Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document No. 109

A PARALLEL ALGORITHM FOR SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEMS

by

Hui-Ming Huang

February 1974

CAC Document No. 109

A PARALLEL ALGORITHM FOR SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEMS

by

Hui-Ming Huang

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

February 1974

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC04-72-C-0001, as monitored by the U. S. Army Research Office-Durham, and was submitted in partial fulfillment of the requirements for the degree of Master of Science in the Graduate College of the University of Illinois at Urbana-Champaign, 1974.

ACKNOWLEDGEMENT

The author wishes to express his sincere appreciation and gratitude to his advisor, Professor Ahmed H. Sameh for his patient guidance and invaluable help throughout the course of his work, without which this work would have been impossible. Sincere thanks also go to Karen Russbach for typing the manuscript.

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense as monitored by the U.S. Army Research Office under Contract No. DAHC04-72-C-0001, in part by the Department of Computer Science, and in part by the Center for Advanced Computation of the University of Illinois at Urbana-Champaign.

TABLE OF CONTENTS

I. INTRODUCTION

It is well known that many problems in physical sciences give rise to the algebraic eigenvalue problem $Ax = \lambda x$ where A is usually a large symmetric matrix. The most effective way to solve such problems consists of reducing A to the tridiagonal form [1, 13] and using either the bisection method or the QL algorithm to find few or all the eigenvalues. The major goal of this work is to study and analyze both algorithms on a parallel machine, namely the ILLIAC IV, and find out the cases in which each algorithm should be used.

The QL algorithm is an effective and reliable algorithm for finding all or most of the eigenvalues on a serial machine. However, the implementation of this algorithm on a parallel machine is very ineffective since only two or three PE's can be used at the most. On the other hand, the bisection method is usually preferred for finding only few of the eigenvalues since the algorithm is especially designed to isolate the individual eigenvalues. The bisection method therefore can be easily generalized for implementation on a parallel machine. We shall describe this modified algorithm which we will call Multisection method, and show how it is implemented on the ILLIAC IV. We will also give comparisons of time estimates, for finding all or few of the eigenvalues for some tridiagonal matrices, between this parallel algorithm and the serial QL algorithm.

Finally, we will provide in an APPENDIX the program listing written in an ILLIAC IV language, GLYPNIR.

II. THE MULTISECTION METHOD

1. THEORETICAL BACKGROUND

(1) Sturm Sequences

It is usually desired to determine lower and upper bounds for the real roots of a polynomial so that errors in approximating them can be easily estimated. This can be done if we are able to determine the number of roots of a polynomial within an interval, this in fact can be achieved using the property of the Sturm sequences.

The Sturm sequence is defined as follows [10]:

DEFINITION. Let $f_o(x)$, $f_1(x)$, ..., $f_m(x)$ be a sequence of continuous functions. Such a sequence is called a Sturm sequence on an interval [a, b], where either a or b may be infinite, if

(1) $f_o(x)$ has at most simple roots in [a, b];

(2) $f_m(x)$ does not vanish in [a, b];

(3) if $f_k(\alpha) = 0$, then $f_{k-1}(\alpha)\, f_{k+1}(\alpha) < 0$ for any root $\alpha \varepsilon [a, b]$;

(4) if $f_o(\alpha) = 0$, then $f_o'(\alpha)\, f_1(\alpha) > 0$ for any root $\alpha \varepsilon [a, b]$.

For every such sequence, there exists the following property.

THEOREM I. [10, p. 126]. The number of zeros of $f_o(x)$ in (a, b) is equal to the difference between the number of sign variations in $\{f_o(a), f_1(a), \ldots, f_m(a)\}$ and in $\{f_o(b), f_1(b), \ldots, f_m(b)\}$ provided that $f_o(a)f_o(b) \neq 0$ and $\{f_o(x), f_1(x), \ldots, f_m(x)\}$ form a Sturm sequence on [a, b].

The result described above is very useful in locating the zeros of a polynomial. Now we discuss how it can be applied to find the eigenvalues.

Let A be a symmetric tridiagonal matrix of order n with d_i and e_i as diagonal and subdiagonal elements respectively,

$$A \equiv \begin{bmatrix} d_1 & e_1 & & & & & \\ e_1 & d_2 & e_2 & & & & \\ & e_2 & d_3 & e_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & e_{n-2} & d_{n-1} & e_{n-1} \\ & & & & & e_{n-1} & d_n \end{bmatrix}, \tag{2.1}$$

recalling that the characteristic polynomial is given by

$$p_n(\lambda) = \det\ (\lambda I - A)$$

and that its principal minors are given by the following recurrence relation

$$p_o(\lambda) = 1,$$

$$p_1(\lambda) = \lambda - d_1,$$

$$p_i(\lambda) = (\lambda - d_i)\ p_{i-1}\ (\lambda) - e_{i-1}^2\ p_{i-2}\ (\lambda) \tag{2.2}$$

$$i = 2, 3, \ldots, n.$$

If any $e_i = 0$, then the determination of the eigenvalues of A is reduced to that of determining the eigenvalues of two smaller matrices. Hence we assume $e_i \neq 0$ for $i = 1, \ldots, n - 1$. We then have

THEOREM II [10, p. 168]. Let the real symmetric tridiagonal matrix A be defined by (2.1), with all $e_i \neq 0$. Then the zeros of each $p_i(\lambda)$, $i = 2, 3, \ldots, n$, are distinct and are separated by the zeros of $p_{i-1}(\lambda)$; and, if $p_n(\lambda) \neq 0$, the number of eigenvalues of A that are larger than λ is equal to the number of sign variations in the sequence $p_n(\lambda), p_{n-1}(\lambda), \ldots, p_1(\lambda), 1$.

Since the eigenvalues of a matrix are but the zeros of the characteristic polynomial of that matrix, and the principal minors of $\lambda I - A$ form a Sturm sequence, then the theorems described in this section can be used as the basis of an algorithm for computing the eigenvalues of A.

(2) <u>The Bisection Method</u>

Suppose the eigenvalues of A are ordered so that $\lambda_1 > \lambda_2 > \ldots > \lambda_n$, if we have two real values a_o and b_o such that

$$b_o > a_o, \qquad V(a_o) < k, \qquad V(b_o) \geq k, \tag{2.3}$$

where $V(x)$ is the number of sign variations in the sequence $\{p_o(x), p_1(x), \ldots, p_n(x)\}$, then from the results in the last section we know that there exists at least an eigenvalue λ_k in the interval (a_o, b_o). The value of λ_k can be determined as accurately as possible by an iterative process called the "Bisection Method." The algorithm is as follows:

Start from the interval (a_o, b_o), suppose that in $(i - 1)$ steps we have established an interval (a_{i-1}, b_{i-1}) such that

$$V(a_{i-1}) < k, \quad V(b_{i-1}) \geq k, \quad b_{i-1} - a_{i-1} = (b_o - a_o)/2^{i-1} \tag{2.4}$$

In the i-th step we compute the mid-point c_i of (a_{i-1}, b_{i-1}), i.e.,

$$c_i = \frac{1}{2}(a_{i-1} + b_{i-1}), \tag{2.5}$$

and evaluate the sequence

$$p_o(c_i),\ p_1(c_i),\ \ldots,\ p_n(c_i) \tag{2.6}$$

to determine $V(c_i)$. Then,

$$\text{if } V(c_i) \geq k, \text{ we take } b_i = c_i,\ a_i = a_{i-1}; \tag{2.7}$$

$$\text{if } V(c_i) < k, \text{ we take } a_i = c_i,\ b_i = b_{i-1}\ . \tag{2.8}$$

In either case, we have

$$(b_i - a_i) = \frac{1}{2}(b_{i-1} - a_{i-1}) \tag{2.9}$$

and

$$V(b_i) \geq k; \quad V(a_i) < k \tag{2.10}$$

so that λ_k is always in the interval (a_i, b_i) and we can locate it in an interval of width $(b_o - a_o)/2^p$ in p steps of the bisection process.

With exact computation, this method can be used to determine any eigenvalue to a prescribed accuracy without referring to other eigenvalues. The process converges with an asymptotic convergence factor of 1/2 [10, p. 128]. It is clearly not a rapid convergence but the algorithm is quite effective.

The major part of the process is the calculation of the sequence (2.6) using formula (2.2). There are roughly 2n multiplications and 2n additions for each evaluation of the sequence. If all the n eigenvalues are determined in t bisection steps, then essentially $2n^2t$ multiplications are required (n is large). Remember that e_i^2, i = 1, 2, ..., n - 2, are computed once and for all, this contributes (n - 1) multiplications to the total number of operations.

The situation is slightly complicated when there are a number of very close eigenvalues, since $p_n(\lambda) = \prod_{i=1}^{n} (\lambda - \lambda_i)$ is very small for any λ which is in the neighborhood of these eigenvalues, the zeros of each $p_i(\lambda)$ separate those of $p_{i+1}(\lambda)$ and accordingly many of the $p_i(\lambda)$ may also become very small and give rise to underflow. Also, if some of the eigenvalues are very large, then overflow may cause troubles during the evaluation of the Sturm sequence.

This difficulty may be avoided by a simple modification [6]. The sequence of $p_i(\lambda)$ is replaced by a sequence of $q_i(\lambda)$ defined by

$$q_i(\lambda) = p_i(\lambda)/p_{i-1}(\lambda), \quad i = 1, 2, \ldots, n . \tag{2.11}$$

$V(\lambda)$ is now given by the number of negative $q_i(\lambda)$. The $q_i(\lambda)$ satisfies the relations

$$q_1(\lambda) = \lambda - d_1,$$

$$q_i(\lambda) = (\lambda - d_i) - e_{i-1}^2/q_{i-1}(\lambda) \quad i = 2, \ldots, n . \tag{2.12}$$

In case $q_{i-1}(\lambda)$ is zero for some i, we just replace the zero $q_{i-1}(\lambda)$ by a suitable small number and the error analysis for the $p_i(\lambda)$ sequence applies almost unaltered to the $q_i(\lambda)$ sequence [6].

Comparing the computation of the sequences $q_i(\lambda)$ and $p_i(\lambda)$, we find that two multiplications have been replaced by one division. Furthermore, we only have to detect the sign of $q_i(\lambda)$ instead of those of $p_{i-1}(\lambda)$ and $p_i(\lambda)$. For serial computers in which the execution of a multiplication or a division uses almost the same time, the replacement of $p_i(\lambda)$ by $q_i(\lambda)$ causes an improvement in the execution time. But, for the parallel computer ILLIAC IV, the execution time for a division is about six times that of a multiplication, so usually the $p_i(\lambda)$ sequence is evaluated except for some special cases which will be discussed later.

The major process in the bisection method is the calculation of the Sturm sequence $p_i(\lambda)$ (or $q_i(\lambda)$). Once the eigenvalues are separated, we may use any algorithm for the calculation of a zero of a real function to find the eigenvalue within each interval. The bisection method can still be used to calculate the eigenvalues up to some specified accuracy after we separate all of them. But, it converges only linearly. Hence some other algorithms with higher convergence rate must be considered. In this paper we select the Zero-in algorithm [11] which is due to Wijngaarden et. al. to calculate the eigenvalues once we separate them. This algorith, a method of order $(\sqrt{5} + 1)/2$ [10, pp. 100-101] is described in APPENDIX I.

2. IMPLEMENTATION

The multisection method contains two major stages. In the first stage we calculate the Sturm sequence $p_i(\lambda)$ (or $q_i(\lambda)$) and find all the intervals each of which contains only one eigenvalue. In the second stage we use the zero-in algorithm to simultaneously determine 64 eigenvalues at a time. We shall consider the implementation of this method on a $N = 16$ PE machine for a symmetric tridiagonal matrix of order $n = N = 16$.

Figure 1 shows the storage allocation scheme for the first stage. We store the diagonal and subdiagonal elements of A across the PEs. This kind of storage allocation is very convenient for the parallel operation which will be discussed later. The quantities e_i^2 $(i = 0, 1, ..., n-1)$ will be used frequently when calculating the Sturm sequence, so we compute them and store them into row P for later use.

If all the eigenvalues are required, we use $(-||A||_\infty, ||A||_\infty)$ as the initial interval, where $||A||_\infty = \max_i g_i$. The g_i's can be obtained and stored into row A by one statement, that is,

$$\text{row A} = \text{row D} + \text{row E} + (\text{row E route right one PE}). \qquad (2.13)$$

Then, one more instruction is required to pick the maximum element in row A as the value of $||A||_\infty$. All the quantities e_i^2 are also computed and stored into row P by one instruction,

$$\text{row P} = \text{row E} \times \text{row E}. \qquad (2.14)$$

If the dimension of the matrix is not greater than N, we then always have this simple situation.

PE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ROW D	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}
ROW E	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}	e_{15}	0
ROW P	e_1^2	e_2^2	e_3^2	e_4^2	e_5^2	e_6^2	e_7^2	e_8^2	e_9^2	e_{10}^2	e_{11}^2	e_{12}^2	e_{13}^2	e_{14}^2	e_{15}^2	0
ROW A	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_{11}	g_{12}	g_{13}	g_{14}	g_{15}	g_{16}

ROW D: DIAGONAL ELEMENTS OF MATRIX A

ROW E: SUBDIAGONAL ELEMENTS OF MATRIX A

ROW P: SQUARE OF e_i

ROW A: $g_i = |d_i| + |e_i| + |e_{i-1}|, \; e_n = e_o = 0$

FIGURE 1. INITIAL STORAGE ALLOCATION SCHEME FOR THE FIRST STAGE--STURM SEQUENCE PROCESS

The major process in the first stage is to calculate $V(\mu_i)$. The storage allocation scheme for this process is shown in Fig. 2. For the given interval (a, b), assume that the value V(b) is known, we divide (a, b) into N = 16 subintervals. Let the left end point of the i-th interval be μ_i, we then have

$$\mu_i = a + (i - 1)\frac{b - a}{N} . \tag{2.15}$$

All the μ_i's are stored into row T by using

$$h = (b - a)/N \tag{2.16}$$

and

$$\text{row } T = a + \text{PEN} \times h, \tag{2.17}$$

where PEN is an integer whose value is j in PE_j. Usually, (2.2) is used to evaluate the sequence $p_i(\lambda)$ because it requires less computation time, on the ILLIAC IV. But if there is any subdiagonal element which is relatively small with respect to the rest of the elements of A, then (2.12) is used. The process is itself sequential, but all N $V(\mu_i)$'s can be calculated simultaneously since each PE contains one μ_i and all the PEs are kept busy. Once the $V(\mu_i)$'s are computed, they are stored into row V (see Fig. 2).

The number of eigenvalues contained in the intervals $(\mu_1, \mu_2), \ldots, (\mu_{15}, \mu_{16})$ can be found by

$$\text{row } M = (\text{row } V \text{ shift left one PE}) - \text{row } V. \tag{2.18}$$

$V(b) - V(\mu_N)$ gives the number of eigenvalues contained in the Nth interval (μ_N, b) where V(b) is known.

PE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ROW T	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9	μ_{10}	μ_{11}	μ_{12}	μ_{13}	μ_{14}	μ_{15}	μ_{16}
ROW V	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}
ROW M	1	1	0	0	0	1	0	1	0	1	1	0	0	3	0	1
ROW EL_1	ℓb_1					ℓb_6		ℓb_8			ℓb_{11}					ℓb_{16}
ROW EL_2 • • • ROW EL_k		ℓb_2								ℓb_{10}						
ROW EU_1	ub_1					ub_6		ub_8			ub_{11}					ub_{16}
ROW EU_2 • • • ROW EU_k		ub_2								ub_{10}						
ROW L_1														ℓ_{14}		
ROW L_2 • • • ROW L_j																
ROW U_1														u_{14}		
ROW U_2 • • • ROW U_j																

ROW T: LEFT TERMINAL POINT OF EACH INTERVAL

ROW V_i: # OF SIGN VARIATIONS IN STURM SEQUENCE AT POINT μ_i

ROW EL_i AND EU_i: LOWER AND UPPER BOUNDS OF THE SUBINTERVAL WHICH CONTAINS EXACTLY ONE EIGENVALUE

ROW E_i AND U_i: LOWER AND UPPER BOUNDS OF THE SUBINTERVAL WHICH CONTAINS MORE THAN ONE EIGENVALUE

FIGURE 2. STORAGE ALLOCATION SCHEME AT THE END OF THE FIRST STAGE

Once row M is obtained, its contents are examined. For these PEs with M equal to 0, nothing is done. For those which have M equal to 1, the μ_i's are stored into the proper position in row EL_j (the value of j may be different in different PE's) as the lower bounds of the intervals that contain only one eigenvalue. The values of $\mu_i + h$ are stored into the corresponding position in row EU_j as the upper bounds (see Fig. 2). The subintervals which contain more than one eigenvalue are treated similarly where μ_i and $\mu_i + h$ are stored in row L_j and U_j respectively.

The process described above can be performed simultaneously for each group of PE's with the same value of V_i. The shaded area in the figure means that some μ_i or $\mu_i + h$ are stored there.

We repeat the above mentioned process for each subinterval which contains more than one eigenvalue (i.e., the subintervals stored in the area between row L_1 and row U_j in Fig. 2) until no more subintervals contain more than one eigenvalue. This can be achieved, at least in principle, since all the eigenvalues are distinct.

The storage allocation scheme for the implementation of the second stage, namely the Zero-in algorithm (App. I), is shown in Fig. 3. Since the diagonal and subdiagonal elements of the matrix A are referred to very often in each PE during the process, they are duplicated into each PE at the beginning of this stage (from rwo D_1 to row E_{n-1}). This is feasible since we assume that the order of the matrix is equal to the number of PE's and 64<<2048 which is the number of rows in the PE memory. The lower bounds and upper bounds of the subintervals that contain only one eigenvalue are stored in rows B and C respectively as initial contents. The b_i and c_i are interchanged if necessary to satisfy the condition

PE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
* ROW A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
* ROW B	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}
* ROW C	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}
* ROW D_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1	d_1
* ROW D_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2	d_2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
ROW D_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n	d_n
ROW E_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1	e_1
ROW E_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2	e_2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
ROW E_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}	e_{n-1}
* ROW PA	pa_1	pa_2	pa_3	pa_4	pa_5	pa_6	pa_7	pa_8	pa_9	pa_{10}	pa_{11}	pa_{12}	pa_{13}	pa_{14}	pa_{15}	pa_{16}
* ROW PB	pb_1	pb_2	pb_3	pb_4	pb_5	pb_6	pb_7	pb_8	pb_9	pb_{10}	pb_{11}	pb_{12}	pb_{13}	pb_{14}	pb_{15}	pb_{16}
* ROW PC	pc_1	pc_2	pc_3	pc_4	pc_5	pc_6	pc_7	pc_8	pc_9	pc_{10}	pc_{11}	pc_{12}	pc_{13}	pc_{14}	pc_{15}	pc_{16}
* ROW INT	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}
* ROW MID	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{16}

ROW PA: VALUES OF $p_n(a_i^{(s)})$, DENOTED BY pa_i.

ROW PB: VALUES OF $p_n(b_i^{(s)})$, DENOTED BY pb_i.

ROW PC: VALUES OF $p_n(c_i^{(s)})$, DENOTED BY pc_i.

ROW INT: VALUES OF LINEAR INTERPOLATION BETWEEN $a_i^{(s)}$ AND $b_i^{(s)}$.

ROW MID: VALUES OF MIDPOINT OF INTERVALS $(b_i^{(s)}, c_i^{(s)})$.

* ALL THE ENTRIES SHOWN IN THESE ROWS ARE THE VALUES AT sTH ITERATION, THE SUPERSCRIPT "s" IS OMITTED.

FIGURE 3. STORAGE ALLOCATION SCHEME FOR THE SECOND STAGE--ZERO-IN ALGORITHM

$|p_n(b_i)| \leq |p_n(c_i)|$. The entries a_i's in row A are then chosen to be the same as c_i. The values of $p_n(a_i)$, $p_n(b_i)$ and $p_n(c_i)$ are computed simultaneously for i = 1, ..., N and stored into rows PA, PB and PC. The interpolations p_i of a_i and b_i are stored in row INT. The midpoints m_i of b_i and c_i are stored in row MID. The new values of b_i in row B are taken to be p_i or m_i depending on whether p_i is between m_i and b_i. This can be done simply by using an IF statement. The new contents of row A and row C are then changed as described in APPENDIX I. We then start the next iteration. When the stopping criteria are satisfied in some PEs, these PEs are disabled. But for those PEs which are still enabled, the process continues until the stopping criteria are satisfied in each PE.

If we have one interval that contains some very close eigenvalues then the number of sturm sequence evaluations may be large, this coupled with the time required for storing and fetching various subintervals for the PE memory can prove to be a time consuming process.

3. TIME ESTIMATES

The time required to find all the eigenvalues for various matrices with different dimensions is discussed in this section. We will discuss first the simple case $n \leq N$, where n is the dimension of the given matrix and N is the number of PEs in the parallel computer.

Two classes of operations will be discussed, data manipulations and arithmetic operations. The clock time taken by each of the four basic arithmetic operations in the ILLIAC IV is shown in TABLE I, part of TABLE 6-2 in [15]. We assume that we use normalized floating-point numbers rounded in 64-bit mode.

TABLE I. Clock Time for Each Arithmetic Operation on ILLIAC IV

Operations	Clock Time*
+ (ADRN)	6
- (SBRN)	7
× (MLRN)	9
÷ (DVRN)	56

* 1 clock time $\approx 60 \times 10^{-9}$ seconds

In what follows we will assume that one addition or subtraction is equivalent to one multiplication, and a division is equivalent to 6 multiplications.

(1) Computation Time (CT)

The computation process in the multisection method can be divided into four major parts:

(a) calculate $||A||_\infty$,

(b) calculate the square of each subdiagonal element,

(c) evaluate the Sturm sequence $\{p_i(\lambda)\}$ or $\{q_i(\lambda)\}$, and

(d) execute the zero-in algorithm.

The norm $||A||_\infty$ is used in deciding upon the stopping criterion. Besides, if all the eigenvalues are required, we can use $(-||A||_\infty, ||A||_\infty)$ as the initial interval. To find the norm, (2.13) is used. This part requires 2 additions and is executed only once.

For part (b), we use (2.14). The square of e_i $(i = 1, \ldots, n - 1)$ is calculated once and for all, so only one multiplication is required.

For the evaluation of the Sturm sequence we assume that (2.2) is used, this requires 2n multiplications and 2n additions. Besides, to find the number of eigenvalues contained in each interval requires 1 subtraction.

The zero-in algorithm is an iterative process. Many iterations may be required before we obtain an eigenvalue. During each iteration, (2.2) is used to evaluate the value of the characteristic polynomial. Then,

$$INT_i = [b_i\, p_n\,(a_i) - a_i\, p_n\,(b_i)]/[p_n\,(a_i) - p_n\,(b_i)] \qquad (2.19)$$

is used for interpolation, hence $2(n + 1)$ multiplications, $2(n + 1)$ subtractions and 1 division are required.

Suppose p is the number of times we repeat part (c) to separate all the eigenvalues and q is the number of required iterations in the Zero-in algorithm. Then the total number of arithmetic operations used to obtain all the eigenvalues is approximately

$$6q + 4n\ (p + q) \qquad (2.20)$$

equivalent multiplications, and the total time, in microseconds, for the arithmetic operations is roughly

$$T_A = \frac{7}{2} q + 2n\ (p + q). \qquad (2.21)$$

If we assume that p and q remain constant as n varies, then (2.21) represents a linear relation between the total CT and the dimension of the matrix, see Fig. 4.

While for $n > N$ where we assume here that $n \leq 1024$, part (a) requires $2 \cdot [\frac{n}{N}]$ additions, and part (b) used $[\frac{n}{N}]$ multiplications, where $[\frac{n}{N}]$ is the smallest integer no less than $\frac{n}{N}$. The number of the required operations in parts (c) and (d) remains the same. Thus, from (2.21), the total CT, in microseconds, for the general case is approximately

$$t_A = \frac{5}{4} [\frac{n}{N}] + \frac{7}{2} q + 2n\ (p + q), \qquad (2.22)$$

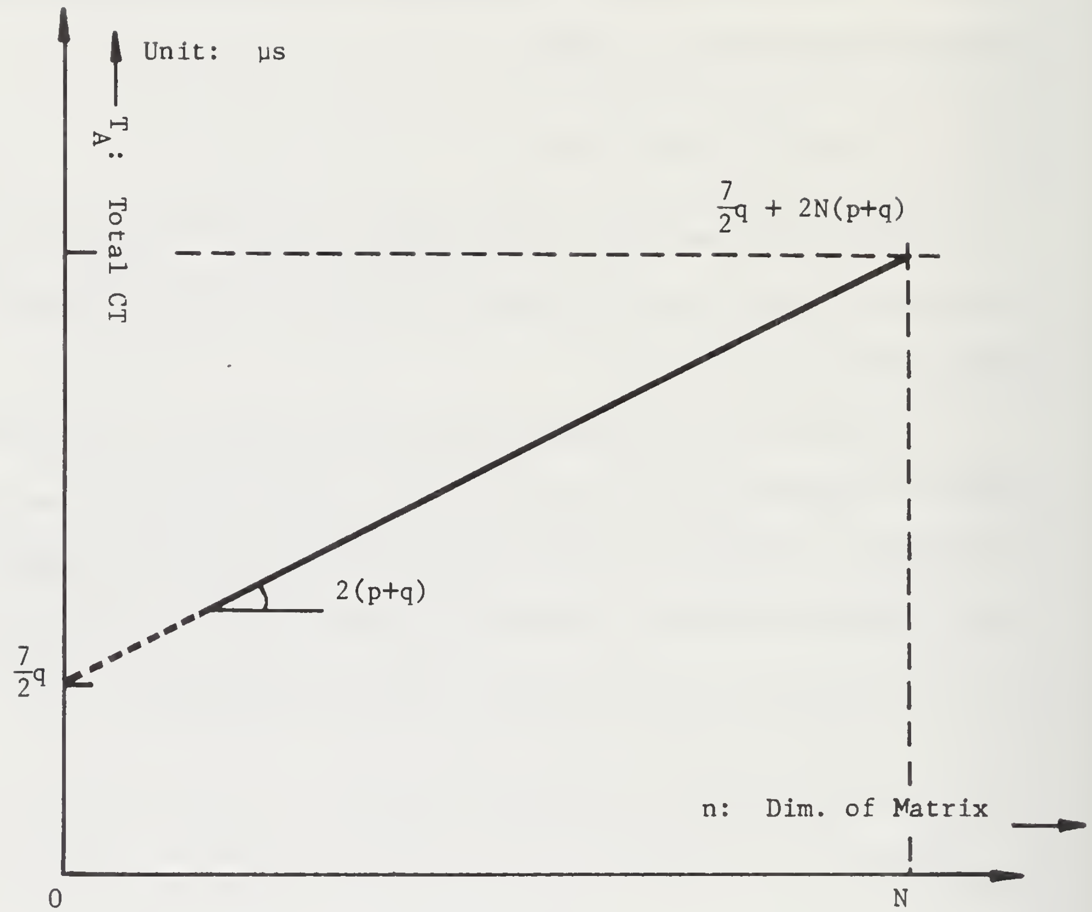
see Fig. 5.

(2) Data Manipulation Time (DMT)

Most of the data manipulation occurs during the following processes:

(a) Dividing the given interval into N subintervals and distributing the left terminals into all PEs.

(b) To evaluate the Sturm sequence in all PEs simultaneously, we have to grab d_i and e_i^2 one by one from the memory because we store them across all PEs. This process takes 2n times of the function: GRABONE.



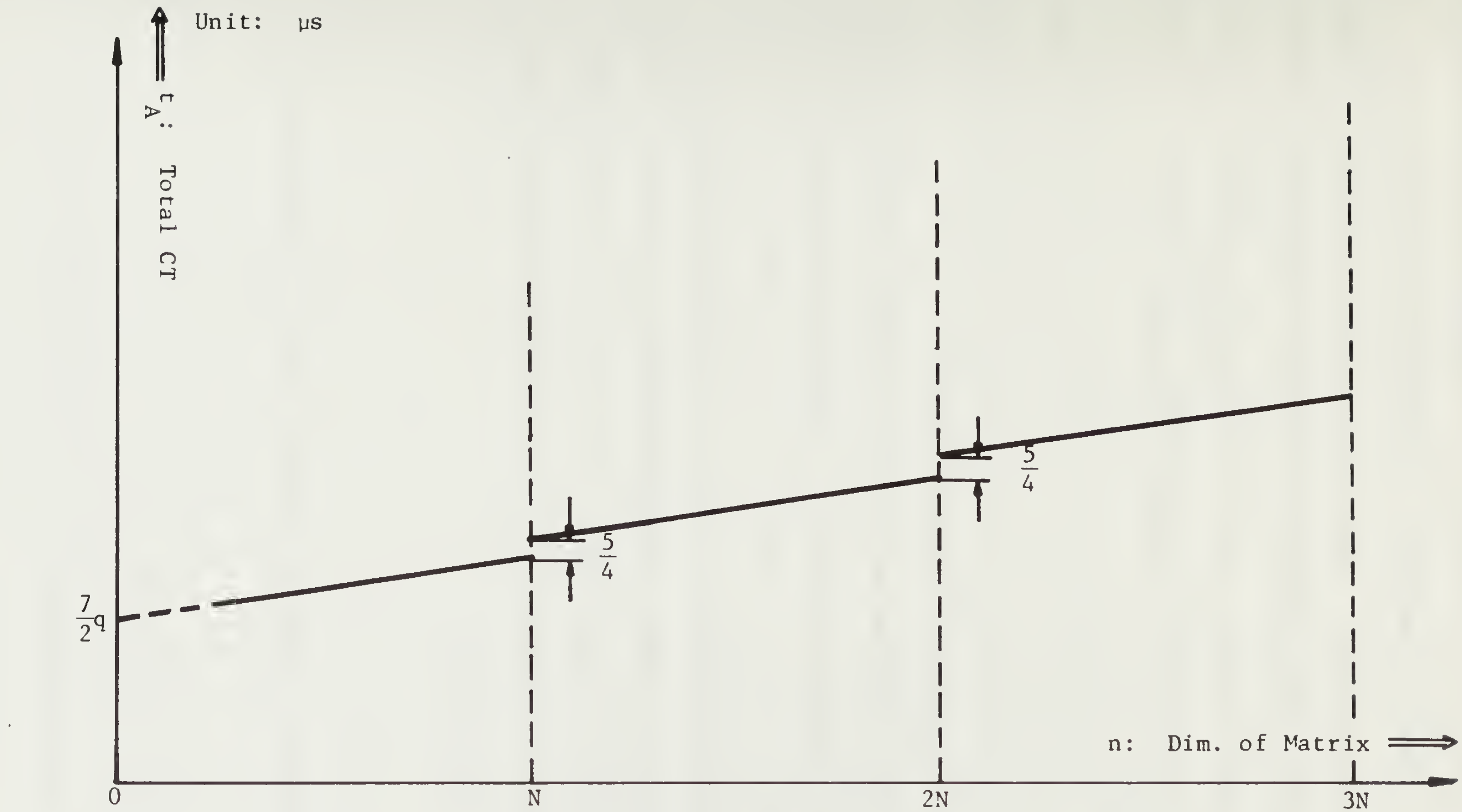
RELATION: $T_A = \frac{7}{2}q + 2(p + q)n$

p: # of times of Sturm sequence evaluations

q: # of required iterations in Zero-in algorithm

N: # of PEs in parallel computer

FIGURE 4. TOTAL COMPUTATION TIME FOR $n \leq N$



RELATION: $t_A = \frac{5}{4}\lceil\frac{n}{N}\rceil + \frac{7}{2}q + 2(p + q)n$

p: # of times of Sturm sequence evaluations

q: # of iterations required in zero-in algorithm

N: # of PEs in parallel computer

FIGURE 5. TOTAL COMPUTATION TIME

(c) As mentioned above, the Sturm sequence may be evaluated more than once to separate all the eigenvalues. Except for the first evaluation, we have to pick up the interval which contains more than one eigenvalue from the PE memory shown in Fig. 2. Since the intervals are stored at random, i.e., not every PE contains such kind of interval, we must test all the PEs and enable those PEs which contain such intervals, then pick up the interval contained in the PE with maximum PEN. This requires one execution of the MAX function.

(d) Before executing the zero-in algorithm, we must distribute the diagonal elements and the squares of the subdiagonal elements into all PEs. This requires 2n executions of GRABONE function.

Let p and q have the same meaning as before, then the data manipulation requires roughly p calls of the function MAX and 2np calls of the function GRABONE.

The execution time for the functions MAX and GRABONE can be calculated according to GLYPNIR PROGRAMMING MANUAL and TABLES G-1, G-2 in [15]. The function MAX requires 62 clocks while the function GRABONE requires 15 clocks. Thus the time, in microseconds, required for data manipulation is approximately

$$T_D = t_D = 2pn. \tag{2.23}$$

By adding the t_A and t_D, we obtain the total execution time,

$$t_T = \frac{5}{4} [\frac{n}{N}] + \frac{7}{2} q + 2n (q + 2p). \tag{2.24}$$

Fig. 6 shows the total execution time for the multisection method to find all the eigenvalues with respect to the dimension of the given matrix for some sets of values p and q.

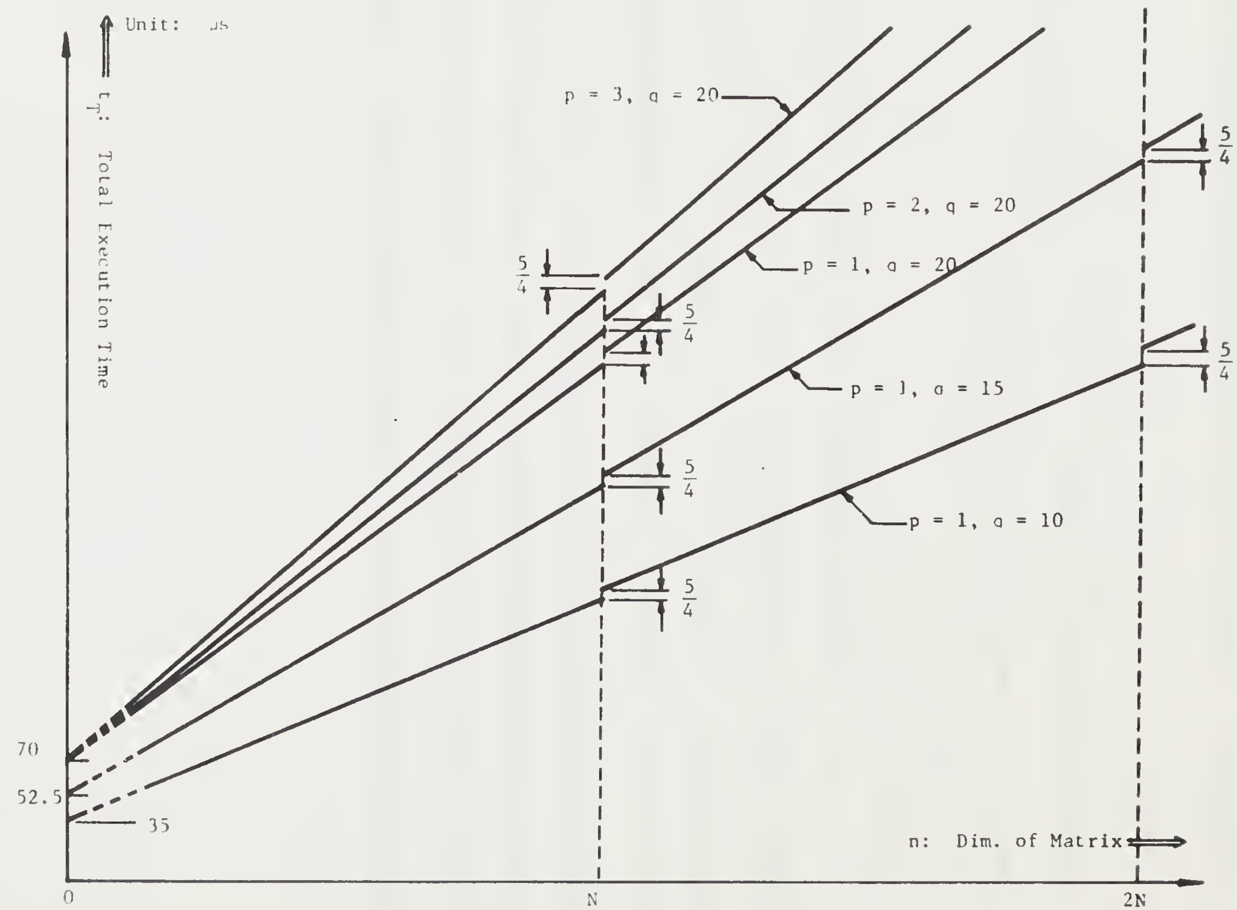
Let n be the dimension of matrix A, $(-||A||_\infty, ||A||_\infty)$ be the initial interval which contains all the eigenvalues of A, and t_B be the number of bisections on a serial machine to obtain an eigenvalue, then

$$2^{1-t_B}||A||_\infty \leq \varepsilon. \tag{2.25}$$

Using the multisection method on a parallel machine to separate the same eigenvalue we require, say, t_M steps, then

$$2||A||_\infty/N^{t_M} \leq \varepsilon. \tag{2.26}$$

Solving for t_B and t_M from (2.25) and (2.26), we find that the ratio t_B/t_M is roughly equal to $\ell n\ N/\ell n\ 2$, i.e., $\ell og_2 N$. For ILLIAC IV, N = 64, the ration t_B/t_M is 6. This is not much of an improvement. However, when we use the multisection method to separate all the eigenvalues we then use either the Zero-in algorithm or the bisection method to compute 64 eigenvalues in those intervals simultaneously. Hence this method is at most $6\cdot[n/64]$ times as fast as the serial computer to find all the eigenvalues. For a matrix with dimension $n \doteq 4096$, the multisection method is at most 284 times faster.



RELATION: $t_T = \frac{5}{4}\left[\frac{n}{N}\right] + \frac{7}{2}q + (2p + q)n$

p: # of times of sturm sequence evaluations
q: # of iterations required in zero-in algorithm
N: # of PEs in parall computer

FIGURE 6. TOTAL EXECUTION TIME

4. ERROR ANALYSIS

The error bounds for the eigenvalues of a symmetric tridiagonal matrix calculated by the bisection method are $6x2^{-t} \; ||A||_{\infty}$, [7, 12], for any reasonable round-off procedure in floating-point binary arithmetic if a t-digit machine is used. Note that the bound is independent of n, the dimension of the matrix.

In the first stage of the multisection method all the eigenvalues are separated by the Sturm sequence process which is substantially the same as the bisection method except that the interval which contains the eigenvalues is divided by N instead of 2. Thus the numerical accuracy in this stage should not be worse than that in the bisection method. In fact, very few iterations of the Sturm sequence processes are required to separate all the eigenvalues.

The Zero-in algorithm can be thought of as a bisection method combined with linear interpolation. Usually, the interpolated point is used as a new end point of the new interval. But if it is outside the old interval, the midpoint of this interval is used as a new end point, this is in fact a bisection process. Hence we have the same error bounds for the computed eigenvalues, $6 \times 2^{-t} \times ||A||_{\infty}$. We see that the relative error may be quite large for small eigenvalues.

III. RESULTS AND DISCUSSION

Consider the matrix B,

$$\begin{bmatrix} x & 1 & & & & & & \\ 1 & -x & 1 & & & & & \\ & 1 & x & 1 & & & & \\ & & 1 & -x & 1 & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & 1 & x & 1 \\ & & & & & & 1 & -x \end{bmatrix}_{n\times n}, \tag{3.1}$$

where x is real.

1. EIGENVALUES

The eigenvalues are computed by both **the** QL algorithm [3, tqℓ1] and the multisection method. TABLE II gives some of the results computed on ILLIAC IV simulator (B6700). The tolerance ε in both methods is set to be 10^{-15}.

The exact eigenvalues of matrix (3.1) can be computed by the following formulae:

$$\lambda_k = [x^2 + 4\cos^2(\frac{k}{n+1})]^{1/2}, \tag{3.2}$$

$$\lambda_{n+1-k} = -\lambda_k, \qquad k = 1, \ldots, [n/2], \tag{3.3}$$

$$\lambda_{\frac{n+1}{2}} = 0 \quad \text{if} \quad n \text{ is odd}. \tag{3.4}$$

TABLE III gives the values of λ_i of some matrices for the comparison with TABLE II. The λ_i's are computed by (3.2) (3.4) on the IBM 360/75 in long precision.

TABLE II. Eigenvalues Computed on ILLIAC IV Simulator

For x = 1.0, N = 4, the eigenvalues:

Multisection	QL	
-1.902113032590307	-1.902113032590400	(0)**
-1.175570504584947*	-1.175570504585025	(1)
1.175570504584947	1.175570504584961	(4)
1.902113032590300	1.902113032590350	(2)

For x = 1.0, n = 8, the eigenvalues:

Multisection	QL	
-2.128870331006084	-2.128870331006297	(0)
-1.829561793253738	-1.829561793253966	(1)
-1.414213562363092	-1.414213562373334	(2)
-1.058590930637600	-1.058590930637820	(4)
1.058590930637600	1.058590930637628	(4)
1.414213562373092	1.414213562373106	(2)
1.829561793253738	1.829561793253809	(3)
2.128870331006098	2.128870331006169	(2)

(Continued on next page)

* The underline shows the digits which do not match with the values shown in Table III.

** The numbers in parentheses indicate the number of iterations required to obtain that eigenvalue by TQL1.

TABLE II (continued)

For $x = 10^{-5}$, $N = 12$, the eigenvalues:

Multisection	QL	
-1.941883634877847	-1.941883634878216	(0)
-1.770912051334648	-1.770912051335017	(1)
-1.497021496375602	-1.497021496375986	(2)
-1.136129493506317	-1.136129493506708	(2)
-0.7092097741555747	-0.7092097741559833	(2)
-0.2410733607180546	-0.2410733607183300	(2)
0.2410733607180529	0.2410733607178379	(2)
0.7092097741555747	0.7092097741555499	(3)
1.136129493506324	1.136129493506324	(4)
1.497021496375602	1.497021496375638	(3)
1.770912051334655	1.770912051334690	(3)
1.941883634877847	1.941883634877868	(2)

For $x = 10^{-5}$, $N = 24$, the eigenvalues:

Multisection	QL	
-1.984229402654165	-1.984229402654982	(0)
-1.937166322283062	-1.937166322283844	(1)
-1.859552971803382	-1.859552971804177	(2)
-1.752613360116257	-1.752613360117010	(2)
-1.618033988780788	-1.618033988781669	(2)
-1.457937254877116	-1.457937254878054	(2)
-1.274847979536602	-1.274847979537455	(2)

(Continued on next page)

TABLE II (continued)

Multisection	QL	
-1.071653590004651	-1.071653590005390	(2)
-0.8515585831888650	-0.8515585831895649	(2)
-0.6180339888307955	-0.6180339888314172	(2)
-0.3747626293048665	-0.3747626293054385	(2)
-0.1255810394567707	-0.1255810394572610	(2)
0.1255810394567760	0.1255810394565282	(2)
0.3747626293048683	0.3747626293046693	(2)
0.6180339888307955	0.6180339888306499	(2)
0.8515585831888650	0.8515585831888370	(3)
1.071653590004651	1.071653590004658	(5)
1.274847979536602	1.274847979536652	(3)
1.457937254877109	1.457937254877180	(3)
1.618033988780795	1.618033988780873	(3)
1.752613360116257	1.752613360116300	(3)
1.859552971803389	1.859552971803424	(3)
1.937166322283083	1.937166322283097	(3)
1.984229402654144	1.984229402654158	(1)

TABLE III. Eigenvalues Computed by (3.2) - (3.4) on the IBM 360/75 in Long Precision

For $x = 1.0$, $N = 4$:

±1.90211 30325 90307

±1.17557 05045 84946

For $x = 1.0$, $N = 8$:

±2.12887 03310 06084

±1.82956 17932 53745

±1.41421 35623 73095

±1.05859 09306 37601

For $x = 10^{-5}$, $N = 12$:

±1.94188 36348 77852

±1.77091 20513 34653

±1.49702 14963 75601

±1.13612 94935 06320

±0.70920 97741 55572

±0.24107 33607 18052

For $x = 10^{-5}$, $N = 24$:

±1.98422 94026 54154

±1.93716 63222 83073

±1.85955 29718 03390

±1.75261 33601 16255

±1.61803 39887 80796

±1.45793 72548 77118

±1.27484 79795 36599

TABLE III (continued)

±1.07165 35900 04650
±0.85155 85831 88861
±0.61803 39888 30797
±0.37476 26293 04867
±0.12558 10394 56776

2. ACCURACY

Comparing the eigenvalues listed in the last section, we find that the eigenvalues obtained by the multisection method agree to 14 decimal digits with those listed in TABLE III, a better agreement than those of the QL algorithm [3, tqℓ1].

There are some cases where the multisection method may get into trouble. Consider the matrix of order n such that

$$\begin{aligned} d_i &= 2, \ i = 1, 2, \ldots, n, \\ e_i &= 1, \ i = 1, 2, \ldots, n-1. \end{aligned} \tag{3.5}$$

The eigenvalues of this matrix are

$$4 \sin^2 \left[\frac{\pi k}{2(n+1)}\right], \ k = 1, 2, \ldots, n. \tag{3.6}$$

If n is large the smallest eigenvalues are much smaller than unity. In computing the Sturm sequence for any value of μ we have

$$P_i(\mu) = (\mu - d_i) P_i(\mu) - e_{i-1}^2 P_{i-2}(\mu) \tag{3.7}$$

and the factor $\mu - d_i$ is $\mu - 2$. Suppose we were working in 15-digit decimal floating-point arithmetic, and μ is of order 10^{-9} in magnitude, then for $\mu - 2$, the last 9 digits of μ are completely lost. In this case we may not obtain many significant figures for the small eigenvalues.

Consider the following two graded matrices with diagonal elements varying from 1 to 10^{12} as example. Matrix X is defined by

$$\begin{aligned} d_1, d_2, \ldots, d_{12} &= 1, 2^{10}, \ldots, 12^{10}, \\ e_i &= 1 \text{ for } i = 1, 2, \ldots, 11. \end{aligned} \tag{3.8}$$

Matrix Y is defined by

$$d_1, d_2, \ldots, d_{12} = 12^{10}, 11^{10}, \ldots, 1, \qquad e_i = 1 \text{ for } i = 1, 2, \ldots, 11. \tag{3.9}$$

Both matrices should have the same eigenvalues.

The eigenvalues of X and Y are computed by using Kahan and Varah's recursection method [14] on IBM 360/75 with long precision. They are listed in TABLE IV. The eigenvalues of X computed by both the multisection method and the QL algorithm are shown in TABLE V. Those of matrix Y are listed in TABLE VI.

For the QL algorithm, the eigenvalues of X computed by this method have very high accuracy. But the eigenvalues, especially those with small magnitude , of Y have larger relative errors. However, for the multisection method, the eigenvalues of X and Y are almost the same and are very close to those shown in TABLE IV.

TABLE IV. Eigenvalues Computed by Recursection Method

Eigenvalues of X	Eigenvalues of Y
0.99902 24838 11381 $_{10}+0$	0.99902 24838 11402 $_{10}+0$
0.10240 00960 28228 $_{10}+4$	0.10240 00960 28229 $_{10}+4$
0.59049 00001 62277 $_{10}+5$	0.59049 00001 62275 $_{10}+5$
0.10485 76000 00097 $_{10}+7$	0.10485 76000 00095 $_{10}+7$
0.97656 25000 00035 $_{10}+7$	0.97656 25000 00048 $_{10}+7$
0.60466 17600 00008 $_{10}+8$	0.60466 17600 00029 $_{10}+8$
0.28247 52490 00017 $_{10}+9$	0.28247 52490 00005 $_{10}+9$
0.10737 41824 00008 $_{10}+10$	0.10737 41824 00002 $_{10}+10$
0.34867 84401 00013 $_{10}+10$	0.34867 84401 00012 $_{10}+10$
0.10000 00000 00006 $_{10}+11$	0.10000 00000 00005 $_{10}+11$
0.25937 42460 10014 $_{10}+11$	0.25937 42460 10017 $_{10}+11$
0.61917 36422 40049 $_{10}+11$	0.61917 36422 40033 $_{10}+11$

TABLE V. Computed Eigenvalues of Matrix X

By Multisection	By QL	
0.99902 24838 11327 $_{10}+0$	0.99902 24838 11341 $_{10}+0$	(0)**
0.10240 00960 28223 $_{10}+4$	0.10240 00960 28224 $_{10}+4$	(1)
0.59049 00001 62229 $_{10}+5$	0.59049 00001 62248 $_{10}+5$	(1)
0.10485 76000 00090 $_{10}+7$	0.10485 76000 00093 $_{10}+7$	(1)
0.97656 25000 00011 $_{10}+7$	0.97656 25000 00047 $_{10}+7$	(1)
0.60466 17600 00000 $_{10}+8$	0.60466 17600 00009 $_{10}+8$	(1)
0.28247 52490 00001 $_{10}+9$	0.28247 52490 00017 $_{10}+9$	(1)
0.10737 41824 00000 $_{10}+10$	0.10737 41824 00004 $_{10}+10$	(1)

TABLE V (continued)

0.34867 84400 99998 $_{10}$ + 10	0.34867 84401 00016 8 $_{10}$ + 10	(1)
0.10000 00000 00000 1 $_{10}$ + 11	0.10000 00000 00007 9 $_{10}$ + 11	(1)
0.25937 42460 10001 2 $_{10}$ + 11	0.25937 42460 10012 2 $_{10}$ + 11	(1)
0.61917 36422 40002 4 $_{10}$ + 11	0.61917 36422 40036 6 $_{10}$ + 11	(1)

** This column gives the number of iterations required to isolate the corresponding eigenvalue by TQL1.

TABLE VI. Computed Eigenvalues of Matrix Y

By Multisection	By QL	
0.99902 24838 11316 $_{10}$ + 0	0.99902 25512 37430 $_{10}$ + 0	(2)
0.10240 00960 28223 $_{10}$ + 4	0.10240 00960 22384 $_{10}$ + 4	(1)
0.59049 00001 62229 $_{10}$ + 5	0.59049 00001 61379 $_{10}$ + 5	(1)
0.10485 76000 00085 $_{10}$ +	0.10485 76000 00083 $_{10}$ + 7	(1)
0.97656 25000 00011 $_{10}$ + 7	0.97656 25000 00000 $_{10}$ + 7	(1)
0.60466 17600 00002 $_{10}$ + 8	0.60466 17600 00000 $_{10}$ + 8	(1)
0.28247 52490 00001 $_{10}$ + 9	0.28247 52490 00000 $_{10}$ + 9	(1)
0.10737 41824 00000 $_{10}$ + 10	0.10737 41824 00000 $_{10}$ + 10	(1)
0.34867 84400 99998 $_{10}$ + 10	0.34867 84401 00000 $_{10}$ + 10	(1)
0.10000 00000 00000 $_{10}$ + 11	0.10000 00000 00000 $_{10}$ + 11	(1)
0.25937 42460 10001 $_{10}$ + 11	0.25937 42460 10000 $_{10}$ + 11	(1)
0.61917 36422 40000 $_{10}$ + 11	0.61917 36422 40000 $_{10}$ + 11	(1)

3. EXECUTION TIME

(1) Complete Eigenvalue Problems

For the matrix B in (4.1), we test those with x = 1 and dimension n = 4, 6, 8, 12, 24, 36. The execution time for finding all the eigenvalues of each matrix by using both the multisection method and the QL algorithm on ILLIAC IV simulator are listed in TABLE VII. The unit time is millisecond (ms). For comparison, we plot the execution time versus the dimension of matrix for both methods in Figure 7.

TABLE VII. Execution Time

Dimension of Matrix A	Execution Time (in ms)	
	Multisection	QL
4	2.0	3.6
6	4.8	8.4
8	2.9	13.0
12	6.8	28.0
24	13.0	99.0
36	41.0	220.0

For most of the test matrices except that with n = 36, all the eigenvalues are separated in one execution of the Sturm sequence process. It takes 7 executions of the Sturm sequence process (i.e., p = 7) to separate the eigenvalues of the matrix with n = 36. The Sturm sequence process actually doesn't consume much execution time, but the data fetching and manipulations during this process causes the execution time to increase obviously as p increases. This explains the spike in the

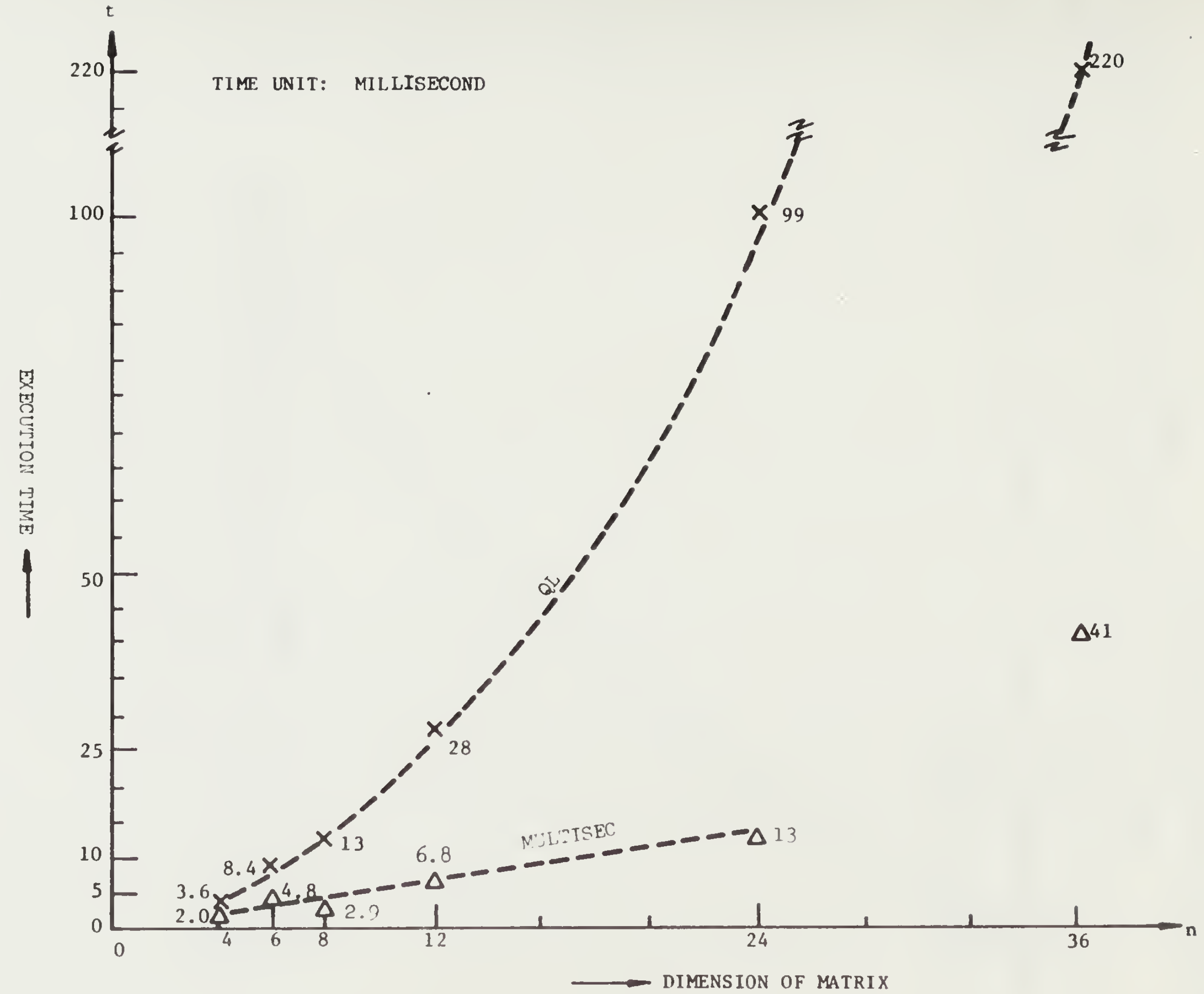


FIGURE 7. COMPARISON ON THE EXECUTION TIME BETWEEN THE MULTISECTION METHOD AND QL ALGORITHM FOR MATRIX B WITH x=1

MULTISEC curve at $n = 36$ in Fig. 7. If the p remains almost constant with respect to n, we can expect the MULTISEC curve to behave similarly for $n > N$. The discrepancy between the execution times in Figs. 6 and 7 is caused by the variation in the number of interpolations (i.e., q in Fig. 6) used in the Zero-in algorithm.

With $x = 10^{-5}$ in the matrix B, we test those with dimension $n = 4, 6, 8, 12, 24$. The execution time for finding all the eigenvalues of each matrix by using both the multisection method and the QL algorithm are shown in Figure 8.

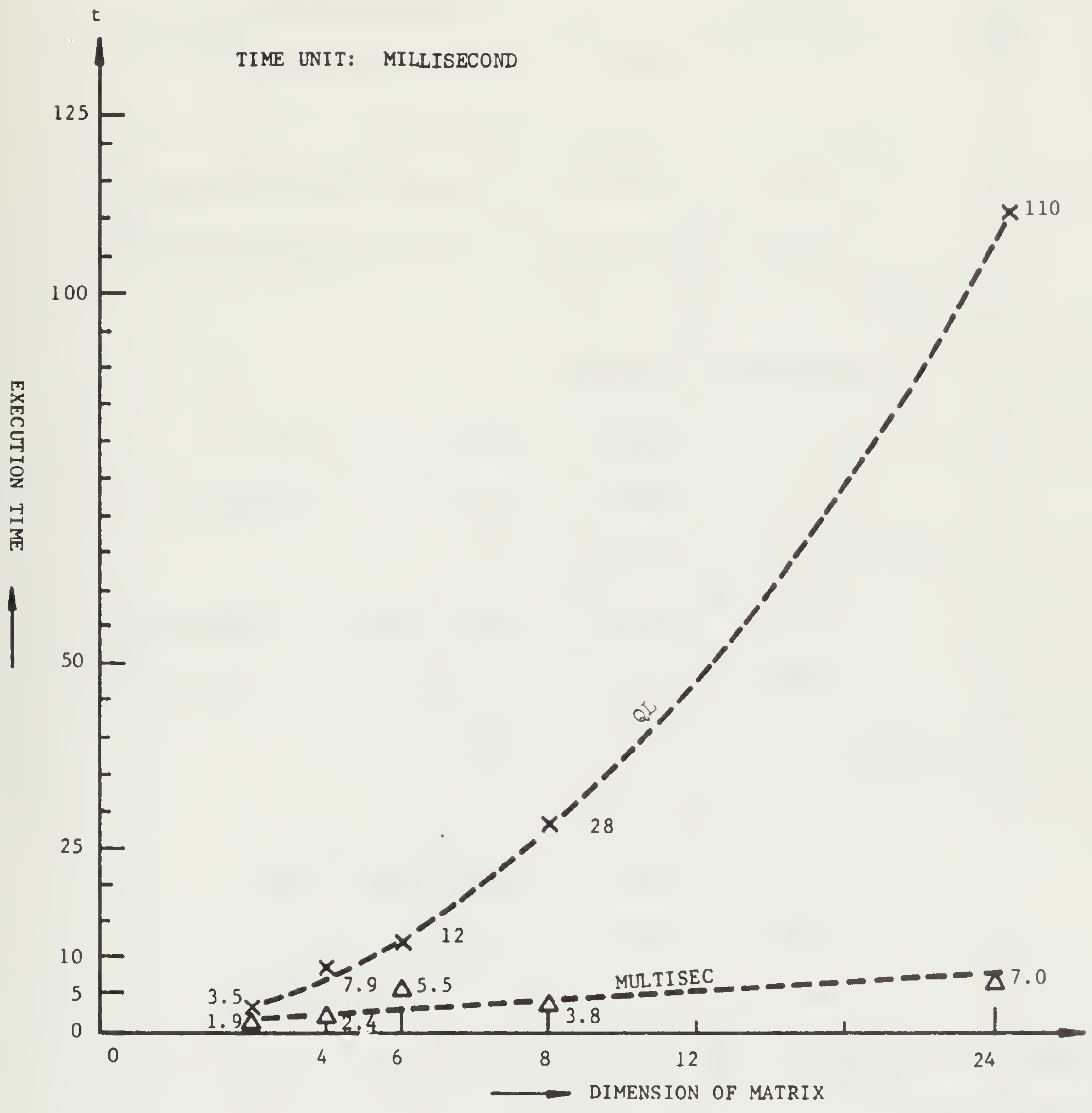


FIGURE 8. COMPARISON ON THE EXECUTION TIME BETWEEN THE MULTISECTION METHOD AND QL ALGORITHM FOR MATRIX B WITH $x=10^{-5}$

(2) Partial Eigenvalue Problems

For partial eigenvalue problems, we have the following results:

(1) Matrix B with $x = 10^{-5}$, $n = 12$:

(a) Multisection method:

3.4 ms to compute 2 leading eigenvalues,

3.2 ms to compute 2 smallest eigenvalues.

(b) Conjugate Gradient method [16]:

23 ms to compute min. eigenvalue with eigenvector,

16 ms to compute max. eigenvalue with eigenvector.

(2) Matrix B with $x = 1.0$, $n = 24$:

(a) Multisection method:

5.7 ms to compute 5 leading eigenvalues,

4.8 ms to compute 5 eigenvalues in the interval (1.0, 1.5).

(b) Conjugate Gradient method:

57 ms to compute min. eigenvalue with eigenvector,

49 ms to compute max. eigenvalue with eigenvector.

(3) Matrix B with $x = 1.0$, $n = 36$:

(a) Multisection method:

14 ms to compute 7 leading eigenvalues,

7.1 ms to compute 7 eigenvalues in the interval (-1.5, -1.0).

If we use the QL algorithm for the cases (1), (2) and (3) we still have to find all the eigenvalues and then select the required eigenvalues from them. The time consumed by the three methods is shown in Figure 9. We didn't test the bisection method, but from the time estimate in Section 2.5, the curve of the bisection method should be between the QL algorithm and the multisection method.

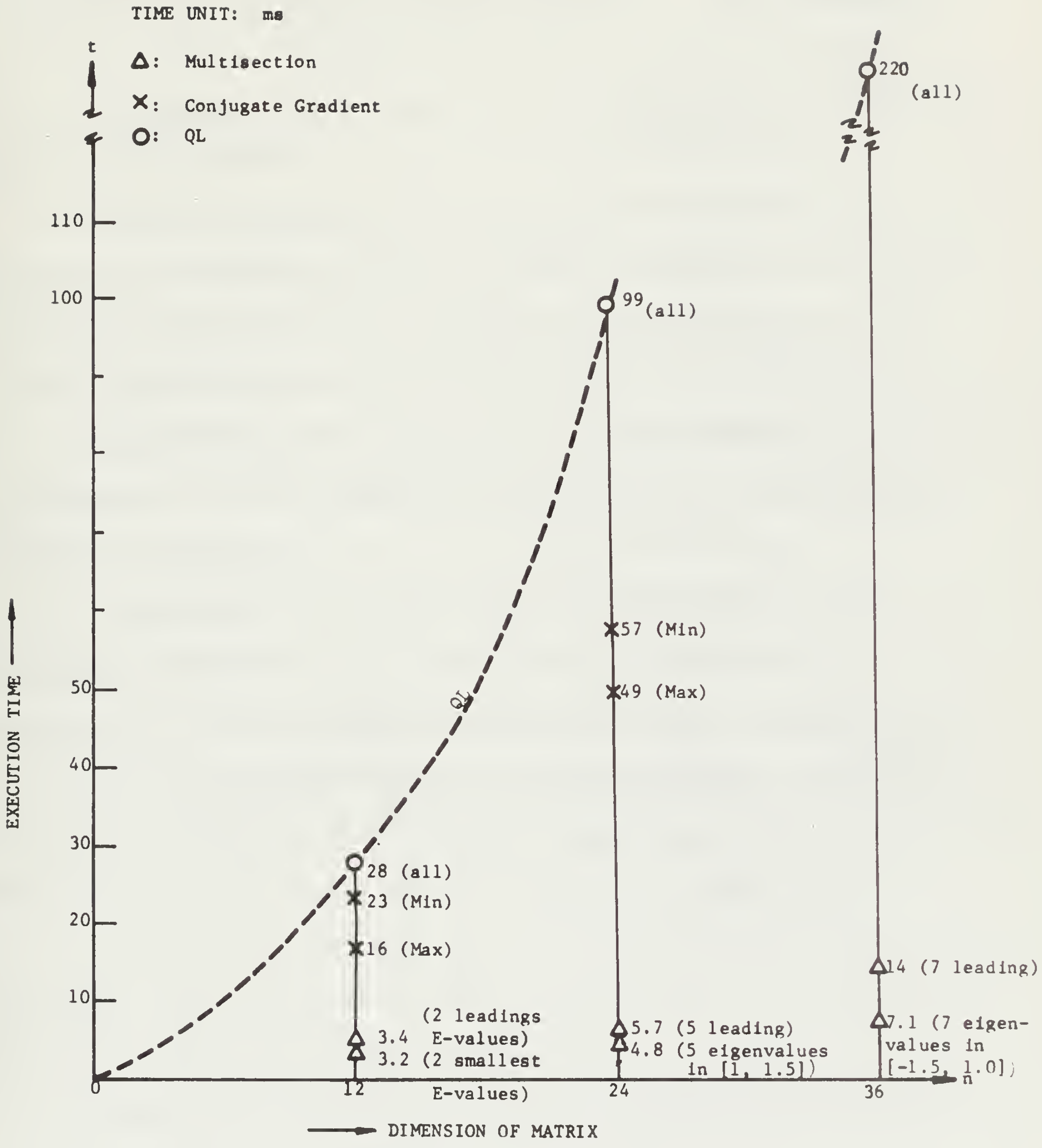


FIGURE 9. COMPARISON ON THE EXECUTION TIME
FOR PARTIAL EIGENVALUE PROBLEMS

(3) Special Cases

The multisection method is not always superior to the QL algorithm in execution time. For example, for the following symmetric tridiagonal matrix,

subdiagonal	diagonal
$+7.83768800584_{10} - 1$	$+4.40021275387_{10} + 0$
$-8.10502269777_{10} + 0$	$+3.80347681618_{10} + 0$
$-2.97985867006_{10} - 11$	$+1.07963104303_{10} + 1$
$-1.92466782539_{10} + 0$	$+4.25675675677_{10} + 0$
$+8.60232526704_{10} + 0$	$+6.74324324323_{10} + 0$
$+0.00000000000$	$+8.00000000000_{10} + 0$

the eigenvalues given by Wilkinson [3] appear as very close pairs,

$-1.59873429360_{10} + 0$,	$-1.59873429346_{10} + 0$
$+4.45598963849_{10} + 0$,	$+4.45598963849_{10} + 0$
$+1.61427446551_{10} + 1$,	$+1.61427446553_{10} + 1$,

thus leading to a very slow process in separating them by the multisection method. The time required to find all the eigenvalues is:

Multisection method: 10 ms

QL algorithm: 5.6 ms.

IV. CONCLUSIONS

Generally, the QL algorithm is an efficient method for computing the eigenvalues of a symmetric tridiagonal matrix on a serial computer. The average number of iterations per eigenvalue is generally less than 2. For example, for the graded matrix (3.8) only one iteration per eigenvalue was required.

However, to implement this algorithm on a parallel machine, we obtain very low efficiency because only two or three PEs are kept busy at any moment during the process. Besides, we have no choice but to find all the eigenvalues, this would be uneconomical for very large matrices.

Comparing the test results shown in the last chapter, we find that the multisection method is much more favorable than the QL algorithm in the following cases:

(1) When some selected eigenvalues are required, for example the largest or smallest, or the eigenvalues in a given interval. We have tried to find the largest eigenvalue of matrix B with $x = 1.0$ and $n = 24$ by using the Conjugate Gradient method implemented in parallel on the ILLIAC IV simulator, it required 49 milliseconds. But the multisection method finds the two leading eigenvalues in only 4.8 milliseconds.

(2) When we are interested only in the general distribution of the eigenvalues rather than their accurate determination. In this case the multisection method gives the result very fast.

(3) When all the eigenvalues are well separated, we may find that we have isolated all the eigenvalues after one step of multisection.

LIST OF REFERENCES

[1] J. H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press, 1965.

[2] J. G. F. Francis, "The QR Transformation," Parts I and II, Computer Journals, 4 (1961-1962), 265-271, 332-345.

[3] H. Bowdler, R. S. Martin, and J. H. Wilkinson, "The QR and QL Algorithm for Symmetric Matrices," Numer. Math., 11 (1968), 293-306.

[4] A. Dubrulle, R. S. Martin, and J. H. Wilkinson, "The Implicit QL Algorithm," Numer. Math., 12 (1968), 377-383.

[5] J. H. Wilkinson, "Global Convergence of Tridiagonal QR Algorithms with Origin Shifts," Linear Algebra and Its Applications, 1 (1968), 409-420.

[6] W. Barth, R. S. Martin, and J. H. Wilkinson, "Calculation of the Eigenvalues of a Symmetric Tridiagonal Matrix by the Method of Bisection," Numer. Math., 9 (1967), 386-393.

[7] J. H. Wilkinson, "Calculation of the Eigenvalues of a Symmetric Tridiagonal Matrix by the Method of Bisection," Numer. Math., 4 (1962), 362-367.

[8] D. J. Kuck, and A. H. Sameh, "Parallel Computation of Eigenvalues of Real Matrices," Center for Advanced Computation, Document No. 9, University of Illinois, Urbana, Illinois, 1971.

[9] A. H. Sameh, and Luke Han, "Eigenvalue Problems," ILLIAC IV Document No. 127, University of Illinois, Urbana, Illinois, 1968.

[10] E. Isaacson, and H. B. Keller, Analysis of Numerical Methods, John Wiley & Sons, Inc., 1966.

[11] G. Peters, and J. H. Wilkinson, "Eigenvalues of $Ax = \lambda Bx$ with Band Symmetric A and B," Comput. J., 12 (1969), 398-404.

[12] J. H. Wilkinson, "Error Analysis of Floating-point Computation," Numer. Math., 2 (1960), 319-340.

[13] C. Reinsch, and J. H. Wilkinson, "Householder's Tridiagonalization of a Symmetric Matrix," Numer. Math., 11 (1968), 181-195.

[14] W. Kahan, and J. Varah, "Two Working Algorithms for the Eigenvalues of a Symmetric Tridiagonal Matrix," Technical Report No. CS43, August 1, 1966, Computer Science Department, Stanford University.

[15] ILLIAC IV Systems Characteristics and Programming Manual, Burroughs Corporation.

[16] W. W. Bradbury, and R. Fletcher, "New Iterative Methods for Solution of the Eigenproblem," Numer. Math., 9 (1966), 259-267.

APPENDIX I. THE ZERO-IN ALGORITHM

Here we describe the zero-in algorithm which is due to Van Wijngaarden, Zonneveld, Dijkstra and Dekker [11, Appendix].

The zero-in algorithm is an extension of the secant method. For a real continuous function f(x) and two given values a and b, $f(a)f(b) < 0$, the secant method is defined by

$$x_o = a, \; x_1 = b, \tag{A1}$$

$$x_{i+1} = (x_i\, f(x_{i-1}) - x_{i-1}\, f(x_i))/(f(x_{i-1}) - f(x_i)) \quad (i = 1, 2, \ldots). \tag{A2}$$

When this method converges to the simple zero x_t of f(x) we have

$$x_i - x_t \sim C(x_{i-1} - x_t)^r, \tag{A3}$$

where $r = (\sqrt{5} + 1)/2$ [10 pp. 100-101] and hence the convergence rate is better than the usual simple iteration method [10, p. 97].

However, the secant method may sometimes give a result which is outside of the interval (a, b) so that it converges to another root or even diverges. To avoid this, it is useful to combine the bisection method and the secant rule, essentially this is the Zero-in algorithm.

It is described as follows: For the given interval (b, c) which contains only the specified zero of f(x), we have $f(b)f(c) < 0$. Three points a_o, b_o and c_o are chosen as follows:

$$\text{If } |f(b)| \leq |f(c)| \text{ then } b_o = b, \; c_o = c, \; a_o = c_o. \tag{A4}$$

$$\text{If } |f(b)| > |f(c)| \text{ then } b_o = c, \; c_o = b, \; a_o = c_o. \tag{A5}$$

Then, at the beginning of the i-th stage, the three points a_i, b_i and c_i are involved and such that

$$f(b_i)f(c_i) < 0, \ |f(b_i)| \leq |f(c_i)|. \tag{A6}$$

The process in this stage is then as follows:

(1) Interpolate between a_i and b_i by (A2) to get P_i.

(2) Determine the mid-point m_i of b_i and c_i.

(3) If P_i is between b_i and m_i, then it is accepted as b_{i+1}. Otherwise m_i as accepted as b_{i+1}.

(4) Take $a_{i+1} = b_i$ and $c_{i+1} = c_i$.

(5) If b_{i+1} and c_{i+1} satisfy the conditions

$$f(b_i+1)f(c_{i+1}) < 0 \text{ and } |f(b_{i+1})| \leq |f(c_{i+1})|$$

then go back to (1) for the next stage, otherwise go to (6) to adjust the values of a_{i+1}, b_{i+1} and c_{i+1}.

(6) If $f(b_{i+1})f(c_{i+1}) > 0$ then we take $c_{i+1} = b_i$; this will ensure that $f(b_{i+1})f(c_{i+1}) < 0$.

(7) If $|f(b_{i+1})| > |f(c_{i+1})|$ then we interchange b_{i+1} and c_{i+1} and take a_{i+1} to be the value of new c_{i+1}. The right conditions now hold for the beginning of next stage. Of course, once we find b_i in each stage, we check to see whether to accept it as a reasonable root of the function $f(x)$.

The stopping criterion for the zero-in algorithm seems to be a bit involved. To use the criterion $|b_i - P_i| < \varepsilon$ is unreliable. For example in the case of Fig. A1 if $|f(b_o)| \ll |f(c_o)|$ the quantity $|b_o - P_o|$ will be very small although neither b_o nor P_o is near the required zero. Since the required zero is between b_i and c_i, to use $|b_i - c_i|$ as the stopping criterion seems all right, but unfortunately it may never be satisfied. See Fig. A2.

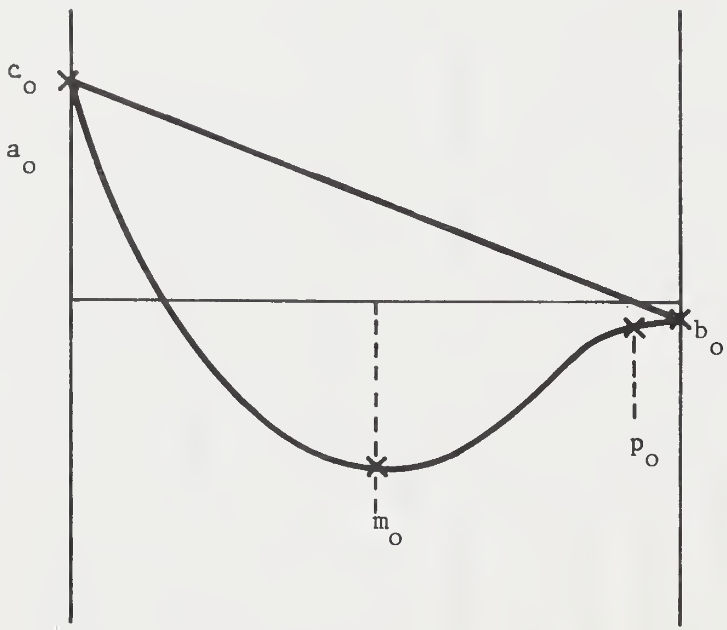


FIGURE A1. CASE IN WHICH THE CRITERION $|b_i - p_i| < \varepsilon$ IS UNRELIABLE

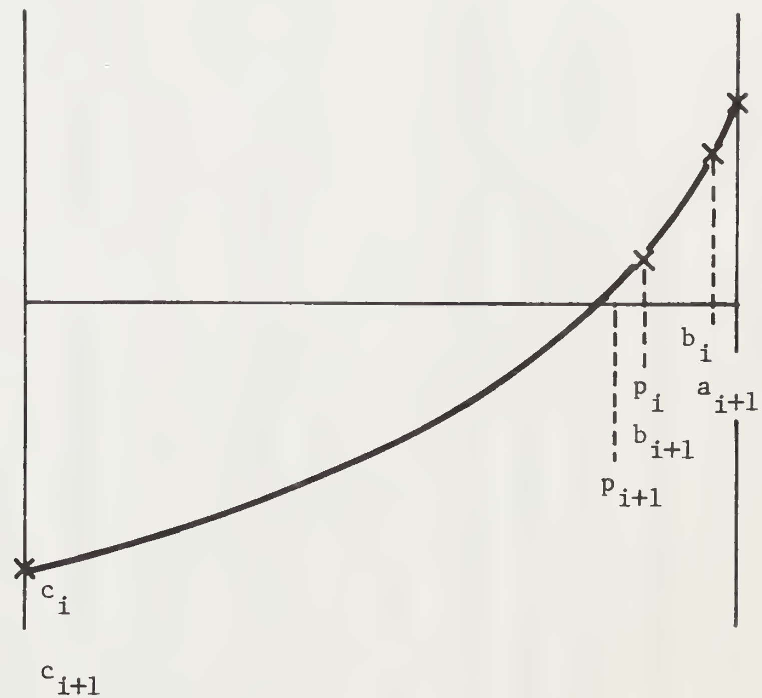


FIGURE A2. CASE IN WHICH THE CRITERION $|b_i - c_i| < \varepsilon$ CAN NEVER BE SATISFIED

To overcome the difficulty described above, Wijngaarden et. al. [11, Appendix] suggests a simple stratagem. Suppose the stopping criterion is $|b_i - c_i| < tol$. Then if $|P_i - b_i| < tol$, then P_i is replaced by $P_i + \text{sign}\,(c_i - b_i) \times tol$, this will ensure that a b_{i+1} is finally beyond the required zero and makes $f(b_{i+1})f(c_{i+1}) > 0$. Then c_{i+1} is switched as mentioned in process (7) and this immediately gives a b_{i+1} and c_{i+1} containing the zero and with $|b_{i+1} - c_{i+1}| < tol$. The choice of tol is experiemental and depends on the machine used.

APPENDIX II. GLYPNIR PROGRAMS

The ILLIAC IV GLYPNIR program, "MULTISEC" is listed below. It contains two major subroutines: "STURMPROC" and "ZEROIN". "STURMPROC" is used to obtain the intervals such that each of them contains exactly one of the eigenvalues of a symmetric tridiagonal matrix A in a given interval. If all the eigenvalues are required the interval is $(-||A||_\infty, ||A||_\infty)$. The "ZEROIN" routine is used to compute the eigenvalues contained in the intervals given by "STURMPROC". Switch "CHOICE" is given to provide the choice of using either one or both of "STURMPROC" and "ZEROIN". If only "ZEROIN" is used, the used should give the upper and lower bounds of each interval containing only one eigenvalue.

```
SUBROUTINE MULTISEC (CINT N,CREAL EPS,PCPOINT D,PCPOINT E,PCPOINT EIGL,
          PCPOINT EIGU,CINT ISW,CREAL HI,CREAL LO,CINT CHOICE,
            CINT MITER);
BEGIN
CINT  NM,MAXN1; LABEL CH1,CH2; PREAL A,B;  PE REAL VECTOR P[1];
% THIS SUBROUTINE CONSISTS OF TWO MAJOR SUBROUTINES: STURMPROC & ZEROIN.
% STURMPROC  FINDS ALL THE INTERVALS SUCH THAT EACH OF THEM CONTAINS
% EXACTLY ONE EIGENVALUE.  ZEROIN  FINDS ALL THE EIGENVALUES CONTAINED
% IN THE INTERVALS FOUND BY STURMPROC OR BY GIVEN.   EITHER ONE OR BOTH
% OF THEM CAN BE CALLED BY USING SWITCH:  CHOICE.
%    CHOICE=1 MEANS CALL STURMPROC ONLY.
%    CHOICE=2 MEANS CALL ZEROIN ONLY.
%    CHOICE=0 MEANS CALL BOTH.
% ISW=0 MEANS COMPLETE EIGENVALUE PROBLEM, OTHERWISE MEANS PARTIAL
% EIGENVALUE PROBLEM.
% MITER: THE MAX NUMBER OF ITERATIONS IN ZERO-IN  ALGORITHM.


SUBROUTINE STURMPROC;
BEGIN
  CU REAL NORM,H,TMP,TMP1;
  CU INTEGER I,J,K,L,VARUP,G,S;
  CU REAL VECTOR MOD,NOM[3];
  PE REAL PEL,TEMP;
  PE INTEGER AGR,VAR,NN,LLMT,KSW,FINISH,N1;
  PE REAL VECTOR UP,LW[10];
  PE INTEGER VECTOR VUP[10];
  LABEL  GOUT,RPT,FINS;
  %
  SUBROUTINE STURM;
  % FIND THE AGREEMENT OF SIGNS IN STURM SEQUENCE
  BEGIN
    CU REAL DD,PP;
    PE REAL P1,Q1,R1;
    LABEL ENDS;
    IF G=0 THEN
    BEGIN
      AGR:=0;
      TEMP:=D[0];
      DD:=GRABONE(TEMP,0);
      R1:=DD-PEL;
      P1:=1;  Q1:=R1;
      IF Q1 GEQ 0 THEN AGR:=AGR+1;
      LOOP L:=1,1,N-1 DO
      BEGIN
      % GRAB D[L] AND ESQ[L-1]
        J:=L DIV 64;
        TEMP:=D[J];
        DD:=GRABONE(TEMP,L);
        J:= (L-1) DIV 64;
        TEMP:=P[J];
        PP:=GRABONE(TEMP,L-1);
      %
        R1:=(DD-PEL)*Q1-PP*P1;
        P1:=Q1;  Q1:=R1;
        IF P1 GEQ 0 EQV Q1 GEQ 0 THEN AGR:=AGR+1
      END;      % END OF LOOP L
        IF Q1=0 AND P1>0 THEN AGR:=AGR-1
    END      ELSE
    BEGIN
```

```
    TEMP:=D[0];
    DD:=GRABONE(TEMP,0);
    Q1:=DD-PEL;     AGR:=0;
    IF Q1 GEQ 0 THEN AGR:=AGR+1;
    LOOP L:=1,1,N-1 DO
    BEGIN
    J:=L DIV 64;
    TEMP:=D[J];
    DD:=GRABONE(TEMP,L);
    J:=(L-1) DIV 64;
    TEMP:=P[J];     PP:=GRABONE(TEMP,L-1);
    TEMP:=E[J];     TMP:=GRABONE(TEMP,L-1);
    IF Q1 NEQ 0 THEN TEMP:=PP/Q1 ELSE TEMP:=ABS(TMP)/EPS;
    %
    Q1:=DD-PEL-TEMP;
    IF Q1 GEQ 0 THEN AGR:=AGR+1
    END;      % END OF LOOP L
    IF Q1=0 THEN AGR:=AGR-1
  END;
ENDS:
END;     % END OF SUBROUTINE STURM
SUBROUTINE IDENTIFY;
BEGIN
  MODE:=TRUE;
  MODE:=PEN<63 AND TRUE;
  VAR:=AGR-RTL(1,,AGR);
  MODE:=PEN=63 AND TRUE;
  IF ISW=1 THEN BEGIN ISW:=ISW-1;  VAR:=0 END ELSE VAR:=AGR-VARUP;
  MODE:=TRUE;
  FOR ALL VAR>1 DO
  BEGIN
    NN:=NN+1;
    LW[NN]:=PEL;
    UP[NN]:=PEL+H;
    VUP[NN]:=AGR-VAR;
    FINISH:=0;
  END;
  FOR ALL VAR=1 DO
  BEGIN
    N1:=N1+1;
    EIGL[N1]:=PEL;
    EIGU[N1]:=PEL+H;
  END;
END;     % END OF IDENTIFY
%
% PREPARE FOR MAIN PROG
% FIND THE NORM OF MATRIX
  K:= (N-1) DIV 64;
  IF K>0 THEN
  LOOP I:=0,1,K-1 DO MOD[I]:=64;
  MOD[K]:=N-K*64;
  LOOP I:=0,1,K DO BEGIN
    MODE:=TRUE;
    MODE:=PEN<MOD[I]   AND PEN>0 AND TRUE;
    P[I]:=ABS(D[I])+ABS(E[I])+ABS(RTR(1,,E[I])) END;
  MODE:=TRUE;      MODE:=PEN=0 AND TRUE;
  P[0]:=ABS(D[0])+ABS(E[0]);
  LOOP I:=1,1,K DO
    P[I]:=ABS(D[I])+ABS(E[I])+ABS(RTR(1,,E[I-1]));
  LOOP I:=0,1,K DO BEGIN
```

```
      MODE:=TRUE;  MODE:=PEN<MOD[K]   AND TRUE;
      NOM[I]:=MAX(P[I])  END;
    NORM:=NOM[0];
    LOOP I:=1,1,K DO
      IF NOM[I]>NORM THEN NORM:=NOM[I];
    SIMWRITE (LINE," NORM IS", 1 WORD REAL: (NORM));
  %
  % SEARCH FOR VERY SMALL E[I]
  LOOP I:=0,1,K DO
  BEGIN
    MODE:=TRUE;  MODE:=PEN<MOD[I] AND TRUE;
    KSW:=0;
    TEMP:=E[I];
    IF ABS(TEMP)/NORM < 1.0@-6  THEN KSW:=1;
    G:=MAX(KSW);
    IF G=1 THEN GO TO GOUT;
  END;
  GOUT:   % FIND P[I]=E[I]*E[I] TO BE USED IN STURM
  MODE:=TRUE;
  LOOP I:=0,1,K DO
  BEGIN
    IF E[I]=0 THEN P[I]:=EPS*NORM*NORM
              ELSE P[I]:=E[I]*E[I]
  END;
  LOOP I:=0,1,K DO  BEGIN
    EIGL[I]:=0.0;  EIGU[I]:=0.0  END;
  LOOP I:=0,1,10 DO BEGIN
    LW[I]:=0.0;  UP[I]:=0.0;  VUP[I]:=0  END;
  % BEGIN MAIN PROG
  % DISTRUBUTE LAMBDA TO ALL PES
  IF ISW=0 THEN BEGIN
  H:=NORM/32;      PEL:=-NORM+PEN*H;
  END ELSE BEGIN H:=(HI-LO)/63;  PEL:=LO+PEN*H END;
  VARUP:=0;  NN:=-1;  N1:=-1;  LLMT:=-1;  FINISH:=0;
  STURM;  IDENTIFY;
RPT:  MODE:=TRUE;
    IF LLMT GEQ NN THEN FINISH:=-(PEN+1) ELSE FINISH:=PEN;
    S:=MAX(FINISH);
    IF S<0 THEN GO TO FINS;
    MODE:=TRUE;    MODE:=PEN=S AND TRUE;
    LLMT:=LLMT+1;
    VARUP:=VUP[LLMT];
    TMP:=LW[LLMT];
    H:=(UP[LLMT]-LW[LLMT])/64;
    IF H<EPS THEN BEGIN
      TMP:=LW[LLMT];    TMP1:=UP[LLMT];
      SIMWRITE (LINE,"  SOME E-VALUES ARE TOO CLOSE TO IDENTIFY");
      SIMWRITE (LINE,"  THEY ARE IN THE INTERVAL:, LOWER",
          1 WORD REAL: (TMP)," UPPER", 1 WORD REAL: (TMP1));
      GO TO RPT   END;
    MODE:=TRUE;    PEL:=TMP+PEN*H;
  STURM;  IDENTIFY;
  GO TO RPT;
  % PRINT ALL INTERVALS WHICH CONTAIN SINGLE E-VALUE
  FINS:  MODE:=TRUE;    K:=MAX(N1);
  SIMWRITE  (LINE," INTERVALS CONTAIN 1 E-VALUE ARE");
  LOOP I:=0,1,K DO
  SIMWRITE  (LINE," EIGL " ,EIGL[I]);
  LOOP I:=0,1,K DO
  SIMWRITE  (LINE," EIGU " ,EIGU[I])  ;
```

```
  MAXN1:=K;
END;    % END  STURMPROC



SUBROUTINE ZEROIN(PREAL A, PREAL B);
BEGIN
  CREAL RTOL;
  CINT I,J,K;
  CU REAL VECTOR MOD[3];
  PREAL C,FA,FB,FC,INT,MID,TOL,TEMP;
  PINT ITR;
  PE REAL VECTOR DD,PP[12];
  LABEL ROOT;
  BOOLEAN SAVEMODE;
  %
  SUBROUTINE  FUN(PREAL X,PREAL OUT FX);
  BEGIN
    PREAL R,S;  CINT I;
    R:=1;  S:=DD[0]-X;
    LOOP I:=1,1,N-1 DO
    BEGIN
      FX:=(DD[I]-X)*S-PP[I-1]*R;
      R:=S;  S:=FX;
    END;
  END;  % END OF  SUBR  FUN
  % MOVE THE MATRIX INTO EACH PE
  K:=(N-1) DIV 64;
  IF K>0 THEN
  LOOP I:=0,1,K-1 DO MOD[I]:=63;
  MOD[K]:=N-K*64-1;
  MODE:=TRUE;
  LOOP I:=0,1,K DO
  BEGIN
    TEMP:=D[I];
    LOOP J:=0,1,MOD[I] DO
    DD[J+64*I]:=GRABONE(TEMP,J);
    TEMP:=P[I];
    LOOP J:=0,1,MOD[I] DO
    PP[J+64*I]:=GRABONE(TEMP,J);
  END;    % END LOOPI
  % BEGIN THE PROCESS OF ZEROIN
  ITR:=0;
  RTOL:=4*EPS;
  MODE:=TRUE;
  MODE:=A NEQ 0 OR  B NEQ 0 AND TRUE;    SAVEMODE:=MODE;
  FUN(A,FA);
  FUN(B,FB);
  C:=A;  FC:=FA;
  IF ABS(FB)=ABS(FC) THEN
  BEGIN
    A:=B;  FA:=FB;  B:=C;  FB:=FC;
    C:=A;  FC:=FA;
  END;
  MID:=(B+C)/2;
  TOL:=RTOL*ABS(B)+EPS;
  %
  WHILE ABS(MID-B)>TOL AND ITR<MITER DO
  BEGIN
  IF FA NEQ FB THEN INT:=(A*FB-B*FA)/(FB-FA)
```

```
              ELSE INT:=MID;
   IF ABS(INT-B)<TOL THEN INT:=B+SIGN(C-B)*TOL;
   A:=B;   FA:=FB;
   IF SIGN(INT-MID)=SIGN(B-INT) THEN B:=INT
                                ELSE B:=MID;
   FUN(B,FB);
   MODE:=ABS(FB)>4*N*EPS AND MODE;
   FOR ALL SIGN(FC)=SIGN(FB) DO
   BEGIN  C:=A;  FC:=FA  END;
   IF ABS(FB)=ABS(FC) THEN
   BEGIN
     A:=B;  FA:=FB;  B:=C;  FB:=FC;
     C:=A;  FC:=FA;
   END;
   MID:=(B+C)/2;
   TOL:=RTOL*ABS(B)+EPS;
   ITR:=ITR+1:
   END:
   %
ROOT:  MODE:=SAVEMODE;
  SIMWRITE (LINE," EIGENVALUES ARE",B);
  SIMWRITE (LINE,"ITER=", ITR)  ;
END:    % END OF ZEROIN
    IF CHOICE=2 THEN GO TO CH2;    STURMPROC;
    IF CHOICE=1 THEN GO TO CH1;
CH2: MODE:=TRUE;  LOOP NM:=0,1,MAXN1 DO
      BEGIN A:=EIGL[NM];  B:=EIGU[NM];  ZEROIN(A,B);  END;
CH1: SIMWRITE (LINE,"THIS IS THE END OF PROGRAM")
END:   % END OF MULTISEC ************************************
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CAC Document No. 109	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): A Parallel Algorithm for Symmetric Tridiagonal Eigenvalue Problems	5. TYPE OF REPORT & PERIOD COVERED: Research Report
	6. PERFORMING ORG. REPORT NUMBER: CAC-109
7. AUTHOR(s): Hui-Ming Huang	8. CONTRACT OR GRANT NUMBER(s): DAHC04-72-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Center for Advanced Computation, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order No. 1899
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia	12. REPORT DATE: February 1974
	13. NUMBER OF PAGES: 57
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): U. S. Army Research Office, Box CM, Duke Station, Durham, North Carolina 27706	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

No restrictions. Copies may be requested from the address in (9) above.

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

Approved for public release; distribution unlimited.

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

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

Parallel algorithm
QL algorithm
GLYPNIR
ILLIAC IV

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

It is well known that many problems in physical sciences give rise to the algebraic eigenvalue problem $Ax = \lambda x$ where A is usually a large symmetric matrix. The most effective way to solve such problems consists of reducing A to the tridiagonal form and using either the bisection method or the QL algorithm to find few or all the eigenvalues. The major goal of this work is to study and analyze both algorithms on a parallel machine, namely the ILLIAC IV, and find out the cases in which each algorithm should be used.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)