AD/A-016 375

NUMERICAL PERFORMANCE OF MATRIX INVERSION WITH BLOCK PIVOTING

GERALD G. BROWN

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

AUGUST 1975

NPS55Zr75081

NAVAL POSTGRADUATE SCHOOL

Monterey, California

NUMERICAL PERFORMANCE OF MATRIX INVERSION

WITH BLOCK PIVOTING

by

Gerald G. Brown

August 1975

Prepared for:
Naval Postgraduate School
Monterey, California 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Isham Linder
Superintendent

Jack R. Borsting
Provost

This work was partially sponsored by a grant from the Research Foundation, Naval Postgraduate School.

Prepared by:

Gerald G. Brown
Department of Operations Research
and Administrative Sciences

Reviewed by:

David A. Schrady, Chairman
Department of Operations Research
and Administrative Sciences

Released by:

Robert R. Fossum
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS55Zr75081	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Numerical Performance of Matrix Inversion with Block Pivoting		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gerald G. Brown		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61152N; RR 000-01-10 N0001475WR50001
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE August 1975 13. NUMBER OF PAGES 12
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

PRICES SUBJECT TO CHANGE

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Large Scale Mathematical Programming
Large Scale Linear Programming
Factorization Methods in Optimization
Paged Memory Arithmetic
Page Processing
Virtual Memory Systems
Matrix Storage Allocation
Numerical Algorithm Performance

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An experiment with matrix inversion using block pivots is presented. Large scale matrix computations can often be performed more efficiently by use of partitioning. Such matrix manipulation lends itself to paged or cache memory systems since computation is staged to be completely performed in local blocks of controllable size. On other systems retrieval overhead can be balanced with computation for "in-memory/out-of-memory" applications. Parallelism in such schema leads to efficient utilization of some multiple

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. Matrix Inversion
Inversion Algorithms
Partitioned Matrices

20. processor environments. Timing results indicate, however, that choice of block size should not necessarily be dictated by hardware page size for most efficient operation and that classical methods of estimating computation times are not always adequate.

1. INTRODUCTION

This paper reports a numerical experiment with an approach to inversion of a real matrix using a block partitioning structure. The study arises in the context of design of a large scale mathematical programming system for use within various computer environments. The scheme permits controlled explicit pagination of mathematical operations to coincide with boundaries specified by hardware memory management. The timing results presented indicate that maximizing work-per-page does not necessarily minimize total execution time as folklore would advise. Further, performance of an inversion scheme such as this is not always adequately estimated by classical means. The implications even for the simple cases reported here are potentially of importance to our further design work.

Obtaining the explicit numerical inverse of a real matrix presents a classic problem in numerical methods which has received intense study in the literature; the wide spectrum of applications requiring inversion of matrices bears testimony to its fundamental nature. Numerous algorithms for matrix inversion have been presented and analyzed for error and speed [8,12,31] as have methods for error reduction by pivot selection and scaling [1,9], iterative improvement of accuracy [33], and exploitation of special features in the matrix to be inverted, especially sparseness [7,22,23,24,28,30,34,35], symmetry and band structure [26]. Studies of the form and complexity of the inversion process have included graph theoretic descriptions [4,28], statistical characterizations [19], exploitation of algorithmic parallelism [20], exercising special features of multiprogramming environments [17], and so forth.

One of the active application areas for numerical matrix inversion has been in large scale mathematical programming. Most successful codes for large problems must resort to some form of inversion (and often, reinversion) technique based either on special structure in the problem recognized a priori, or on special storage mechanisms for the inverse matrix. The motivation for these efforts is that the available main memory on modern digital computers is not sufficiently large to store the inverse explicitly for problems of contemporary scale (say, thousands of rows).

Unfortunately, many methods based on a priori structure in problems are inherently ad hoc in nature - decomposition methods [2,6,11,25] are considered by many to exemplify this shortcoming. On the other hand, for some important classes of problems sharing a common special structure, new techniques developed in concert with new data structures for problem representation have led to remarkable breakthroughs. In fact, sometimes a triangulated basis is superior to an explicit inverse. As an example, a primal (simplex) network code [3] has triangulated node-arc incident matrices of rank 10,000 in 20 seconds (IBM 360/67) with no rounding error. Other successful special methods have included element generation by generalized upper bounding [5], factorization [15], and other compact working inverse basis methods [14].

Special storage mechanisms for the inverse have typically included columnwise product form representation (with "ETA" columns) [18] and modifications attempting to maintain sparcity in the inverse [13]. Also, inversion and manipulation of sparce matrices is possible based on other structural decompositions, including doubly linked lists, bit maps, etc. [22,29].

The introduction of computers with memory organized in pages which are transferred in bulk swapping transactions between high speed magnetic devices

and main memory has led many to conclude that the vastly increased virtual memory capacity will allow design of large scale algorithms ignorant of main, or cache, memory configurations. Unfortunately, algorithms designed originally for conventional systems have performed very poorly in the paged environment. The study of dynamic program behavior within paging systems indicates that the global cost of page swaps is far from negligible. Thus, with a given amount of work to perform in inverting a matrix, design criteria now include consideration of both minimizing page swaps and maximizing the work performed on each page during its cache residence [16, 17].

This paper presents an approach to inverting a real matrix using a partitioning structure and block pivots ("B pivots") which perform bulk inversions of interior submatrices [21,27]. The technique is demonstrated to have theoretical computational efficiency in terms of long (floating point) operations equal to classical methods, and parametric high resolution timing of a FORTRAN routine executed on a dedicated processor with blocks of varied size is given, with some discussion of surprising behavior.

This scheme employs explicit pagination of the mathematical operations in inversion and is potentially useful for large scale mathematical programming. The reason that the empirical study concentrates on a dedicated processor is two-fold. For contemporary large scale programming applications, the wall clock time, rather than central processor active compute time, often dominates the attention of the user - these applications are capable of laying seige to the entire computer system, whether paged or not. Second, we are interested in the performance characteristics of various numerical techniques imbedded in a large scale mathematical programming code now under development. The design of the new system, based on parametric studies such as this, includes

the provision for robust performance within varied computer environments.

Similarly, the examples are chosen to be full rank and dense in the interests of emphasizing the execution performance of the partitioned algorithm, rather than that of exercising some supporting storage structure, list mechanism, or other feature not under study here. Even for large, sparce problems, however, nonzero elements can often be aggregated to dense blocks over subsets of the matrix rows and columns.

The advent of minicomputers, micro computers, and various multiple processor configurations portends further application of B-pivot inversion even for relatively small or intermediate scale problems.

2. INVERSION TRANSFORMATION

The classical inversion method used here is in-situ Gauss-Jordan pivoting, chosen for speed, generality, and storage economy. To illustrate, let A be the nonsingular real square matrix of rank n to be inverted, with a_{ij} a general element of A. An in-situ scaler pivot on $a_{k\ell}$ is defined as follows for $a_{k\ell} \neq 0$ with "$\leftarrow$" indicating elemental replacement:

$$a_{k\ell} \leftarrow 1 \,/\, a_{k\ell} \quad \text{(pivot element)} \tag{1}$$

$$a_{kj} \leftarrow a_{kj} \,/\, a_{k\ell};\ j=1,..,n;\ j\neq\ell \quad \text{(pivot row)}$$

$$a_{i\ell} \leftarrow -a_{i\ell} \,/\, a_{k\ell};\ i=1,..,n;\ i\neq k \quad \text{(pivot column)}$$

$$a_{ij} \leftarrow a_{ij} - a_{i\ell}\, a_{kj} \,/\, a_{k\ell};\ i=1,..,n;\ i\neq k \quad j=1,..,n;\ j\neq\ell \quad \text{(general elements)}$$

An inverse is developed in place by performing n sequential pivotal transformations of this type on the main diagonal elements a_{pp}, $p=1,..,n$. However, since the accuracy of the computations can be adversely affected by the relative magnitude of the elements in A, and since the transformation is not defined for $a_{k\ell} = 0$, a pivot selection strategy is usually employed in inversion.

Notably, "partial" pivots use sequential rows for k and in each select ℓ by finding the largest absolute element in a previously unused column. "Full" pivots select the largest absolute matrix element remaining in an unused row and column. Often equilibration, or scaling, of the matrix columns for partial pivoting, and of rows and columns for full pivoting can further increase accuracy.

Both partial and full pivot strategies require that the inverse be recovered by recording the coordinates for each pivot and using this history to permute rows and columns of the result matrix left by pivoting. If $r_p = k$ and $c_p = \ell$ for pivot p, $p=1,..,n$, then inverse row c_i is located in matrix row r_i, and inverse column r_j is located in matrix column c_j. Thus, the increased accuracy of these pivot selection schemes comes at the cost of extra scanning of elements in pivot selection tournaments, bookkeeping and in reordering the resulting inverse.

A simple computation timing estimate for Gauss-Jordan pivoting can be arrived at by concentrating only on the floating point arithmetic required by (1).[33] Assuming an efficient program utilizing partial results in the pivot rows, or columns, in computing the general elements, a total of n pivots will be performed, each requiring $1 + 2(n - 1)^2$ multiplications and $(n - 1)^2$ additions. Aggregating operations for each pivot gives the total

$$W(n) = 2n^2 - 2n + 1 , \tag{2}$$

and a complete inversion time of $n\,W(n)$. For simplicity the effect of pivot selection strategy and other program overhead has been ignored.

Now let us introduce B-pivots to the inversion by partitioning A into submatrices as follows:

$$A = \begin{array}{|c:c|}\hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline \end{array}\ \begin{matrix} b \\ \\ \end{matrix} \qquad \begin{matrix} b & n \end{matrix} \tag{3}$$

If A_{11} is $b \times b$, an in-situ B-pivot on A_{11} is defined as follows for nonsingular A_{11}:

$$A_{11} \leftarrow A_{11}^{-1} \qquad \text{(B-pivot block)} \tag{4}$$

$$A_{12} \leftarrow A_{11}^{-1} A_{12} \qquad \text{(B-pivot row)}$$

$$A_{21} \leftarrow -A_{21} A_{11}^{-1} \qquad \text{(B-pivot column)}$$

$$A_{22} \leftarrow A_{22} - A_{21} A_{11}^{-1} A_{12} \qquad \text{(general elements)}$$

Since A_{11}^{-1} is found by using the Gauss-Jordan transformations in the first b rows and columns, it is clear that if we continue with another B-pivot in the next n - b rows and columns of A_{22}, A^{-1} will have been formed. The resulting inverse in terms of the original elements in (3) is:

$$A^{-1} = \begin{array}{|c|c|} \hline A_{11}^{-1} + A_{11}^{-1} A_{12} B A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} B \\ \hline -B A_{21} A_{11}^{-1} & B \\ \hline \end{array} \begin{matrix} b \\ \\ \\ n \end{matrix} \quad ,$$

$$\begin{matrix} \\ \\ b \end{matrix}$$

with

$$B = (A_{22} - A_{21} A_{11}^{-1} A_{12})^{-1} \quad ,$$

and may be verified by multiplication.

This B-pivot process may be continued analytically to any number P of B-pivots of varying size b_p, p=1,..,P, indeed, with $b_1=\ldots=b_p = 1$, and P = n, or with $b_1 = n$, and P = 1, the process is simply the Gauss-Jordan transformation.

A computation timing estimate using the simple approach producing (2) for a B-pivot of size b will give

$$b\,W(b) = 2b^3 - 2b^2 + b$$

operations for the inversion of the pivot block in (4), for the B-pivot row 2 (n - b) (2b - 1) operations, an equal number in the B-pivot column, and

$(n - b)^2 (2b - 1) + (n - b)^2$ floating point computations for the general elements. This again sssumes an efficient program which uses partial results in the B-pivot row, or column, to compute general elements. For the complete B-pivot we have the total

$$b (2n^2 - 2n + 1) = bW(n) . \qquad (5)$$

Thus, the B-pivot approach nominally requires no more computational effort, stated in terms of floating point operations, than scalar pivots.

3. APPLICATION SCHEME

Let $b_1 = b_2 = \ldots = b_{p-1}$, $b_p = n - \sum_{p=1}^{p-1} b_p$. This "fixed block" strategy requires nonsingular $A_{11}, \ldots, A_{pp}$. Applying this method to test matrices of dimensionality $n = 10(10)50$ with $b_p = 1(1)n$ in a FORTRAN test program produced the timing results shown in Figures 1 and 2 when run on a dedicated IBM 360/67. The times shown are for active computation with memory resident matrices and are precise to four significant digits. As indicated, both partial and full pivot selection strategies within pivot blocks were tested.

The FORTRAN program specifies a matrix of appropriate dimensionality, the block size to be used, and then exercises subroutines to perform the block inversions, row block transformations, column block transformations, and general block transformations. The program is compiled and optimized for run time efficiency by the IBM FORTRAN (H) compiler.

The timing prediction of (5) appears to hold at least as a polynomial form for the full block $(b_p = n)$ execution times in Figure 1. However, note the surprising gain in speed for intermediate sizes of b in Figure 2. This is partially due to the work avoided in restricting to the pivot blocks the range of scalar pivot selection and subsequent matrix permutations.

Conversely, for very small B-pivots and those requiring a last B-pivot much smaller than b, an "odd B-pivot", the housekeeping overhead of the program increases the execution time significantly. Thus, it may be advantageous to choose block sizes which partition the problem into blocks of homogeneous size, rather than to fill a page.

For these examples, the classical assumption that long floating point operations required by an algorithm dominate performance on a computer appears to be contradicted. It is certainly true that the time required for floating point operations relative to others such as register loads, compares, and so forth, has decreased greatly from the days of software arithmetic subroutines to contemporary floating point hardware (such as on the IBM 360/67 used here).

As a practical matter, resident memory for three blocks is required for each complete B-pivot as shown in (4). The program can easily be modified to emulate hardware delays caused by page swaps, or other interference by a paging mechanism. The times reported do not reflect such modifications.

4. DISCUSSION

For matrix inversion and other matrix arithmetic the "fixed block" approach allows b to be chosen to fit within a page, or cache area, and minimize boundary violations by the computation. However, the folklore which specifies that each page should be as nearly filled as possible is not necessarily true. The problem dimension (or other consideration) can in some cases dictate a less than full page scheme for fastest (or best) execution.

It is important to note that the block approach generates, a priori, demands for the matrix blocks from the out-of-memory device. Thus, an

agenda of input/output operations can be used to achieve simultaneity with computation. This can be much faster than a memory page fault and interrupt mechanism. Even in traditional single user environments the block size may be chosen to balance access-transfer time for slow magnetic devices with computation time, minimizing retrieval overhead with little memory cost. Note from Figures 1 and 2 that even a relatively small block pivot requires sufficient computation time (e.g., more than one second for $n = 40$) to permit simultaneous access to a disc or drum device.

The fixed blocks provide a convenient parcelling of matrix manipulation effort as well as storage, and permit for many applications a more convenient access structure than traditional column by column methods. This is useful for large mathematical programming problems with block angular, staircase, or other special block structure, [14]. These B-pivots on fixed blocks may also be used to equitably distribute computation among available parallel processors in sizeable bulk to permit relatively lengthy independent operation.

5. CONCLUSION

The usefulness and speed of B-pivot schemes come with one important disadvantage: the pivot blocks must be nonsingular mathematically and in the stricter numerical sense. Since scalar pivot selection is restricted to each pivot block, one should try to assemble the matrix A accordingly. For instance, in large linear programming problems a history of algorithm progress can be used to provide $A_{11},\ldots,A_{pp}$. In fact, B-pivots may be considered in this context as an extension of the concept of product form inverse. The B-pivot method works well for such problems, providing a

convenient tool for fast inversion, or reinversion, of kernels embedded in technological coefficients.

Other classes of matrices exhibit a natural diagonal dominance permitting effective B-pivot inversion. For instance, the noise covariance matrix has been proposed for such use in [21], which includes an extensive group theoretic characterization of admissability. Also, algorithms have been proposed for rearranging matrices to block angular form [32].

The accuracy of B-pivots may be improved by computing inner products with extended precision and special ordered addition. In some special cases of A, such as zero-one or sparse conditions as

$$A = \begin{array}{|c|c|} \hline A_{11} & \underline{0} \\ \hline A_{21} & A_{22} \\ \hline \end{array}$$

significant effort can be avoided by the B-pivot approach coupled with appropriate modifications of program logic. Several approaches to avoiding B-pivot singularity failures have been suggested, including "dynamic" blocks, "gang" blocks, pivot block selection procedure, and a matrix construction to be performed concurrently with a scaling algorithm as in [9]. Fortunately, none has been necessary in applications to date.

Continuing research focuses on block triangulation and dynamic factorization schemes for large scale mathematical programming, combined with generalized upper bounding in a hybrid system with both explicit and logically generated elements. It is becoming increasingly clear from experiments such as this that the (logical) algorithm and data representation of the program, and the (physical) organization of computations performed on the host computer under operating system control interact in subtle ways to give aggregate performance which is often counter-intuitive and seldom improved by ignoring the

details of either. The effect can be so pronounced that we are reexamining several classical techniques in this new light.

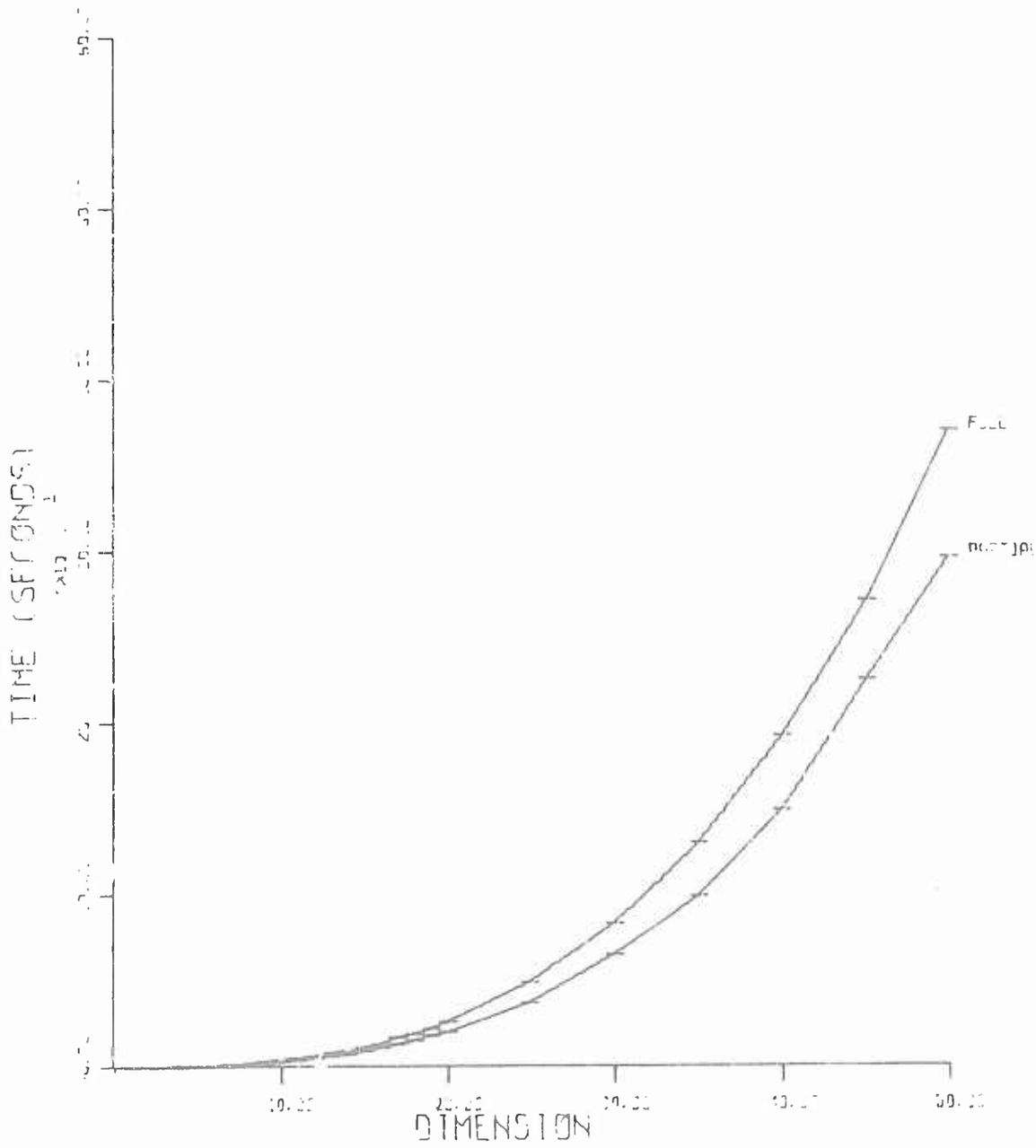

Figure 1 - Execution times for maximal "fixed block" pivots, $b_r = n$

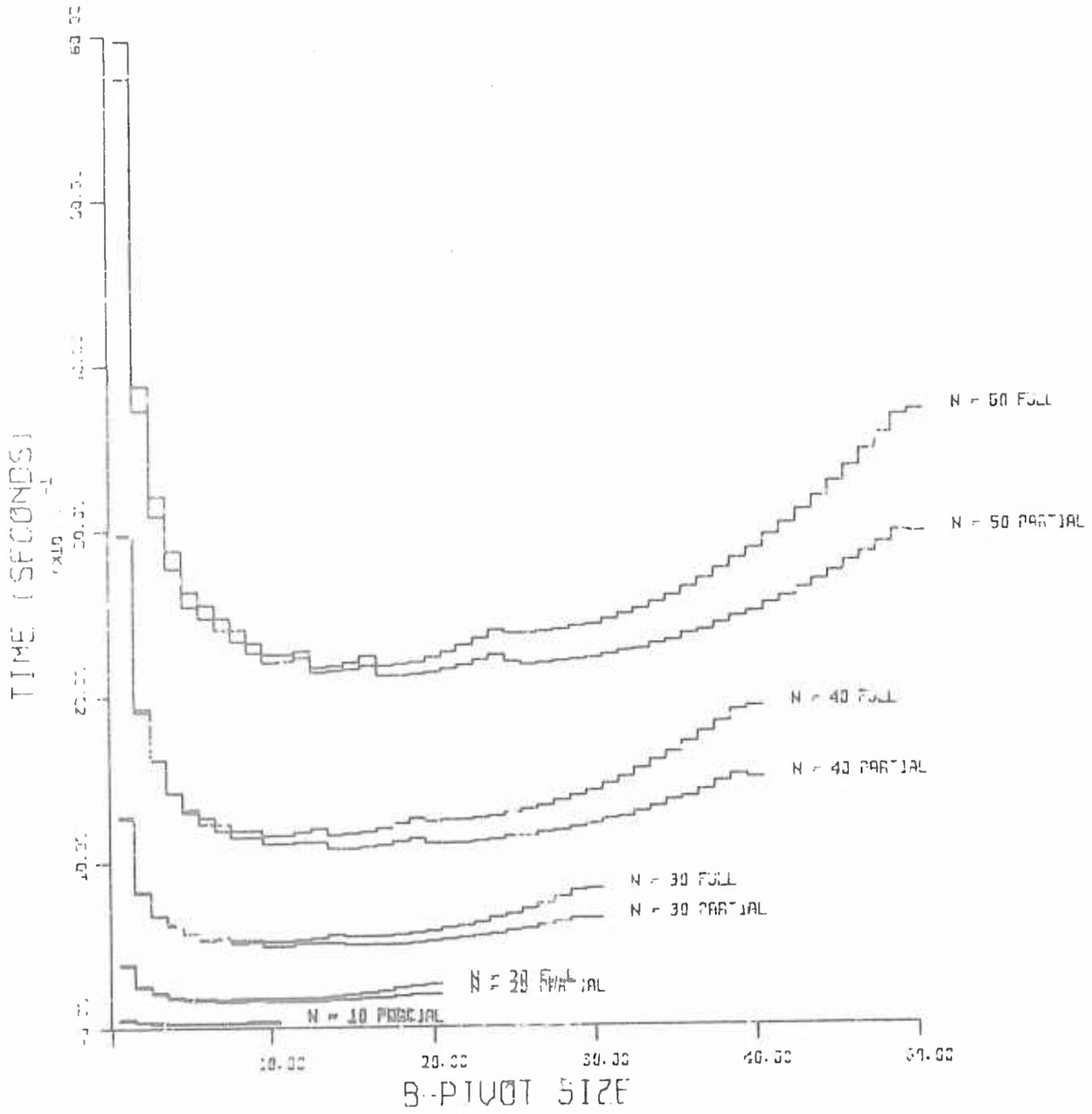

FIGURE 2 - Execution times for "fixed block" B-pivots

6. REFERENCES

1. Bauer, F., "Optimally Scaled Matrices," Numerische Math 5 (1963), 73-87.

2. Benders, J., "Partitioning Procedures for Solving Mixed Variables Programming Problems," Numerische Math 4, 1962, 238-252.

3. Bradley, G., Brown, G. and Graves, G., "A Comparison of Storage Structures for Primal Network Codes," ORSA/TIMS Meeting, Chicago, April 1975.

4. Chen, W., "The Inversion of Matrices by Flow Graphs," SIAM J. 12, 3(September 1964), 676-685.

5. Danzig, G. and Van Slyke, R., "Generalized Upper Bounding Techniques," J. Computer System Sci. 1, 1967, 213-226.

6. Danzig, G. and Wolfe, P., "Decomposition Principle for Linear Programs," Operations Research 8, 1960, 101-111.

7. Dulmage, A. and Mendelsohn, N., "On the Inversion of Sparce Matrices," Math. Comp. 16, 1962, 494-496.

8. Fadeeva, V., Computational Methods of Linear Algebra. Dover, New York 1959.

9. Fulkerson, D. and Wolfe, P., "An Algorithm for Scaling Matrices," SIAM Review 4, 2 (April 1962), 142-146.

10. Graves, G. and McBride, R., "Factorization in Large Scale Linear Programming," to appear in Mathematical Programming.

11. Graves, R. and Wolfe, P., Recent Advances in Mathematical Programming, McGraw-Hill, New York, 1963.

12. Householder, A., The Theory of Matrices in Numerical Analysis. Blaisdell, New York, 1964.

13. Larsen, L., "A Modified Inversion Procedure for Product Form of the Inverse Linear Programming Codes," Communications of ACM 7, 1962.

14. Lasdon, L., Optimization Theory for Large Systems, Macmillan, New York, 1970.

15. McBride, R., "Factorization in Large-Scale Linear Programming," UCLA/WMSI Report 200, June 1973.

16. McKellar, A. and Coffman, F., "Organizing Matrices and Matrix Operations for Paged Memory Systems," Communications of ACM 14, 3 (March 1969), 153-165.

17. Moler, C., "Matrix Computations with Fortran and Paging," Communications of ACM 4(April 1972), 268-270

18. Orchard-Hays, W., Advanced Linear Programming Techniques. McGraw-Hill, New York, 1968.

19. Oswald, F., "Matrix Inversion by Monte Carlo Methods," in Ralston, A. and Wilf, H., Mathematical Methods for Digital Computers, Wiley, New York, 1960. 78-83.

20. Pease, M., "Matrix Inversion Using Parallel Processing," J. of ACM 14, 4(October 1967), 757-764.

21. Pease, M., "Inversion of Matrices by Partitioning," J. of ACM 16, 2(April 1969), 302-314.

22. Pooch, U. and Nieder, A., "A Survey of Indexing Techniques for Sparce Matrices," ACM Computing Surveys 5, 2(June 1973), 109-133.

23. Reid, J. (editor), Large Sparce Sets of Linear Equations. Academic Press, New York, 1971.

24. Rose, D. and Willoughby, R. (editors), Sparce Matrices and Their Applications. Plenum Press, New York, 1972.

25. Rosen, J., "Primal Partition Programming for Block Diagonal Matrices," Numerische Math. 6, 1964, 250-260.

26. Schwartz, H., Rutishauser, H. and Stiefel, E., Numerical Analysis of Symmetric Matrices. Prentice-Hall, Englewood Cliffs, New Jersey, 1973

27. Swift, G., "A Comment on Matrix Inversion by Partition," SIAM Review 2, 2(April 1960), 132-133.

28. Tewarson, R., "The Product Form of Inverses of Sparce Matrices and Graph Theory," SIAM Review 9, 1(January 1967), 91-99.

29. Tewarson, R., "Row Column Permutation of Sparce Matrices," Computer J. 10, 1967, 300-305.

30. Tewarson, R., Sparce Matrices. Academic Press, New York, 1973.

31. von Neuman, J. and Goldstine, H., "Numerical Inverting of Matrices of High Order," Bulletin Am. Math. Soc. 55 (1947), 1021-1099.

32. Weil, R. and Kettler, P., "Rearranging Matrices to Block-angular Form for Decomposition (and other) Algorithms," Management Science 18, 1(September 1971), 98-108.

33. Wilkinson, J., "The Solution of Ill-conditioned Linear Equations," in Ralston, A. and Wilf, H., Mathematical Methods for Digital Computers, Wiley, New York, 1967 (v.2) 65-93.

34. Willoughby, R., ed., IBM Sparce Matrix Proceedings," IBM Report RA1-11077, 1969.

35. Willoughby, R., "Sparce Matrix Algorithms and Their Relation to Problem Classes and Computer Architecture," IBM Report RC 2833, 1970.