were required to convert existing theory into a working computer program. Clearly, although many engineering problems would benefit from analysis by projective methods, very few of them generate sufficient interest to merit this kind of individual attention.

Consequently, in order for a discretization procedure for arbitrary differential equations to have maximum impact it must be coupled with the development of an efficient computer implementation of the method. By providing a general computer program which performs each of the discretization steps for differential equations automatically, both the difficulty of discretizing arbitrary differential equations and program development costs can be eliminated. For this reason, the ultimate objective of this thesis is to develop general computer procedures by which large classes of differential equations can be solved with a minimum of analysis and data preparation.

In pursuit of this objective, a special purpose computer language has been developed for the symbolic representation of differential equations and two computer programs based on this language - one for the solution of arbitrary linear ordinary differential equations and the other for arbitrary linear two-dimensional partial differential equations - have been written. In addition, the linear ordinary differential equation program has been adapted to solve nonlinear ordinary differential equations by means of Newton's method.

Listings of these computer programs will be presented in the thesis along with numerous examples of their operation. Although many of the problems selected can be solved by using alternative methods, it will be demonstrated that none of these alternatives can furnish solutions as profitably as the present programs, in terms of accuracy, computing speed and, especially, units of an engineer's time.

In summary form, the major original contributions of this thesis are:

(1) The development of a purely algebraic discretization procedure for arbitrary linear ordinary and partial differential equations.

(2) The definition and evaluation of rectangular matrix equivalents for the three elementary components of differential equations

(3) The application of matrix generalized inversion to determine approximate general solutions of differential equations.

(4) The development of a computer language and computer programs for the automatic solution of arbitrary linear ordinary and two-dimensional partial differential equations and for the solution of many nonlinear ordinary differential equations by Newton's method.

CHAPTER II

ELEMENTARY OPERATORS

The first step in developing a projective procedure capable of discretizing arbitrary differential equations is to devise a simple method for obtaining their numerical equivalents. In order to do this, it is important to observe that with analytic formulations, any differential equation is simply an algebraic combination of functions with the differential operator. It follows that the construction of a numerical equivalent of a differential equation is most easily accomplished by recognizing the primary numerical constituents of the method used and assembling complicated matrix equations from these components.

In order to proceed, it will be helpful to define the term elementary operators to refer only to those operators which cannot be decomposed further into sums or products of other operators. Although other possibilities exist, for differential equations the simplest and most generally applicable quantities to designate as elementary operators are $\frac{d}{dx}$ and $f(x)$. Any operator that is composed of a product of elementary operators will be called a simple operator, and operators containing a sum of simple operators will be referred to as compound operators. As an example, the compound operator $D = \frac{d^2}{dx^2} + k^2$ consists of two simple operators $\frac{d^2}{dx^2}$ and k^2. These two simple operators are composed of the elementary operators $\frac{d}{dx}$ and k.

The use of matrices to replace the process of differentiation in finite element discretization has been independently

advocated by Boisserie [37] and by Silvester [38]. In the first case, the differentiation matrices are square and corresponded to one-dimensional differentiation, with matrix products generating two and three dimensional finite elements. In the second, rectangular matrices are used to differentiate Newton-Cotes interpolation polynomials over triangular regions. However, neither paper properly develops a complete, consistent approach for obtaining discretized forms of compound operators from these matrix factors.

2.1 DIFFERENTIATION MATRICES

The elementary factor $\frac{d}{dx}$ is the most basic operator in the theory of differential equations. As is well known, its customary definition is

$$Dy = \frac{dy}{dx} = \lim_{x_2 \to x_1} \frac{y_2 - y_1}{x_2 - x_1} \qquad (2.1.1)$$

where y_1 and y_2 are the values of a function $y(x)$ at the points x_1 and x_2, respectively. Definition (2.1.1) is only valid if the limit exists and is unique for both $x_2 > x_1$ and $x_1 > x_2$. In this equation, the following convention, which will be pursued throughout this thesis, has been introduced: wherever practical, functions are denoted by lower case letters and operators by capital letters; italics are used to denote analytic expressions while Roman type denotes discrete representations.

The standard numerical equivalent of (2.1.1) is to write

a finite difference approximation [39]

$$Dy \simeq \frac{y_2 - y_1}{x_2 - x_1} \qquad (2.1.2)$$

which converges to (2.1.1) as the length of the finite interval $L = x_2 - x_1$ tends to zero. It is apparent that this procedure is equivalent to approximating the function y on the interval $I = [\, x_1, x_2 \,]$ by the linear polynomial which passes through y_1 and y_2 at the endpoints of the interval. The error introduced by using (2.1.2) instead of (2.1.1) in the interval I is therefore related to the accuracy with which the function y may be approximated by a linear polynomial in I.

It is well known that given a finite number of values, a function can be approximated much more accurately by a few high order polynomials than by many low order ones. Therefore, it is reasonable to seek more accurate finite difference representations of (2.1.1) by approximating y with high order polynomials and then determining the corresponding finite difference formula. Since these formulae are to be in terms of point values, the approximating polynomials should be expressed in terms of an interpolatory basis and may be written as follows

$$y \simeq \sum_{i=0}^{n} y_i l_i^{(n)}(x) \qquad (2.1.3)$$

where the $\{ l_i^{(n)}(x) \}$ are the n'th order Lagrange interpolation polynomials

$$l_i^{(n)}(x) = \frac{p_i^{(n)}(x)}{p_i^{(n)}(x_i)} \tag{2.1.4}$$

Here

$$p_i^{(n)}(x) = \prod_{\substack{m=0 \\ m \neq i}}^{n} (x - x_i) \tag{2.1.5}$$

The key to forming high order discrete representations of the differential operator (2.1.1) is obtained by recognizing that if an n'th order polynomial is used to approximate y, the function generated by the process of differentiation will be a polynomial of $(n-1)$'st order. Therefore, whenever y is an n'th order polynomial, the function Dy may be written as

$$Dy = z = \sum_{i=0}^{n-1} z_i \, l_i^{(n-1)}(x) \tag{2.1.6}$$

where again, since point values are desired, the expansion functions are Lagrange interpolation polynomials. Equations (2.1.1), (2.1.3), and (2.1.6) yield

$$\sum_{i=0}^{n-1} z_i \, l_i^{(n-1)}(x) = \sum_{i=0}^{n} y_i \, \frac{d \, l_i^{(n)}(x)}{dx} \tag{2.1.7}$$

Now

$$l_i^{(n)}(x_j^{(n)}) = \delta_{ij} \qquad j=0,\ldots,n \tag{2.1.8}$$

So that

$$z_j = \sum_{i=0}^{n} \frac{d\, l_i^{(n)}(x_j^{(n-1)})}{dx}\, y_i \qquad j=0,\ldots,n-1 \tag{2.1.9}$$

In matrix form this is

$$z = Dy \tag{2.1.10}$$

where z is an n-component vector containing the numbers z_i, y is an $(n+1)$ component vector containing the numbers y_i and D is the $nx(n+1)$ matrix with elements

$$D_{i+1,\,j+1} = \frac{d\, l_i^{(n)}(x_j^{(n-1)})}{dx} \tag{2.1.11}$$

$$= \begin{cases} \dfrac{\sum\limits_{\substack{m=0 \\ m\neq j}}^{n} p_m^{(n)}(x_i^{(n-1)})}{(x_i^{(n-1)} - x_j^{(n)})\, p_j^{(n)}(x_j^{(n)})} & \text{if } x_i^{(n-1)} \neq x_j^{(n)} \\ \qquad\qquad j=0,\ldots,n \\ \dfrac{1}{x_k^{(n)} - x_j^{(n)}} \dfrac{p_k^{(n)}(x_k^{(n)})}{p_j^{(n)}(x_j^{(n)})} & \text{if } x_i^{(n-1)} = x_k^{(n)} \neq x_j^{(n)} \end{cases}$$

$$\left| \sum_{\substack{m=0 \\ m\neq j}}^{n} \frac{1}{x_j - x_m} \right| \qquad \text{if} \quad x_i^{(n-1)} = x_j^{(n)}$$

As introduced here, in this thesis, non-subscripted lower case letters will indicate vectors having as components the corresponding subscripted discrete values of a function. Similarly, non-subscripted capital letters will refer to matrices which give the discrete representation of an operator.

The simplest interpolation polynomials are the Lagrangian polynomials in which the nodal points are equi-spaced in the interval I of length L. Taking

$$x_i^{(n-1)} = \frac{L}{n-1}\, i \qquad i=0,\ldots,n-1$$

$$x_j^{(n)} = \frac{L}{n}\, j \qquad j=0,\ldots,n \tag{2.1.12}$$

for $n\geq 2$ the components of the matrix D become

$$D_{i+1,j+1} = \begin{cases} \dfrac{1}{L}\,\dfrac{n(n-1)}{ni-(n-1)j} \displaystyle\sum_{\substack{m=0 \\ m\neq j}}^{n} (-1)^{j-m}\,\frac{(n-m)!\,m!}{(n-j)!\,j!} & \text{if } x_i^{(n-1)} \neq x_j^{(n)} \\ \dfrac{n(-1)^{j-k}}{L(k-j)}\,\dfrac{(n-k)!\,k!}{(n-j)!\,j!} & \text{if } x_i^{(n-1)} = x_k^{(n)} \neq x_j^{(n)} \\ \dfrac{n}{L} \displaystyle\sum_{\substack{m=0 \\ m\neq j}}^{n} \frac{1}{j-m} & \text{if } x_i^{(n-1)} = x_j^{(n)} \end{cases} \qquad j=0,\ldots,n \tag{2.1.13}$$

The differentiation matrices (or D matrices) corresponding to equi-spaced Lagrangean polynomials have been evaluated exactly using Formac symbol manipulator language [40] for polynomial orders ranging from one to nine and are presented in Table 2.1.1. In this table, the following antisymmetry property

$$D_{ij} = -D_{n+2-i,\ n+1-j} \tag{2.1.14}$$

is evident. The lowest order case, with $n=1$, yields for equation (2.1.10).

$$[z_1] = \frac{1}{L}[-1\ \ 1]\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{2.1.15}$$

This is equation (2.1. 2) again, only in matrix form. The next higher approximation of (2.1.1) results when y is approximated by a quadratic ($n=2$) in which case equation (2.1.10) is

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \frac{1}{L}\begin{bmatrix} -3 & 4 & -1 \\ 1 & -4 & 3 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{2.1.16}$$

The higher order approximations produce similar rectangular matrix equations.

The elements of the D matrices possess a number of interesting and potentially useful properties. First, it may be noted that the rank of each matrix is one less than the dimension of its domain and that, in fact, the row sums of each and every one of these matrices are zero. This is a necessary consequence of the fact that differentiating a constant, in

Table 2.1.1

The first nine D matriees for equi-spaced Lagrangean polynomials.

FIRST ORDER

-0.1000000E 01	0.1000000E 01

SECOND ORDER

-0.3000000E 01	0.4000000E 01	-0.1000000E 01
0.1000000E 01	-0.4000000E 01	0.3000000E 01

THIRD ORDER

-0.5500000E 01	0.9000000E 01	-0.4500000E 01	0.1000000E 01
0.1250000E 00	-0.3375000E 01	0.3375000E 01	-0.1250000E 00
-0.1000000E 01	0.4500000E 01	-0.9000000E 01	0.5500000E 01

FOURTH ORDER

-0.8333333E 01	0.1600000E 02	-0.1200000E 02	0.5333333E 01	-0.1000000E 01
-0.8641970E-01	-0.4543209E 01	0.5481481E 01	-0.9876543E 00	0.1358024E 00
-0.1358024E 00	0.9876543E 00	-0.5481481E 01	0.4543209E 01	-0.8641970E-01
0.1000000E 01	-0.5333333E 01	0.1200000E 02	-0.1600000E 02	0.8333333E 01

FIFTH ORDER

-0.1141667E 02	0.2500000E 02	-0.2500000E 02	0.1666666E 02	-0.6250000E 01	0.1000000E 01
-0.2065430E 00	-0.6115723E 01	0.8455403E 01	-0.2856445E 01	0.8422852E 00	-0.1189778E 00
-0.2343750E-01	0.3255208E 00	-0.5859375E 01	0.5859375E 01	-0.3255208E 00	0.2343750E-01
0.1189778E 00	-0.8422852E 00	0.2856445E 01	-0.8455403E 01	0.6115723E 01	0.2065430E 00
-0.1000000E 01	0.6250000E 01	-0.1666666E 02	0.2500000E 02	-0.2500000E 02	0.1141667E 02

SIXTH ORDER

-0.1470000E 02	0.3800000E 02	-0.4500000E 02	0.4000000E 02	-0.2250000E 02	0.7200000E 01	-0.1000000E 01
-0.2899340E 00	-0.7946495E 01	0.1227024E 02	-0.6136319E 01	0.2802239E 01	-0.8040960E 00	0.1044160E 00
0.1091200E-01	0.1612800E-01	-0.6880320E 01	0.7861759E 01	-0.1228319E 01	0.2465280E 00	-0.2668800E-01
0.2668800E-01	-0.2465280E 00	0.1228319E 01	-0.7861759E 01	0.6880320E 01	-0.1612800E-01	-0.1091200E-01
-0.1044160E 00	0.8040960E 00	-0.2802239E 01	0.6136319E 01	-0.1227024E 02	0.7946495E 01	0.2899340E 00
0.1000000E 01	-0.7200000E 01	0.2250000E 02	-0.4000000E 02	0.4500000E 02	-0.3800000E 02	0.1470000E 02

SEVENTH ORDER

-0.1814999E 02	0.4900000E 02	-0.7350000E 02	0.8166666E 02	-0.6125000E 02	0.2939999E 02	-0.8166666E 01	0.1000000E 01
-0.3533074E 00	-0.9978644E 01	0.1693700E 02	-0.1125226E 02	0.6980412E 01	-0.3031933E 01	0.7916695E 00	-0.9293956E-01
0.2466087E-01	-0.1788675E 00	-0.8204755E 01	0.1041465E 02	-0.2785703E 01	0.9152492E 00	-0.2076207E 00	0.2238988E-01
0.4882813E-02	-0.6699216E-01	0.5582682E 00	-0.8374023E 01	0.8374023E 01	-0.5582682E 00	0.6699216E-01	-0.4882813E-02
-0.2238988E-01	0.2076207E 00	-0.9152492E 00	0.2785703E 01	-0.1041465E 02	0.8204755E 01	-0.1788675E 00	0.2466087E-01
0.9293956E-01	-0.7916695E 00	0.3031933E 01	-0.6980412E 01	0.1125226E 02	-0.1693700E 02	0.9978644E 01	0.3533074E 00
-0.1000000E 01	0.8166666E 01	-0.2939999E 02	0.6125000E 02	-0.8166666E 02	0.7350000E 02	-0.4900000E 02	0.1814999E 02

EIGHTH ORDER

-0.2174284E 02	0.6400000E 02	-0.1120000E 03	0.1493333E 03	-0.1400000E 03	0.8959999E 02	-0.3733333E 02	0.9142857E 01	-0.1000000E 01
-0.4038324E 00	-0.1217923E 02	0.2267006E 02	-0.1864162E 02	0.1463445E 02	-0.8526219E 01	0.3351299E 01	-0.7887137E 00	0.8380908E-01
0.3077522E-01	-0.3203885E 00	-0.9734117E 01	0.1349625E 02	-0.5181765E 01	0.2372499E 01	-0.8254480E 00	0.1804458E 00	-0.1824857E-01
-0.1902870E-02	0.6405771E-02	0.1773061E 00	-0.9320146E 01	0.1033605E 02	-0.1493566E 01	0.3536136E 00	-0.6344265E-01	0.5684670E-02
-0.5684670E-02	0.6344265E-01	-0.3536136E 00	0.1493566E 01	-0.1033605E 02	0.9320146E 01	-0.1773061E 00	0.6405771E-02	-0.1902870E-02
0.1824857E-01	-0.1804458E 00	0.8254480E 00	-0.2372499E 01	0.5181765E 01	-0.1349625E 02	0.9734117E 01	-0.3203885E 00	0.3077522E-01
-0.8380908E-01	0.7887137E 00	-0.3351299E 01	0.8526219E 01	-0.1463445E 02	0.1864162E 02	-0.2267006E 02	0.1217923E 02	0.4038324E 00
0.1000000E 01	-0.9142857E 01	0.3733333E 02	-0.8959999E 02	0.1400000E 03	-0.1493333E 03	0.1120000E 03	-0.6400000E 02	0.2174284E 02

NINTH ORDER

0.2268000E 03	-0.1460000E 03	0.4628571E 02	-0.1012500E 02	0.1000000E 01	-0.2546071E 02	0.8100000E 02	-0.1620000E 03	0.2520000E 03	-0.2835000E 03
-0.1998264E 02	0.1049893E 02	-0.3712512E 01	0.7899240E 00	-0.7639217E-01	0.4454859E 00	-0.1452567E 02	0.2888249E 02	-0.2875063E 02	0.2732198E 02
0.5065068E 01	-0.2381735E 01	0.7873353E 00	-0.1601081E 00	0.1498817E-01	-0.3350372E-01	0.4313614E 00	-0.1142455E 02	0.1710728E 02	-0.8610425E 01
-0.1270405E 02	0.1011144E 01	-0.2869058E 00	0.5322668E-01	-0.4686568E-02	0.4190784E-02	-0.4141372E-01	0.7950306E-01	-0.1051599E 02	0.1270405E 02
0.1090118E 02	-0.8074951E 00	0.1245849E 00	-0.1589094E-01	0.1068115E-02	-0.1068115E-02	0.1589094E-01	-0.1245849E 00	0.8074951E 00	-0.1090118E 02
-0.1270405E 02	0.1051599E 02	-0.7950306E-01	0.4141372E-01	-0.4190784E-02	0.4686568E-02	-0.5322668E-01	0.2869058E 00	-0.1011144E 01	0.2918559E 01
0.8610425E 01	-0.1710728E 02	0.1142455E 02	-0.4313614E 00	0.3350372E-01	-0.1498817E-01	0.1601081E 00	-0.7873353E 00	0.2381735E 01	-0.5065068E 01
-0.2732198E 02	0.2875063E 02	-0.2888249E 02	0.1452567E 02	-0.4454859E 00	0.7639217E-01	-0.7899240E 00	0.3712512E 01	-0.1049893E 02	0.1998264E 02
0.2835000E 03	-0.2520000E 03	0.1620000E 03	-0.8100000E 02	0.2546071E 02	-0.1000000E 01	0.1012500E 02	-0.4628571E 02	0.1260000E 03	-0.2268000E 03

which case all of the y_i are equal, results in zero. Thus the nullspace of a D matrix corresponds to the nullspace of the operator $\frac{d}{dx}$.

Similarly, by noting that the n'th order D matrix differentiates polynomials of degree less then or equal to n, one surmises from the equation

$$\frac{d}{dx}\, x^p = p\, x^{p-1} \tag{2.1.17}$$

that

$$\sum_{j=0}^{n} D_{ij}\, (x_j^{(n)})^p = p(x_i^{(n-1)})^{p-1} \qquad \begin{matrix} i=0,\ldots,n-1 \\ p=0,\ldots,n \end{matrix} \tag{2.1.18}$$

This result is valid for general differentiation matrix elements. In particular, with the Lagrangean matrix elements (2.1.13), the following formulae are obtained.

$$\sum_{j=1}^{n} \frac{(-1)^{j+1}\, n!}{(n-j)!\, j!}\, j^{p-1} = 0 \qquad p=0,\ldots,n \tag{2.1.19a}$$

$$\sum_{j=0}^{n} \sum_{\substack{m=0 \\ m\neq j}}^{n} \frac{(-1)^{j-m}\, (n-m)!m!}{(ni-(n-1)j)\, (n-j)!j!} \left(\frac{j}{n}\right)^p = \frac{p}{n}\, \frac{i^{p-1}}{(n-1)^p} \qquad \begin{matrix} i=1,\ldots,n-2 \\ p=0,\ldots,n \end{matrix} \tag{2.1.19b}$$

$$\sum_{j=0}^{n-1} \frac{1}{n-j} \left\{ \frac{(-1)^{j-n}\, n!}{(n-j)!\, j!} \left(\frac{j}{n}\right)^p + 1 \right\} = \frac{p}{n} \qquad p=0,\ldots,n \tag{2.1.19c}$$

Here, equation (2.1.12) and the fact that $x_i \neq x_j$ unless $i \neq j = 0$ or $i = n-1$ and $j = n$ has been used. These formulae may also be proved by induction.

It is also interesting to note in passing that the differentiation matrices (2.1.10) may be used to provide an alternative definition of differentiation. First note that equation (2.1.1) may be written as

$$Dy = \lim_{L \to 0} \frac{1}{L} \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \lim_{L \to 0} D^{(1)} y^{(1)} \qquad (2.1.20)$$

where $D^{(1)}$ indicates the first order D matrix. One may also write

$$Dy = \lim_{L \to 0} \frac{1}{L} \begin{bmatrix} -3 & 4 & -1 \\ 1 & -4 & 3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \lim_{L \to 0} D^{(2)} y^{(2)} \qquad (2.1.21)$$

which will converge faster, but under similar conditions as (2.1.20). Formulae of this type are essentially the same as (2.1.1) since differentiable functions may be approximated by straight lines in the limit as $L \to 0$. Consider, however, the sequence of functions

$$z^{(n)} = \sum_{i=o}^{n} z_i^{(n)} \, l_i^{(n)}(x) \qquad (2.1.22)$$

defined by the equations

$$z^{(n-1)} = D^{(n)}\ y^{(n)} \tag{2.1.23}$$

An alternative definition of the derivative of y is provided by

$$y'(x) = \lim_{n\to\infty} z^{(n)}(x) \tag{2.1.24}$$

whenever the sequence $\{z^{(n)}\}$ is uniformly convergent.

2.2 SIMPLE OPERATORS

In this section, methods of developing matrix representations of simple operators will be considered. These will be obtained by taking products of the elementary D matrices of the previous section and some new elementary matrices which correspond to functions.

Fortunately, as each elementary operator $\frac{d}{dx}$ of a product $\frac{d^p}{dx^p}$ acts on a polynomial expression, it generates a polynomial of one lower degree. Therefore, high order differentiation matrices can be obtained by simply multiplying together the succession of matrices which corresponds to the range of polynomials upon which the elementary operators act. For example, to obtain the quadratic equivalent of $\frac{d^2}{dx^2}$ one forms

$$D^{(1)}D^{(2)}y^{(2)} = \frac{1}{L}[-1\ \ 1]\ \frac{1}{L}\begin{pmatrix}-3 & 4 & -1\\ 1 & -4 & 3\end{pmatrix}\begin{pmatrix}y_1\\ y_2\\ y_3\end{pmatrix} \qquad (2.2.1)$$

$$= \frac{4}{L^2}\ [1\ -2\ 1]\ \begin{pmatrix}y_1\\ y_2\\ y_3\end{pmatrix}$$

This equation is identical to the well known finite difference formula for $\frac{d^2}{dx^2}$ [39].

In a similar manner, by using the matrix values given in Table 2.1.1, high-order polynomial representations of any differential operator of degree less than ten may be formed. These high-order D matrices have similar properties to low order ones. For example, the dimensionality of the nullspace of the D matrices corresponding to $\frac{d^2}{dx^2}$ is equal to two, with row sums and linearly weighted row sums equal to zero.

It remains to develop matrix representations for simple operators which contain functions. In order to do this, consider the expression

$$z(x) = f(x)y(x) \qquad (2.2.2)$$

where $y(x)$ is given by an n'th order polynomial

$$y^{(n)}(x) = \sum_{i=0}^{n} y_i^{(n)} l_i^{(n)}(x) \qquad (2.2.3)$$

and assume that $z(x)$ is to be approximated by an m'th order polynomial

$$z^{(m)}(x) = \sum_{i=o}^{m} z_i^{(m)} l_i^{(m)}(x) \qquad (2.2.4)$$

The problem is to determine the matrix operator F which produces the mapping

$$z = Fy \qquad (2.2.5)$$

corresponding to (2.2.1).

Taking

$$y_i^{(n)} = \delta_{ij} \qquad j=o,\ldots,n \qquad (2.2.6)$$

in (2.2.5) produces the n sets of m equations

$$z_i^{(m)} = F_{ij} \qquad \begin{matrix} i=o,\ldots,m \\ j=o,\ldots,n \end{matrix} \qquad (2.2.7)$$

Now if $z_i^{(m)}$ is taken to be the value of the function $f(x)y(x)$ at the point $x_i^{(m)}$, then, using (2.2.6)

$$F_{ij} = z_i^{(m)} = f(x_i^{(m)}) \sum_{j=o}^{n} y_j^{(n)}(x_i^{(m)}) \qquad (2.2.8)$$

$$= f(x_i^{(m)}) l_j^{(n)}(x_i^{(m)})$$

Notice, first of all, that if $f(x)=1$ and $n\neq m$ then (2.2.7) defines an operator which maps a function into a higher or a lower order space. The raising operator with $m>n$ is exact and provides a convenient method for imbedding a low order polynomial into a

higher order space. Table 2.2.1 contains the values of this matrix with $m=n+1$ for polynomial orders ranging from zero to eight.

The most useful form of (2.2.7) arises, however, when $n=m$ since then $l_i^{(n)}(x_i^{(m)}) = \delta_{ij}$ and

$$F_{ij} = f(x_i^{(m)})\delta_{ij} \qquad (2.2.9)$$

Thus, in order to find the product of a function $y^{(n)}$ expressed in terms of n'th order interpolation polynomials and an arbitrary function $f(x)$, it is only necessary to multiply the vector of coefficients of y by a diagonal matrix containing the values of the function $f(x)$ at the interpolation nodes. This procedure has the same effect as weighting the vector y by the nodal values of $f(x)$.

2.3 COMPOUND OPERATORS

Before the matrix equivalent of an arbitrary differential operator can be determined, the effect of additions on the discretization procedure of the previous sections must be determined. That such operations may not be performed in a straightforward manner may be seen by considering the compound operator $D = \frac{dy}{dx} - y$. If y is approximated by a quadratic, then the first term in this expression is a linear polynomial having a two component vector representation while the second term is a quadratic having a three component representation. Clearly, it is not possible to add these two vectors.

Table 2.2.1

The first nine FU matrices for equi-spaced Lagrangean polynomials.

1 ST ORDER

0.1000000E 01	
0.1000000E 01	

2 ND ORDER

0.1000000E 01	0.0
0.5000000E 00	0.5000000E 00
0.0	0.1000000E 01

3 TH ORDER

0.1000000E 01	0.0	0.0
0.2222222E 00	0.8888888E 00	-0.1111111E 00
0.1111111E 00	0.8888888E 00	0.2222222E 00
0.0	0.0	0.1000000E 01

4 TH ORDER MATRIX

0.1000000E 01	0.0	0.0	0.5625000E 00
0.1171875E 00	0.1054688E 01	-0.2109375E 00	0.1049805E 00
-0.6250000E-01	0.5625000E 00	0.5625000E 00	-0.6250000E-01
0.1049805E 00	-0.2109375E 00	0.1054688E 01	0.1171875E 00
0.5625000E 00	0.0	0.0	0.1000000E 01

5 TH ORDER MATRIX

0.1000000E 01	0.0	0.0	0.0	-0.1536000E 00
0.7040000E-01	0.1126400E 01	-0.2816000E 00	0.1023999E 00	-0.2841344E-01
-0.3360000E-01	0.3584000E 00	0.8063999E 00	-0.1536000E 00	0.2756096E-01
0.2756096E-01	-0.1536000E 00	0.8063999E 00	0.3584000E 00	-0.3360000E-01
-0.2841344E-01	0.1023999E 00	-0.2816000E 00	0.1126400E 01	0.7040000E-01
-0.1536000E 00	0.0	0.0	0.0	0.1000000E 01

6 TH ORDER MATRIX

0.1000000E 01	0.0	0.0	0.0	0.5859375E 00	0.5733449E-01
0.4632309E-01	0.1158076E 01	-0.3308792E 00	0.1781657E 00	-0.3380900E-01	-0.9322089E-01
-0.1920439E-01	0.2400548E 00	0.9602194E 00	-0.2400548E 00	0.5733449E-01	-0.3249779E-01
0.1171875E-01	-0.9765625E-01	0.5859375E 00	0.5859375E 00	-0.9765625E-01	0.1171875E-01
-0.3249779E-01	0.5733449E-01	-0.2400548E 00	0.9602194E 00	0.2400548E 00	-0.1920439E-01
-0.9322089E-01	-0.3380900E-01	0.1781657E 00	-0.3308792E 00	0.1158076E 01	0.4632309E-01
0.5733449E-01	0.5859375E 00	0.0	0.0	0.0	0.1000000E 01

7 TH ORDER MATRIX

0.1000000E 01	0.0	0.0	0.0	0.0	-0.1716632E 00	-0.3470582E-01
0.3253746E-01	0.1171349E 01	-0.3660464E 00	0.2602997E 00	-0.1331078E 00	0.3480585E-01	0.3948175E-01
-0.1172980E-01	0.1689092E 00	0.1055682E 01	-0.3127948E 00	0.1319603E 00	-0.3470582E-01	0.1173713E-01
0.6357893E-02	-0.6242297E-01	0.4291579E 00	0.7629474E 00	-0.1716632E 00	0.3929992E-01	-0.7442292E-02
-0.7442292E-02	0.3929992E-01	-0.1716632E 00	0.7629474E 00	0.4291579E 00	-0.6242297E-01	0.6357893E-02
0.1173713E-01	-0.3470582E-01	0.1319603E 00	-0.3127948E 00	0.1055682E 01	0.1689092E 00	-0.1172980E-01
0.3948175E-01	0.3480585E-01	-0.1331078E 00	0.2602997E 00	-0.3660464E 00	0.1171349E 01	0.3253746E-01
-0.3470582E-01	-0.1716632E 00	0.0	0.0	0.0	0.0	0.1000000E 01

8 TH ORDER MATRIX

0.1000000E 01	0.0	0.0	0.0	0.0	0.5981445E 00	0.8675355E-01	0.6205067E-02
0.2399191E-01	0.1175603E 01	-0.3918679E 00	0.3457658E 00	-0.2351207E 00	0.1212236E 00	-0.1689447E 00	-0.2166624E-01
-0.7587433E-02	0.1239281E 00	0.1115353E 01	-0.3717842E 00	0.2065468E 00	-0.9033471E-01	0.6205067E-02	0.1884082E-01
0.3643781E-02	-0.4120275E-01	0.3213814E 00	0.8927263E 00	-0.2634708E 00	0.8675355E-01	-0.1453304E-01	0.9497818E-02
-0.2441406E-02	0.2392578E-01	-0.1196289E 00	0.5981445E 00	0.5981445E 00	-0.1196289E 00	0.2392578E-01	-0.2441406E-02
0.9497818E-02	-0.1453304E-01	0.8675355E-01	-0.2434708E 00	0.8927263E 00	0.3213814E 00	-0.4120275E-01	0.3643781E-02
0.1884082E-01	0.6205067E-02	-0.9033471E-01	0.2065468E 00	-0.3717842E 00	0.1115353E 01	0.1239281E 00	-0.7587433E-02
-0.2166624E-01	-0.1689447E 00	0.1212236E 00	-0.2351207E 00	0.3457658E 00	-0.3918679E 00	0.1175603E 01	0.2399191E-01
0.6205067E-02	0.8675355E-01	0.5981445E 00	0.0	0.0	0.0	0.0	0.1000000E 01

9 TH ORDER MATRIX

0.1000000E 01	0.0	0.0	0.0	0.0	0.0	-0.1813876E 00	-0.4971430E-01	-0.8695211E-02
0.1836341E-01	0.1175258E 01	-0.4113403E 00	0.4329898E 00	-0.3672681E 00	0.2223461E 00	-0.9275264E-01	0.8372003E-01	0.1249007E-01
-0.5147040E-02	0.9411728E-01	0.1152937E 01	-0.4192498E 00	0.2882342E 00	-0.1590258E 00	0.6161451E-01	-0.8695211E-02	-0.9161662E-02
0.2201561E-02	-0.2817998E-01	0.2465748E 00	0.9862995E 00	-0.3082185E 00	0.1408999E 00	-0.4971430E-01	0.9212058E-02	-0.3842480E-02
-0.1315370E-02	0.1464620E-01	-0.8421564E-01	0.4716078E 00	0.7368872E 00	-0.1813876E 00	0.5383047E-01	-0.1001270E-01	0.1464620E-01
0.2042285E-02	-0.1001270E-01	0.5383047E-01	-0.1813876E 00	0.7368872E 00	0.4716078E 00	-0.8421564E-01	0.1464620E-01	-0.2817998E-01
-0.3842480E-02	0.9212058E-02	-0.4971430E-01	0.1408999E 00	-0.3082185E 00	0.9862995E 00	0.2465748E 00	-0.2817998E-01	0.9411728E-01
-0.9161662E-02	-0.8695211E-02	0.6161451E-01	-0.1590258E 00	0.2882342E 00	-0.4192498E 00	0.1152937E 01	0.9411728E-01	-0.5147040E-02
0.1249007E-01	0.8372003E-01	-0.9275264E-01	0.2223461E 00	-0.3672681E 00	0.4329898E 00	-0.4113403E 00	0.1175258E 01	0.1836341E-01
-0.8695211E-02	-0.4971430E-01	-0.1813876E 00	0.0	0.0	0.0	0.0	0.0	0.1000000E 01

A way out of this dilemma is to form two separate approximations of y, one in the quadratic space and another in the linear space

$$y \simeq y^{(2)} = \sum_{i=0}^{2} y_i^{(2)} l_i^{(2)}(x) \qquad (2.3.1a)$$

$$y \simeq y^{(1)} = \sum_{i=0}^{1} y_i^{(1)} l_i^{(1)}(x) \qquad (2.3.1b)$$

Then, using (2.3.1a) for the first term of D and (2.3.1b) for the second, this yields, just as before

$$D = \frac{1}{L}\begin{bmatrix} -3 & 4 & -1 \\ 1 & -4 & 3 \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} - \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} \qquad (2.3.2)$$

Now, although the terms of the matrix expression (2.3.2) are consistent in that they are both two component vectors, the expression can be further simplified by defining a relationship between the two approximations (2.3.1a) and (2.3.1b). One such relationship is provided by (2.2.5) with $m=2$ and $n=0$

$$\begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} \qquad (2.3.3)$$

However, since this relationship is obtained by matching the function values at the endpoints of the interval, it provides a very crude method of approximating a quadratic by a linear polynomial.

A much better approximation is obtained by determining the best linear approximation to a quadratic in a least squares (L_2) sense in the interval I. Under this condition, the norm of the difference of the two functions $y^{(1)}$ and $y^{(2)}$

$$Q = \int_I |y^{(2)} - y^{(1)}|^2 \, dx \tag{2.3.4}$$

must be a minimum. Using (2.3.1), this means that

$$\frac{\partial Q}{\partial y_i^{(1)}} = 0 = -\sum_{j=o}^{2} y_j^{(2)} \int l_i^{(1)} l_j^{(2)} dx + \sum_{j=o}^{1} y_j^{(1)} \int l_i^{(1)} l_j^{(1)} dx \qquad i=o,1 \tag{2.3.5}$$

Written in matrix form, this equation is

$$S\, y^{(2)} = T\, y^{(1)} \tag{2.3.6}$$

where the elements of S and T are

$$S_{ij} = \int_I l_i^{(1)} l_j^{(2)} dx \tag{2.3.7}$$

$$T_{ij} = \int_I l_i^{(1)} l_j^{(1)} dx$$

Since the $\{l_i^{(1)}\}$ are linearly independent, the metric T is non-singular and may be inverted to yield

$$y^{(1)} = P y^{(2)} \tag{2.3.8}$$

Table 2.3.1

The first nine P matrices for equi-spaced Lagrangean polynomials.

FIRST ORDER

0.5000000E 00

SECOND ORDER

0.6666663E 00	0.6666658E 00	-0.3333331E 00
-0.3333331E 00	0.6666658E 00	0.6666663E 00

THIRD ORDER

0.7750000E 00	0.6749994E 00	-0.6749998E 00
-0.6250000E-01	0.5624996E 00	0.5624996E 00
0.2250000E 00	-0.6749998E 00	0.6749994E 00

0.2250000E 00	-0.6250000E-01	0.7750000E 00

FOURTH ORDER

0.8476199E 00	0.6095246E 00	-0.9142850E 00	0.6095240E 00	-0.1523812E 00
-0.4303357E-01	0.6659608E 00	0.4825394E 00	-0.1241621E 00	0.1869489E-01
0.1869489E-01	-0.1241621E 00	0.4825394E 00	0.6659608E 00	-0.4303357E-01
-0.1523812E 00	0.6095240E 00	-0.9142850E 00	0.6095246E 00	0.8476199E 00

FIFTH ORDER

0.8966596E 00	0.5166998E 00	-0.1033399E 01	0.1033397E 01	-0.5166997E 00	0.1033398E 00
-0.3748272E-01	0.7513772E 00	0.3771259E 00	-0.1085714E 00	0.1766453E-01	-0.1149662E-03
0.1171878E-01	-0.9765595E-01	0.5859367E 00	0.5859367E 00	-0.9765595E-01	0.1171878E-01
-0.1149662E-03	0.1766453E-01	-0.1085714E 00	0.3771259E 00	0.7513772E 00	-0.3748272E-01
0.1033398E 00	-0.5166997E 00	0.1033397E 01	-0.1033399E 01	0.5166998E 00	0.8966596E 00

SIXTH ORDER

0.9298682E 00	0.4207785E 00	-0.1051947E 01	0.1402593E 01	-0.1051945E 01	0.4207786E 00	-0.7012987E-01
-0.3249792E-01	0.8078514E 00	0.2786120E 00	-0.3100245E-01	-0.4970780E-01	0.3370729E-01	-0.6981860E-02
0.1263948E-01	-0.1057889E 00	0.7137529E 00	0.4460873E 00	-0.7248598E-01	0.4802525E-02	0.9915829E-03
0.9915829E-03	0.4802525E-02	-0.7248598E-01	0.4460873E 00	0.7137529E 00	-0.1057889E 00	0.1263948E-01
-0.6981860E-02	0.3370729E-01	-0.4970780E-01	-0.3100245E-01	0.2786120E 00	0.8078514E 00	-0.3249792E-01
-0.7012987E-01	0.4207786E 00	-0.1051945E 01	0.1402593E 01	-0.1051947E 01	0.4207785E 00	0.9298682E 00

SEVENTH ORDER

0.9523873E 00	0.3332787E 00	-0.9998332E 00	0.1666388E 01	-0.1666386E 01	0.9998300E 00	-0.3332774E 00	0.4761118E-01
-0.2692467E-01	0.8382511E 00	0.2143154E 00	0.5630188E-01	-0.1779187E 00	0.1416493E 00	-0.5403583E-01	0.8359618E-02
0.1237332E-01	-0.1100858E 00	0.8231683E 00	0.2710915E 00	0.5751620E-01	-0.8380115E-01	0.3540195E-01	-0.5667053E-02
-0.2441231E-02	0.2392584E-01	-0.1196292E 00	0.5981441E 00	0.5981441E 00	-0.1196292E 00	0.2392584E-01	-0.2441231E-02
-0.5667053E-02	0.3540195E-01	-0.8380115E-01	0.5751620E-01	0.2710915E 00	0.8231683E 00	-0.1100858E 00	0.1237332E-01
0.8359618E-02	-0.5403583E-01	0.1416493E 00	-0.1779187E 00	0.5630188E-01	0.2143154E 00	0.8382511E 00	-0.2692467E-01
0.4761118E-01	-0.3332774E 00	0.9998300E 00	-0.1666386E 01	0.1666388E 01	-0.9998332E 00	0.3332787E 00	0.9523873E 00

EIGHTH ORDER

0.9676681E 00	0.2586508E 00	-0.9052740E 00	0.1810544E 01	-0.2263177E 01	0.1810539E 01	-0.9052674E 00	0.2586472E 00	-0.3233135E-01
-0.2142016E-01	0.8503695E 00	0.1924124E 00	0.1026679E 00	-0.3111444E 00	0.3193315E 00	-0.1803761E 00	0.5543256E-01	-0.7274166E-02
0.1054308E-01	-0.1031985E 00	0.8890578E 00	0.1221955E 00	0.2431535E 00	-0.2776696E 00	0.1581689E 00	-0.4835462E-01	0.6301545E-02
-0.4438974E-02	0.4009373E-01	-0.1787941E 00	0.7935565E 00	0.3704581E 00	0.8853078E-03	-0.3347039E-01	0.1371503E-01	-0.2006311E-02
-0.2006311E-02	0.1371503E-01	-0.3347039E-01	0.8853078E-03	0.3704581E 00	0.7935565E 00	-0.1787941E 00	0.4009373E-01	-0.4438974E-02
0.6301545E-02	-0.4835462E-01	0.1581689E 00	-0.2776696E 00	0.2431535E 00	0.1221955E 00	0.8890578E 00	-0.1031985E 00	0.1054308E-01
-0.7274166E-02	0.5543256E-01	-0.1803761E 00	0.3193315E 00	-0.3111444E 00	0.1026679E 00	0.1924124E 00	0.8503695E 00	-0.2142016E-01
-0.3233135E-01	0.2586472E 00	-0.9052674E 00	0.1810539E 01	-0.2263177E 01	0.1810544E 01	-0.9052740E 00	0.2586508E 00	0.9676681E 00

NINTH ORDER

0.9780403E 00	0.1976281E 00	-0.7905123E 00	0.1844526E 01	-0.2766784E 01	0.2766784E 01	-0.1844520E 01	0.7905053E 00	-0.1976236E 00	0.2195871E-01
-0.1657708E-01	0.8521119E 00	0.2065551E 00	0.8037233E-01	-0.3772748E 00	0.5014911E 00	-0.3831653E 00	0.1780190E 00	-0.4695141E-01	0.5420014E-02
0.8031312E-02	-0.8775628E-01	0.9080182E 00	0.4737662E-01	0.3930955E 00	-0.5196846F 00	0.3858394E 00	-0.1751310E 00	0.4537535E-01	-0.5166575E-02
-0.4730690E-02	0.4579636E-01	-0.2147574E 00	0.9676048E 00	-0.8806598E-01	0.2671953E 00	-0.2345209E 00	0.1113424E 00	-0.2937223E-01	0.3373881E-02
0.3373881E-02	-0.2937223E-01	0.1113424E 00	-0.2345209E 00	0.2671953E 00	0.8806598E-01	0.9676048E 00	-0.2147574E 00	0.4579636E-01	0.5338341E-03
-0.5166575E-02	0.4537535E-01	-0.1751310E 00	0.3858394E 00	-0.5196846E 00	0.3930955E 00	0.4737662E-01	0.9080182E 00	-0.8775628E-01	-0.4730690E-02
0.5420014E-02	-0.4695141E-01	0.1780190E 00	-0.3831653E 00	0.5014911E 00	-0.3772748E 00	0.8037233E-01	0.2065551E 00	0.8521119E 00	0.8031312E-02
0.2195871E-01	-0.1976236E 00	0.7905053E 00	-0.1844520E 01	0.2766784E 01	-0.2766784E 01	0.1844526E 01	-0.7905123E 00	0.1976281E 00	-0.1657708E-01
									0.9780403E 00

where

$$P = T^{-1}S \tag{2.3.9}$$

Equation (2.3.8) gives the coefficients of the approximation (2.3.1a) and has the effect of projecting the second order polynomial $y^{(2)}$ into the best linear polynomial in the L_2 sense. Using (2.3.8), equation (2.3.2) may be written as

$$Dy = \frac{1}{L}\begin{bmatrix}-3 & 4 & -1\\ 1 & -4 & 3\end{bmatrix}\begin{bmatrix}y_1^{(2)}\\ y_2^{(2)}\\ y_3^{(2)}\end{bmatrix} - \frac{1}{3}\begin{bmatrix}2 & 2 & -1\\ -1 & 2 & 2\end{bmatrix}\begin{bmatrix}y_1^{(2)}\\ y_2^{(2)}\\ y_3^{(2)}\end{bmatrix} \tag{2.3.10}$$

The projection matrix (or P matrix) given in equation (2.3.9) may also be defined between polynomial spaces of differing degree and have been evaluated using the Formac language for all polynomial orders ranging from one to nine. These values are presented in Table 2.3.1. The P matrices have an analogous topological behavior to the differentiation matrices, although their physical interpretation is different. They are symmetric about their centroids

$$P_{ij} = P_{n+2-i,\ n+1-j} \tag{2.3.11}$$

and not anti-symmetric, as were the D matrices. Their rank is again one less than the dimension of their domain space, however, in this case, the nullvectors of P are symmetric about $y = 0$ in an L_2 sense.

As was the case with differentiation matrices, projection matrices between spaces differing by more than one order can be determined by ordinary matrix multiplication of the corresponding

elementary P matrices. For example, the best possible zeroth order polynomial approximation of the quadratic $y^{(2)}$ is given by

$$\begin{bmatrix} y_1^{(o)} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \end{bmatrix} \frac{1}{3} \begin{bmatrix} 2 & 2 & -1 \\ -1 & 2 & 2 \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix}$$

$$= \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} \qquad (2.3.12)$$

In a point matching sense, the equation corresponding to (2.3.12) would be from (2.2.5)

$$\begin{bmatrix} y_1^{(2)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \end{bmatrix} \qquad (2.3.13)$$

where the interpolation "node" of the zeroth order polynomial is taken in the midpoint of the interval. Therefore, for the differential operator

$$D = \frac{d^2 y}{dx^2} + y \qquad (2.3.14)$$

the following two finite difference representations are obtained

$$D = \frac{4}{L^2} \left[y_{i-1} - 2y_i + y_{i+1} \right] + \frac{1}{6} \left[y_{i-1} + 4y_i + y_{i+1} \right] \qquad (2.3.15a)$$

and

$$D = \frac{4}{L^2} \left[y_{i-1} - 2y_i + y_{i+1} \right] + y_i \qquad (2.3.15b)$$

Equation (2.3.15b) is the standard finite difference formula for (2.3.14) while (2.3.15a) is the "asymptotically optimal"

finite difference formula for (2.3.14) derived by Miranker [32] using a more circuitous theory. The only difference between the two is in the type of approximation made between the polynomial spaces. Miranker compared the formulae (2.3.15) computationally [32] and found that the L_2 projective approximation was as much as 10^4 times more accurate in the Euclidean error norm of point values than the point matching approximation.

CHAPTER III

PROJECTIVE APROXIMATION

In the preceding chapter, the elementary matrix concept was introduced by using heuristic arguments. In this chapter, elementary matrices will be developed more formally, using the theory of projective approximation. The procedures outlined in Chapter 2 will thus be established rigorously here, resulting in a foolproof algorithm for the solution of differential equations.

Somewhat surprisingly, very little work has been done in the area of projective approximations of pure differential equations, considering the wealth of similar material on boundary value problems. The only reference available in the literature on this subject appears to be a paper by Locker [41], who considers the problem of determining the least-squares solution of an n'th order differential equation to which k boundary conditions have been added, with k not necessarily equal to n. Unfortunately, the theory developed in this paper has several limitations to practical application of the method. It assumes that the homogeneous solutions of the differential equation are known, a condition very rarely occurring in practice, and concentrates on determining only the inhomogeneous solution. Moreover, the procedure is theoretically proved to converge to a unique inhomogeneous solution, but the paper does not specify how the com-

putation should be constructed numerically.

3.1 GALERKIN'S METHOD

In this section, the basic mathematical equations for the projective solution of differential equations will be given. These equations are related to the classically used Galerkin's method, except that the procedure will be applied to differential equations independently of boundary conditions. The main result of this section will be to determine a unique range space for differential operators which result in approximate general solutions of differential equations.

In a projective method, the objective of the analysis is to determine an approximate solution of the equation

$$Dy = f \tag{3.1.1}$$

by replacing the function y, which belongs to a real-valued Hilbert space Σ with norm $|| f || = < f \mid f >^{\frac{1}{2}}$, with a function $y^{(f)}$ in a finite dimensional subspace $\Sigma^{(f)}$ of Σ. In the present case, D is taken to be a linear differential operator defined over a geometric region I and f is a given function of Σ which lies in the range of D. In the following analysis, it will be helpful to decompose the differential operator D into a sum of simple operators D_s

$$D = \sum_{s=1}^{S} D_s \tag{3.1.2}$$

and to consider the range spaces $\Sigma^{(s)}$ generated by each of these operators separately. These range spaces are defined by the equations

$$\Sigma^{(s)} = \{ z : z = D_{sf}\, y^{(f)},\ y^{(f)} \in \Sigma^{(f)} \} \qquad (3.1.3)$$

Here and in the following, subscripts on an operator indicate that

$$\text{Range}(A_{ij}) = \Sigma^{(i)} \qquad (3.1.4a)$$

$$\text{Domain}(A_{ij}) = \Sigma^{(j)} \qquad (3.1.4b)$$

$$\text{Domain}(A_{i}) = \Sigma \qquad (3.1.4c)$$

Now, it is necessary to provide a general definition of the projective mapping introducted in section 2.3. This is done as follows. A projection operator P_{ij} is defined to be an operator which maps any element of a space $\Sigma^{(j)}$ into the closest (in the norm sense) element of $\Sigma^{(i)}$

$$P_{ij} = \{ A_{ij} : y^{(i)} = A_{ij}\, y^{(j)},\ y^{(j)} \in \Sigma^{(j)} \qquad (3.1.5)$$

$$\Rightarrow \| y^{(i)} - y^{(j)} \| = \underset{z^{(i)} \in \Sigma^{(i)}}{\text{minimum}} \| z^{(i)} - y^{(j)} \| \}$$

Projection operators are well known in the theory of linear spaces [42,43] and have a number of interesting properties. The most important of these are that they are linear

$$P_{ij}\,(y^{(j)} + z^{(j)}) = P_{ij}\, y^{(j)} + P_{ij}\, z^{(j)} \qquad (3.1.6)$$

and that for three nested subspaces $\Sigma^{(i)}$, $\Sigma^{(j)}$, $\Sigma^{(k)}$ with $\Sigma^{(i)} \subseteq \Sigma^{(j)} \subseteq \Sigma^{(k)}$

$$P_{ik} = P_{ij}\, P_{jk} \qquad (3.1.7)$$

Note, however, that in the event that the space $\Sigma^{(i)}$ is larger than the space $\Sigma^{(j)}$, P_{ij} is not equal to the identity operator, as is commonly assumed, since the projective embedding of a smaller space into a larger one is not unique.

The underlying idea of many projective methods is to replace the problem of solving equation (3.1.1) by the task of solving the equation

$$\sum_{s=1}^{S} P_{0s} D_{sf} y^{(f)} = f^{(0)} \qquad (3.1.8)$$

where $f^{(0)} = P_0 f$ and $\Sigma^{(0)}$ is some finite-dimensional subspace of Σ that is called the range space of the approximation. If $\Sigma^{(0)}$ is taken to be equal to $\Sigma^{(f)}$ then (3.1.8) yields the Bubnov-Galerkin method, while the Galerkin-Petrov method results if $\Sigma^{(0)}$ does not coincide with $\Sigma^{(f)}$ [2].

Unfortunately, however, the usual choices for $\Sigma^{(0)}$ that appear in the literature have a major flaw: they do not preserve the dimension of the nullspace of the differential operator D. As a result, with these projective methods, it is not possible to obtain the general solution of the differential equation (3.1.1), but only particular solutions to specified boundary value problems. In order to produce finite-dimensional equations of the form (3.1.8) which behave in a manner analogous to that of (3.1.1), a space $\Sigma^{(0)}$ must be found such that the operator $\sum_{s=1}^{S} P_{0s} D_{sf}$ has approximately the same nullspace as the original operator D. For ordinary differential equations, this implies that the range space $\Sigma^{(0)}$

should be of dimension

$$\text{Dim}(\Sigma^{(0)}) = \text{Dim}(\Sigma^{(f)}) - \text{Nullity}(D) \qquad (3.1.9)$$

where the nullity of a compound ordinary differential operator is given by the highest power of differentiation among its simple components. Extension of this concept to partial differential equations will be considered in section 3.5.

A comparison can be made between the method contained in equation (3.1.8) and the asymptotically optimal finite difference formulations of Babuska, Prager and Vitasek [27] and of Miranker [32]. In their theory, the criterion for establishing the asymptotic optimality of a finite difference formula is to determine whether or not it minimizes the norm of the residual of the Galerkin process:

$$y_{op}^{(f)} = \{\, y^{(f)} : \| Dy^{(f)} - f \| = \underset{z^{(f)} \in \Sigma^{(f)}}{\text{minimum}} \| Dz^{(f)} - f \| \,\} \qquad (3.1.10)$$

In the method developed in this thesis, solutions are obtained by saying that functions in the space $\Sigma^{(0)}$ are the best possible approximations of functions in the spaces $\Sigma^{(s)}$. When the differential equation (3.1.1) has only two terms, as in equation (2.3.14), the two approaches yield identical results; if the equation has more than two terms, they do not. The advantage of the present formulation is the ease with which numerical matrix equivalents can be generated for arbitrary differential equations, compared to the difficulty of evaluating asymptotically optimal finite difference formula.

3.2 INTRODUCING BASIS FUNCTIONS

The first step in discretizing equation (3.1.8) is to introduce a set of basis functions $\{b_i^{(f)}\}$ in the space $\Sigma^{(f)}$ and a basis $\{b_i^{(o)}\}$ in $\Sigma^{(o)}$. Then

$$y^{(f)} = \sum_{i=1}^{n} y_i b_i^{(f)} \tag{3.2.1}$$

$$f^{(o)} = \sum_{i=1}^{m} f_i b_i^{(o)} \tag{3.2.2}$$

and (3.1.8) becomes

$$\sum_{i=1}^{n} y_i \sum_{s=1}^{S} P_{os} D_{sf} b_i^{(f)} = \sum_{i=1}^{m} f_i b_i^{(o)} \tag{3.2.3}$$

Taking the scalar product of this equation with each function in a conjugate basis set $\{ b_{(o)}^i \}$ in $\Sigma^{(o)}$ results in the equation

$$\sum_{s=1}^{S} A_s y = Bf \tag{3.2.4}$$

Here, as before, y and f are vectors containing the coefficients of the expansion (3.2.1) and (3.2.2), respectively, the A_s are $m \times n$ matrices with elements

$$A_{ij} = \langle b_{(o)}^i | P_{os} D_{sf} b_j^{(f)} \rangle \tag{3.2.5}$$

and B is an $m \times m$ matrix with elements

$$B_{ij} = \langle b_{(o)}^i | b_j^{(o)} \rangle \tag{3.2.6}$$

Hence, in a finite dimensional subspace, a differential equation is equivalent to a rectangular matrix equation with analytic operators replaced by rectangular matrices and functions by vectors. The correspondence between the vectors and the con-

tinuous functions is provided by equation (3.2.1)

Consider now the problem of determining the function $f^{(o)}$, the best approximant to the function f in the space $\Sigma^{(o)}$. This problem is treated in linear analysis and the presentation below draws from references [42,43,23]. Using (3.1.5) the definition of $f^{(0)}$ given by (3.2.2) implies that

(3.2.7)

$$\begin{aligned} ||f-f^{(o)}||^2 &= \langle f - f^{(o)} | f - f^{(o)} \rangle \\ &= \langle f - \sum_{i=1}^{m} f_i b_i^{(o)} | f - \sum_{i=1}^{m} f_i b_i^{(o)} \rangle \\ &= \langle f | f \rangle - 2 \sum_{i=1}^{m} f_i \langle b_i^{(o)} | f \rangle + \sum_{i,j=1}^{m} f_i f_j \langle b_i^{(o)} | b_j^{(o)} \rangle \\ &= \underset{f^{(o)} \in \Sigma^{(o)}}{\text{minimum}} \end{aligned}$$

Differentiating this with respect to f_k yields

$$\langle b_k^{(o)} | f \rangle = \sum_{i=1}^{m} T_{ki} f_i \qquad k=1,\ldots,m \qquad (3.2.8)$$

where

$$T_{ij} = \langle b_i^{(o)} | b_j^{(o)} \rangle \qquad (3.2.9)$$

Therefore

$$f_i = \sum_{j=1}^{m} T^{ij} \langle b_j^{(o)} | f \rangle = \langle h_{(o)}^{i} | f \rangle \qquad (3.2.10)$$

where

$$h_{(o)}^{i} = \sum_{j=1}^{m} T^{ij} b_j^{(o)} \qquad (3.2.11)$$

and T^{ij} are the elements of the inverse of T

$$\sum_{j=1}^{m} T^{ij} T_{jk} = \delta_{ik} \qquad (3.2.12)$$

Using this result, the function $f^{(o)}$ becomes

$$f^{(o)} = P_o f = \sum_{ij} b_i^{(o)} T^{ij} \langle b_j^{(o)} | f \rangle \qquad (3.2.13)$$

Consequently, the operator P_0 may be written as

$$P_o = \sum_{ij} b_i^{(o)} T^{ij} \langle b_j^{(o)} | \qquad (3.2.14)$$

where the open inner product is understood to be completed with whatever function P_0 acts on. Although (3.2.14) has been derived specifically between the spaces Σ and $\Sigma^{(o)}$, it is clear that the projection operator P_{ij} between any two spaces $\Sigma^{(i)}$ and $\Sigma^{(j)}$ will have the same form, with the basis $\{b^{(o)}\}$ replaced by $\{b^{(i)}\}$.

The projective property of the operator P_o is demonstrated in the following manner. If a function f is decomposed into two functions, one in $\Sigma^{(0)}$ and the other in $\Sigma^{(x)} = \Sigma \backslash \Sigma^{(o)}$

$$f = f^{(o)} + f^{(x)}, \quad f^{(x)} \in \Sigma^{(x)} \qquad (3.2.15)$$

then, taking the scalar product of this with $b_i^{(o)}$ and using (3.2.2) gives

$$\langle b_i^{(o)} | f \rangle = \sum_j T_{ij} f_j + \langle b_i^{(o)} | f^{(x)} \rangle \qquad (3.2.16)$$

Together with (3.2.10), this implies that

$$\langle b_i^{(o)} | f^{(x)} \rangle = 0 \qquad i=1,\ldots,m \qquad (3.2.17)$$

As a result, when the f_i are chosen as in (3.2.10), the operator P separates any function in Σ into a function $f^{(o)}$ in $\Sigma^{(o)}$ and a function $f^{(x)}$ which is orthogonal to $f^{(o)}$.

The functions $\{h_{(o)}^{\ i}\}$ in (3.2.11) are biorthogonal to the basis set $\{b_i^{(o)}\}$ in $\Sigma^{(o)}$ since

$$\langle h_{(o)}^{\ i} \,|b_j^{(o)}\rangle = \sum_k T^{ik}\, T_{kj} = \delta_{ij} \qquad (3.2.18)$$

In terms of finite element analysis, their use has been introduced by Brauchli and Oden [29] and cultivated by Oden [23]. Since (3.2.10) states that the best approximation of a function f in the space $\Sigma^{(o)}$ occurs when the expansion coefficients f_i equal the inner products of f with the biorthogonal basis functions in $\Sigma^{(o)}$, their application in approximating functions cannot be avoided. However, in determining the optimal solution of differential equations, it is fortunately not necessary to restrict the conjugate functions $\{b_{(o)}^{\ i}\}$ to be biorthogonal. In order to see why, note that a typical entry of the term Bf in equation (3.2.4) is

$$t_i = \sum_j B_{ij}\, f_j = \sum_{jk} B_{ij}\, T^{jk}\, \langle b_k^{(o)} \,|f\rangle \qquad (3.2.19)$$

Since both sets $\{b_i^{(o)}\}$ and $\{b_{(o)}^{\ i}\}$ form a basis in $\Sigma^{(o)}$, they are related by a linear transformation G

$$b_{(o)}^{\ i} = \sum_j G^{ij}\, b_j^{(o)} \qquad (3.2.20)$$

The elements of the matrix B may then be written

$$B_{ij} = \langle b_{(o)}^{\ i} \,|b_j^{(o)}\rangle = \sum_l G^{il}\, T_{lj} \qquad (3.2.21)$$

and equation (3.2.19) becomes

$$t_i = \sum_{ljk} G^{il} T_{lj} T^{jk} \langle b_k^{(o)} | f \rangle$$

$$= \sum_k G^{ik} \langle b_k^{(o)} | f \rangle \qquad (3.2.22)$$

$$= \langle b_{(o)}^{\ i} | f \rangle$$

In this equation, the relationship (3.2.20) between the bases $\{b_i^{(o)}\}$ and $\{b_{(o)}^{\ i}\}$ is immaterial. Therefore, in order to obtain the best approximation of f in (3.2.4), the components of the term Bf must simply equal the scalar products of the conjugate basis functions $\{b_{(o)}^{\ i}\}$ with the function f. Provided that all terms of (3.2.4) use it consistently, any conjugate basis set pair will generate an optimal approximation of f.

The evaluation of the remaining terms $\sum_{s=1}^{S} A_s y$ of equation (3.2.4) will now be examined in terms of the elementary matrix concept. First of all, each simple differential operator $P_{1s} D_{sf}$ may be factored into a sequence of elementary operators

$$P_{1s}\, D_{sf} = E_{12}\, E_{23} \cdots E_{f-1,f} \qquad (3.2.23)$$

and a sequence of spaces $\Sigma^{(i)}$ with bases $\{b^{(i)}\}$ can be defined by

$$\Sigma^{(i-1)} = \{z : z = E_{i-1,i}\ y\ ,\ y \in \Sigma^{(i)}\}\ i=1,\ldots,f \qquad (3.2.24)$$

The matrix elements in (3.2.5) may be written

$$A_{sij} = \langle b_{(o)}^{i} | E_{01} E_{12} \cdots E_{f-1,f} b_j^{(f)} \rangle \qquad (3.2.25)$$

$$= \langle b_{(o)}^{i} | E_{01} \cdots E_{f-2,f-1} P_{f-1,f-1} E_{f-1,f} b_j^{(f)} \rangle$$

$$= \sum \langle b_{(o)}^{i} | E_{01} \cdots E_{f-2,f-1} b_p^{(f-1)} \rangle T^{pq} G_{qr} \langle b_{(f-1)}^{r} | E_{f-1,f} b_j^{(f)} \rangle$$

$$= \sum \langle b_{(o)}^{i} | E_{01} b_p^{(1)} \rangle T^{pq} G_{qr} \langle b_{(1)}^{r} | E_{12} b^{(2)} \rangle$$

$$\cdots T^{tu} G_{uv} \langle b_{(f-1)}^{v} | E_{f-1,f} b_j^{(f)} \rangle$$

In the event that the basis functions $\{b^{(p)}\}$ and $\{b_{(q)}\}$ are biorthogonal

$$\langle b_{(p)}^{i} | b^{(q)}_{j} \rangle = \sum_q G^{pq} T_{qr} = \delta_{pr} \qquad (3.2.26)$$

and the calculation of A_s may be performed by multiplying together the sequence of elementary matrices formed by the elementary operators

$$A_s = \prod_{k=0}^{f} E_k \qquad (3.2.27)$$

where

$$E_{k_{ij}} = \langle b_{(k)}^{i} | E_{k,k+1} b^{(k+1)} \rangle \qquad (3.2.28)$$

Otherwise, the matrix products $T^{-1}G^{-1}$ must be inserted between

the elementary matrices as shown in section (3.2.2).

3.3 RELATION TO THE ENERGY NORM APPROXIMATION

It is well known that an energy norm can be defined for every boundary value problem such that the extremization of this energy norm in a projective subspace results in a least residual approximation [43]. It is therefore instructive to compare the conditions in equation (3.2.4) with the conditions obtained from energy norm extremization.

In order to do this, the differential equation (3.1.1) must be replaced by a variational functional. Of the commonly used variational principles, the most generally applicable is the one contained in the following theorem.

THEOREM 3.3.1 [43]. Let D be an arbitrary differential operator and D^* its adjoint

$$D^* = \{A: \langle Aw \,|y\rangle = \langle w|\, Dy\rangle,\ y \in \text{Domain}(D),\ w \in \text{Range}(D)\} \tag{3.3.1}$$

Then the solutions of the pair of equations

$$Dy = f \tag{3.3.2}$$

and

$$D^* u = f \tag{3.3.3}$$

are the one and only pair of functions in a Hilbet space Σ which makes the functional

$$F(v,z) = \langle v|\ Dz\rangle - \langle v + z\ |f\rangle \qquad (3.3.4)$$

stationary.

The proof of this theorem may be found in reference [43]. Note that the functional

$$F(z) = \langle z|\ Dz\rangle - 2\langle\ z|\ f\rangle \qquad (3.3.5)$$

often used with symmetric operators ($D^* = D$) is a special case of (3.3.4) with the arbitrary function v set equal to z.

Approximate solutions of the differential equation (3.3.1) are obtained from the functional (3.3.4) by determining those elements $y^{(f)}$ and $u^{(1)}$ of the finite dimensional subspaces $\Sigma^{(f)}$ and $\Sigma^{(1)}$ which make F stationary. These approximate solutions will converge to the exact solution of (3.3.2) as the space $\Sigma^{(f)}$ is enlarged to Σ. As before, let $\{b_i^{(f)}\}$ be the basis in $\Sigma^{(f)}$ and $\{b_{(1)}^{\ i}\}$ be the conjugate basis of $\{b_i^{(1)}\}$ in $\Sigma^{(1)}$. Then

$$z^{(f)} = \sum_{i=1}^{n} z_i\ b_i^{(f)} \qquad (3.3.6a)$$

$$v^{(1)} = \sum_{i=1}^{m} v_i b_{(1)}^{\ i} \qquad (3.3.6b)$$

and

$$F(v^{(1)},\ z^{(f)}) = \sum_{i=1}^{n}\sum_{j=1}^{m} z_i\ v_j\ \langle b_{(1)}^{\ j}\ |Db_i^{(f)}\rangle - \sum_{i=1}^{m} v_i\langle b_{(1)}^{\ i}\ |f\rangle - \sum_{i=1}^{n} z_i\langle b_i^{(f)}\ |f\rangle \qquad (3.3.7)$$

Differentiating this with respect to the coefficients z_k and v_k gives

$$\frac{\partial F}{\partial z_k} = 0 = \sum_{j=1}^{m} v_j \langle b_{K1)}^{j} \, | Db_k^{(f)} \rangle - \langle b_k^{(f)} \, | f \rangle$$
$$= \sum_{j=1}^{m} v_j \langle b_k^{(f)} \, | D^* \, b^j \rangle_{(1)} - \langle b_k^{(f)} \, | f \rangle \qquad (3.3.8)$$

$$\frac{\partial F}{\partial v_k} = 0 = \sum_{i=1}^{n} z_i \langle b_{(1)}^{k} | \, Db_i^{(f)} \rangle - \langle b_{(1)}^{k} | \, f \rangle \qquad (3.3.9)$$

Equation (3.3.9) differs from (3.2.4) only in that (3.3.9) does not contain the projection operators P_{0s}. Provided that these projection operators are added to the operator D in the above equations, (3.3.9) will result in the same rectangular matrix equation for (3.3.2) as is obtained from Galerkin's method. Equation (3.3.8) needs to be solved only if a solution to the adjoint problem (3.3.3) is desired.

3.4 ELEMENTARY MATRICES REVISITED

In this section, the elementary matrices of Chapter 2 will be derived from the expressions contained in section 3.2. The chief impediment to doing this is to determine functions $\{b_{(k)}\}$which possess the sifting property in (2.1.9). Fortunately, Brauchli and Oden give finite dimensional delta functions with this property in reference [29]. These finite dimensional delta functions may also be produced by projecting

the Dirac delta function into a finite dimensional subspace by using (3.2.14). Let $\delta(x - x_0)$ be the Dirac delta function

$$\delta(x-x_0) = 0 \qquad \text{If } x \neq x_0$$
$$\langle 1 | \delta(x - x_0)\rangle = 1 \tag{3.4.3}$$

and define $\Delta(x - x_0)$ to be the function obtained by acting with P_1 on $\delta(x - x_0)$

$$\begin{aligned}\Delta(x-x_0) &= \sum_{ij} b_i^{(1)}\, T^{ij} \langle b_j^{(1)} |\ \delta(x - x_0)\rangle \\ &= \sum_{ij} b_i^{(1)}\, T^{ij}\ \ b_j^{(1)}(x_0) \\ &= \sum_i \Delta_i\, b_i^{(1)}\end{aligned} \tag{3.4.4}$$

where

$$\Delta_i = \sum_j T^{ij}\ b_j^{(1)}(x_0) \tag{3.4.5}$$

The remarkable property of the function Δ is that while it is a smooth, well-defined function in the finite dimensional subspace $\Sigma^{(1)}$, it has preserved the important sifting property of the Dirac delta function for the space $\Sigma^{(1)}$. For example, let $y^{(1)}$ be an arbitrary function in $\Sigma^{(1)}$. Then

$$\begin{aligned}\langle \Delta^{(1)}(x-x_0) |\ y^{(1)}\rangle &= \sum_{ijk} T^{ij}\ b_j^{(1)}(x_0)\, \langle b_i^{(1)} |\ y_k b_k^{(1)}\rangle \\ &= \sum_{ijk} y_k\, b_j^{(1)}(x_0)\, T^{ij}\, T_{ik} \\ &= \sum_k y_k\, b_k^{(1)}(x_0) = y^{(1)}(x_0)\end{aligned} \tag{3.4.6}$$

Now let $y^{(1)}$ be equal to the basis function $l_i^{(1)}$ which

interpolates on the set of points $\{x_j\}$. Then

$$\langle \Delta^{(1)}(x - x_j) \mid l_i^{(1)} \rangle = l_i^{(1)}(x_j) = \delta_{ij} \qquad (3.4.7)$$

Consequently, the finite dimensional delta functions $\{\Delta^{(1)}(x - x_i)\}$ form a biorthogonal set to the interpolatory basis functions.

Using the delta functions for the conjugate basis set $\{b_{(k)}\}$ in (3.2.2), the elements of the matrix E_k are

$$E_{k_{ij}} = \langle \Delta^{(k)}(x - x_i) \mid E_{k,k+1}\, l_j^{(k+1)} \rangle$$

$$= E_{k,k+1}\, l_j^{(k+1)} \mid_{x = x_i}$$

If $E_{k,k+1}$ is equal to $\frac{d}{dx}$ or to $f(x)$, this expression reduces to (2.1.11) or to (2.2.8), in that order. Note that this result is independent of the norm used to define the space. If $E_{k,k+1}$ is equal to $P_{k,k+1}$ and an L_2 norm is taken

$$E_{k_{ij}} = \sum_{ml} b_m(x_i)\, T^{ml} \langle b_l^{(k)} \mid b_j^{(k+1)} \rangle$$

$$= \sum_{l} T^{il} S_{lj} \qquad (3.4.9)$$

This result is the same as (2.3.9). Therefore, the elementary matrices of Chapter 2 are identical to those obtained from (3.2.2).

The general form of the elementary matrix in equation (3.2.2) permits, however, many other basis functions to be used, besides the interpolatory basis of section 2.2, and some of these will be considered here. The most obvious such set is, of course, the monomials

$$b_i^{(k)} = x^i \qquad (3.4.10)$$

With the conjugate functions being the biorthogonal set, the following elementary D and P matrices are obtained

$$D = \frac{1}{L}\begin{pmatrix} 0 & 1 & & & \\ & 2 & & & \\ & & 3 & & \\ & & & \ddots & \\ & & & & n \end{pmatrix} \qquad P = \begin{pmatrix} 1 & & & p_1 \\ & 1 & & p_2 \\ & & 1 & p_3 \\ & & & \vdots \\ & & & p_n \end{pmatrix} \qquad (3.4.11)$$

where the p_i depend on the order of the approximation and all blank entries are zero. The function matrix in this case requires the expansion of the function in a Taylor series.

Another set of basis functions for which the D and P matrices generate simple forms are the Legendre polynomials. In this case

$$D=\frac{1}{L}\begin{pmatrix} 0 & 1 & 0 & 1 & \circ & \circ & \circ & 1 \\ & & 3 & 0 & \circ & \circ & \circ & 0 \\ & & & 5 & \circ & \circ & \circ & 5 \\ & & & & & & & \circ \\ & & & & & & & \circ \\ & & & & & & & \circ \\ & & & & & & & 2n-1 \end{pmatrix} \qquad P=\begin{pmatrix} 1 & & & & & & & 0 \\ & 1 & & & & & & 0 \\ & & 1 & & & & & 0 \\ & & & \circ & & & & \circ \\ & & & & \circ & & & \circ \\ & & & & & \circ & & \circ \\ & & & & & & 1 & 0 \end{pmatrix} \qquad (3.4.12)$$

Again, however, the function matrix must be evaluated analytically to obtain the Legendre polynomial expression of the function. With a non-interpolatory basis, it is not possible to obtain a purely numerical method to generate function matrices.

3.5 MULTI-DIMENSIONAL REGIONS

In section 3.2, the technique for obtaining the approximate solution of a differential equation in a projective subspace was restricted to ordinary differential equations. Here, the extension of this method to partial differential equations will be made in a simple and natural way.

In order to retain the usage of the ordinary elementary matrices in the discretization of partial differential equations, it is necessary to use multi-dimensional basis functions which are separable into a product of one-dimensional Newton-Cotes interpolation polynomials. There are two known sets of multi-dimensional functions in two different geometric regions having this property. In N-dimensional cubes, the product separable multi-dimensional functions are

$$b_{i_1 \ldots i_N} = \prod_{k=1}^{N} l_{i_k}^{(n_k)}(x_k) \quad , i_k = 0, \ldots, n_k \qquad (3.5.1)$$

where the $\{x_k\}$ are N independent variables and the $\{l_i^{(n)}\}$ are, as before, the equi-spaced Lagrangean interpolation polynomials

$$l_i^{(n)}(x) = \frac{(-1)^{n-1}}{i!(n-i)!} \prod_{\substack{m=0 \\ m \neq i}}^{n} \left(\frac{n}{L}x - m\right) \qquad (3.5.2)$$

$$l^{(0)}(x) = 1$$

In N-dimensional simplexes, the required functions are given by [45]

$$b_{i_1 \ldots i_N} = \prod_{k=1}^{N+1} l_{i_k}^{(i_k)}(w_k) \qquad i_k = 0, \ldots, n \qquad (3.5.3)$$

with

$$\sum_{k=1}^{N+1} i_k = n \qquad (3.5.4)$$

Here $\{w_k\}$ are homogeneous coordinates [45]. The spaces generated by the functions (3.5.1) will be called $\Phi^{(n_1 \ldots n_N)}$ and those generated by (3.5.3) $\psi^{(n)}$. If N=1, then the two spaces coincide

$$l_i^{(n)}(x) = l_i^{(i)}(x)\, l_{n-i}^{(n-i)}(L - x) \qquad (3.5.5)$$

since the one-dimensional form of both the cube and of the simplex is a line segment.

Of the two spaces Φ and ψ, only ψ contains polynomials of degree n which are complete in every monomial of x_k of degree less or equal to n. (This is not to be confused with completeness in the sense of a Hilbert space; both (3.5.1) and (3.5.3) form a complete set of basis functions since any polynomial in x_k may be obtained by taking n_k large enough). However, in the functions (3.5.3), products of polynomials of different degree are unavoidable; with (3.5.1) all the n_k may be taken to be equal. This, plus the fact that the independent variables in a differential equation must be transformed into homogeneous

coordinates when using (3.5.3), make the cubic basis functions the easier of the two to work with.

The ability of ordinary elementary matrices to generate derivatives, projections and function multiplications on the functions (3.5.1) and (3.5.3) is a result of both of these function sets defining an equi-spaced, right-angled grid of interpolation nodes in their respective coordinate systems. On this grid of points may be superimposed the coefficients of the expansion of a function in these bases as shown in Figure 3.5.1 for a three-dimensional space. The effect of an elementary operator acting on this function will be equivalent to a matrix multiplication of this cubic block of coefficients by the elementary matrices in the proper direction. Thus, if a function $y(x_1,\ldots,x_N)$ is expanded in either the functions (3.5.1) or (3.5.3)

$$y(x_1,\ldots,x_N) = \prod_{k=1}^{N} \sum_{i_k=1}^{n_k} y_{i_1\ldots i_N} b_{i_1\ldots i_N} \qquad (3.5.6)$$

then the elements of the matrix equivalent of the equation

$$z(x_1,\ldots,x_N) = E(x_p)\, y(x_1,\ldots,x_N) \qquad (3.5.7)$$

are

$$Z_{i_1\ldots i_p\ldots i_N} = \sum_{q=1}^{n_p} E_{i_p q}\, y_{i_1\ldots i_{p-1} q i_{p+1}\ldots i_N} \qquad (3.5.8)$$

where E is the n_p'th order elementary matrix of $E(x_p)$. Here $E(x_p)$ may act only on the variable x_p.

For the case N=2, equation (3.5.8) provides a particularly

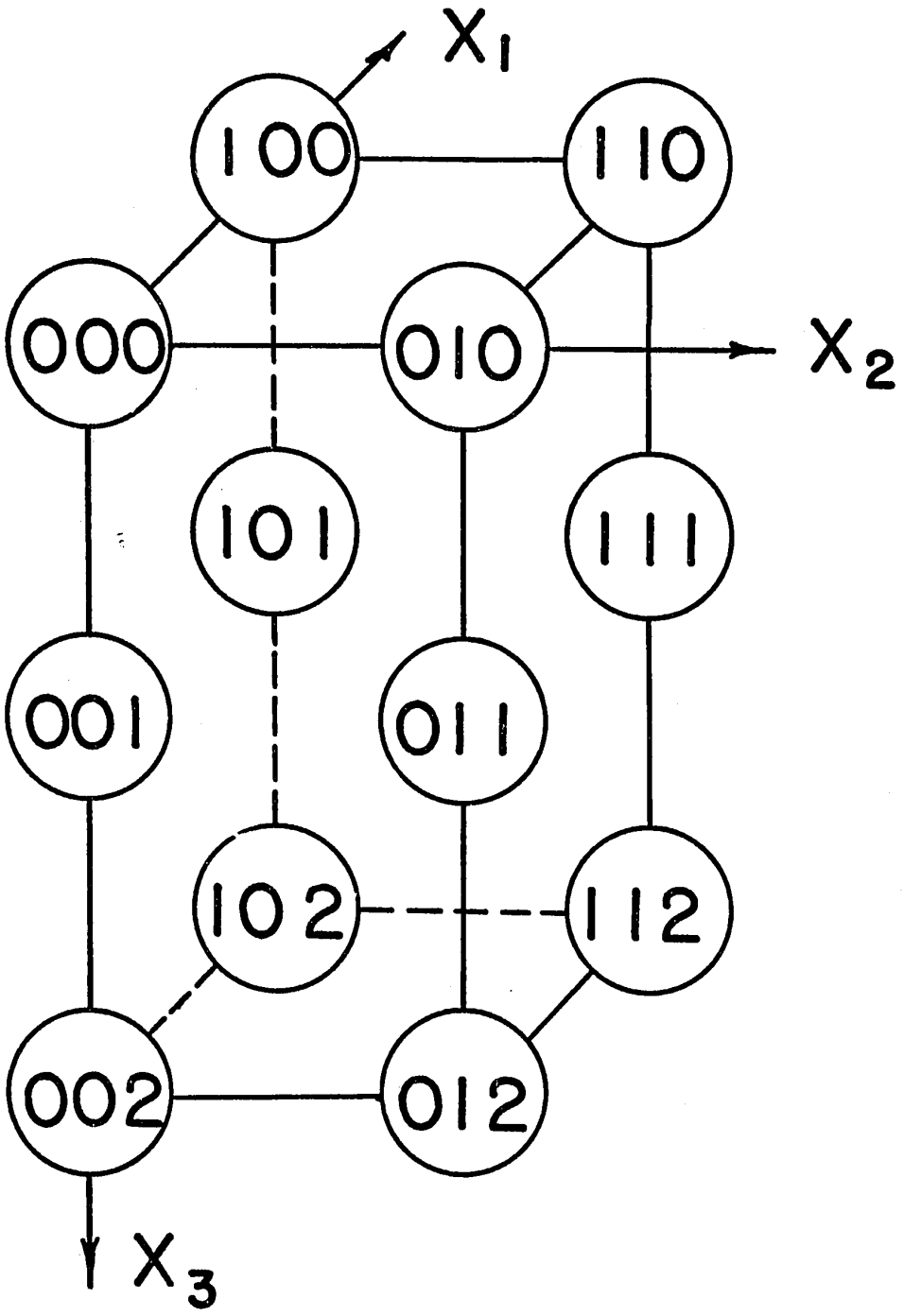


Figure 3.5.1

Nodal points where $b_{i_1 i_2 i_3} = 0$ if $n_1 = 1$, $n_2 = 1$ and $n_3 = 2$. The numbers in the circles designate the values of i_1, i_2 and i_3.

useful result. In this case for an elementary operator acting purely in the direction x_1, the matrix elements Z become

$$Z_{ij} = \sum_{q=1}^{n_1} E_{iq}\, y_{qj} \tag{3.5.9}$$

and for an elementary operator acting in the direction x_2

$$Z_{ij} = \sum_{q=1}^{n_2} E_{jq}\, y_{iq} = \sum_{q=1}^{n_2} y_{iq}\, E^T_{qj} \tag{3.5.10}$$

where E^T is the transpose of the matrix E. Therefore, in discretizing equations in two dimensional spaces, operators acting in the direction x_1 result in multiplication of the coefficient matrix from the left by the matrix equivalent of the operator, and operators acting in the direction x_2 result in multiplication of the coefficient matrix from the right by the transpose of the matrix equivalent of the operator.

Now, any partial differential equation can be reduced to an ordinary differential equation by holding all but one of the independent variables constant. Thus, along the line

$$x_k = a_k\,, \quad k = 1,\ldots,i-1,i+1,\ldots,N \tag{3.5.11}$$

the partial differential equation

$$D(x_1,\ldots,x_N)\;\; y(x_1,\ldots,x_N) = f(x_1,\ldots,x_N) \tag{3.5.12}$$

reduces to the ordinary differential equation

$$D(a_1,\ldots,x_i,\ldots,a_N)\;\; y(a_1,\ldots,x_i,\ldots,a_N) = f(a_1,\ldots,x_i,\ldots,a_N) \tag{3.5.13}$$

The approximate range space $\Sigma^{(0)}$ can be easily determined for this equation and will be called $\Sigma_i^{(0)}$. An N-dimensional partial differential equation will generate N such spaces $\Sigma_i^{(0)}$ and, since the partial differential equation (3.5.12) is a combination of the N equations (5.3.13), the approximation range space $\Sigma^{(0)}$ of a partial differential equation must coincide with the intersection of the range spaces $\Sigma_i^{(0)}$

$$\Sigma^{(0)} = \bigcap_{i=1}^{N} \Sigma_i^{(0)} \qquad (5.3.14)$$

An example will help to fix these ideas. Consider the two dimensional Helmholtz equation in Cartesian coordinates

$$\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = k^2 z \qquad (3.5.15)$$

and let the domain of the operator $(\nabla^2 - k^2)$ be approximated by the spaces $\Phi^{(2,2)}$. Then z may be expanded in the functions (3.5.1)

$$z(x,y) = \sum_{i,j=o}^{2} Z_{ij}\, l_i^{(2)}(x)\, l_j^{(2)}(y) \qquad (3.5.16)$$

The first simple operator $\frac{\partial^2}{\partial x^2}$ in (3.5.15) acting on this space gives

$$\frac{\partial^2 z}{\partial x^2} = \sum_{i,j=o}^{2} Z_{ij}\, l_j^{(2)}(y)\, \frac{\partial^2}{\partial x^2}\, l_i^{(2)}(x) \qquad (3.5.17)$$

$$= \sum_{k=o}^{0} \sum_{j=o}^{2} W_{kj}\, l_j^{(2)}(y)\, l_k^{(0)}(x) = w(x,y)$$

Since $w(x,y) \in \Phi^{(0,2)}$, the space $\Sigma^{(1)}$ of equation (3.1.3) is $\Phi^{(0,2)}$. Similarly, if $\Sigma^{(f)} = \Phi^{(2,2)}$, the range spaces of the second and third simple operators, $\frac{\partial^2}{\partial y^2}$ and $-k^2$, in (3.5.15) are $\Sigma^{(2)} = \Phi^{(2,0)}$ and $\Sigma^{(3)} = \Phi^{(2,2)}$

$$\Sigma^{(0)} = \Sigma^{(1)} \cap \Sigma^{(2)} \cap \Sigma^{(3)} = \Phi^{(0,0)} \tag{3.5.18}$$

Therefore, according to equation (5.3.14), the solution of (3.5.15) in the subspace $\Phi^{(2,2)}$ is given by the solution of the equation

$$P_{(o,o),\ (o,2)} \frac{\partial^2 z}{\partial x^2} + P_{(o,o),\ (2,o)} \frac{\partial^2 z}{\partial y^2} = k^2 P_{(o,o),\ (2,2)} z \tag{3.5.19}$$

Now $P_{(o,o),\ (o,2)}$ is a function of y only since it must leave functions in x intact. Similarly, $P_{(o,o),\ (2,o)}$ is only a function of x. Consequently, according to the methods of section 3.2 and equations (3.5.9) and (3.5.10), the discretized form of (3.5.15) is

$$DZP^T + PZD^T = k^2 PZP^T \tag{3.5.20}$$

where

$$D = \frac{4}{L^2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \tag{3.5.21a}$$

$$P = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 \end{bmatrix} \tag{3.5.21b}$$

$$Z = \begin{array}{c} \\ x \downarrow \end{array} \overset{\longrightarrow y}{\begin{bmatrix} z_{11} & z_{12} & z_{13} \\ z_{21} & z_{22} & z_{23} \\ z_{31} & z_{32} & z_{33} \end{bmatrix}} \tag{3.5.21c}$$

The x and y coordinate axes have been drawn on equation (3.5.21c) to indicate how the coefficients are related to the coordinates of the corresponding interpolation nodes.

Of course, many multi-dimensional functions cannot be separated into a product of functions of one independent variable. For these functions multi-dimensional function matrices are defined by

$$F_{i_1 \ldots i_N} = f\ (x_{i_1}^{(1)}, \ldots, x_{i_N}^{(N)}) \tag{3.5.22}$$

The function muliplication $z = fy$ is then given by

$$Z_{i_1 \ldots i_N} = F_{i_1 \ldots i_N}\ y_{i_1 \ldots i_N} \tag{3.5.23}$$

In order to facilitate the evaluation of the matrix product (3.5.23), and also the solution of equations such as (3.5.20), it is necessary to introduce the Kronecker product of matrices. This is done as follows. Let A be an m x n matrix and B be an s x t matrix. The Kronecker product of A and B is defined to be the ms x nt matrix [46]

$$A \otimes B = \begin{bmatrix} A_{11}B & \circ\ \circ\ \circ & A_{1n}B \\ \circ & & \circ \\ \circ & & \circ \\ \circ & & \circ \\ A_{m_1}B & \circ\ \circ\ \circ & A_{mn}B \end{bmatrix} \tag{3.5.24}$$

It is known to obey the following associative and distributive laws [46]

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD) \qquad (3.5.25a)$$

$$(A + B) \otimes (C + D) = A \otimes C + A \otimes D + B \otimes C + B \otimes D \qquad (3.5.25b)$$

In addition, let A_j denote the j'th column of A and define the matrix operation vec A to give the following mn component vector [47]

$$\text{vec } A = \begin{bmatrix} A_1 \\ \circ \\ \circ \\ \circ \\ A_n \end{bmatrix} \qquad (3.5.26)$$

The application of Kronecker products to matrix equations of the form (3.5.20) is a result of the following [47]

$$\text{vec } (AB) = (I_p \otimes A) \text{ vec } B = (B^T \otimes I_m) \text{ vec } A \qquad (3.5.27)$$

By using this equation and property (3.5.25a), an equation of the form

$$\sum_k A_k Z_k B_k^T = F \qquad (3.5.28)$$

where the A_k are mxn matrices, Z is an $n$xt matrix of unknown coefficients, the B_k are $s$xt matrices and F is a specified mxs matrix, yields

$$\sum_k (B_k \otimes A_k) \text{ vec } Z = \text{vec } F \qquad (3.5.29)$$

where the $B_k \otimes A_k$ are ms x nt matrices, vec Z is an nt component vector and vec F is an ms component vector. Equation (3.5.29) is a standard rectangular matrix equation in terms

of the unknown coefficients Z_{ij}.

In terms of the Helmholtz equation (3.5.15), the above result means that the discretized form of the equation (3.5.20) may be written as

$$(P \otimes D + D \otimes P) \text{ vec } Z = k^2 \ (P \otimes P) \text{ vec } Z \qquad (3.5.30)$$

From (3.5.25b) and the identity

$$(A + B)^2 = A^2 + A \otimes B + B \otimes A + B^2 \qquad (3.5.31)$$

equation (3.5.30) may be written in the symmetric forms

$$[\ (D + P)^2 - D^2] \text{ vec } Z = (1 + k^2)\ P^2 \text{ vec } Z \qquad (3.5.32a)$$

$$[\ (D + P)^2 + (D - P)^2\ ] \text{vec } Z = 2k^2 P^2 \text{ vec } Z \qquad (3.5.32b)$$

These results are valid for all D and P matrices of second degree. If a biquadratic approximation is used for the unknown, equation (3.5.30) becomes

$$\frac{4}{3L^2} [\ 1\ 1\ 1\ 1\ -8\ 1\ 1\ 1\ 1\ ] \text{ vec } Z$$

$$= \frac{k^2}{36} [\ 1\ 4\ 1\ 4\ 16\ 4\ 1\ 4\ 1\ ] \text{ vec } Z \qquad (3.5.33)$$

For two-dimensional matrix products of the form (3.5.23), equation (3.5.27) implies that

$$\text{vec } Z = \text{vec } (FY) = (F^T \otimes I_N) \text{ vec } Y \qquad (3.5.34)$$

Therefore, as in the one-dimensional case, the function matrix for two-dimensional problems is a diagonal matrix of the two interpolation point values of the multiplying function.

CHAPTER IV

DISCRETIZATION OF ARBITRARY DIFFERENTIAL EQUATIONS

In the numerical solution of differential equations, the step that has always been the most laborious and time consuming has been the discretization process. Aside from a single known attempt at automation [48], the usual procedure has been to churn through either Taylor series expansions or tables of integration formulae by hand or in sophisticated cases with the aid of computational machinery until the coefficients in the discretization process were evaluated for the particular differential equation of interest.

The most automatic discretization procedure presently available is the Taylor series method of producing finite difference equations. In this method [49], the differential terms in the Taylor series expansion of the solution of a differential equation are matched to the components of the differential operator on an arbitrary set of points. Since a general expression for the Taylor series expansion of an arbitrary function is known, the procedure does not require algebraic manipulation once the point set is chosen.

Silvester has reported a fully automatic computer program for discretizing arbitrary differential operators by this method [48]. However, since its discretization accuracy depends upon the locations of the point set, and since there is no a priori way of determining good locations, this "automatic" procedure is a hit or miss approach.

Using the elementary matrix concept of Chapter 2, the automatic discretization of any differential equation can be accomplished in a direct manner. First, it is necessary to break the operator into its elementary analytic factors and to form the corresponding elementary matrices. The assembly of these matrices to form simple operators is performed according to (3.2.27) by multiplying the elementary matrices together, keeping in mind that the degree of the range space of each operator must coincide with the degree of the domain of the next elementary operator acting upon it. Provided that the simple matrices are augmented by the proper projection matrices, the compound operator is assembled according to (3.2.4) by adding the matrices corresponding to the simple operators. Finally, the forcing function in equation (3.2.4) is added by evaluating the analytic function on the interpolation point set for the space $\Sigma^{(o)}$ as given by (3.2.22) and (3.4.6). Thus, the method of assembling elementary matrix factors to form a discrete matrix equation is completely analogous to the mathematical operations performed on the parent operators in the analytic differential equation.

This chapter contains the steps required to convert this procedure into a direct numerical algorithm for the automatic discretization of arbitrary differential equations. The discussion is complemented by listings of Fortran subroutines for the generation of matrix equivalents of ordinary and two-dimensional partial differential equations. The aim of these programs is to provide a very simple, accurate, and efficient

method of discretizing differential equations for anyone having access to a Fortran compiler.

4.1 COORDINATE-INDEPENDENT FACTORS

Since discretized equations are often used repeatedly in different locations, a major practical consideration in developing a discretization algorithm is to separate the coordinate-dependent factors from the coordinate-independent ones. Of the three elementary matrices, only the function matrix is entirely dependent on coordinate information, although the differentiation matrix does depend on the length of the interval taken. This suggests that the discretization process should be pursued on two levels:the high-order differentiation and projection matrices should be evaluated and stored separately from the function matrices, which should be generated only when needed in a specific location.

In order to avoid the time-consuming and involved computations required for generating the elementary D, P and FU matrices, it is natural to store the numbers in computer programs which use them. Figure 4.1.1 contains one method for storing these matrices. It is a Fortran block data subprogram in which the matrix elements given in Chapter 2 are stored in hexadecimal units for a 32-bit word computer. Due to the symmetry properties of these matrices, only the top halves of even rowed matrices and the top halves plus one line of odd rowed matrices are stored. This requires 535 memory locations for all matrices corresponding to the polynomials of degrees one through nine.

Figure 4.1.1a

```
      BLOCK DATA                                                                DECL9315
C                                                                               DECL9320
C     THIS SUBPROGRAM CONTAINS THE ELEMENTARY D, P AND FU MATRICES OF           DECL9325
C     FIRST TO NINTH ORDER. THE NUMBERS ARE STORED IN HEXADECIMAL               DECL9330
C     FORM UNDER THE APPROPRIATE MATRIX NAME. THE ELEMENTARY MATRICES           DECL9335
C     ARE PLACED IN A COMMON BLOCK CALLED 'DPFU1' 544 WORDS LONG. THE           DECL9340
C     ARRAY 'NMA' CONTAINS THE LOCATIONS MINUS ONE OF THE  FIRST                DECL9345
C     ELEMENT OF EACH OF THE P MATRICES.                                        DECL9350
C                                                                               DECL9355
      DIMENSION NMA(9)                                                          DECL9360
      DIMENSION P1(2),D1(2),   P2(3),D2(3),   P3(8),D3(8),                      DECL9365
     *   P4(10),D4(10),   P5(18),D5(18),   P6(21),D6(21),                       DECL9370
     *   P7(32),D7(32),   P8(36),D8(36),   P9(50),D9(50)                        DECL9375
      DIMENSION FU0(1),FU1(4),FU2(6),FU3(12),FU4(15),FU5(24),FU6(28),           DECL9380
     =          FU7(40),FU8(45)                                                 DECL9385
      COMMON /DPFU1/ NMA,P1,D1,FU0, P2,D2,FU1, P3,D3,FU2, P4,D4,FU3,            DECL9390
     =    P5,D5,FU4, P6,D6,FU5, P7,D7,FU6, P8,D8,FU7, P9,D9,FU8                  DECL9395
      DATA NMA/0,5,15,37,69,129,186,278,390/                                    DECL9400
      DATA    P1 /Z40800000, Z40800000/                                         DECL9405
      DATA    D1 /ZC1100000, Z41100000/                                         DECL9410
      DATA   FU0 /Z41100000/                                                    DECL9415
      DATA    P2 /Z40AAAA9C, Z40AAAA9C, ZC0555551/                              DECL9420
      DATA    D2 /ZC1300000, Z41400000, ZC1100000/                              DECL9425
      DATA   FU1 /Z41100000, Z40800000, Z00000000, Z40800000/                   DECL9430
      DATA    P3 /Z40C66667, ZC0100000, Z40ACCCC3, Z408FFFF9, ZC0ACCCCA,DECL9435
     1 Z408FFFF9, Z40399999, ZC0100000/                                         DECL9440
      DATA    D3 /ZC1580000, Z40200000, Z41900000, ZC1360000, ZC1480000,DECL9445
     1 Z41380000, Z41100000, ZC0200000/                                         DECL9450
      DATA   FU2 /Z41100000, Z4038E38E, Z00000000, Z40E38E38, Z00000000,DECL9455
     1 ZC0107107/                                                               DECL9460
      DATA    P4 /Z40D8F99E, ZBFB043F7, Z409C09CE, Z40AA7C69, ZC0EA0E95,DECL9465
     1 Z4078B7B3, Z409C09C4, ZC01FC917, ZC0270275, Z3F4C9303/                   DECL9470
      DATA    D4 /ZC1855555, ZC0161F9A, Z42100000, ZC148B0FC, ZC1C00000,DECL9475
     1 Z41578425, Z41555555, ZC0FCD6E9, ZC1100000, Z4022C3F3/                   DECL9480
      DATA   FU3 /Z41100000, Z401E0000, ZC0100000, Z00000000, Z4110E000,DECL9485
     1 Z4090C000, Z00000000, ZC0360000, Z40900000, Z00000000, Z3FA00000,DECL9490
     2 ZC0104000/                                                               DECL9495
      DATA    P5 /Z40E58B7C, ZBF99877C, Z3F300007, Z40844670, Z40C05A42,DECL9500
     1 ZC018FFFB, ZC110B8CD, Z40608B53, Z4095FFF3, Z411088CB, ZC01BCB55,DECL9505
     2 Z4093FFF4, ZC084466F, Z3F485A99, ZC018FFFE, Z401A747A, ZBF788D00,DECL9510
     3 Z3F300010/                                                               DECL9515
      DATA    D5 /ZC1B6AAAA, ZC034E000, ZBF600000, Z42190000, ZC161DA00,DECL9520
     1 Z40535555, ZC2190000, Z41874955, ZC15DC000, Z4210AAAA, ZC12DB400,DECL9525
     2 Z4158C000, ZC1640000, Z4007A000, ZC0535555, Z41100000, ZC01E7555,DECL9530
     3 Z3F600000/                                                               DECL9535
      DATA   FU4 /Z41100000, Z401205BC, ZBF89A027, Z00000000, Z411205BC,DECL9540
     1 Z4058C01A, Z00000000, ZC04816F0, Z40CE703A, Z00000000, Z4C1A36E2,DECL9545
     2 ZC0275254, Z00000000, ZBF4816F0, Z3F5BC01A/                              DECL9550
      DATA    P6 /Z40EE03D8, ZBF651C8A, Z3F33C575, Z406BB824, Z40CECF5A,DECL9555
     1 ZC01F14FC, ZC110B4C6, Z4047531E, Z40B6B883, Z411671O5, ZBF7EFC6E,DECL9560
     2 Z4072B2C8, ZC110B4C4, ZBFC89A67, ZC0128E71, Z40688826, Z3F8A10A7,DECL9565
     3 Z3F13-BDC, ZC011F408, ZBF1C840A, Z3E40FC00/                              DECL9570
      DATA    D6 /ZC1EB3333, ZC04A3C64, Z3F2CB20F, Z42240000, ZC17F24D8,DECL9575
     1 Z3F429F6F, ZC2200000, Z41C452E7, ZC16E15CA, Z42280000, ZC1622E5D,DECL9580
     2 Z417DC9C4, ZC2168000, Z412CD5F9, ZC113A732, Z41733333, ZC CDD33,DECL9585
     3 Z403F1C75, ZC1100000, Z401A3B01, ZBF655C65/                              DECL9590
      DATA   FU5 /Z41100000, Z3FBDBD47, Z8F4EA943, Z3F3000000, ZC 000000,      DECL9595
     1 Z4112877B, Z403D743C, ZC0190000, ZC 000000, ZC0548475, Z4 F8C7F1,DECL9600
     2 Z40960000, Z00000000, Z402D9C44, ZC 3D743C, Z4096C000, ZC 000000,DECL9605
     3 ZBFF9A535, Z40118EEC, ZC0190000, ZC 000000, Z3F25F247, ZBF2F3A4A,DECL9610
     4 Z3F300000/                                                               DECL9615
      DATA    P7 /Z40F3CFA7, ZBF6E488F, Z3 33E5E, ZBE9FFD1C, Z4 E5E111,DECL9620
     1 Z40D6979F, ZC01C2E95, Z3F620011, ZC 4F512, Z40361D60, Z4 D2E82F,DECL9625
     2 ZC01E4005, Z411AA9B6, Z3FE69CCD, Z4 6E640, Z40991FF8, ZC1 A93E4,DECL9630
     3 ZC02D8C14, Z3FEB961B, Z40991FF5, Z4 9FF4DC, Z40244821, ZC1 573FE,DECL9635
     4 ZC01EA002, ZC05551AB, ZBF0D54AD, Z3F9101A3, Z3F620000, Z3 C83FC,DECL9640
     5 Z3F223DB2, ZBF173656, ZBE9FFFE0/                                         DECL9645
      DATA    D7 /ZC2122666, ZC05A725A, Z3F65D2CD, Z3F1400000, Z4 31000,DECL9650
     1 ZC19FA887, ZC02DC443, ZC0112666, Z4298000, Z421DEF0F, ZC 83 ,DECL9655
     2 Z408EEAAA, Z42514AAA, ZC1840946, Z41A61265, ZC185F000, ZC 3 ,DECL9660
     3 Z416FAFC5, ZC12C923D, Z4185F000, Z4210A666, ZC130F200, Z4 E4C1E,DECL9665
     4 ZC08EEAAA, ZC182AAAA, Z40CAAADA, ZC03576A2, Z40116666, Z4 000,DECL9670
     5 ZC0170AE3, Z3F5B857D, ZBF140000/                                         DECL9675
      DATA   FU6 /Z41100000, Z3F854600, ZBF300897, Z3F1A048C, ZC 000000,DECL9680
     1 Z41120DD8, Z402B30A2, ZBFFFAF3A, Z00000000, ZC05D538, Z41 E41E,DECL9685
     2 Z406DUD48, Z00000000, Z4042A300, ZC0501352, Z40C35085, ZC 000000,DECL9690
     3 ZC022135A, Z40210326, ZC023F21E, Z00000000, Z3FA57162, ZBF966718,DECL9695
     4 Z3FA57162, Z00000000, ZBF163655, Z3F133706, ZBF13F80D/                   DECL9700
      DATA    P8 /Z40F7B919, ZBF57BCA8, Z3F2A2F38, ZBF12E894, Z4 42385,DECL9705
     1 Z40D9B1D1, ZC01A6838, Z3FA43953, ZC0E7CC09, Z40314FC, Z4 E3B ,DECL9710
     2 ZC02DC573, Z411CF7FD, Z40144871, Z401F4834, Z40C87685, ZC 2435F,DECL9715
     3 ZC04FA729, Z403E3F4E, Z405ED658, Z411CF7FB, Z40515F85, ZC 4722 ,DECL9720
     4 Z3E3A0500, ZC0E7BF9A, ZC02E2D21, Z4028700C1, ZBF891840, Z4 43B ,DECL9725
     5 Z3FE30D40, ZBFC60F7E, Z3F382D3F, ZBF846DE0, ZBF1DC384, Z3F190F A,DECL9730
     6 Z8E837C50/                                                               DECL9735
      DATA    D8 /ZC215BE2B, ZC067618F, Z3F7E0E27, Z8E7CF4CC, Z4 C ,DECL9740
     1 ZC1C20E23, ZC05204FB, Z3F1A3CF0, ZC270F000, Z42167856, ZC 8 F ,DECL9745
     2 Z402D63EE, Z42955555, ZC212A441, Z41D7FC40, ZC195FF51, ZC 8C ,DECL9750
     3 Z41EA2686, ZC152E882, Z41456073, Z42599999, ZC188E365, Z4 25F ,DECL9755
     4 ZC117E5A5, ZC2255555, Z41359EEC, ZC0D35090, Z4052E668, Z4 924 ,DECL9760
     5 ZC0C9E924, Z402E31B3, ZC0103DC7, ZC1100000, Z40157483, ZB 4A ,DECL9765
     6 Z3F174CCF/                                                               DECL9770
      DATA   FU7 /Z41100000, Z3F624558, ZBF1F1400, Z3EEECC80, ZB C ,DECL9775
     1 Z00000000, Z4112CF45, Z401FB9C0, ZBFA8C438, Z3F620000, Z 000 ,DECL9780
     2 ZC0645174, Z4111D87C, Z4052460E, ZC01EA000, Z00000000, Z4 8E ,DECL9785
     3 ZC05F2D40, Z40E48986, Z40992000, Z00000000, ZC03C30F, Z4 4E ,DECL9790
     4 ZC03E941A, Z40992000, Z00000000, Z40165C08, ZC015F6C0, Z4 5F ,DECL9795
     5 ZC01E4000, Z00000000, ZBF757218, Z3F599400, ZBF514208, ZB 20 ,DECL9800
     6 Z00000000, Z3EEC9E80, ZBEA5C000, Z3E8F4780, ZBEA0C000/                   DECL9805
      DATA    P9 /Z40FA60DA, ZBF43E654, Z3F20E571, ZBF13607D, ZB 2F ,DECL9810
     1 Z403297C1, Z40DA2401, ZC0167732, Z3FBB94F8, ZBF19514 , ZC 5 ,DECL9815
     2 Z4034E0CB, Z40E873E2, ZC036FA58, Z3F80C1D5, Z411D832E, Z4 49 ,DECL9820
     3 Z3FC20DFD, Z40F7B4F3, ZC0227424, ZC12C48BF, ZC0609515, Z4 64 ,DECL9825
     4 Z4016D87E, Z409B0A1F, Z412C48BF, Z40806189, ZC0650C0C, Z4 44 ,DECL9830
     5 Z409509D5, ZC11D8327, ZC062171F, Z4062C65F, ZC03C98F, ZC 27 ,DECL9835
     6 Z40CA5E8E, Z402D92A8, ZC02CD562, Z4010B0EF, Z3F8D1040, ZC 29 ,DECL9840
     7 Z8FC05020, Z3FB9D880, ZBF784F04, ZBF1950C0, Z3F59F160, Z8 3B ,DECL9845
     8 ZBF15298C, Z3ED01C56, Z3E22FD60/                                         DECL9850
```

Figure 4.1.1b

```
 DATA    D9 /ZC21975F1, ZC0720B5E, Z3F893B32, ZBF112A5B, ZBF460000,DECL9855
1 Z42510000, ZC1E86924, ZC06E6DB4, Z3FA9A16E, Z3F4116DB, ZC2A20000,DECL9860
2 Z421CE1EB, ZC1B6CAF8, ZC0145A50, ZC01FE4CC, Z42FC0000, ZC21CC029,DECL9865
3 Z42111B77, ZC1A8417A, Z40CEB800, ZC3118B00, Z421B526D, ZC189C44D,DECL9870
4 Z41CB43C7, ZC1AE6B40, Z42E2CCCC, ZC213FB8E, Z41510A85, ZC12EB26B,DECL9875
5 Z41AE6B40, ZC27E0000, Z41A7FB9C, ZC1261B96, Z41102DA5, ZC0CEB800,DECL9880
6 Z422E4924, ZC13B6673, Z40C98ECE, ZC04972A9, Z401FE4CC, ZC1A20000,DECL9885
7 Z40CA3876, ZC028FCD8, Z3FDA0439, ZBF4116DB, Z41100000, ZC0138E70,DECL9890
8 Z3F3D643B, ZBF133239, Z3E460000/                                 DECL9895
 DATA    FU8 /Z41100000, Z3F49376D, ZBF151510, Z3E904812, ZBE563CA2,DECL9900
1 Z00000000, Z4112C0DB, Z40181812, ZBF736CDB, Z3F3BFDA8, Z00000000,DECL9905
2 ZC0694D99, Z4112726E, Z403F1F87, ZC0158F28, Z00000000, Z4C6ED86B,DECL9910
3 ZC06B53F5, Z40FC7E1F, Z4078BB4A, Z00000000, ZC05E0548, Z4049C9B8,DECL9915
4 ZC04EE769, Z40BCA4A3, Z00000000, Z4038EBAC, ZC02885EA, Z4C241204,DECL9920
5 ZC02E6F6B, Z00000000, ZC016E458, Z3FF88C89, ZBFC9FE7F, Z3FDB8312,DECL9925
6 Z00000000, Z3F578650, ZBF396A62, Z3F2C64F1, ZBF2C8264, Z00000000,DECL9930
7 ZBE966EDA, Z3E60604A, ZBE482409, Z3E44FD4E/                      DECL9935
 END                                                               DECL9940
```

A program for building up matrix operators of an arbitrary order from the block data subprogram is given in Figure 4.1.2. Written in Fortran, the program is in the form of a subroutine called OPRATR, the input to which is an NC by ND array in OP which may contain any elementary matrix, including the identity matrix. It returns an NR by ND array in OP which is equal to the product of the original matrix times (NC - NR) successive elementary matrices. The type of matrix used depends upon the value of KIND; if KIND = 0, differentiation matrices are used; KIND = 1 results in the application of projection matrices; KIND = 2, the FU matrices. Thus, if on input OP is a 6×6 identity matrix, NR is equal to 3 and KIND equal to 0, on output OP would contain the 3 x 6 matrix which differentiates the coefficients of fifth order polynomials into second order ones.

The subroutine OPRATR uses the symmetry or antisymmetry properties of the elementary matrices to speed up the computation. This means that an additional index called IEO must be supplied to OPRATR to designate whether the input matrix is symmetric (IEO =1) or antisymmetric (IEO = -1) about the matrix centroid.

In order to use the subroutine OPRATR to generate the coordinate-independent matrices for ordinary differential equations automatically, a method of separating these operators in the differential equation must be devised. On a computer, one method of distinguishing the mathematical components of a differential equation is to assign each mathematical entity in the equation a numerical code. If the code is chosen cleverly,

Figure 4.1.2

```
      SUBROUTINE DPRATR(DP,IED,NR,NC,ND,KIND)                           DECL7525
C                                                                       DECL7530
C     DPRATR BUILDS UP ELEMENTARY DIFFERENTIATION AND PROJECTION        DECL7535
C     MATRICES FROM VALUES STORED IN THE COMMON BLOCK DPFU1. THE        DECL7540
C     INPUT TO DPRATR MUST BE AN 'NC' BY 'ND' ARRAY IN 'DP' WHICH IS    DECL7545
C     CONVERTED TO AN 'NR' BY 'ND' ARRAY BY THE PRE-MULTIPLICATION OF   DECL7550
C     'DP' WITH THE APPROPRIATE ELEMENTARY MATRIX. (KIND = 0:           DECL7555
C     PROJECTION; KIND = 1: DIFFERENTIATION; KIND = 2: RAISING)         DECL7560
C                                                                       DECL7565
      DOUBLE PRECISION VECTOR(5)                                        DECL7570
      DIMENSION DP(1)                                                   DECL7575
      DIMENSION NMA(9),D(535)                                           DECL7580
      COMMON / DPFU1 / NMA,D                                            DECL7585
      IF(NR .EQ. NC) RETURN                                             DECL7590
      NSIGN = 1                                                         DECL7595
      IF(KIND .EQ. 1) NSIGN = - 1                                       DECL7600
      M = IABS(NC - NR)                                                 DECL7605
      NR = NC                                                           DECL7610
      DO 6 N = 1,M                                                      DECL7615
      NC = NR                                                           DECL7620
C                                                                       DECL7625
C     IADD IS THE LOCATION OF THE ELEMENT BEFORE THE FIRST ELEMENT      DECL7630
C     OF THE REQUIRED ELEMENTARY MATRIX IN DPFU1                        DECL7635
C                                                                       DECL7640
      IF(KIND .EQ. 2) GO TO 3                                           DECL7645
      NR = NR - 1                                                       DECL7650
      ILIMIT = (NR + 1)/2                                               DECL7655
      IADD = NMA(NR) + ILIMIT*NC*KIND                                   DECL7660
      GO TO 4                                                           DECL7665
    3 NR = NR + 1                                                       DECL7670
      ILIMIT = (NR + 1)/2                                               DECL7675
      IADD = NMA(NC) + (NR/2)*NR*2                                      DECL7680
    4 J1NC = - NC                                                       DECL7685
      J1NR = - NR                                                       DECL7690
C                                                                       DECL7695
C     THE ARRAY 'DP' IS MULTIPLIED BY ONE HALF OF THE ELEMENTARY        DECL7700
C     MATRIX                                                            DECL7705
C                                                                       DECL7710
      DO 2 J = 1,ND                                                     DECL7715
      J1NC = J1NC + NC                                                  DECL7720
      J1NR = J1NR + NR                                                  DECL7725
      DO 1 I = 1,ILIMIT                                                 DECL7730
      VECTOR(I) = 0.D0                                                  DECL7735
      IK = I + IADD - ILIMIT                                            DECL7740
      DO 1 K = 1,NC                                                     DECL7745
      IK = IK + ILIMIT                                                  DECL7750
    1 VECTOR(I) = VECTOR(I) + DBLE(D(IK)*DP(K + J1NC))                  DECL7755
      DO 2 I = 1,ILIMIT                                                 DECL7760
    2 DP(I + J1NR) = SNGL(VECTOR(I))                                    DECL7765
      IF(ILIMIT .GE. NR) GO TO 6                                        DECL7770
C                                                                       DECL7775
C     THE SECOND HALF OF 'DP' IS SET EQUAL TO + OR - TIMES THE FIRST    DECL7780
C     HALF                                                              DECL7785
C                                                                       DECL7790
      IED = NSIGN*IED                                                   DECL7795
      SIGN = FLOAT(IED)                                                 DECL7800
      J1NR = - NR                                                       DECL7805
      J1NC = NR*(ND + 1) + 1                                            DECL7810
      DO 5 J = 1,ND                                                     DECL7815
      J1NR = J1NR + NR                                                  DECL7820
      J1NC = J1NC - NR                                                  DECL7825
      DO 5 I = 1,ILIMIT                                                 DECL7830
    5 DP(J1NC - I) = SIGN*DP(I + J1NR)                                  DECL7835
    6 CONTINUE                                                          DECL7840
      RETURN                                                            DECL7845
      END                                                               DECL7850
```

then certain ranges of values will indicate the proper algebraic steps to be followed. Consider, for example, the following coding procedure

NUMBER	OPERATION
3000	end
2000	minus sign
1000	plus sign
200	y (the unknown function)
-1	$\frac{d}{dx}$
∘	∘
∘	∘
∘	∘
-9	$\frac{d^9}{dx^9}$

With this code, a number greater than or equal to 1000 designates the separation locations for simple operators, the presence of the number 200 in a simple operator indicates that the term contains the unknown while its absence signifies that the quantity is a known function, and the absolute values of the numbers from -1 to -9 indicate the corresponding orders of differentiation. Furthermore, the number of derivatives in a simple operator may be obtained by adding the magnitudes of the numbers between -1 and -9 for that operator. The order of the projection matrix to be used in each simple operator is then found by subtracting each of these numbers from their largest value.

A subroutine OPGEN1 has been designed to operate on the

Figure 4.1.3

```
      SUBROUTINE OPGEN1(INSTR,ISTART,NR,ND,LOC,LO,LFCN,LF,
     *                  OPS,LROW,LPOWER,OP,IPROJ)

      THIS SUBROUTINE EVALUATES THE COORDINATE-INDEPENDENT FACTORS
      FOR THE DISCRETIZATION OF LINEAR ORDINARY DIFFERENTIAL
      EQUATIONS. THE EQUATION DISCRETIZED IS CODED IN THE ARRAY
      'INSTR' STARTING IN THE 'ISTART' LOCATION. THE MATRIX FACTORS
      EVALUATED ARE RETURNED IN THE ARRAY 'OPS' WITH THE NUMBER OF
      ROWS IN EACH MATRIX BEING GIVEN BY THE ARRAY 'LROW'.

      DIMENSION INSTR(1),LOC(1),LPOWER(1),LFCN(1),OPS(1),LROW(1),OP(1)
      LO = 0
      LF = 0
      LOCL = ISTART
      LOCS = ISTART

      FIRST THE LOCATION OF THE + OR - SIGNS IS DETERMINED

    1 LOCS = LOCS + 1
      IF(INSTR(LOCS) .LT. 1000) GO TO 1
      SIGN = 1.
      IF(INSTR(LOCL) .EQ. 2000) SIGN = - 1.
      LOCL = LOCS
      L = LOCS - 1
      IF(INSTR(L - 1) .EQ. 200) L = L - 1
      IF(INSTR(L) .EQ. 200) GO TO 3

      THE CURRENT TERM IS A KNOWN FUNCTION - ITS LOCATION AND SIGN
      ARE STORED IN 'LFCN'

      LF = LF + 1
      LFCN(LF) = L*IFIX(SIGN)
    2 L = L - 1
      IF(INSTR(L) .LT. 1000) GO TO 2
      LF = LF + 1
      LFCN(LF) = L + 1
      IF(INSTR(LOCS) .GE. 3000) RETURN
      GO TO 1

      THE SIMPLE OPERATOR ENCOUNTERED CONTAINS THE UNKNOWN Y

    3 M = ND
      IDP = 1
    4 LO = LO + 1
      LOC(LO) = L - 1
      LPOWER(LO) = 0
      N = M
      MTYMSN = M*N

      THE APPROPRIATE UNIT MATRIX IS CREATED

      DO 5 I = 1,MTYMSN
    5 OP(I) = 0.
      MN = N + 1
      DO 6 I = 1,MTYMSN,MN
    6 OP(I) = 1.*SIGN
      IEO = 1
      MN = N
      IF(N .NE. ND) GO TO 9
      IF(IPROJ .EQ. 1) M = NR

C
C     THE REQUIRED PROJECTION MATRIX IS EVALUATED
C
      I = LOCS
    7 I = I - 1
      IF(INSTR(I) .EQ. 300) IDP = - 1
      IF(IPROJ .NE. 1) GO TO 8
      IF(IABS(INSTR(I)) .GE. 10) GO TO 8
      M = M - INSTR(I)
    8 IF(INSTR(I) .LT. 1000) GO TO 7
      IF(IPROJ .NE. 1) GO TO 9
      CALL OPRATR(OP,IEO,M,MN,N,0)
      MN = M
    9 L = L - 1
      IF(IABS(INSTR(L)) .GE. 10) GO TO 10
C
C     THE REQUIRED DIFFERENTIATION MATRIX IS EVALUATED
C
      LPOWER(LO) = LPOWER(LO) - INSTR(L)
      M = MN + INSTR(L)
      CALL OPRATR(OP,IEO,M,MN,N,1)
      MN = M
      GO TO 9
   10 IF(INSTR(L) .GE. 1000 .OR. IABS(INSTR(L)) .LT. 10) GO TO 11
      L = L - 1
      GO TO 10
   11 L = L + 1
C
C     THE MATRIX COMPUTED IS STORED IN 'OP'
C
      LOC(LO) = IDP*LOC(LO)
      LROW(LO) = M
      MTYMSN = M*N
      IF(LO .EQ. 1) LOPS = - MTYMSN
      LOPS = LOPS + MTYMSN
      DO 12 I = 1,MTYMSN
   12 OPS(I + LOPS) = OP(I)
      LO = LO + 1
      LOC(LO) = L
      SIGN = 1.
      IF(INSTR(L - 1) .LT. 1000) GO TO 4
      LOC(LO) = - LOC(LO)
      IF(INSTR(LOCS) .GE. 3000) RETURN
      GO TO 1
      END
```

above principles and is presented in Figure 4.1.3. The instructions for the differential equation to be discretized are contained in the array INSTR with the first instruction in location ISTART. The numbers NR and ND give the number of interpolation points (degree of the interpolation polynomial plus one) in the range and domain spaces of the operator, respectively. During execution, the program checks successive values of INSTR, noting the starting and ending locations of simple operators in LOC and of functions in LFCN. As differential and projection operators are detected, their matrix equivalents are generated by calling OPRATR and stored consecutively in OPS. The arrays LROW and LPOWER give the number of rows in these matrices and the power to which the quantity (1/L) in the differentiation matrix is to be raised.

4.2 COORDINATE-DEPENDENT FACTORS

OPGEN1 completes the first level of the discretization process by evaluating all of the numbers in the matrix equivalent of a differential equation that are independent of coordinates. The second stage of the process, adding the coordinate dependent factors, is accomplished by the companion subroutine OPGEN2 shown in Figure 4.2.1. This subroutine takes each simple operator separately and evaluates the function matrices at the interpolation nodal coordinates given in XD. The previously calculated differentiation and projection matrices are then combined

Figure 4.2.1

```
      SUBROUTINE OPGEN2(INSTR,CONST,LOC,LPOWER,LROW,L,ND,
     *                  XD,WT,N,NPT,OPS,LOPS,OP,TEMP)

      THE SUBROUTINE OPGEN2 TAKES THE COORDINATE-INDEPENDENT FACTORS
      STORED IN THE ARRAY 'OPS' AND MULTIPLIES THEM WITH THE
      APPROPRIATE COORDINATE DEPENDENT PARAMETERS. THE POINT
      COORDINATES ARE CONTAINED IN THE ARRAY 'XD' WITH THE QUANTITY
      (1/L) STORED IN 'WT'.

      DIMENSION INSTR(1),CONST(1),LOC(1),LPOWER(1),LROW(1),
     *          XD(1),WT(1),OPS(1),OP(1),TEMP(1)
      DOUBLE PRECISION DOUBLE
      MN = ND
    1 M = LROW(L)
      LOPS = LOPS + M*MN
      IF(MN .EQ. ND) GO TO 4

      IF A MATRIX IS ALREADY STORED IN THE ARRAY 'OP', IT IS
      MULTIPLIED BY THE NEXT MATRIX FACTOR

      IJ = 0
      LOM = LOPS - M
      JIMN = -MN
      DO 3 J = 1,ND
      JIMN = JIMN + MN
      DO 3 I = 1,M
      IJ = IJ + 1
      IK = I + LOM
      KJ = JIMN
      DOUBLE = 0.D0
      DO 2 K = 1,MN
      IK = IK + M
      KJ = KJ + 1
    2 DOUBLE = DOUBLE + DBLE(OPS(IK)*OP(KJ))
    3 TEMP(IJ) = SNGL(DOUBLE)
    4 LOCL = IABS(LOC(L))
      LOCL1 = IABS(LOC(L + 1))

      THE NODAL POINTS ARE STEPPED OFF

      WEIGHT = WT(N)**LPOWER(L)
      IF(M .EQ. 1) GO TO 5
      STEP = 1.0/(FLOAT(M - 1)*WT(N))
      X = XD(NPT) - STEP
      GO TO 6
    5 STEP = 0.5/WT(N)
      X = XD(NPT)
    6 DO 8 I = 1,M
      X = X + STEP

      THE VALUE OF THE FUNCTION AT THE NODAL POINT IS EVALUATED

      FUNCT = FCN(X,INSTR,CONST,LOCL,LOCL1)*WEIGHT
      IJ = I - M
      DO 8 J = 1,ND
      IJ = IJ + M

      THE COMPUTED MATRIX IS STORED IN THE ARRAY 'OP'

      IF(MN .EQ. ND) GO TO 7
      OP(IJ) = FUNCT* TEMP(IJ)
      GO TO 8
    7 OP(IJ) = FUNCT*OPS(IJ + LOPS)
    8 CONTINUE
      IF(LOC(L + 1) .LT. 0) RETURN
      MN = M
      L = L + 2
      GO TO 1
      END
```

with the function matrices in the specified order to produce, in the array OP, the matrix corresponding to the simple operator.

The function values used by OPGEN2 are computed by the subprogram FCN given in Figure 4.2.2. This program requires the following extension of the instruction code

NUMBER	OPERATION
600	- sign within a function
500	+ sign within a function
400	)
300	(
100	X (the independent variable)
19 ~ 15	auxiliary functions
14	logarithmic function
13	exponential function
12	cosine function
11	sine function
-101 ~ -199	constants
-201 ~ -299	exponentiation

With this code, FCN can generate the point values of any function configuration. The numerical values of constants and the powers to which an argument are to be raised are contained in the array CONST.

In order to permit the use of functions different from the four standard functions sine, cosine, exponential and log-

Figure 4.2.2

```
      FUNCTION FCN(X,INSTR,CONST,LOC1,LOC2)

      THIS FUNCTION SUBPROGRAM RETURNS THE VALUE OF A FUNCTION STORED
      IN LOCATIONS 'LOC2' TO 'LOC1' OF THE ARRAY 'INSTR' AT THE
      LOCATION 'X'

      INTEGER*2 INSTR,LEVEL,IOP
      DIMENSION INSTR(1),CONST(1),F(20),LEVEL(20),IOP(20)
      I = 0
      IPARA = 0
      L = LOC1 + 1

      THE VALUE OF THE FUNCTION IS DETERMINED BY READING THE INSTRUC-
      TIONS FROM RIGHT TO LEFT

    1 L = L - 1
      INSTR4 = INSTR(L)
      IN = MOD(INSTR4,100)
      IF(IN .NE. 0) GO TO 9
      IN = INSTR4 / 100
      GO TO (2,2,3,6,7,8,22),IN
    2 I = I + 1

      THE FUNCTION 'X' ORIGINATES A NEW TERM

      F(I) = X
      IOP(I) = 0
      GO TO 22
    3 IN = LEVEL(IPARA) + 1

      A CLOSING PARENTHESIS MEANS THAT THE FUNCTION WITHIN NEEDS TO
      BE EVALUATED

      IF(IN .EQ. I) GO TO 22
      I = I - 1
      DO 5 K = IN,I
      IF(IOP(K) .EQ. 0) GO TO 4
      F(IN) = F(K + 1) + IOP(K)*F(IN)
      GO TO 5
    4 F(IN) = F(K + 1) * F(IN)
    5 CONTINUE
      I = IN
      IOP(I) = 0
      IPARA = IPARA - 1
      GO TO 22
    6 IPARA = IPARA + 1

      OPENING PARENTESIS ARE NOTED

      LEVEL(IPARA) = I
      GO TO 22
    7 IOP(I) = 1
      GO TO 22
    8 IOP(I) = -1
      GO TO 22
    9 IF(INSTR4 .GT. - 200) GO TO 10

      THE ARGUMENT IS TO BE RAISED TO SOME POWER

      F(I) = F(I)**CONST(- 200 - INSTR4)
      GO TO 22
   10 IF(INSTR4 .GT. - 100) GO TO 12
      IF(INSTR4 .LT. - 150) GO TO 22

      CONSTANTS ALSO ORIGINATE A NEW TERM

      I = I + 1
      F(I) = CONST( - 100 - INSTR4)
      IOP(I) = 0
      GO TO 22
   12 IN = INSTR4 - 10
      GO TO (13,14,15,16,17,18,19,20,21), IN

      ONE OF THE FOLLOWING FUNCTIONS HAS BEEN ENCOUNTERED

   13 F(I) = SIN(F(I))
      GO TO 22
   14 F(I) = COS(F(I))
      GO TO 22
   15 F(I) = EXP(F(I))
      GO TO 22
   16 F(I) = ALOG(F(I))
      GO TO 22
   17 F(I) = FN1(F(I))
      GO TO 22
   18 F(I) = FN2(F(I))
      GO TO 22
   19 F(I) = FN3(F(I))
      GO TO 22
   20 F(I) = FN4(F(I))
      GO TO 22
   21 F(I) = FN5(F(I))
   22 IF(L .GT. LOC2) GO TO 1

      IF L IS LESS OR EQUAL TO LOC2,RETURN THE VALUE OF THE FUNCTION

      FCN = F(1)
      IF(I .EQ. 1) RETURN
      DO 23 L = 2,I
   23 FCN = F(L) * FCN
      RETURN
      END
```

arithm included in the program, the numbers 15 to 19 are reserved for the auxiliary functions FN1 to FN5. These auxiliary functions allow users of FCN to specify any desired function by supplying separate Fortran function subprograms called FN1 - FN5 to generate them. In this way, the statements in FCN never need to be altered.

Finally, the full matrix equivalent of an ordinary differential equation can be obtained by adding together the set of simple matrices generated by OPGEN2 and setting the result equal to the forcing function values at the interpolation nodes. This operation is performed by the subroutine DIFFEQ given in Figure 4.2.3. DIFFEQ is a master subroutine that requires as input only the correct values in the instruction code, plus a few other constants, and generates by calling OPGEN1 and OPGEN2 the matrix equivalent of any differential equation. Some additional features of DIFFEQ, which have not been theoretically explained at this point, will be described in the next chapter.

The use and operation of the automatic discretization subroutine package will now be illustrated by a simple example. Consider the differential equation

$$\frac{d^3y}{dx^3} - \sin(x)\, y = \cos(1.2 + \sin(x)) \qquad (4.2.1)$$

where y is to be approximated by a sixth order polynomial in the interval 0 to 1. The instruction code corresponding to (4.2.1) is

Figure 4.2.3

```
      SUBROUTINE DIFFEQ(DN,MDN,WT,NELMTS,INSTR,CONST,XD,NR,ND,
     *     NULLD,NULLV,LAMDA)

      DIFFEQ DETERMINES THE MATRIX EQUIVALENT OF AN ARBITRARY
      ORDINARY DIFFERENTIAL EQUATION AND RETURNS 'NELMTS' SECTIONAL
      GENERAL SOLUTIONS. THE DIFFERENTIAL EQUATION IS DEFINED IN THE
      ARRAY 'INSTR' AND THE SOLUTION IS RETURNED IN 'DN'.

      DIMENSION DN(1),WT(1),INSTR(1),CONST(1),XD(1)
      DIMENSION D(300),IROW(10),JCOL(30),LOC(100),LPOWER(100),LROW(100),
     *          LFCN(100),OP(100),TEMP(100),OPS(1)
      COMMON D,IFCN,JCOL,LOC,LPOWER,LROW,LFCN,OP,TEMP,OPS
      LOGICAL LAMDA

      DETERMINE THE COORDINATE-INDEPENDENT FACTORS IN THE MATRIX
      EQUIVALENT

      CALL OPGEN1(INSTR,1,NR,ND,LOC,LD,LFCN,LF,OPS,LROW,LPOWER,OP,1)
      IF(LD .LE. 0 .OR. LD .GT. 100 .OR. LF .GT. 100) GO TO 990

      DETERMINE THE MATRIX SIZE PARAMETERS

      NR1 = NR + 1
      MSIZE = NR*ND
      MBLANK = MSIZE + NR
      MDN = MD
      NTOTAL = ND + 1
      NULLV = NULLD + 1
      IF(.NOT. LAMDA) GO TO 2
      MBLANK = 2 = MSIZE
      MDN = 2*ND
      NTOTAL = 2 * ND
      NULLV = MDN - NR
    2 NDSIZE = MDN*NULLV
      NPT = - ND + 1
      MADDDN = 0

      TAKE EACH ELEMENT SEPARATELY

      DO 10 N = 1,NELMTS
      NPT = NPT + ND
      DO 3 I = 1,MBLANK
    3 D(I) = 0.
      LOPS = - LROW(1)*ND

      TAKE EACH SIMPLE OPERATOR SEPARATELY

      DO 4 L = 1,LD,2

      ADD THE COORDINATE-DEPENDENT FACTORS

      CALL OPGEN2(INSTR,CONST,LOC,LPOWER,LROW,L,ND,XD,WT,N,NPT,
     *            OPS,LOPS,OP,TEMP)
      IJ = 0
      IF(LOC(L) .LT. 0) IJ = MSIZE
      DO 4 I = 1,MSIZE
      IJ = IJ + 1

C
C     ADD THE SIMPLE OPERATORS IN A COMPOUND OPERATOR TOGETHER
C
    4 D(IJ) = D(IJ) + OP(I)
      IF(LF .EQ. 0) GO TO 8
      IF(LAMDA) GO TO 990
      IF(NR .EQ. 1) GO TO 5
      STEP = 1.0/(FLOAT(NR - 1)*WT(N))
      X = XD(NPT) - STEP
      GO TO 6
    5 STEP = 0.5/WT(N)
      X = XD(NPT)
C
C     THE MATRIX IS AUGMENTED WITH THE SUM OF THE FORCING FUNCTION
C     POINT VALUES
C
    6 DO 7 L = 1,LF,2
      LOCL = IABS(LFCN(L))
      LOCL1 = LFCN(L + 1)
      WEIGHT = FLOAT(LOCL)/FLOAT(LFCN(L))
      XR = X
      DO 7 I = 1,NR
      XR = XR + STEP
      IJ = I + MSIZE
    7 D(IJ) = D(IJ) + FCN(XR,INSTR,CONST,LOCL,LOCL1)*WEIGHT
C
C     DETERMINE THE GENERAL SOLUTION IN EACH ELEMENT
C
    8 CALL NULL(D,NR,ND,NTOTAL,IRANK,IROW,JCOL,DN,MDN,NULLV,MADDDN)
      IF(IRANK .NE. NR) GO TO 991
      MADDDN = MADDDN + NDSIZE
   10 CONTINUE
      RETURN
C
C     ERRORS ENCOUNTERED
C
  990 WRITE(6,200) LAMDA,LD,LF
      ND = 0
      RETURN
  991 WRITE(6,201) IRANK,NR
      ND = 0
      RETURN
  200 FORMAT('-SORRY, THIS EQUATION CANNOT BE SOLVED BY THIS PROGRAM',
     *       3I5)
  201 FORMAT('-THE CALCULATED RANK',I3,' DOES NOT EQUAL THE TRUE RANK',
     *       I3)
      END
```

I	INSTR (I)	I	INSTR (I)
1	1000	11	12
2	-3	12	300
3	200	13	-101
4	2000	14	500
5	11	15	11
6	300	16	300
7	100	17	100
8	400	18	400
9	200	19	400
10	2000	20	3000

In the above instructions, the function on the right hand side of eqution (4.2.1) has been transferred to the left hand side. Also, the constant 1.2 of instruction 13 is to be stored in CONST (1).

Given these instructions, the subroutine OPGEN1 performs the following operations. It detects the four simple operator separation locations - numbers 1, 4, 10 and 20 - and determines, by looking for the number 200, that in the first and second group of instructions (numbers 1 to 3 and 4 to 9), the unknown y appears but that it is absent in the last group (numbers 10 to 19). This information is stored in the arrays LOC and LFCN as follows

LOC (1) = 1 LOC (3) = 4 LFCN (1) = 10

LOC (2) = 3 LOC (4) = 9 LFCN (2) = 19

Consequently, coordinate-independent matrix operators need to be evaluated only with the first two groups of instructions. Since there is a -3 in the first group and no negative numbers whatsoever in the second, and since sixth order polynomials are to be used, the values of NR and ND become 4 and 7, respectively. Furthermore, the first simple operator stored in OPS is the 4 x 7 projection matrix.

Upon completion of these calculations, these numbers are returned to DIFFEQ which then calls OPGEN2 twice, first with instruction numbers 1 to 3 and then with instruction numbers 4 to 9. Since there are no functions in instructions 1 to 3, OPGEN2 only multiplies the elements of the first simple matrix by $(1/L)^3$ with L=1.0. However, in the next case, according to statements 5 to 8, the rows of the second simple matrix are multiplied by the value of $\sin(x)$ at the four points $x_i = i/3$, i=0,1,2,3.

As these two 4 x 7 matrices are produced, they are added together in the subroutine DIFFEQ and stored in the array D. Following this, the function in instructions 10 to 19 is augmented to the matrix D calling FCN at the aforementioned four points x_i.

This problem was run on an IBM 360/75 computer to determine an estimate of the computation times required by the programs. The generation of the 4 x 7 third order differentiation and projection matrices by the subroutine OPRATR required 16 msec each. Including this time, the complete determination of the coordinate-independent components by the subroutine OPGEN1 took 33 msec. In performing its computations for the single region considered above, OPGEN2 used 16 msec. Altogether

the complete discretization process, including all of the relevant operations performed by DIFFEQ, required 67 msec.

4.3 TWO-DIMENSIONAL EQUATIONS

Considering the physical differences between ordinary differential equations and two-dimensional partial differential equations, the formation of matrix equivalents for the two types of problems is remarkably similar. As explained in section 3.5, in a product separable space, the matrix equivalent of a two-dimensional coordinate-independent differential operator factors into two one-dimensional matrix equivalents, one of which multiplies the matrix of unknown coefficients from the right and the other multiplies it transposed from the left. As a consequence of this reliance upon one-dimensional matrices, computer programs for the discretization of partial differential equations by the methods of Chapter 3 can be constructed by properly adapting the one-dimensional programs given in the preceding section.

In fact, the generation of the high-order one-dimensional differentiation and projection matrices needed for two-dimensional partial differential equations can be performed without modification by the subroutine OPRATR in Figure 4.1.2. The essential difference in the two-dimensional case is that OPRATR needs to be called twice, once for the x-dependent operators and once for the y-dependent operators. A subroutine, OPG2D1, which has been designed to perform this operation for any linear two-dimensional partial differential

equation, is given in Figure 4.3.1. OPG2D1 is very similar in form to OPGEN1 and, as a result, large blocks of statements which are identical to the original one-dimensional program have not been repeated. In their place, comment statements have been inserted to indicate the statement numbers deleted. Thus the comment between statement numbers 3175 and 3185 of OPG2D1 indicates that Fortran statements 6715 through 6860 of OPGEN1 should appear in that location. This method of shortening program listings will be used wherever possible in this thesis.

OPG2D1 requires the following modification of the instruction code given in sections 4.1 and 4.2

200	Z (the unknown function)
101	Y (one of the independent variables)
100	X (the other independent variable)
9~1	D9/DY9 ~ D/DY
-1~-9	D/DX ~ D9/DX9

Given a set of instructions according to this code for a partial differential equation, OPG2D1 determines the number of x-dependent differential operators and the number of y-dependent differential operators in each term of the equation. It then evaluates the corresponding matrix equivalents and applies the proper projection matrices to both the x-dependent and the y-dependent operators to keep the polynomial orders of their spaces consistent.

Figure 4.3.1

```
      SUBROUTINE OPG2D1(INSTR,ISTART,NRX,NRY,ND,LOC,LO,LFCN,LF,
     *     OPS,LROW,LPOWER,OPX,OPY,IPROJ)

      THIS SUBROUTINE EVALUATES THE COORDINATE-INDEPENDENT FACTORS
      FOR THE DISCRETIZATION OF LINEAR TWO-DIMENSIONAL PARTIAL
      DIFFERENTIAL EQUATIONS.

                              DECL 6715-6860

    3 MX = ND                                                          DE2D3145
      MY = ND                                                          DE2D3150
      IDP = 1                                                          DE2D3155
    4 LO = LO + 1                                                      DE2D3160
      LO1 = LO + 1                                                     DE2D3165
      LOC(LO) = L - 1                                                  DE2D3170
      LPOWER(LO) = 0                                                   DE2D3175
      LPOWER(LO1) = 0                                                  DE2D3180
      NX = MX                                                          DE2D3185
      NY = MY                                                          DE2D3190
      MTYMSN = NX ** 2                                                 DE2D3195

      CREAT TWO UNIT MATRICES                                          DE2D3200

      DO 5 I = 1,MTYMSN                                                DE2D3205
    5 OPX(I) = 0.                                                      DE2D3210
      MNX = NX + 1                                                     DE2D3215
      DO 6 I = 1,MTYMSN,MNX                                            DE2D3220
    6 OPX(I) = 1. * SIGN                                               DE2D3225
      MTYMSN = NY ** 2                                                 DE2D3230
      DO 7 I = 1,MTYMSN                                                DE2D3235
    7 OPY(I) = 0.                                                      DE2D3240
      MNY = NY + 1                                                     DE2D3245
      DO 8 I = 1,MTYMSN,MNY                                            DE2D3250
    8 OPY(I) = 1.                                                      DE2D3255
      IEOX = 1                                                         DE2D3260
      IEOY = 1                                                         DE2D3265
      MNX = MX                                                         DE2D3270
      MNY = MY                                                         DE2D3275
      IF(INSTR(L) .NE. 200) GO TO 13                                   DE2D3280
      I = LOCS                                                         DE2D3285
    9 I = I - 1                                                        DE2D3290
      IF(INSTR(I) .EQ. 300) IDP = - 1                                  DE2D3295
      IF(IPROJ .NE. 1) GO TO 11                                        DE2D3300
      IF(IABS(INSTR(I)) .GE. 10) GO TO 11                              DE2D3305
      IF(INSTR(I) .GE. 0) GO TO 10                                     DE2D3310
      MX = MX + INSTR(I)                                               DE2D3315
      GO TO 11                                                         DE2D3320

      ADD THE PROJECTION MATRICES                                      DE2D3325

   10 MY = MY - INSTR(I)                                               DE2D3330
   11 IF(INSTR(I) .LT. 1000) GO TO 9                                   DE2D3335
      IF(MX .EQ. NRX) GO TO 12                                         DE2D3340
      MX = NRX + ND - MX                                               DE2D3345
      CALL OPRATR(OPX,IEOX,MX,MNX,NX,0)                                DE2D3350
      MNX = MX                                                         DE2D3355
   12 IF(MY .EQ. NRY) GO TO 13                                         DE2D3360
      MY = NRY + ND - MY                                               DE2D3365
      CALL OPRATR(OPY,IEOY,MY,MNY,NY,0)                                DE2D3370
      MNY = MY                                                         DE2D3375
   13 L = L - 1                                                        DE2D3380
      IF(IABS(INSTR(L)) .GE. 10) GO TO 15                              DE2D3385
C                                                                      DE2D3390
C     ADD THE DIFFERENTIATION MATRICES                                 DE2D3395
C                                                                      DE2D3400
      IF(INSTR(L) .GE. 0) GO TO 14                                     DE2D3405
      LPOWER(LO) = LPOWER(LO) - INSTR(L)                               DE2D3410
      MX = MNX + INSTR(L)                                              DE2D3415
      CALL OPRATR(OPX,IEOX,MX,MNX,NX,1)                                DE2D3420
      MNX = MX                                                         DE2D3425
      GO TO 13                                                         DE2D3430
   14 LPOWER(LO1) = LPOWER(LO1) + INSTR(L)                             DE2D3435
      MY = MNY - INSTR(L)                                              DE2D3440
      CALL OPRATR(OPY,IEOY,MY,MNY,NY,1)                                DE2D3445
      MNY = MY                                                         DE2D3450
      GO TO 13                                                         DE2D3455
   15 IF(INSTR(L) .GE. 1000 .OR. IABS(INSTR(L)) .LT. 10) GO TO 16      DE2D3460
      L = L - 1                                                        DE2D3465
      GO TO 15                                                         DE2D3470
C                                                                      DE2D3475
C     BOTH MATRICES ARE STORED IN (OPS)                                DE2D3480
C                                                                      DE2D3485
   16 L = L + 1                                                        DE2D3490
      LOC(LO) = IDP * LOC(LO)                                          DE2D3495
      LROW(LO) = MX                                                    DE2D3500
      LROW(LO1) = MY                                                   DE2D3505
      MTYMSN = MX * MX                                                 DE2D3510
      IF(LO .EQ. 1) LOPS = 0                                           DE2D3515
      DO 17 I = 1, MTYMSN                                              DE2D3520
   17 OPS(I + LOPS) = OPX(I)                                           DE2D3525
      LOPS = LOPS + MTYMSN                                             DE2D3530
      MTYMSN = MY * MY                                                 DE2D3535
      DO 18 I = 1,MTYMSN                                               DE2D3540
   18 OPS(I + LOPS) = OPY(I)                                           DE2D3545
      LOPS = LOPS + MTYMSN                                             DE2D3550
      LO = LO1                                                         DE2D3555
C                                                                      DE2D3560
C                              DECL 7150-7175                          DE2D3565
C                                                                      DE2D3570
      END                                                              DE2D3575
```

The coordinate-dependent factors of two-dimensional equations are added to the system by the subroutine OPG2D2 in Figure 4.3.2. OPG2D2 is patterned after the one-dimensional program OPGEN2 and, again, many statements in OPG2D2 are similar to those in OPGEN2. The basic difference between the two programs is that OPG2D2 must convert the pair of one-dimensional matrices obtained from OPG2D1 for each differential operator into a single coefficient matrix by taking the Kronecker product of the matrices. This operation is performed by the subroutine KRON given in Figure 4.3.3.

The major new problem encountered when changing from one to two-dimensional differential equations is the rapid growth in the size of the matrix equivalents of the operators. In the one-dimensional program, the largest possible matrix equivalent of a differential operator results when a ninth order polynomial approximation is made of a first order differential operator and produces a matrix with 90 elements. With two-dimensional problems, the Kronecker product of two of these one-dimensional matrices results in a matrix with $90^2 = 8100$ elements. Although matrices of this size are within the limits of the storage capacities of modern computers, they require a sizable fraction of core memory and operations with them consume considerable computational time. Therefore, the generation of very high-order matrix equivalents for two-dimensional problems requires much more consideration than is necessary in the one-dimensional case. Since the number of matrix elements in a two-dimensional problem varies approximately as n^4 where n is the order of the approximating polynomial, it is often wise to use a lower order element,

Figure 4.3.2

```
      SUBROUTINE OPG2D2(INSTR,CONST,LOC,LPOWER,LROW,L,ND,XD,YD,WTX,WTY,  DE2D3645
     *     N,NPT,OPS,LOPX,OP,TEMP)                                      DE2D3650
                                                                        DE2D3655
      THE SUBROUTINE OPG2D2 TAKES THE COORDINATE-INDEPENDENT FACTORS OF DE2D3660
      A TWO-DIMENSIONAL PARTIAL DIFFERENTIAL EQUATION AND MULTIPLES     DE2D3665
      THEM WITH THE APPROPRIATE COORDINATE-DEPENDENT PARMAMETERS.       DE2D3670 C
                                                                        DE2D3675 C
      DOUBLE PRECISION DOUBLE                                           DE2D3680 C
      DIMENSION INSTR(1),CONST(1),LOC(1),LPOWER(1),LROW(1),             DE2D3685
     *      XD(1),YD(1),WTX(1),WTY(1),OPS(1),OP(1),TEMP(1)              DE2D3690
      MNX = ND                                                          DE2D3695
      MNY = ND                                                          DE2D3700
    1 L1 = L + 1                                                        DE2D3705
      MX = LROW(L)                                                      DE2D3710 C
      MY = LROW(L1)                                                     DE2D3715 C
      LOPY = LOPX + MX * MNX                                            DE2D3720 C
      IF(MNX .EQ. ND .AND. MNY .EQ. ND) GO TO 4                         DE2D3725
                                                                        DE2D3730
      EVALUATE THE NEXT MATRIX OPERATOR                                 DE2D3735
                                                                        DE2D3740
      CALL KRON(OPS,MX,MNX,LOPX,OPS,MY,MNY,LOPY,TEMP,MOP,MNOP,0)        DE2D3745
      IJ = 0                                                            DE2D3750
      KJ = 0                                                            DE2D3755
      DO 3 J = 1,NOP                                                    DE2D3760
      KJ = KJ + MNOP                                                    DE2D3765
      DO 3 I = 1,MOP                                                    DE2D3770
      IJ = IJ + 1                                                       DE2D3775
      IK = I - MOP                                                      DE2D3780
      DO 2 K = 1,MNOP                                                   DE2D3785
      IK = IK + MOP                                                     DE2D3790
    2 DOUBLE = DOUBLE + DBLE(TEMP(IK) * OP(K + KJ))                     DE2D3795
    3 OP(IJ) = SNGL(DOUBLE)                                             DE2D3800
      GO TO 10                                                          DE2D3805
    4 CALL KRON(OPS,MX,MNX,LOPX,OPS,MY,MNY,LOPY,OP,MOP,NOP,0)           DE2D3810
   10 LOPX = LOPY + MY * MNY                                            DE2D3815
      LOCL = IABS(LOC(L))                                               DE2D3820
      LOCL1 = IABS(LOC(L1))                                             DE2D3825
                                                                        DE2D3830
      THE NODAL POINTS ARE STEPPED OFF                                  DE2D3835
                                                                        DE2D3840
      WEIGHT = (WTX(N) ** LPOWER(L)) * (WTY(N) ** LPOWER(L1))           DE2D3845
      IF(MX .EQ. 1) GO TO 5                                             DE2D3850
      STEPX = 1.0 / ((MX - 1) * WTX(N))                                 DE2D3855
      X = XD(NPT) - STEPX                                               DE2D3860
      GO TO 6                                                           DE2D3865
    5 STEPX = 0.5 / WTX(N)                                              DE2D3870
      X = XD(NPT)                                                       DE2D3875
    6 IF(MY .EQ. 1) GO TO 7                                             DE2D3880
      STEPY = 1.0 / ((MY - 1) * WTY(N))                                 DE2D3885
      YS = YD(NPT) + STEPY                                              DE2D3890
      GO TO 8                                                           DE2D3895
    7 STEPY = 0.5 / WTY(N)                                              DE2D3900
      YS = YD(NPT)                                                      DE2D3905
    8 IK = 0                                                            DE2D3910
      DO 9 I = 1,MX                                                     DE2D3915
      X = X + STEPX                                                     DE2D3920
      Y = YS                                                            DE2D3925
      DO 9 K = 1,MY                                                     DE2D3930
      Y = Y - STEPY                                                     DE2D3935
                                                                        DE2D3940
      EVALUATE THE FUNCTION AT THE NODAL POINT                          DE2D3945
                                                                        DE2D3950
      FUNCT = FCN2D(X,Y,INSTR,CONST,LOCL,LOCL1) * WEIGHT                DE2D3955
      IK = IK + 1                                                       DE2D3960
      IJ = IK - MOP                                                     DE2D3965
      KJ = IJ                                                           DE2D3970
      IF(LOC(L1) .GE. 0) KJ = IJ + MOP                                  DE2D3975
                                                                        DE2D3980
      THE COMPUTED MATRIX IS STORED IN 'OP'                             DE2D3985
                                                                        DE2D3990
      DO 9 J = 1,NOP                                                    DE2D3995
      IJ = IJ + MOP                                                     DE2D4000
      KJ = KJ + MOP                                                     DE2D4005
    9 OP(KJ) = FUNCT * OP(IJ)                                           DE2D4010
      IF(LOC(L1) .LT. 0) RETURN                                         DE2D4015
      MNX = MX                                                          DE2D4020
      MNY = MY                                                          DE2D4025
      L = L + 2                                                         DE2D4030
      GO TO 1                                                           DE2D4035
      END                                                               DE2D4040
```

Figure 4.3.3

```
      SUBROUTINE KRON(A,MA,NA,IAADD,B,MB,NB,IBADD,C,MC,NC,ICADD)          DE2D4045
                                                                          DE2D4050
      KRON TAKES THE KRONECKER PRODUCT OF THE 'MA' BY 'NA' MATRIX 'A'     DE2D4055
      AND THE 'MC' BY 'NB' MATRIX 'B' AND RETURNS THE RESULTING 'MC'      DE2D4060
      BY 'NC' MATRIX IN 'C'.                                              DE2D4065
                                                                          DE2D4070
      DIMENSION A(1),B(1),C(1)                                            DE2D4075
      MC = MA * MB                                                        DE2D4080
      NC = NA * NB                                                        DE2D4085
      IJC = ICADD                                                         DE2D4090
      IBA = IBADD + MB * NB + MB + 1                                      DE2D4095
      JACOL = IAADD - MA                                                  DE2D4100
      DO 1 JA = 1,NA                                                      DE2D4105
      JACOL = JACOL + MA                                                  DE2D4110
      JBCOL = IBA                                                         DE2D4115
      DO 1 JB = 1,NB                                                      DE2D4120
      JBCOL = JBCOL - MB                                                  DE2D4125
      IJA = JACOL                                                         DE2D4130
      DO 1 IA = 1,MA                                                      DE2D4135
      IJA = IJA + 1                                                       DE2D4140
      DO 1 IB = 1,MB                                                      DE2D4145
      IJC = IJC + 1                                                       DE2D4150
    1 C(IJC) = A(IJA) * B(JBCOL - IB)                                     DE2D4155
      RETURN                                                              DE2D4160
      END                                                                 DE2D4165
```

Figure 4.3.4

```
      FUNCTION FCN2D(X,Y,INSTR,CONST,LOC1,LOC2)                           DE2D4170
                                                                          DE2D4175
C     THIS FUNCTION SUBPROGRAM RETURNS THE VALUE OF A TWO-DIMENSIONAL     DE2D4180
C     FUNCTION STORED IN LOCATIONS 'LOC2' TO 'LOC1' OF THE ARRAY          DE2D4185
C     'INSTR' AT THE LOCATION '(X,Y)'.                                    DE2D4190
C                                                                         DE2D4195
C                              DECL 7890-7940                             DE2D4200
C                                                                         DE2D4205
      IF(INSTR4) .EQ. 101) GO TO 24                                       DE2D4210
C                                                                         DE2D4215
C                              DECL 7945-7995                             DE2D4220
C                                                                         DE2D4225
   24 I = I + 1                                                           DE2D4230
      F(I) = Y                                                            DE2D4235
      IOP(I) = 0                                                          DE2D4240
      GO TO 22                                                            DE2D4245
C                                                                         DE2D4250
C                              DECL 6000-8360                             DE2D4255
C                                                                         DE2D4260
      END                                                                 DE2D4265
```

particularly if several different matrix equivalents are needed simultaneously.

After converting the matrix of unknown coefficients into vector form by applying the Kronecker product, OPG2D2 multiplies the resulting product matrices by any necessary function matrices, as specified by equation (3.5.30). The function values are obtained from the function subprogram FCN2D given in Figure 4.3.4. FCN2D is similar to FCN except that it has two independent variables and calls auxiliary functions FN2D1 - FN2D5, which are dependent on both X and Y.

Last, Figure 4.3.5 contains DIFF2D, the two-dimensional version of the subroutine DIFFEQ. The principles of operation of DIFF2D are identical to those of DIFFEQ, although a large number of indexing changes have been made. By calling DIFF2D with a suitable set of instructions in the array INSTR, the matrix equivalent of any two-dimensional partial differential equation may be obtained.

Figure 4.3.5

```
      SUBROUTINE DIFF2D(DN,MDN,WTX,WTY,NELMTS,INSTR,CONST,XD,YD,
     *      NRX,NRY,ND,NULLD,NULLV,LAMDA)
C
C     DIFF2D DETERMINES THE MATRIX EQUIVALENT OF AN ARBITRARY LINEAR
C     TWO-DIMENSIONAL PARTIAL DIFFERENTIAL EQUATION.
C
      DIMENSION DN(1),WTX(1),WTY(1),INSTR(1),CONST(1),XD(1),YD(1)
      DIMENSION D(4096),IROW(100),JCOL(300),LOC(100),LPOWER(100),
     *      LROW(100),LFCN(100),DP(4096),TEMP(4096),DPS(1)
      COMMON D,IROW,JCOL,LOC,LPOWER,LROW,LFCN,DP,TEMP,DPS
      CALL DPG2D1(INSTR,1,NRX,NRY,ND,LOC,LD,LFCN,LF,DPS,LROW,LPOWER,
     *      DP,TEMP,1)
      IF(LD .LE. 0 .OR. LD .GT. 100 .OR. LF .GT. 100) GO TO 990
      NR2D = NRX * NRY
      ND2D = ND ** 2
      NULLD = ND2D - NR2D
      MSIZE = NR2D * ND2D
      MBLANK = MSIZE + NR2D
      MDN = ND2D
      NTOTAL = ND2D + 1
      NULLV = NULLD + 1
      IF(.NOT. LAMDA) GO TO 2
      MBLANK = 2 * MSIZE
      MDN = 2 * ND2D
      NTOTAL = MDN
      NULLV = MDN - NR2D
    2 NDSIZE = MDN*NULLV
      NPT = - ND2D + 1
C
C                              DECL 2985-3005
C
      NPT = NPT + ND2D
C
C                              DECL 3010-3015
C
      LOPS = 0
      DO 4 L = 1,LD,2
      CALL DPG2D2(INSTR,CONST,LOC,LPOWER,LROW,L,ND,XD,YD,WTX,WTY,
     *      N,NPT,DPS,LOPS,DP,TEMP)
C
C                              DECL 3075-3110
C
      IF(LF .EQ. 0) GO TO 11
      IF(LAMDA) GO TO 990
      IF(NRX .EQ. 1) GO TO 5
      STEPX = 1.0 / ((NRX - 1) * WTX(N))
      X = XD(NPT) - STEPX
      GO TO 6
    5 STEPX = 0.5 / WTX(N)
      X = XD(NPT)
    6 IF(NRY .EQ. 1) GO TO 7
      STEPY = 1.0 / ((NRY - 1) * WTY(N))
      YS = YD(NPT) + STEPY
      GO TO 8
    7 STEPY = 1.0 / WTY(N)
      YS = YD(NPT)
    8 DO 9 L = 1,LF,2
C
C                              DECL 3180-3195
C
      DO 9 I = 1,NRX
      X = X + STEPX
      Y = YS
      DO 9 J = 1,NRY
      Y = Y - STEPY
      IJ = IJ + 1
    9 D(IJ) = D(IJ) + FCN2D(X,Y,INSTR,CONST,LOCL,LOCL1) * WEIGHT
   11 CONTINUE
      CALL NULL(D,NR2D,ND2D,NTOTAL,IRANK,IROW,JCOL,DN,MDN,NULLV,MADDDN)
      IF(IRANK .NE. NR2D) GO TO 991
C
C                              DECL 3245-3320
C
      END
```

CHAPTER V

SOLUTION TECHNIQUES

There are two possible approaches for solving the rectangular matrix equations which result from the discretization of differential equations by the projective method in Chapter 3. In one, boundary conditions are added to the system until the coefficient matrix becomes non-singular. This matrix equation may then be solved using standard methods to yield an approximate particular solution of the differential equation. In the other, the matrix equation is solved by using the matrix generalized inverse concept to determine the approximate general solution of the differential equation. Boundary conditions are added subsequently if a particular solution is desired. Since the latter method is applicable to a much wider range of problems than the first, it is the approach followed in this thesis.

The general solutions which result from the application of matrix generalized inversion to the singular matrix equations of Chapter 3 are completely analogous to the general solutions encountered classically in the analytic theory of differential equations. In both cases, the general solution of an operator equation is given by an inhomogeneous solution and a set of homogeneous solutions which form a basis in the nullspace of the operator. Furthermore, the procedures used to obtain both types of general solutions are independent from the addition of boundary conditions. The numerical method developed in this thesis is unique in this feature among all of the available

approximate solution techniques for the solution of differential equations.

In addition to the ability to produce approximate general solutions, a further advantage of applying matrix generalized inversion to the projective solution of differential equations is the highly efficient matrix solution algorithm that results. This efficiency is derived from substituting manipulations on several small full matrices in multi-regional solutions for the more costly treatment of one large sparse matrix.

The only published work which uses matrix generalized inversion with a finite element type of formulation is a paper by Berkovic [50]. However, since this paper does not deal with differential equations but rather the algebraic Hooke's law equation, the extremely beneficial results which follow from its application to differential equations are not evident there.

5.1 GENERALIZED MATRIX INVERSION

Let A be an $m \times n$ matrix of rank r and suppose that all vectors y satisfying the equation

$$Ay = f \tag{5.1.1}$$

are desired. The simplest method of determing these vectors is to decompose A by the Gaussian elimination process, interchanging rows and columns as needed, until only zero pivots remain [48,

51]. Then A may be factored as

$$A = LU = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} [U_1 \; U_2] \tag{5.1.2}$$

where L_1 and U_1 are non-singular. With this factorization (5.1.1) may be written

$$y_1 = U_1^{-1} L_1^{-1} f_1 - U_1^{-1} U_2 y_2 \tag{5.1.3a}$$

$$f_2 = L_2 L_1^{-1} f_1 \tag{5.1.3b}$$

Equation (5.1.3a) contains all of the useful information from (5.1.1);(5.1.3b) merely indicates that solution is possible only if no conflicts arise. Consequently, (5.1.3a) together with the equation $y_2 = y_2$ gives

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} U_1^{-1} L_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} 0 & -U_1^{-1} U_2 \\ 0 & I \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{5.1.4}$$

Since the columns of the second matrix in (5.1.4) form (n-r) independent nullvectors of the matrix A, a general solution of (5.1.1) is given by [52]

$$y = A^{+} f + Nz \tag{5.1.5}$$

where A^{+} and N are the row and column reinterchanged forms of

$$A^{+} = \begin{pmatrix} U_1^{-1} L_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} \qquad (5.1.6)$$

$$N = \begin{pmatrix} -U_1^{-1} U_2 \\ I \end{pmatrix} \qquad (5.1.7)$$

and the $(n-r)$ components of z are arbitrary.

The matrix A^{+} is called a weak generalized inverse of A [53] and satisfies the following two properties

$$AA^{+}A = A \qquad (5.1.8a)$$

$$A^{+}AA^{+} = A^{+} \qquad (5.1.8b)$$

However, the products

$$AA^{+} = \begin{pmatrix} I & 0 \\ L_2 L_1^{-1} & 0 \end{pmatrix} \qquad (5.1.9a)$$

$$A^{+}A = \begin{pmatrix} I & U_1^{-1} U \\ 0 & 0 \end{pmatrix} \qquad (5.1.9b)$$

are not symmetric in general.

The significance of equation (5.1.3b) may be understood by considering the vector norm $|.|$ of the residual vector

$$|Ay - f| = |AA^{+}f + ANz - f|$$

$$= \left| \left(AA^{+} - I \right) f \right|$$

$$= \left| \begin{pmatrix} 0 & 0 \\ L_2 L_1^{-1} & -I \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \right| \qquad (5.1.10)$$

$$= \left| \begin{pmatrix} 0 & \\ L_2 L_1^{-1} & f_1 - f_2 \end{pmatrix} \right|$$

Therefore, if (5.1.3b) holds, the solution y will be exact ; if (5.1.3b) is not true, y will satisfy (5.1.1) only approximately, the amount of error being dependent on the values of f_1 and f_2.

In most physical problems that are properly formulated, $f \in$ Range(a) so that (5.1.3b) will be satisfied. However, if this is not true, a more accurate approximate solution to (5.1.1) is provided by the Moore-Penrose inverse of a matrix [54] which minimizes the norm (5.1.10) in an L_2 sense. The Moore-Penrose inverse is related to the weak generalized inverse (5.1.6) by [55]

$$A^{MP} = A^* \left(A^* A A^* \right)^{+} A^* \qquad (5.1.11)$$

where A^* is the transpose conjugate of A.

It is also of interest to determine the solution of a system of equations part of which has already been solved in the form (5.1.5). Suppose, for example, that the system of equations (5.1.1) is augmented by another matrix equation

$$By = g \qquad (5.1.12)$$

where B is an m_b x n matrix of rank r_b. Inserting (5.1.5) into (5.1.12) gives

$$BA^{+}f + BNz = g \qquad (5.1.13)$$

The solution of this equation is

$$z = (BN)^{+} (g - BA^{+}f) + Mu \qquad (5.1.14)$$

where the $(n - r) \times (n - r - r_b)$ matrix M is in the nullspace of the matrix (BN) and the vector u is arbitrary. It follows that the solution of (5.1.1) together with (5.1.12) is

$$y = (A^{+} - N(BN)^{+} BA^{+})f + N(BN)^{+}g + NMu \qquad (5.1.15)$$

It may be observed from (5.1.15) that each time an independent equation is added to the system, the solution contains one less unknown. Initially, when the n dimensional space of the vector y is created, all of its components are arbitrary. Then as conditions are imposed in the form (5.1.1), unwanted vector behavior is filtered out, leaving a solution of the kind (5.1.5). As more equations are added, fewer degrees of freedom remain, until, when n linearly independent conditions are imposed, only one particular vector survives.

In actual computation of the general solution of the matrix equation (5.1.5), the most economical procedure is to augment the matrix A with the forcing function f to form the new matrix equation [56]

Figure 5.1.1a

```
      SUBROUTINE NULL(A,M,N,NTOTAL,IRANK,IROW,JCOL,AN,MAN,NAN,MADDAN)

      NULL COMPUTES A LINEARLY INDEPENDENT SET OF NULLVECTORS FOR THE
     'M' BY 'N' MATRIX 'A' AND STORES THE RESULT AS AN 'MAN' BY'NAN'
     MATRIX IN THE ARRAY 'AN'. IT ALSO RETURNS THE PRODUCT (A+)F IN
     THE LAST '(NTOTAL - IRANK)' COLUMNS OF 'AN', WHERE 'F' IS A
     FORCING FUNCTION MATRIX AUGMENTED TO 'A'. 'IROW' AND 'JCOL' ARE
     WORK ARRAYS OF DIMENSION 'M' AND 'N'.

      DOUBLE PRECISION DOUBLE
      DIMENSION A(12),IROW(2),JCOL(6),AN(12)

      THE ROW AND COLUMN VECTORS ARE FIRST INITIALIZED

      DO 1 I = 1,M
    1 IROW(I) = I
      DO 2 J = 1,NTOTAL
    2 JCOL(J) = J
      PIV = 0.
      IJ = 0

      THE LARGEST ELEMENT IN THE MATRIX IS LOCATED

      DO 3 J = 1,N
      DO 3 I = 1,M
      IJ = IJ + 1
      IF(ABS(A(IJ)) .LE. ABS(PIV)) GO TO 3
      PIV = A(IJ)
      IR = I
      JC = J
    3 CONTINUE
      TOL = 1.5E-7*ABS(PIV)
      MSIZE = M*N
      IRANK = 0
      DO 9 NCOL = 1,N
      IF(ABS(PIV) .LE. TOL) GO TO 10

      IF AN ELEMENT LARGER THAN 'TOL' HAS BEEN FOUND, THE RANK OF THE
      MATRIX IS UPDATED

      IRANK = IRANK + 1
      IF(IR .EQ. IRANK) GO TO 5

      THE IRANK'TH ROW IS INTERCHANGED WITH THE ROW CONTAINING THE
      LARGEST ELEMENT

      I = IRANK - M
      IJ = IR - M
      DO 4 JJ = 1,NTOTAL
      I = I + M
      IJ = IJ + M
      STORE = A(I)
      A(I) = A(IJ)
    4 A(IJ) = STORE
      IJ = IROW(IRANK)
      IROW(IRANK) = IROW(IR)
      IROW(IR) = IJ
    5 IF(JC .EQ. IRANK) GO TO 7

C     THE IRANK'TH COLUMN IS INTERCHANGED WITH THE COLUMN CONTAINING
C     THE LARGEST ELEMENT
C
      J = (IRANK - 1)*M + 1
      JJ = J + M - 1
      IJ = (JC - 1)*M
      DO 6 I = J,JJ
      IJ = IJ + 1
      STORE = A(I)
      A(I) = A(IJ)
      A(IJ) = STORE
    6 CONTINUE
      IJ = JCOL(JC)
      JCOL(JC) = JCOL(IRANK)
      JCOL(IRANK) = IJ
    7 IF(IRANK .GE. M) GO TO 10
      I1 = IRANK + 1
      IPIV = M*(IRANK - 1) + IRANK
      I2 = IPIV
      STORE = PIV
      PIV = 0.
C
C     THE GAUSSIAN ELIMINATION STEP IS PERFORMED NEXT
C
      DO 9 I = I1,M
      I2 = I2 + 1
      FACTOR = A(I2)/STORE
      A(I2) = FACTOR
      IF(IRANK .GE. NTOTAL) GO TO 9
      JJ = IPIV
      IJ = I2
      DO 8 J = I1,NTOTAL
      IJ = IJ + M
      JJ = JJ + M
      A(IJ) = A(IJ) - FACTOR*A(JJ)
      IF(ABS(A(IJ)) .LE. ABS(PIV) .OR. J .GT. N) GO TO 8
      PIV = A(IJ)
      IR = I
      JC = J
    8 CONTINUE
    9 CONTINUE
   10 IF(IRANK .EQ. 0 .OR. IRANK .GE. NTOTAL) RETURN
      J1 = IRANK + 1
      I1 = IRANK*M + J1
C
C     NOW THE BACK SUBSTITUTION STEP IS PERFORMED
C
      DO 14 I = 1,IRANK
      I2 = I1 - M
```

```
      IPIV = I1 - I                                                    DECL8905
      DO 13 J = J1,NTOTAL                                              DECL8910
      I2 = I2 + M                                                      DECL8915
      JJ = IPIV                                                        DECL8920
      LL = I2                                                          DECL8925
      DOUBLE = 0.00                                                    DECL8930
      IF(I .EQ. 1) GO TO 20                                            DECL8935
      DO 12 K = 2,I                                                    DECL8940
      JJ = JJ - M                                                      DECL8945
      LL = LL - 1                                                      DECL8950
   12 DOUBLE = DOUBLE - DBLE(A(JJ)*A(LL))                              DECL8955
   20 JJ = JJ - M                                                      DECL8960
      LL = LL - 1                                                      DECL8965
   13 A(LL) =(SNGL(DOUBLE) - A(LL))/A(JJ)                              DECL8970
   14 CONTINUE                                                         DECL8975
      JJ = (IRANK - 1)*M                                               DECL8980
      LL = MADDAN - MAN                                                DECL8985
C                                                                      DECL8990
C     THE NULL VECTORS ARE TRANSFERED TO THE ARRAY 'AN',               DECL8995
C     REINTERCHANGING COLUMNS IN THE PROCESS                           DECL9000
C                                                                      DECL9005
      DO 17 J = J1,NTOTAL                                              DECL9010
      JJ = JJ + M                                                      DECL9015
      LL = LL + MAN                                                    DECL9020
      DO 17 I = 1,MAN                                                  DECL9025
      IJ = JCOL(I) + LL                                                DECL9030
      IF(I .GT. IRANK) GO TO 15                                        DECL9035
      AN(IJ) = A(I + JJ)                                               DECL9040
      GO TO 17                                                         DECL9045
   15 IF(I .EQ. J) GO TO 16                                            DECL9050
      AN(IJ) = 0.                                                      DECL9055
      GO TO 17                                                         DECL9060
   16 AN(IJ) = 1.                                                      DECL9065
   17 CONTINUE                                                         DECL9070
   18 CONTINUE                                                         DECL9075
      RETURN                                                           DECL9080
      END                                                              DECL9085
```

Figure 5.1.1b

$$[\ A\ \vdots\ f]\ \begin{pmatrix} y \\ \hline 1 \end{pmatrix} = [0] \qquad (5.1.16)$$

In this way, only the quantity $U_1^{-1} U_2$ needs to be evaluated for the augmented matrix and, provided that interchanges are not allowed on the last column, the product A^+f will appear in it. If the actual matrix A^+ is required, it can be obtained by augmenting to A the identity matrix instead of f. It must be stressed however, that the inverse A^+ should never be computed purely for the purpose of determining the solution of a matrix equation since this operation alone requires more computation than is needed to find the product A^+f.

Figure 5.1.1 contains a Fortran subroutine NULL which operates on the above principles. The input to the program is the M x NTOTAL array A of which N columns belong to the coefficient matrix and the rest are forcing functions. It returns the nullvectors of the system in the array AN, storing them as MAN component vectors, where MAN need not equal N.

5.2 GENERAL SOLUTIONS

The similarity of solutions produced by generalized inversion of the matrix equivalent of a differential equation and by the analytic theory of differential equations is established in the following two theorems.

THEOREM 5.2.1 (DISCRETE FORM OF THE FIRST FUNDAMENTAL THEOREM OF CALCULUS). Let D^+ be the generalized inverse of a differentiation matrix as defined by (2.1.11) and suppose that y is given by $y = D^+z$. Then

$$Dy = z \tag{5.2.1}$$

PROOF: In the definition of D, $r = m$ so that

$$Dy = DD^+z = [L_1]\,[U_1\ U_2] \begin{pmatrix} U_1^{-1} & L_1^{-1} \\ & 0 \end{pmatrix} [z] = z$$

THEOREM 5.2.2 (DISCRETE FORM OF THE SECOND FUNDAMENTAL THEOREM OF CALCULUS). Let z be any vector such that $z = Dy$ where D is a differentiation matrix as defined by (2.1.11). Then

$$D^+z = y + c \tag{5.2.2}$$

where c is a vector with arbitrary but equal components.

PROOF: This result is a direct consequence of equation (5.1.5) and the fact that the nullspace of D contains equi-component vectors.

The relationship between the discrete and the continuous forms of these theorems will be illustrated by the following

simple example. Consider the problem of determining the general solution of the matrix equivalent of the equation

$$\frac{dy}{dx} = f \tag{5.2.3}$$

when y is approximated by a quadratic polynomial. According to (2.1.16), the rectangular matrix equation is

$$\frac{1}{L}\begin{pmatrix} -3 & 4 & -1 \\ 1 & -4 & 3 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \tag{5.2.4}$$

This coefficient matrix may be decomposed as

$$\frac{1}{L}\begin{pmatrix} -3 & 4 & -1 \\ 1 & -4 & 3 \end{pmatrix} = \frac{1}{L}\begin{pmatrix} 1 & 0 \\ -\frac{1}{3} & 1 \end{pmatrix}\begin{pmatrix} -3 & 4 & -1 \\ 0 & -\frac{8}{3} & \frac{8}{3} \end{pmatrix} \tag{5.2.5}$$

so that

$$L_1^{-1} = \begin{pmatrix} 1 & 0 \\ \frac{1}{3} & 1 \end{pmatrix} \qquad U_1^{-1} = \begin{pmatrix} -\frac{1}{3} & -\frac{1}{2} \\ 0 & -\frac{3}{8} \end{pmatrix} \qquad U_2 = \begin{pmatrix} -1 \\ \frac{8}{3} \end{pmatrix} \tag{5.2.6}$$

Consequently, the general solution of (5.2.4) is

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = L\begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{8} & -\frac{3}{8} \\ 0 & 0 \end{pmatrix}\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} c \tag{5.2.7}$$

where the constant c is arbitrary. By the second fundamental theorem of calculus, the general solution of (5.2.3) is

$$y = \int f \, dx + c' \tag{5.2.8}$$

where c′ is an arbitrary constant. It can be easily verified that multiplying (5.2.7) by the matrix (5.2.4) or multiplying (5.2.8) by $\frac{d}{dx}$ returns each of these equations to their original forms. Hence, Theorems 5.2.1 and 5.2.2 imply for discrete representations of differential equations the same properties that are obtained for continuous spaces from the two fundamental theorems of calculus.

Consider next the differential equation

$$\frac{dy}{dx} = y \tag{5.2.9}$$

According to (2.3.10), the discretized form of this equation in a quadratic space is

$$\frac{1}{L} \begin{pmatrix} -3 & 4 & -1 \\ 1 & -4 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} 2 & 2 & -1 \\ -1 & 2 & 2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{5.2.10}$$

Using (5.1.5), the homogeneous solution of (5.2.10) is

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \frac{1}{L^2 - 6L + 12} \begin{pmatrix} L^2 - 6L + 12 \\ -1/2 \; L^2 + 12 \\ L^2 + 6L + 12 \end{pmatrix} [c]$$

Equation (5.2.11) gives all of the possible quadratics which have their derivatives equal to their L_2 projections on the space of linear polynomials in an interval of length L.

From (5.1.5), the coefficients of the approximate general solution of the arbitrary differential equation (3.1.1) will be given by

$$y = A^{+}f + Nz \qquad (5.2.12)$$

Here, for a p'th order differential equation, z is in general a vector of p arbitrary constants. However, if an interpolation point happens to coincide with a singular point in the differential operation, the rank of the matrix A may be decreased by one and the number of arbitrary constants in (5.2.12) increased by one. In this case, it is best to add a boundary condition equation at the singular point to the differential equation in order to remove the undesired ambiguity.

A feature of the subroutines DIFFEQ and DIFF2D that has not yet been described is the fact that they call the subroutine NULL near the end of the programs. The effect of this call statement is to generate solutions of the form (5.2.12) for each of the matrix equivalents developed. Thus, DIFFEQ and DIFF2D not only evaluate matrix equivalents for arbitrary differential equations, but determine their general solutions as well.

5.3 PARTICULAR SOLUTIONS

One of the most important features of the analytic theory of differential equations is that particular solutions of differential equations may be obtained by adding boundary conditions to general solutions. This procedure will now be derived for the numerical solution of differential equations using the approximate general solution technique of the preceding section and equation (5.1.15).

In general, a boundary condition for a p'th order ordinary differential equation is an equation of the form

$$\sum_{k=1}^{B} B_k\, y(x_k) = g\,(x_o) \qquad (5.3.1)$$

where the $\{B_k\}$ are linear operators of order less than P, g is a given function and the $\{x_k\}$ are specified points in an interval I. If y is the projective solution of a differential equation, then it is given by

$$y = \sum_{i=o}^{n.} y_i\, b_i \qquad (5.3.2)$$

Therefore, (5.3.1) becomes

$$\sum_{k=1}^{B} \sum_{i=o}^{n} b_{k_i}\, y_i = g_o \qquad (5.3.3)$$

where

$$b_{k_i} = B_k\, b_i\,(x_k) \tag{5.3.4}$$

$$g_o = g(x_0) \tag{5.3.5}$$

The numbers b_{k_i} are easily evaluated by using the elementary differentiation and function matrices of Chapter 2.

If q independent boundary conditions are specified for a differential equation, then q independent equations of the kind (5.3.3) are produced. These may be written in matrix form as

$$By = g \tag{5.3.6}$$

In conjunction with the general soution (5.2.12), this yields, equation (5.1.15)

$$y = (A^+ - N(BN)^+BA^+)f + N(BN)^+g + NMu \tag{5.3.7}$$

where u has (p - q) arbitrary components. Thus, as long as it does not overspecify the problem, any number of boundary conditions may be added to the general solution (5.2.12) at one time. With each equation added, the vector of unknown coefficients becomes shorter until, when enough conditions are imposed, only a particular solution remains.

It is important to observe that any particular solution obtained from a general solution of a differential equation by

this procedure will satisfy both the differential equation and the boundary conditions, irrespective of what they might be. Hence, the locations of the points x_k where the boundary conditions (5.3.1) are applied is immaterial. If the point locations do not coincide with the endpoints of the interval, then the solution region will overlap the interval defined by the boundary conditions. However, this will not affect the validity or the accuracy of the particular solution within the confines of the specific boundary value problem.

This development is in sharp contrast to the solutions obtained by the finite element method or by ordinary projective methods in general where the solution region invariably coincides with the region defined by the boundary conditions. In fact, a large part of the difficulties and complications encountered in ordinary finite element analysis [22,23] is a direct result of defining unwieldy elements, such as the triangular, tetrahedral and iso-parametric elements, which are used to form complicated geometric shapes. With the necessity of matching boundary and solution regions removed, simple element shapes such as the rectangle and cube have an obvious advantage.

The procedure of specifying boundary conditions by restricting the behavior of the general solution of a differential equation is implemented for ordinary differential equations by the subroutine BOUND, shown in Figure 5.3.1. This subroutine accepts any number of arbitrary boundary conditions of the type (5.3.1) and generates particular solutions

Figure 5.3.1a

```
      SUBROUTINE BOUND(C,NDPTS,NONHOM,NULLD,NULLV,Y,NWBC,NR,ND,
     *     INSTR,CONST,XBC,NUM,XD,NELMTS,WT,LAMDA,RITEP)

      BOUND ADDS THE BOUNDARY CONDITIONS TO A GENERAL SOLUTION STORED
      IN THE ARRAY 'C'. ANY NUMBER OF BOUNDARY CONDITIONS MAY BE
      SPECIFIED; THE NUMBER OF SOLUTIONS FOUND IS DESIGNATED BY
      'NWBC'. THE BOUNDARY CONDITION INSTRUCTIONS NEED TO BE STORED
      IN THE ARRAY 'INSTR', STARTING IN LOCATION 'NUM'.

      DIMENSION C(1),Y(1),INSTR(1)CONST(1),XD(1),WT(1),XBC(1)
      DIMENSION IROW(100),JCOL(200),LOC(100),LPOWER(100),LROW(100),
     *     LFCN(100),B(100),OP(100),TEMP(100),OPS(1)
      COMMON IRO,JCOL,LOC,LPOWER,LROW,LFCN,B,OP,TEMP,OPS
      DOUBLE PRECISION DOUBLE,DOUB2
      LOGICAL LAMDA,RITEP
      NBCPT = 0
      NUL = NULLD
      NFREE = NULLD
      MSIZE = NUL + NONHOM
      IF(.NOT. LAMDA) GO TO 1
      NUL = 2*NULLD
      NFREE = NONHOM
      NDELTS = ND*NELMTS
    1 IEQ = 0

      DETERMINE IF THERE ARE MORE BOUNDARY CONDITIONS

    3 IF(INSTR(NUM) .NE. 4000) GO TO 4
      IF(IEQ .EQ. 0) RETURN
      GO TO 13
    4 NUM = NUM + 1

      FIRST THE COORDINATE-INDEPENDENT FACTORS OF THE CURRENT
      BOUNDARY CONDITION ARE FOUND

      CALL OPGEN1(INSTR,NUM,ND,ND,LOC,LD,LFCN,LF,OPS,LROW,LPOWER,OP,0)
      IF(LD .LE. 1 .AND. LF .LE. 1) GO TO 992
      IF(LF .LE. 1) GO TO 11
      NUM = IABS(LFCN(LF - 1)) + 1
   11 IF(LD .LE. 1) GO TO 12
      NUM = MAXO(NUM,IABS(LOC(LD - 1)) + 3)
   12 IEQ = IEQ + 1
      IF(IEQ .GT. 1) GO TO 32
      DO 31 I = 1,MSIZE
   31 Y(I) = 0.
   32 IF(LD .EQ. 0) GO TO 7
      LOPS = 1 + LROW(1)*ND
      DO 6 L = 1,LD,2
      IF(LOC(L) .LT. 0) GO TO 990

      THE ELEMENT NUMBER OF THE BOUNDARY CONDITION POINT IS 'NEL'

      NBCPT = NBCPT + 1
      I = 0
      DO 20 NEL = 1,NELMTS
      I = I + ND
      IF(XBC(NBCPT) .LE. XD(I)) GO TO 5
   20 CONTINUE

      NOW ADD THE COORDINATE-DEPENDENT FACTORS

      CALL OPGEN2(INSTR,CONST,LOC,LPOWER,LROW,L,ND,XD,WT,NEL,NPT,
     *          OPS,LOPS,OP,TEMP)
      XO = XD(NPT)
      MX = LROW(L)
      DO 21 I = 1,MX
   21 TEMP(I) = ALPHA(XBC(NBCPT),I,MX,XO,WT(NEL))

      THE BOUNDARY CONDITION OPERATOR IS STORED IN THE ARRAY 'B'

      IJ = - MX
      DO 23 J = 1,ND
      DOUBLE = 0.D0
      IJ = IJ + MX
      DO 22 I = 1,MX
   22 DOUBLE = DOUBLE + DBLE(TEMP(I)*OP(I + IJ))
   23 B(J) = SNGL(DOUBLE)
      IJ = IEQ - NUL
      KJ = NPT - 1
      IF(LAMDA) KJ = 2 * KJ
      KJ = KJ - NDPTS
      DO 6 J = 1,NONHOM
      IJ = IJ + NUL
      IJ1 = IJ + NULLD
      KJ = KJ + NDPTS
      KJ1 = KJ + ND

      THE CONDITIONS ON THE ARBITRARY CONSTANTS IN THE GENERAL
      SOLUTION ARE STORED IN THE ARRAY 'Y'

      DOUBLE = DBLE(Y(IJ))
      DOUB2 = DBLE(Y(IJ1))
      DO 14 K = 1,ND
      DOUBLE = DOUBLE + DBLE(B(K) * C(K + KJ))
      IF(.NOT. LAMDA) GO TO 14
      DOUB2 = DOUB2 + DBLE(B(K) * C(K + KJ1))
   14 CONTINUE
      Y(IJ) = SNGL(DOUBLE)
      IF(.NOT. LAMDA) GO TO 6
      Y(IJ1) = SNGL(DOUB2)
    6 CONTINUE
    7 IF(LF .EQ. 0 .OR. LAMDA) GO TO 9

      THE MATRIX IS AUGMENTED BY THE POINT VALUES OF THE BOUNDARY
      CONDITION FORCING FUNCTION

      DO 8 L = 1,LF,2
      NPT = 1
```

Figure 5.3.1b

```
      LOCL1 = IABS(LFCN(L))
      IF(INSTR(LOCL1) .LT. -1000) NPT = - INSTR(LOCL1) - 1000
      IJ = IEQ + NUL**2
    3 Y(IJ) = Y(IJ) + (FLOAT(LOCL1)/FLOAT(LFCN(L)))
     *                *FCN(XO(NPT),INSTR,CONST,LOCL1,LFCN(L + 1))
    9 IF(IEQ .LT. NULLD) GO TO 3
   13 IF(LAMDA) IEQ = NUL
      NWBC = NONHOM - IEQ
      CALL NULL(Y,NUL,NFREE,NONHOM,L,IROW,JCOL,B,NONHOM,NWBC,0)
      IF(L .NE. NUL .AND. LAMDA) GO TO 991

      THE PRODUCT 'C * B' IS EVALUATED

      IJ = - ND
      LO = NDELTS*NWBC
      LF = 1
      IF(LAMDA) LF = 2
      KJ = - NONHOM
      DO 10 J = 1,NWBC
      KJ = KJ + NONHOM
      L = 0
      DO 10 N = 1,NELMTS
      IJ = IJ + ND
      DO 10 LOCL1 = 1,LF
      DO 10 I = 1,ND
      L = L + 1
      IK = L - NDPTS
      DOUBLE = 0.D0
      DO 15 K = 1,NONHOM
      IK = IK + NDPTS
   15 DOUBLE = DOUBLE + DBLE(C(IK)*B(K + KJ))
      IF(LOCL1 .EQ. 1) GO TO 16
      Y(I + IJ + LO) =-SNGL(DOUBLE)
      GO TO 10
   16 Y(I + IJ) = SNGL(DOUBLE)
   10 CONTINUE
      IF(LAMDA .OR. .NOT. RITEP) RETURN

      IN NON-EIGENVALUE PROBLEMS, THE PARTICULAR SOLUTION IS PRINTED

      WRITE(6,100)
      CALL RITE(Y,NDPTS,NWBC,0,LAMDA)
      GO TO 1
  990 WRITE(6,201)
      ND = 0                                                            DECL5490
      RETURN                                                            DECL5500
  991 WRITE(6,202) L,IEQ                                                DECL5505
      ND = 0                                                            DECL5510
      RETURN                                                            DECL5515
  100 FORMAT(1H1/1HO,43X,'THE SOLUTION WITH BOUNDARY CONDITIONS')       DECL5520
  201 FORMAT(1-ERROR IN BOUNDARY CONDITION INSTRUCTIONS')               DECL5525
  202 FORMAT(1-THE CALCULATED RANK',I2,' DOES NOT EQUAL THE TRUE RANK', DECL5530
     *       I2)                                                        DECL5535
      END                                                               DECL5540
```

```
      FUNCTION ALPHA(X,I,ND,XO,WT)                                      DECL9210
C                                                                       DECL9220
C     THE ALPHA FUNCTION RETURNS THE VALUE AT X OF THE I'TH NEWTON-     DECL9230
C     COTES INTERPOLATION POLYNOMIAL (I = 1,ND) OF (ND - 1)'TH ORDER    DECL9240
C     (ND .LE. 10) IN AN INTERVAL OF LENGTH 1./WT STARTING AT XO.       DECL9250
C                                                                       DECL9260
      DIMENSION FAC(10)                                                 DECL9270
      DATA FAC / 1.,1.,2.,6.,24.,120.,720.,5040.,40320.,362880.         DECL9280
      Q = (ND - 1) * WT * ABS(X - XO)                                   DECL9290
      ALPHA = 1.0 / (FAC(I) * FAC(ND - I + 1))                          DECL9300
      IF(MOD(ND - I,2) .NE. 0) ALPHA = - ALPHA                          DECL9310
      DO 1 M = 1,ND                                                     DECL9320
      IF(M .EQ. I) GO TO 1                                              DECL9330
      ALPHA = ALPHA * (Q - (M - 1))                                     DECL9340
    1 CONTINUE                                                          DECL9350
      RETURN                                                            DECL9360
      END                                                               DECL9370
```

in the form (5.3.7) from the general solutions produced by the subroutine DIFFEQ. In defining the boundary operators B_k, BOUND uses the same instruction code as DIFFEQ and also relies on the subroutines OPGEN1 and OPGEN2 to generate their matrix equivalents.

The point locations x_k in equation (5.3.1) are specified in BOUND by adding an extra instruction at the end of the instructions for each simple operator. This instruction must have a value of $-(200 + n)$ where n is the location in the array XD of the coordinate of the point of application of the simple operator. This coordinate need not be equal to that of an interpolation node.

The principles developed in this section for the specification of boundary conditions are also applicable to partial differential equations in N-dimensional space, except that the points x_k in equation (5.3.1) need to be replaced by (N - 1)-dimensional surfaces. As discussed earlier, the logical solution region for these problems is the N-dimensional cube with the basis functions of equations (3.5.6) and (3.5.1). In this case, the number of linearly independent functions in the domain space of the equation is

$$N_d = \prod_{k=1}^{N} (n_k+1) \qquad (5.3.8)$$

Acting on this basis with a partial differential equation having only a p_i'th order differential operator in the x_i direction, yields a range space of dimension

$$N_r = (n_i - p_i + 1) \prod_{\substack{k=1 \\ k\neq i}}^{N} (n_k + 1) \qquad (5.3.9)$$

Consequently, the number of arbitrary constants in the solution is

$$N_n = p_i \prod_{\substack{k=1 \\ k\neq i}}^{N} (n_k + 1) \qquad (5.3.10)$$

This is exactly p_i times the number of interpolation nodes in an (N - 1) - dimensional surface orthogonal to the x_i direction. Therefore, if p_i boundary conditions are specified on this set of interpolation nodes, the particular solution of the differential equation will be unique.

In general, the dimensionality of the null space of an approximate general solution of a partial differential equation will be

$$N_n = \prod_{k=1}^{N} (n_k + 1) - \prod_{k=1}^{N} (n_k - p_k + 1) \qquad (5.3.11)$$

where p_k is the order of the differential equation in the x_k direction. Therefore, a unique solution of a differential equation will result when N_n linearly independent equations are added to its general solution. This is most readily accomplished by enforcing the boundary conditions at a sufficient number of distinct points in each of the (N - 1)- dimensional subregions.

The subroutine BOUN20 shown in Figure 5.3.2 performs this boundary condition procedure for two-dimensional partial dif-

Figure 5.3.2

```
      SUBROUTINE BOUN2D(C,NDPTS,NONHOM,NULLD,NULLV,Y,NWBC,NRX,NRY,ND,     DE2D2190
     *     INSTR,CONST,NUM,XD,YD,NELMTS,WTX,WTY,LAMDA,RITEP,YI)           DE2D2195
C                                                                        DE2D2200
C     BOUN2D ADDS THE BOUNDARY CONDITIONS TO A TWO-DIMENSIONAL           DE2D2205
C     GENERAL SOLUTION OF A PARTIAL DIFFERENTIAL EQUATION STORED IN THE  DE2D2210
C     ARRAY 'C'.                                                         DE2D2215
C                                                                        DE2D2220
      DIMENSION C(1),Y(1),INSTR(1),CONST(1),XD(1),YD(1),WTX(1),WTY(1),   DE2D2225
     *     YI(1)                                                         DE2D2230
      DIMENSION IROW(100),JCOL(200),LOC(100),LPOWER(100),LROW(100),      DE2D2235
     *     LFCN(100),B(4096),OP(4096),TEMP(4096),OPS(1)                  DE2D2240 C
      NDSQ = ND ** 2                                                     DE2D2245 C
C                                                                        DE2D2250 C
C                          DECL 4810-4905                                DE2D2255
C                                                                        DE2D2260 C
      CALL OPG2D1(INSTR,NUM,ND,ND,ND,LOC,LO,LFCN,LF,OPS,LROW,            DE2D2265 C
     *     LPOWER,OP,TEMP,0)                                             DE2D2270 C
C                                                                        DE2D2275
C                          DECL 4915-4960                                DE2D2280 C
C                                                                        DE2D2285 C
      LOPS = 0                                                           DE2D2290 C
      DO 17 L = 1,LO,2                                                   DE2D2295
C                                                                        DE2D2300
C                          DECL 4975-5005                                DE2D2305 C
C                                                                        DE2D2310 C
      I = I + NDSQ                                                       DE2D2315 C
      IF(XBC(NBCPT) .GE. XD(I) .AND. XBC(NBCPT) .LE. XD(I + NDSQ)        DE2D2320
     *     .AND. XBC(NBCPT) .GE. YD(I) .AND. YBC(NBCPT) .LE. YD(I + NDSQDE2D2325
     *     )) GO TO 5                                                    DE2D2330
   20 CONTINUE                                                           DE2D2335 C
    5 CALL OPG2D2(INSTR,CONST,LOC,LPOWER,LROW,L,ND,XD,YD,WTX,WTY,        DE2D2340 C
     *     NEL,KJ,OPS,LOPS,OP,TEMP)                                     DE2D2345 C
      XO = XD(KJ)                                                        DE2D2350
      YO = YD(KJ)                                                        DE2D2355
      WX = WTX(NEL)                                                      DE2D2360
      WY = WTY(NEL)                                                      DE2D2365
      MX = LROW(L)                                                       DE2D2370
      MY = LROW(L + 1)                                                   DE2D2375
      K = 0                                                              DE2D2380
      DO 5 I = 1,MX                                                      DE2D2385
      VALUE = ALPHA(XD(NPT),I,MX,XO,WX)                                  DE2D2390
      DO 5 J = 1,MY                                                      DE2D2395
      K = K + 1                                                          DE2D2400
    5 YI(K) = VALUE * ALPHA(YD(NPT),J,MY,YO,WY)                          DE2D2405
      IJ = - K                                                           DE2D2410
      DO 30 J = 1,NDSQ                                                   DE2D2415
      DOUBLE = 0.00                                                      DE2D2420
      IJ = IJ + K                                                        DE2D2425
      DO 6 I = 1,K                                                       DE2D2430
    6 DOUBLE = DOUBLE + DBLE(YI(I) * OP(I + IJ))                         DE2D2435
   30 B(J) = SNGL(DOUBLE)                                                DE2D2440
      IJ = IEQ - NUL                                                     DE2D2445
      KJ = KJ - 1                                                        DE2D2450
      IF(LAMDA) KJ = 2 * KJ                                              DE2D2455
      KJ = KJ - NDPTS                                                    DE2D2460
      KJ1 = KJ + NDPTS/2                                                 DE2D2465
      DO 17 J = 1,NONHOM                                                 DE2D2470
      IJ = IJ + NUL                                                      DE2D2475
      IJ1 = IJ + NULLD                                                   DE2D2480
      KJ = KJ + NDPTS                                                    DE2D2485
      KJ1 = KJ1 + NDPTS                                                  DE2D2490
      DOUBLE = DBLE(Y(IJ))                                               DE2D2495
      DOUB2 = DBLE(Y(IJ1))                                               DE2D2500
      DO 14 K = 1,NDSQ                                                   DE2D2505
C                                                                        DE2D2510
C                          DECL 5195-5220                                DE2D2515
C                                                                        DE2D2520
      IF(.NOT. LAMDA) GO TO 17                                           DE2D2525
C                                                                        DE2D2530
C                          DECL 5230-5290                                DE2D2535
C                                                                        DE2D2540
     *   + FCN2D(XD(NPT),YD(NPT),INSTR,CONST,LOCL1,LFCN(L + 1))          DE2D2545
C                                                                        DE2D2550
C                          DECL 5300-5335                                DE2D2555
C                                                                        DE2D2560
      IJ = - NDSQ                                                        DE2D2565
      LO = NWBC * NELMTS * NDSQ                                          DE2D2570
C                                                                        DE2D2575
C                          DECL 5345-5375                                DE2D2580
C                                                                        DE2D2585
      IJ = IJ + NDSQ                                                     DE2D2590
      DO 16 LOCL1 = 1,LF                                                 DE2D2595
      DO 10 I = 1,NDSQ                                                   DE2D2600
C                                                                        DE2D2605
C                          DECL 5400-5535                                DE2D2610
C                                                                        DE2D2615
      END                                                                DE2D2620
```

ferential equations. In most operations, BOUN2D is similar to BOUND. However, BOUN2D requires two coordinates for every boundary condition point location and these are stored in the arrays XD and YD.

Finally, note that if equation (5.3.6) is solved directly, it yields

$$y = B^{+} g + N_B w \qquad (5.3.12)$$

This equation defines a function space in which every function satisfies the boundary conditions. Hence, matrix generalized inversion can be used to advantage in conventional projective methods where such function spaces are often desired. In particular, many finite element formulations result in a matrix equation of the form [22]

$$Ay = f \qquad (5.3.13)$$

where A is a symmetric matrix. Consequently, inserting (5.3.12) into (5.3.13) and pre-multiplying by N_B^T to retain symmetry of the coefficient matrix, gives

$$N_B^T A N_B w = N_B^T f - N_B^T A B^{+} g \qquad (5.3.14)$$

When solved for w, (5.3.14) yields, in conjunction with (5.3.12), a solution of the boundary value problem which satisfies the boundary conditions (5.3.6).

The above process is generally performed implicitly in the finite element method [22] by numbering adjacent points with the same number and by eliminating rows and columns in the coefficient matrix in order to specify Dirichlet boundary conditions. Equation (5.3.14) explains why the procedures work and extends them to problems involving mixed boundary condition.

5.4 SOLUTION IN MULTIPLE REGIONS

Since the accuracy of numerical approximations such as (5.2.11) increases as the size of the interval used decreases, a basic method for obtaining highly accurate solutions of differential equations over a large region is to determine the general solution of the equation in several adjacent sub-regions or elements and then to tie these solutions together with continuity conditions. Provided that the continuity requirements are sufficiently stringent and of the right number, the composite solution obtained in this way will provide a good approximation of the general solution over the entire region.

In forming such composite solutions for a differential equation, one must be aware that they are, in general, not optimal in the sense of definition (3.1.3). This is because the projection matrices in a multi-regional solution are designed to minimize the residual function in each element separately and not in the overall region. If it is desired, the optimal approximate solution of a differential equation can be determined by evaluating the projection matrix (2.3.9) for every multi-regional combination of basis functions used. However, this procedure is not recommended as a practical solution technique for differential equations because the evaluation of complicated projection matrices requires considerable computation while the projection matrices for individual elements are pre-calculated. Thus, in most cases, given equal amounts of computer time, a much more accurate solution will result if in the place of evaluating fancier projection matrices, more ele-

ments are used.

In many respects, these ideas are similar to those encountered in ordinary finite element analysis. The main difference is that while in the finite element method continuity conditions are usually satisfied implicitly by using compatible elements, with the piecewise general solution procedure, the requirements are best imposed explicitly. The superiority of using overall conjugate approximation basis functions to local ones has, however, already been established in finite element analysis by Brauchli and Oden [29].

The most obvious method of obtaining a consistent set of continuity requirements for the overall solution of a p'th order differential equation is to set the solution and its $(n - 1)$'st derivatives equal across inter-element boundaries. This procedure has the effect of transforming the distinct sets of interpolatory basis functions in each of the elements into a single set of spline-like functions in the overall region. Therefore, for an ordinary p'th order differential equation that has had its solution region divided into m elements, the following $p(m - 1)$ boundary conditions are imposed

$$\frac{d^i}{dx^i}\, y(x_n^s) = \frac{d^i}{dx^i}\, y(x_0^{s-1}) \qquad \begin{matrix} 0 \le i \le p-1 \\ 1 \le s \le m-1 \end{matrix} \qquad (5.4.1)$$

where the index s designates the element from which the approximate value of y should be taken. When assembled for the entire region, these conditions yield the matrix equation

$$Cy = 0 \qquad (5.4.2)$$

where C is a $p(m-1) \times (mn)$ matrix with components

$$C_{ij} = \begin{cases} D_{nj}^{(i)} & \text{if } j \text{ is in element } s \\ -D_{ij}^{(i)} & \text{if } j \text{ is in element } s+1 \\ 0 & \text{otherwise} \end{cases} \qquad (5.4.3)$$

Together with (5.2.12), this gives, from (5.1.15)

$$y = A^{+}f - NC(CN)^{+} CA^{+}f + NMu \qquad (5.4.4)$$

where M is a $pm \times p$ matrix of the nullvectors of C and u is an arbitrary p component vector. Equation (5.4.4) gives the general solution of the differential equation, including an approximate inhomogeneous solution and p approximate homogeneous solutions, each of which is C^{p-1} continuous in the entire region.

The continuity conditions (5.4.1) are closely associated with the theory of spline approximation [57]. In this theory, interpolating polynomials are defined in which some of the function specifying parameters are the values of the derivatives of the function at the endpoints of the interval. For example, in the case of cubic splines, the four parameters defining the polynomial are the function and derivative values at the endpoints of the interval. When applied to approximating functions, derivative continuity of cubic splines are established by setting adjacent derivative parameters equal, leaving two free parameters in each interval.

The functions corresponding to the solution (5.4.4) also

satisfy the continuity conditions (5.4.1) at the edges of elements and are, therefore, similar to spline functions. In fact, when both sets of functions are n'th order polynomials, they span identical function spaces, although their representations are different. In the case of cubic equi-spaced Lagrangean interpolation polynomials, for example, the four function specifying parameters are the nodal point values $y_i = y(x_i)$, $x_i = x_o + (L/3)i$, i=0,1,2,3. When this polynomial is required to satisfy derivative continuity at both ends, the nodal values will be given by

$$y = Gu \qquad (5.4.5)$$

where G is a 4 x 2 matrix and u is an arbitrary two component vector. Here, as with the cubic splines, there are two independent parameters in each element, although in this case the components of u do not equal the values of the function at the endpoints of the interval.

The advantage of using the second type of polynomial instead of the first, is that the equi-spaced Lagrangean interpolation polynomials, as well as the procedures in equations (5.4.2) to (5.4.4), are both simple to define algebraically and easy to work with computationally - even in high order cases. By comparison, the calculation and use of general high order spline functions requires exceedingly complicated algebraic manipulation [57].

As an illustration of the procedure, suppose that the differential equation (5.2.9) is solved in n consecutive intervals

of length L. In the i'th interval, the solution is given by (5.2.11) with $c = c_i$. According to (5.4.2), in this case CN=0 and this yields

$$\begin{bmatrix} b & -a & & & 0 \\ & b & -a & & \\ & & \circ & \circ & \\ & & & \circ & \circ \\ 0 & & & & \circ \\ & & & & b & -a \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \circ \\ \circ \\ \circ \\ c_n \end{bmatrix} = \begin{bmatrix} \\ 0 \\ \\ \end{bmatrix} \qquad (5.4.6)$$

where

$$\begin{aligned} a &= L^2 - 6L + 12 \\ b &= L^2 + 6L + 12 \end{aligned} \qquad (5.4.7)$$

Consequently

$$c_i = \left(\frac{b}{a}\right)^{i-1} c_1 \qquad (5.4.8)$$

and the entire solution contains only one arbitrary constant.

The solution (5.2.11) with (5.4.8) can be shown to converge to the exact solution of (5.2.9) by observing that

$$\lim_{\substack{n\to\infty \\ L\to 0}} c_n = 1 + nL + \frac{n^2L^2}{2!} + \frac{n^3L^3}{3!} + \ldots = e^{nL} \qquad (5.4.9)$$

Thus the value of the approximate solution at the third point in the n'th element is

Figure 5.4.1a

```
      SUBROUTINE CONTIN(DN,NR,ND,MDN,NDN,C,NDPTS,NONHOM,NULLD,NULLV,
     *     NELMTS,WT,LAMDA,RITEG)
C
C     CONTIN TAKES THE GENERAL SOLUTION OF A DIFFERENTIAL EQUATION
C     IN ' NELMTS' SEPARATE REGIONS AND ESTBLISHES CONTINUITY OF THE
C     FUNCTION AND ITS (NULLD - 1) DERIVATIVES ACCROSS THE ELEMENT
C     BOUNDARIES. THE CONTINUOUS SOLUTION IS RETURNED IN THE 'NDPTS'
C     BY 'NONHOM' MATRIX 'C', WHERE 'NDPTS' IS THE NUMBER OF ELEMENTS
C     IN THE SOLUTION VECTOR AND 'NONHOM' IS THE NUMBER OF SOLUTIONS.
C
      DIMENSION DN(1),C(1),WT(1)
      DIMENSION IROW(300),JCOL(300),OP(1)
      COMMON IROW,JCOL,OP
      DOUBLE PRECISION DOUBLE,DOUB2
      LOGICAL LAMDA,RITEG
      IGEN = 1
      IF(LAMDA) IGEN = 2
      NPEREL = NULLD
      IF(LAMDA) NPEREL = NULLV
      NUNOWN = NPEREL*NELMTS
C
C     THE CONTINUITY CONDITIONS GENERATED ARE STORED IN THE 'MC' BY
C     'NC' MATRIX 'C'
C
      MC = IGEN*NULLD*(NELMTS - 1)
      NC = NUNOWN + 2 - IGEN
      NONHOM = NC - MC
      N = MC*NC
      DO 3 K = 1,N
    3 C(K) = 0.
      I = 0
      IF(NELMTS .EQ. 1) GO TO 28
      N = ND**2
C
C     THE DIFFERENTIATION MATRIX IS STORED IN 'OP'
C
      DO 1 K = 1,N
    1 OP(K) = 0.
      J = ND + 1
      DO 2 K = 1,N,J
    2 OP(K) = 1.
      IEO = 1
      IPOW = 0
      NCC = ND
      JOP = NUNDON*MC
C
C     IN EIGENVALUE PROBLEMS, THE CONTINUITY CONDITIONS ARE IMPOSED
C     TWICE
C
      DO 17 L = 1,IGEN
      I = I - 1
      JDN = - MDN
      IF(L .EQ. 2) JDN = - ND
      JC = - MC
C
C     EACH ELEMENT IS TAKEN SEPARATELY
C
      DO 17 N = 1 ,NELMTS
      I = I + 1
      WEIGHT = WT(N)**IPOW
      DO 6 J = 1,NDN
      JDN = JDN + MDN
      IF(.NOT. LAMDA .AND. J .EQ. NDN) GO TO 24
      JC = JC + MC
      IJ = I + JC
      GO TO 25
   24 IJ = I + JCP
   25 IJ1 = IJ + 1
      K1 = 1 - NCC
      K2 = 0
C
C     THE FUNCTION AND THE APPROPRIATE DERIVATIVES ARE EVALUATED
C     AT BOTH ENDS OF EACH ELEMENT
C
      DOUBLE = DBLE(C(IJ))
      DOUB2 = DBLE(C(IJ1))
      DO 12 K = 1,ND
      K1 = K1 + NCC
      K2 = K2 + NCC
      DNW = DN(K + JDN)*WEIGHT
      IF(N .EQ. 1) GO TO 20
      DOUBLE = DOUBLE + DBLE(OP(K1)*DNW)
   20 IF(N .EQ. NELMTS) GO TO 12
      DOUB2 = DOUB2 - DBLE(OP(K2)*DNW)
   12 CONTINUE
      IF(N .EQ. 1) GO TO 5
      C(IJ) = SNGL(DOUBLE)
    5 IF(N .EQ. NELMTS) GO TO 6
      C(IJ1) = SNGL(DOUB2)
    6 CONTINUE
   17 CONTINUE
      IJ = NCC
      NCC = NCC - 1
      IF(NCC .EQ. NR) GO TO 28
C
C     IN EACH PASS, THE ORDER OF THE DIFFERENTIATION MATRIX IS
C     INCREASED BY ONE
C
      CALL OPRATR(OP,IEO,NCC,IJ,ND,1)
      IPOW = IPOW + 1
      GO TO 4
   19 CALL NULL(C,MC,NUNOWN,NC,IRANK,IROW,JCOL,OP,NUNOWN,NONHOM,0)
      IF(IRANK .NE.MC) GO TO 990
      IPOW = NDPTS*NONHOM
      DO 7 I = 1,IPOW
    7 C(I) = 0.
      NCC = NDN*JDN
      IF(LAMDA) GO TO 9
```

Figure 5.4.1b

```
C
C     USING THE EQUATION 'C(LAST COLUMN) = DN(LAST COLUMN)' THE
C     THE INHOMOGENEOUS SOLUTION IS EVALUATED
C
      IJ = NDPTS*NPEREL
      JDN = MDN*NPEREL- NCC
      DO 8 N = 1,NELMTS
      JDN = JDN + NCC
      DO 8 I = 1,MDN
      IJ = IJ + 1
    8 C(IJ) = DN(I + JDN)
      IPOW = NDPTS*NR + MDN
      IJ = NDPTS*NULLV - IPOW
      JDN = NDN*NULLV - NCC
      DO 10 N = 1,NELMTS
      IJ = IJ + IPOW
      JDN = JDN + NCC
      IJ1 = IJ - NDPTS
      IJDN = JDN
      DO 10 J = 1,NR
      IJ1 = IJ1 + NDPTS
      DO 10 I = 1,MDN
      IJDN = IJDN + 1
   10 C(I + IJ1) = DN(IJDN)
    9 IC = C
C
C     USING THE EQUATION 'C = DN * OP,  (OP = C+)', THE HOMOGENEOUS
C     SOLUTION IS EVALUATED
C
      JDN = - NCC
      IPOW = - NPEREL
      DO 11 N = 1,NELMTS
      JDN = JDN + NCC
      IPOW = IPOW + NPEREL
      DO 11 I = 1,MDN
      IJDN = I + JDN
      IC = IC + 1
      IEO = IPOW - NONDAN
      IJ = IC - NDPTS
      DO 11 J = 1,NONHOM
      IJ = IJ + NDPTS
      IEO = IEO + NONDAN
      IJ1 = IJDN - MDN
      DOUBLE = DBLE(C(IJ))
      DO 13 K = 1,NPEREL
      IJ1 = IJ1 + MDN
   13 DOUBLE = DOUBLE + DBLE(DN(IJ1)*OP(K + IEO))
   11 C(IJ) = SNGL(DOUBLE)
      GO TO 14
   28 IF(I .GT. 0) GO TO 19
C
C     IF THERE IS ONLY ONE ELEMENT, THE SOLUTION IS SIMPLY TRANSFERED
C     FROM DN TO C
C
      IJ = MDN*NDN
      DO 18 I = 1,IJ
   18 C(I) = DN(I)
   14 WRITE(6,100)
      IF(LAMDA .OR. .NOT. RITEG) RETURN
C
C     THE GENERAL SOLUTION IS PRINTED
C
      CALL RITE(C,NDPTS,NONHOM,0,LAMDA)
      RETURN
  990 WRITE(6,200) IRANK,NM
      ND = 0
      RETURN
  100 FORMAT(1H1/1H0,54X,'THE GENERAL SOLUTION')
  200 FORMAT(1-THE CALCULATED RANK',I4,'DOES NOT EQUAL THE TRUE RANK',
     *           I4)
      END
```

$$\lim_{\substack{n \to \infty \\ L \to 0}} y_{3n} = c_1 e^x \qquad (5.4.10)$$

where $x = nL$.

The subroutine DIFFEQ in Figure 4.2.4 has been designed with multi-element solutions in mind. The program calculates the matrix equivalent of a differential equation in NELMTS consecutive intervals and evaluates the general solution of the equation in each of these separately. The continuous general solution is produced from these sectional solutions by the subroutine CONTIN shown in Figure 5.4.1. By calling DIFFEQ and CONTIN in succession, highly accurate, continuous multi-element general solutions of ordinary differential equations can be obtained.

The procedure of imposing continuity requirements on multi-element solutions of partial differential equations is very similar to that used with ordinary differential equations except that it must be done repeatedly and in different directions. For two-dimensional problems, the region is divided into matching rectangles in each of which the interpolation polynomials are of the same degree. The derivative continuity conditions (5.4.1) are imposed along each line of interpolation nodes in both the x and the y directions. These must be sufficient to satisfy the continuity requirements of the partial differential equation in a direction normal to the edge of a rectangle. Derivatives tangent to an edge are, of course, equal whenever the function is made equal on both sides of the edge. The degree to which cross derivatives are continuous is given in the following theorem.

THEOREM 5.4.1. Let two functions $z(x,y)$ and $w(x,y)$ be given by

$$z(x,y) = \sum_{i=o}^{m} \sum_{j=o}^{n} Z_{ij}\, l_i^{(m)}(x)\, l_j^{(n)}(y) \qquad \begin{matrix} a \le x \le b \\ d \le y \le e \end{matrix} \tag{5.4.11a}$$

$$w(x,y) = \sum_{i=o}^{m} \sum_{j=o}^{n} W_{ij}\, l_i^{(m)}(x)\, l_j^{(n)}(y) \qquad \begin{matrix} b \le x \le c \\ d \le y \le e \end{matrix} \tag{5.4.11b}$$

where $l_i^{(m)}$ is defined in (3.5.2). Then, whenever

$$\left.\frac{\partial^i z}{\partial x^i}\right|_{x=b} = \left.\frac{\partial^i w}{\partial x^i}\right|_{x=b} \qquad i=o,\ldots,p \tag{5.4.12}$$

the cross derivatives $\dfrac{\partial^{p+q}}{\partial x^p\, \partial y^q}$ and $\dfrac{\partial^{p+q}}{\partial y^q\, \partial x^p}$ are continuous across the line $x=b$

$$\left.\frac{\partial^{p+q} z}{\partial x^p\, \partial y^q}\right|_{x=b} = \left.\frac{\partial^{p+q} w}{\partial x^p\, \partial y^q}\right|_{x=b} \qquad q=o,\ldots,n \tag{5.4.13a}$$

$$\left.\frac{\partial^{p+q} z}{\partial y^q\, \partial x^p}\right|_{x=b} = \left.\frac{\partial^{p+q} w}{\partial y^q\, \partial x^p}\right|_{x=b} \qquad q=o,\ldots,n \tag{5.4.13b}$$

PROOF: The elements of the bottom row of the matrix form of $u = \dfrac{\partial^p}{\partial x^p}\left(\dfrac{\partial^q z}{\partial y^q}\right)$ are

$$U_{m-p,j} = \sum_{k=o}^{m} \sum_{l=o}^{n} D^{(p)}_{m-p,k} Z_{kl} D^{T(q)}_{lj} \tag{5.4.14}$$

where $D^{(p)}$ is the p'th order differentiation matrix. Similarly, the elements of the top row of the matrix form of $v = \frac{\partial^p}{\partial x^p}\left(\frac{\partial^q w}{\partial y^q}\right)$ are

$$V_{1j} = \sum_{k=o}^{m} \sum_{l=o}^{n} D^{(p)}_{1k} \; W_{kl} \; D^{T(q)}_{lj} \qquad (5.4.15)$$

Hence

$$\begin{aligned} r_j &= U_{m-p,j} - V_{1j} \\ &= \sum_{k=o}^{m} \sum_{l=o}^{n} \left\{ D^{(p)}_{m-p,k} \; Z_{kl} - D^{(p)}_{1k} \; W_{kl} \right\} D^{T(q)}_{lj} \end{aligned} \qquad (5.4.16)$$

Now (5.4.12) implies that

$$\sum_{k=o}^{m} D^{(p)}_{m-p,k} \; Z_{kl} = \sum_{k=o}^{m} D^{(p)}_{1k} \; W_{kl} \qquad (5.4.17)$$

so that $r_j = 0$ and (5.4.13a) is true. The proof of (5.4.13b) is similar.

As was the case with DIFFEQ, the two-dimensional version of the program DIFF2D also computes the general solution of a differential equation in NELMTS separate elements. These must be placed in a matching pattern as described earlier. These elemental general solutions are then made continuous by the subroutine CONT2D shown in Figure 5.4.2. CONT2D imposes C^{p-1} continuity on the solution of a partial differential equation in the x direction and C^{q-1} continuity in the y direction, where p and q are the highest orders of the x and y derivatives

Figure 5.4.2

```
      SUBROUTINE CONT2D(DN,NRX,NRY,ND,MDN,NDN,C,NDPTS,NONHOM,NULLV,     DE2D1755
     *     NCCOND,IELCX,IELCY,NELMTS,WTX,WTY,LAMDA,RITEG)               DE2D1760
C                                                                       DE2D1765
C     CONT2D TAKES THE GENERAL SOLUTION OF A TWO-DIMENSIONAL PARTIAL    DE2D1770
C     DIFFERENTIAL EQUATION IN (NELMTS) SEPARATE REGIONS AND ESTABLISHESDE2D1775
C     CONTINUITY OF THE FUNCTION AND ITS DERIVATIVES ACROSS THE         DE2D1780
C     ELEMENT BOUNDARIES.                                               DE2D1785
C                                                                       DE2D1790
      DIMENSION DN(1),C(1),WTX(1),WTY(1),IELCX(1),IELCY(1)              DE2D1795
C                                                                       DE2D1800
C                             DECL 3385-3410                            DE2D1805
C                                                                       DE2D1810
      LOGICAL LAMDA,FIRST,LAST,RITEG                                    DE2D1815
      NPEREL = ND ** 2 - NRX * NRY                                      DE2D1820
C                                                                       DE2D1825
C                             DECL 3420-3445                            DE2D1830
C                                                                       DE2D1835
      MC = IGEN * NCCOND                                                DE2D1840
      NC = NUNDWN + 2 - IGEN                                            DE2D1845
      MNDN = MDN * NDN                                                  DE2D1850
      MCNDN = MC * NDN                                                  DE2D1855
C                                                                       DE2D1860
C                             DECL 3460-3485                            DE2D1865
C                                                                       DE2D1870
      NR = NRX                                                          DE2D1875
      LINADD = 1                                                        DE2D1880
      KADD = ND                                                         DE2D1885
      GO TO 31                                                          DE2D1890
   30 NR = NRY                                                          DE2D1895
      LINADD = ND                                                       DE2D1900
      KADD = 1                                                          DE2D1905
   31 ICONT = 0                                                         DE2D1910
C                                                                       DE2D1915
C                             DECL 3490-3580                            DE2D1920
C                                                                       DE2D1925
      ISTART = 1                                                        DE2D1930
      IF(L .EQ. 2) ISTART = 1 + MDN / 2                                 DE2D1935
   32 FIRST = .TRUE.                                                    DE2D1940
      LAST = .FALSE.                                                    DE2D1945
   33 ICONT = ICONT + 1                                                 DE2D1950
      IF(KADD .EQ. 1) GO TO 34                                          DE2D1955
      N = IELCX(ICONT)                                                  DE2D1960
      IF(N .LT. 0) GO TO 30                                             DE2D1965
      IF(N .EQ. 0) GO TO 32                                             DE2D1970
      N1 = IELCX(ICONT + 1)                                             DE2D1975
      WEIGHT = WTX(N) ** IPOW                                           DE2D1980
      GO TO 35                                                          DE2D1985
   34 N = IELCY(ICONT)                                                  DE2D1990
      IF(N .LT. 0) GO TO 17                                             DE2D1995
      IF(N .EQ. 0) GO TO 32                                             DE2D2000
      N1 = IELCY(ICONT + 1)                                             DE2D2005
      WEIGHT = WTY(N) ** IPOW                                           DE2D2010
   35 IF(N1 .EQ. 0) LAST = .TRUE.                                       DE2D2015
      N1 = N - 1                                                        DE2D2020
      JCADD = MCNDN * N1                                                DE2D2025
      NDNA = ISTART + MNDN * N1                                         DE2D2030
      JLINE = NDNA - LINADD                                             DE2D2035
      DO 6 NLINE = 1,ND                                                 DE2D2040
      JLINE = JLINE + LINADD                                            DE2D2045
      JDN = JLINE - MDN                                                 DE2D2050
      JC = JCADD - MDN                                                  DE2D2055
      I = I + 1                                                         DE2D2060
C                                                                       DE2D2065
C                             DECL 3630-3705                            DE2D2070
C                                                                       DE2D2075
      KJ = JDN - KADD                                                   DE2D2080
C                                                                       DE2D2085
C                             DECL 3710-3720                            DE2D2090
C                                                                       DE2D2095
      KJ = KJ + KADD                                                    DE2D2100
      DNW = DN(KJ) * WEIGHT                                             DE2D2105
      IF(FIRST) GO TO 20                                                DE2D2110
      DOUBLE = DOUBLE + DBLE(DP(K1)*DNW)                                DE2D2115
   20 IF(LAST) GO TO 12                                                 DE2D2120
      DOUB2 = DOUB2 - DBLE(DP(K2)*DNW)                                  DE2D2125
   12 CONTINUE                                                          DE2D2130
      IF(FIRST) GO TO 5                                                 DE2D2135
      C(IJ) = SNGL(DOUBLE)                                              DE2D2140
    5 IF(LAST) GO TO 6                                                  DE2D2145
      C(IJ1) = SNGL(DOUB2)                                              DE2D2150
    6 CONTINUE                                                          DE2D2155
      FIRST = .FALSE.                                                   DE2D2160
      GO TO 33                                                          DE2D2165
C                                                                       DE2D2170
C                             DECL 3780-4215                            DE2D2175
C                                                                       DE2D2180
      END                                                               DE2D2185
```

in the equation, respectively.

5.5 EIGENVALUE PROBLEMS

The procedures developed thus far are not capable of solving eigenvalue problems. These are problems of the form

$$D_1 y = k D_2 y \qquad (5.5.1)$$

that are accompanied by homogeneous boundary conditions. The difficulty stems from the fact that the constant k in this equation is not determined until all of the boundary conditions are specified. Therefore, it is not possible to treat differential equation eigenvalue problems independently from the boundary conditions with a numerical method and only particular solutions can be obtained directly. Of course, if a complete set of independent particular solutions of a differential equation are determined, together they constitute its general solution.

Despite the impossibility of separating the differential equation (5.5.1) from its boundary conditions, it is still extremely desirable to evaluate each contribution in the discretization process independently. A simple but effective device for developing such a procedure is to write the matrix equivalent of (5.5.1) in each element as

$$Az = 0 \tag{5.5.2}$$

where A is the m x2n matrix

$$A = [\, D_1 \,\vdots\, D_2 \,] \tag{5.5.3}$$

$$z = \begin{pmatrix} y \\ --- \\ -ky \end{pmatrix} \tag{5.5.4}$$

Equation (5.5.2) can be easily solved for z using (5.1.5)

$$z = Nu \tag{5.5.5}$$

where u has $(2n - m)$ arbitrary elements. The $(n - m)$ continuity and boundary conditions can then be imposed on both the top and the bottom halves of z. Since these conditions are all homogeneous, they will reduce the number of arbitrary constants in (5.5.5) to m but will not alter its form:

$$z = Mw \tag{5.5.6}$$

Consequently, setting the bottom half of z equal to $-k$ times the top half results in the rectangular matrix eigenvalue problem

$$M_b w = -k \; M_t \; w \tag{5.5.7}$$

where M_b and M_t are n x m matrices of rank m. Thompson and Weil have proved [58] that the rank reducing numbers of this

equation are identical to the eigenvalues of the standard, square matrix eigenvalue equation

$$Lw = -kw \tag{5.5.8}$$

where L is the $m \times m$ matrix

$$L = M_t^+ M_b \tag{5.5.9}$$

Equation (5.5.8) may be solved by using standard techniques to determine the eigenvalues $-k$ and the eigenvectors w. Then the vector y is found by using (5.5.6) and (5.5.4).

A simple problem will illustrate these properties. Consider the boundary value problem

$$\frac{d^2y}{dx^2} = -\lambda^2 y$$
$$y(0) = 0 \qquad \frac{dy}{dx}(1) = 0 \tag{5.5.10}$$

For the sake of simplicity, let y be approximated by a single quadratic element. Equation (5.5.10) becomes

$$\begin{bmatrix} 4 & -8 & 4 \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = -\lambda^2 \begin{pmatrix} \frac{1}{6} & \frac{4}{6} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{5.5.11}$$

The scheme is to solve the equation

$$\begin{bmatrix} 1 & -2 & 1 & 1 & 4 & 1 \end{bmatrix} z = 0 \tag{5.5.12}$$

with

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \frac{24}{\lambda^2} \begin{pmatrix} z_4 \\ z_5 \\ z_6 \end{pmatrix} \qquad (5.5.13)$$

The solution of (5.5.12) is

$$z = \begin{pmatrix} 2 & 1 & -1 & -4 & -1 \\ 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} u \qquad (5.5.14)$$

where u is a vector of five unknowns. The boundary conditions imply that

$$z_1 = [\ 2 \quad -1 \quad -1 \quad -4 \quad -1\ ]\, u = 0 \qquad (5.5.15a)$$

$$z_4 = [\ 0 \quad 0 \quad 1 \quad 0 \quad 0\ ]\, u = 0 \qquad (5.5.15b)$$

$$\frac{dz_3}{d\varpi} = [\ 1 \quad -4 \quad 3\ ] \begin{pmatrix} 2 & -1 & -1 & -4 & -1 \\ 1 & & & & \\ & 1 & & & \end{pmatrix} u = [\ -2 \quad 2 \quad -1 \quad -4 \quad -1\ ]\, u = 0 \qquad (5.5.15c)$$

$$\frac{dz_6}{d\varpi} = [\ 1 \quad -4 \quad 3\ ] \begin{pmatrix} 0 & 0 & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} u = [\ 0 \quad 0 \quad 1 \quad -4 \quad 3\ ]\, u = 0 \qquad (5.5.15d)$$

Thus

$$u = \begin{pmatrix} .6 \\ 8 \\ 0 \\ 3/4 \\ 1 \end{pmatrix} w \qquad (5.5.16)$$

and (5.5.14) becomes

(5.5.17)

$$z = \begin{pmatrix} 2 & -1 & -1 & -4 & -1 \\ 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} \begin{pmatrix} 6 \\ 8 \\ 0 \\ 3/4 \\ 1 \end{pmatrix} w = \begin{pmatrix} 0 \\ 6 \\ 8 \\ 0 \\ 3/4 \\ 1 \end{pmatrix} w$$

Imposing (5.5.13) results in

$$\begin{pmatrix} 0 \\ 6 \\ 8 \end{pmatrix} x = \frac{24}{\lambda^2} \begin{pmatrix} 0 \\ 3/4 \\ 1 \end{pmatrix} w \qquad (5.5.18)$$

This is equivalent to the standard matrix eigenvalue equation

$$[\ 0 \quad 0 \quad 1\ ] \begin{pmatrix} 0 \\ 6 \\ 8 \end{pmatrix} w = [\ 8\ ] w = \frac{24}{\lambda^2} w \qquad (5.5.19)$$

Therefore the approximate eigenvalue is $\lambda = \sqrt{3} = 1.73$ and the approximate eigenvector is

$$y = \begin{pmatrix} 0 \\ .75 \\ 1. \end{pmatrix} c \qquad (5.5.20)$$

This is to be compared to the exact solution $\lambda = \frac{\pi}{2} = 1.57$ and

$$y = c \sin \frac{\pi}{2} x \qquad (y(\tfrac{1}{2}) = .707).$$

It may be added that the above procedure is not stationary in the eigenvalues so that their accuracy will be of about the same precision as that of the eigenvectors. If more accurate eigenvalues of (5.5.1) are desired, they can be obtained by inserting the eigenvectors calculated by the method of this section into the Rayleigh quotient [43]

$$k = \frac{\langle y | D_1 y \rangle}{\langle y | D_2 y \rangle} \tag{5.5.21}$$

In the above example, for instance, using (5.5.20) in (5.5.21) results in $\lambda = 1.65$ - a 50% decrease in the error of the eigenvalue. It is interesting to note, however, that the new eigenvalue-eigenvector combination results in a much larger residual function than the original pair.

All of the computer programs for the discretization of differential equations given in the preceding sections of this thesis have been designed with the capability of handling differential equation eigenvalue problems in addition to the standard ones. Therefore, computer programs of all of the operations described for the solution of differential equation eigenvalue problems up to equation (5.5.7) are already available. The remaining steps are performed for both ordinary and partial differential equations by EIGEN shown in Figure 5.5.1. In

Figure 5.5.1a

```
      SUBROUTINE EIGEN(Y,ND,NELMTS,NDPTS,NWBC,C,YR,YI,ROOTR,ROOTI,
     =     NEIGEN)

      THIS SUBROUTINE EVALUATES THE EIGENVALUES AND EIGENVECTORS
      OF THE RECTANGULAR MATRICES STORED IN THE ARRAY 'Y'. THE NUMBER
      OF EIGENVECTORS PRINTED IS 'NEIGEN'.

      DIMENSION Y(1),C(1),YR(1),YI(1),ROOTR(1),ROOTI(1)
      DIMENSION IROW(100),JCOL(200),LOC(100),B(100),OP(1)
      COMMON IROW,JCOL,LOC,B,OP
      DOUBLE PRECISION DOUBLE,DOUB2
      NDELTS = NELMTS * ND
      IJ = NWBC*NDELTS

      THE LEFT HAND SIDE OF THE MATRIX NEEDS TO BE STORED FOR LATER
      USAGE

      DO 19 I = 1,IJ
   19 YR(I) = Y(I)
      NEL = 2*NWBC

      THE MATRIX SYSTEM IS FIRST REDUCED TO STANDARD FORM

      CALL NULL(Y,NDELTS,NWBC,NEL,L,IROW,JCOL,C,NWBC,NWBC,0)
      IEQ = NWBC
      IF(L .NE. NWBC) GO TO 991
      LO = NWBC

      THE EIGENVALUES AND EIGENVECTORS OF THE SINGLE STANDARD SQUARE
      MATRIX ARE EVALUATED

      CALL RILMAT(C,Y,ROOTI,YI,NWBC,NWBC,0,LO)
      DO 20 I = 1,LO
   20 LOC(I) = I
      IF(LO .EQ. 1) GO TO 27
      K = LO - 1

      THE EIGENVALUES ARE ORDERED ACCORDING TO THEIR MAGNITUDES -
      SMALLEST FIRST

      DO 21 I = 1,K
      VALUE = ABS(ROOTI(LOC(I)))
      IJ = I + 1
      DO 21 J = IJ,LO
      IF(ABS(ROOTI(LOC(J))) .GE. VALUE) GO TO 21
      VALUE = ABS(ROOTI(LOC(J)))
      L = LOC(I)
      LOC(I) = LOC(J)
      LOC(J) = L
   21 CONTINUE
   27 DO 22 I = 1,LO
   22 ROOTR(I) = ROOTI(LOC(I))
      DO 23 I = 1,LO
   23 ROOTI(I) = YI(LOC(I))

C
C     ALL OF THE EIGENVALUES ARE PRINTED
C
      WRITE(6,101)
      CALL RITE(ROOTR,LO,1,0,1)
      CALL RITE(ROOTI,LO,1,0,1)
      IF(NEIGEN .EQ. 0) NEIGEN = 5
      IF(LO .GT. NEIGEN) LO = NEIGEN
      NEIGEN = LO
      IJ = 0
C
C     THE EIGENVECTORS ARE ORDERED IN THE SAME WAY AS THE EIGENVALUES
C     - THEY ARE ALSO TRANSFORMED BACK INTO THE ORIGINAL BASIS
C
      DO 29 J = 1,LO
      B(J) = 0.
      KJ = NWBC*(LOC(J) - 1)
      DO 25 I = 1,NDELTS
      IJ = IJ + 1
      DOUBLE = 0.DO
      DOUB2 = 0.DO
      IK = I - NDELTS
      DO 24 K = 1,NWBC
      IK = IK + NDELTS
      IJ1 = K + KJ
      DOUBLE = DOUBLE + DBLE(YR(IK)*C(IJ1))
   24 DOUB2 = DOUB2 + DBLE(YR(IK) * Y(IJ1))
      OP(IJ) = SNGL(DOUBLE)
      B(J) = B(J) + ABS(OP(IJ))
   25 YI(IJ) = SNGL(DOUB2)
      B(J) = NDELTS/B(J)
   29 CONTINUE
      IJ = 0
C
C     THE EIGENVECTORS ARE ALSO SCALED
C
      DO 28 J = 1,LO
      DO 28 I = 1,NDELTS
      IJ = IJ + 1
      Y(IJ) = OP(IJ) * B(J)
   28 YI(IJ) = YI(IJ) * B(J)
      IJ = - NDELTS
      DO 26 I = 1,LO
      IJ = IJ + NDELTS
C
C     THE TRANSFORMED EIGENVECTORS ARE PRINTED
C
      WRITE(6,102) I
      CALL RITE(Y,NDELTS,1,IJ,1)
   26 CALL RITE(YI,NDELTS,1,IJ,1)
```

```
      NWBC = LD                                                          DECL6065
      RETURN                                                             DECL6070
  991 WRITE(6,202) L,NWBC                                                DECL6075
      ND = 0                                                             DECL6080
      RETURN                                                             DECL6085
  101 FORMAT(    1HO,56X,'THE EIGENVALUES')                             DECL6090
  102 FORMAT(1H-,56X,'EIGENVECTOR NUMBER',I2)                            DECL6095
  202 FORMAT('-THE CALCULATED RANK',I2,' DOES NOT EQUAL THE TRUE RANK',  DECL6100
     *     I2)                                                           DECL6105
      END                                                                DECL6110
```

Figure 5.5.1b

order to solve the standard matrix eigenvalue problem, these subroutines call a single precision version of the program RILMAT, published in the Union Carbide programming handbook [59]. Since the eigenvalues and eigenvectors of a general, real matrix may be complex, all of these programs have two arrays for each of these quantities, one for the real and one for the imaginary part.

5.6 NONLINEAR EQUATIONS

The techniques developed thus far provide a powerful method for solving linear differential equations. In this section, the application of this method to problems involving nonlinear differential operators will be considered. The procedure used is the well-known Newton's method for solving nonlinear operator equations and the discussion will draw heavily from reference [60]. In this procedure, the solution of a nonlinear equation is obtained by solving a sequence of linear equations, the solutions of which converge to the solution of the original equation.

The first concept which needs to be introduced when dealing with nonlinear differential equations is the Fréchet derivative of an operator. This may be given as follows [60]:

Definition 5.6.1. Let $A(y)$ be a nonlinear operator. The Fréchet derivative of $A(y)$ at the point y_o will be a linear operator $A'(y_o)$ such that

$$\lim_{||\Delta y|| \to 0} \frac{|| A'(y_o)\,\Delta y - A(y_o + \Delta y) + A(y_o) ||}{|| \Delta y ||} = 0 \qquad (5.6.1)$$

provided that such an operator exists.

As a consequence of this definition, the quantity

$$r(y,z) = A(y) - A(z) - A'(z)\,(y - z) \tag{5.6.2}$$

will have the property that

$$\lim_{||y-z||\to 0} \frac{||\, r(y,z)\,||}{||\, y - z\,||} = 0 \tag{5.6.3}$$

Thus, if the two functions y and z are sufficiently close, the norm of the expression (5.6.2) approaches zero at least as quickly as the square of the norm of the difference of the two functions y and z.

Consider now the problem of solving the nonlinear differential equation

$$A(y) = 0 \tag{5.6.4}$$

and suppose that some function $z \epsilon \Sigma$ is given. Then, from (5.6.2),

$$A(y) = A(z) + A'(y - z) + r(y,z) = 0 \tag{5.6.5}$$

In Newton's method, it is assumed that a function z can be found sufficiently close to the solution y of (5.6.4) so that, by virtue of (5.6.3), the quantity $r(y,z)$ in (5.6.5)

can be neglected [60]. If this is done, then (5.6.5) yields

$$A'(z)\, y \simeq A'(z)\, z - A(z) \qquad (5.6.6)$$

This equation is linear in the unknown y and can be solved by the methods in the previous section. However, since (5.6.6) is an approximation, the solution obtained in this way will not satisfy (5.6.4) exactly. Presumably, though, it will be closer to the correct solution y than was the function z in (5.6.6) and can be used as z in (5.6.6) to obtain an equation having an even better approximate solution. By repeating the above steps enough times, the solution of the linear equation (5.6.6) can be made arbitrarily close to the solution of the nonlinear equation (5.6.4). The conditions on the function z and on the operator $A(y)$ for this procedure to converge are discussed in reference [60].

The effectiveness of Newton's method in treating nonlinear differential equations is limited by two principal defects. The first of these occurs whenever the Fréchet derivative of the operator $A(y)$ does not exist in a neighborhood of the solution y. In these cases, Newton's method is invalid and cannot be applied to determine the solution. The other defect in Newton's method is the difficulty of determining an initial function z within the radius of convergence of the solution for some problems. Unless an a priori estimate of such a function is known, the problem of finding a useful initial function may require considerable effort and ingenuity.

In evaluating the Fréchet derivative $A'(y)$ in (5.6.6) for arbitrary nonlinear differential equations, it is profitable to use the following two properties [60]

$$(A(y) + B(y))'\Big|_{y=z} = A'(z) + B'(z) \tag{5.6.7a}$$

$$(A(B(y)))'\Big|_{y=z} = A'(B(z))\ B'(z) \tag{5.6.7b}$$

These properties permit one to form the Fréchet derivative of a differential operator by using the ordinary rules of differentiation, treating the unknown y as the independent variable

$$A'(z) = \frac{dA(y)}{dy}\Big|_{y=z} \tag{5.6.8}$$

In this way, the Fréchet derivative of an arbitrary differential operator can be easily determined.

As a simple illustration of the procedure, consider the initial value problem

$$A(y) = \frac{d^3y}{dx^3} + y\,\frac{d^2y}{dx^2} = 0$$

$$y(0) = \frac{dy}{dx}(0) = 0 \tag{5.6.9}$$

$$\frac{d^2y}{dx^2} = 1$$

which governs the flow of a laminar boundary layer on a flat plate parallel to the stream [49]. The Fréchet derative of this operator is

$$A'(z) = \frac{d^3}{dx^3} + \frac{d^2z}{dx^2} + z\,\frac{d^2}{dx^2} \tag{5.6.10}$$

Consequently, equation (5.6.6) becomes

$$\frac{d^3y}{dx^3} + \frac{d^2z}{dx^2}\, y + z\, \frac{d^2y}{dx^2} = z\, \frac{d^2z}{dx^2} \qquad (5.6.11)$$

Given an initial value for z, say $z = 0$, this equation may be solved by using the subroutines DIFFEQ, CONTIN and BOUND to obtain an approximate solution of (5.6.9). This approximate solution may then be used for z in (5.6.11) and the process repeated. It will be shown in the next chapter that after several interations, a highly accurate solution of (5.6.9) is obtained.

There are two minor program modifications of the linear subroutine package which increase their efficiency with nonlinear problems. One of these is to store the coordinate independent factors obtained from OPGEN1 in the first iteration. Since these factors contain no functions, they are unchanged by the iteration procedure and may be used repeatedly. Consequently, the step, CALL OPGEN1, may be skipped after the first iteration. The second modification is to place the z function values in a labeled common block accessible to subprogram FCN and the auxiliary functions FN1 - FN5. Then, whatever the value of z in the current iterate, the values of the functions dependent on z can be easily generated.

CHAPTER VI

AUTOMATIC SOLUTION OF DIFFERENTIAL EQUATIONS

The computer programs given in preceding chapters provide a computationally efficient means of solving any ordinary or two-dimensional partial differential equation. However, in their present form, these programs are impractical for everyday use by a wide spectrum of engineers because of their special input and output characteristics. This chapter presents the results of a major effort to write a differential equation solving program which is compatible with ordinary engineering training and experience. In order to do so, it has been necessary to develop a special-purpose computer language which is capable of deciphering mathematical statements and generating computer understandable instructions according to the code given in Chapter 4. By incorporating this special-purpose language and the programs given previously, the solution of any linear ordinary or two-dimensional partial differential equation requires no analysis beyond the ability to write the differential equation on a standard keypunch or teletype terminal. In addition, with the simple modifications described in section 5.6, many nonlinear differential equations can be solved simply by evaluating the Fréchet derivative of the differential operator analytically and forming equation (5.6.6) beforehand.

The use of the automatic differential equation solving programs will also be illustrated in this chapter by solving numerous examples from linear and nonlinear ordinary differential equations and from linear two-dimensional partial differential equations. Whenever possible, the results from the programs will be checked with analytical solutions or with previously computed answers. These comparisons will indicate the extremely accurate solutions obtained by using high-order polynomial matrix equivalents for differential operators.

The development of convenient computer languages for differential equations has attracted considerable interest in recent years. The main impetus behind this movement has been a desire to free the numerical solution of differential equations from cumbersome Fortran programming. However, since all of the languages developed to date have been tied to inefficient finite difference formulae, none of these languages has received widespread acceptance.

For initial value problems, a formal computer language and associated program compiler has been designed by Barton, Willers and Zaher [61]. With this system, commands such as INTEGRATE, WITH INITIAL CONDITIONS, END and the mathematical symbols + - * / ↑ ' define the operations to be performed by the computer. As described in [61], the language is capable of handling arbitrary systems of ordinary differential equations for which initial conditions are known.

A language has also been written for partial differential equations, this one by Cardenas and Karplus [62]. Called PDEL, the language is a superset of PL/1 and may be mixed

with it. There are ten functionally different types of statements in PDEL and the syntactic definition of each has been designed to simplify coding procedures for differential equations. At the time reference [62] was written, the PDEL translator was able to handle five different types of differential equations.

A second language for partial differential equations has been written by Roberts and Boris using a symbolic style of Algol [63]. In this language, real procedures such as CURL, CROSS and DELSQ are defined which produce the numerical operations corresponding to their analytic or algebraic counterparts. By providing a library of such procedures, Symbolic Algol allows three dimensional partial differential equations to be written in a form very closely resembling the formalism of mathematical physics.

6.1 SIMPLIFIED INPUT AND OUTPUT PROCEDURES

In this section, computer programs are described which provide a convenient engineer-to-computer interfacing for the programs given in Chapters 4 and 5. With these additions, the operation of the differential equation solving programs in this thesis is made extremely easy: the user simply writes the differential equation and associated boundary conditions on a standard keypunch or teletype terminal in easily recognizable mathematical form and the computer automatically returns its general and particular solutions in both analytical and graphical forms.

The input portion of the new programs is based upon a new

symbolic computer language for the representation of differential equations. Called DECL for Differential Equation Computer Language, this language has been designed to maintain the free-form writing of differential equations as far as is possible on a computer. Accordingly, the elementary units of the language are taken to be the standard ASCII 60 character set, with numbers obeying the same syntax rules as numbers in Fortran. The characters X, Y and Z are reserved to represent the dependent and independent variables of differential equations in the following way

one dimension

<independent variable>::=X

<dependent variable>::=Y

two dimensions

<independent variables>::=X|Y

<dependent variable>::=Z

The metalinguistic notation used here is that of reference [64]. When any of the remaining letters is used by itself, it denotes a numerical constant

<constant>::= < any individual letter except X,Y or Z >

In DECL, multi-literal names are not allowed.

Functions are represented in DECL by the following character strings

<*sin x*>::= SIN(X)
<*cos x*>::= COS(X)
<*log x*>::= LOG(X)
<*exp x*>::= EXP(X)
<*auxiliary functions*>::= FN1(X) ∿ FN5(X)
<x^{number}>::= POW<*number*>(X)

The use of the auxiliary functions FN1 to FN5 has already been explained in Chapter 4. The reason for adopting the above notation for exponentiation is that DECL, in contrast to such number manipulation languages as Fortran and Algol, is a symbol manipulation language. Thus, while the double star in the Fortran expression X**2 or the up arrow in the Algol expression X↑2 means that the numerical value of X is to be squared, the DECL expression POW2(X) refers to the functional operation of multiplying the function X with itself. Accordingly, the form of exponentiation in DECL is chosen to be the same as for other functional operations with the function preceding its argument.

In DECL, the arguments of functions may be composed of any combination of the independent variable(s), constants, numbers, other functions, and the symbols +-*(). In evaluating a function with a compound argument, DECL follows the same structural conventions as Fortran. Note that the symbol / may not appear in a functional expression in DECL and, as a result, division can only be obtained by applying the function POW-1.

Differential operators are specified in DECL by the following expressions

$< \frac{d}{dx}$ or $\frac{\partial}{\partial x} >$::= D/DX

$< \frac{\partial}{\partial y} >$::= D/DY

$< \frac{d^n}{dx^n}$ or $\frac{\partial^n}{\partial x^n}$;n = 2,...,9 >::= D<n>/DX<n>

$< \frac{\partial^n}{\partial y^n}$;n = 2,...,9 >::= D<n>/DY<n>

Any valid DECL expression may follow a differential operator symbol.

A DECL statement is a string of valid DECL expressions in a predefined and proper order. DECL statements may be written anywhere on an 80 column computer card with the first character appearing in column one. Blanks may be inserted between any two DECL expressions in a DECL statement as long as they do not alter the appearance of the expressions themselves.

There are six functionally different statements in DECL:

(1) Control Statement

<control statement>::=*<number><title>

The number in a control statement may be any integer from 1 to 9 and specifies the order of the approximating polynomial to be used in the solution. The characters given in the title will be used to provide a heading for the output. An example of a control statement is

```
* 9      TITLE
```

(2) Differential Equation Statement

<differential equation statement>::=

DE:<any valid DECL differential equation>

A DECL differential equation is any DECL expression containing at least one differential operator and one and only one equal sign. Differential equation statements may be continued on consecutive computer cards provided that the letters DE:begin each one of them. A typical differential equation statement is

```
DE: D2/DX2(Y)  = K*  SIN(X) * Y
```

(3) Boundary Condition Statement

<boundary condition statement>::=

BC:<any valid DECL boundary condition>

A DECL boundary condition is any DECL expression containing one equal sign in which each of the dependent and independent variables is followed by a number or pair of numbers in parenthesis. These numbers provide the coordinate values of the boundary condition locations. An example of a boundary condition statement is

BC: D/DX(Y(0.0)) + EXP(1.5) * Y(1.5) = P

(4) Constant Specification Statement

<constant specification statement>::=

CST:<constant>=<number>

Constant specification statements are used to provide numerical values for known constant parameters in differential equation or boundary condition statements. A typical constant specification statement is

CST:P=3.141592

Any constant in a differential equation statement which does not have its value defined by a constant specification statement is assumed to be a symbol for an eigenvalue. Although they may appear repeatedly in a differential equation statement, only one eigenvalue symbol may be used per differential equation. Eigenvalue symbols are not allowed in boundary condition statements.

(5) Geometry Statement

one dimension

<geometry statement>::=

X=<number>,<number>(<integer>ELEMENTS)

two dimensions

<geometry statement>::=

REC: X=<number>,<number>(<integer>SUBDIVISIONS),

Y=<number>,<number>(<integer>SUBDIVISIONS)

In one dimension, the geometry statement specifies the end-points of a line segment to be used in the solution and provides the number of equal elements to be taken in that segment. In two dimensions, it specifies the edges of the rectangular region to be used and the number of equal elemental sub-divisions in each direction. If the number of elements or the number of subdivisions to be used is not specified, one element or one subdivision is assumed. Unequal one-dimensional elements or irregular two-dimensional regions may be defined by using more than one geometry statement. Examples of geometry statements are

X=0.0, 1.0 (2 ELEMENTS)

REC:X=0.0, 1.0 (2 SUBDIVISIONS), Y= 0.0,2.0

(6) Output Statements

regular problems

<output statement #1>::=WRITE<parameters>SOLUTIONS

<output statement #2>::=PLOT<parameters>SOLUTIONS

where

<parameter #1>::=GENERAL

<parameter #2>::=AND

<parameter #3>::=PARTICULAR

eigenvalue problems

<output statement #1>::=WRITE<number>EIGENVECTORS

<output statement #2>::=PLOT<number>EIGENVECTORS

Output statements control the manner in which the results are displayed by the computer. Typical output statements are

```
WRITE  GENERAL  AND  PARTICULAR  SOLUTIONS
PLOT   PARTICULAR  SOLUTION
WRITE  10  EIGENVECTORS
PLOT    2  EIGENVECTORS
```

A DECL program consists of DECL statements in the order given above. Each program must contain one control statement, one differential equation statement (which may be continued on any number of cards), any number of boundary condition statements, the number of constant specification statements required by the differential equation and boundary condition statements, one or more geometry statements and two output statements. For programming convenience, the two output statements may be replaced by a single blank card. In this case, the default option of the program is to write and plot both the general and particular solutions of ordinary differential equations, write and plot only the particular solution of partial differential equations, and to write and plot the first five eigenvectors of eigenvalue problems. Examples of valid DECL programs will be given in the following sections.

Computer programs which read boundary value problems written in DECL and translate them into instructions for the differential equation solving programs given earlier are presented in Figures 6.1.1 and 6.1.2. The first of these is the Fortran subroutine INSTR for use with ordinary differential equations and the second is INST2D for use with two-dimensional partial differential equations. These compilers are written in Fortran instead of a machine language so that they are machine

Figure 6.1.1a

```
      SUBROUTINE INSTR(INSTR,CONST,NUMDE,XBC,XD,WT,NELMTS,
     *     NRANGE,NDOM,LAMDA,RITEP,RITEG,PLOTP,PLOTG,NEIGWR,NEIGPL)

C
C     THIS SUBROUTINE READS DIFFERENTIAL EQUATIONS, BOUNDARY CONDI-
C     ITIONS AND GEOMETRICAL DATA AND CONVERTS THEM INTO CODED
C
C     INSTRUCTIONS. THE DIFFERENTIAL EQUATION AND BOUNDARY CONDITIONS
C
C     ARE RETURNED IN THE ARRAY 'INSTR' AND THE POINT LOCATIONS ARE
C     STORED IN 'XD'.
C
      INTEGER*2 INSTR
      DIMENSION INSTR(1),CONST(1),XBC(1),XD(1),WT(1)
      DIMENSION ARRAY(80),ISKIP(100),IHERE(100),IPAR(100),TYPE(8)
      LOGICAL RSIDE,YCND,FCNCND,PARCND,ARGUE,LAMDA,RITEP,RITEG,PLOTP,
     *     PLOTG,CHARTF
      COMMON ARRAY,ISKIP,IHERE,IPAR
      REAL LBRACK,MINUS
      EQUIVALENCE (TYPE(1),STAR),   (TYPE(2),DEE),   (TYPE(3),BEE),
     *     (TYPE(4),GEE),   (TYPE(5),EX),   (TYPE(6),DBLU),   (TYPE(7),
     *     PEA),   (TYPE(8),BLANK)
      DATA BLANK,LBRACK,RBRACK,PLUS,MINUS,STAR,SLASH,EQUAL
     *       /1H ,1H(,1H),1H+,1H-,1H*,1H/,1H=/
      DATA BEE,CEE,DEE,EEE,EF ,GEE,EYE,EL ,EN ,PEA,OH ,ARE,ES,DBLU,EX ,
     *     YnY
     *    /1HB,1HC,1HD,1HE,1HF,1HG,1HI,1HL,1HN,1HP,1HO,1HR,1HS,1HW,1HX,
     *     1HY /

C
C     READ THE DATA CARDS
C
    1 READ(9,110,END=999) ARRAY
C
C     GO TO THE LOCATION INDICATED BY THE TYPE OF CARD
C
      DO 2 I = 1,8
      IF(ARRAY(1) .EQ. TYPE(I)) GO TO (3,15,13,4,7,82,84,86),I
    2 CONTINUE
      GO TO 990
    3 PRCARD = STAR
C
C     DETERMINE THE ORDER OF ELEMENT REQUESTED AND PRINT THE TITLE
C
      NDOM = IFIX(FNUM(ARRAY,2,I1) + .1)
      IF(NDOM .GE. 10) GO TO 990
      IF(I1 .GT. 80) I1 = 80
      WRITE(6,111)
      WRITE(6,102) (ARRAY(I),I = I1,80)
C
C     INITIALIZE THE PARAMETERS
C
      INDEX = 0
      NCSTS = 0
      NSCPT = 0
      IPARA = 0
      NPROD = 0
      LASIGN = 0
      NSKIP = 0
      ISIGN = 1
      LAMDA = .FALSE.
      EVALUE = BLANK
      RSIDE = .FALSE.
      FCNCND = .FALSE.
      PARCND = .FALSE.
      ARGUE = .TRUE.
      WRITE(6,100)
      GO TO 1
C
C     A BOUNDARY CONDITION CARD HAS BEEN READ
C
   13 IF(PRCARD .EQ. STAR .OR. PRCARD .EQ. EX) GO TO 990
      WRITE(6,106)
      INDEX = INDEX + 1
      INSTR(INDEX) = 4000
      NUMDE = INDEX
   14 INDEX =INDEX + 1
      INSTR(INDEX) = 3000
      IF(.NOT. RSIDE) GO TO 990
      RSIDE = .FALSE.
C
C     A DIFFERENTIAL EQUATION CARD HAS BEEN READ
C
   15 YCND = .FALSE.
      IF(IPARA .NE. 0 .OR. FCNCND .OR. PARCND) GO TO 990
      PRCARD = ARRAY(1)
      INDEX = INDEX + 1
      INSTR(INDEX) = 1000
      IATS = -1
      WRITE(6,102) (ARRAY(I),I =4,80)
C
C     'I' INDICATES THE LOCATION ON THE DATA CARD
C
      I = 3
   16 I = I + 1
C
C     IF SOME LOCATIONS WERE SKIPPED ON THE DATA CARD, RETURN TO THEM
C
      IF(NSKIP .EQ. 0) GO TO 58
      DO 57 M = NSTART,NPROD
      IF(I .NE. ISKIP(M)) GO TO 57
      I = IHERE(M)
      IF(ARRAY(I - 1) .EQ. MINUS) ISIGN = -1 * ISIGN
   57 CONTINUE
C
C     BLANKS AND ASTERISCS ARE IGNORED
C
   58 IF(ARRAY(I) .EQ. BLANK) GO TO 18
      IF(ARRAY(I) .EQ. STAR) GO TO 54
      INDEX = INDEX + 1
```

Figure 6.1.1b

```
C
C     CHECK FOR: '('
C
      IF(ARRAY(I) .NE. LBRACK) GO TO 56
      IPARA = IPARA + 1
      INSTR(INDEX) = 300
      IF(FCNCND .OR. PARCND) GO TO 54
      M = I
      N = IPARA
   59 M = M + 1
      IF(ARRAY(M) .EQ. WHY .OR. ARRAY(M) .EQ. SLASH) GO TO 60
      IF(ARRAY(M) .EQ. LBRACK) N = N + 1
      IF(ARRAY(M) .EQ. RBRACK) N = N - 1
      IF(N .GE. IPARA) GO TO 59
      IF(FCNCND) GO TO 54
      FCNCND = .TRUE.
      ARGUE = .FALSE.
      IFPARA = IPARA
      GO TO 54
   60 PARCND = .TRUE.
      IATP = I
      GO TO 54
C
C                 ')'
C
   56 IF(ARRAY(I) .NE. RBRACK) GO TO 17
      IPARA = IPARA - 1
      INSTR(INDEX) = 400
      IF(IPARA .EQ. IFPARA) FCNCND = .FALSE.
      GO TO 55
C
C                 'Y'
C
   17 IF(ARRAY(I) .NE. WHY) GO TO 22
      IF(FCNCND) GO TO 990
      YCND = .TRUE.
      ARGUE = .TRUE.
      INSTR(INDEX) = 200
      IF(ARRAY(I) .EQ. DEE) GO TO 55
      NBCPT = NBCPT + 1
      XBC(NBCPT) = FNUM(ARRAY,I2,I3)
      I = I3
      INDEX = INDEX + 1
      INSTR(INDEX) = - (300 + NBCPT)
      GO TO 55
C
C                 'X'
C
   22 IF(ARRAY(I) .NE. EX) GO TO 23
      INSTR(INDEX) = 100
      ARGUE = .TRUE.
      GO TO 54
C
C                 '='
DECL 880 C
DECL 885    23 IF(ARRAY(I) .NE. EQUAL) GO TO 24
DECL 890       IF(PARCND) GO TO 50
DECL 895       IF(RSIDE) GO TO 990
DECL 900       RSIDE = .TRUE.
DECL 905       ARRAY(I) = PLUS
DECL 910 C
DECL 915 C                 '+' OR '-'
DECL 920 C
DECL 925    24 IF(ARRAY(I) .NE. PLUS .AND. ARRAY(I) .NE. MINUS) GO TO 32
DECL 930    50 IF(.NOT. ARGUE) GO TO 990
DECL 935       IF(.NOT. FCNCND) GO TO 25
DECL 940       INSTR(INDEX) = 500
DECL 945       IF(ARRAY(I) .EQ. MINUS) INSTR(INDEX) = 600
DECL 950       GO TO 55
DECL 955    25 IF(.NOT. YCND .OR. IPARA .NE. 0) GO TO 26
DECL 960       YCND = .FALSE.
DECL 965 C
DECL 970 C     IF THE + OR - IS WITHIN A PARENTHESIS, THE REMAINING TERMS NEED
DECL 975 C     TO BE SKIPPED
DECL 980 C
DECL 985    26 IF(.NOT. PARCND) GO TO 30
DECL 990       INDEX = INDEX - 1
DECL 995       IF(IPARA .EQ. 0) GO TO 63
DECL1000       IF(NSKIP .NE. 0) GO TO 28
DECL1005    27 NPROD = NPROD + 1
DECL1010       ISKIP(NPROD) = IATP + 1
DECL1015       IHERE(NPROD) = I + 1
DECL1020       IPAR(NPROD) = IPARA
DECL1025       LASIGN = NPROD
DECL1030       GO TO 62
DECL1035    28 DO 29 N = 1,NPROD
DECL1040       IF(I .LT. ISKIP(N)) GO TO 61
DECL1045    29 CONTINUE
DECL1050       N = N + 1
DECL1055    61 LASIGN = N - 1
DECL1060       ILASIN = I
DECL1065    62 I = I + 1
DECL1070 C
DECL1075 C     LOCATE THE END OF THE PARENTHESIS CONDITION
DECL1080 C
DECL1085       IF(I .GT. 77) GO TO 990
DECL1090       IF(ARRAY(I) .EQ. LBRACK) IPARA = IPARA + 1
DECL1095       IF(ARRAY(I) .EQ. RBRACK) IPARA = IPARA - 1
DECL1100       IF(IPARA .GE. IPAR(LASIGN)) GO TO 62
DECL1105       GO TO 16
DECL1110    63 IF(LASIGN .EQ. 0) GO TO 65
DECL1115 C
DECL1120 C     DETERMINE THE SIGN OF THE CURRENT SIMPLE OPERATOR
DECL1125 C
DECL1130       IF(NSKIP .EQ. 0) NSKIP = 1
DECL1135       NSTART = NPROD - NSKIP + 1
DECL1140       I = IATS
DECL1145       IHERE(LASIGN) = ILASIN + 1
DECL1150
DECL1155
DECL1160
DECL1165
DECL1170
DECL1175
DECL1180
DECL1185
DECL1190
DECL1195
DECL1200
DECL1205
DECL1210
DECL1215
DECL1220
DECL1225
DECL1230
DECL1235
DECL1240
DECL1245
DECL1250
DECL1255
DECL1260
DECL1265
DECL1270
DECL1275
DECL1280
DECL1285
DECL1290
DECL1295
DECL1300
DECL1305
DECL1310
DECL1315
DECL1320
DECL1325
DECL1330
DECL1335
DECL1340
DECL1345
DECL1350
DECL1355
DECL1360
DECL1365
DECL1370
DECL1375
DECL1380
DECL1385
DECL1390
DECL1395
DECL1400
DECL1405
DECL1410
DECL1415
```

Figure 6.1.1c

```
      IF(LASIGN .EQ. NPROD) GO TO 55
      NSKIP = NSKIP + 1
      N = LASIGN + 1
      DO 64 M = N,NPROD
   64 THERE(M) = ISKIP(M)
      GO TO 55
   65 NSKIP = 0
      NPROD = 0
      ISIGN = 1
      PARCND = .FALSE.
      INDEX = INDEX + 1
      IF(ARRAY(I) .EQ. EQUAL) GO TO 23
   30 IF(INSTR(INDEX - 1) .EQ. 1000) INDEX = INDEX - 1
      IATS = I
      INSTR(INDEX) = 1000
C
C     SIGNS ON THE RIGHT SIDE OF AN EQUATION ARE REVERSED
C
      IF(.NOT. RSIDE .AND. ARRAY(I) .EQ. PLUS) GO TO 55
      IF(RSIDE .AND. ARRAY(I) .EQ. MINUS) GO TO 55
      INSTR(INDEX) = 2000
      ISIGN = - 1 * ISIGN
      GO TO 55
   32 I1 = I  + 1
      I2 = I1 + 1
      I3 = I2 + 1
      I4 = I3 + 1
      IF(I4 .GE. 80) GO TO 49
C
C                 'D/DX'
C
      IF(ARRAY(I) .NE. DEE .OR. ARRAY(I1) .NE. SLASH
     *     .OR. ARRAY(I2) .NE. DEE .OR. ARRAY(I3) .NE. EX) GO TO 33
      IF(FCNCND) GO TO 990
      INSTR(INDEX) = - 1
      I = I3
      GO TO 54
C
C                 'DN/DXN'
C
   33 IF(ARRAY(I) .NE. DEE .OR. ARRAY(I2) .NE. SLASH
     *     .OR. ARRAY(I3) .NE. DEE .OR. ARRAY(I4) .NE. EX) GO TO 36
      IF(FCNCND) GO TO 990
      INSTR(INDEX) = - IFIX(FNUM(ARRAY,I1,I2) + .1)
      IF(I2 .NE. I1 + 1) GO TO 990
      I = I4 + 1
      GO TO 54
C
C                 'PO.'
C
   36 IF(ARRAY(I) .NE. PEA .OR. ARRAY(I1) .NE. OH
     *     .OR. ARRAY(I2) .NE. DBLU) GO TO 42
      NCSTS = NCSTS + 1
      INSTR(INDEX) = - (200 + NCSTS)
      CONST(NCSTS) = FNUM(ARRAY,I3,I4)
      IF(I4 .EQ. I3) GO TO 990
      I = I4 - 1
      GO TO 53
C
C                 'SIN'
C
   42 IF(ARRAY(I) .NE. ES .OR. ARRAY(I1) .NE. EYE
     *     .OR. ARRAY(I2) .NE. EN) GO TO 43
      INSTR(INDEX) = 11
      GO TO 52
C
C                 'COS'
C
   43 IF(ARRAY(I) .NE. CEE .OR. ARRAY(I1) .NE. OH
     *     .OR. ARRAY(I2) .NE. ES) GO TO 44
      INSTR(INDEX) = 12
      GO TO 52
C
C                 'EXP'
C
   44 IF(ARRAY(I) .NE. EEE .OR. ARRAY(I1) .NE. EX
     *     .OR. ARRAY(I2) .NE. PEA) GO TO 45
      INSTR(INDEX) = 13
      GO TO 52
C
C                 'LOG'
C
   45 IF(ARRAY(I) .NE. EL .OR. ARRAY(I1) .NE. OH
     *     .OR. ARRAY(I2) .NE. GEE) GO TO 46
      INSTR(INDEX) = 14
      GO TO 52
C
C                 'EN'
C
   46 IF(ARRAY(I) .NE. EE .OR. ARRAY(I1) .NE. EN) GO TO 49
      I1 = IFIX(FNUM(ARRAY,I2,I3) + .1)
      IF(I3 .NE. I2 + 1 .OR. I1 .GT. 5) GO TO 990
      INSTR(INDEX) = 14 + I1
      GO TO 52
C
C                 'A NUMBER'
C
   49 NCSTS = NCSTS + 1
      ARGUE = .TRUE.
      INSTR(INDEX) = - (100 + NCSTS)
      CONST(NCSTS) = FNUM(ARRAY,I,I1)
      IF(I1 .EQ. I) GO TO 47
      I = I1 - 1
      GO TO 54
C
C     WHATEVER REMAINS MUST BE A CONSTANT
C
   47 IF(FCNCND) GO TO 990
```

Figure 6.1.1d

```
      INSTR(INDEX) = 800
      XD(INDEX) = ARRAY(I)
      ARGUE = .TRUE.
      GO TO 54
   52 I = I2
   53 ARGUE = .FALSE.
      IF(FCNCND) GO TO 55
      FCNCND = .TRUE.
      IFPARA = IPARA
   54 IF(YCND) GO TO 990
   55 CONTINUE
   18 IF(I .LT. 80) GO TO 16
      GO TO 1

      EVALUATE THE CONSTANT SPECIFICATION STATEMENT

    4 IF(PRCARD .EQ. CEE) GO TO 5
      IF(PRCARD .EQ. STAR .OR. PRCARD .EQ. EX) GO TO 991
      PRCARD = CEE
      WRITE(6,107)
    5 DO 75 I = 5,80
      IF(ARRAY(I) .NE. BLANK) GO TO 76
   75 CONTINUE
      GO TO 991
   76 I1 = I + 1
      VALUE = FNUM(ARRAY,I1,I2)
      IF(I1 .EQ. I2) GO TO 991
      WRITE(6,108) (ARRAY(I3),I3=1,I2)
      DO 77 I3 = 1,INDEX
      IF(XD(I3) .EQ. ARRAY(I3)) GO TO 78
   77 CONTINUE
      GO TO 1
   78 NCSTS = NCSTS + 1
      INSTR(I3) = - (100 + NCSTS)
      CONST(NCSTS) = VALUE
      GO TO 77

      EVALUATE THE GEOMETRY STATEMENT

    7 IF(PRCARD .EQ. EX) GO TO 71
      IF(PRCARD .EQ. STAR) GO TO 991

      DETERMINE IF THE PROBLEM IS AN EIGENVALUE PROBLEM

      DO 79 I = 1,INDEX
      IF(INSTR(I) .EQ. 800) GO TO 80
   79 CONTINUE
      GO TO 81
   80 IF(LAMDA .AND. XD(I) .NE. EVALU) GO TO 991
      IF(LAMDA) GO TO 79
      LAMDA = .TRUE.
      EVALU = XD(I)
      GO TO 79
   81 IF(INDEX .LE. 2) GO TO 991

DECL1960        NUMDE = NUMDE + 1                                       DECL2230
DECL1965        INDEX = INDEX + 1                                       DECL2235
DECL1970        INSTR(INDEX) = 4000                                     DECL2240
DECL1975        WRITE(6,104) (I,INSTR(I),I = 1,INDEX)                   DECL2245
DECL1980      8 I = 1                                                   DECL2250
DECL1985        NRANGE = NDOM                                           DECL2255
DECL1990      9 N = 0                                                   DECL2260
DECL1995 C                                                            DECL2265
DECL2000 C      DETERMINE THE ORDER OF THE RANGE SPACE                  DECL2270
DECL2005 C                                                            DECL2275
DECL2010     10 I = I + 1                                             DECL2280
DECL2015        IF(INSTR(I) .GE. 10) GO TO 11                         DECL2285
DECL2020        IF(INSTR(I) .LE. -10) GO TO 11                        DECL2290
DECL2025        N = N + INSTR(I)                                      DECL2295
DECL2030     11 IF(INSTR(I) .LT. 1000) GO TO 10                       DECL2300
DECL2035        NRANGE = MINO(NRANGE,NDOM + N)                        DECL2305
DECL2040        IF(INSTR(I) .LT. 4000) GO TO 9                        DECL2310
DECL2045        IF(NRANGE .GE. 0) GO TO 12                            DECL2315
DECL2050        NDOM = NDOM - NRANGE                                  DECL2320
DECL2055        NRANGE = 0                                            DECL2325
DECL2060     12 WRITE(6,105) NDOM,NRANGE                              DECL2330
DECL2065        NELMTS = 0                                            DECL2335
DECL2070        NPT = 0                                               DECL2340
DECL2075     71 DO 72 I = 2,80                                        DECL2345
DECL2080        IF(ARRAY(I) .EQ. EQUAL) GO TO 73                      DECL2350
DECL2085     72 CONTINUE                                              DECL2355
DECL2090        GO TO 991                                             DECL2360
DECL2095     73 I = I + 1                                             DECL2365
DECL2100 C                                                            DECL2370
DECL2105 C      FIND THE LOCATIONS OF THE ENDPOINTS OF THE ELEMENTS   DECL2375
DECL2110 C                                                            DECL2380
DECL2115        X1 = FNUM(ARRAY,I,I1)                                 DECL2385
DECL2120        IF(I1 .EQ. I) GO TO 991                               DECL2390
DECL2125        IF(NELMTS .NE. 0 .AND. X1 .NE. X2) GO TO 991          DECL2395
DECL2130        I = I1 + 1                                            DECL2400
DECL2135        X2 = FNUM(ARRAY,I,I1)                                 DECL2405
DECL2140        IF(I1 .EQ. I) GO TO 991                               DECL2410
DECL2145        I = I1 + 1                                            DECL2415
DECL2150 C                                                            DECL2420
DECL2155 C      READ THE NUMBER OF ELEMENTS IN THE INTERVAL           DECL2425
DECL2160 C                                                            DECL2430
DECL2165        NEL = IFIX(FNUM(ARRAY,I,I1) + .1)                     DECL2435
DECL2170        IF(NEL .LE. 0) NEL = 1                                DECL2440
DECL2175        XLNGTH = (X2 - X1)/NEL                                DECL2445
DECL2180        STEP = XLNGTH / (NDOM - 1)                            DECL2450
DECL2185        XPT = X1                                              DECL2455
DECL2190 C                                                            DECL2460
DECL2195 C      DETERMINE THE INTERPOLATION POINTS                    DECL2465
DECL2200 C                                                            DECL2470
DECL2205        DO 74 I = 1,NEL                                       DECL2475
DECL2210        NELMTS = NELMTS + 1                                   DECL2480
DECL2215        WT(NELMTS) = 1.0 / XLNGTH                             DECL2485
DECL2220        XPT = XPT - STEP                                      DECL2490
DECL2225        DO 74 I1 = 1,ND                                       DECL2495
```

Figure 6.1.1e

```
      XPT = XPT + STEP                                                  DECL2500
      NPT = NPT + 1                                                     DECL2505
   74 XD(NPT) = XPT                                                     DECL2510
      PRCARD = EX                                                       DECL2515
      GO TO 1                                                           DECL2520
                                                                        DECL2525
      DEFINE THE OUTPUT PROCEDURE                                       DECL2530
                                                                        DECL2535
   82 IF(PRCARD .NE. EX) GO TO 991                                      DECL2540
      PRCARD = DBLU                                                     DECL2545
      IF(LAMDA) GO TO 83                                                DECL2550
      RITEP = CHARTF(ARRAY,PEA)                                         DECL2555
      RITEG = CHARTF(ARRAY,GEE)                                         DECL2560
      GO TO 1                                                           DECL2565
   83 NEIGWR = IFIX(FNUM(ARRAY,6,I) + 0.1)                              DECL2570
      IF(I .EQ. 6) GO TO 991                                            DECL2575
      GO TO 1                                                           DECL2580
   84 IF(PRCARD .NE. DBLU) GO TO 991                                    DECL2585
      IF(LAMDA) GO TO 85                                                DECL2590
      PLOTP = CHARTF(ARRAY,PEA)                                         DECL2595
      PLOTG = CHARTF(ARRAY,GEE)                                         DECL2600
      GO TO 86                                                          DECL2605
   85 NEIGPL = MINO(IFIX(FNUM(ARRAY,5,I) + 0.1),NEIGWR)                 DECL2610
      IF(I .EQ. 5) GO TO 991                                            DECL2615
      GO TO 86                                                          DECL2620
   86 IF(PRCARD .NE. EX) GO TO 991                                      DECL2625
      IF(LAMDA) GO TO 87                                                DECL2630
      RITEP = .TRUE.                                                    DECL2635
      RITEG = .TRUE.                                                    DECL2640
      PLOTP = .TRUE.                                                    DECL2645
      PLOTG = .TRUE.                                                    DECL2650
      GO TO 88                                                          DECL2655
   87 NEIGWR = 5                                                        DECL2660
      NEIGPL = 5                                                        DECL2665
   88 WRITE(6,101)                                                      DECL2670
      WRITE(5,101)                                                      DECL2675
      CALL RITE(XD,NPT,1,0,1)                                           DECL2680
      RETURN                                                            DECL2685
                                                                        DECL2690
      ERRORS IN THE EQUATION                                            DECL2695
                                                                        DECL2700
   90 WRITE(6,103) ARRAY(I),I,IPARA,RSIDE,YCND,FCNCND,PARCND,ARGUE      DECL2705
      WRITE(6,104) (I,INSTR(I),I=1,INDEX)                               DECL2710
      GO TO 1                                                           DECL2715
   91 WRITE(6,109)                                                      DECL2720
      WRITE(6,102) ARRAY                                                DECL2725
      GO TO 1                                                           DECL2730
   99 STOP                                                              DECL2735
  100 FORMAT(1H-,51X'THE DIFFERENTIAL EQUATION IS'//)                   DECL2740
  101 FORMAT(1H-,55X,'THE POINT COORDINATES'//)                         DECL2745
  102 FORMAT(1HO,25X,80A1)                                              DECL2750
  103 FORMAT('-ERROR IN DIFFERENTIAL EQUATION:   ',A1,7I3)              DECL2755
  104 FORMAT(1H-,54X,'THE INSTRUCTION CODE IS'//(1H ,25X,5(I10,I6)))    DECL2760
  105 FORMAT(1H1,27X,'THE POLYNOMIAL ORDER OF THE DOMAIN SPACE IS',     DECL2765
     *       12,' AND THE ORDER OF THE RANGE IS',I2)                    DECL2770
  106 FORMAT(1H-,51X,'THE BOUNDARY CONDITIONS ARE'//)                   DECL2775
  107 FORMAT(1HO,52X,'CONST SPECIFICATION LIST'//)                      DECL2780
  108 FORMAT(56X,75A1)                                                  DECL2785
  109 FORMAT('-ERROR:')                                                 DECL2790
  110 FORMAT(80A1)                                                      DECL2795
  111 FORMAT(1H1)                                                       DECL2800
      END                                                               DECL2805
```

```
      FUNCTION CHARTF(A,C)                                              DECL6595
C                                                                       DECL6600
C     CHARTF DETERMINES WHETHER OR NOT CHARACTER 'C' IS CONTAINED IN    DECL6605
C     THE ARRAY 'A'.                                                    DECL6610
C                                                                       DECL6615
      LOGICAL CHARTF                                                    DECL6620
      DIMENSION A(1)                                                    DECL6625
      CHARTF = .TRUE.                                                   DECL6630
      DO 1 I = 1,80                                                     DECL6635
      IF(A(I) .EQ. C) RETURN                                            DECL6640
    1 CONTINUE                                                          DECL6645
      CHARTF = .FALSE.                                                  DECL6650
      RETURN                                                            DECL6655
      END                                                               DECL6660
```

Figure 6.1.2a

```
      SUBROUTINE INST2D(INSTR,CONST,NUMDE,XBC,YBC,XD,YD,WTX,WTY,          DE2D 200
     *     IELCX,IELCY,NCCOND,NELMTS,NRANGX,NRANGY,NDOM,LAMDA,RITEP,     DE2D 205
     *     RITEG,PLOTP,PLOTG,NEIGWR,NEIGPL)                              DE2D 210
C                                                                        DE2D 215
C     THIS SUBROUTINE READS LINEAR TWO-DIMENSIONAL PARTIAL DIFFERENTIAL  DE2D 220
C     EQUATIONS, BOUNDARY CONDITIONS AND GEOMETRICAL DATA AND CONVERTS   DE2D 225
C     THEM INTO CODED INSTRUCTIONS.                                      DE2D 230
C                                                                        DE2D 235
      DIMENSION YBC(1),YD(1),WTY(1),IELCX(1),IELCY(1)                    DE2D 240
      DIMENSION INSTR(1),CONST(1),XBC(1),XD(1),WTX(1)                    DE2D 245
      DIMENSION ARRAY(80),ISKIP(100),IHERE(100),IPAR(100),NELC(39),      DE2D 250
     *     TYPE(10)                                                      DE2D 255
C                                                                        DE2D 260
C                            DECL 410-430                                DE2D 265
C                                                                        DE2D 270
     *     (TYPE(4),CEE),   (TYPE(5),ARE),   (TYPE(6),OBLU),   (TYPE(7),DE2D 275
     *     PEA),   (TYPE(8),EX),   (TYPE(9),WHY),   (TYPE(10),BLANK)     DE2D 280
C                                                                        DE2D 285
C                            DECL 445-455                                DE2D 290
C                                                                        DE2D 295
     *     WHY,ZEE                                                       DE2D 300
C                                                                        DE2D 305
C                            DECL 465-465                                DE2D 310
C                                                                        DE2D 315
     *     1HY,1HZ /                                                     DE2D 320
C                                                                        DE2D 325
C                            DECL 475-505                                DE2D 330
C                                                                        DE2D 335
      DO 2 I = 1,10                                                      DE2D 340
      IF(ARRAY(1) .EQ. TYPE(I)) GO TO (3,15,4,7,82,84,150,150,82),I      DE2D 345
C                                                                        DE2D 350
C                            DECL 520-925                                DE2D 355
C                                                                        DE2D 360
      IF(ARRAY(M) .EQ. ZEE .OR. ARRAY(M) .EQ. SLASH) GO TO 60            DE2D 365
C                                                                        DE2D 370
C                                                                        DE2D 375
C                            DECL 935-1025                               DE2D 380
C                       'Z'                                              DE2D 385
C                                                                        DE2D 390
   17 IF(ARRAY(I) .NE. ZEE) GO TO 22                                     DE2D 395
C                                                                        DE2D 400
C                            DECL 1050-1115                              DE2D 405
C                                                                        DE2D 410
      IF(ARRAY(I) .NE. EX) GO TO 89                                      DE2D 415
C                                                                        DE2D 420
C                            DECL 1125-1135                              DE2D 425
C                                                                        DE2D 430
C                       'Y'                                              DE2D 435
C                                                                        DE2D 440
   89 IF(ARRAY(I) .NE. WHY) GO TO 23                                     DE2D 445
      INSTR(INDEX) = 101                                                 DE2D 450
      ARGUE = .TRUE.                                                     DE2D 455
      GO TO 54                                                           DE2D 460
C                                                                        DE2D 465
C                                                                        DE2D 470
C                            DECL 1140-1570                              DE2D 475
C                                                                        DE2D 480
     *     .OR. ARRAY(I2) .NE. DEE) GO TO 36                             DE2D 485
      IF(FCNCND) GO TO 990                                               DE2D 490
      I = I3                                                             DE2D 495
      IF(ARRAY(I3) .NE. EX) GO TO 90                                     DE2D 500
      INSTR(INDEX) = -1                                                  DE2D 505
      GO TO 54                                                           DE2D 510
C                                                                        DE2D 515
C                       'D/DY'                                           DE2D 520
C                                                                        DE2D 525
   90 IF(ARRAY(I3) .NE. WHY) GO TO 990                                   DE2D 530
      INSTR(INDEX) = 1                                                   DE2D 535
C                                                                        DE2D 540
C                            DECL 1600-1620                              DE2D 545
C                                                                        DE2D 550
     *     .OR. ARRAY(I3) .NE. DEE) GO TO 36                             DE2D 555
      IF(FCNCND) GO TO 990                                               DE2D 560
      I = I4 + 1                                                         DE2D 565
      IF(ARRAY(I4) .NE. EX) GO TO 91                                     DE2D 570
C                                                                        DE2D 575
C                            DECL 1635-1640                              DE2D 580
C                                                                        DE2D 585
      GO TO 54                                                           DE2D 590
   91 IF(ARRAY(I4) .NE. WHY) GO TO 990                                   DE2D 595
      INSTR(INDEX) = IFIX(FNUM(ARRAY,I1,I2) + 0.1)                       DE2D 600
      IF(I2 .NE. I1 + 1) GO TO 990                                       DE2D 605
C                                                                        DE2D 610
C                            DECL 1650-2150                              DE2D 615
C                                                                        DE2D 620
    7 IF(PRCARD .EQ. ARE) GO TO 71                                       DE2D 625
C                                                                        DE2D 630
C                            DECL 2160-2250                              DE2D 635
C                                                                        DE2D 640
      NRANGX = NDOM                                                      DE2D 645
      NRANGY = NDOM                                                      DE2D 650
    9 N = 0                                                              DE2D 655
      M = 0                                                              DE2D 660
C                                                                        DE2D 665
C                            DECL 2265-2290                              DE2D 670
C                                                                        DE2D 675
      IF(INSTR(I) .GE. 0) GO TO 92                                       DE2D 680
      N = N + INSTR(I)                                                   DE2D 685
      GO TO 11                                                           DE2D 690
   92 M = M + INSTR(I)                                                   DE2D 695
   11 IF(INSTR(I) .LT. 1000) GO TO 10                                    DE2D 700
      NRANGX = MIN0(NRANGX,NDOM + N)                                     DE2D 705
      NRANGY = MIN0(NRANGY,NDOM - M)                                     DE2D 710
      IF(INSTR(I) .LT. 4000) GO TO 9                                     DE2D 715
      IF(NRANGX .GE. 0) GO TO 93                                         DE2D 720
      NDOM = NDOM - NRANGX                                               DE2D 725
      NRANGY = NRANGY - NRANGX                                           DE2D 730
      NRANGX = 0                                                         DE2D 735
   93 IF(NRANGY .GE. 0) GO TO 12                                         DE2D 740
```

Figure 6.1.2b

```
      NDOM = NDOM - NRANGY                                              DE2D 740
      NRANGX = NRANGX - NRANGY                                          DE2D 745
      NRANGY = 0                                                        DE2D 750
   12 WRITE(6,105) NDOM,NRANGX,NRANGY                                   DE2D 755
      ND = NDOM + 1                                                     DE2D 760
      NDSQ = ND ** 2                                                    DE2D 765
      WRITE(6,101)                                                      DE2D 770
C                                                                       DE2D 775
C                              DECL 2335-2430                           DE2D 780
C                                                                       DE2D 785
C     FIND THE NUMBER OF ELEMENTS IN THE X DIRECTION                    DE2D 790
C                                                                       DE2D 795
      NELX = IFIX(FNUM(ARRAY,I,I1) + 0.1)                               DE2D 800
      IF(NELX .LE. 0) NELX = 1                                          DE2D 805
      XLNGTH = (X2 - X1) / NELX                                         DE2D 810
      DO 94 I = I1,80                                                   DE2D 815
      IF(ARRAY(I) .EQ. EQUAL) GO TO 95                                  DE2D 820
   94 CONTINUE                                                          DE2D 825
      GO TO 991                                                         DE2D 830
   95 I = I + 1                                                         DE2D 835
      Y1 = FNUM(ARRAY,I,I1)                                             DE2D 840
      IF(I1 .EQ. I) GO TO 991                                           DE2D 845
      I = I1 + 1                                                        DE2D 850
      Y2 = FNUM(ARRAY,I,I1)                                             DE2D 855
      IF(I1 .EQ. I) GO TO 991                                           DE2D 860
      I = I1 + 1                                                        DE2D 865
C                                                                       DE2D 870
C     FIND THE NUMBER OF ELEMENTS IN THE Y DIRECTION                    DE2D 875
C                                                                       DE2D 880
      NELY = IFIX(FNUM(ARRAY,I,I1) + 0.1)                               DE2D 885
      IF(NELY .LE. 0) NELY = 1                                          DE2D 890
      YLNGTH = (Y2 - Y1) / NELY                                         DE2D 895
      X2 = X1 - XLNGTH                                                  DE2D 900
      DO 97 J = 1,NELX                                                  DE2D 905
      X2 = X2 + XLNGTH                                                  DE2D 910
      Y1 = Y2 + YLNGTH                                                  DE2D 915
      DO 97 I = 1,NELY                                                  DE2D 920
      Y1 = Y1 - YLNGTH                                                  DE2D 925
      NELMTS = NELMTS + 1                                               DE2D 930
      WTX(NELMTS) = 1.0 / XLNGTH                                        DE2D 935
      WTY(NELMTS) = 1.0 / YLNGTH                                        DE2D 940
      XSTEP = XLNGTH / NDOM                                             DE2D 945
      YSTEP = YLNGHT / NDOM                                             DE2D 950
      X22 = X2 - XSTEP                                                  DE2D 955
C                                                                       DE2D 960
C     ADD THE POINTS IN EACH ELEMENT SEPARATELY                         DE2D 965
C                                                                       DE2D 970
      DO 96 J1 = 1,ND                                                   DE2D 975
      X22 = X22 + XSTEP                                                 DE2D 980
      Y11 = Y1 + YSTEP                                                  DE2D 985
      DO 96 I1 = 1,ND                                                   DE2D 990
      Y11 = Y11 - YSTEP                                                 DE2D 995
      NPT = NPT + 1                                                     DE2D1000
      XD(NPT) = X22                                                     DE2D1005
   96 YD(NPT) = Y11                                                     DE2D1010
      NPT1 = NPT - NDSQ + 1                                             DE2D1015
      WRITE(6,112) NELMTS, (XD(J), J = NPT1,NPT,ND)                     DE2D1020
      NPT1 = NPT1 - 1                                                   DE2D1025
      DO 97 I1 = 1,ND                                                   DE2D1030
      NPT1 = NPT1 + 1                                                   DE2D1035
   97 WRITE(6,113) YD(NPT), (J, J = NPT1,NPT,ND)                        DE2D1040
      PRCARD = ARE                                                      DE2D1045
      GO TO 1                                                           DE2D1050
C                                                                       DE2D1055
C     A CONTINUITY STATEMENT HAS BEEN READ                              DE2D1060
C                                                                       DE2D1065
  150 IF(PRCARD .NE. ARE) GO TO 151                                     DE2D1070
      PRCARD = EL                                                       DE2D1075
      I = 0                                                             DE2D1080
      J = 0                                                             DE2D1085
      NCCOND = 0                                                        DE2D1090
      WRITE(6,114)                                                      DE2D1095
  151 IF(PRCARD .NE. EL) GO TO 991                                      DE2D1100
      I3 = 0                                                            DE2D1105
      DO 152 I1 = 6,80                                                  DE2D1110
      I3 = I3 + 1                                                       DE2D1115
      NELC(I3) = IFIX(FNUM(ARRAY,I1,I2) + 0.1)                          DE2D1120
      I1 = I2                                                           DE2D1125
  152 CONTINUE                                                          DE2D1130
      NELC(I3 + 1) = 0                                                  DE2D1135
      WRITE(6,116) ARRAY(1),NELC                                        DE2D1140
      NCCOND = NCCOND - ND                                              DE2D1145
      INDEX = 1                                                         DE2D1150
      IF(ARRAY(1) .EQ. WHY) GO TO 154                                   DE2D1155
C                                                                       DE2D1160
C     PUT THE X CONTINUITY CONDITIONS IN THE ARRAY 'IELCX'              DE2D1165
C                                                                       DE2D1170
  153 NCCOND = NCCOND + ND                                              DE2D1175
      J = J + 1                                                         DE2D1180
      IELCX(J) = NELC(INDEX)                                            DE2D1185
      INDEX = INDEX + 1                                                 DE2D1190
      IF(NELC(INDEX) .NE. 0) GO TO 153                                  DE2D1195
      J = J + 1                                                         DE2D1200
      IELCX(J) = 0                                                      DE2D1205
      GO TO 1                                                           DE2D1210
C                                                                       DE2D1215
C     PUT THE Y CONTINUITY CONDITIONS IN THE ARRAY 'IELCY'              DE2D1220
C                                                                       DE2D1225
  154 NCCOND = NCCOND + ND                                              DE2D1230
      I = I + 1                                                         DE2D1235
      IELCY(I) = NELC(INDEX)                                            DE2D1240
      INDEX = INDEX + 1                                                 DE2D1245
      IF(NELC(INDEX) .NE. 0) GO TO 154                                  DE2D1250
      I = I + 1                                                         DE2D1255
      IELCY(I) = 0                                                      DE2D1260
      GO TO 1                                                           DE2D1265
   82 IF(PRCARD .NE. ARE) GO TO 155                                     DE2D1270
      I = 0                                                             DE2D1275
```

```
      J = 0                                                              DE2D1280
      NCCOND = 0                                                         DE2D1285
      GO TO 156                                                          DE2D1290
  155 IF(PRCARD .NE. EL) GO TO 991                                       DE2D1295
C                                                                        DE2D1300
C     UPON COMPLETION, ADD ENDING VALUES TO 'IELCX' AND 'IELCY'          DE2D1305
C                                                                        DE2D1310
  156 J = J + 1                                                          DE2D1315
      IELCX(J) = -1                                                      DE2D1320
      I = I + 1                                                          DE2D1325
      IELCY(I) = -1                                                      DE2D1330
      IF(ARRAY(1) .EQ. BLANK) GO TO 86                                   DE2D1335
C                                                                        DE2D1340
C                              DECL 2545-2800                            DE2D1345
C                                                                        DE2D1350
  112 FORMAT(1HO,60X,'ELEMENT NUMBER',I3 / 1HO,20X,10G11.3)              DE2D1355
  113 FORMAT(1HO,9X,G11.3,I6,9I11)                                       DE2D1360
  114 FORMAT(1H1 / 1HO,42X,'THE FOLLOWING ELEMENTS ARE TAKEN TO BE ADJAC DE2D1365
     *ENT' / 1HO,7X,'TYPE',12X,'ELMENTS'//)                              DE2D1370
  116 FORMAT(1H ,6X, A2,'ONT',3X,39I3)                                   DE2D1375
      END                                                                DE2D1380
```

Figure 6.1.2c

independent and can be taken to any computer installation which accepts Fortran programs.

In defining the instructions for a differential equation statement or a boundary condition statement, INSTR and INST2D operate by evaluating the branching structure and syntax tree of the statement by noting the number and locations of the symbols +, -, (, and). The syntax tree is then reduced to the canonical form

$$\sum_{s=1}^{S} D_s \, y = f \qquad (6.1.1)$$

by applying the distributive law to compound operators enclosed in parentheses. Instructions generated by this procedure are compatible with the subroutines DIFFEQ and DIFF2D.

A separate subprogram FNUM, given in Figure 6.1.3, is called by the subroutines INSTR and INST2D to determine the value of a number stored in Hollerith form in the Fortran language. FNUM establishes the machine-independent, free-format feature of numbers in DECL since it accepts numbers in any Fortran permissible form and in any location.

The diagnostic capabilities of INSTR and INST2D have been designed to detect most errors in DECL that lead to logical inconsistency. For this purpose, a number of parameters and conditions are checked in the programs after each symbol in a statement has been translated. If an error is spotted, the programs write an appropriate message and proceed to the next DECL program.

Conversational versions of the programs INSTR and INST2D for use in time-sharing situations have also been

Figure 6.1.3

```
      FUNCTION FNUM(ARRAY,I,J)
C
C     THIS FUNCTION SUBPROGRAM RETURNS THE NUMERICAL VALUE OF A
C     NUMBER STORED IN HOLLERITH FORM IN 'ARRAY' STARTING IN LOCATION
C     'I'
C
      LOGICAL NUMBER,MINUS,EXPON,DECMAL,ONLYE
      DIMENSION ARRAY(80),ANUM(15)
      DATA ANUM / 1H ,1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,
     *      1H.,1HE,1H-,1H+ /
C
C     INITIALIZE THE TYPES OF NUMBERS POSSIBLE
C
      NUMBER = .FALSE.
      MINUS = .FALSE.
      EXPON = .FALSE.
      DECMAL = .FALSE.
      ONLYE = .FALSE.
      NUM = 0
      J = I
C
C     DETERMINE THE TYPE OF CHARACTER IN LOCATION J
C
    1 DO 2 N = 1,15
      IF(ARRAY(J) .EQ. ANUM(N)) GO TO 8
    2 CONTINUE
    3 IF(EXPON) GO TO 6
    4 IF(.NOT. NUMBER) GO TO 990
    5 FNUM = NUM
C
C     ADD A MINUS SIGN
C
      IF(MINUS) FNUM = - FNUM
      IF(.NOT. DECMAL) RETURN
      GO TO 7
    6 IF(.NOT. NUMBER) GO TO 5
C
C     ADD THE EXPONENT
C
    7 LPOWER = LPOWER + NUM
      IF(MINUS) LPOWER = LPOWER - 2 * NUM
      FNUM = FNUM * 10.0 ** LPOWER
      RETURN
    8 J = J + 1
      IF(J .GT. 80) GO TO 3
      IF(N .EQ. 1) GO TO 9
      IF(N .GT. 11) GO TO 10
      IF(ONLYE) GO TO 5
C
C     THE CURRENT CHARACTER IS A NUMBER
C
      NUMBER = .TRUE.
      NUM = 10 * NUM + N - 2
      IF(.NOT. DECMAL) GO TO 1
      LPOWER = LPOWER - 1
      GO TO 1
C
C     BLANKS MAY NOT APPEAR IN A NUMBER, EXCEPT BEFORE AN E
C
    9 IF(NUMBER) ONLYE = .TRUE.
      GO TO 1
   10 N = N - 11
      GO TO (11,12,13,13),N
C
C     THE DECIMAL POINT IS LOCATED
C
   11 DECMAL = .TRUE.
      LPOWER = 0
      GO TO 1
   12 IF(EXPON .OR. .NOT. DECMAL .OR. .NOT. NUMBER) GO TO 990
C
C     THE CURRENT CHARACTER IS AN E
C
      EXPON = .TRUE.
      FNUM = NUM
      IF(MINUS) FNUM = - FNUM
      MINUS = .FALSE.
      NUMBER = .FALSE.
      ONLYE = .FALSE.
      NUM = 0
      GO TO 1
   13 IF(NUMBER) GO TO 3
C
C     THE CURRENT CHARACTER IS A PLUS OR MINUS SIGN
C
      IF(N .EQ. 4) GO TO 1
      MINUS = .TRUE.
      GO TO 1
C
C     THE INPUT ARRAY DOES NOT CONTAIN A NUMBER
C
  990 J = I
      RETURN
      END
```

written. However, since the programming changes required for this type of operation are minor, the listings of these versions of the programs are not presented here. The modifications are designed to make the input procedures more convenient in on-line operation. For example, the programs will prompt the user with the following statements

```
WHAT IS THE DIFFERENTIAL EQUATION?
STATE A BOUNDARY CONDITION
WHAT IS THE VALUE OF THE CONSTANT P?
IN REGION 1, WHAT ARE THE LIMITS OF THE
   X COORDINATE?
HOW MANY ELEMENTS ARE TO BE USED?
```

In this way, the operation of the program is made virtually foolproof and requires very little specialized knowledge.

Output programs have also been developed which write and graph the solutions of a differential equation in terms of their numerical values at the interpolation nodes. The program which writes the solutions is called RITE and is given in Figure 6.1.4. This subroutine has already been called by some of the programs in Chapter 5. A program to graph the solutions of ordinary differential equations is presented in Figure 6.1.5. Called GRAPH, the subroutine can plot up to ten functions simultaneously. Finally, Figure 6.1.6 contains a plotting program called PICT for use with two-dimensional partial differential equations. Typical output from these programs is shown in the following sections.

```
      SUBROUTINE RITE(A,M,N,MADDA,IPART)                                DECL9090
C                                                                       DECL9095
C     THIS SUBROUTINE WRITES THE 'M' BY 'N' MATRIX 'A' FOR OUTPUT BY    DECL9100
C     THE PROGRAM. IF 'IPART' IS FALSE, THAN THE LAST COLUMN OF 'A'     DECL9105
C     IS ASSUMED TO BE AN INHOMOGENEOUS SOLUTION AND IS WRITTEN FIRST   DECL9110
C                                                                       DECL9115
      LOGICAL IPART                                                     DECL9120
      DIMENSION A(1)                                                    DECL9125
      NM1 = N                                                           DECL9130
      IF(IPART .NE. 0) GO TO 2                                          DECL9135
      WRITE(6,100)                                                      DECL9140
      NM1 = N - 1                                                       DECL9145
      IADD = MADDA + M*NM1                                              DECL9150
      WRITE(6,101) (I,A(I + IADD),I = 1,M)                              DECL9155
      IF(NM1 .EQ. 0) RETURN                                             DECL9160
      WRITE(6,102)                                                      DECL9165
    2 IADD = MADDA - M                                                  DECL9170
      DO 1 J = 1,NM1                                                    DECL9175
      IADD = IADD + M                                                   DECL9180
    1 WRITE(6,101) (I,A(I + IADD),I = 1,M)                              DECL9185
      RETURN                                                            DECL9190
  100 FORMAT(1H-,53X,'THE INHOMOGENEOUS SOLUTION'/)                     DECL9195
  101 FORMAT(1H0,100(5X,5(I6,E18.6)/1H ))                               DECL9200
  102 FORMAT(1H-,54X,'THE HOMOGENEOUS SOLUTION'/)                       DECL9205
      END                                                               DECL9210
```

Figure 6.1.4

Figure 6.1.5

```
      SUBROUTINE GRAPH(X,Y,N,NPER)
C
C     THIS SUBROUTINE GRAPHS THE 'N' POINTS (X,Y) ON A STANDARD LINE
C     PRINTER. THE NUMBER 'N' MAY BE ANY MULTIPLE OF NPER, THE NUMBER
C     OF 'X' COORDINATE POINTS, SO THAT SEVERAL FUNCTIONS CAN BE
C     GRAPHED AT ONE TIME.
C
      DIMENSION X(1),Y(1),IY(1),IX(500),CHAR(111),ANUM(6),SYMBOL(10)
      COMMON CHAR,IX,IY
      DATA BLANK,VERT,HOR,PLUS,SYMBOL / 1H ,1H|,1H-,1H+,1H1,1H2,1H3,
     *     1H4,1H5,1H6,1H7,1H8,1H9,1H0 /
      IF(MOD(N,NPER) .NE. 0) RETURN
      IF(X(NPER) .EQ. X(1)) RETURN
C
C     THE POINTS ARE SCALED PROPERLY AND PUT IN THE INTEGER ARRAYS
C     'IX' AND 'IY'
C
      SCALEX = 110.1/(X(NPER) - X(1))
      YMAX = Y(1)
      YMIN = Y(1)
      DO 1 I = 1,N
      IF(Y(I) .GT. YMAX) YMAX = Y(I)
    1 IF(Y(I) .LT. YMIN) YMIN = Y(I)
      IF(YMAX .EQ. YMIN) RETURN
      SCALEY = 50.1/(YMAX - YMIN)
      DO 15 I = 1,NPER
   15 IX(I) = IFIX( (X(I) - X(1) )*SCALEX) + 1
      DO 2 I = 1,N
    2 IY(I) = IFIX( (Y(I) - YMIN) *SCALEY) + 1
C
C     'IXO' AND 'IYO' ARE THE LOCATIONS OF THE COORDINATE AXES
C
      IXO = MAXO(IFIX( -X(1)*SCALEX + 1.),1)
      IYO = MAXO(IFIX( -YMIN*SCALEY + 1.),1)
      IF(IXO .GT. 111) IXO = 1
      IF(IYO .GT. 51) IYO = 1
      WRITE(6,100)
      SCALEX = (X(NPER) - X(1))/5.
      SCALEY = (YMAX - YMIN) /5.
      YVALUE = YMAX + SCALEY
      LINE = 52
    3 LINE = LINE - 1
      IF(LINE .EQ. IYO) GO TO 16
      IF(LINE .EQ. IYO - 1) GO TO 10
      IF(LINE .LE. 0) RETURN
C
C     THE CHARACTER ARRAY 'CHAR' IS INITIALIZED
C
      DO 4 I = 1,111
    4 CHAR(I) = BLANK
      CHAR(IXO) = VERT
      GO TO 6
   16 DO 5 I = 1,111
    5 CHAR(I) = HOR
      DO 12 I = 1,111,22
   12 CHAR(I) = PLUS
    6 I = 0
      J2 = 0
    7 I = I + 1
C
C     'I' IS THE NUMBER OF THE FUNCTION GRAPHED
C
      IF(I .GT. 10) I = 1
      J1 = J2 + 1
      J2 = J2 + NPER
      JX = 0
      DO 14 J = J1,J2
      JX = JX + 1
      IF(IY(J) .NE. LINE) GO TO 14
C
C     IF THE Y COORDINATE OF THE POINT CORRESPONDS TO THE LINE TO BE
C     PRINTED, THE FUNCTION NUMBER IS PLACED IN THE APPROPRIATE
C     LOCATION
C
      CHAR(IX(JX)) = SYMBOL(I)
   14 CONTINUE
      IF(J2 .LT. N) GO TO 7
      IF(LINE .EQ. 25) GO TO 8
      IF(MOD(LINE - 1,10) .EQ. 0) GO TO 9
C
C     THE ARRAY 'CHAR' IS WRITTEN USING THE PROPER FORMAT
C
      WRITE(6,101) CHAR
      GO TO 3
    8 WRITE(6,102) CHAR
      GO TO 3
    9 YVALUE = YVALUE - SCALEY
      IF(CHAR(IXO) .NE. VERT) GO TO 13
      CHAR(IXO) = PLUS
   13 WRITE(6,103) YVALUE,CHAR
      GO TO 3
   10 ANUM(1) = X(1)
      DO 11 I = 2,6
   11 ANUM(I) = ANUM(I - 1) + SCALEX
      WRITE(6,104) ANUM
      GO TO 3
  100 FORMAT(1H1/1HO,55X,'GRAPH OF THE SOLUTION'///)
  101 FORMAT(1H ,15X,111A1)
  102 FORMAT(1H ,10X,1HY,4X,111A1)
  103 FORMAT(1H ,5X,G9.2,1X,111A1)
  104 FORMAT(1H ,10X,G9.2,2(13X,G9.2),6X,1HX,6X,2(G9.2,13X),G9.2)
      END
```

Figure 6.1.6

```
      SUBROUTINE PICT(X,Y,Z,N,NUMBER)                                   DE2D2625
                                                                        DE2D2630
      THIS SUBROUTINE PRODUCES A PERSPECTIVE GRAPH OF THE 'N' POINTS    DE2D2635
      '(X,Y,Z)' ON A STANDARD LINE PRINTER. THE INTEGER 'NUMBER'        DE2D2640
      INDICATES WHICH  OF SEVERAL SOLUTIONS STORED IN THE ARRAY 'Z' IS  DE2D2645
      TO BE GRAPHED.                                                    DE2D2650
                                                                        DE2D2655
      DIMENSION X(1),Y(1),Z(1)                                          DE2D2660
      DIMENSION XNEW(500),ZNEW(500),IX(500),IY(500),IZ(500)             DE2D2665
      DIMENSION CHAR(111),SYMBOL(10)                                    DE2D2670
      COMMON CHAR,XNEW,YNEW,IX,IY,IZ                                    DE2D2675
      DATA BLANK,VERT,HOR,PLUS,SLASH,WHY / 1H ,1H|,1H-,1H+,1H/,1HY /    DE2D2680
      DATA SYMBOL / 1HO,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9 /           DE2D2685
      NZ1 = N * (NUMBER - 1) + 1                                        DE2D2690
      NZN = NZ1 + N - 1                                                 DE2D2695
                                                                        DE2D2700
      THE POINTS ARE SCALED PROPERLY AND PUT IN THE INTEGER ARRAYS      DE2D2705
      'IX', 'IY' AND 'IZ'                                               DE2D2710
                                                                        DE2D2715
      CALL SCALE(X,1,N,XMAX,XMIN,85.1,XSCALE)                           DE2D2720
      CALL SCALE(Y,1,N,YMAX,YMIN,25.0,YSCALE)                           DE2D2725
      CALL SCALE(Z,NZ1,NZN,ZMAX,ZMIN,25.1,ZSCALE)                       DE2D2730
      IF(XSCALE .EQ. 0. .OR. YSCALE .EQ. 0. .OR. ZSCALE .EQ. 0.)        DE2D2735
     *     GO TO 990                                                    DE2D2740
      XSTEP = (XMAX - XMIN) * .259                                      DE2D2745
      NZ1 = NZ1 - 1                                                     DE2D2750
      DO 2 I = 1,N                                                      DE2D2755
      YDIFF = (Y(I) - YMIN) * YSCALE                                    DE2D2760
      IX(I) = IFIX((X(I) - XMIN) * XSCALE + YDIFF) + 1                  DE2D2765
      IZ(I) = IFIX((Z(I + NZ1) - ZMIN) * ZSCALE + YDIFF) + 1            DE2D2770
    2 IY(I) = IFIX(YDIFF * 0.395) + 1                                   DE2D2775
      WRITE(6,100)                                                      DE2D2780
      ZSCALE = (ZMAX - ZMIN) / 2.5                                      DE2D2785
      ZVALUE = 2. * ZMAX + ZSCALE                                       DE2D2790
      LINE = 52                                                         DE2D2795
    3 LINE = LINE - 1                                                   DE2D2800
      IF(LINE .EQ. 1) GO TO 16                                          DE2D2805
                                                                        DE2D2810
      THE CHARACTER ARRAY 'CHAR' IS INITIALIZED                         DE2D2815
                                                                        DE2D2820
      DO 4 I = 1,111                                                    DE2D2825
    4 CHAR(I) = BLANK                                                   DE2D2830
      CHAR(1) = VERT                                                    DE2D2835
      GO TO 6                                                           DE2D2840
   16 DO 5 I = 1,111                                                    DE2D2845
    5 CHAR(I) = HOR                                                     DE2D2850
      DO 12 I = 1,111,22                                                DE2D2855
   12 CHAR(I) = PLUS                                                    DE2D2860
    6 IF(LINE .EQ. 27) CHAR(27) = WHY                                   DE2D2865
      IF(LINE .GT. 25) GO TO 7                                          DE2D2870
      CHAR(LINE) = SLASH                                                DE2D2875
                                                                        DE2D2880
      IF THE Z COORDINATE OF A POINT CORRESPONDS TO THE LINE TO BE      DE2D2885
      PRINTED, A Y COORDINATE DEPENDENT SYMBOL IS PLACED IN THE         DE2D2890
      APPROPRIATE LOCATION                                              DE2D2895
                                                                        DE2D2900
    7 DO 14 J = 1,N                                                     DE2D2905
      IF(IZ(J) .NE. LINE) GO TO 14                                      DE2D2910
      CHAR(IX(J)) = SYMBOL(IY(J))                                       DE2D2915
   14 CONTINUE                                                          DE2D2920
      IF(LINE .EQ. 25) GO TO 8                                          DE2D2925
      IF(MOD(LINE - 1,10) .EQ. 0) GO TO 9                               DE2D2930
                                                                        DE2D2935
      THE ARRAY 'CHAR' IS WRITTEN USING THE PROPER FORMAT               DE2D2940
                                                                        DE2D2945
      WRITE(6,101) CHAR                                                 DE2D2950
      GO TO 3                                                           DE2D2955
    8 WRITE(6,102) CHAR                                                 DE2D2960
      GO TO 3                                                           DE2D2965
    9 ZVALUE = ZVALUE - ZSCALE                                          DE2D2970
      IF(CHAR(1) .NE. VERT) GO TO 13                                    DE2D2975
      CHAR(1) = PLUS                                                    DE2D2980
   13 WRITE(6,103) ZVALUE,CHAR                                          DE2D2985
      IF(LINE .EQ. 1) GO TO 10                                          DE2D2990
      GO TO 3                                                           DE2D2995
   10 CHAR(1) = XMIN                                                    DE2D3000
      DO 11 I = 2,6                                                     DE2D3005
   11 CHAR(I) = CHAR(I - 1) + XSTEP                                     DE2D3010
      WRITE(6,104) (CHAR(I),I = 1,6)                                    DE2D3015
      RETURN                                                            DE2D3020
  990 N = 0                                                             DE2D3025
      RETURN                                                            DE2D3030
  100 FORMAT(1H1/1HO,55X,'GRAPH OF THE SOLUTION'///)                    DE2D3035
  101 FORMAT(1H ,15X,111A1)                                             DE2D3040
  102 FORMAT(1H ,10X,1HZ,4X,111A1)                                      DE2D3045
  103 FORMAT(1H ,5X,G9.2,1X,111A1)                                      DE2D3050
  104 FORMAT(1H ,10X,G9.2,2(13X,G9.2),6X,1HX,6X,2(G9.2,13X),G9.2)       DE2D3055
      END                                                               DE2D3060
                                                                        DE2D3065
      SUBROUTINE SCALE(X,N1,N2,XMAX,XMIN,SIZE,XSCALE)                   DE2D3070
                                                                        DE2D3075
      THIS SUBROUTINE DETERMINES THE MINIMUM AND MAXIMUM VALUES AND     DE2D3080
      A SCALE FACTOR FOR THE ARRAY 'X'.                                 DE2D3085
                                                                        DE2D3090
      DIMENSION X(1)                                                    DE2D3095
      XMAX = X(N1)                                                      DE2D3100
      XMIN = X(N1)                                                      DE2D3105
      XSCALE = 0.                                                       DE2D3110
      DO 1 I = N1,N2                                                    DE2D3115
      IF(X(I) .GT. XMAX) XMAX = X(I)                                    DE2D3120
    1 IF(X(I) .LT. XMIN) XMIN = X(I)                                    DE2D3125
      IF(XMAX .EQ. XMIN) RETURN                                         DE2D3130
      XSCALE = SIZE / (XMAX - XMIN)                                     DE2D3135
      RETURN                                                            DE2D3140
      END
```

6.2 ORDINARY DIFFERENTIAL EQUATIONS

A computer program for the automatic solution of arbitrary linear ordinary differential equations is now virtually complete. The only remaining step is to provide a main program to link the various subprograms together in the proper order. Such a main program is given in Figure 6.2.1. The complete program package, including this main program and twenty-one independent subprograms,requires 190 K bytes of core storage. A large part of these memory requirements is for the three arrays D, C and COM in which all of the major matrix operations are performed.

The program has been extensively tested with both standard differential equations and eigenvalue boundary value problems and some of these results will be presented here. For the first example, consider the differential equation (5.2.3), the solution of which is the exponential function. A complete DECL program listing for this problem is as follows

```
* 8    DIFFERENTIAL EQUATION WITH EXPONENTIAL SOLUTION
DE:   D/DX(Y) = Y                                    (6.2.1)
BC:   Y(0.0) = 1.0                                   (6.2.2)
X = 0.0, 10.0    (4 ELEMENTS)                        (6.2.3)
WRITE PARTICULAR SOLUTION
GRAPH PARTICULAR SOLUTION
```

This program specifies that equation (5.2.3) is to be solved using four equal eighth order elements in the interval [0.0, 10.0].

The above program was run on an IBM 360/75 computer (as were all examples in this thesis) and required 1.8 seconds

Figure 6.2.1

```
C
C     THIS PROGRAM WAS WRITTEN BY ZOLTAN CSENDES OF MCGILL UNIVERSITY
C     MONTREAL , QUEBEC, TO SOLVE ARBITRARY LINEAR ORDINARY
C     DIFFERENTIAL EQUATIONS. THE DIFFERENTIAL EQUATIONS ARE READ IN
C     THE DECL COMPUTER LANGUAGE AND THEIR GENERAL SOLUTIONS ARE
C     RETURNED IN TERMS OF BOTH POINT VALUES AND A GRAPHICAL PLOT.
C     PARTICULAR SOLUTIONS ARE ALSO PRODUCED, IF BOUNDARY CONDITIONS
C     ARE SPECIFIED.
C
      DIMENSION INST(500),CONST(500),XD(500),WT(100),DN(8000),C(8000),
     *     COM(8000),YR(4000),YI(3900),ROOTR(100),ROOTI(100),XBC(300)
      EQUIVALENCE (YR(1),C(4001)),  (ROOTR(1),DN(3901)),
     *      (ROOTI(1),DN(4001)),  (YI(1),DN(4101))
      COMMON COM
      LOGICAL LAMDA,RITEP,RITEG,PLOTP,PLOPG
C
C     READ THE DIFFERENTIAL EQUATION, BOUNDARY CONDITIONS AND
C     GEOMETRIC INFORMATION
C
    1 CALL INSTR(INST,CONST,NUMDE,XBC,XD,WT,NELMTS,NRANGE,NDOM,
     *     LAMDA,RITEP,RITEG,PLOTP,PLOTG,NEIGWR,NEIGPL)
      NR = NRANGE + 1
      ND = NDOM + 1
      NULLD = ND - NR
C
C     SOLVE THE DIFFERENTIAL EQUATION IN EACH ELEMENT SEPARATELY
C
      CALL DIFFEQ(DN,MDN,WT,NELMTS,INST,CONST,XD,NR,ND,
     *     NULLD,NULLV,LAMDA)
      IF(ND .EQ. 0) GO TO 1
      NDPTS = MDN*NELMTS
C
C     MAKE THE ELEMENTAL SOLUTIONS CONTINUOUS AND PRINT THE GENERAL
C     SOLUTION
C
      CALL CONTIN(DN,NR,ND,MDN,NULLV,C,NDPTS,NONHOM,NULLD,NULLV,
     *     NELMTS,WT,LAMDA,RITEG)
      IF(ND .EQ. 0) GO TO 1
      IF(LAMDA .OR. .NOT. PLOTG) GO TO 2
      N = NDPTS*NULLV
C
C     GRAPH THE GENERAL SOLUTION
C
      CALL GRAPH(XD,C,N,NDPTS)
C
C     ADD THE BOUNDARY CONDITIONS
C
    2 CALL BOUND(C,NDPTS,NONHOM,NULLD,NULLV,DN,NWBC,NR,ND,
     *     INST,CONST,XBC,NUMDE,XD,NELMTS,WT,LAMDA,RITEP)
      IF(ND .EQ. 0) GO TO 1
      NDPT = ND * NELMTS
      N = NDPT * NWBC
      IF(.NOT. LAMDA) GO TO 3
C
C     CALCULATE THE EIGENVALUES AND EIGENVECTORS
C
      CALL EIGEN(DN,ND,NELMTS,NDPTS,NWBC,C,YR,YI,ROOTR,ROOTI,NEIGWR)
      IF(ND .EQ. 0) GO TO 1
      N = NDPT * NEIGPL
      GO TO 4
    3 IF(.NOT. PLOTP) GO TO 1
C
C     GRAPH THE PARTICULAR SOLUTION
C
    4 CALL GRAPH(XD,DN,N,NDPT)
      GO TO 1
      END
```

of C.P.U. time for solution. The value of the solution at x=10.0 was 22025.7, which is correct to five significant figures. This value is considered to be round-off error limited since the IBM 360/75 has a 24-bit mantissa in single precision.

Equation (6.2.2) has also been solved with the differential equation solving program using first, second, fourth and eighth order polynomial approximations and one, two and four element subdivisions in the interval [0.0, 10.0]. In each case, the difference between the answer and the exact solution was calculated and the L_2 norm of this difference function was evaluated. A similar analysis was performed for the initial value problem.

DE: D2/DX2(Y) = -Y (6.2.4)

BC: Y(0.) = 0.0 (6.2.5)

BC: D/DX(Y(0.)) = 1.0

using second, fourth and eighth order polynomials in the interval [0.0, 10.0]. Figure 6.2.2 contains a plot of the L_2 norm of the computed error functions against the number of interpolation nodes for these problems. The lower limiting value of approximately 10^{-4} is interpreted to be the result of round-off error accumulation. The convergent behavior of these error norms can be approximated fairly well with the following empirical formula

$$|| y - y \text{ appox } || = \frac{40.}{(n-p+1)^3} \left(\frac{8}{N}\right)^n \qquad (6.2.6)$$

where n is the polynomial order, p is the order of the of the differential operator and N is the total number of points. This behavior indicates that higher order polynomials result not only in more accurate solutions for a given number

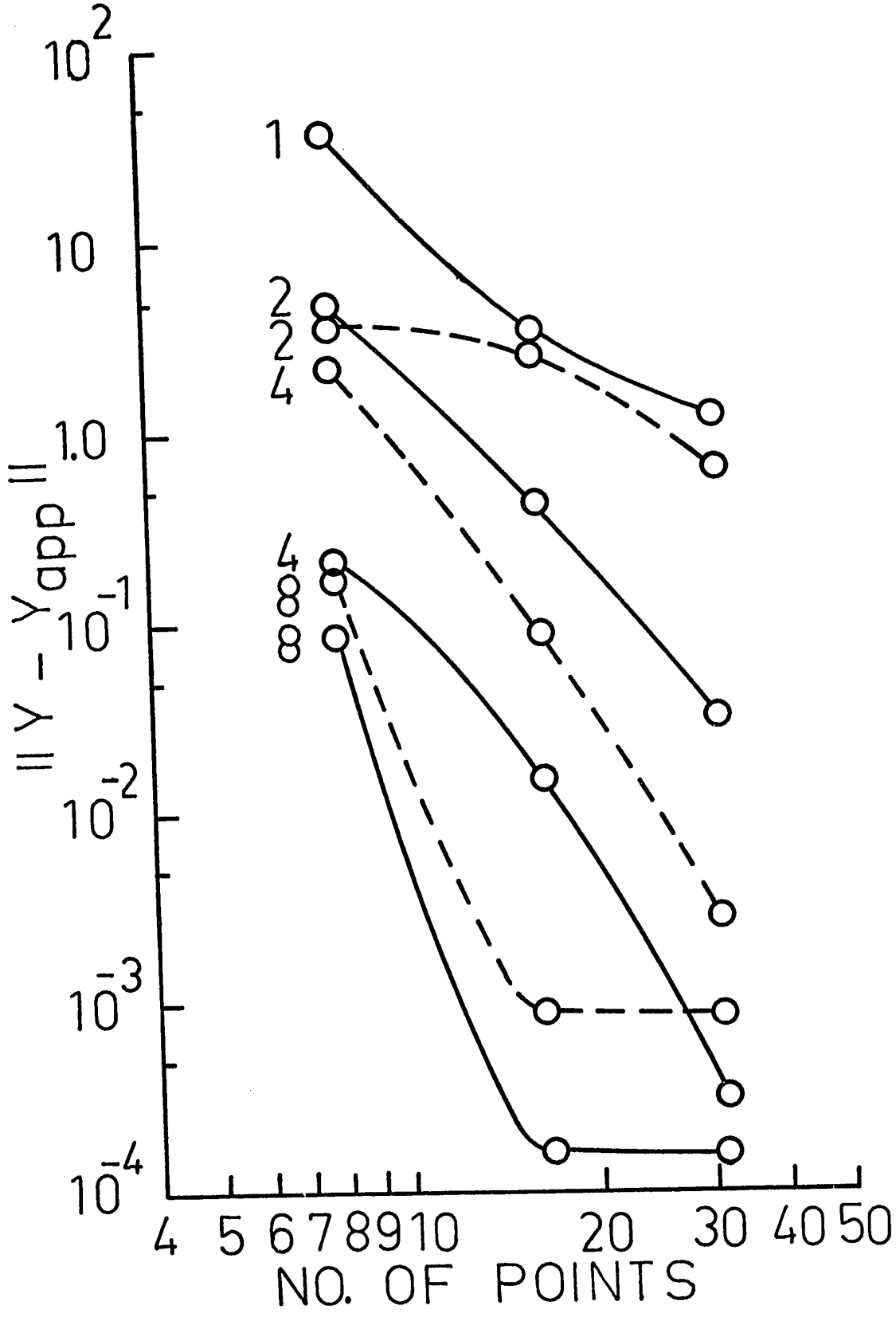


Figure 6.2.2

L_2 norm of the error in the solution. (6.2.1) ——; (6.2.4) ---.

of points but also in a faster convergence rate.

As a practical example of the utility of the differential equation solving program in finding general solutions for engineering problems, consider the equation

```
DE:  D3/DX3(Y)+(A+B*SIN(X))*D2/DX2(Y)
     +(C+D*SIN(X)+B*COS(X))*D/DX(Y)+E*Y = 0     (6.2.7)
```

which governs the behavior of hydraulic copying mechanisms used in metal cutting [65]. If the homogeneous solutions of this equation decay, the mechanism is stable, if they grow, it is unstable.

Solutions of this problem have been obtained by choosing three linearly independent initial conditions, integrating (6.2.7) numerically for each case, and analyzing the results to determine their growth or decay [65]. When solved with the differential equation solving program of this thesis, none of these complicated procedures is necessary;the equation is simply read in as it appears in (6.2.7), and no boundary conditions are specified. The program automatically returns a full set of homogeneous solutions and, provided the region used encompasses several periods, it is only necessary to observe the shape of the solution in order to determine its stability.

To be specific, let the values of the constants in (6.2.7) be

```
CST:A = 0.55
CST:B = 0.08
CST:C = 0.825
CST:D = 0.004
CST:E = 0.34
```

The general solution of the equation in the interval X=0. to 30. is shown in Figure 6.2.3. It is obviously stable. If, however, the value of E is

CST:E = 0.76

the unstable behavior given in Figure 6.2.4 results. These conclusions are confirmed by reference [65]. The graphs in Figures 6.2.3 and 6.2.4 were taken directly from the program output and each analysis, using five ninth order elements, required only 3.0 seconds on an IBM 360/75.

The next examples will indicate the application of the automatic differential equation solving program to eigenvalue boundary value problems. Consider, first, the Mathieu equation

DE: D2/DX2(Y) -2. *Q*COS(2.*X)*Y = A*Y (6.2.8)

where Q is a parameter. The solution interval of interest is $[0,2\pi]$ and, in order to obtain a complete set of particular solutions, both the Dirichlet and the Neumann boundary conditions need to be specified at the endpoints. Table 6.2.1 contains the first three eigenvalues calculated for these problems by the program in single precision on an IBM 360/75 along with the exact eigenvalues for several values of Q. In these calculations, two ninth-order elements were used and each computation required 3.3 seconds of execution time Note that all eigenvalues produced by the program are accurate to at least four significant figures. The corresponding eigenvectors displayed similar accuracy. Two of the graphs of the approximate eigenvectors which were produced automatically by the subroutine GRAPH for the Mathieu equation

Figure 6.2.3

The DECL computer program graph of the solution of the copying problem.

E = 0.34

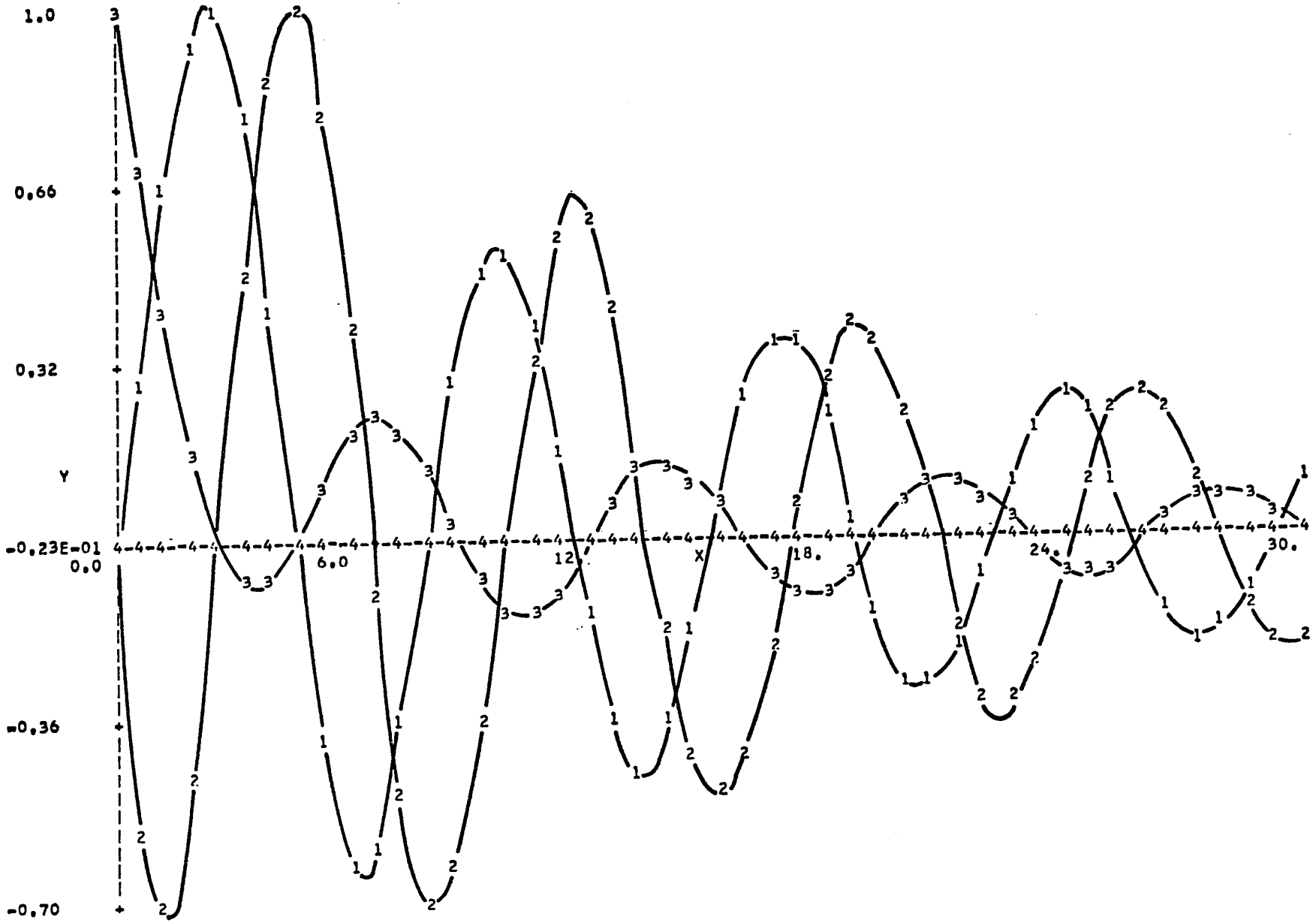


Figure 6.2.4

The DECL computer program graph of the solution of the copying problem.

E = 0.76

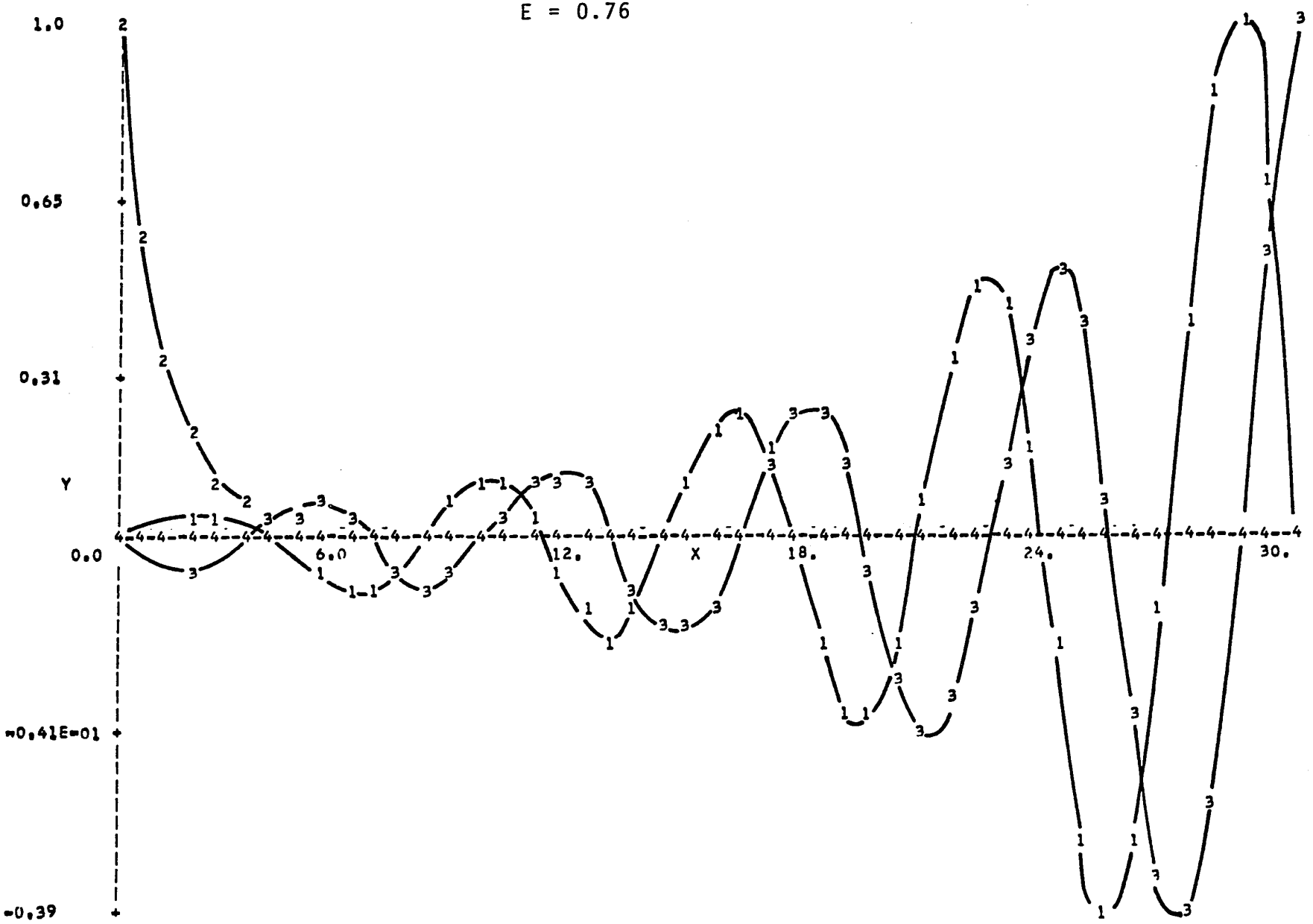


TABLE 6.2.1

EIGENVALUES OF THE MATHIEU EQUATION N = NEUMANN, D = DIRICHLET

q	Boundary Conditions		Number 1		Number 2		Number 3	
	x = 0	x = L	Calculated	Exact (66)	Calculated	Exact (66)	Calculated	Exact (66)
1	N	N	-.4545	-.4551	4.37137	4.37130	16.0332	16.0338
	D	N	-.1093	-.1102	9.0503	9.0477	25.0178	25.0208
	N	D	1.8577	1.8591	9.0770	9.0784	25.0205	25.0208
	D	D	3.9181	3.9170	16.0306	16.0330	36.0120	36.0143
10	N	N	7.7163	7.7174	-13.9359	-13.9369	21.1046	21.1046
	D	N	7.9855	7.8861	-13.9351	-13.9365	26.7693	26.7664
	N	D	-2.3994	-2.3991	15.5006	15.5028	27.6989	27.7038
	D	D	-2.3824	-2.3822	17.3761	17.3814	37.4167	37.4199
20	N	N	1.1577	1.1543	27.5919	27.5946	-31.3087	-31.3134
	D	N	1.1638	1.1607	28.4692	28.4682	-31.3084	-31.3134
	N	D	-14.487	-14.491	15.3736	15.3958	36.6396	36.6450
	D	D	-14.488	-14.491	15.4662	15.4940	40.5943	40.5897

are presented in Figures 6.2.5 and 6.2.6.

The next example presented is the analysis of the longitudinal vibration of a non-prismatic rod. This example is used because it is a semi-classical problem familiar to most engineers and yet presents difficult differential equations to solve in all but a few special cases. In addition, it is also a problem of considerable practical importance in ultrasonic processing and has a number of known solutions which may be used to assess the accuracy of the results [67].

The four shapes solved were the conical rod, the exponentially tapered rod, the catenoidal rod and a non-uniformly tapered rod having no simple analytical description. For the conical vibrating rod, the differential equation to be solved is

```
DE: D2/DX2(Y) + 2.0*POW-1(X-0.4)*D/DX(Y)=K*Y          (6.2.9)
```

and the boundary conditions are

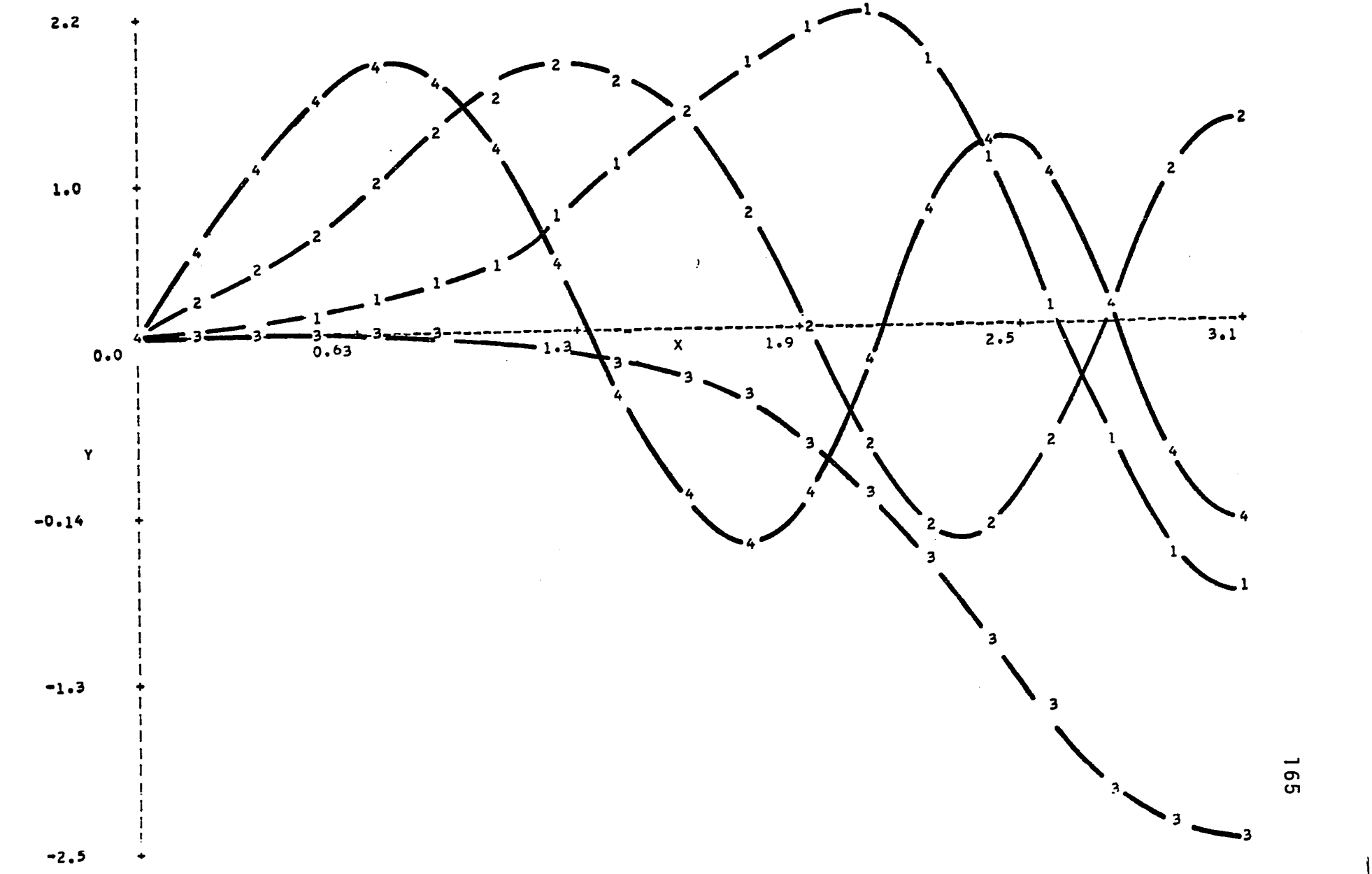
```
BC: Y(0.0) = 0.                                      (6.2.10)
BC: D/DX(Y(1.0)) = 0.
```

In this case, the solution interval [0.0, 1.0] was divided into two equal ninth order elements.

The problem required 3.8 seconds of execution time to solve for the eigenvalues and eigenvectors of the system, including a graphical plot of the eigenvectors. The eight lowest eigenvalues are presented in Table 6.2.2 along with their analytically computed values. Notice that the first seven eigenvalues are accurate to four significant figures and have an apparently random error distribution. These errors can be attributed directly to round-off error accumulation in the

Figure 6.2.5

The first four eigenvectors of the Mathieu equation. Y(0) = 0, D/DX(Y(L))=0.



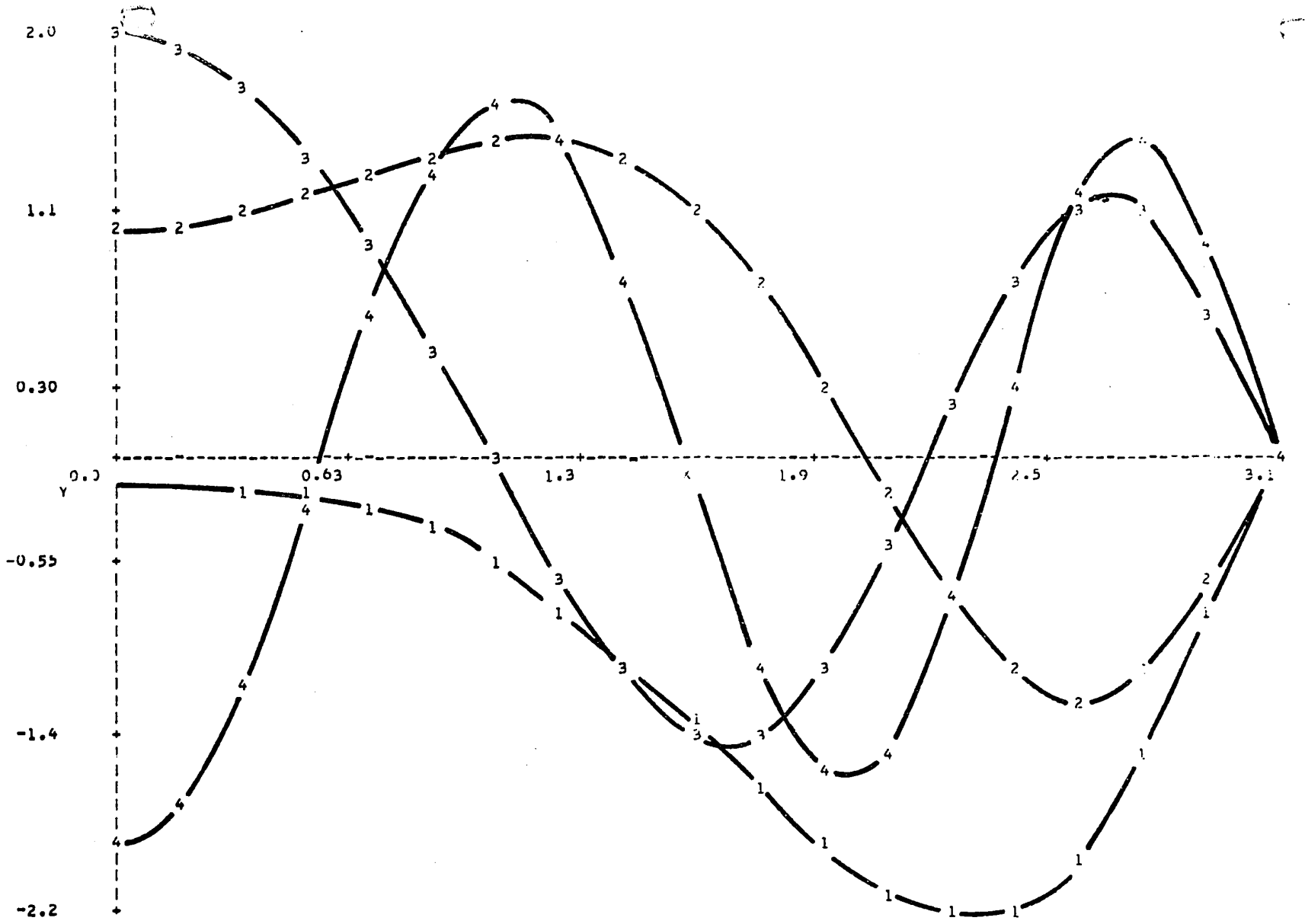


Figure 6.2.6

The first four eigenvectors of the Mathieu equation. D/DX(Y(0)) = 0, Y(L) = 0.

TABLE 6.2.2

Longitudinal Vibration Eigenvalues of Non-uniform Rods

	Cone Shaped		Exponential		Catenoidal	
Eigenvalue Number	Calc.	Exact.	Calc.	Exact	Calc.	Exact
1	0.80996	0.80976	.77317	.77310	6.1751	6.1730
2	20.7603	20.7654	21.2535	21.2583	25.9185	25.9122
3	60.2590	60.2520	60.750	60.745	65.387	65.390
4	119.473	119.472	119.967	119.964	124.611	124.608
5	198.39	198.43	198.921	198.922	203.557	203.565
6	297.04	297.13	297.76	297.62	302.11	302.26
7	415.62	415.56	415.99	416.05	420.67	420.70
8	554.7	553.7	555.0	554.2	560.7	558.9

computation. The eighth eigenvalue is, however, slightly worse, as were the successively higher ones, and in these the discretization error is dominant.

The eigenvectors exhibited a similar behavior. The accuracy of the first to seventh eigenvectors was limited by round-off error to four significant figures while eigenvectors higher than the seventh were less accurate. It would appear that above the seventh eigenvector two ninth-order polynomials cannot approximate the true solution with this accuracy. Consequently, if for some reason more accuracy is desired with these high-order eigenvalues and eigenvectors, the problem should be run with the solution region divided into more than two sections.

The exponentially tapered rod and the catenoidal rod were solved using similar data sets, except that their differential equation statements were [68]

```
DE: D2/DX2(Y) -2.5055 *D/DX(Y) = K*Y                    (6.2.11)
```

for the exponentially tapered rod and

```
DE: D2/DX2(Y) -2.*B*(EXP(B* (1-X))-EXP(-B*(1-X)))
      * POW-1(EXP(B*(1-X)) + EXP(-B*(1-X)))*D/DX(Y)
        = K * Y                                          (6.2.12)
CST: B = 1.925
```

for the catenoidal rod.

The eight lowest eigenvalues of these problems are also given in Table 6.2.2, along with the exact values. The agreement is similar to the conical rod.

The non-uniformly tapered rod shown in Figure 6.2.7 was also solved. In this case the differential equation was specified by

DE: D2/ DX2(Y)+FN1 (X) *D/DX (Y) = -K*Y (6.2.13)

The function FN1 (X) was supplied by providing the numerical values of the curve in Figure 6.2.7 at the nodal points of two ninth-order interpolation polynomials.

Part of the output from the computer program is shown in Figures 6.2.8 and 6.2.9. The eigenvalues and eigenvectors are given by two sets of numbers, the first of which gives the real part of the solution and the second the imaginary part. In this case, the problem is self-adjoint and the imaginary

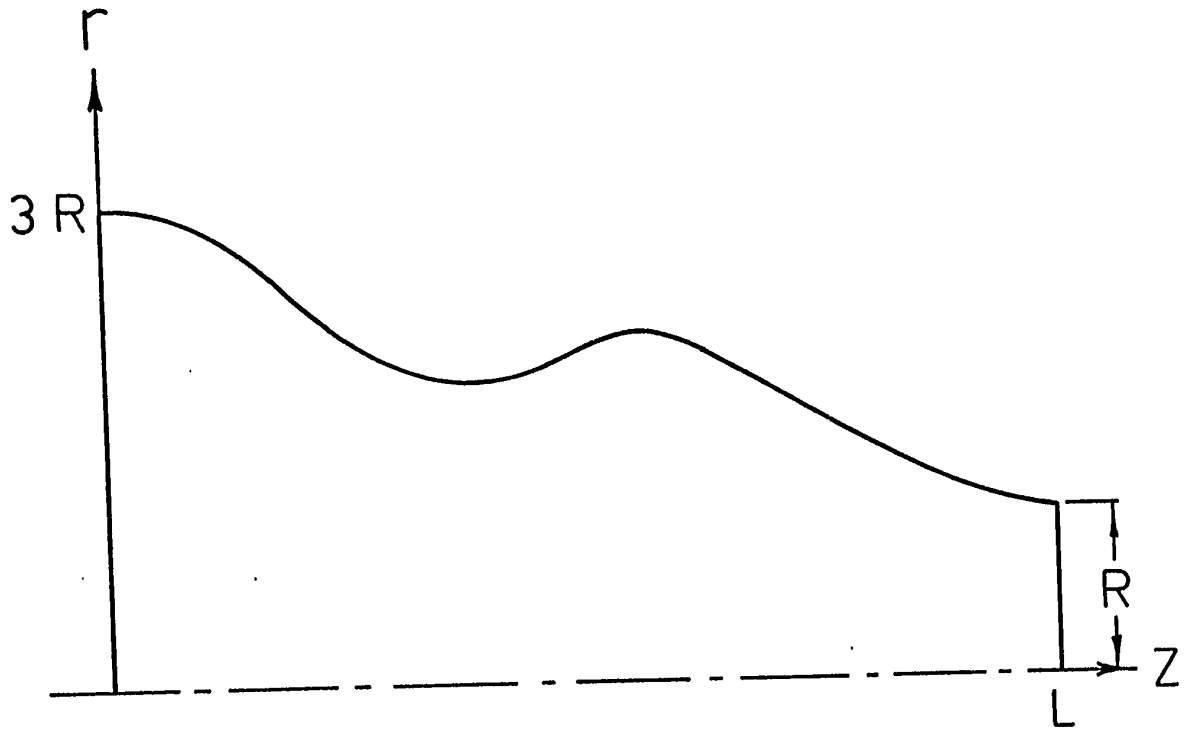


Figure 6.2.7
Profile of the non-uniformly tapered rod analyzed in Figures 6.2.8 and 6.2.9.

Figure 6.2.8

Part of the computer program output for the vibrating rod of Figure 6.2.7.

**** 'COKE BOTTLE' SHAPED VIBRATING ROD ****

THE DIFFERENTIAL EQUATION IS

D2/DX2(Y) + FN1(X) * D/DX(Y) = - K * Y

THE BOUNDARY CONDITIONS ARE

Y(1) = 0

D/DX(Y(3)) = 0

THE GENERAL SOLUTION

THE EIGENVALUES

1	0.332795E 01	2	0.266087E 02	3	0.612339E 02	4	0.122639E 03	5	0.201319E 03
6	0.300963E 03	7	0.415225E 03	8	0.552151E 03	9	0.738636E 03	10	0.934256E 03
1	0.0	2	0.0	3	0.0	4	0.0	5	0.0
6	0.0	7	0.0	8	0.0	9	0.0	10	0.0

EIGENVECTOR NUMBER 1

1	0.122405E-06	2	0.111557E 00	3	0.224610E 00	4	0.336788E 00	5	0.453565E 00
6	0.581843E 00	7	0.721721E 00	8	0.865722E 00	9	0.100463E 01	10	0.113229E 01
11	0.113229E 01	12	0.124178E 01	13	0.133161E 01	14	0.140701E 01	15	0.147181E 01
16	0.152740E 01	17	0.157367E 01	18	0.160931E 01	19	0.163220E 01	20	0.164022E 01
1	0.0	2	0.0	3	0.0	4	0.0	5	0.0
6	0.0	7	0.0	8	0.0	9	0.0	10	0.0
11	0.0	12	0.0	13	0.0	14	0.0	15	0.0
16	0.0	17	0.0	18	0.0	19	0.0	20	0.0

EIGENVECTOR NUMBER 2

1	0.656569E-06	2	-0.381497E 00	3	-0.741593E 00	4	-0.104507E 01	5	-0.128131E 01
6	-0.143854E 01	7	-0.149311E 01	8	-0.142665E 01	9	-0.124162E 01	10	-0.956854E 00
11	-0.956856E 00	12	-0.612743E 00	13	-0.243629E 00	14	0.143087E 00	15	0.539966E 00
16	0.931725E 00	17	0.129451E 01	18	0.159634E 01	19	0.180062E 01	20	0.187432E 01
1	0.0	2	0.0	3	0.0	4	0.0	5	0.0
6	0.0	7	0.0	8	0.0	9	0.0	10	0.0
11	0.0	12	0.0	13	0.0	14	0.0	15	0.0
16	0.0	17	0.0	18	0.0	19	0.0	20	0.0

parts of the eigenvalues and eigenvectors are zero. (Non-self-adjoint problems may yield complex eigensystem solutions). Figure 6.2.9 contains the graph of the eigenvectors which was automatically produced by the program.

The differential equation which appears in the analysis of the transverse vibration of a beam is generally regarded to be a more difficult equation to solve than the longitudinal equation because of the higher order derivatives involved. Coded in DECL, the equation which governs the transverse vibrations of a non-prismatic beam is [68]

DE: D2/ DX2(FN1 (X)*(D2/ DX2(Y))) = K *FN2 (X)*Y (6.2.14)

where FN1 (X) is the product of Young's modulus and the moment of inertia of the beam and FN2(X) is the product of the longitudinal density of the beam and its cross-sectional area. The boundary conditions of interest are those of simply supported beams

BC: Y(0) = 0.

BC: Y(1) = 0.

BC: D2/DX2(Y(0.)) = 0.

BC: D2/DX2(Y(1.)) = 0.

and beams with clamped ends

BC: Y(0.) = 0.

BC: Y(1.) = 0.

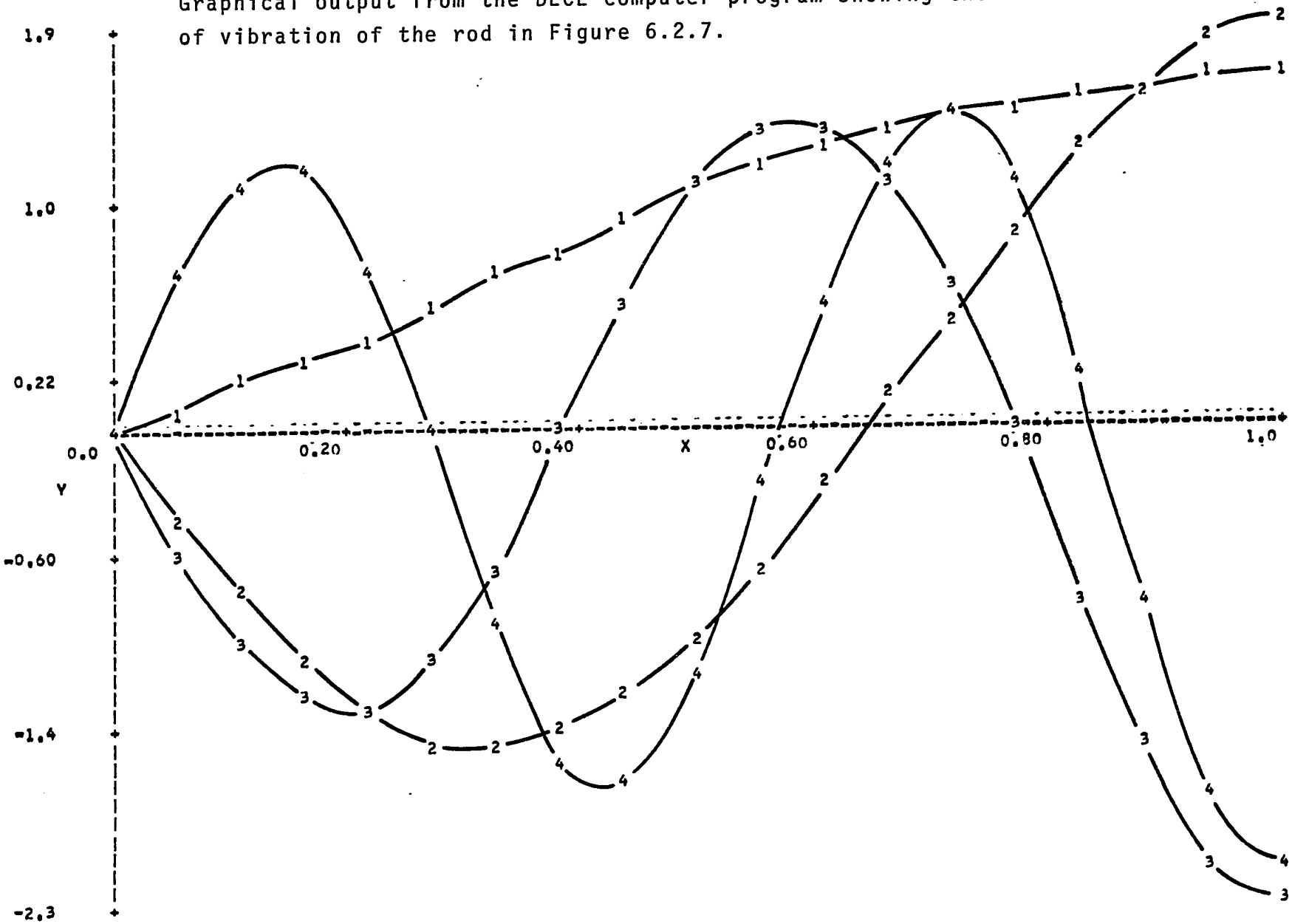
BC: D/DX(Y(0.)) = 0.

BC: D/DX(Y(1.)) = 0.

In order to demonstrate the accuracy of the program with this problem,the first case solved was that of a uniform

Figure 6.2.9

Graphical output from the DECL computer program showing the first four modes of vibration of the rod in Figure 6.2.7.



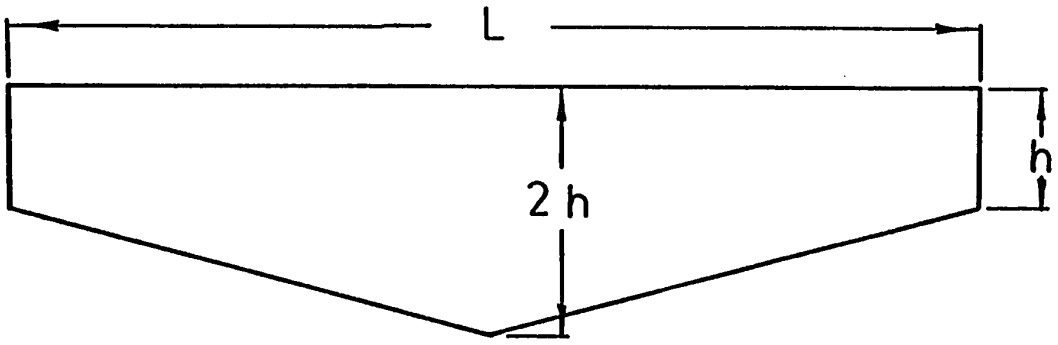


Figure 6.2.10
Profile of the haunched beam corresponding to Table 6.2.3. h = L/10.0.

beam. Again using two ninth-order elements, the problems required 3.5 seconds of execution time on the IBM 360/75 and the first four eigenvalues are presented in Table 6.2.3. This time it was found that only the first six eigenvalues were accurate to four significant figures.

The program was also used to analyze the vibration of the haunched beam [69] shown in Figure 6.2.10. The resulting eigenvalues with both sets of boundary conditions are given in Table 6.2.3.

Longitudinally accelerated transversely vibrating uniform beams with clamped ends were also analyzed with the program. In this case, the differential equation card read by the program was

DE: D4/DX4(Y)+A*D/DX((B-X)*D/DX(Y)) = K*Y (6.2.15)

The eigenvalues generated by the program are presented in Table 6.2.4 along with variationally derived approximate

TABLE 6.2.3

Natural Frequencies of Vibrating Beams

Mode Number	Prismatic Beam				Haunched Beam	
	Simply Supported		Clamped Ends		Simply Supported	Clamped Ends
	Calculated	Exact (nπ)	Calculated	Exact		
1	3.1434	3.1415	4.7295	4.7300	.50089	.75222
2	6.2838	6.2832	7.8539	7.8532	.91343	1.17049
3	9.4241	9.4247	10.9949	10.9956	1.38097	1.63003
4	12.5673	12.5664	14.1387	14.1372	1.82442	2.07074
5	15.719	15.708	17.295	17.279	2.29374	2.51808
6	19.01	18.85	20.73	20.42	2.7978	3.0434

TABLE 6.2.4

Eigenvalues of Accelerated Clamped Beams

Constant Values	Eigenvalue Number	Program Results	Rayleigh-Ritz Solution [6]	Solution by Kato's Method [6]
A = 25.0	1	261.14	264.34	241.50
B = 0.25	2	2934.3	2932.2	2881.4
	3	12,757.8	12,756.1	12,669.1
	4	36,734.1	36,720.5	36,635.3
A = 25.0	1	102.64	104.34	51.29
B = 0.75	2	2354.61	2349.54	2237.66
	3	11,534.2	11.519.0	11.337.2
	4	34,586.8	34,574.8	34,344.0
A = 100.0	1	464.24	464.42	349.23
B = -0.50	2	3739.95	3738.80	3492.14
	3	14,553.6	14,536.3	14,210.1
	4	39,854.8	39,855.6	39,429.1

lower and upper bounds [70] for some value of the constants A and B.

6.3 NONLINEAR ORDINARY DIFFERENTIAL EQUATIONS

It was shown in section 5.6 that the solution of many nonlinear differential equations can be accomplished via Newton's method by solving a sequence of linear differential equations. Since a computer program has been developed in this thesis which is able to solve any linear ordinary differential equation, the extension of this program to solving nonlinear ordinary differential equations is straightforward. By converting the program to iterative usage, it is possible to determine the Newton sequence of solutions which converge, if they do, to a solution of the original nonlinear equation.

In order to make the linear program suitable for solving a sequence of linear differential equations, the existing programs need to be modified slightly, as has already been described in section 5.6. Figure 6.3.1 contains a main program which incorporates these modifications. This program has been designed to call the subroutines DIFFEQ, CONTIN and BOUND repeatedly, until the successive particular solutions generated by BOUND have converged satisfactorily.

In defining the instructions which determine the differential equation to be solved, the new program calls the subroutine INSTR, which was used in the original non-iterative program. The only additional data that need to be specified

Figure 6.3.1a

```
C
C     THIS PROGRAM WAS WRITTEN BY ZOLTAN CSENDES OF MCGILL UNIVERSITY,
C     MONTREAL QUEBEC, TO SOLVE ARBITRARY NONLINEAR ORDINARY
C     DIFFERENTIAL EQUATIONS BY NEWTON ITERATION. THE QUASI-LINEAR
C     FORM OF THE DIFFERENTIAL EQUATION MUST BE READ IN THE DECL
C     COMPUTER LANGUAGE, FOLLOWED BY A DATA CARD GIVING THE NUMBER
C     OF ITERATIONS, THE NUMBER OF THE MODE THE SOLUTION IS TO CONVERGE
C     TO ( ONLY IN EIGENVALUE PROBLEMS), AND THE INITIAL VALUE OF THE
C     SOLUTION.
C
C                              DECL 50-75
C
      DIMENSION ENORM(100)
      COMMON /ANS/ AA,NEADD(10), E(3000),EDERIV(3000)
      LOGICAL LAST
C
C                              DECL 80-120
C
      READ(5,100) LAS,IGENUM,AA
      WRITE(6,101) AA
      ITER = 0
      LAST = .FALSE.
      NUMDES = NUMDE
      NDPTS = ND*NELMTS
      NEADD(1) = 4*NDPTS
      N = ND + 1
      DO 5 I = 1,NULLD
      N = N - 1
    5 NEADD(I + 1) = NEADD(I) + N * NELMTS
      J = NEADD(NULLD + 1) + NR * NELMTS
C
C     INITIALIZE THE SOLUTION
C
      DO 6 I = 1,J
    6 E(I) = AA
    9 ITER = ITER + 1
      MO = MOD(ITER - 1,5)
      NEADD(1) = MO * NDPTS
      NUMDE = NUMDES
C
C                              DECL 125-290
C
     *       IGENUM,LAST)
      IF(MO .EQ. 0) GO TO 1
      ENORM(ITER) = 0.
      J = NEADD(1) - NDPTS
      IF(J .LT. 0) J = 4 * NDPTS
C
C     EVALUATE THE DIFFERENCE NORM
C
      DO 7 I = 1,NDPTS
      ENORM(ITER) = ENORM(ITER) + ABS(DN(I) - E(I + J))
    7 E(I + NEADD(1)) = DN(I)
      ENORM(ITER) = ENORM(ITER)/NDPTS
      N = ND**2
      DO 8 I = 1,N
      COM(I) = 0.
    8 C(I) = 0.
      J = ND + 1
      DO 9 I = 1,N,J
      COM(I) = 1.
    9 C(I) = 1.
      IE = NEADD(2)
      K = ND
   12 L = K
      K = K - 1
C
C     FIND THE DERIVATIVE OF THE SOLUTION
C
      CALL OPRATR(C,1,K,L,ND,0)
      IF(L .EQ. NR) GO TO 15
      IEO = 1
      CALL OPRATR(COM,IEO,K,L,ND,0)
      IF(L .NE. ND) GO TO 15
      I = NEADD(1) + 1
      J = NEADD(1) + NDPTS
      DO 16 N = I,J
   16 EDERIV(N) = E(N)
      GO TO 17
   15 CALL OPRATR(COM,IEO,K,L,ND,1)
   17 CONTINUE
      JED = - ND + NEADD(1)
      DO 10 N = 1,NELMTS
      JED = JED + ND
      DO 10 I = 1,K
      IE = IE + 1
      E(IE) = 0.
      EDERIV(IE) = 0.
      IJ = I - K
      DO 10 J = 1,ND
      IJ = IJ + K
      EDERIV(IE ) = EDERIV(IE) + COM(IJ) * EDERIV(J + JED)
   10 E(IE) = E(IE) + C(IJ) * E(J + JED)
      I = NEADD(MO - K + 1)
      J = NELMTS*K
      IF(K .GT. NR) GO TO 12
C
C     DETERMINE IF THE ITERATIONS ARE COMPLETED
C
      IF(MO .EQ. 4 .OR. LAST) GO TO 14
   13 IF(ENORM(ITER) .LT. 1.E-7) LAST = .TRUE.
      IF(ITER .LT. LAS) GO TO 9
      LAST = .TRUE.
      GO TO 9
   14 N = NDPTS * NWPC * (MO + 1)
      CALL RITE(E,NDPTS,MO + 1,0,1)
```

```
      CALL GRAPH(XD,E,N,NDPTS)                                          DENL 535
      IF(.NOT. LAST) GO TO 13                                           DENL 540
   11 CONTINUE                                                          DENL 545
      WRITE(6,102) (I,ENORM(I),I = 1,ITER)                              DENL 550
      GO TO 1                                                           DENL 555
  100 FORMAT(I2,I2,E20.10)                                              DENL 560
  101 FORMAT(1H-,40X, 'THE INITIAL VALUE OF THE SOLUTION IS',G13.4)     DENL 565
  102 FORMAT(1H-,45X,'ITERATION',5X,'NORM OF THE RESIDUAL'//            DENL 570
     *       100(1H ,45X,I5,15X,G15.5/))                                DENL 575
      END                                                               DENL 580
```

Figure 6.3.1b

to the program are the maximum permissible number of iterations in the solution process and the initial value of the function z of equation (5.6.6) on the interpolation point set. These function values are stored in the array Z, which shares common storage locations with the function subprograms FCN and FN1-FN5. Since the function z and its derivatives often need to be approximated by several different orders of polynomials in a single equation, provision has been made in the program to store the numerical values of up to ten different functions.

Using the initial values in the array Z the program calls the subroutines DIFFEQ, CONTIN and BOUND in succession, in exactly the same manner as they were called by the non-iterative program. The particular solution obtained in this way is then compared to the function z. If the L_1 norm of the difference of the point values of these two functions

$$L_1 = \sum_{I=1}^{N} | Y(I) - Z(I) | \qquad (6.3.1)$$

is less than a pre-defined value, the solution is assumed to have converged and the results printed. If, however, the L_1 norm is too large, the array Z is updated with the new particular solution values and the process repeated. The iterations stop whenever the L_1 norm of the difference function is sufficiently small or when the number of iterations exceeds the specified value. In most cases, it is found that if the procedure is convergent, about five iterations are sufficient to produce a solution limited only by round-off error.

The convergence of the above process in five iterations can be explained theoretically by noting that according to equations (5.6.3) and (5.6.6), convergence of Newton's method is quadratic. This means that the accuracy of the solution in each successive iterate is twice as good as the accuracy in the preceding iterate. Therefore, if the initial function contains one correct binary digit, the fifth iterate will contain $2^5=32$ correct binary digits. Since this is more than the wordlength of the computer used, with most initial functions, five iterations are sufficient.

The operation of the program will now be illustrated by solving several representative nonlinear differential equations. Consider, first, the differential equation (5.6.9) which was used as an example in section 5.6. The sequence of linear differential equations to be solved in this case is, from (5.6.11), in DECL,

```
DE: D3/DX3(Y)  +FN1(X)*Y
    +FN2(X)*D2/DX2(Y)=FN1(X)*FN2(X)                    (6.3.2)
```

where

```
FN1(X)=D2/DX2(Z)
FN2(X)=Z
```

The boundary condition statements for this problem are

```
BC:  Y(0.0)=0.0
BC:  D/DX(Y(0.0))=0.0                                 (6.3.3)
BC:  D2/DX2(Y(0.0))=1.0
```

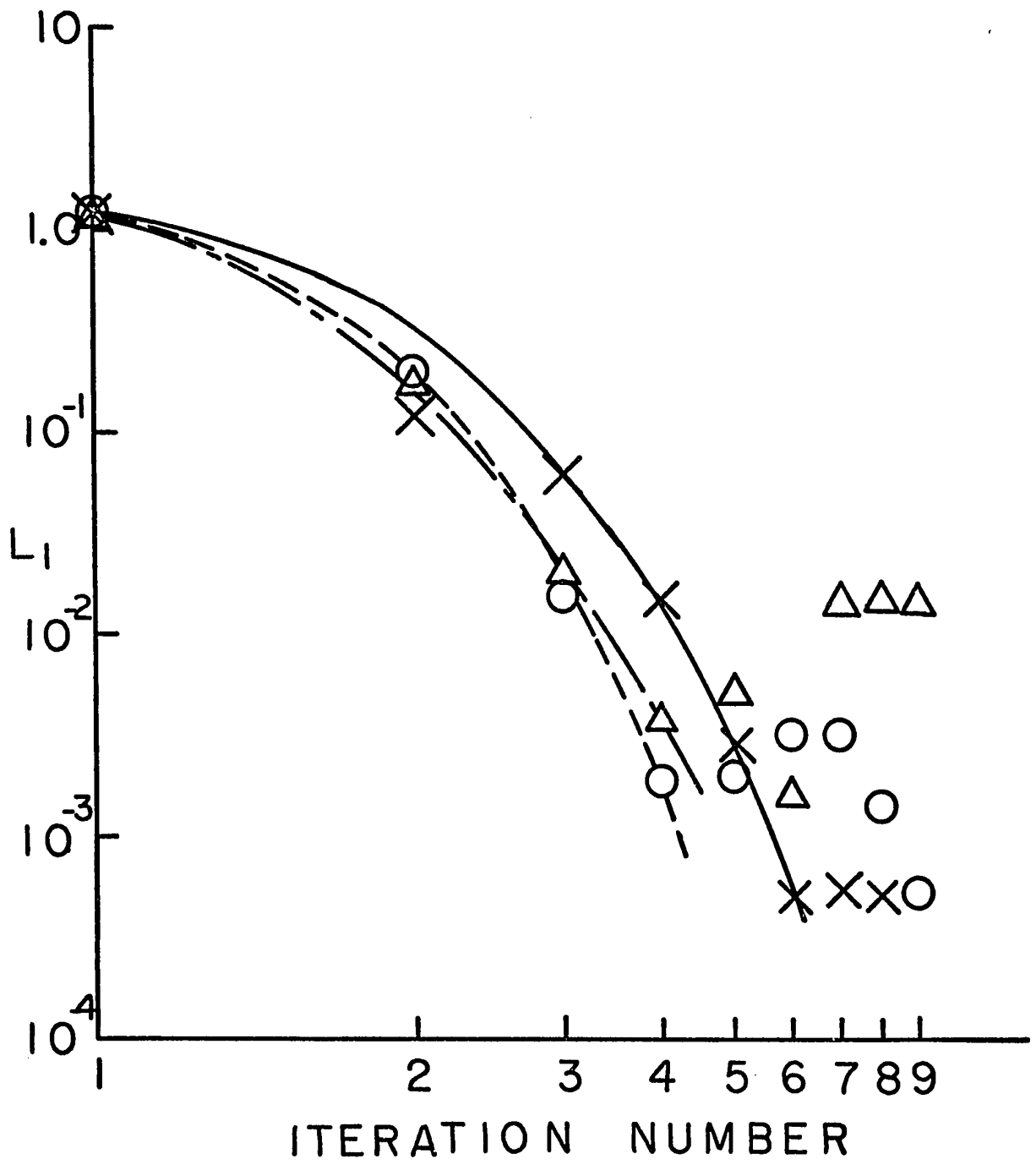


Figure 6.3.2
Convergence of Newton's method to the solution of (6.3.2).

× one element
o two elements
Δ three elements

TABLE 6.3.1

Solution of a nonlinear third order differential equation.

	Projective Method			Runge-Kutta [49]	
X	1 element (6 iterations)	2 elements (4 iterations)	3 elements (4 iterations)	h=1.0	h=0.5
0.33333	0.0553901	0.0555265	0.0557907	6	6
0.5		0.124789		0.125	0.124739
0.66667	0.220251	0.221253	0.222116	6	6
1.0	0.489594	0.492254	0.494041	0.491425	0.491908
1.33333	0.851949	0.856874	0.859315	6	6
1.5		1.06869		1.06863	1.06792
1.66667	1.28903	1.29656	1.29853	6	6
2.0	1.77900	1.78976	1.78989	1.78738	1.78792
2.33333	2.30135	2.31563	2.31214	6	6
2.5		2.58561		2.57913	2.58121
2.66667	2.84065	2.85832	2.84920	6	6
3.0	3.38751	3.40822	3.39273	3.37957	3.40061

In this case, the initial value of z was taken to be the zero function and the problem was solved in the interval [0,3] using one, two and three ninth-order elements.

Figure 6.3.2 contains a graph of the logarithm of the L_1 norm of the error in the approximate solution versus the logarithm of the number of iterations for this problem. Notice that the L_1 norm decreases until the sixth iterate in the one element solution and until the fourth iterate in the two and three element solutions. In this region, the convergence is seen to be approximately quadratic. Beyond these iterations, the change in the solution appears to be random and, as a result, can be attributed to round-off error. Table 6.3.1 contains the solutions obtained from the computer program in these cases. The solution values converge to an accuracy of about four significant figures as the number of elements used increases.

As a check on the results, included in Table 6.3.1 are two Runge-Kutta calculations of fifth-order accuracy for the differential equation (5.6.9)[49]. It may be observed that there is good agreement between the projective solutions and those obtained by Runge-Kutta integration.

For the next example, consider the initial value problem

$$\frac{dy}{dx} - (1 + y^2) = 0 \qquad (6.3.4)$$

$$y(0) = 0$$

which has the solution y=tan x. In applying Newton's method to this problem, the sequence of linear initial value problems

DE: D/DX(Y) - 2.0*FN1(X)*Y = 1.0 - POW2(FN1(X)) (6.3.5)

BC: Y(0.) = 0.

where

FN1(X) = Z

needs to be solved. Using zero as the initial solution function, the sequence of solutions obtained from this procedure is known to converge to the solution of (6.3.4) in the interval $0 \leq x < \frac{\pi}{2}$ [60].

Equations (6.3.5) were solved with the iterative computer program of this section in the interval [0.0, 1.5] by using two ninth-order approximating functions. In this case, since a much more rapid variation was expected in the solution near x=1.5 than at x=0.0, the first element was chosen to span the interval from z=0.0 to x=1.0 and the second from x=1.0 to x=1.5. Figure 6.3.3 presents the graph of the solutions obtained from the program for the first five equations in the sequence (6.3.5). It can be seen that each successive solution rises above the preceding one and provides a better approximation to the exact solution y=tan x. For this problem, the L_1 norm of the difference function decreased until the eighth iteration and the solution obtained on this iterate is given in Table 6.3.2 along with the exact values. It is observed from this table that while the approximate solution in the first element from x=0.0 to x=1.0 is accurate to at least four significant figures, the solution in the second element from x=1.0 to x=1.5 becomes progressively less accurate until, at x=1.5, there is a 6% error. This behavior is the result of the singularity of the solution at $x=\pi/2$.

Figure 6.3.3

Convergence of Newton's method to the solution of (6.3.5).

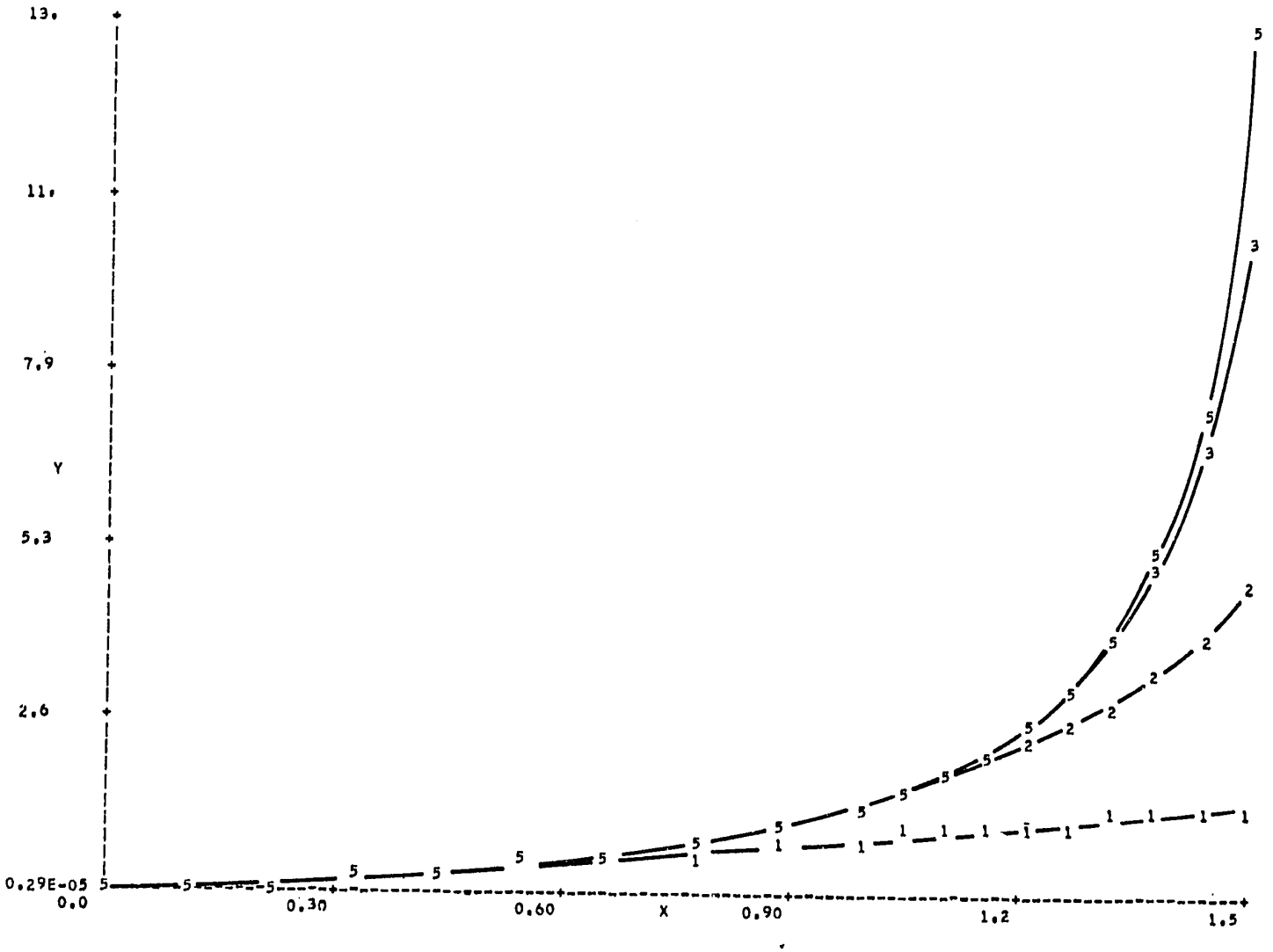


Table 6.3.2

Solution of Equation (6.3.5).

Calculated

1	-0.596046E-07	2	0.111551E 00	3	0.225933E 00	4	0.346233E 00	5	0.476200E 00
6	0.620751E 00	7	0.786813E 00	8	0.984837E 00	9	0.123174E 01	10	0.155737E 01
11	0.155737E 01	12	0.173892E 01	13	0.199219E 01	14	0.230028E 01	15	0.270325E 01
16	0.324714E 01	17	0.402759E 01	18	0.526841E 01	19	0.750886E 01	20	0.131813E 02

Exact

1	0.0	2	0.111571E 00	3	0.225954E 00	4	0.346254E 00	5	0.476221E 00
6	0.620775E 00	7	0.786843E 00	8	0.984874E 00	9	0.123180E 01	10	0.155741E 01
11	0.155741E 01	12	0.176597E 01	13	0.201997E 01	14	0.233825E 01	15	0.275168E 01
16	0.331450E 01	17	0.413171E 01	18	0.543649E 01	19	0.787214E 01	20	0.141010E 02

A third example of a nonlinear initial value problem that has been solved with the iterative differential equation solving computer program is provided by

$$\frac{dy}{dx} = y - \frac{2x}{y} \qquad (6.3.6)$$

$$y(0) = 1$$

In this case, the quasi-linearized form of the differential equation is

$$\text{DE: } D/DX(Y) - Y - 2.0*X*POW2(FN1(X))*Y = -4.0*X*FN1(X) \qquad (6.3.7)$$

where

$$FN1(X) = 1.0/Z$$

and Z is taken to be equal to 1.0 initially. Table 6.3.3 contains the solution of this problem using two equal ninth-order elements after five iterations. The execution time required for this problem was 5.6 seconds. Comparison with the exact solution $y=(2x+1)^{\frac{1}{2}}$ [49], also given in Table 6.3.3, shows that the projective solution is accurate to at least four significant figures everywhere. The error remaining is

Table 6.3.3
Solution of Equation (6.3.6).

Calculated

1	0.100000E 01	2	0.110552E 01	3	0.120182E 01	4	0.129096E 01	5	0.137433E 01
6	0.145292E 01	7	0.152748E 01	8	0.159855E 01	9	0.166660E 01	10	0.173197E 01
11	0.173197E 01	12	0.179494E 01	13	0.185578E 01	14	0.191468E 01	15	0.197182E 01
16	0.202734E 01	17	0.208136E 01	18	0.213401E 01	19	0.218536E 01	20	0.223552E 01

Exact

1	0.100000E 01	2	0.110554E 01	3	0.120185E 01	4	0.129099E 01	5	0.137437E 01
6	0.145297E 01	7	0.152752E 01	8	0.159861E 01	9	0.166667E 01	10	0.173205E 01
11	0.173205E 01	12	0.179505E 01	13	0.185592E 01	14	0.191485E 01	15	0.197203E 01
16	0.202759E 01	17	0.208167E 01	18	0.213437E 01	19	0.218581E 01	20	0.223607E 01

again attributed to round-off error accumulation on the IBM 360/75.

Consider now the problem of solving nonlinear boundary value problems by Newton's method, instead of initial value problems. In form, the procedures used are identical in both cases. As an illustration of the technique, the nonlinear boundary value problem

$$\frac{d^2y}{dx^2} = \frac{3}{2} y^2 \qquad (6.3.8)$$

$$y(0) = 4$$

$$y(1) = 1$$

will be solved and the results compared with the exact solution [49]

$$y=\frac{4}{(1+x)^2} \qquad (6.3.9)$$

The quasi-linearized form of (6.3.8) is

$$\text{DE: D2/DX2(Y) - 3.0* FN1(X)*Y=-1.5*POW2(FN1(X))} \qquad (6.3.10)$$

where

FN1(X) = Z

The interval [0.0, 1.0] was divided into two equal ninth-order elements. After five iterations, the solution had

Table 6.3.4

Solution of Equation (6.3.8).

Calculated

1	0.400000E 01	2	0.359003E 01	3	0.323999E 01	4	0.293874E 01	5	0.267762E 01
6	0.244981E 01	7	0.224987E 01	8	0.207343E 01	9	0.191697E 01	10	0.177756E 01
11	0.177756E 01	12	0.165288E 01	13	0.154087E 01	14	0.143987E 01	15	0.134849E 01
16	0.126554E 01	17	0.119001E 01	18	0.112106E 01	19	0.105793E 01	20	0.100000E 01

Exact

1	0.400000E 01	2	0.359003E 01	3	0.324000E 01	4	0.293878E 01	5	0.267769E 01
6	0.244991E 01	7	0.225000E 01	8	0.207360E 01	9	0.1917716E 01	10	0.177778E 01
11	0.177778E 01	12	0.165306E 01	13	0.154102E 01	14	0.144000E 01	15	0.134860E 01
16	0.126563E 01	17	0.119008E 01	18	0.112111E 01	19	0.105796E 01	20	0.100000E 01

converged satisfactorily and this answer is presented in Table 6.3.4. The C.P.U. time used in this solution was 4.8 seconds. Once again, the solution values are accurate to four or more significant figures.

Another example of a nonlinear boundary value problem solved with the iterative computer program is

$$\frac{d^3y}{dx^3} + 6y + y^2 = -\frac{3}{2}\cos x \qquad (6.3.11)$$

$$y(0) = y(2\pi)$$

$$\frac{dy(0)}{dx} = \frac{dy(2\pi)}{dx}$$

Although the exact solution of (6.3.11) is not known, Collatz has expanded the solution in terms of a Fourier series and obtained the following approximate solution [49]

$$y \simeq -\frac{3}{320000}(805.5 + 32240\cos x + 2418\cos 2x - 240\cos 3x - 2.7\cos 4x) \qquad (6.3.12)$$

In solving (6.3.11) using Newton's method, the following sequence of linear boundary value problems need to be solved

Table 6.3.5

Solution of Equation (6.3.12).

Calculated

1	-0.330338E 00	2	-0.307883E 00	3	-0.244180E 00	4	-0.149698E 00	5	-0.400719E-01
6	0.671440E-01	7	0.157045E 00	8	0.221138E 00	9	0.257938E 00	10	0.269718E 00
11	0.269717E 00	12	0.257938E 00	13	0.221139E 00	14	0.157048E 00	15	0.671473E-01
16	-0.400692E-01	17	-0.149697E 00	18	-0.244181E 00	19	-0.307883E 00	20	-0.330338E 00

Exact

1	-0.330195E 00	2	-0.307809E 00	3	-0.244174E 00	4	-0.149605E 00	5	-0.398413E-01
6	0.673786E-01	7	0.157144E 00	8	0.221149E 00	9	0.257984E 00	10	0.269805E 00
11	0.269805E 00	12	0.257985E 00	13	0.221150E 00	14	0.157145E 00	15	0.673801E-01
16	-0.398399E-01	17	-0.149604E 00	18	-0.244173E 00	19	-0.307809E 00	20	-0.330195E 00

```
DE:  D2/DX2(Y) + 6.0*Y + FN1(X)*(Y)
                                                    (6.3.13)
     = POW2(FN1(X)) - 1.5*COS(X)
BC:  Y(0.) = Y(6.283184)
BC:  D/DX(Y(0.)) = D/DX(Y(6.283184))
```

where FN1(X) =Z.

Using two equal ninth-order elements and the zero function as Z on the initial iterate, the sequence of solutions of these boundary value problems converge very rapidly with the L_1 norm of the difference between iterates decreasing to 3.6×10^{-7} on the fourth iterate. Table 6.3.5 contains the values of the solution from the fourth iterate as well as the values of the function (6.3.12) on the set of interpolation points. It can be seen that the two approximate solutions agree very closely.

6.4 PARTIAL DIFFERENTIAL EQUATIONS

The technique and required subprograms for the solution of linear two-dimensional partial differential equations have already been described. A main computer program which connects these subprograms together in the proper order is presented in Figure 6.4.1. When this main program is used with the overlay procedure in Figure 6.4.2, the entire two-dimensional program package requires 194K bytes of core memory. In this way, the arrays DN, C and COM which contain the large two-dimensional coefficient matrices are enlarged without increasing the memory requirements of the program.

Figure 6.4.1

```
C                                                                       DE2D   5
C     THIS PROGRAM WAS WRITTEN BY ZOLTAN CSENDES OF MCGILL UNIVERSITY,  DE2D  10
C     MONTREAL, QUEBEC, TO SOLVE ARBITRARY LINEAR TWO-DIMENSIONAL       DE2D  15
C     PARTIAL DIFFERENTIAL EQUATIONS.                                   DE2D  20
C                                                                       DE2D  25
      DIMENSION YD(500),WTY(100),IELCX(100),IELCY(100),YBC(300)         DE2D  30
      DIMENSION INST(500),CONST(500),XD(500),WTX(100),DN(8000),C(8000), DE2D  35
     *     COM(19000),YR(1),YI(1),ROOTR(100),ROOTI(100),XBC(300)        DE2D  40
C                                                                       DE2D  45
C                              DECL 60-95                               DE2D  50
C                                                                       DE2D  55
    1 CALL INST2D(INST,CONST,NUMDE,XBC,YBC,XD,YD,WTX,WTY,IELCX,IELCY,   DE2D  60
     *     NCCOND,NELMTS,NRANGX,NRANGY,NDOM,LAMDA,RITEP,RITEG,PLOTP,    DE2D  65
     *     PLOTG,NEIGWB,NEIGPL)                                         DE2D  70
      NRX = NRANGX + 1                                                  DE2D  75
      NRY = NRANGY + 1                                                  DE2D  80
      ND = NDOM + 1                                                     DE2D  85
      CALL DIFF2D(DN,MDN,WTX,WTY,NELMTS,INST,CONST,XD,YD,NRX,NRY,ND,    DE2D  90
     *     NULLD,NULLV,LAMDA)                                           DE2D  95
C                                                                       DE2D 100
C                              DECL 150-175                             DE2D 105
C                                                                       DE2D 110
      CALL CONT2D(DN,NRX,NRY,ND,MDN,NULLV,C,NDPTS,NONHOM,NULLV,         DE2D 115
     *     NCCOND,IELCX,IELCY,NELMTS,WTX,WTY,LAMDA,RITEG)               DE2D 120
C                                                                       DE2D 125
C                              DECL 190-210                             DE2D 130
C                                                                       DE2D 135
      DO 5 I = 1,NULLV                                                  DE2D 140
    5 CALL PICT(XD,YD,C,NDPTS,I)                                        DE2D 145
    2 CALL BOUN2D(C,NDPTS,NONHOM,NULLD,NULLV,DN,NWBC,NRX,NRY,ND,        DE2D 150
     *      INST,CONST,NUMDE,XD,YD,NELMTS,WTX,WTY,LAMDA,RITEP,YI)       DE2D 155
C                                                                       DE2D 160
C                              DECL 250-320                             DE2D 165
C                                                                       DE2D 170
      IF(LAMDA) NWBC = NEIGPL                                           DE2D 175
      DO 4 I = 1,NWBC                                                   DE2D 180
    4 CALL PICT(XD,YD,DN,NDPT,I)                                        DE2D 185
      GO TO 1                                                           DE2D 190
      END                                                               DE2D 195
```

Figure 6.4.2

```
1   ENTRY MAIN
2   INSERT MAIN,NULL,RITE
3   OVERLAY ALPHA
4   INSERT INST2D,CHARTF,FNUM,PICT,SCALE
5   OVERLAY ALPHA
6   INSERT DIFF2D,CONT2D,BOUN2D,DPG2D1,DPG2D2,DPRATR,FCN2D,KRON,DPFU1,ALPHA
7   OVERLAY ALPHA
8   INSERT EIGEN,RILMAT,EIGQR,DQRT
```

In order to illustrate the operation of the program, consider the elliptic partial differential equation

$$\nabla^2 z = -2 \sin x \sin y \qquad (6.4.1)$$

with the boundary conditions

$$z(0,y)=z(\pi,y)=z(x,0)=z(x,\pi)=0 \qquad (6.4.2)$$

Using one seventh-order element, a working DECL computer program to solve this problem is

```
* 7            POISSON'S EQUATION IN A RECTANGLE
DE:    D2/DX2(Z)+D2/DY2(Z)=-2*SIN(X)*SIN(Y)
BC:    Z(0.0, 0.0)=0.0
BC:    Z(0.0, 0.449)=0.0
       .
       .
       .
BC:    Z(3.14, 3.14)=0.0
REC:   X=0.0, 31.41592, Y=0.0, 3.141592
```

In this program, the boundary conditions (6.4.2) have been translated into twenty-eight separate boundary condition statements specifying the solution value at each of the twenty-eight nodal points on the perimeter of the element. Due to the interpolatory nature of the approximating functions b_{ij} the twenty-eight boundary conditions in the above statements are linearly independent and completely equivalent to (6.4.2) in the approximating space. Although it would be a desirable extension of the DECL language, present program capabilities do not include generation of the above boundary condition statements from compact statements such as

```
BC: Z(0.0, Y)=0.0
```

The above DECL program was run on the IBM 360/75 and required 18.1 seconds of C.P.U. time for execution. The nontrivial solution values are given in Table 6.4.1 for one half-quadrant of the solution region along with the corresponding values of the exact solution.

$$z=sin\ x\ sin\ y \qquad (6.4.3)$$

Taking advantage of the symmetry in the above problem, equation (6.4.1) was also solved using one seventh-order element in the rectangle $x=[0,\pi/2],y=[0,\pi/2]$ with Neumann boundary conditions along the lines $x=\pi/2$ and $y=\pi/2$ and this solution is also presented in Table 6.4.1. Notice that the solution obtained without using the symmetry properties of the problem is accurate to about three significant figures and that the solution obtained by using symmetry is accurate to about four significant figures.

Table 6.4.1

Solution point values for Poisson's equation in a rectangle.

x	y	Numerical without using symmetry	Numerical using symmetry	Exact
$\pi/7$	$\pi/7$	0.188149	0.188239	0.188256
$\pi/7$	$2\pi/7$	0.339105	0.339204	0.339224
$\pi/7$	$3\pi/7$	0.422871	0.422955	0.423005
$2\pi/7$	$2\pi/7$	0.611182	0.611237	0.611261
$2\pi/7$	$3\pi/7$	0.762153	0.762205	0.762230
$3\pi/7$	$3\pi/7$	0.950418	0.950470	0.950485

Table 6.4.2

Point values of the solution of parabolic partial differential equations.

Point coordinates		(6.4.4)		(6.4.6)	
x	y	Numerical	Exact	Numerical	Exact
$\pi/7$	$\pi/7$	0.27704	0.27699	0.67997	0.67965
$\pi/7$	$4\pi/7$	0.072070	0.072066	2.6128	2.6123
$\pi/7$	π	0.018739	0.018750	10.002	10.040
$2\pi/7$	$\pi/7$	0.49917	0.49912	1.2251	1.2247
$2\pi/7$	$4\pi/7$	0.12989	0.12986	4.7081	4.7071
$2\pi/7$	π	0.033774	0.033786	18.049	18.092
$3\pi/7$	$\pi/7$	0.62235	0.62239	1.5273	1.5272
$3\pi/7$	$4\pi/7$	0.16203	0.16193	5.8704	5.8697
$3\pi/7$	π	0.042142	0.042131	22.546	22.561

Figure 6.4.3 contains the graph produced of the solution in the first case by the subroutine PICT. Smooth lines have been drawn by hand on this figure to round out the surface contours between point values.

The next example presented is the two-dimensional parabolic partial differential equation

DE:D2/DX2(Z)= D/DY(Z) (6.4.4)

with

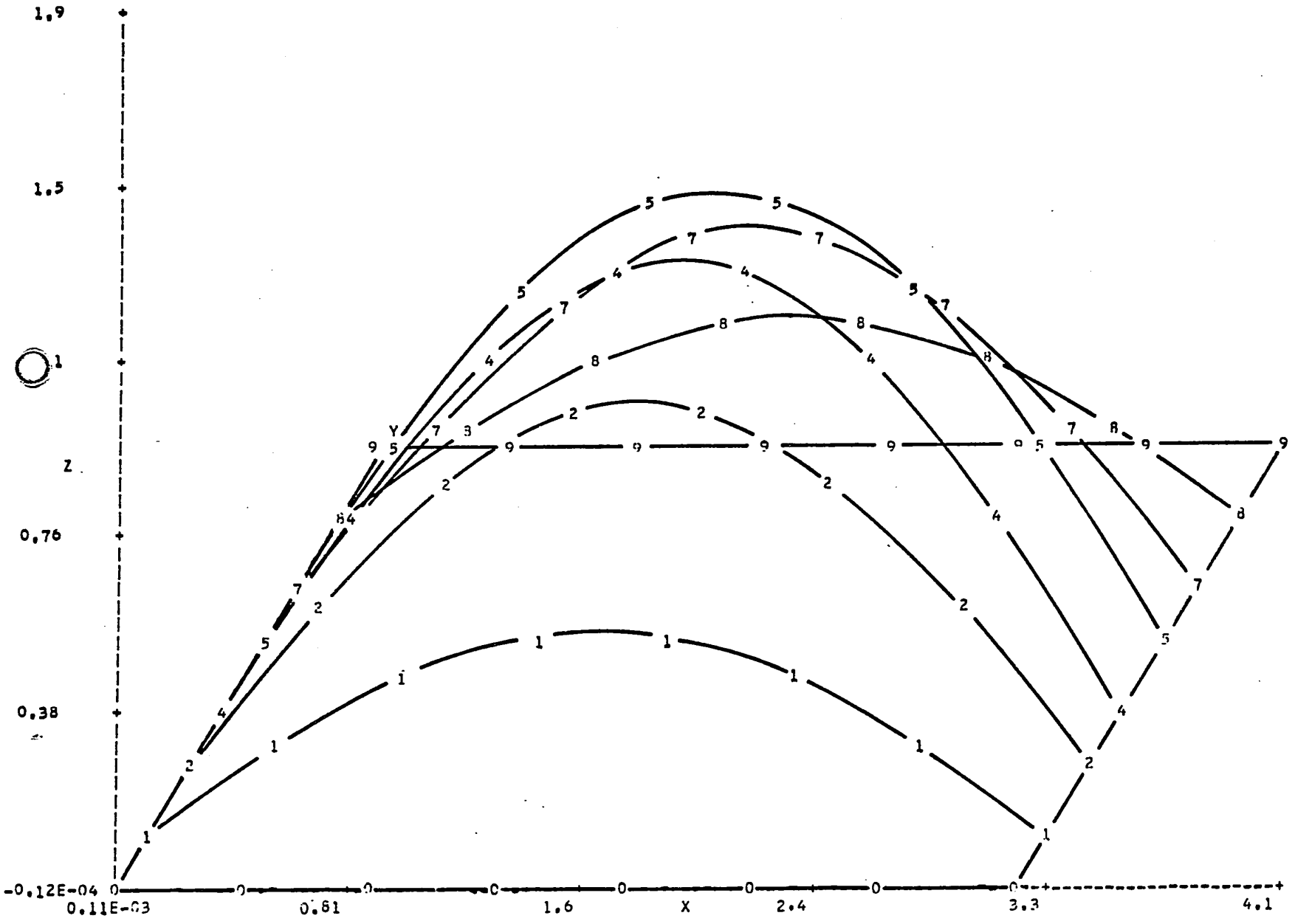
$$z(0,y)=z(\pi,y) = 0$$

$$z(x,0)=\sin x$$

A single seventh-order element was used in the rectangle x=[0,π],y=[0,π]. The DECL computer program for this problem contained twenty-two boundary condition statements and

Figure 6.4.3

Solution of the elliptic partial differential equation (6.4.1).



required 15.3 seconds of C.P.U. time to produce a solution. Selected point values of this solution are given in Table 6.4.2 along with values of the exact solution

$$z=\sin x\, e^{-y} \tag{6.4.5}$$

This table also contains the values of the solution of the complementary problem

$$\text{DE: D2/DX2(Z)= - D/DY(Z)} \tag{6.4.6}$$

which has the exact solution

$$z=\sin x\, e^{y} \tag{6.4.7}$$

Both numerical solutions agree with the exact solutions to three or four significant figures. The graph produced by the program with the solution of equation (6.4.6) is shown in Figure 6.4.4.

For the third example, consider the eigenvalue boundary value problem

$$\text{DE: D2/DX2(Z) + D2/DY2(Z) = K*Z} \tag{6.4.8}$$

$$z(0,y)= z(x,0)=\frac{d}{dx}z(1,y)=\frac{d}{dy}z(x,1)=0 \tag{6.4.9}$$

The eigenvalues of this problem are known to be

$$K=(\tfrac{\pi}{2})^2[(2n-1)^2+ (2m-1)^2] \quad n,m=1,2,\ldots \tag{6.4.10}$$

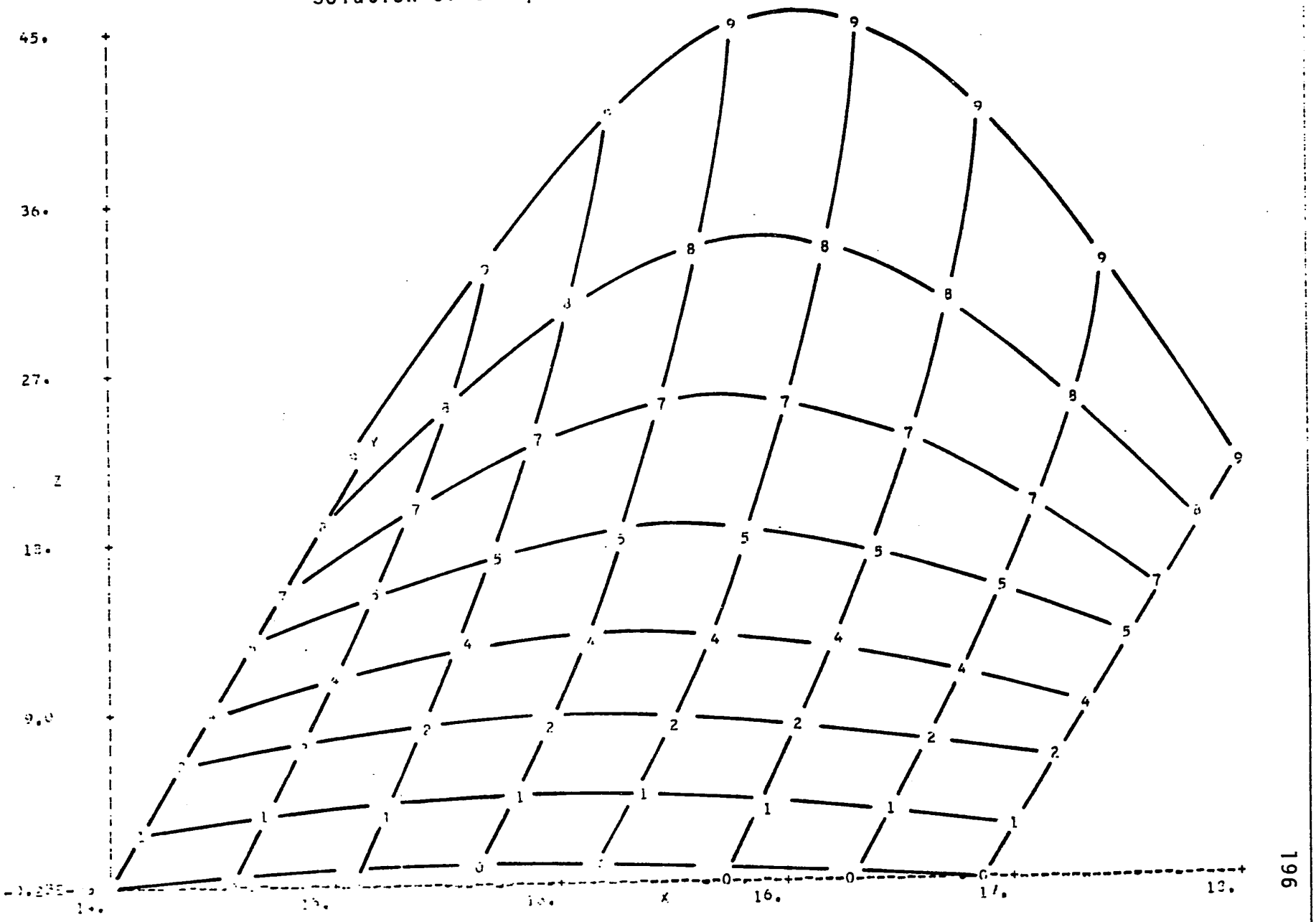
and the eigenfunctions are

$$z=\sin(\tfrac{n\pi}{2} x)\sin(\tfrac{m\pi}{2} y) \tag{6.4.11}$$

Equation (6.4.8) was solved using one sixth-order element and required 25.5 seconds of C.P.U. time for solution. It is of interest to note that in this problem the solution was comprised of 49 independent point values but that with the procedure of section 5.5, the order of the matrix eigenvalue problem solved was only 25. Table 6.4.3 contains the values of the first eight eigenvalues produced by the program as

Figure 6.4.4

Solution of the parabolic equation (6.4.6).



seen that the first three eigenvalues are accurate to four significant figures but that the higher eigenvalues became progressively less accurate with a 1.5% error in the eighth eigenvalue. Computer plots of the first and fourth modes of this problem are shown in Figures 6.4.5 and 6.4.6.

Table 6.4.3

Eigenvalues of the Helmholtz equation in a rectangle.

Eigenvalue number	Numerical	Exact
1	4.93452	4.93480
2	24.6784	24.6740
3	24.6793	24.6740
4	44.426	44.413
5	65.49	64.15
6	65.49	64.15
7	85.24	83.89
8	85.24	83.89

All of the examples shown up to this point have been two-dimensional partial differential equations in rectangular regions for which the locations of the boundary conditions coincided with the edges of the elements. Consider now the problem of solving Helmholtz's equation (6.4.8) in a circular region where

$$z(x^2 + y^2 = 1)=0 \qquad (6.4.12)$$

Figure 6.4.5

One quadrant of the dominant mode of vibration in a rectangle.

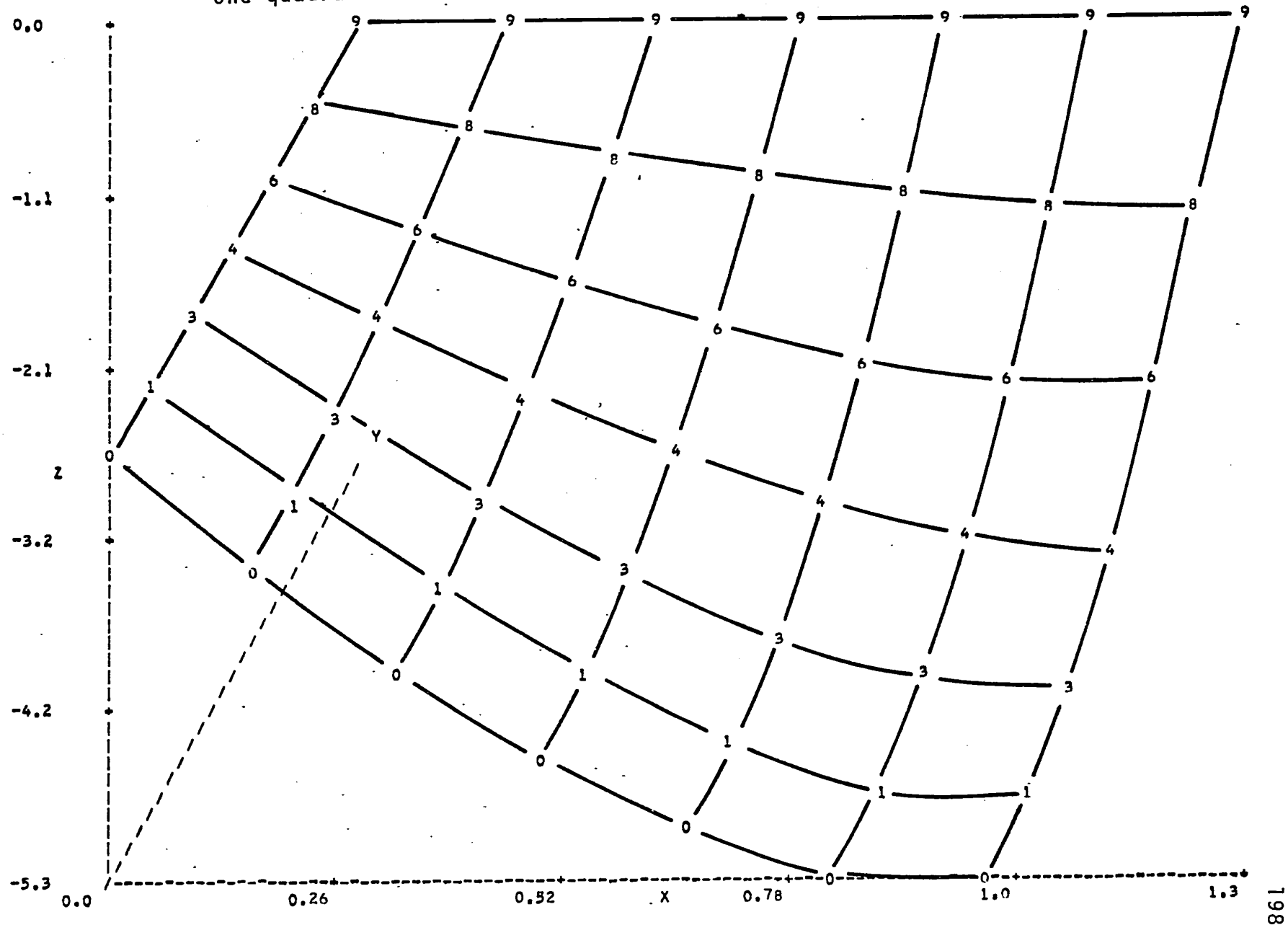
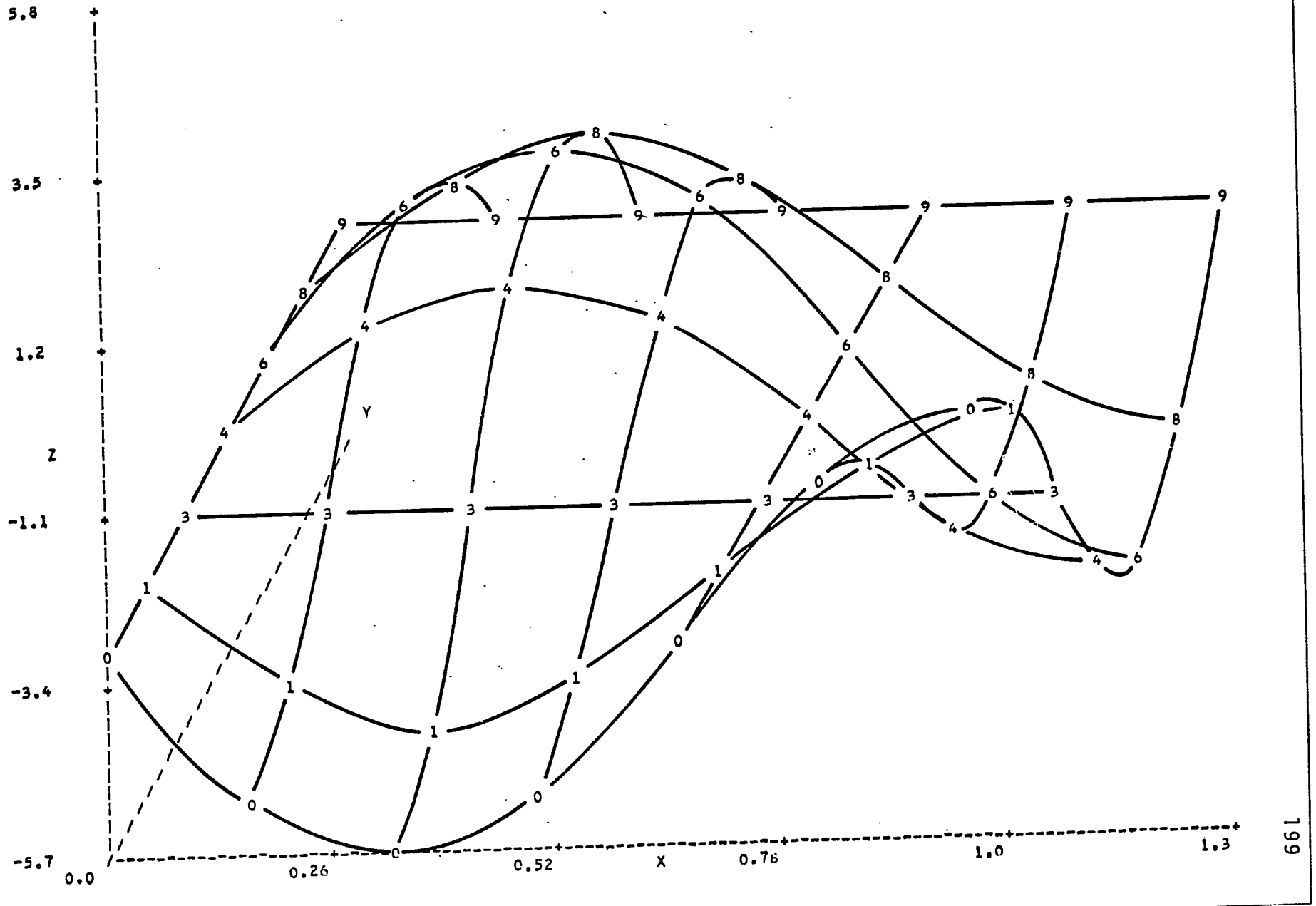


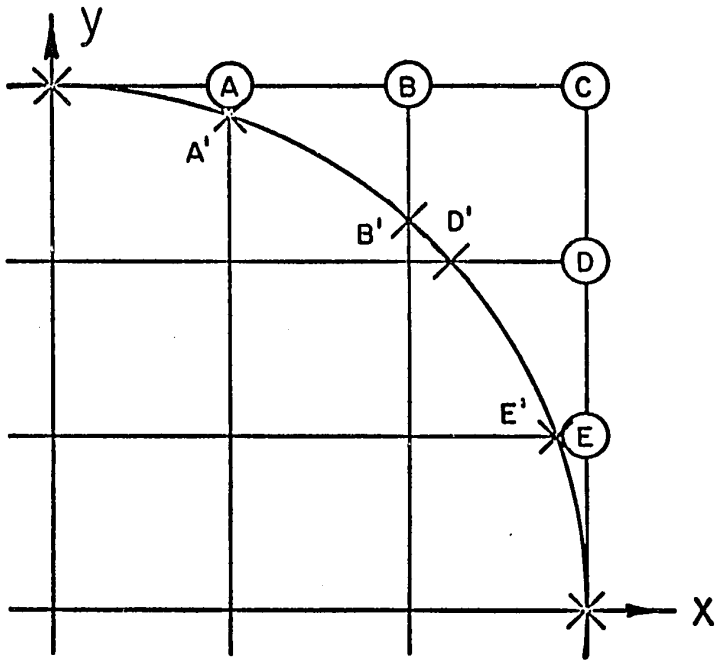
Figure 6.4.6.

One quadrant of the fourth mode of vibration in a rectangle.



In this problem, the projective solution region will not match the region defined by the boundary conditions for any rectangle or combination of rectangles. Consequently, a method for specifying boundary conditions at locations other than the edges of the elements must be employed.

As pointed out in section 5.3, equations (5.3.3) and (5.3.4) may be used to determine the numerical conditions on a projective solution for it to satisfy any boundary condition in any geometric location. However, as was the case with rectangular regions, when working with an irregular region it is wise to take advantage of the orthogonality properties of the interpolation polynomials b_{ij} in choosing boundary condition point locations. In the case of the circular boundary condition (6.4.12), for example, a sixth-order element may be super-imposed on the circle as shown by the



One quadrant of a sixth order element super-imposed on a circular boundary.

Figure 6.4.7

quadrant in Figure 6.4.7. The straight lines drawn on this figure indicate the locations of the zeros of the interpolation polynomials b_{ij} and the crosses denote the points of intersection of these lines with the circular boundary.
In analogy with the rectangular boundary value problem (6.4.9), it is natural to take these points of intersection to be the locations on the circle where the approximate solution is made equal to zero. This procedure has the effect of shifting the four boundary condition locations A,B,D and E used in the rectangular case to the locations A′, B′, D′ and E′ on the circular boundary. However, a simple count of the number of arbitrary constants in the general solution reveals that one additional boundary condition needs to be specified besides the six imposed at the crosses in Figure 6.4.7.

The addition of this extra boundary condition is, however, unusual in the sense that it is not used to restrict the behavior of the solution on the circular boundary but rather as a means of redefining the basis functions in the domain space of the differential operator. In the above problem, the complete set of basis functions b_{ij} which interpolate over the entire square region in Figure 6.4.7 does not constitute a satisfactory basis set with the circular boundary conditions (6.4.12) because it contains an extraneous function. The extraneous function in this case is the interpolation polynomial which has a unit value at point C in Figure 6.4.7 and governs the solution behavior in the corners of the element. By defining a new set of basis functions in which this interpolation polynomial does not

exist, a basis is formed in which the number of arbitrary constants in the general solution equals the number of intersection points of the nodal lines with the circle.

It is important to note in this analysis that the new set of basis functions is derived from the established set by eliminating some of its members. In the present case, this is most easily accomplished by setting the coefficient of the unwanted corner interpolation polynomial equal to zero with the boundary condition statement

BC: Z(1.0, 1.0)=0.0 (6.4.13)

In this way, the solution of problem (6.4.12) can be obtained directly from the DECL computer program without performing any program modifications.

There are, of course, some disadvantages to using the elimination procedure described above to solve problem (6.4.12) with rectangular elements instead of defining special circular elements which fit the geometry exactly. Solutions obtained with the above procedure are optimized over the entire element and not in the smaller circular region. Furthermore, the DECL computer program will perform unnecessary arithmetic operations with function coefficients which will be eliminated. However, in most engineering applications, the resulting inefficiencies are not serious enough to justify solving the problem with special techniques.

A DECL computer program was written to solve problem (6.4.12) using the boundary condition statements described above. The program required 27.1 seconds of C.P.U. time for execution and the first eight eigenvalues produced are given in Table 6.4.4. The problem was also solved in one quarter

of the region using a sixth-order element and Neumann boundary conditions along the axes. These results are also presented in Table 6.4.4 along with the exact eigenvalues. A comparison of the answers shows that the lower order eigenvalues are accurate to two or three significant figures. Bearing in mind that the solutions are not stationary in the eigenvalues, this accuracy is quite acceptable. It is somewhat lower than was the case with the rectangular problem (6.4.9), however, since in that case a larger and more suitable basis set was used. Plots of the first and sixth modes for the circular Helmholtz boundary value problem are shown in Figures 6.4.8 and 6.4.9.

Table 6.4.4

Eigenvalues of the Helmholtz equation in a circle.

Eigenvalue number	Numerical full circle	Numerical quarter circle	Exact
1	2.389	2.393	2.405
2	3.877	-	3.832
3	3.877	-	3.832
4	5.110	5.137	5.135
5	5.260	-	5.135
6	5.638	5.524	5.520
7	5.712	-	5.520
8	7.705	7.609	7.016

In order to demonstrate the feasibility of solving two-dimensional partial differential equations in complicated regions with the DECL program, Helmholtz's equation was also

Figure 6.4.8

Dominant mode in a circular clamped vibrating membrane.

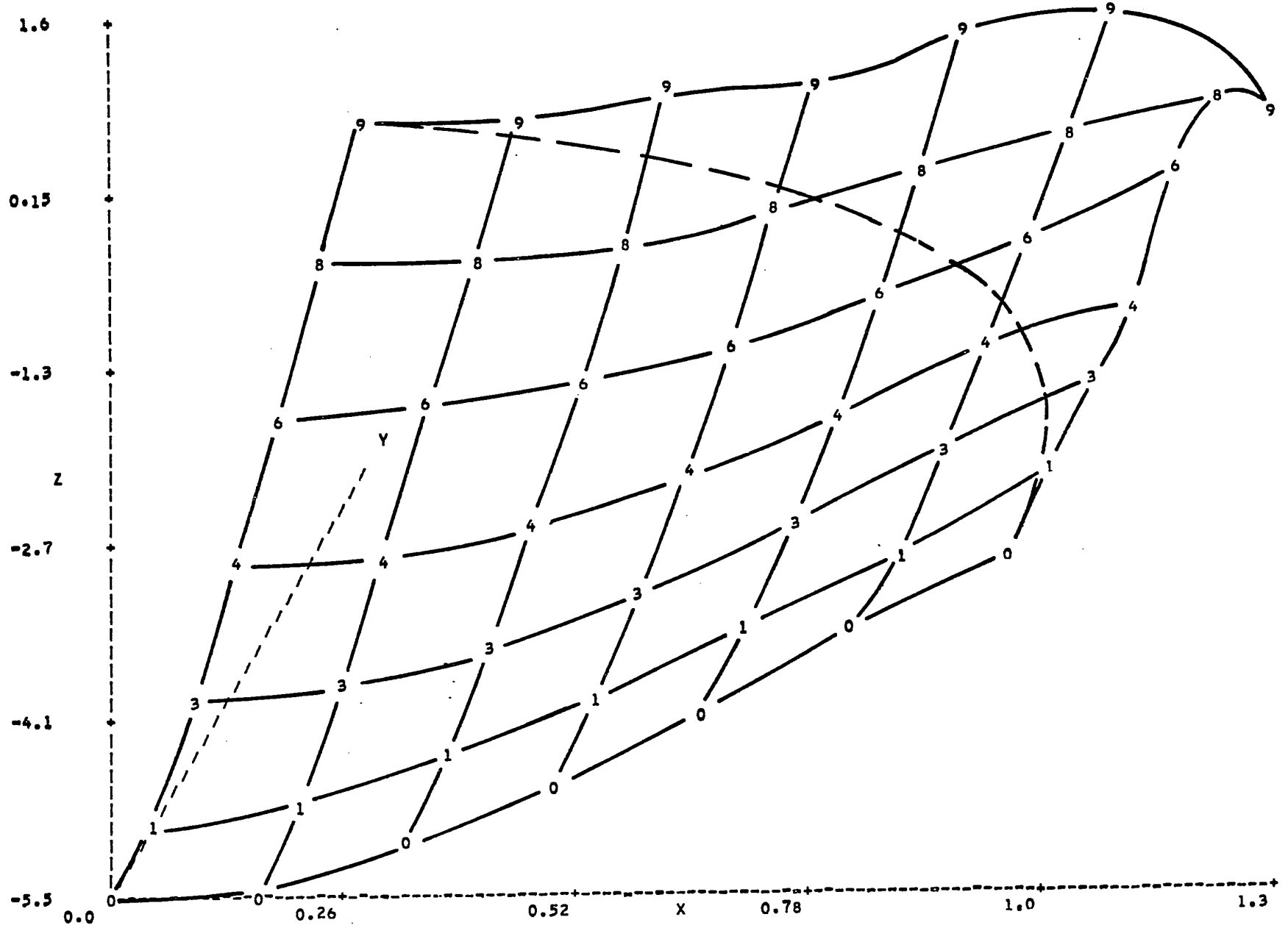
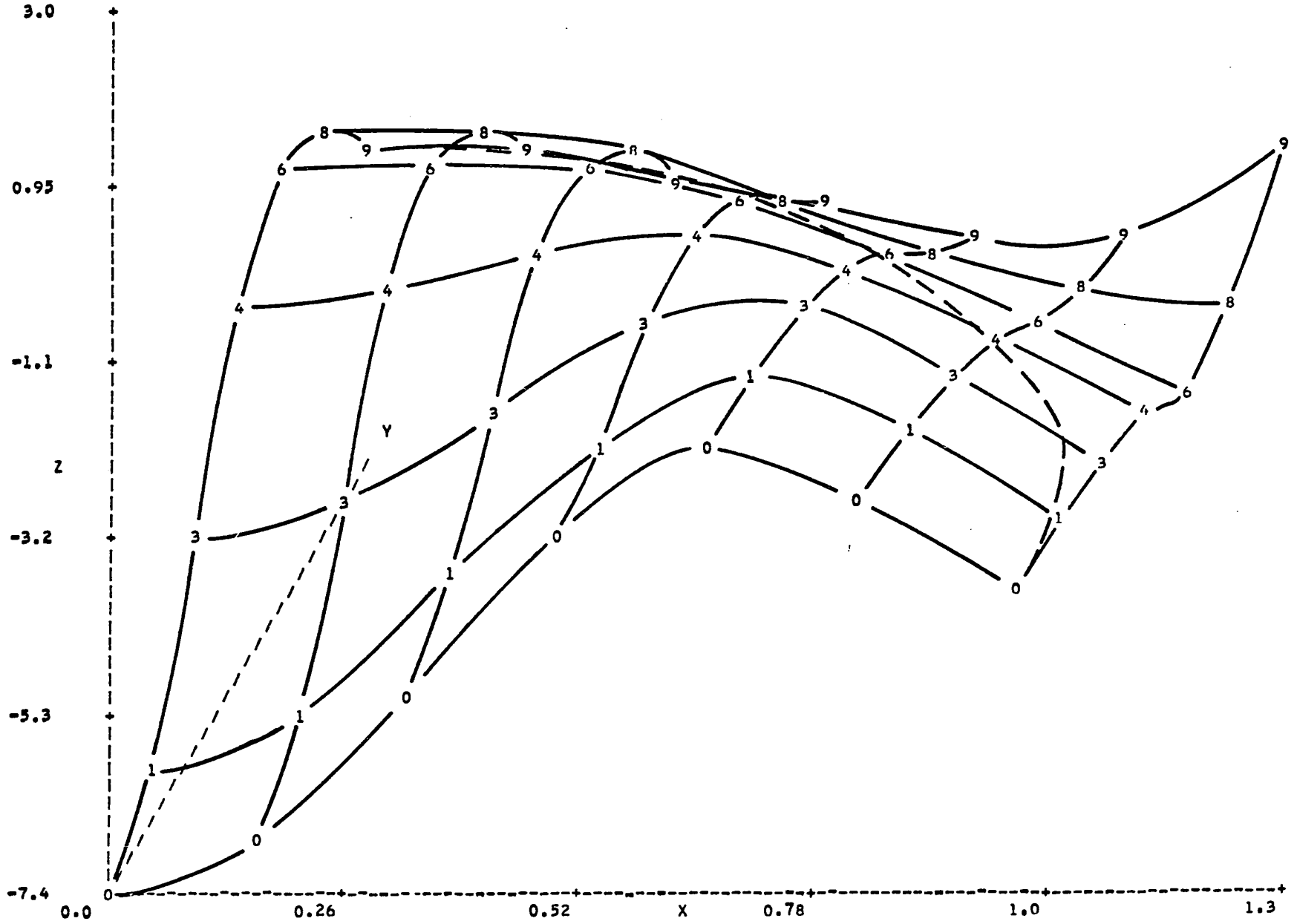
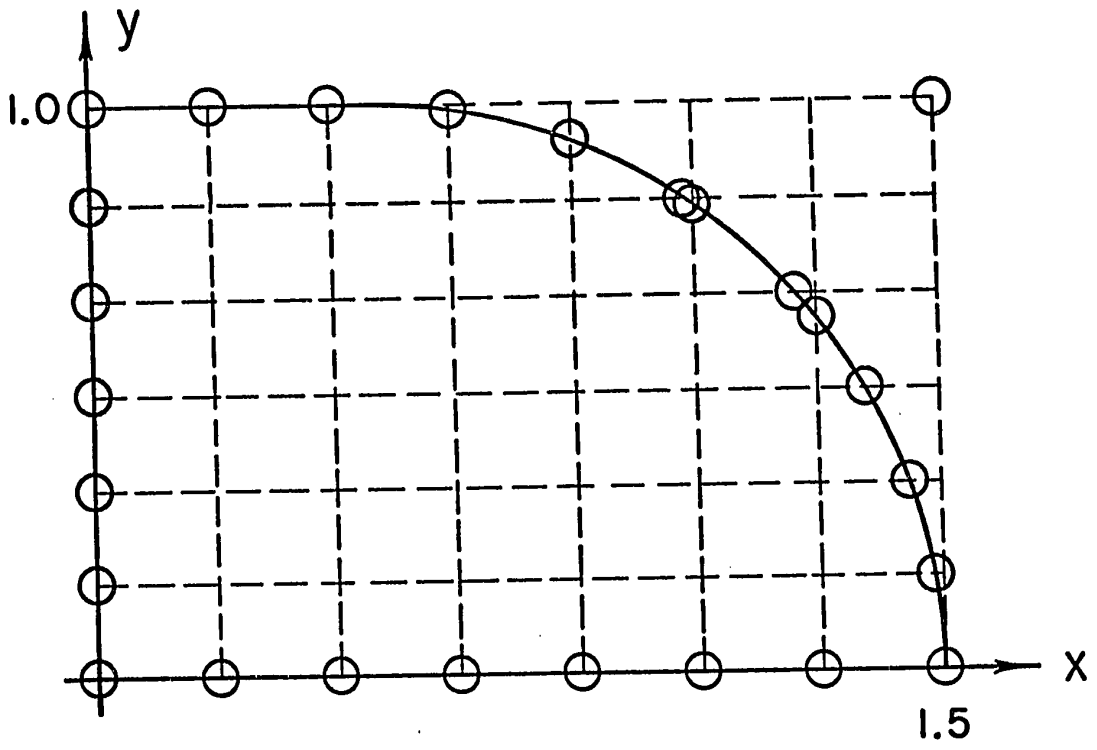


Figure 6.4.9

Sixth mode of a circular clamped vibrating membrane.



sixth-order element was used in the approximation with the boundary condition point locations chosen as indicated by the circles in Figure 6.4.10. Here Neumann conditions were used for points along the coordinate axes and Dirichlet conditions were imposed elsewhere. The first eigenvalue for this problem was equal to 2.027 with the corresponding eigenvector behaving similarly to the dominant eigenvector in the circular case. These results agree with an analysis of this problem using ordinary finite elements to approximately three significant figures [71].



Cigar-shaped region showing boundary condition point locations.

Figure 6.4.10

Finally, consider the following two-dimensional Bessel-type equation

```
DE:  POW2(X) * (D2/DX2(Z) + D2/DY2(Z))
DE:  +X*D/DX(Z) -Z = K*POW2(X)*Z                    (6.4.14)
```

which governs the behavior of the azimuthal electric and magnetic fields in a rotationally symmetric resonant cavity. The boundary conditions of interest are

$$z(boundary)=0.0 \tag{6.4.15}$$

for electric fields and for magnetic fields

$$x\,\frac{dz}{dn}(boundary)=-z(boundary)\frac{dz}{dn} \tag{6.4.16}$$

where n is the direction normal to the boundary. This boundary value problem has been solved with the DECL program and some of these results will be presented here. Consider first a rectangular toroidal cavity with x=[2,5] and y=[$-\frac{1}{2}$,$\frac{1}{2}$].

Table 6.4.5

Resonant frequencies of a rectangular toroidal cavity with x=[2,5], y =[$\frac{1}{2}$,$\frac{1}{2}$].

Eigenvalue number	Electric field		Magnetic field	
	Numerical	Exact*	Numerical	Exact*
1	3.32117	3.32150	1.03668	1.03660
2	3.78948	3.78526	2.08585	2.08868
3	4.475	4.451	3.14177	3.13768
4	6.397	6.375	3.30820	3.30820
5	6.655	6.629	3.77096	3.77250
6	7.068	7.030	4.588	4.440

* These numbers were computed by A. Konrad.

The resonant frequencies of the electric and magnetic fields obtained for this cavity from the DECL program using one sixth-order element are presented in Table 6.4.5 along with the exact values. There is excellent agreement with all low order eigenvalues. In addition, Figures 6.4.11 and 6.4.12 contain two of the field plots produced by the DECL program.

Now consider a right cylindrical cavity in which the solution of equation (6.4.14) is required in the region $x=[0,3]$, $y=[-\frac{1}{2},\frac{1}{2}]$. Unfortunately, equation (6.4.14) has a singularity along the line $x=0.0$ due to the $1/x^2$ coefficient in Bessel's equation and function values along this line cannot appear in the solution vector. Consequently, in order to obtain meaningful results, the solution region must be made to overlap the line $x=0.0$. It is not necessary to avoid the line $x=0.0$ when specifying boundary conditions however, since the boundary conditions (6.4.15) and (6.4.16) are valid in any location.

The calculation of the electric and magnetic fields in the right cylindrical cavity has been performed by the DECL program using a sixth-order element in the region $x=[-0.05, 3.0]$, $y=[-0.5, 0.5]$ and the eigenvalues produced are given in Table 6.4.6. In this calculation, the boundary conditions were applied along the lines $x=0.0$, $x=3.0$, $y=-0.5$ and $y=0.5$. The eigenvalues have approximately the same accuracy as in the previous example. Figures 6.4.13 and 6.4.14 contain examples of the field plots produced by the DECL program for this problem.

Figure 6.4.11

The fifth resonant mode of the electric field in a rectangular toroidal cavity.

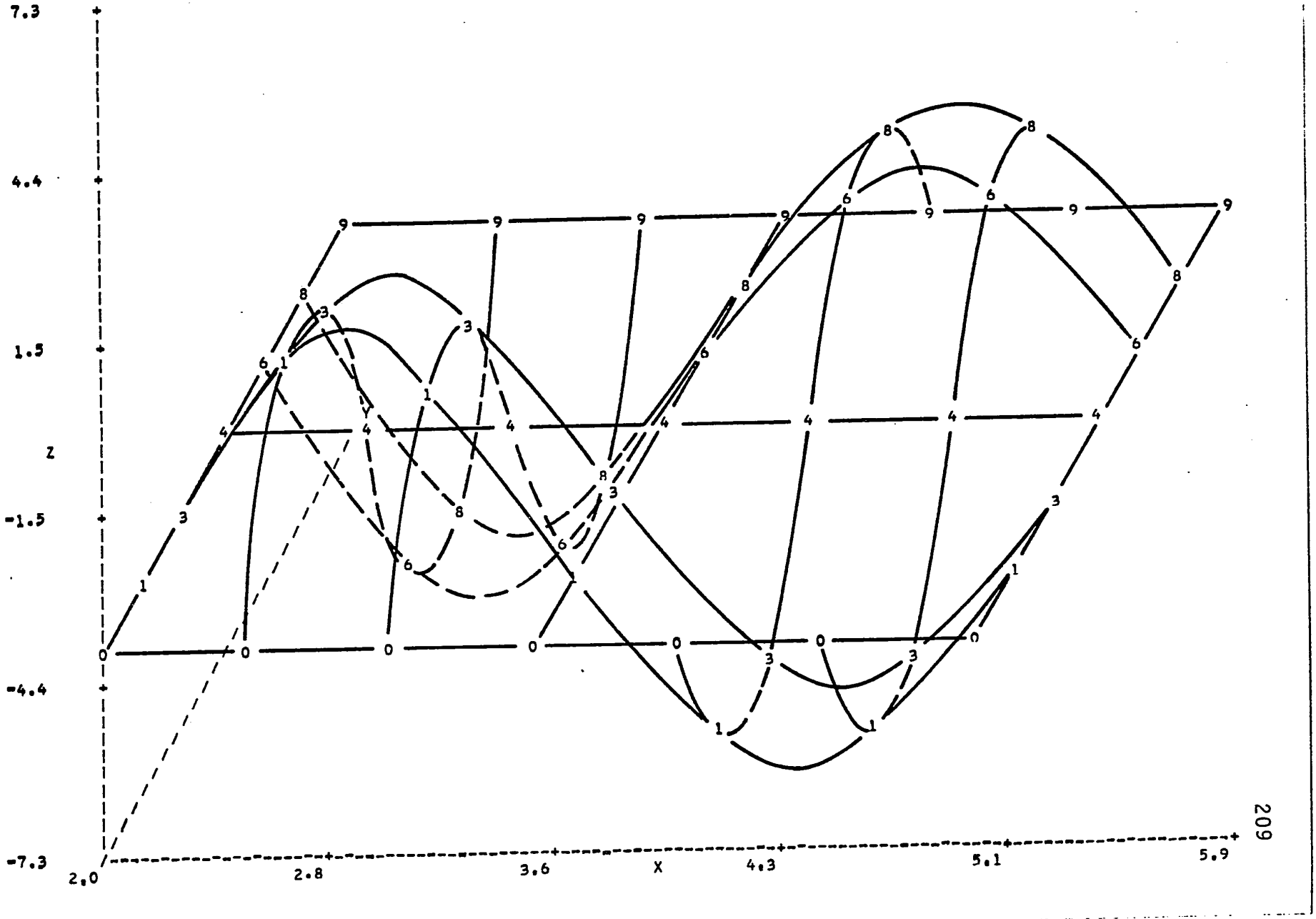


Figure 3.4.12

The fifth resonant mode of the magnetic field in a rectangular toroidal cavity.

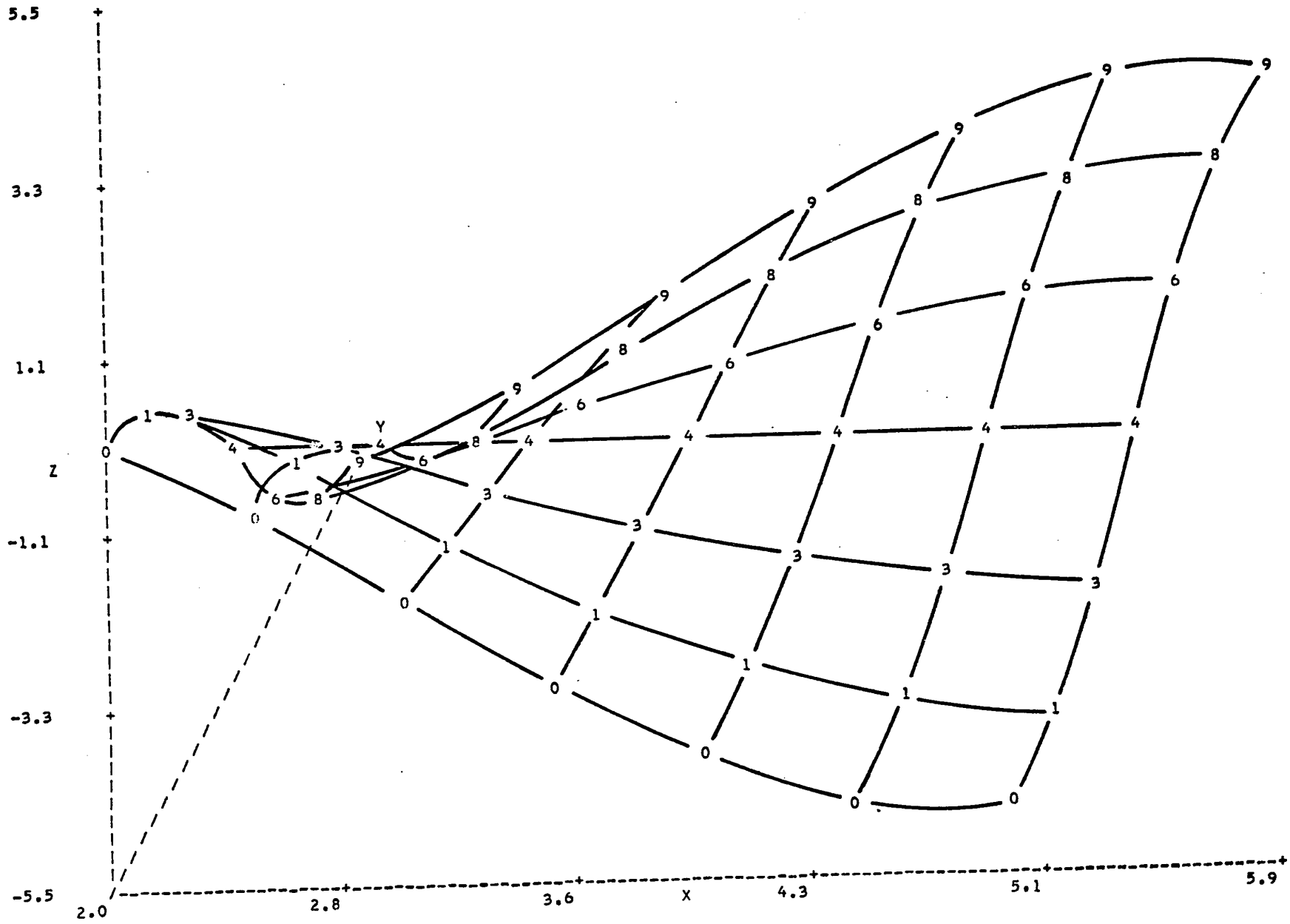


Figure 6.4.13

The second resonant mode of the electric field in a right cylinderical cavity.

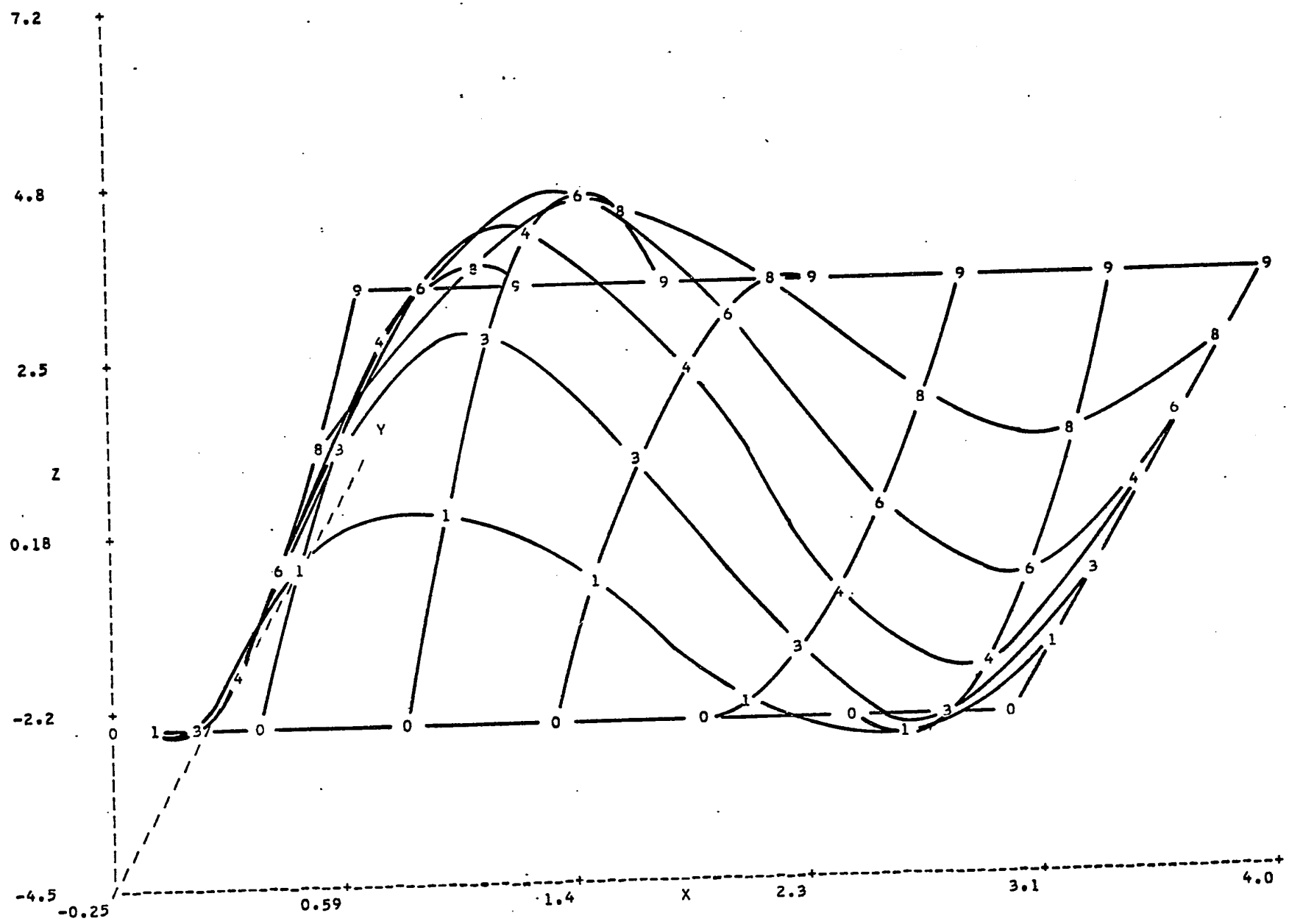


Figure 6.4.14

The third resonant mode of the magnetic field in a right cylindrical cavity.

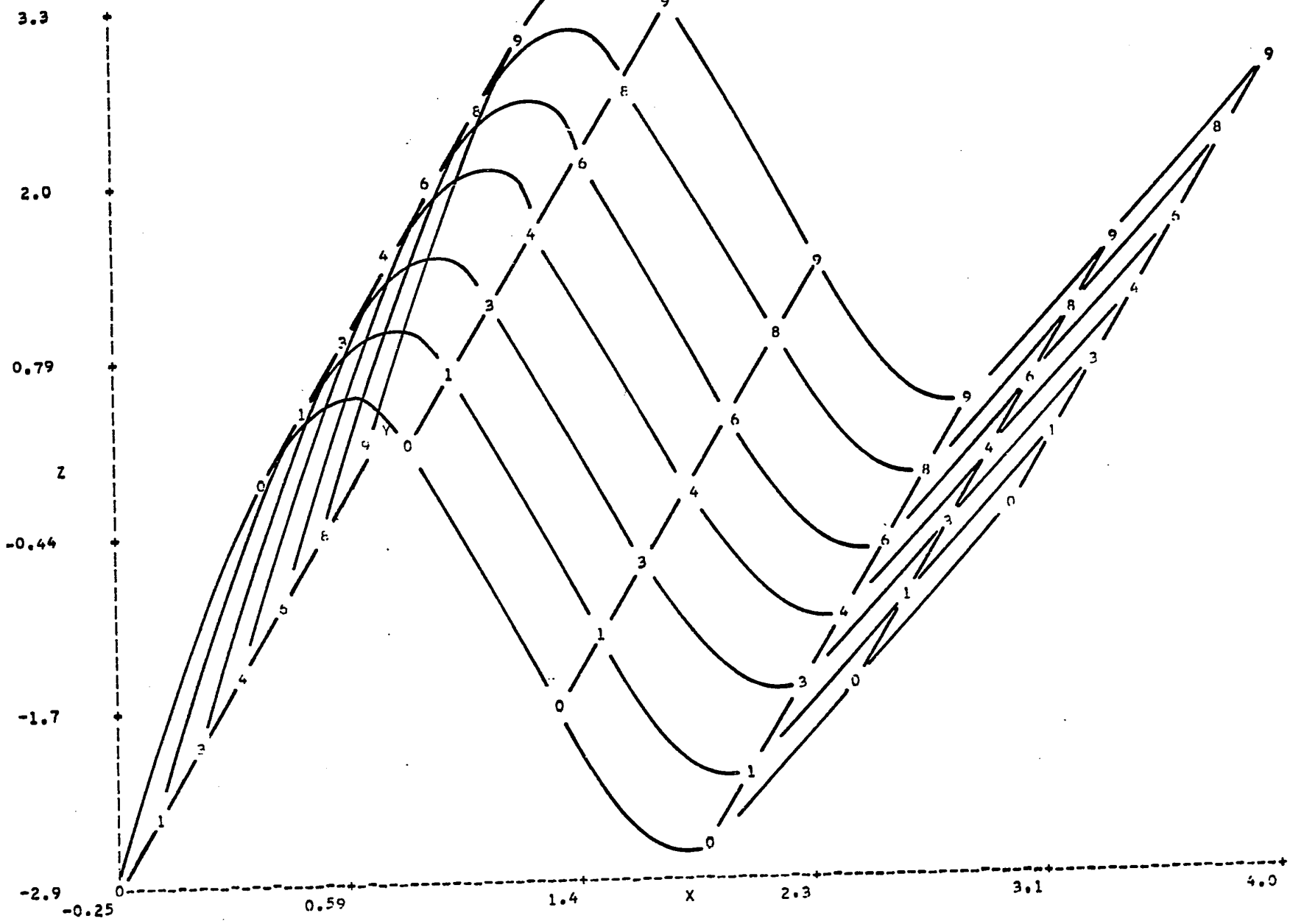


Table 6.4.6

Resonant frequencies of a right cylindrical cavity with $x=[0,3]$, $y=[-\frac{1}{2},\frac{1}{2}]$.

Eigenvalue number	Electric field		Magnetic field	
	Numerical	Exact*	Numerical	Exact*
1	3.39112	3.39130	0.80129	0.80160
2	3.92052	3.91649	1.8201	1.8400
3	4.847	4.623	2.9437	2.8845
4	6.433	6.412	3.242	3.152
5	6.728	6.704	3.631	3.641
6	6.975	7.140	4.305	3.931

* These numbers were computed by A. Konrad.

CHAPTER 7 CONCLUSIONS

In this thesis, two different but nevertheless interrelated themes have been developed. The first and perhaps primary theme has been the introduction of a discretization procedure which uses different orders of polynomials to approximate the domain and the range spaces of differential equations. The notion of assembling complicated operators from simple ones, using projection operators between different spaces, and employing generalized matrix inversion to solve the resulting rectangular matrix equations are all logical consequences of this numerical approach. The second theme has been the development of differential equation solving programs with versatile and easy to use input and output characteristics. This development essentially constitutes the creation of a new special purpose computer language which makes it possible to solve many differential equations automatically.

The advantages of using the discretization procedure developed in this thesis are numerous. First of all, it is computationally efficient. The method relies primarily on matrix addition and multiplication for its execution, a trait well-suited to the operating characteristics of digital computers. Second, the method is highly accurate. In addition to providing numerical procedures of up to ninth-order sectional polynomial accuracy, the solutions are optimized in a least squares sense in their respective approximation spaces. Third,

the method produces general solutions of differential equations as well as particular ones. With these general solutions, it is possible to solve a differential equation separately in many regions using relatively small matrices, the connecting inter-element conditions being added subsequently. Further, unlike most ordinary projective methods where the boundary conditions are located at the edges of the region considered, general solutions allow boundary conditions to be specified anywhere.

As for the development of the automatic differential equation solving programs in this thesis, these too have been highly successful. As shown by the examples in Chapter 6, the DECL computer programs provide a practical alternative for solving many ordinary and two-dimensional partial differential equations to established methods. The program is sufficiently accurate for most engineering applications and their computational times are also acceptable, considering the general nature of the programs. Most importantly, however, the programs have a simple input and output structure that allows many differential equations to be solved with a minimum of analysis and data preparation.

It must be mentioned, however, that although the theory and the computer programs in this thesis have many attractive features, they also have several serious shortcomings. On the theoretical side, the most prominent of these is the neglect of this thesis to provide a rigorous error bound for the solutions

generated. As a result, in those cases where the accuracy of the solution must be defined precisely, the user is required to examine the convergence of the solution using previously established procedures. In addition to requiring a further knowledge of numerical analysis, these procedures often magnify the computation required to treat a differential equation by an order of magnitude.

Some of the other mathematically unavoidable deficiencies in the method are the large sizes of matrices which result with high-order, two-dimensional elements, the failure of Newton's method to converge in all cases, and the necessity of avoiding singular points in the calculations. Fortunately, however, while these problems are troublesome, they do not preclude alternative formulations of the differential equation to avoid the occurence of these problems.

The DECL computer programs also have some limitations which are not based on theoretical reasons. Among these program limitations are the cumbersome technique used to specify two-dimensional boundary conditions, the lack of double precision versions of the programs, and the necessity of evaluating the quasi-linear form of a non-linear differential equation by hand, instead of having the DECL compiler do it automatically. In terms of future development, these areas indicate some of the features of the DECL computer programs and of the DECL computer language which should be improved. Hopefully, in a continuing evolutionary development, newer, more

sophisticated procedures will eventually replace the inadequate ones existing presently in DECL. Thus, while the DECL computer programs given in this thesis are insufficient for some purposes, they do constitute an important first step in the development of a general computer language for the automatic solution of differential equations.

Even in their present form, however, the overall performance of the DECL programs in treating most ordinary and two-dimensional partial differential equations accurately and efficiently indicates that these programs can be a valuable tool in engineering applications. By taking advantage of the ability of these programs to solve a large variety of problems without modification, it is often possible for practicing engineers to bypass the difficulty and expense of writing a special computer program to solve each separate type of differential equation. In this way, the valuable skills of many engineers who out of necessity are presently part-time numerical analysts and programmers can be deployed more efficiently.

References

1. S.G. Mikhlin, Variational Methods in Mathematical Physics, Oxford: Pergamon Press, 1957.

2. S.G. Mikhlin and K.L. Smolitskiy, Approximate Methods for the Solution of Differential and Intergal Equations, New York: American Elsevier, 1967.

3. L.V. Kantorovich and V.I. Krylov, Approximate Methods of Higher Analysis, New York:Interscience, 1964.

4. J.L. Turner, R.W. Clough, H.C. Martin and L.J. Topp, "Stiffness and deflection analysis of complex structures", J. Aero Science, Vol. 23, 1956, pp. 805-825.

5. R.W. Clough, "The finite-element in plane stress analysis, Proc. 2nd A.S.C.E. Conf. on Electronic Computation, Pittsburgh, Pa., 1960.

6. A. Adini and R.W. Clough, "Analysis of plate bending by the finite element method", Nat. Sci. Found. Rept. G7337, U. of California, Berkeley, 1961.

7. R.H. Gallagher, "A correlation study of methods of matrix structural analysis", AGARDograph 69, Pergamon Press, 1962.

8. J.H. Argyris, "Matrix methods of structural analysis", Proc. 14th Meeting of AGARD, AGARDograph 72, 1962.

9. O.C. Zienkiewicz and Y.K. Cheung,"The finite element method for the analysis of elastic isotropic and orthotropic slabs", Proc. Inst. Civil Eng. Vol. 28, 1964, pp. 471-488.

10. O.C. Zienkiewicz and Y.I. Cheung,"Finite elements in the solution of field problems", The Engineer, Vol. 200, 1965, pp. 507-510.

11. J.T. Oden,"Numerical formulation of nonlinear elasticity problems", J. Struct. Div. ASCE, Vol. 93, 1967, pp. 235-255.

12. J.T. Oden, "Finite element analysis of nonlinear problems in the dynamic theory of coupled termoelasticity", Nuclear Eng. and Design, Vol. 10, 1969, pp. 465-475.

13. P. Silvester, "Finite element solution of homogeneous waveguide problems", 1968 URSI Symp. Electromagnetic Waves, published in Alta Frequenza, Vol. 38, 1969, pp. 313-317.

14. S. Ahmed and P. Daly, "Waveguide solutions by the finite element method", Radio and Electronic Eng., Vol. 38, 1969, pp. 217-223.

15. P. Silvester and M.V.K. Chari, "Finite-element solution of saturable magnetic field problems," IEEE Trans., Vol. PAS-89, 1970, pp. 1642-1651.

16. Z. Csendes and P. Silvester, "Numerical solution of dielectric loaded waveguides", IEEE Trans. on MTT, Vol. MTT-18, 1970, pp. 1124-1131 and Vol. MTT-19, 1971, pp. 504-509.

17. P. Daly, "Hybrid-mode analysis of microstrip by finite element methods", IEEE Trans. on MTT, Vol. MTT-19, 1971, pp. 19-25.

18. M. Zlamal, "On the finite element method", Numer. Math., Vol. 12, 1968, pp. 394-409.

19. G. Birkhoff, M.H. Schultz and R.S. Varga, "Piecewise Hermite interpolation in one and two variables with applications to partial differential equations", Numer.Math., Vol. 11, 1968, pp. 232-256.

20. I. Babuska, "Error bounds for the finite element method", Numer. Math., Vol. 16, 1971, pp. 322-333.

21. M.H. Schultz, "Error bounds for polynomial spline interpolation", Math. Comp. Vol. 24. 1970, pp. 507-515.

22. O.C. Zienkiewicz, The Finite Element Method in Engineering Science, London: McGraw Hill, 1971.

23. J.T. Oden, Finite Elements of Nonlinear Continua, New York: McGraw Hill, 1972.

24. M. Zlamal, "Some recent advances in the mathematics of finite elements," in Whiteman (ed.), The Mathematics of Finite Elements and Applications, London:Academic Press, to appear.

25. J.R. Whiteman, "A bibliography for finite element methods", in Whiteman (ed.), The Mathematics of Finite Elements and Applications, London: Academic Press, to appear.

26. R.S. Varga, Functional Analysis and Approximation Theory in in Numerical Analysis, Philadelphia: Society for Industrial and Applied Mathematics, 1971.

27. I. Babuska, M. Prager and E. Vitasek, Numerical Processes in Differential Equations, New York: Interscience, 1966.

28. P. Silvester and M.S. Hsieh, "Finite-element solution of 2-dimensional exterior-field problems,"IEE Proceedings, Vol. 118, 1971, pp. 1743-1947.

29. H.J. Brauchli and J.T. Oden, "Conjugate approximation functions in finite element analysis", Quarterly of Applied Math., Vol. 29, 1971, pp. 65-90.

30. T.G. Hazel and A. Wexler, "Variational formulation of the Dirichlet boundary condition", IEEE Trans. on MTT, Vol. MTT-20, 1972, pp.385-390.

31. K. Reichert and W. Vogt, "Das Finite-Element-Verfahren, angewandt auf die Analyse magnetischer Kreise," Bull. Assoc. Suisse Electrot. Vol. 62, 1971, pp. 1074-1082.

32. W.L. Miranker," Galerkin approximations and the optimization of difference schemes for boundary value problems", SIAM J. Numerical Analysis, Vol. 8, 1971, pp. 486-496.

33. O.C. Zienkiewicz with Y.K. Cheung, The Finite Element Method in Structural and Continuum Mechanics, New York: McGraw Hill, 1967.

34. P. Silvester, "High-order polynomial finite elements for potential problems", Int. J. Engineering Science, Vol. 7, 1969, pp. 849-861.

35. B.M. Irons, "Economical computer techniques for numerically integrating finite elements", Int. J. Numerical Methods Eng. Vol. 1,1969,pp.201-204.

36. P.V. Marcal, "On general purpose programs for finite element analysis, with special reference to geometric and material nonlinearities," In Hubbard (ed.), Numerical Solution of Partial Differential Equations-II, New York:Academic Press,1971.

37. J.M. Boisserie, "Generation of two-and three-dimensional finite elements, "Int. J. Num. Meth. Eng.,Vol. 3, 1971, pp. 327-347.

38. P. Silvester and C.R.S. Haslam, "Megnetotelluric modeling by the finite element method," to appear in Geophysical Prospecting, Dec. 1972.

39. E. Isaacson and H.B. Keller, Analysis of Numerical Methods, New York: John Wiley, 1966.

40. J. Xenakis, PL/1 FORMAC Symbolic Mathematics Interpreter, Hawthorne, New York: IBM, 1969.

41. J. Locker, "The method of least squares for boundary value problems," Trans. Am. Math. Society, Vol. 154, 1971, pp. 57-68.

42. P.R. Halmos, Finite Dimensional Vector Spaces, Princeton, New Jersey: D. Van Nostrand, 1958.

43. I. Stakgold, Boundary Value Problems of Mathematical Physics, New York: MacMillan, 1968.

44. P.J. Davis, Interpolation and Approximation, New York: Blaisdell, 1963.

45. P. Silvester, "Symmetric quadrature formulae for simplexes," Math. Comp., Vol. 24, 1970, pp. 95-100.

46. R. Bellman, Introduction to Matrix Analysis, New York: McGraw-Hill, 1960.

47. H. Neudecker, "A note on Kronecker matrix products and matrix equation systems," SIAM J. Appl. Math. Vol. 17, 1969, pp. 603-606.

48. P. Silvester, "Numerical formation of finite-difference operators," IEEE Trans., Vol. MTT-18, 1970, pp. 740-743.

49. L. Collatz, The Numerical Treatment of Differential Equations, New York: Springer-Verlag, 1966.

50. M. Berkovic, "Free-free structures in finite element analysis," X-th Yugoslavian Congress on Mechanics, 1970.

51. MFGR, System S/360 Scientific Subroutine Package, White Plains, New York: IBM, 1968.

52. Z. Csendes, A. Gopinath and P. Silvester, "Generalised matrix inverse techniques for local approximations of operator equations," in Whiteman (ed.), The Mathematics of Finite Elements and Applications, London: Academic Press, to appear.

53. A.W. Gilles, "On the classification of matrix generalized inverses," SIAM Review, Vol. 12, 1970, pp. 573-576.

54. R. Penrose, "A generalized inverse for matrices," Proc. Cambridge Philos. Soc. Vol. 51, 1955, pp. 406-413.

55. S. Zlobec, "An explicit form of the Moore-Penrose inverse of an arbitrary complex matrix," SIAM Review, Vol. 12, 1970, pp. 132-134.

56. J.S. Frame, "Matrix operations and generalized inverses," IEEE Spectrum, 1964, pp. 209-220.

57. J.H. Ahlberg, E.N. Nilson and J.L. Walsh, The Theory of Splines and their Applications, New York: Academic Press, 1967.

58. C.L. Thompson and R.L. Weil, "Reducing the rank of (A-λB), "Proc. Am. Math. Soc., Vol. 26, 1970, pp. 548-554.

59. G.W. Westley and J.A. Watts, Computing Technology Center Numerical Analysis Library, Oak Ridge: Union Carbide, 1972.

60. L.B. Rall, Computational Solution of Nonlinear Operator Equations, New York: John Wiley, 1969.

61. D. Barton, I.M. Willers and R.V.M. Zaher, "The automatic solution of systems of ordinary differential equations by the method of Taylor series," Computer J., Vol. 14, 1971, pp. 243-248.

62. A.F. Cardenas and W.J. Karplus, "PDEL-A language for partial differential equations," Communications of the ACM, Vol. 13, 1970, pp. 184-191.

63. K.V. Roberts and J.P. Boris, "The solution of partial equations using a symbolic style of Algol," J. Comp. Phys., Vol. 8, 1971, pp. 83-105.

64. D. Gries, Compiler Construction for Digital Computers, New York: John Wiley, 1971.

65. W.M. Mansour, M.O.M. Osman and G.M.L. Gladwell, "Stability and kinematic accuracy of hydraulic copying mechanisms in metal cutting," to appear.

66. N.W. McLachlan, Theory and Applications of Mathieu Functions, Oxford: Oxford University Press, 1947.

67. E. Eisner, "The design of resonant vibrators," In Manson (ed.), Physical Acoustics Vol. 1, Pt.B, New York: Academic Press, 1964.

68. J.C. Snowdon, Vibration and Shock in Damped Systems, New York: John Wiley, 1968.

69. N. Basdekas and M. Chi, "Non-linear forced vibration of a non-prismatic beam due to combined flexure and stretching," J. Sound Vib., Vol. 15, 1971, pp. 447-454.

70. G. Fauconneau, "Variational approaches to the natural frequencies of a clamped uniform beam-column," J. Sound Vib., Vol. 14, 1971, pp. 339-355.

71. P. Lagasse and J. Van Bladel, "Square and rectangular waveguides with rounded corners," IEEE Trans., Vol. MTT-20, 1972, pp. 331-337.