Spring 1982

A Comparative Study of In-Core and Out-of-Core Equation Solvers for Microcomputer Applications

Salahuddin A. Siddiqui
University of Central Florida

A COMPARATIVE STUDY OF IN-CORE AND OUT-OF-CORE EQUATION SOLVERS FOR MICROCOMPUTER APPLICATIONS

BY

SALAHUDDIN AHMED SIDDIQUI, P.E.

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1982

ABSTRACT

This research evaluates the applicability to microcomputers of various methods for determining the solution of large systems of simultaneous linear algebraic equations. Such systems of equations characterize physical systems often encountered in Civil Engineering and other engineering disciplines.

Many methods of solution involving either in-core or out-of-core storage of data have been developed for use with large digital computers. These methods are reviewed and their applicability to microcomputers is evaluated. A comparison of several schemes is made regarding core size required, time of execution, and precision of results. The out-of-core solution schemes for banded matrices are found to be most applicable to microcomputers with large out-of-core storage capacity.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to Dr. Wayne E. Carroll, Dr. Martin P. Wanielista and Dr. Christian S. Bauer, members of his committee, for their encouragement and advice in preparing this thesis. Dr. Carroll has been instrumental in stimulating the author's interest in numerical methods in engineering and their computer applications.

Thanks are due to Dr. W. T. Segui of Memphis University, for critical information, and to Dr. G. Cantin of the Naval Post-Graduate School, Monterey, California, for his suggesting valuable reference material.

The author also wishes to acknowledge Mr. Harry Reynolds of Engineering Design, Inc., Longwood, Florida, for providing the computer facility.

Finally, warm personal appreciation goes to Priscilla V. Brain, Instructor, Department of English, University of Central Florida, for editorial assistance in English and to Willard Brain, II, head of the Computer Department, Creative Engineering, Inc., Orlando, Florida, for editorial assistance in numerical portions of this paper.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Analytical methods such as finite element methods find their application not only in the analysis of large trusses and frames, but also in analysis of plates and shells, as well as in hydraulics and other disciplines of engineering (1,2). These schemes generate a large set of algebraic simultaneous equations. Digital computers offer considerable savings in time compared to the manual methods for solving these equations.

The techniques available for solving the linear system of equations can be classified into two methods: direct or iterative. In the direct method, a solution for unknowns is obtained directly by a single application of a computational procedure. In the iterative method, the solution requires the repeated application of an algorithm. The direct method can be divided into two categories: in-core and out-of-core equation solvers. In the in-core method, the entire coefficient matrix resides in the random access memory of the computer throughout the operation. Out-of-core methods differ in that only portions of the coefficient matrix are brought from a peripheral device such as a disk into the computer memory as they are needed.

In the 1950's, the core storage of computers was small and the rates of transferring the data to and from magnetic tape were

slow (4). Because of this, direct methods of solution were restricted to small and simple problems. The larger and more complex problems were solved by iterative methods. Since then, large computers have been improving both in capacity and in the rate of transfer between the computer core and the peripheral storage. Therefore, one encounters fewer of the earlier limitations in solving the equations.

Due to the rapid improvement of the performance (capability of various applications) and price (affordability) of the microcomputer, many engineering firms have found that selecting microcomputers to fit their needs is now an economically feasible investment. Most microcomputers on today's market have from 4K to 64K bytes of in-core memory, although microcomputers with much larger capacity have recently become available. However, even though microcomputers are becoming more widely used, one drawback still remains; these computers, like the earlier larger computers, are limited in storage capacity and data transfer rate. Therefore, before selecting any microcomputer, one should make a comparative study of the different equation solvers and their techniques.

To demonstrate the application of microcomputers in the solution of large systems of equations, a typical truss analysis program was converted from FORTRAN to the BASIC computer language. The equations generated by this program were then solved by either in-core or out-of-core methods.

The computer support subroutine used for this work requires 2K bytes of memory. The disk operating system and the BASIC interpreter require an additional 10.5K bytes of memory, leaving 19.5K bytes available for program and data storage. Less sophisticated BASIC interpreters are available but are unsuitable for many engineering applications.

In preparing this thesis, a typical microcomputer system manufactured by the Apple Corporation with 32K memory and a single disk drive was used. The disk drive uses mini diskettes which can hold about 100,000 bytes of information. (Detailed information about II computers can be obtained from references 4 through 8.)

CHAPTER II

DIRECT METHODS

Frequently, in engineering analysis, one encounters a system of equations

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + z_{22}x_2 + \ldots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$$

where the x's are unknowns, and the a's and b's are constants, appearing in matrix form as seen below:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{Bmatrix}$$

or,

$$[A]\{X\} = \{B\}$$

In this case, [A] is the coefficient matrix, {X} is the vector of unknowns, and {B} is the vector of constants.

There are various methods of solving these equations. Direct methods, unlike the iterative technique, solve a system of linear

equations using a finite number of operations known in advance so that the number of operations performed is independent of the accuracy desired. An exact solution would be obtained if there were no roundoff errors (9), or truncation errors.

The accuracy of a solution is dependent on the condition and size of matrix, the precision of arithmetic performed by the computer and the algorithm used for the solution.

Two schemes for the solution of linear equations are presented here to illustrate the direct methods: the Gauss Elimination and the Cholesky Decomposition Method, also known as the Cholesky Square-Root Method. Both these methods require the matrices [A] and {B} in computer core for solution. Algorithms in which these matrices are stored out-of-core on a peripheral device, such as a disk drive, are reviewed in Chapter VI.

Gauss Elimination

The objective in Gauss Elimination is to transform the equation from the general form shown earlier to an upper triangular form as shown below:

$$\begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & & & \\ 0 & 0 & & a'_{nn} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} = \begin{Bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b_n \end{Bmatrix}$$

Once this transformation has been achieved, the last equation can be solved directly for x_n, and the remaining n-1 values of x_i can be obtained by the back substitution process. The entire solution vector can, thus, be obtained once this upper triangular system has been generated.

The method of converting a coefficient matrix to upper triangular form follows.

Consider a general system:

$$a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n = b_1$$
$$a_{12}x_1 + a_{22}x_2 + \ldots a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots a_{nn}x_n = b_n$$

Divide by a_{11}, obtaining:

$$x_1 + a^1_{12}x_2 + a^1_{13}x_3 + \ldots a^1_{1n}x_n = b^1_1$$

where:

$$a^1_{ij} = a_{1j}/a_{11} \qquad j = 2,n$$
$$b^1_1 = b_1/a_{11}$$

x_1 can now be eliminated from subsequent equations to obtain

$$a^1_{22}x_2 + a^1_{23}x_3 + \ldots a^1_{2n}x_n = b^1_2$$
$$a^1_{32}x_2 + a^1_{33}x_3 + \ldots a^1_{3n}x_n = b^1_3$$
$$a^1_{n2}x_2 + a^1_{n3}x_3 + \ldots a^1_{nn}x_n = b^1_n$$
$$\cdot$$

where

$$a^{1}_{ij} = a_{ij} - a_{il}a^{1}_{lj}$$

$$b^{1}_{i} = b_{i} - a_{i1}b_{1}$$

Similarly, each x (x_2, x_3, ...) can be eliminated from the equations. A general algorithm for the elimination of x_k is

$$\left.\begin{aligned} a^{k}_{kj} &= a^{k-1}_{kj} / a^{k-1}_{kk} \\ b^{k}_{k} &= b^{k-1}_{k} / a^{k-1}_{kk} \end{aligned}\right\} \quad ; j = k + 1, \ldots, n$$

and

$$\left.\begin{aligned} a^{k}_{ij} &= a^{k-1}_{ij} - a^{k-1}_{ik} a^{k}_{kj} \\ b^{k}_{i} &= b^{k-1}_{i} - a^{k-1}_{ik} b_{k} \end{aligned}\right\} \quad i, j = k + 1, \ldots, n$$

After the above procedure has been applied n-1 times, the original set of equations is reduced to

$$a^{n-1}_{nn} x_n = b^{n-1}_{n}$$

which is solved directly for x_n

$$x_n = \frac{b^{n-1}_{n}}{a^{n-1}_{nn}}$$

After the elimination is completed, the original system of equations has been transformed into the upper triangular system with

unit diagonal coefficients previously discussed. The general form of Kth Row is now

$$x_k + a^k_{k,k+1}\, x_{k+1} + \cdots a^k_{k,n}\, x_n = b^k_k$$

After obtaining the value of x_n, one can compute the remaining unknowns successively, applying, in reverse order, the formula

$$x_k = b^k_k - \sum_{j=k+1}^{n} a^k_{kj}\, x_j$$

Cholesky's Method

When matrix [A] is symmetric and is a positive definite, Cholesky's method presents a very efficient solution. This method is described below.

This method is based upon the fact that a symmetric matrix can be expressed as the product of a lower triangular matrix and an upper triangular matrix.

The solution of the system $[A]\{X\} = \{B\}$ reduces to finding the solution of two equivalent systems $[S]^T\{C\} = \{B\}$and $[S]\{X\} = \{C\}$. Now that the triangular matrix [S] is known, vector {C} can be found by back substitution. Once {C} is known, {X} can be found from $[S]\{X\} = \{C\}$.

The whole process involves the decomposition of the symmetric matrix [A] to find [S] and the vector {C}. Back substitution as in Gauss Elimination is then used to determine the unknown {X}. The algorithms and computer programs are given in reference 1.

To illustrate this solution technique, consider the following system:

$$[A] = [S]^T[S]$$

or

$$\begin{bmatrix} a_{11}a_{12}a_{13} \cdots a_{1n} \\ a_{21}a_{22}a_{23} \cdots a_{2n} \\ \vdots \\ a_{n1}a_{n2}a_{n3} \cdots a_{nn} \end{bmatrix} = \begin{bmatrix} s_{11}0 & \cdots 0 \\ s_{12}s_{22}0 & \cdots 0 \\ s_{13}s_{23}s_{33} & \cdots 0 \\ \vdots \\ s_{1n}s_{2n}s_{3n} & \cdots s_{nn} \end{bmatrix} \begin{bmatrix} s_{11}s_{12}s_{13} \cdots s_{1n} \\ 0 \quad s_{22}s_{23} \cdots s_{2n} \\ 0 \quad 0 \quad s_{33} \cdots s_{3n} \\ \vdots \\ 0 \quad \cdots \quad s_{nn} \end{bmatrix}$$

Multiplying the two [S] matrices yields:

$$s_{11}^2 = a_{11}$$

$$s_{11}s_{12} = a_{12}$$

$$s_{11}s_{13} = a_{13}$$

$$s_{11}s_{1n} = a_{1n}$$

Therefore, the coefficients of the first row of [S] become

$$s_{11} = \sqrt{a_{11}}$$

and

$$s_{1j} = \frac{a_{1j}}{s_{11}}$$

Continuing this process further,

$$s_{12}s_{11} = a_{12}$$

$$s_{12}^2 + s_{22}^2 = a_{22}$$

$$s_{12}s_{13} + s_{22}s_{23} = a_{23}$$

$$\vdots$$

$$s_{12}s_{1n} + s_{22}s_{2n} = a_{2n}$$

and

$$s_{1n}s_{11} = a_{1n}$$

$$s_{1n}s_{12} + s_{2n}s_{22} = a_{2n}$$

$$\vdots$$

$$s_{1n}^2 + s_{2n}^2 + \ldots s_{nn}^2 = a_{nn}$$

Therefore, a general algorithm for coefficients of [S] is:

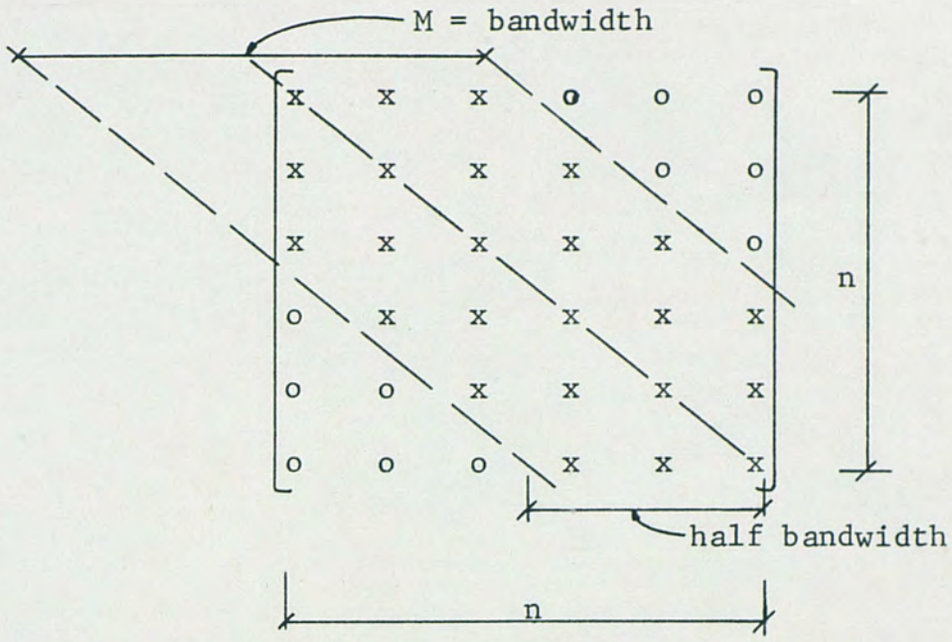
$$s_{ii} = (a_{ii} - \sum_{k=1}^{j-1} s^2_{ki})^{\frac{1}{2}}$$

$$s_{ij} = \frac{1}{s_{ii}} (a_{ij} - \sum_{k=1}^{i-1} s_{ki} s_{kj}), \text{ for } j > i$$

Cholesky's Method is efficient and in some cases, is faster than Gauss Elimination Scheme for symmetric matrices (2).

Symmetric Banded Matrices and Storage Requirements

In some applications, the non-zero coefficients in matrix [A] are located within a narrow band around the diagonal as shown in the following diagram:



In the case of banded matrices, one can store and work with only the terms in the bandwidth. Therefore, one can save storage and computation time because fewer terms are involved than when using the whole of matrix [A]. When the banded matrix is also symmetrical, only the terms on the diagonal plus the terms in the right-hand side up to M need to be stored and worked with, thus providing further savings in storage requirements and computation time for both the Gauss and Cholesky Methods. This can be seen from the formulae presented here.

For a non-symmetric non-banded matrix [A], the storage needed is nxn locations. For a banded matrix, the storage requirement is Mxn. For a symmetric banded matrix, the storage needed is $\frac{(M + 1)}{2}$ x n.

The most effective direct solution techniques currently used are basically applications of Gauss Elimination (10). However, this scheme can be applied to almost any set of simultaneous linear equations; and its effectiveness depends on the specific properties of matrix [A] such as symmetry, positive definiteness and bandedness (10).

CHAPTER III

ITERATIVE METHODS

When the application of iterative methods is to be used, an estimate is made for the values of the unknown vector {X}. This vector is corrected to its correct value in a series of successive iterations. The series of iterations is continued until the solution converges. Convergence is defined by selecting a small number by which the solution differs from the solution of preceding iteration.

Iterative methods are preferred for solving large sparse systems because they do not operate on zeros in the matrix and tend to be self-correcting and minimize round-off error (9). Such methods are particularly good for almost-diagonal or dominant-diagonal systems. A drawback to iterative methods is the possibility of slow or irregular convergence (9).

Of the many iterative techniques currently in use, the Jacobi and Gauss-Seidel Schemes, because of their widespread acceptance and utilization, are reviewed in the following sections.

Jacobi Scheme

The details of the Jacobi iterative technique are as follows:

$$a_{11}x_1^0 + a_{12}x_2^0 + a_{13}x_3^0 + \ldots + a_{1n}x_n^0 = b_1$$

$$a_{21}x_1^0 + a_{22}x_2^0 + a_{23}x_3^0 + \ldots + a_{2n}x_n^0 = b_2$$

$$\vdots$$

$$a_{n1}x_1^0 + a_{n2}x_2^0 + a_{n3}x_3^0 + \ldots + a_{nn}x_n^0 = b_n$$

The subscript zero denotes initial estimates of the values of $\{X\}$.

A subsequent trial yields new values of $\{X\}$ from the following equations:

$$x_1^1 = \frac{-a_{12}}{a_{11}} x_2^0 - \frac{a_{13}}{a_{11}} x_3^0 - \ldots - \frac{a_{1n}}{a_{11}} x_n^0 + \frac{b_1}{a_{11}}$$

$$x_2^1 = \frac{-a_{21}}{a_{22}} x_1^0 - \frac{a_{23}}{a_{22}} x_3^0 - \ldots - \frac{a_{2n}}{a_{22}} x_n^0 + \frac{b_2}{a_{22}}$$

$$\vdots$$

$$x_n^1 = \frac{-a_{n1}}{a_{nn}} x_1^0 - \frac{a_{n2}}{a_{nn}} x_2^0 - \ldots\ldots\ldots + \frac{b_n}{a_{nn}}$$

At this time, x_i^1 is compared with

$$|x_i^1 - x_i^0| \leq \varepsilon, \quad i = 1, 2, \ldots, n$$

where ε is the selected small number to check convergence. If this condition is satisfied, $\{X_i\}$ is taken as an acceptable approximate solution, otherwise further iteration is carried out.

In general, at the K_{th} iteration, the values of $\{X\}^K$ are given by

$$x_1^k = \frac{-a_{12}}{a_{11}} x_2^{k-1} - \frac{a_{13}}{a_{11}} x_3^{k-1} - \ldots - \frac{a_{1n}}{a_{11}} x_n^{k-1} + \frac{b_1}{a_{11}}$$

$$x_2^k = \frac{-a_{21}}{a_{22}} x_1^{k-1} - \frac{a_{23}}{a_{22}} x_3^{k-1} - \ldots - \frac{a_{2n}}{a_{22}} x_n^{k-1} + \frac{b_2}{a_{22}}$$

$$\vdots$$

$$x_n^k = \frac{-a_{n1}}{a_{nn}} x_1^{k-1} - \frac{a_{n2}}{a_{nn}} x_2^{k-1} - \ldots\ldots\ldots\ldots - \frac{b_n}{a_{nn}}$$

and the previously used convergence criteria are given by:

$$|x_i^k - x_i^{k-1}| \leq \varepsilon, \; i = 1, 2, \ldots, n$$

The Jacobi Method is slow to converge, but it serves as a yardstick against which most of the other iterative methods may be measured.

Gauss-Siedel Method

Because the Jacobi Method can be slow in its convergence, the Gauss-Siedel Method is often used instead. In this method, the solution for an unknown during an iteration is used in the computation of subsequent unknowns as shown below:

$$x_1^k = \frac{-a_{12}}{a_{11}} x_2^{k-1} - \frac{a_{13}}{a_{11}} x_3^{k-1} - \ldots - \frac{a_{1n}}{a_{11}} x_n^{k-1} + \frac{b_1}{a_{11}}$$

$$x_2^k = \frac{-a_{21}}{a_{22}} x_1^k - \frac{a_{23}}{a_{22}} x_3^{k-1} - \ldots - \frac{a_{2n}}{a_{22}} x_n^{k-1} + \frac{b_2}{a_{22}}$$

$$x_n^k = \frac{-a_{n1}}{a_{nn}} x_2^k - \frac{a_{n2}}{a_{nn}} x_2^k - \ldots\ldots\ldots\ldots\ldots + \frac{b_n}{a_{nn}}$$

The same criteria for convergence may be used;

$$|x_i^k - x_i^{k-1}| \leq \varepsilon, \; i = 1, 2, \ldots, n$$

Although it has been shown that the Gauss-Siedel Method converges twice as fast as the Jacobi Method, it is still not efficient enough for use with the large sparse matrices for which iterative methods are particularly suitable. This problem occurs because a considerably larger number of iterations must be performed in order to achieve the level of accuracy desired. From the tests shown by Segui (11), it can be seen that the results did improve during successive iterations in the Gauss-Siedel Scheme. The accuracy of the Gauss Elimination Method could not be obtained even after two hundred iterations taking as much as one hundred times longer than the Gauss Elimination Method.

CHAPTER IV

GRADIENT METHODS

The basis for gradient methods is that the solution to equation $[A]\{X\} = \{B\}$ can be shown to be equivalent to minimizing the quadratic function:

$$F = \{X\}^T[A]\{X\} - 2\{B\}^T\{X\}$$

This equation can be used to define a family of similar ellipsoids whose common center corresponds to a point on the surface of a particular ellipsoid. An iterative gradient method consists of successive steps from a larger to a smaller elipsoid with the point corresponding to the approximate solution moving closer to the common center.

Various gradient methods differ in their choices of direction for each step. In the steepest descent method, the iteration step is along the normal to ellipsoid. In the conjugate gradient method, each iterative step consists of two sub-steps: the first along the inward normal, and the second parallel to the previous iterative step. Fox and Evans (12,13) describe these methods with clarity.

Consider minimization of a certain quadratic form of the components $x_1, \ldots, x_n$ of the required solution of $[A]\{X\} = \{B\}$. The vector $\{X\} = [A]^{-1}\{B\}$, for example, minimizes the sum of

squares of the components of the residual vector $\{\gamma\} = [B] - [A]\{X\}$, given by

$$\{\gamma\}^T\{\gamma\} = (\{B\}^T - \{X\}^T[A]^T)(\{B\} - [A]\{X\})$$

Instead of solving the resulting normal equations directly, the method proceeds by successive approximation, making successive changes in the components of a starting approximation $\{X\}_o$ so that the quadratic form is steadily reduced to a minimum. This process can be presented by the non-stationary iterative scheme where $\{X\}_{i+1} = \{X\}_i + \alpha_i\{P\}_i$, where α_i is a scalar and $\{P\}_1$ a direction vector yet to be determined.

In the steepest descent method, the iteration step is along the inward normal to the ellipsoid. For symmetric positive-definite matrices, $\{P\}_i = \{\gamma\}_i$ and leads to the equation

$$\{X\}_{i+1} = \{X\}_i + \alpha_i(\{B\} - [A]\{X\}_i)$$

where:

$$\alpha_i = \frac{\{P\}_i^T\{\gamma\}_i}{\{P\}_i^T[A]\{\gamma\}_i} = \frac{\{\gamma\}_i^T\{\gamma\}_i}{\{\gamma\}_i^T[A]\{\gamma\}_i}$$

and

$$\{\gamma\}_i = \{B\} - [A]\{X\}_i$$

Convergence is relatively slow in the steepest descent method; therefore, it is not recommended for practical use (3).

The Conjugate Gradient Method requires that each iterative step consist of two sub-steps: the first along the inward

normal; and the second parallel to the previous iterative step, which improves the convergence rate. Isaac Fried (14) and Peter Y. Ko (15) present this method and further improvements to this method. Referring back to the equation

$$\{X\}_{i+1} = \{X\}_i + \alpha_i \{P\}_i$$

For positive definite symmetric matrices

$$\alpha_i = \frac{\{P\}_i^T \{\gamma\}_i}{\{P\}_i^T [A] \{\gamma\}_i}$$

For $\{P\}_i$, a combination of current residual and the previous vector $\{P\}_{i-1}$ is taken so that

$$\{P\}_i = \{\gamma\}_i + \beta_{i-1} \{P\}_{i-1}$$

and successive constants can be so chosen that the process is terminated in exactly N steps, N being equal to the number of unknowns. To this end, $\{P\}_i$ is made conjugate; that is orthagonal with respect to the matrix [A], to the previous $\{P\}_{i-1}$.

That is: $\{P\}_i^T [A] \{P\}_{i-1} = 0$, resulting in

$$\beta_{i-1} = \frac{-\{\gamma\}_i^T [A] \{P\}_{i-1}}{\{P\}_{i-1}^T [A] \{P\}_{i-1}}$$

or

$$\beta_i = \frac{-\{\gamma\}_{i+1}^T [A] \{P\}_i}{\{P\}_i^T [A] \{P\}_i}$$

with $\{X\}_o$ arbitrary and P-1 = 0 so that $\{P\}_o = \{\gamma\}_o$, further simplification leads to

$$\alpha_i = \frac{\sum_{j=1}^{n} \{\gamma^2\}_i}{\{P\}_i^T [A] \{P\}_i}$$

and

$$\beta_i = \frac{\sum_{j=1}^{n} \{\gamma^2\}_{i+1}}{\sum_{j=1}^{n} \{\gamma^2\}_i}$$

The gradient method, thus, becomes (14)

$$\{P\}_o = \{\gamma\}_o = \{B\} - [A]\{X\}_o$$

$$\alpha_i = \frac{\sum_{j=1}^{n} \{\gamma^2\}_i}{\{P\}_i^T [A] \{P\}_i}$$

$$\{X\}_{i+1} = \{X\}_i + \alpha_i \{P\}_i$$

$$\{\gamma\}_{i+1} = \{\gamma\}_i - \alpha_i [A] \{P\}_i$$

$$\beta_i = \frac{\sum_{j=1}^{n} \{\gamma^2\}_{i+1}}{\sum_{j=1}^{n} \{\gamma^2\}_i}$$

$$\{P\}_{i+1} = +\{\gamma\}_{i+1} + \beta_i \{P\}_i$$

After N cycles, the solution vector X is obtained as shown on the following page :

$$\{X\} = \alpha_{N-1}\{P\}_{N-1} + \{X\}_{N-1}$$

In this method, convergence is achieved quickly and efficiently, requiring a finite number of steps, N. Another important feature of this method is that multiplication is performed on the elementary level, and assembly is carried out only on the resulting vector. Therefore, there is no need to assemble the Global Matrix during the multiplication. This results in substantial savings in the amount of storage needed.

CHAPTER V

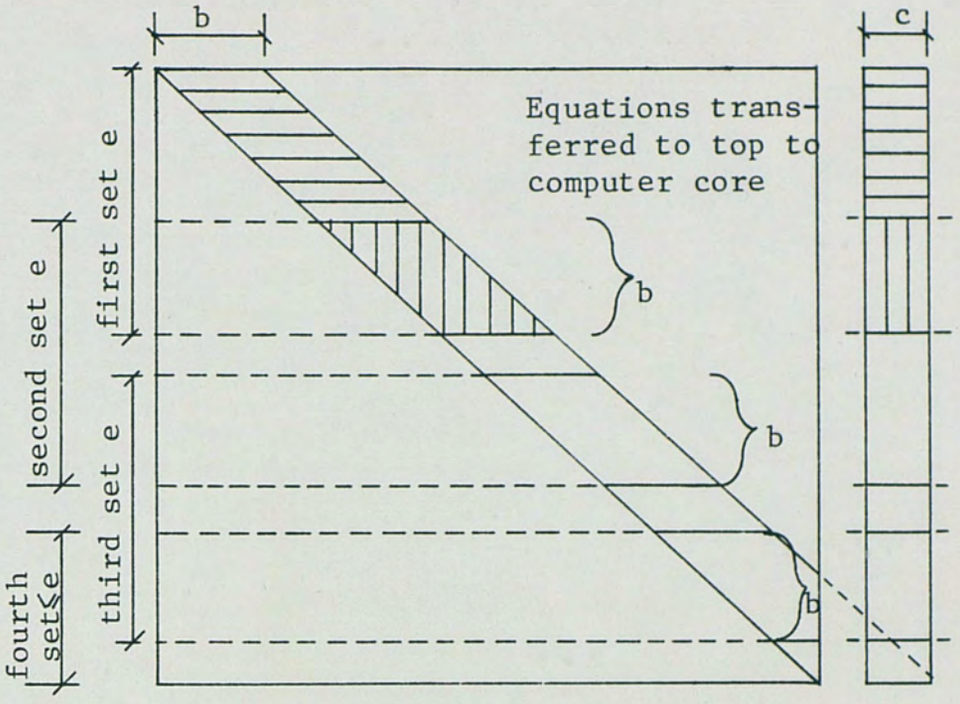
OUT-OF-CORE EQUATION SOLVERS

Blocked Banded Schemes

In Chapter II, the concept of banded matrices was explained, and the savings in storage space which the banded algorithms offer was also discussed. The in-core direct methods require the entire banded matrix [A] to be in the core of the computer throughout the process. However, if a suitable peripheral storage device such as a disk drive of large capacity is available with the computer, the coefficient matrix [A] and matrix [B] can be stored on the disk, and successive portions of coefficient matrix [A] and matrix [B] are then brought into the core of the computer for elimination.

Initially, all the equations are stored consecutively on the disk, the coefficients for each equation being arranged as a string of numbers within the half-band preceded by the corresponding load vector. Only a limited area of coefficients is required at any one time in the computer core, and the necessary partitioning is based on the available core.

Figure 1 illustrates that for each set of equations, e, held in computer core, the elimination proceeds as far as possible; that is, in the forward elimination, only the first e-b



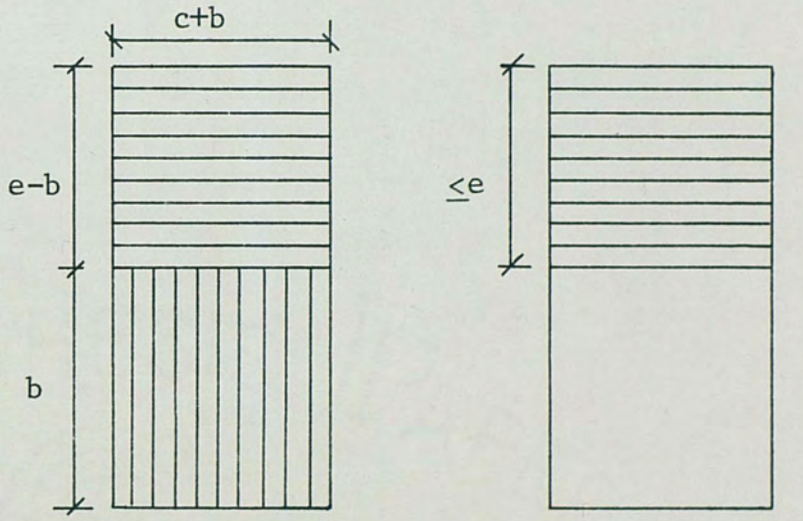


Fig. 1. Equation Partitioning.

equations can be used as pivot equations. The elimination area, hatched horizontally in the sketch, is then transferred back to disk; the equations still to be eliminated, hatched vertically in Figure 1, are transferred to the first locations in the computer core, and a new set of e-b equations is brought to core from disk. This procedure continues throughout the forward eliminations and is reversed during the back substitutions. Brooks and Botton (16) give a good explanation of this process.

If the data are shifted in and out of the core of the computer for each individual unknown, there will be too many in and out-of-core operations requiring a prohibitive amount of time. For this reason, the equations are divided into blocks, and each time a complete block is shifted in and out of computer core. William T. Sequi (11) has compiled two such programs. One is written by E.L. Wilson (17) and the other by T.J. Chung of the University of Alabama. The essential difference between the two algorithms lies in the number of blocks that are brought into the computer core from the disk or tape at any one time and in the method of elimination used.

In Wilson's Scheme (17), the coefficient matrix [A] is stored out-of-core on a magnetic tape or disk. It brings the coefficients of the upper half-band into core and eliminates the unknowns by the Gaussian Elimination Method. It requires two blocks of coefficients to be in-core at one time. This

program is for symmetric banded matrix [A] and has been converted from FORTRAN to the BASIC language. A detailed treatment is given in Chapter VI, where a comparison is made between this method and the in-core equation solver.

T.J. Chung's Scheme works for positive definite matrices only. In this method, the coefficients of symmetric banded matrix [A] are stored out-of-core. The coefficients of the lower half band are brought into core and are eliminated by Cholesky's Symmetric Decomposition Method (also known as the Square Root Method), rather than Wilson's Scheme, in which the upper half-band is brought into core and is eliminated by the Gauss Elimination Method. Another significant difference is that Chung's Scheme requires only one block of coefficients in-core at one time. The accuracy achieved is identical in both cases, but Chung's Scheme is considerably slower than Wilson's Scheme because of more shifting of data in- and out-of-core in Chung's Algorithm (11).

Equation Solver by Giles Cantin

The Equation Solver by Giles Cantin (18) uses symmetric matrix inversion of square sub-matrices as opposed to the direct Gaussian Elimination which occurs in Wilson's and Chung's Schemes. The basic concept of this scheme is briefly presented here. Consider the following system:

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ \cdot \\ R_M \\ \cdot \\ R_I \\ \cdot \\ R_N \end{Bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & \cdots & \cdots & K_{1M} & \cdots & \cdots & \cdots \\ K_{12}^T & K_{22} & K_{23} & K_{24} & \cdots & \cdots & K_{2,M+1} & \cdots & \cdots \\ K_{13}^T & K_{23}^T & K_{33} & K_{34} & K_{35} & \cdots & \cdots & K_{3,M+2} & \cdots \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ K_{1M}^T & K_{2M}^T & \cdots & \cdots & K_{M-1,M}^T & K_{M,M} & K_{M,M+1} & \cdots & K_{M,2M} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdots & K_{1-M+1,I}^T & \cdots & \cdots & K_{I-1,I}^T & K_{II} & K_{I,I+1} & \cdots & K_{I,I+M-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdots & \cdots & \cdots & \cdots & K_{N-M+1,N}^T & \cdots & \cdots & K_{N-1,N}^T & K_{NN} \end{bmatrix} \begin{Bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \cdot \\ \gamma_M \\ \cdot \\ \gamma_I \\ \cdot \\ \gamma_N \end{Bmatrix}$$

where $[K_{ij}]$ can be treated as square sub-matrices. The first equation is solved for $\{\gamma\}_1$ to get:

$$\{\gamma\}_1 = [K_{11}]^{-1}(\{R\}_1 - [K_{12}]\{\gamma\}_2 - [K_{13}]\{\gamma\}_3 \cdots [K_{1M}]\{\gamma\}_M)$$

After substitutions from 2 to M,

$$\{\bar{R}\}_2 = \{R\}_2 - [K_{12}]^T[K_{11}]^{-1}\{R\}_1$$

$$\{\bar{R}\}_M = \{R\}_M - [K_{1M}]^T[K_{11}]^{-1}\{R\}_1$$

for the load vectors. All the coefficients in rows two to M from column two to M are reduced by similar operations. A typical example is

$$[\bar{K}_{34}] = [K_{34}] - [K_{13}]^T[K_{11}]^{-1}[K_{14}]$$

Then equation two is solved for γ_2 and is substituted in equations 3 to M + 1, and the entire process is repeated until the last equation becomes

$$\{\bar{R}\}_N = [\bar{K}_{NN}]\{\gamma\}_N$$

This equation is now solved for γ_n, and a simple process of back substitution in reverse order gives all the unknowns. The entire process is valid whether the elements of matrix [K] are individual coefficients or square sub-matrices of coefficients.

Giles Cantin (18) developed this algorithm for symmetric banded matrices. The scheme requires no more than three different blocks of coefficients at the same time in-core, for elimination, together with corresponding three blocks of load vectors for back substitution. This scheme does not pose any limits on the half bandwidth or on the total number of equations (18); however, it does involve symmetric matrix inversions. Matrix inversion is found to be an inefficient technique (11). Elimination and back substitution, or their compact equivalents, are always faster, and the inverse should be obtained only if it is needed explicitly (12).

Frontal Solution Method

Bruce M. Irons (19) presents a scheme for positive definite matrices. Frontal solution is a variation of Gaussian Elimination (20) that utilizes the external storage effectively. It is

based upon the fact that Gauss Elimination can be performed in stages with only the coefficients within the so-called active area being required in-core at any stage. In most solutions, when algorithms are used in finite element programs, the stiffness matrix is assembled entirely before starting Gaussian Elimination. Frontal solution schemes effect considerable savings in backup storage requirements by assembling and reducing the equations at the same time, thus eliminating the need to save the total unreduced stiffness matrix.

The stiffness matrix is built by summing the contributions $[A]_i$ of each element i properly addressed by the element localizing matrices $[L]_i$. Therefore,

$$[A] = \Sigma_i [L]_i^T [A]_i [L]_i$$

In the frontal method of solution, the Global Stiffness Matrix is not assembled completely. The operation $\Sigma_i [L]_i^T [A]_i [L]_i$ is limited to a relatively small number of finite elements, and the corresponding part of the Global Stiffness Matrix is partitioned in the form

$$[A]_{\text{step } j} = \begin{bmatrix} A_{CC} & A_{CR} \\ A_{RC} & A_{RR} \end{bmatrix}$$

where the subscript C corresponds to degrees of freedom that may be eliminated at that step; that is, degrees of freedom which are not coupled with those of the elements that are still to be

assembled. The subscript R indicates that the degree of freedom is retained. At each step j, the Gaussian Elimination is achieved for condensable degrees of freedom at the end of each elimination step. Therefore,

$$[A^*]_{RR_j} = [A]_{RR_j} - [A]_{RC_j}[A]^{-1}_{CC_j}[A]_{CR_j}$$

$$\{B^*\}_{R_j} = \{B\}_{R_j} - [A]_{RC_j}[A]^{-1}_{CC_j}\{B\}_{C_j}$$

The $[A^*]_{RR_j}$ matrix is retained for assembling with the next sequence of finite elements to form a system similarly for step j + 1:

$$[A]_{j+1} = [L^*]^T_{j+1}[A^*]_{RR_{j+1}}[L^*]_{j+1} + \Sigma[L]^T_i[A]_i[L]_i$$

$$[A]_{j+1} = \begin{bmatrix} [A]_{CC_{j+1}} & [A]_{CR_{j+1}} \\ [A]_{RC_{j+1}} & [A]_{RR_{j+1}} \end{bmatrix}$$

A similar operation is achieved to form the load vectors.

In order to understand the "front" of the equations, one should consider the finite element grid for plane stress as shown in Figure 2. There are two equations associated with each node, corresponding to displacements u and v. To eliminate degrees of freedom of node 1, one has to assemble equations corresponding to nodes 1, 2, m and m+1. This requires the stiffness matrices of elements 1, 2, q and q+1 to be calculated and assembled; and

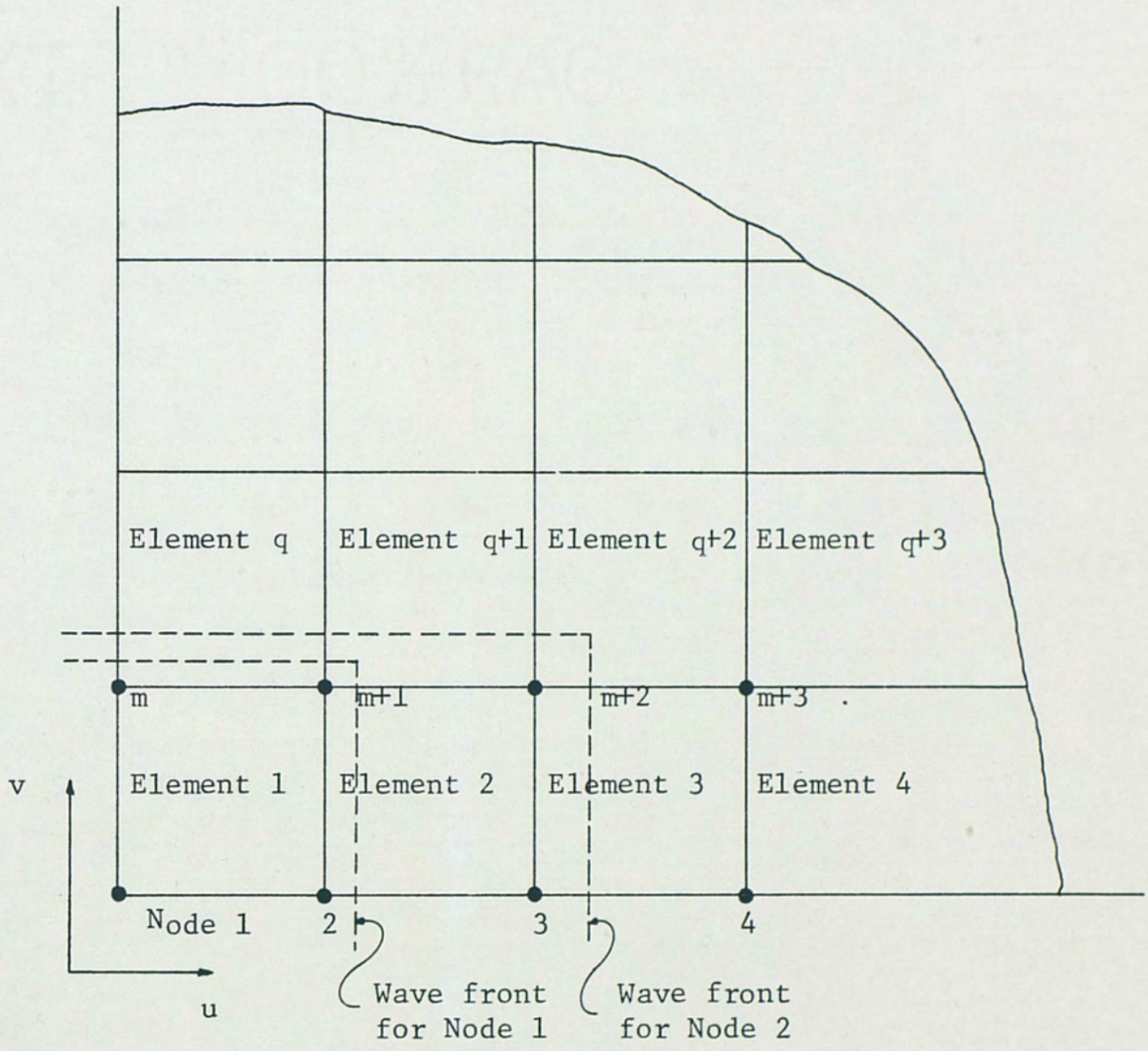


Fig. 2. Frontal idealization for plane stress.

then the degrees of freedom corresponding to node 1 are statically condensed out. Next, in order to eliminate the equations for node 2, the final equation corresponding to degrees of freedom at nodes 3 and m+2 is also needed, requiring that stiffness matrices of elements 3 and q+2 be first calculated and assembled and so on. The procedure consists of statically condensing out one degree of freedom after the other, assembling only those element stiffness matrices that are actually required during the specific condensation. The finite elements that must be considered in this condensation corresponding to one specific node define the wave front at that time.

The equations are assembled in the order of the elements. The length of wave front and, therefore, the half bandwidth are determined by the element numbering; thus, an effective ordering of elements is necessary.

The frontal solution technique has the advantage that at any one time only the equations that are currently needed are assembled in-core. It should also be noted that elements can be added with relative ease because no nodal point renumbering is required (10).

The essential and important limitation of the frontal method appears to be its often excessive requirements in core storage as the front becomes large. When this happens, there is no simple way of avoiding the requirements of a large memory allocation which may override the intrinsic advantages of simplicity and efficiency in dealing with variable bandwidth (21).

CHAPTER VI

A PROGRAM FOR MICROCOMPUTERS

General

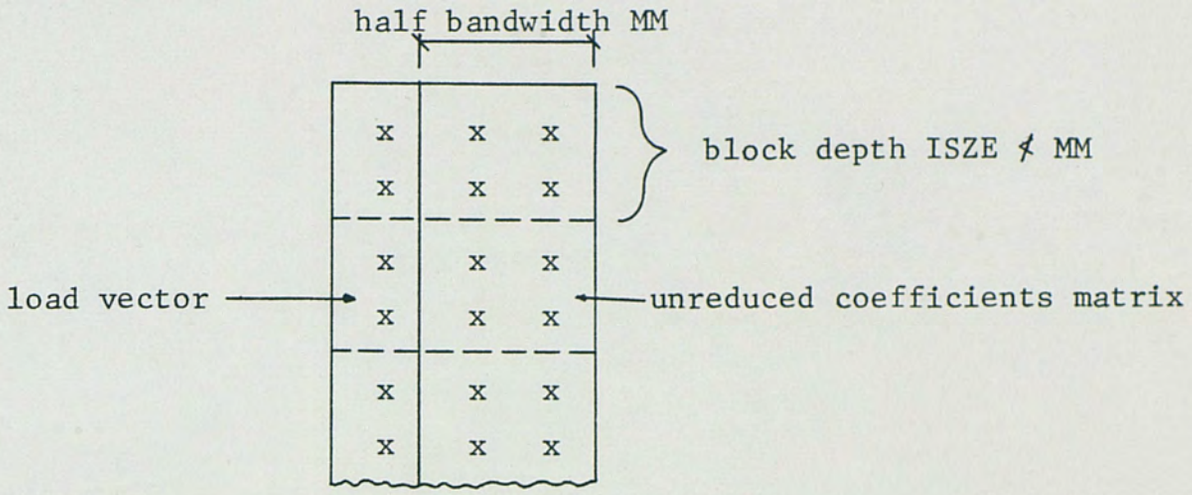
In the preceding chapters, a brief survey was made of several methods to solve a system of linear simultaneous equations. A microcomputer has a limited core memory, and a large system of equations cannot be solved in the core alone by direct methods even when a symmetric banded matrix is used. In iterative methods such as the Jacobi and Gauss-Siedel Methods, there is a possibility of slow or irregular convergence. In general, iterative methods are only suitable when the system matrix is almost diagonal or dominant-diagonal, and thus gives rapid convergence. Front solvers produce savings of storage and execution time, but they prohibit the solutions of very large systems (18) because of large front width that cannot be accommodated in computer core. The blocked banded equation solvers seem to be a good choice for microcomputers for use for a wide range of problems encountered in finite elements and frame analysis. All the three blocked banded schemes discussed deserve serious consideration for microcomputer application. The out-of-core scheme presented by Wilson (17) requires less in- and out-of-core operations than the scheme presented by Chung (in Segui)(11) and uses elimination

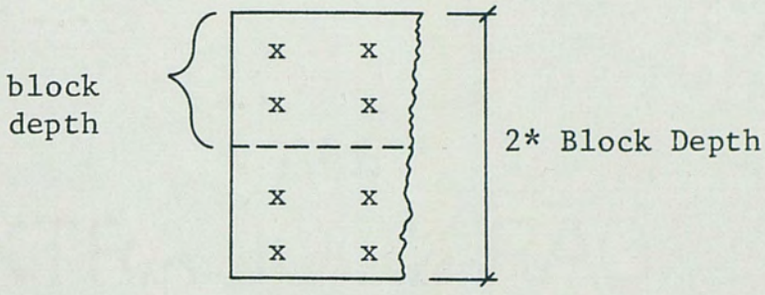
and back substitution rather than matrix inversion as in Cantin's Scheme (18). Because of this, the blocked banded scheme presented by Wilson was selected for implementation.

This author has implemented the technique presented by E.L. Wilson into a BASIC language program for the microcomputer. The input and output operations depend upon the microcomputer used. Still, the applicability of the processes as a whole is general enough. It is assumed that a fast out-of-core device of large capacity like a suitable disk drive is available.

The upper half-band and constant vector must be written on a disk file in blocks. Referring to Figure 3, the blocks can now be made with the following restrictions. The width of block should at least be equal to the half bandwidth. The depth of the block should at least be equal to half bandwidth for reduction of the block to proceed. Consider matrix B:

$$\begin{bmatrix} b_1 \\ b_2 \\ \hline b_3 \\ b_4 \\ \hline b_5 \\ b_6 \\ \hline b_7 \\ 0 \end{bmatrix}$$





At forward elimination stage matrix A holds two blocks of coefficients

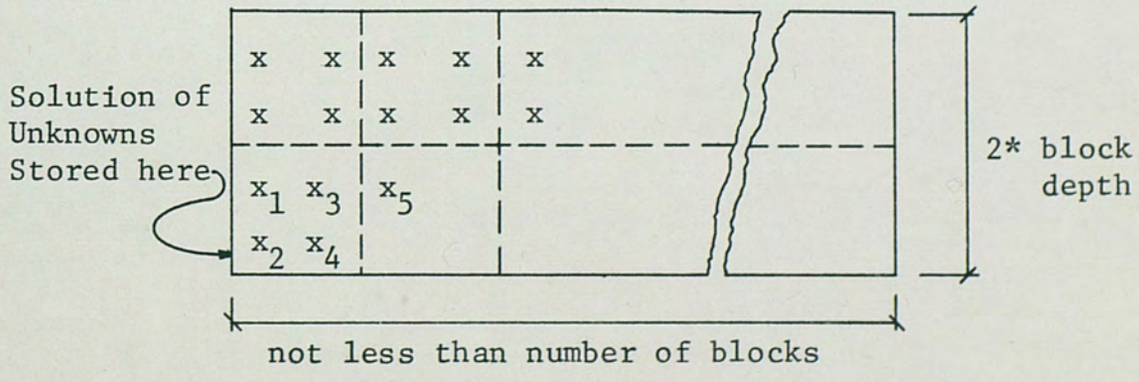


Fig. 3. Matrix blocks in disk file and computer core.

And upper half-band coefficients:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \hline a_{31} & a_{32} \\ a_{41} & a_{42} \\ \hline a_{51} & a_{52} \\ a_{61} & a_{62} \\ \hline a_{71} & a_{72} \\ 0 & 0 \end{bmatrix}$$

Here the block size is 2 x 2. Note that dummy zeros have to be put in last rows of both matrices [A] and {B} to keep the same size for all of the blocks.

To write these coefficients in the disk file, start with the first term of vector {B}, then write the coefficients on the first row of the upper half-band. After that, the second term of vector {B} is followed by coefficients of the second row of the upper half-band and so on as shown below:

$$\begin{array}{lll} b_1 & a_{11} & a_{12} \\ b_2 & a_{21} & a_{22} \\ b_3 & a_{31} & a_{32} \\ b_4 & a_{41} & a_{42} \\ \vdots & & \\ b_7 & a_{71} & a_{72} \\ 0 & 0 & 0 \end{array}$$

Once the complete file is written, one can proceed with the solution. A general flow diagram of this process is shown in Figure 17. Detailed flow charts are given in Figure 18 in the Appendix. A listing of the program is also given.

Refer to Figure 3 and note that matrix [A] in the program represents the coefficients that are required at any one time in the core. The first dimension of this matrix must at least be equal to twice the depth of the block because this matrix holds two blocks of coefficients for reduction; and, at the end of back substitution, the lower half of this matrix contains the solution of the unknowns. The second dimension of matrix [A] must at least be equal to the half bandwidth and at least equal to the number of blocks. For an explanation of this restriction, one must look at the flow chart for back substitution in Figure 18 in the Appendix. The back substitution starts from the last unknown and, after finding the value, proceeds upward. But before it proceeds upward, the solution of this unknown is stored in lower half of the matrix A. This is needed because the terms of vector {B} keep changing in the entire process. Figure 3 shows these restrictions.

Test Matrices

To determine the accuracy of the solution obtained by the program, test matrices of known solutions were used. Gregory and Karney (22) document some of these. For a half bandwidth of 2, this test matrix was used in this program:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ & -1 & 2 & -1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix} \begin{Bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_n \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{Bmatrix}$$

With the solutions $X_i = i$ for $i = 1, \ldots, n$.

For half bandwidth = 4, the following test matrix was used:

$$\begin{bmatrix} 5 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 5 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 5 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 5 & 1 & 1 & 1 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 1 & 1 & 1 & 5 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 5 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 5 \end{bmatrix} \begin{Bmatrix} X_1 \\ X_2 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ X_n \end{Bmatrix} \begin{Bmatrix} 8 \\ 9 \\ 10 \\ 11 \\ 11 \\ . \\ 11 \\ 10 \\ 9 \\ 8 \end{Bmatrix}$$

Which has solutions $X_i = 1$ for $i = 1, \ldots, n$. Sub-routines for generation of the upper half-band coefficients for these matrices are included in the program listing.

Maximum Capability

The computations for approximately finding the maximum number of equations that can be handled by the in-core and out-of-core banded solvers can proceed as follows. The in-core program takes up close to 2200 bytes, and the out-of-core program takes up about 4200 bytes. Both include sub-routines for test matrix generation. 10.5K is taken by the disk operations system, and 2K is taken by the computer operating system. Space available for matrix terms in the in-core solver is then:

$$(32 - 12.5) \times 1024 - 2200 = 17768 \text{ bytes}$$

The system configuration used requires five bytes for each matrix term. Therefore, the number of matrix terms which can be stored in core is 17768/5 or 3553 terms. For a half bandwidth of 2, we need B(N), A(N,2), requiring N + 2N = 3N terms. The maximum number of equations is then 3553/3 or 1184 equations. Similarly, with a half bandwidth of 4, the maximum number of equations is 3553/5 or 710. Table 1 gives the values of N for various half bandwidths. For the out-of-core solver, the space available is given by

$$(32 - 12.5) \times 1024 - 4200 = 15768 \text{ bytes}$$

The number of matrix terms which can be in-core at any one time is then 15768/5 = 3153 terms. For a half bandwidth of 2, a figure of at least B(N), A(2 x 2, N/2), requiring

$$N + \frac{2 \times 2}{2} N = 3N \text{ terms}$$

The maximum number of equations = 3153/3 = 1051. Actually, N = integer $\frac{1051}{\text{block depth}}$ x depth of block = 1050. The half bandwidth of 4 requires

$$N + \frac{2 \times 4}{4} N = 3N \text{ terms}$$

Therefore, the number of equations = 3153/3 or 1051 equations. Actually, N = Integer $\frac{1051}{4}$ x 4 = 1048.

TABLE 1

MAXIMUM NUMBER OF EQUATIONS THAT CAN BE SOLVED WITH A PARTICULAR HALF BANDWIDTH

Half Bandwidth	In-Core Solver	Out-of-Core Solver
2	1184	1050
4	710	1048
5	592	1050
10	323	1050
15	222	1050
20	169	1040
25	136	1050
30	114	1050

Table 1 shows that as the half bandwidth increases, the in-core equation solver is capable of solving a decreasing number of equations. The out-of-core solver is not similarly limited provided that the disk system has a large capacity; otherwise,

the storage needed will exceed the disk capacity before the solution is obtained. In the system configuration selected the real number can have nine digits of precision excluding the decimal point and the sign. To be recorded on the disk file, each character of this real number requires one byte for the record length. For example, using a disk file record length of 12 bytes, a disk of 100,000 bytes capacity can solve 1048 equations with a half bandwidth of 4. To solve the same number of equations with a half bandwidth of 30, one needs a disk of close to 800,000 bytes capacity. The writing and reading commands and lengths of records all play important roles in this regard.

Therefore, a larger system of equations can be solved with disks of greater capacity, using the smallest possible record length for the disk files and making efficient use of writing and reading commands.

Time of Execution

The test matrices described in the previous section on test matrices were used for this section and are shown below. For a half bandwidth of 2, this matrix was used:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ & -1 & 2 & -1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix} \begin{Bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_n \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{Bmatrix}$$

For a half bandwidth of 4, the following matrix was used:

$$\begin{bmatrix} 5 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 5 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 5 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 5 & 1 & 1 & 1 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 1 & 1 & 1 & 5 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 5 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 5 \end{bmatrix} \begin{Bmatrix} X_1 \\ X_2 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ X_n \end{Bmatrix} = \begin{Bmatrix} 8 \\ 9 \\ 10 \\ 11 \\ 11 \\ . \\ 11 \\ 10 \\ 9 \\ 8 \end{Bmatrix}$$

The time of execution was recorded by varying the number of unknowns. In Figure 4, execution time can be seen to be linearly dependent on the number of equations being solved. The reasons for this follow.

In a computer, addition and subtraction are performed many times faster than multiplication and division. Because of this, the time of execution depends largely on the number of multiplications and divisions used. Morris, Wilbur and Utku (23) observe that the number of multiplications involved for the decomposition of matrix [A] is basically proportional to T when

$$T = n \times \text{square of half bandwidth}$$

demonstrating that for a constant bandwidth, T will be proportional to the total number of unknowns, n; and therefore, the time of

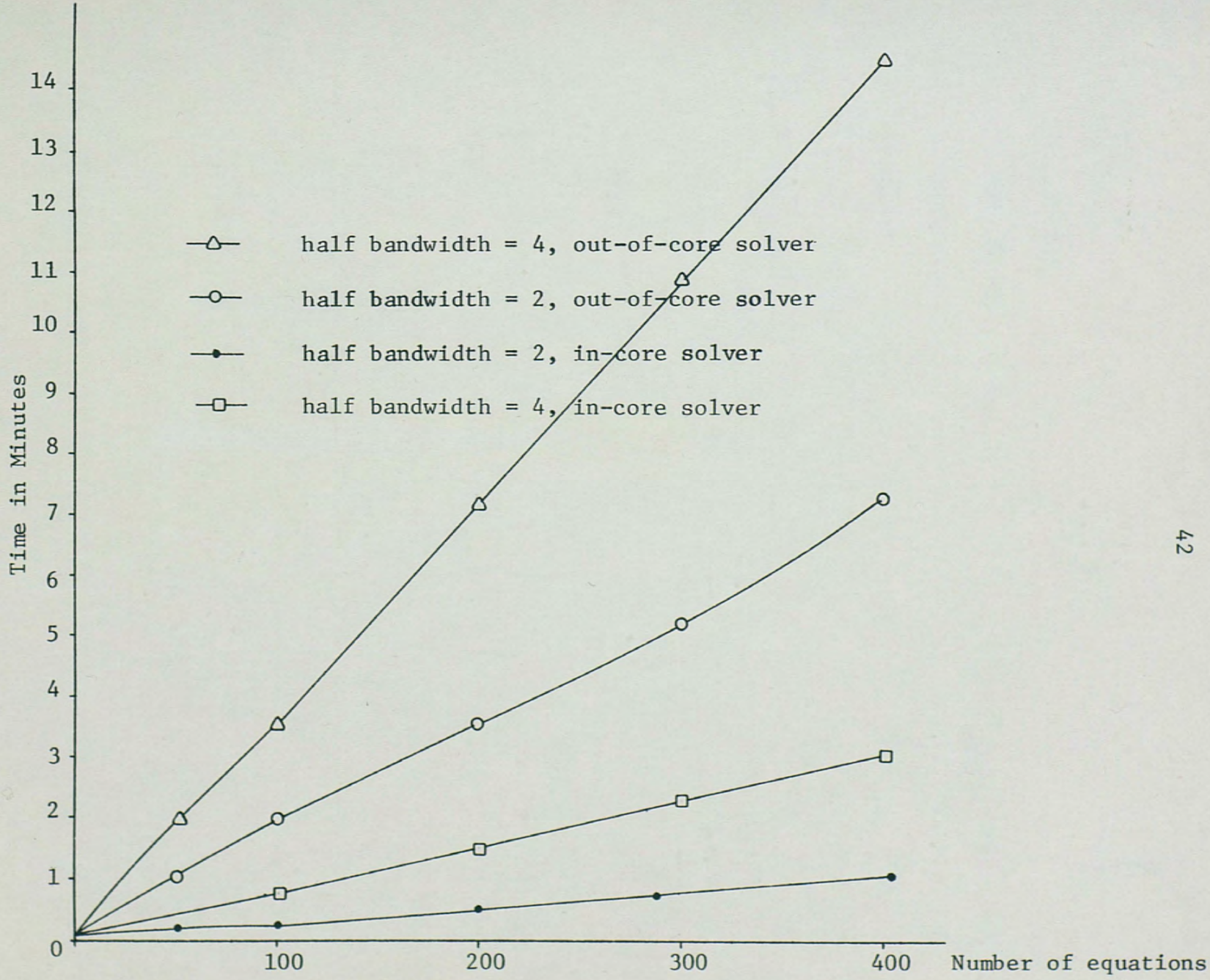


Fig. 4. Execution time versus number of equations.

execution will vary almost linearly with the increase in n. This is confirmed by Figure 4.

If the half bandwidth is taken as n rather than a constant, the matrix is full; and the number of multiplications is proportional to n^3, as reported by many authors.

Execution time for banded and symmetric in-core equation solvers is also plotted on the same graph, from which one notices that the out-of-core solver took almost six times as long for solving the same number of equations. However, when the system of equations is large, to need storage greater than the in-core capacity of the computer as discussed in the previous section, the equations cannot be solved using in-core system and one has to use out-of-core solver.

Truncation Error

In several instances, the exact number obtained from a multiplication or division is larger than the word length used by the computer. This number is truncated to conform to the word length of the computer. As the number of unknowns increases, the multiplications and divisions increase considerably, thus increasing the truncation error.

Sequi (16) gives two error measurements to compare the accuracy in different situations;

$$\text{Overall Inaccuracy} = \text{Error Measure} = \frac{1}{n} \left| \Sigma e_i^2 \right|^{1/2}$$

$$\text{Maximum Error} = \text{Max} \left| e_i \right|$$

where e_i represents the difference between the exact and computed value of X.

In Figure 5 overall error measure has been plotted versus the number of equations for a half bandwidth of 2. This curve is non-linear, increasing rapidly with an increasing number of equations. Exactly the same curve was obtained when trials were made varying the depths of the block. In Figure 6, maximum error has been plotted versus the number of equations for a half bandwidth of two. This curve is also non-linear and increases rapidly with increases in the number of equations. The overall error measure and the maximum error curve do not change with changes in block depth, as can be seen from Figures 7 and 8.

In the tests performed, both the overall error measure and the maximum error remained the same for the in-core and out-of-core equation solvers. Therefore, the out-of-core solver can be used where the precision of the in-core banded solver is acceptable.

To perform a curve fit for the overall error measure curve in Figure 5 and the maximum error curve in Figure 6, a second order regression equation of the following form was used:

$$y = c + a_1 x + a_2 x^2$$

where:

y = dependent variable

c = constant

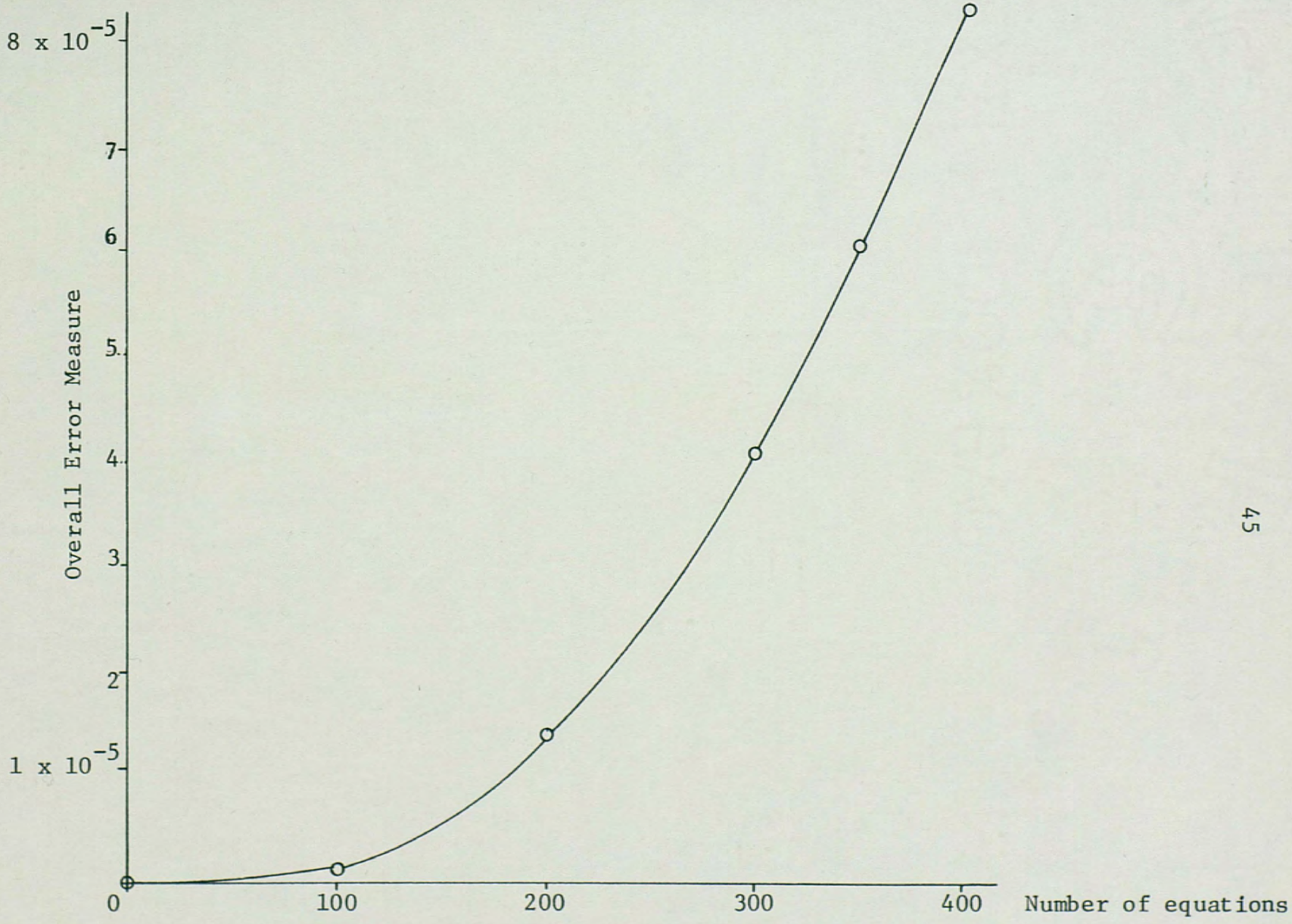


Fig. 5. Overall error measure versus number of equations.

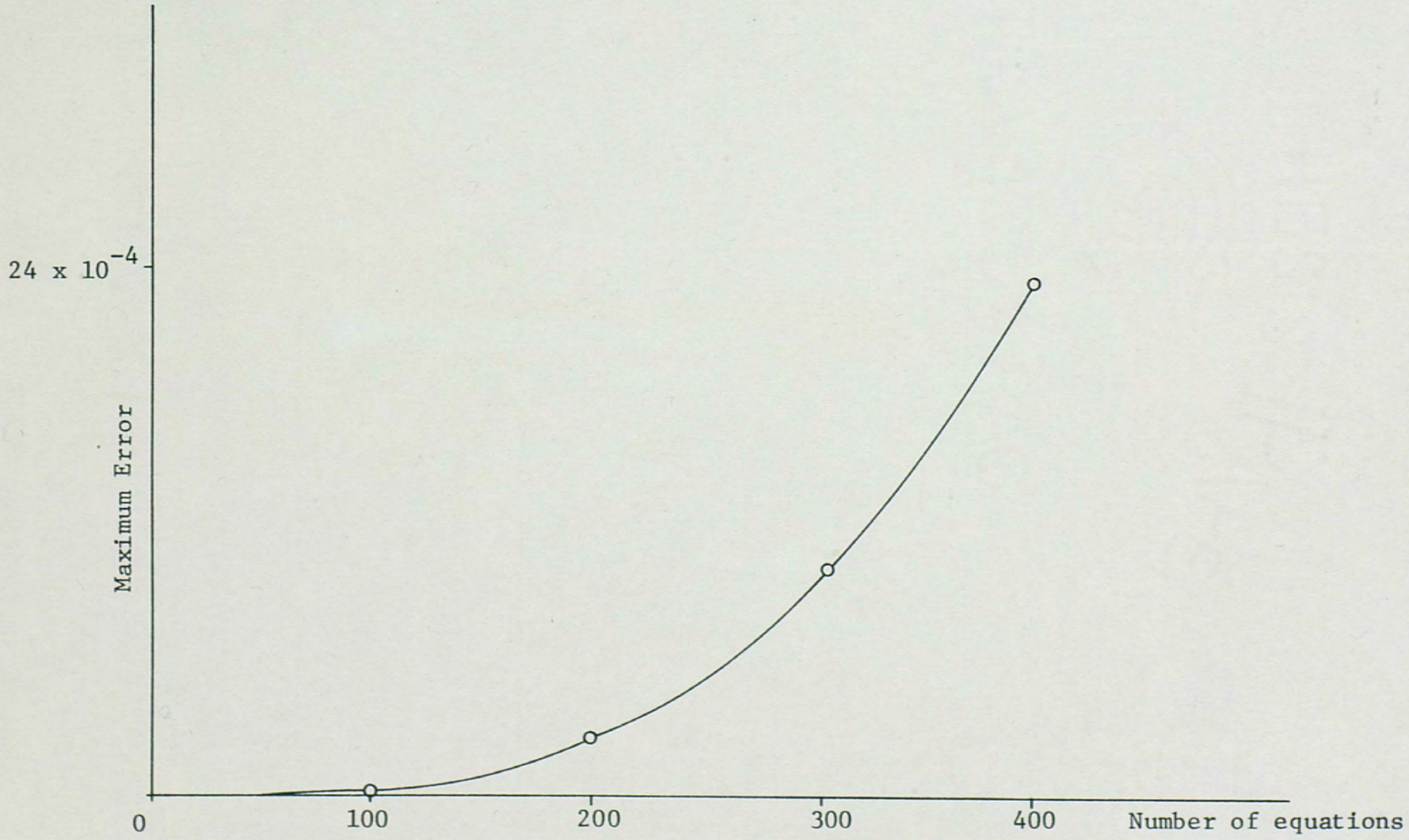


Fig. 6. Maximum error versus number of equations.

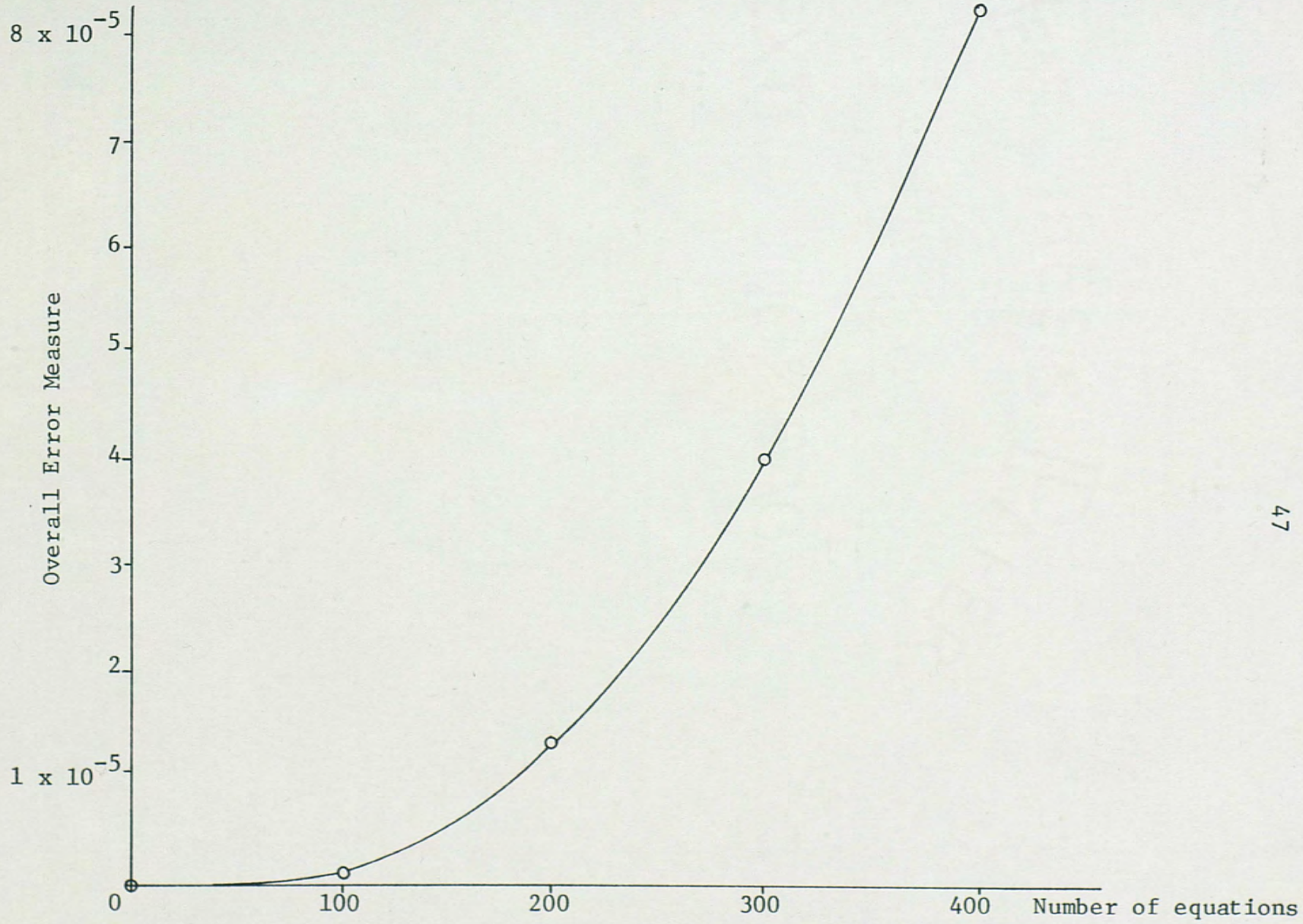


Fig. 7. Overall error measure versus number of equations with different block depths.

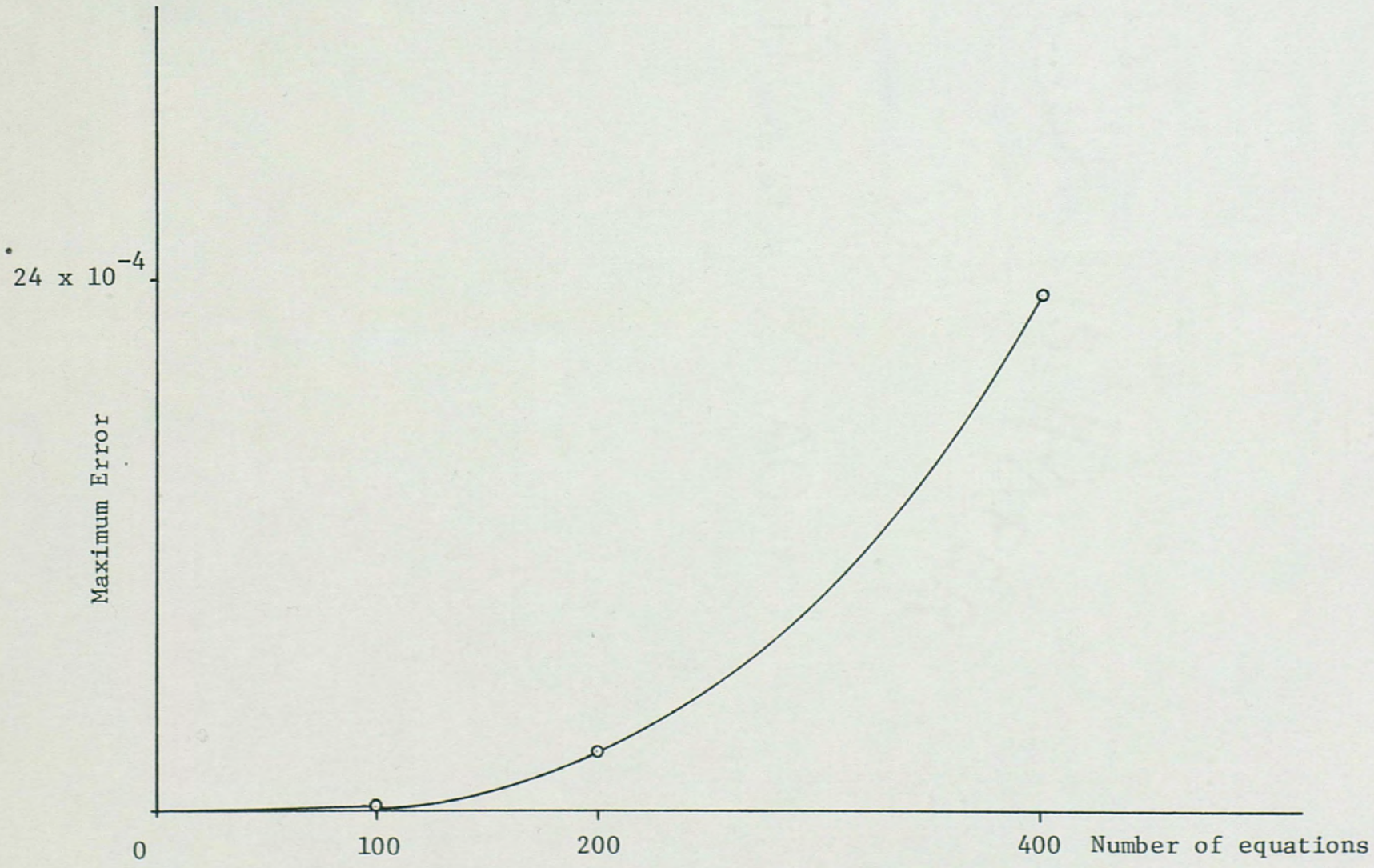


Fig. 8. Maximum error versus number of equations with different block depths.

a_1 and a_2 = coefficients of independent variables x, x^2, ..., x^n, respectively. A computer program from Pool and Borchers (24) was used to find the values of the various coefficients of the regression equation.

Tables 2 and 3 show the coordinates of the overall error measure and maximum error curves that were used as input to the computer. Tables 4 and 5 show the output from the computer, giving the constants and the coefficients of the regression equation. Figures 9 and 10 show the curve fits for the overall error measure and maximum error computed from the regression equations shown in Tables 4 and 5. From these curve fits, one can estimate the number of equations that can be solved to the accuracy desired. For example, if the overall accuracy required in an application is .1%, the curve in Figure 9 shows that 1300 equations can be solved. When greater precision is needed, one has to use double precision arithmetic. In many microcomputers, double precision arithmetic needs special sub-routines and requires almost twice the storage space in the computer core and disk as needed for single precision arithmetic.

Effect of Depth of Block on Time of Execution

The depth of block refers to the number of rows in each block. The present version of this program requires each block to have the same depth. A smaller depth of block results in a larger number of blocks for the same total number of unknowns.

TABLE 2

COORDINATES DATA FROM OVERALL ERROR MEASURE GRAPH

X	Y
Number of Equations	Error Measure
0	0
100	$.2614 \times 10^{-5}$
200	1.2868×10^{-5}
300	4.1919×10^{-5}
400	8.2150×10^{-5}

TABLE 3

COORDINATES DATA FROM MAXIMUM ERROR GRAPH

X	Y
Number of Equations	Maximum Error
0	0
100	$.3734 \times 10^{-4}$
200	2.5636×10^{-4}
300	10.5500×10^{-4}
400	23.3793×10^{-4}

TABLE 4

REGRESSION EQUATION FOR OVERALL ERROR

c	6.2219952E-07
a_1	-6.50549926E-08
a_2	6.71649983E-10
Equation	y = (6.2219952E-07) - (6.50549926E-08) * X + (6.71649983E-10) * X^2

TABLE 5

REGRESSION EQUATION FOR MAXIMUM ERROR

c	3.73076976E-05
a_1	-3.08019402E-06
a_2	2.19342851E-08
Equation	y = (3.73076976E-05) - (3.08019402E-06) * X + (2.19342851E-08) * X^2

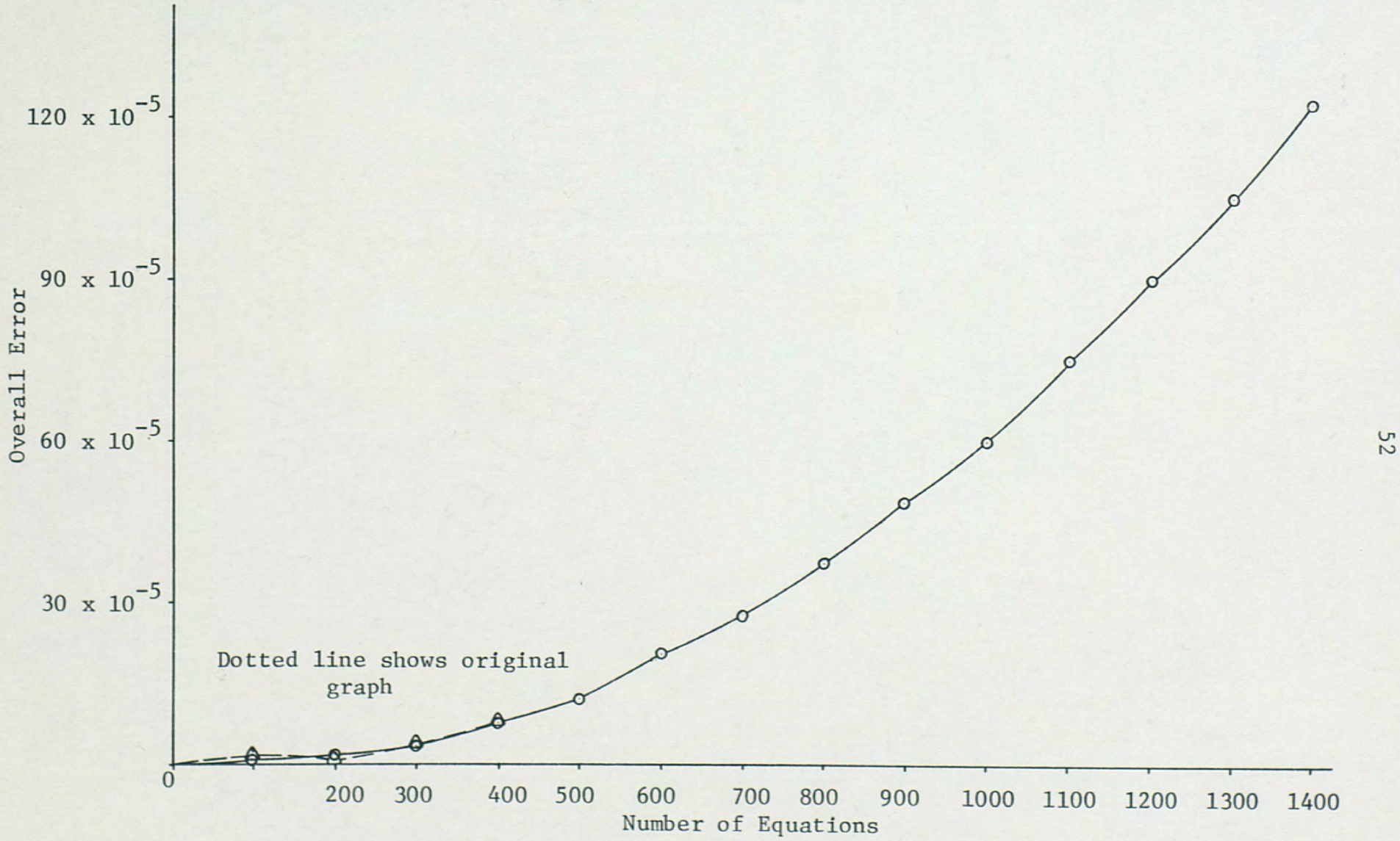


Fig. 9. Curve fit for overall error.

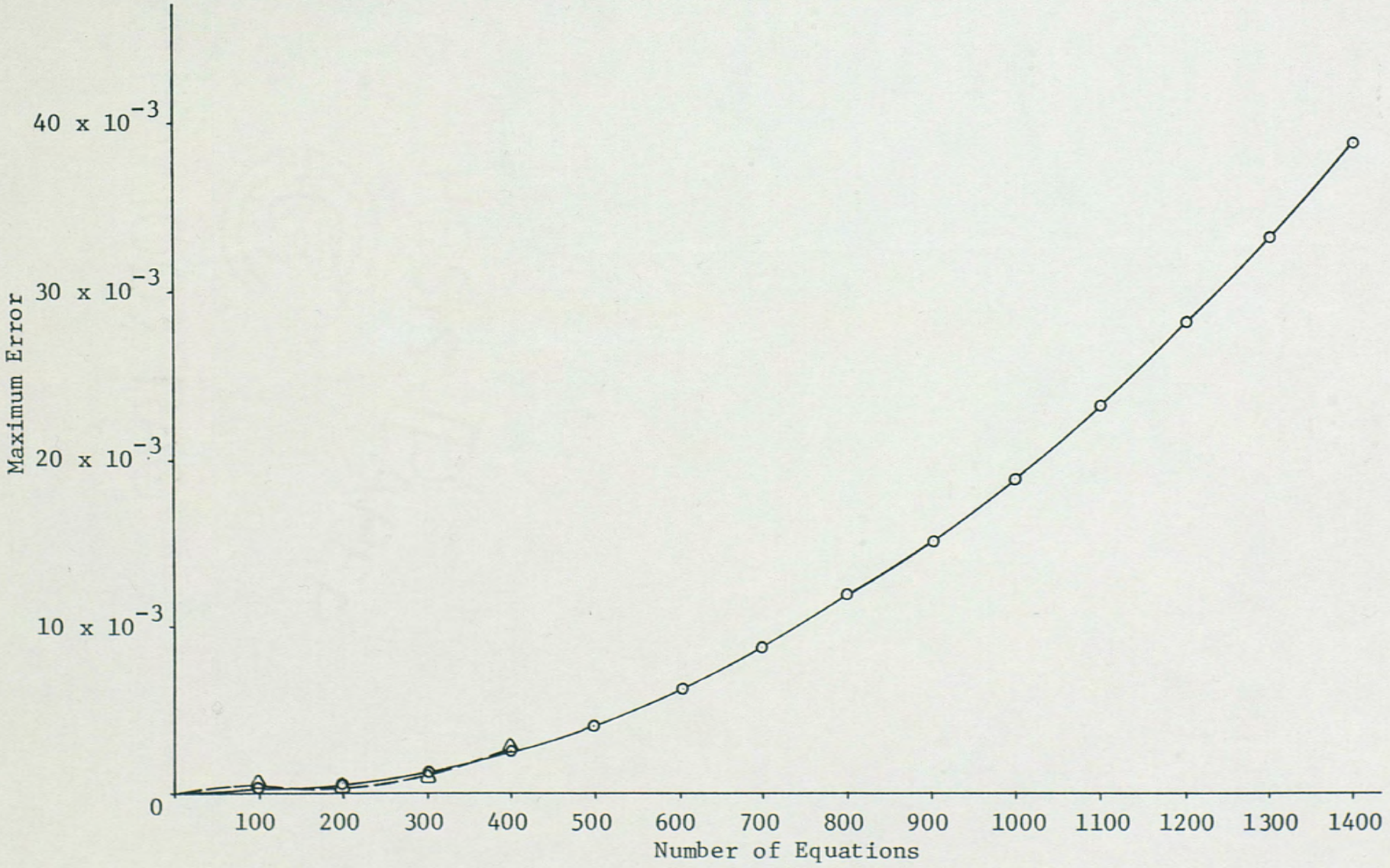


Fig. 10. Curve fit for maximum error.

This increases the shifting of data in and out of core, thus increasing the time of execution. Figure 11 shows this clearly. In general, the maximum possible depth of block results in minimum time of execution. However, if the total number of equations is not a multiple of the block depth selected, the last block will not be of the same size as other blocks and, therefore, will be filled with dummy zeros to maintain the same depth of block as others. Operation with too many of these dummy zeros increases the time of execution. Therefore, for solving 200 equations, a block depth of 175 will require greater execution time than a block depth of 100. Tables 6 and 7 and Figure 12 reflect this phenomenon.

TABLE 6

SHOWING EFFECT OF BLOCK DEPTH ON EXECUTION TIME

Half Bandwidth	Block Depth	Execution Time For 200 Equations Min:Secs
2	4	3:36
2	8	3:08
2	25	2:43
2	50	2:18
2	100	1:52
2	200	1:21
4	4	7:12
4	8	6:23
4	25	5:06
4	100	4:22
4	200	3:02

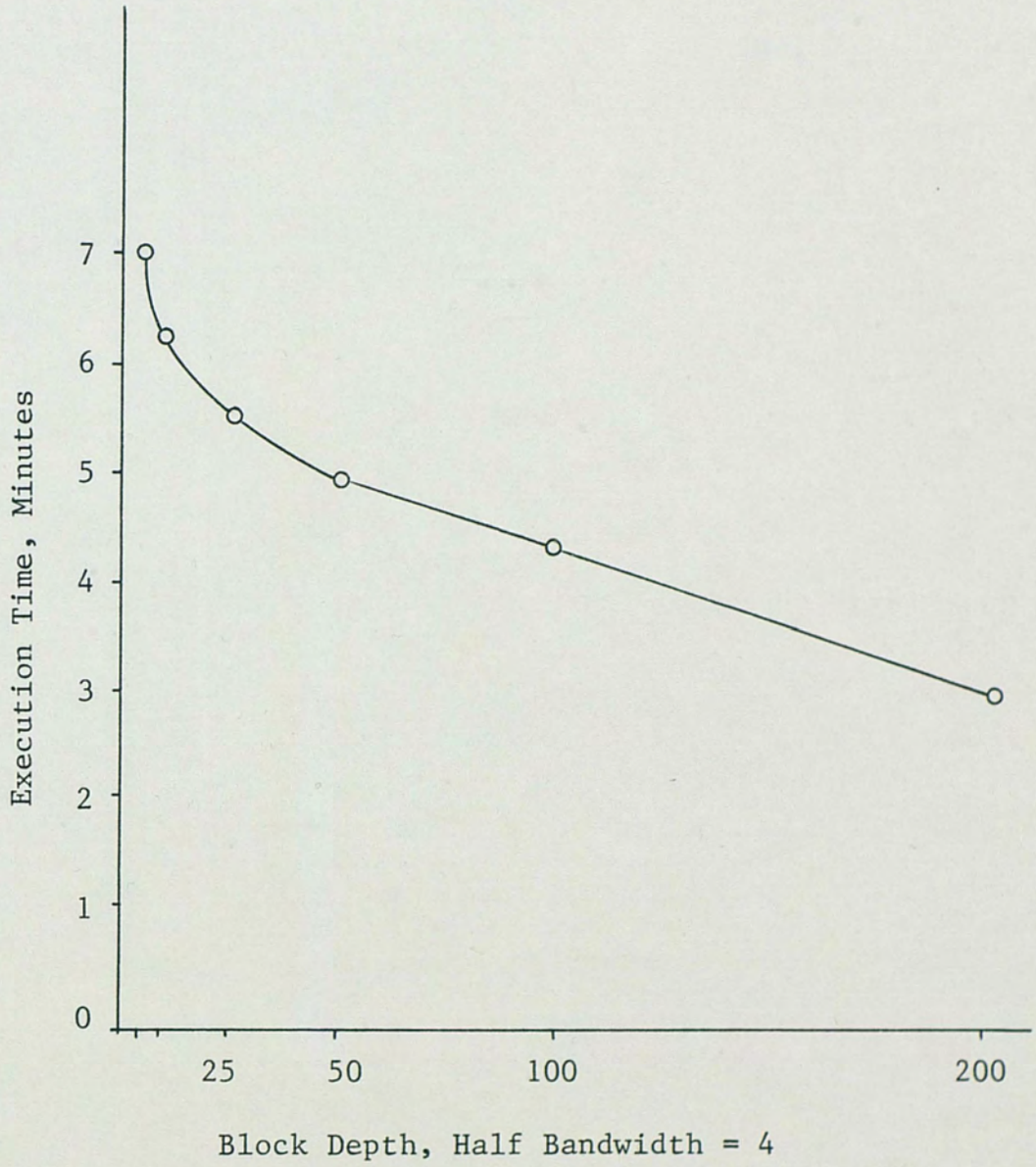


Fig. 11. Block depth versus execution time (no dummy zeros).

TABLE 7

SHOWING EFFECT OF DUMMY ZEROS ON EXECUTION TIME

Half Bandwidth	Block Depth	Number of Rows of Dummy Zeros	Execution Time for 200 Equations Min:Secs
4	100	0	4:22
	125	50	4:57
	150	100	5:44
	175	150	6:25
	197	194	7:16
	200	0	3:02

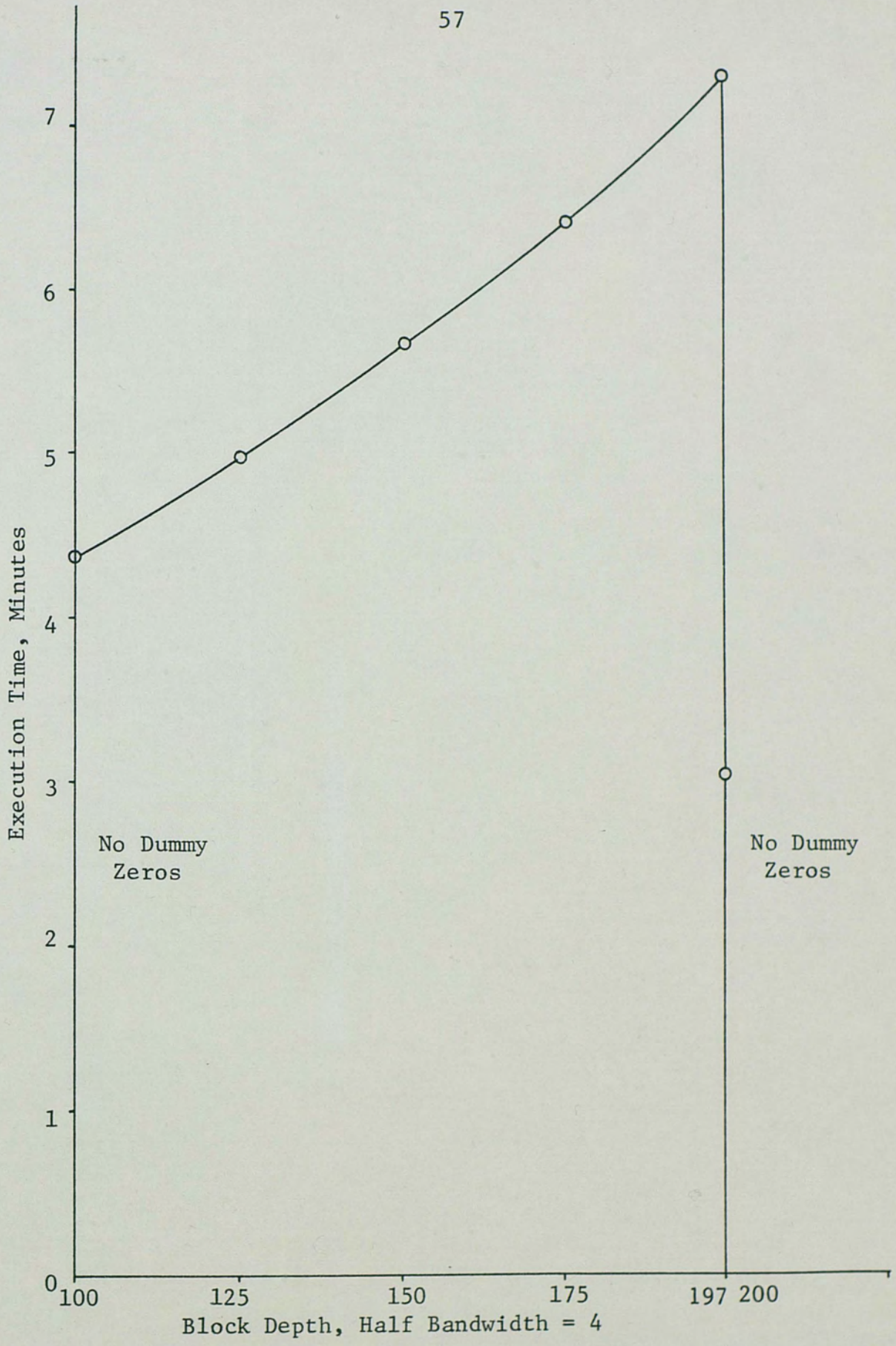


Fig. 12. Block depth versus execution time (with dummy zeros).

CHAPTER VII

TRUSS PROGRAM EXAMPLES

To demonstrate the feasibility of the out-of-core equation solver presented in Chapter VI in practical engineering applications, a number of truss analysis problems were solved with the help of a computer program developed by the author on the basis of the truss analysis program given by Brebbia and Ferrante (1). The program uses the stiffness method for assembling the system matrix. Initially, each element is regarded as isolated from the structure to establish a matrix equation defining the individual element behavior. Then, considering the intersection of each element with the remaining elements of the structure according to its connectivity, the total behavior of the structure is defined, leading to the problem solution.

The basic steps of this scheme are as shown below:

1. Numbering of nodes and elements for identification purposes and entering required data to the computer which includes the nodal coordinates, connectivity table, element properties, boundary and support conditions and a description of the applied loads,

2. Evaluation of the element matrix equations,

3. Assembling of the total system of equations,

$$\{P\} = [K]\{U\}$$

where:

$\{P\}$ = the total applied load vector

$[K]$ = the total stiffness matrix

$\{U\}$ = the total nodal displacements vector

4. Introduction of boundary conditions in the above mentioned equations,

5. Solution of the equation $\{P\} = [K]\{U\}$ to obtain the values for vector $\{U\}$,

6. Finding the forces in members from the equation; force in the element = [element rotation matrix] * {element nodal displacement vector}.

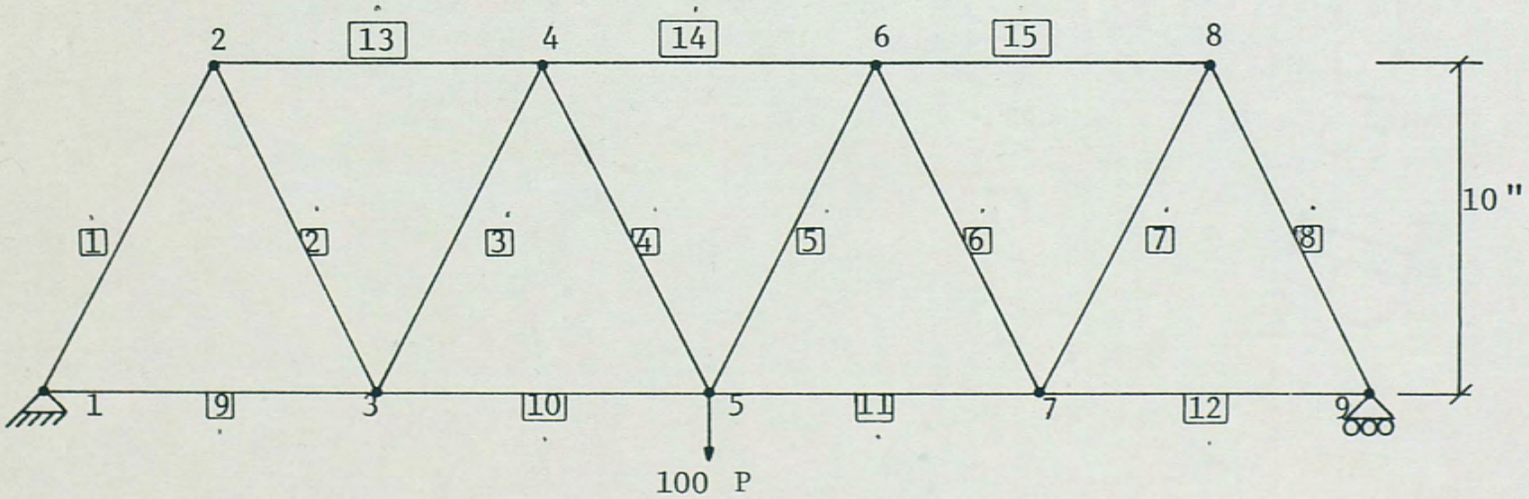
Two versions of this program were prepared. In the first version, the total stiffness matrix is assembled from the element stiffness matrices in the computer core; and the system equations is solved by the in-core symmetric banded scheme. The second version of this program assembles the total stiffness matrix on a disk file and the system of equations is solved by out-of-core blocked banded equation solver

The half bandwidth has a great effect on the efficiency of the system and on the total number of equations that can be solved. An increase in half bandwidth reduces both the efficiency of the system and the total number of equations that can be

solved. The half bandwidth, counted in nodes, is obtained by adding 1 to the greatest difference in number for the two nodes associated with a bar. It is, therefore, recommended that the nodal numbering should be selected, minimizing the differences in number of the two nodes connected to each bar.

The truss examples in Figures 13-15 have a half bandwidth of 6. Each element of these trusses has an area of .05 square inches and a modulus of elasticity of 30×10^6 pounds per square inch. Each truss has two support nodes; one support node is hinged, and the other node is supported on a roller. Each of these trusses was solved by the in-core and out-of-core versions of the truss analysis program. A comparison of the times of execution for these trusses is shown in Figure 16.

The 97 nodes and 191 elements truss shown in Figure 15 requires close to 10,000 bytes of computer core memory for the program, 10,752 bytes for disk operating system, 2048 bytes for computer operation, and 5 bytes for each term of the matrix. The storage requirement exceeds the computer core memory. Therefore, this problem cannot be solved by the in-core scheme on the system configuration used with a 32K byte computer core, but it can be solved by the out-of-core scheme with the same core capacity.

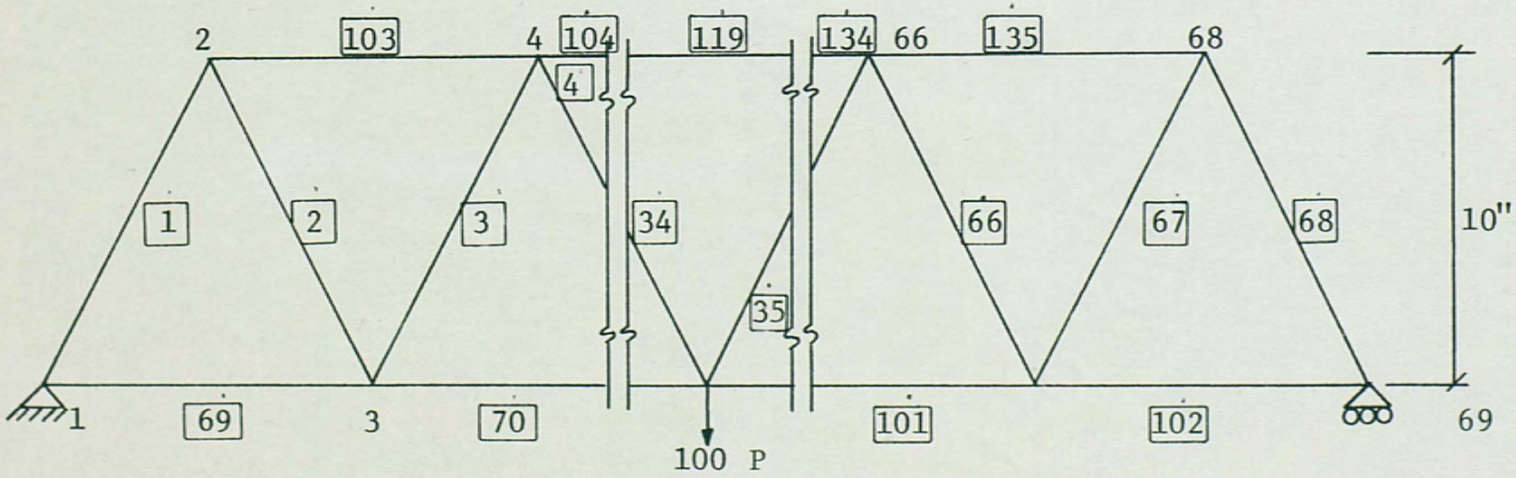


4 Bays at 10"; height = 10"

Area of each element = .05 square inch

E = 30 x 10^6 pounds per square inch

Fig. 13. Nine nodes and fifteen elements truss.

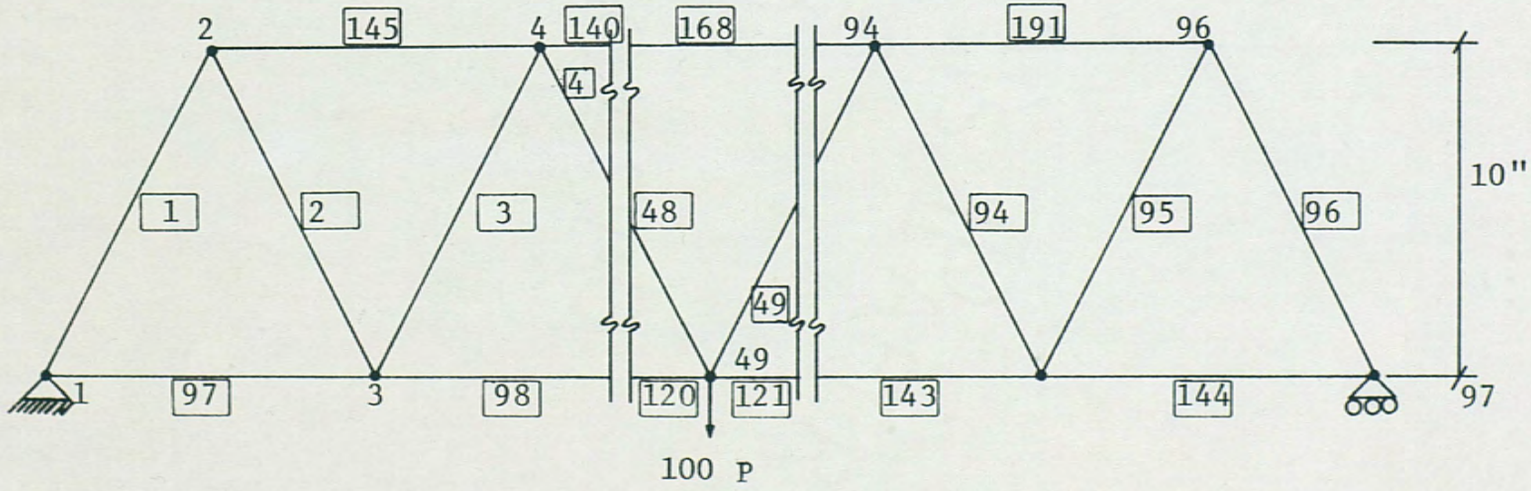


34 Bays at 10"; height = 10"

Area of each element = .05 square inch

E = 30 x 10^6 pounds per square inch

Fig. 14. Sixty-nine nodes and 135 elements truss.



48 Bays at 10"; height = 10"

Area of each element = .05 square inch

E = 30 x 10^6 pounds per square inch

Fig. 15. Ninety-seven nodes and 191 elements truss.

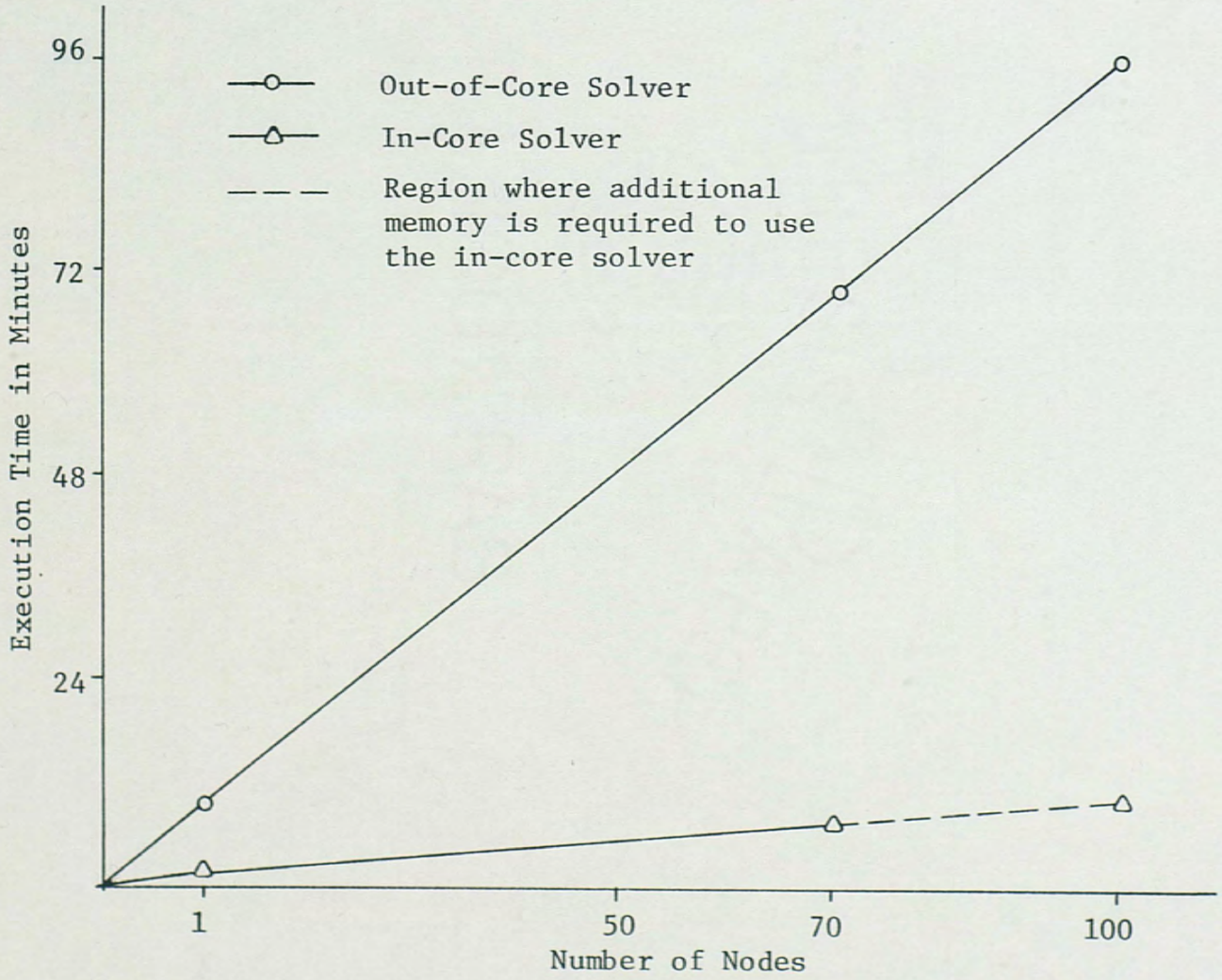


Fig. 16. Execution time versus number of nodes in a truss.

16K bytes of memory were added to the computer core of the system configuration used to solve the problem mentioned earlier, by the in-core scheme to check the results obtained by out-of-core methods. The results obtained by both methods were identical. Core storage requirements for stiffness matrix in out-of-core solver is reduced to one-third of that of in-core solver.

It can be seen from Figure 16 that the in-core version of the truss analysis program cannot be used for trusses with nodes greater than 70 and elements greater than 140, the other conditions in the program remaining are the same. The out-of-core version of truss analysis program can be used for trusses with nodes close to 100 and elements close to 200. The precision of results obtained by both methods is the same, but the time of execution for out-of-core solver is considerably greater.

CHAPTER VIII

CONCLUSIONS

The in-core banded equation solver is an efficient program for a microcomputer and can be used when the system of linear algebraic equations is small enough that the global stiffness matrix can be accommodated in the core of the computer. However, in engineering analysis, a much larger system of equations is usually encountered; and the in-core storage is not large enough to accommodate the data. In the equations derived by minimization or energy principles, one frequently encounters symmetric banded matrices which are usually well conditioned. In such cases, the out-of-core blocked banded program presented in this report is suitable. The ninety-seven node truss example presented in Chapter VII demonstrates this result using an Apple II computer. The tests in this report show the overall error measure and the maximum error computed by this program to be the same as for the in-core program, but the time taken for execution is many times greater, as shown in Chapter VI. These tests also confirm William T. Segui's observation that maximum possible block size results in maximum efficiency (25). However, as discussed in Chapter VI of this paper, this program requires each block to have the same depth; therefore, the maximum efficiency is obtained only when this maximum

depth of block is chosen so as not to fill up the last block with too many dummy zeros.

It was shown in Chapter VI of this report that whereas the block size is limited only by the core capacity of the computer, the total number of unknowns that can be solved is also limited by the capacity of the out-of-core peripheral device such as a disk drive. In conclusion, it can be said that the out-of-core program is suitable for a microcomputer which has a high speed disk or tape drive of large capacity and where time for execution is of no particular importance.

The research done for this report suggests several areas where further work might be done. First, the program used might be improved further. The current algorithm does not take variable bandwidth into account. Consideration of this possibility might decrease processing time. Also, because BASIC interpreters are relatively slow in operating, a compiled program in machine language will result in significant time saving. It should also be noted that the current algorithm does not test for singular or ill-conditioned matrices. If the matrix is singular, the program will reach an error condition and halt; but if the matrix is ill-conditioned, a false result might be obtained. Several techniques exist to test matrices for ill-conditioning, and incorporation of such a test in the program would increase its utility.

The other schemes discussed in this report provide a second area for further study. Cantin's scheme (18) for blocked banded

matrices and the gradient methods, particularly the conjugate gradient method show great promise. It would be interesting to implement these methods on a microcomputer and compare accuracy and execution time.

APPENDIX

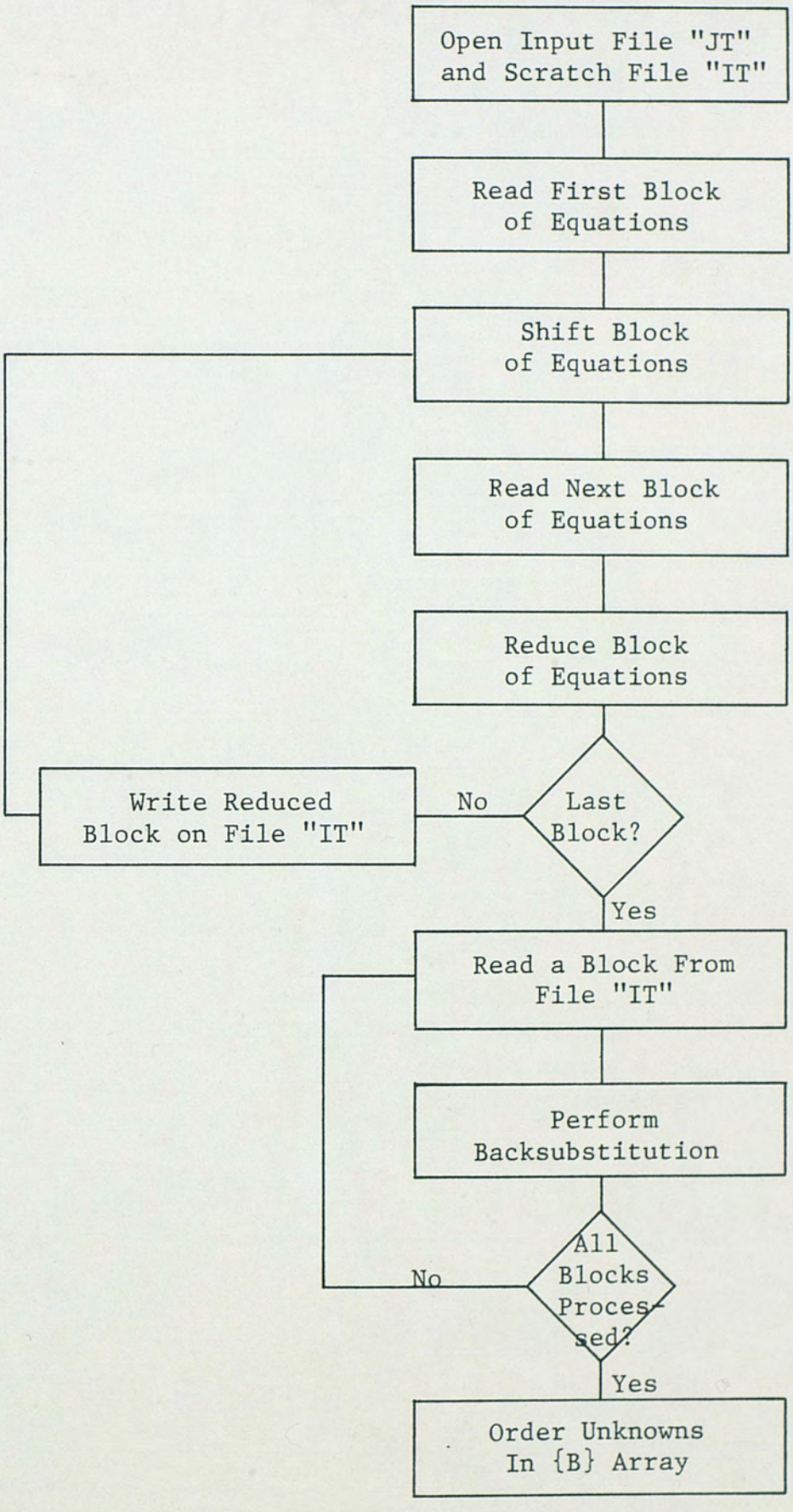


Fig. 17. Flow chart for out-of-core solver.

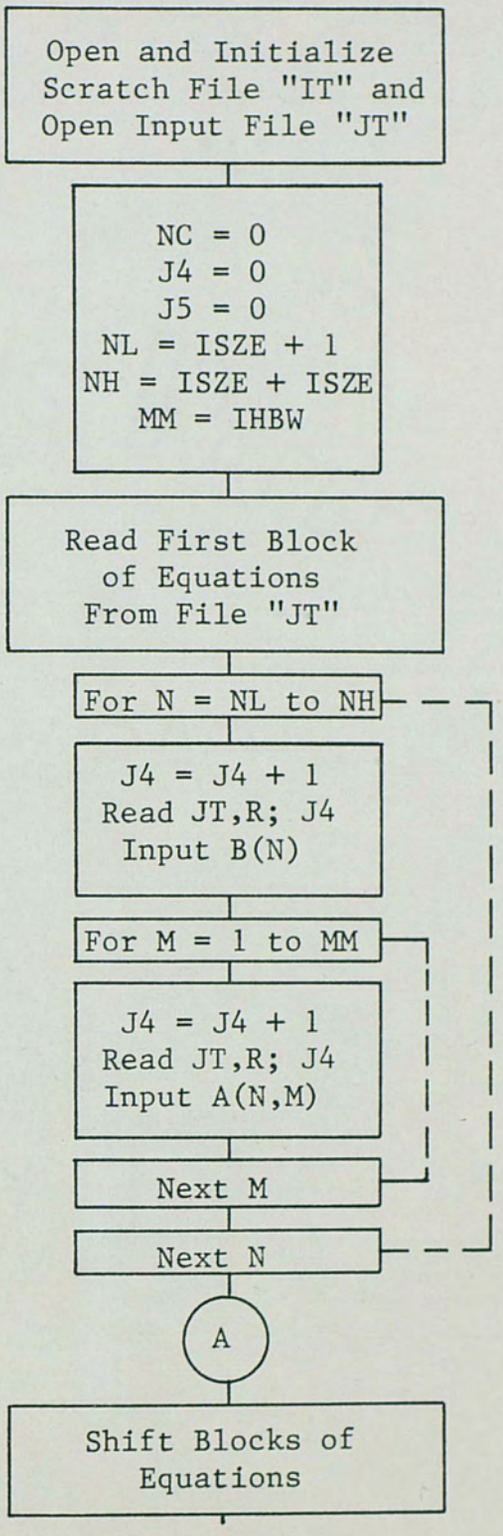


Fig. 18. Detailed flow chart for out-of-core solver.

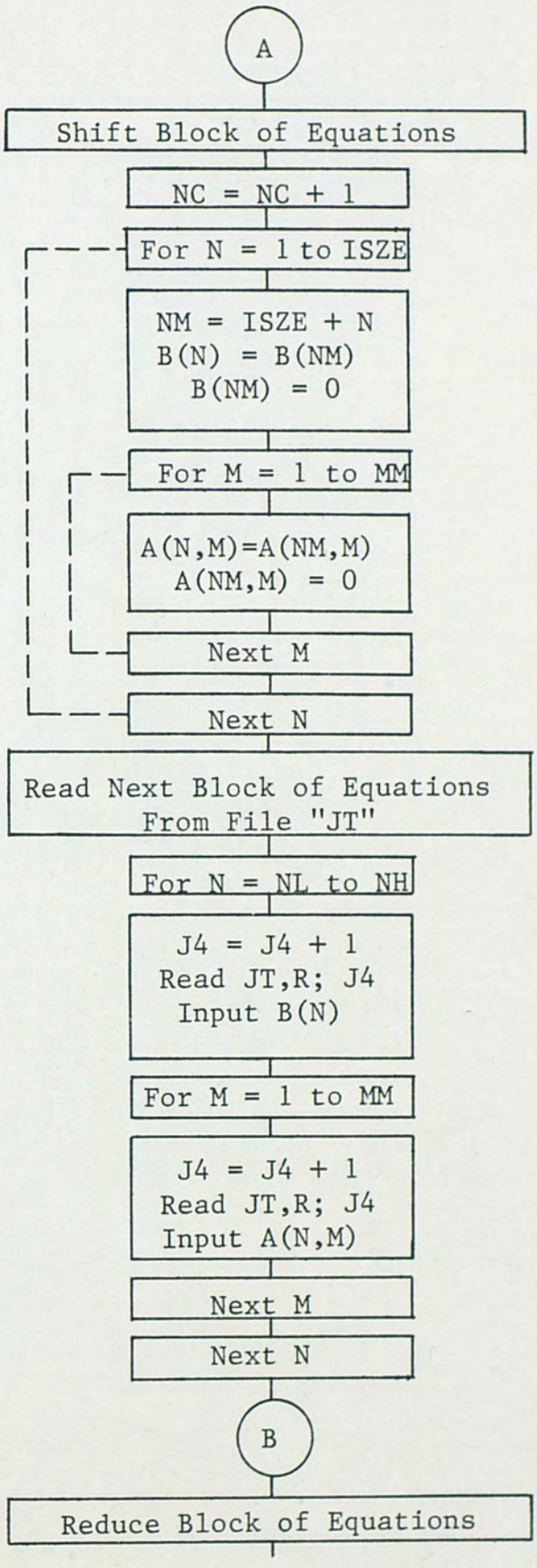


Fig. 18. Detailed flow chart for out-of-core solver (continued).

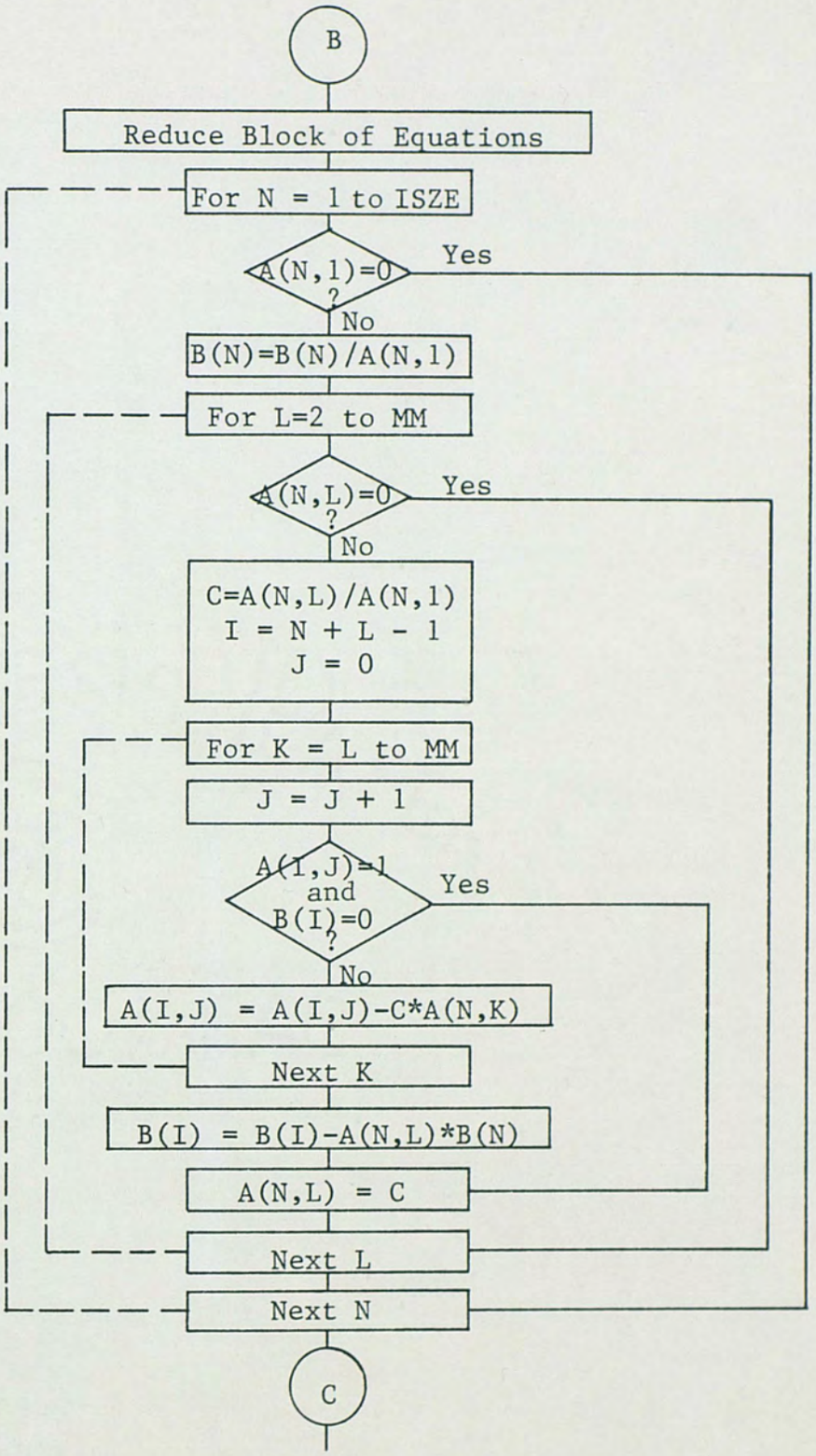


Fig. 18. Detailed flow chart for out-of-core solver (continued).

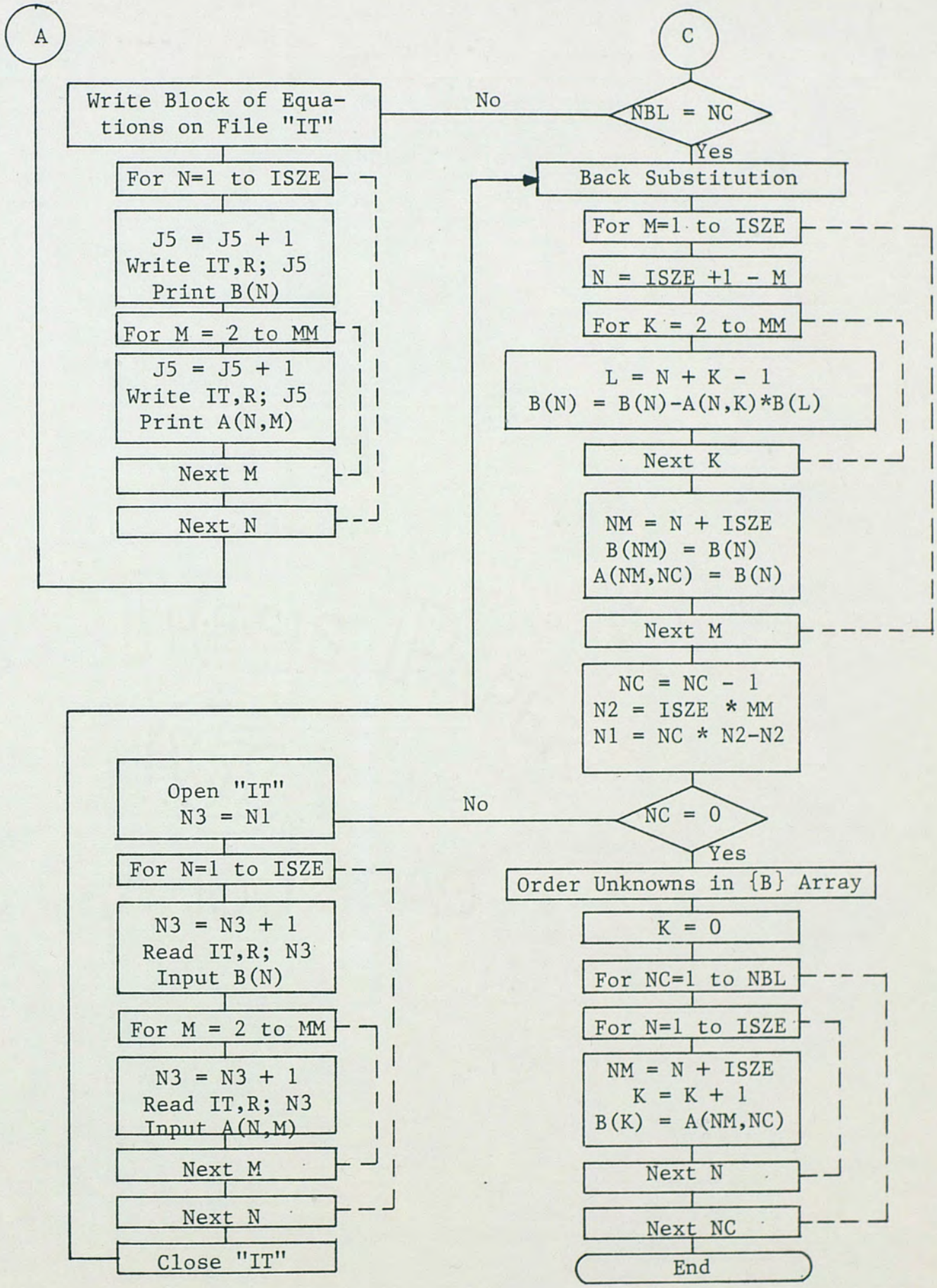


Fig. 18. Detailed flow chart for out-of-core solver (continued).

COMPUTER PROGRAM

```
2 REM:  THIS PROGRAM GENERATES A LARGE SET OF SIMULTANEOUS EQUA-
        TIONS AND SOLVES THESE EQUATIONS WITH THE HELP OF A BANDED
        BLOCKED EQUATION SOLVER SUB-ROUTINE.

3 REM:  'JT' IS A DISK FILE IN WHICH THE UPPER HALF BAND AND CON-
        STANT VECTORS ARE WRITTEN.  'IT' IS A DISK FILE USED AS A
        SCRATCH FILE.

4 REM:  B IS A SOLUTION VECTOR WITH DIMENSION GREATER THAN OR EQUAL
        TO THE PRODUCT OF THE BLOCK DEPTH AND THE NUMBER OF BLOCKS.

5 REM:  A IS A BLOCK OF COEFFICIENTS OF THE UPPER HALF BAND.  THE
        DEPTH OF EACH BLOCK MUST BE AT LEAST AS LARGE AS THE HALF
        BAND WIDTH BUT NO LARGER THAN HALF THE FIRST DIMENSION OF A.

6 REM:  THE SECOND DIMENSION OF A MUST BE EQUAL TO OR GREATER THAN
        THE HALF BANDWIDTH AND EQUAL TO OR GREATER THAN THE NUMBER
        OF BLOCKS.

7 REM:  MM = HALF BANDWIDTH OF THE SYSTEM BEING SOLVED.

8 DIM B(200),A(4,100)
10 NO = 200
20 IHBW = 2
30 ISZE = 2
40 D$ = CHR$ (4)
50 PRINT D$;"OPEN JT,L16"
60 PRINT D$;"DELETE JT"
70 PRINT D$;"OPEN JT,L16"
80 REM:  DETERMINE NUMBER OF BLOCKS NBL
100 BL = NO / ISZE
110 NBL = INT (BL)
120  IF BL > NBL THEN NBL = NBL + 1
130  PRINT " NUMBER OF BLOCKS=";NBL
140  REM:  DETERMINE TRUE DEPTH KBL OF THE LAST BLOCK
150 KBL = ISZE
160  IF (NBL * ISZE - NO) > 0 THEN KBL = NO - (NBL - 1) * ISZE
170  REM:  GENERATE UPPER HALF BAND AND RIGHT HAND SIDE VECTOR BY
           BLOCKS
180 J3 = 0
200  FOR J = 1 TO NBL
210  FOR K = 1 TO ISZE
220 B(K) = 0
230  FOR L = 1 TO IHBW
240 A(K,L) = 0
250  NEXT L
```

COMPUTER PROGRAM (Continued)

```
260  NEXT K
270  REM:  DETERMINE THE CURRENT BLOCK DEPTH KHBW
300 KHBW = ISZE
310  IF J = NBL THEN KHBW = KBL
320  FOR I = 1 to KHBW
330 A(I,1) = 2
340 A(I,2) =  -1
350  NEXT I
360  IF J < > NBL THEN 510
370  REM:  GENERATE THE LAST ROW OF COEFFICIENTS
400 B(KHBW) = 1
410 A(KHBW,1) = 1
420 A(KHBW,2) = 0
440  REM:  WRITE THE BLOCK OF COEFFICIENTS ON DISK FILE JT
510  FOR N = 1 TO ISZE
512 J3 = 1 + J3
515  PRINT D$;"WRITE JT,R";J3
520  PRINT B(N)
530  FOR M = 1 TO IHBW
532 J3 = J3 + 1
535  PRINT D$;"WRITE JT,R";J3
540  PRINT A(N,M)
550  NEXT M
560  NEXT N
570  NEXT J
580  PRINT D$;"CLOSE JT"
590  GOSUB 1000
600  PRINT D$;"BLOAD KSR80"
610  CALL 775
700  FOR I = 1 TO NO
710  PRINT I,B(I)
720  NEXT I
800  REM:  COMPUTE ERROR MEASURE
810 SUM = 0
820 AMAX = 0
830  FOR I = 1 TO NO
840 R = I - B(I)
850  IF ABS (R) > AMAX THEN AMAX = ABS (R)
860 SUM = SUM + R * R
865  NEXT I
870 ER1 = SQR (SUM) / NO
880  PRINT "ERROR MEASURE=";ER1
890  PRINT "MAXIMUM ERROR=";AMAX
900  PRINT D$;"PR#0"
950  END
1000  REM:  BANDED BLOCKED EQUATION SOLVER
```

COMPUTER PROGRAM (Continued)

```
1010 NL = ISZE + 1
1020 NH = ISZE + ISZE
1030  PRINT D$;"OPEN IT,L16"
1040  PRINT D$;"DELETE IT"
1050  PRINT D$;"OPEN IT,L16"
1060  PRINT D$;"OPEN JT,L16"
1070 NC = 0
1072 J4 = 0
1073 J5 = 0
1075 MM = IHBW
1080  GO TO 2020
1090  REM:  REDUCE EQUATIONS BY BLOCKS
1095  REM:  SHIFT BLOCKS OF EQUATIONS
1100 NC = NC + 1
1110  FOR N = 1 TO ISZE
1120 NM = ISZE + N
1130 B(N) = B(NM)
1140 B(NM) = 0
1150  FOR M = 1 TO MM
1160 A(N,M) = A(NM,M)
1170 A(NM,M) = 0
1180  NEXT M
1190  NEXT N
1195  REM:  READ NEXT BLOCK OF EQUATIONS INTO CORE
2000  IF NBL = NC THEN 2200
2020  FOR N = NL TO NH
2022 J4 =  J4 + 1
2030  PRINT D$;"READ JT,R";J4
2040  INPUT B(N)
2050  FOR M = 1 TO MM
2052 J4 = J4 + 1
2055  PRING D$;"READ JT,R";J4
2060  INPUT A(N,M)
2070  NEXT M
2080  NEXT N
2100  IF NC = 0 THEN 1100
2150  REM:  REDUCE BLOCK OF EQUATIONS
2200 FOR N = 1 TO ISZE
2210  IF A(N,1) = 0 THEN 2350
2220 B(N) = B(N) / A(N,1)
2230  FOR L = 2 TO MM
2240  IF A(N,L) = 0 THEN 2340
2250 C = A(N,L) / A(N,1)
2260 I = N + L - 1
2270 J = 0
```

COMPUTER PROGRAM (Continued)

```
2280  FOR K = L TO MM
2290 J = J + 1
2295  IF A(I,J) = 1 AND B(I) = 0 THEN 2330
2300 A(I,J) = A(I,J) - C * A(N,K)
2310  NEXT K
2320 B(I) = B(I) - A(N,L) * B(N)
2330 A(N,L) = C
2340  NEXT L
2350  NEXT N
2360  REM:  WRITE BLOCK OF REDUCED EQUATIONS ON DISK IT
2400  IF NBL = NC THEN 2485
2415  FOR N = 1 TO ISZE
2417 J5 = J5 + 1
2420  PRING D$;"WRITE IT,R";J5
2430  PRINT B(N)
2440  FOR M = 2 TO MM
2442 J5 = J5 + 1
2445  PRINT D$;"WRITE IT,R";J5
2450  PRINT A(N,M)
2460  NEXT M
2470  NEXT N
2480  GO TO 1100
2485  PRINT D$;"CLOSE IT"
2490  REM:  BACK SUBSTITUTION
2500  FOR M = 1 TO ISZE
2510 N = ISZE + 1 - M
2520  FOR K = 2 TO MM
2530  L = N + K - 1
2540  B(N) = B(N) - A(N,K)*B(L)
2550  NEXT K
2560 NM = N + ISZE
2570  B(NM) = B(N)
2580  A(NM,NC) = B(N)
2590  NEXT M
2600 NC = NC - 1
2603 N2 = ISZE * MM
2605 N1 = NC * N2 - N2
2610  IF NC = 0 THEN 2697
2620  PRINT D$;"OPEN IT,L16"
2625 N3 = N1
2630  FOR N = 1 TO ISZE
2632 N3 = N3 + 1
2635  PRINT D$;"READ IT,R";N3
2640  INPUT B(N)
2650  FOR M = 2 TO MM
```

COMPUTER PROGRAM (Continued)

```
2652 N3 = N3 + 1
2655  PRINT D$;"READ IT,R";N3
2660  INPUT A(N,M)
2670  NEXT M
2680  NEXT N
2690  PRINT D$;"CLOSE IT"
2695  GO TO 2500
2697  REM:  ORDER UNKNOWNS IN B ARRAY
2700 K = 0
2710  FOR NC = 1 TO NBL
2720  FOR N = 1 TO ISZE
2730 NM = N + ISZE
2740 K = K + 1
2750 B(K) = A(NM,NC)
2760  NEXT N
2770  NEXT NC
2780  RETURN

]PR#0
```

REFERENCES

1. Brebbia, C.A., and Ferrante, A.J. Computational Methods for the Solution of Engineering Problems. New York: Crane, Russak, 1978.

2. Huebner, Kenneth H. The Finite Element Method for Engineers. New York: John Wiley and Sons, 1975.

3. Norrie, D.H., and DeVries, G. An Introduction to Finite Element Analysis. New York: Academic Press, 1978.

4. Espinosa, Christopher. Apple II Reference Manual. Cupertino, CA: Apple Computer, Inc., 1979.

5. Applesoft II BASIC Programming Reference Manual. Cupertino, CA: Apple Computer, Inc., 1978.

6. Disk Operating System Instruction and Reference Manual 3.2 Version. Cupertino, CA: Apple Computer, Inc., 1979.

7. Poole, Lon; McNiff, Martin; and Cook, Steven. Apple II User's Guide. Berkeley, CA: Osborne/McGraw-Hill, 1981.

8. The Applesoft Tutorial. Cupertino, CA: Apple Computer, Inc., 1979.

9. Westlake, Joan P. A Handbook of Numerical Matrix Inversion and Solution of Linear Equations. New York: John Wiley and Sons, 1968.

10. Bathe, K., and Wilson, E.L. Numerical Methods in Finite Element Analysis. Englewood Cliffs, NJ: Prentice Hall, Inc., 1976.

11. Segui, William T. Computer Programs for the Solution of Linear Algebraic Equations. NASA Contractors Rep. CR-2173, January, 1973.

12. Fox, L. An Introduction to Numerical Linear Algebra. New York: Oxford University Press, 1964.

13. Evan, D.J. "The Analysis and Application of Sparse Matrix Algorithms in the Finite Element Method." In Mathematics of Finite Elements and Applications. Ed. by J.R. Whiteman. New York: Academic Press, 1973.

14. Fried, Issac. "A Gradient Computational Procedure for the Solution of Large Problems Arising from the Finite Element Discretization Method." Internat. J. Numer. Methods in Engrg. 2 (October-December 1970): 477-494.

15. Ko, Peter Y. "Conjugate Gradient for Finite Element Equations." In Proceedings of the Seventh Conference on Electronic Computation, August 1979, New York: ASCE, 1979.

16. Brooks, David F., and Brotton, Derik M. "Computer System for Analysis of Large Frameworks." J. Struct. Div. Am. Soc. Civ. Engrs. 93 (December 1964): 1-23.

17. Wilson, E.L. Analysis of Axisymmetric Solids. Computer Programming Series, University of California, 1967.

18. Cantin, Giles. "An Equation Solver of Very Large Capacity." Internat. J. Numer. Methods in Engrg. 3 (July-September 1971): 379-388.

19. Irons, Bruce M. "A Frontal Solution Program for Finite Element Analysis." Internat. J. Numer. Methods in Engrg. 2 (January-March 1970): 5-32.

20. Tang, Pin, and Rossettos, J.N. Finite Element Method. Cambridge, Mass.: MIT Press, 1977.

21. Beckers, P., and Sander, G. Proceedings of the Seventh Conference on Electronic Computation, August 1979. New York: ASCE, 1979.

22. Gregory, R.T., and Karney, D.L. A Collection of Matrices for Testing Computational Algorithms. New York: Wiley Interscience, 1969.

23. Norris, C.H.; Wilber, J.B.; and Utku, Senol. Elementary Structural Analysis. New York: McGraw-Hill, 1976.

24. Pool, Lon, and Borchers, Mary. Some Common BASIC Programs. Berkeley, CA: Osborne/McGraw-Hill, 1979.

25. Segui, William T. "Computer Programs for the Solution of Systems of Linear Algebraic Equations." Internat. J. Numer. Methods in Engrg. 7, No. 4 (1973): 479-490.