A COMPARISON OF THE PLU AND QR METHODS FOR DETERMINING EIGENVALUES OF REAL HESSENBERG MATRICES

by

J. T. Panttaja

Memorandum No. ERL-M440

May 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A COMPARISON OF THE PLU AND QR METHODS FOR DETERMINING EIGENVALUES OF REAL HESSENBERG MATRICES

by

J. T. Panttaja[†]

May 1974

[†] Mathematics Department

Present Address: IBM Corporation, Poughkeepsie, New York.

The author gratefully acknowledges partial support by the Office of Naval Research under Contract N00014-69-A-0200-1017. Reproduction in whole or part is permitted for any purpose of the United States Government.

CONTENTS

A COMPARISON OF THE PLU AND QR METHODS FOR DETERMINING EIGENVALUES OF REAL HESSENBERG MATRICES

J. T. Panttaja

ABSTRACT

The LU algorithm with interchanges (herein after PLU) should be substantially faster than the QR, based on an order n^2 count of floating point arithmetic operations, and is essentially stable. If this comparison held in actual executions of these two algorithms, and if the PLU provided convergence on most matrices, we would have a strong case for using the PLU instead of QR.

In actual timings of the algorithms on test matrices, the PLU algorithm did not yield as substantial a savings as was predicted by this traditional operation count. We study this anomaly here. A complete order n^2 count of all operations predicted results closer to those observed, but still were considerably off. The number of iterations required for the two methods, however, were comparable for each of the matrices tested.

The LU algorithm failed on some matrices, as predicted by the convergence theory. The PLU algorithm never failed on the matrices tested.

Eigenvalues found by the PLU and QR algorithms were of equivalent accuracy.

Key Words: Eigenvalues. LR and QR. Operation Counts. Timings.

1. Introduction

The eigenvalues of matrices are useful in many disciplines. A need has arisen, therefore, for methods with which to determine eigenvalues quickly and accurately.

In 1959 Rutishauser presented the LR (we will refer to it in this papaer as LU) method which utilizes a sequence of triangular decompositions of the matrix using elementary transformations. In 1961 Francis introduced the QR method which uses unitary triangular transformations [7].

By restricting ourselves to real matrices of Hessenberg form (Martin and Wilkinson present stable methods for reduction to Hessenberg form [3]), we can consider very fast variants of these methods: the double LU and double QR methods.

Each iteration of the double LU method requires $2n^2$ multiplications, while the double QR requires $5n^2$ multiplications per iteration (see Section 4). It has been suggested by Wilkinson that the number of iterations required for the LU method is consistently more than for the double QR [8]. Parlett and Wang have suggested that operation counts (particularly only of multiplications) may not be adequate in considering differences in running times of algorithms [6].

The LU method has stricter requirements for convergence, and is unstable. To eliminate stability problems, interchanges are introduced at each double step, to form the PLU method. Little is known about the convergence of this method (see Section 2). On the other hand, more is known about the convergence of the QR method (see Section 2).

With these ideas in mind, I have set out to consider three questions:

1. What is the difference in the number of iterations of the three algorithms?

2. What is the difference in the timing of the three algorithms?

3. How often do the LU and PLU algorithms fail?

2. Theoretical Presentation

We are considering methods which produce a sequence of similar matrices which usually tend to a form from which the eigenvalues may be recovered easily. In the case we are considering we have real matrices with possibly complex eigenvalues. Our methods will tend to block triangular form with 1x1 and 2x2 blocks on the diagonal corresponding to real eigenvalues and complex conjugate pairs [4].

The QR method produces a sequence of unitarily similar matrices. The LU method use elementary transformations. The PLU method introduces pivoting at each step in the LU algorithm.

In the following discussion, convergence refers to convergence to block triangular form, since the elements above the diagonal do not necessarily converge. For our methods, however, this essential convergence gives the eigenvalues in a convenient form.

For the basic LU algorithm (no origin shifts), convergence has been shown for $A = U \operatorname{diag}(l_1, l_2, \ldots, l_n)\ U^{-1}$ if we have eigenvalues of distinct modulus, and if both U and U^{-1} have LU decompositions [4]. This method can fail if at some point we cannot continue the triangular decomposition. The method is also unstable (division by small elements may lead to element growth) [8].

Introduction of interchanges at each step, as is done with triangular decomposition, takes care of both of these problems. Unfortunately, interchanges may continue after many steps, and convergence cannot be guaranteed [8].

The basic QR method converges when the eigenvalues are of distinct modulus [4].

We now note that the Hessenberg form is invariant under all three methods, and that if a matrix is real, it remains real under each of the three methods [8]. This has a significant impact on computation.

Shifts of origin may be introduced at each step. If chosen properly they improve convergence rates. In particular, this leads us to the double QR, LU and PLU methods. These involve combining two successive steps using shifts corresponding to the eigenvalues of the bottom 2x2 matrix on the diagonal. This retains real Hessenberg form and decreases the number of operations in each iteration [8].

The convergence theory of LU with shifts is limited. Parlett has shown convergence in a special case with restrictions on the shifts. These restrictions are not reasonable computationally [4].

A little more is known for the QR algorithm [5]: for symmetric matrices, with shifts by a_{nn}, we have convergence almost always.

In the normal, non-symmetric case, Buurema has shown convergence almost always with the standard double shift strategy [5].

There exists a k_0 such that if we change from no shift to the usual shift at the k_0 step we will always have convergence. However, k_0 cannot be estimated a priori.

We have, therefore, a convergence theory which is not nearly complete. We know very little about the PLU method.

3. Environment

The programs (see Appendix C) were patterned after the QR algorithm in the Eispack Library [2]. All programs were written in Fortran. The eispack balancing and reduction routines (BALANC and ELMHES) were used to reduce the test matrices to Hessenberg form (3,6).

The programs were run on a CDC 6400 using the RUN compiler and FTN 3.0, an experimental optimizing compiler provided by CDC (1971). The 6400 uses a 60 bit floating point word.

The programs were also run on an IBM 360/50 using the H compiler. Short words (32 bits floating point) were used.

Finally, the programs were run on a Honeywell 437. This machine uses a 48 bit floating point word.

See Appendix A for instruction execution times.

4. Iterations

Table 1 demonstrates that in the special matrices used, the number of iterations is quite comparable for the three methods. In some cases as many as 19 iterations were required to find a single eigenvalue, however, later eigenvalues took fewer iterations. We ended up with an average in each matrix from one to three iterations per eigenvalue. In the case of random matrices on the CDC 6400, the PLU method required 10 to 30% more iterations.

This oversimplifies the situation, because iterations at the beginning will take more time than iterations near the end (when the matrix being transformed is smaller). There did not appear to be any trend, with one method consistently requiring more, or fewer iterations with larger n.

These results are rather surprising. For positive definite matrices one QR step yields the same matrix as two LU steps (no shifts).

Since the number of iterations required varies in most cases less than 10%, and at worst by 35%, it is reasonable to evaluate the number of operations in one iteration in order to obtain an estimate of relative execution times.

5. Operation Counts

Each of the three methods being evaluated transforms a Hessenberg matrix into another Hessenberg matrix, in an attempt to deflate it. We will evaluate here the number of operations involved in one iteration (one of these transformations).

It has been traditional in considering operation counts to evaluate only multiplications, and perhaps additions. Since we are going to give a more detailed account later, we quote here the standard multiplication counts (nxn matrix) [8]:

Double LU $2n^2$

Double PLU $2n^2$

Double QR $5n^2$

There are, however, many machines where the execution times of the arithmetic operations are not significantly longer than other operations. Also, Parlett and Wang have suggested that different compilers will affect the comparative timings of algorithms [7]. For these reasons I have done

TABLE 1

ITERATIONS - CDC 6400

Order	HQR	HLU	HPLU	
12	0	0	0	a
12	13	14 b	13	
12	14	14	14	
12	12	13	13	
12 (5.8 & 5.4)	16	15	17	e
12 (5.5 & 5.4)	16	15	15	e
15 (5.20)	16	15	15	e
15 (5.21)	21	c	31	e
15 (5.22)	41	c	41	e
18 (5.12 & 5.1)	44	48	44	de
20 (5.23)	33	60 b	26	e
24	39	36	41	
59	87	c	89	
RANDOM MATRICES				
25	43	49	58	
25	50	58	57	
50	89	111	118	
50	87	109	96	

a. All eigenvalues isolated by BALANC.
b. Norm very large, eigenvalues not even agreeing to 1 decimal place.
c. HLU broke down.
d. Triple eigenvalues agreement to only 5 decimal places.
e. Reference to example number in Gregory and Karney [1].

the operation counts for the methods, as compiled for the four compilers used, and considered actual timings of operations.

Detailed Order n^2 Count

In considering an order n^2 operation count, we need only count the inner loops in the routines. In the QR and LU algorithms we have first a DO loop which modifies rows M through EN (Loop 1) (see Figure 1), and second a DO loop which modifies columns L through EN. In the PLU algorithm we add a DO loop which interchanges two rows of the matrix (Loop 3), and one which interchanges two columns of the matrix (Loop 4).

In the following calculations, n is the dimension of the matrix being transformed, and we assume L = M (n = EN - L + 1 = EN - M + 1) (see Figure 1).

For an operation which is executed once in a typical iteration of Loop 1, we have:

$(\frac{n^2}{2} + \frac{n}{2} - 3)$ plus two times the number of occurrences in each of the last two steps.

For an operation which is executed once in a typical occurrence of Loop 2, we have:

$(\frac{n^2}{2} + 3\frac{n}{2} - 6)$ plus n times the number in the last step.

Loops 3 and 4 correspond to Loops 1 and 2 respectively.

For an n^2 evaluation we need only consider the number of steps involved in a typical iteration. See Table 2 (summary of Appendix D).

FIGURE 1

VARIABLES

x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x
0	x	x	x	x	x	x	x	x	x
0	0	a	x	x	x	x	x	x	x
0	0	0	x	x	x	x	x	x	x
0	0	0	0	b	x	x	x	x	x
0	0	0	0	0	c	x	x	x	x
0	0	0	0	0	0	x	x	x	x
0	0	0	0	0	0	0	0	x	x
0	0	0	0	0	0	0	0	x	x

a small compared to A(3,3), A(4,4)

b, c small compared to A(5,5), A(6,6), A(7,7)

L = 4

M = 6

EN = 8

TABLE 2

SUMMARY OF OPERATION COUNTS

CDC 6400 FTN — minor cycles times $\frac{n^2}{2}$

QR	%	LU	%	PLU	%	Predicted by Multiplication Count %
1223	100	613	50	797	65	40

CDC 6400 RUN — minor cycles times $\frac{n^2}{2}$

QR	%	LU	%	PLU	%	%
2782	100	1604	58	2567	92	40

IBM 360/50 H — microseconds times $\frac{n^2}{2}$

QR	%	LU	%	PLU	%	%
444.56	100	225.66	51	289.91	65	40

Honeywell 437 — microseconds times $\frac{n^2}{2}$

QR	%	LU	%	PLU	%	%
1916.6	100	974.0	51	1654.4	86	40

Note that the percentage of the execution time attributed to floating point multiplies varies from less than 10% of the total for the Honeywell 437, to 37% for the FTN compiler on the CDC 6400.

The CDC computer does not have a fixed point multiply. The FTN compiler does array subscripting by updating previous accesses using additions. The RUN compiler requires additional floating point multiplies in the inner loops.

In the PLU algorithm, I am assuming that interchanges will be required during each iteration. If there is no interchange, the n^2 operation count reverts to the LU algorithm count.

These operation counts suggest that rather than the traditional ratios for LU or PLU to QR, we get the values in Table 2.

Order n Operation Count

The FTN compiler on the CDC 6400 does a large amount of the work for array subscripting outside the loop generated to represent a DO loop. For that reason I have chosen to do an order n operation count for this compiler, in the hope of gaining a better estimate of execution time.

The first loop within an iteration of each of the three methods is identified in the tables, (see Appendix E), as 80 (since it is the body of the statement "DO 80"). It finds the first "small" subdiagonal element. This determines the value of "L". Each instruction in this loop is executed EN - L times.

The next loop, 140, finds two consecutive relatively small elements, and determines the value of "M". Each instruction in this loop is executed EN - M - 2 times.

The final loop is the one which actually does the transformation (QR = 260, LU and PLU = 300). It includes the initialization for inner loops, 1, 2, 3 and 4 described in the previous section. Each instruction in these loops is executed EN - M - 2 times. The inner loops 1 and 3 are executed $\left(\left(\frac{EN-M+1}{2}\right)^2 + \left(\frac{EN-M+1}{2}\right)\right)$ times.

The inner loops 2 and 4 are executed

$$\left(\frac{EN^2}{2} + \frac{5}{2}\ EN-L*EN+M*L+L-4-\frac{7}{2}M - \frac{M^2}{2}\right) \text{ times.}$$

In loops 2 and 4, we have an additional EN + 1 - L iterations of the sub-loop executed when K = EN - 1 (which occurs at the end of the loop 260 (or 300)).

I include a computation with N = 25, L = M = 1. For one iteration on this matrix we see that the order n contribution is about one third of the total. This suggests that, particularly for smaller submatrices, the order n contribution will be significant. The formulas suggest further, that if L and M are much bigger than one, the execution times will be significantly changed. (Notice that this appears to be more sensitive to M.)

6. Results

As can be seen by looking at Table 23 (Appendix F), the percent of time required for LU as opposed to QR using FTN, varies between 48% and 76%. The percent required for PLU as opposed to QR varies between 60% and 82%.

Table 24 shows a range of 46% to 97% for LU as opposed to QR using FTN. The percent for PLU as opposed to QR varies between 62% and 93%.

Even the operation counts do not suggest this difference. We first note that the number of iterations is about comparable for each matrix, although a potential source of difference is the number of iterations at each n (see section 4), or the values of L and M (see section 5).

Fitting cubics to the values for random matrices on the CDC 6400, we get:

QR RUN

$$.090n^3 - .283n^2 + 56.3n - 379.$$

FTN

$$.035n^3 - .071n^2 + 33.8n - 314.$$

PLU RUN

$$.114n^3 + .327n^2 + 51.7n - 511.$$

FTN

$$.056n^3 - .708n^2 + 51.7n - 378.$$

Both compilers yield high n^2 and n coefficients.

TABLE 3

Summary of table for order n

Loop	Times	QR	LU	PLU
		minor cycles		
80	EN-L	649	649	649
140	EN-M-2	2131	2131	2131
160	EN-M-2	135	135	135
230 (300)	EN-M-2	3436	2104	3900
1 & 3	$((\frac{EN-M+1}{2})^2 + (\frac{EN-M+1}{2}))$			
		609	309	401
2 & 4	$(\frac{EN^2}{2} + \frac{EN}{2} - L*EN+M*L+L-4-\frac{7M}{2} - \frac{M^2}{2})$			
		614	304	396
2 & 4	EN+1-L	396	183	267

TABLE 4

Sample Evaluation of Order n Counts

EN = 25 M = 1 L = 1

		minor cycles	
Loop	QR	LU	PLU
80	15576	15576	15576
140, 160, 260, (300)	125444	96140	135652
1 & 3	197925	100425	130325
2 & 4 (n^2)	211216	106296	136224
2 & 4 (n^2)	9900	4575	6675
Total	560061 100%	323012 58%	424452 76%

It appears that an order n^2 operation count may be inadequate, and perhaps a large order n component cannot be ignored. Further, even an order n operation count involves so many assumptions, that it is inadequate.

Whereas the previous evaluations suggested a timing of 40% for LU, or PLU compared to QR: I have proposed about 51% and 66% respectively, using FTN, or 58% and 93%, using RUN. However, the values for actual observations are about 64% for LU and 70% for PLU, and 65% and 74%, using RUN.

7. Failure

It is known that the LU algorithm can break down, and is unstable, and we question convergence in the PLU algorithm. How did these problems manifest themselves in actual runs?

The LU algorithm failed in four matrices due to break down in the algorithm. In two other cases the LU algorithm gave results which didn't even agree to one decimal place with the results of QR and PLU. In these two cases, observing the following norm - square root of the sum of the squares of the elements - , we noted an increase of several orders of magnitude in norm during execution.

In all but one other case we had agreement to 10 decimal places (relative to the largest eigenvalue). In that one example we had a triple eigenvalue, and had an agreement of 5 decimal places, which is as good as can be hoped for (cube root of round off error) [8].

8. Conclusions

The evaluation of the QR, LU and PLU methods in a practical investigation has produced some interesting results, and posed new questions.

I have found that the number of iterations required for the three methods are about the same on each of the matrices considered.

With an operation count (based on the generated code from the FTN 3.0 compiler) we expect ratios of 51:100 for LU:QR and 66:100 for PLU:QR in timing. These values are probably impossible to predict, and vary from compiler to compiler (in particular, they are quite different using the RUN compiler or on an IBM 360/50 or Honeywell 437). They are significantly different from the values predicted by considering only multiplications and additions. By investigating, we see that the average of actual comparisons are 64:100 for LU:QR and 70:100 for PLU:QR. (The expected values based on operation counts, and the actual values using a non-optimizing compiler were less favorable for LU and PLU.)

The failure for LU was as predicted. The PLU algorithm (which has little convergence theory) never failed on the matrices tested.

APPENDIX A

Instruction Execution Times

CDC 6400

IBM 360/50

Honeywell 437

Instruction Execution Times

CDC 6400

Ten minor cycles are performed in 1 microsecond.

Operation	Minor Cycles
Fetch	12
Store	10
Compare branch	13
no branch	5
Integer add	5
Floating add	11
Floating multiply	57
Floating divide	57
Boolean operation	5
Normalize	7
Pack	7
Unpack	7
Count	68
Integer add (a)	6
Increment	6
Shift	6
Return jump	21
No op	3

Instruction Execution Times

IBM 360/50

Arithmetic timings are average values.

Operation		Microseconds
Load	LR	2.5
Load (short)	LE	4.0
Load (short)	LER	2.75
Load	L	4.0
Load Complement	LCER	3.75
Store (short	STE	4.0
Add	A	4.0
Add Normalized	AE	8.73
Add Normalized	AER	7.97
Add	AR	3.25
Subtract	SR	3.25
Subtract Normalized	SER	8.75
Multiply	ME	21.5
Multiply	MER	20.75
Compare no branch	BXLE	4.5
branch		5.5
Compare branch	BC	3.0
Compare	C	3.25

Instruction Execution Times

Honeywell 437

Times for arithmetic operations are average times.

Operations		Microseconds
Store	floating	8.4
	fixed	5.6
Load	floating	8.4
	fixed	5.6
Add	floating	12.25
	fixed	5.6
Multiply	floating	17.85
	fixed	15.05
Compare	fixed	5.6
Branch	greater	2.8
	unconditional	2.8
	zero	5.6
Exchange A-Q		2.8
Shift binary		4.55
Index		2.8
Index Pointer		5.6

APPENDIX B

Test Matrices

Two different sets of matrices were used. The first were specific, predetermined matrices. Some of these were taken from Gregory and Karney [1]. These have been identified in the tables. These matrices were only used on the CDC computer.

The second set of matrices are matrices with randomly generated elements. The order of these matrices vary on the different computers considered, due to memory considerations.

APPENDIX C

Programs

HQR (From the Eispack Library NATS Project
Argonne National Lab., Illinois 60440)

HLU (Omitted)

HPLU (Listing Given)

```
      SUBROUTINE HPLU(NM,NR,LOW,IGH,H,WR,WI,IERR)
      REAL H(NM,NM),WR(NM),WI(NM)
      INTEGER EN
      INTEGER ENM2
      LOGICAL NOTLAS
      REAL MACHEP
C     THIS SUBROUTINE  FINDS THE EIGENVALUES OF A REAL
C     UPPER HESSENBERG MATRIX BY THE LU METHOD.
C     ON INPUT-
C          NM MUST BE SET TO THE ROW DIMENSION OF TWO DIMENSIONAL
C            ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C            DIMENSION STATEMENT,
C          N IS THE ORDER OF THE MATRIX
C          LOW AND IGH ARE INTEGERS DETERMINED BY THE BALANCING
C            SUBROUTINE BALANC. IF BALANC HAS NOT BEEN USE,
C            SET LOW=1, IGH=N,
C          H CONTAINS THE UPPER HESSENBERG MATRIX. INFORMATION ABOUT
C            THE TRANSFORMATION USED IN THE REDUCTION TO HESSENBERG
C            FORM BY ELMHES OR ORTHES, IF PERFORMED, IS STORED
C            IN THE REMAINING TRIANGLE UNDER THE HESSENBERG MATRIX.
C     ON OUTPUT-
C          H HAS BEEN DESTROYED. THEREFORE, IT MUST BE SAVED
C            BEFORE CALLING HQR IF SUBSEQUENT CALCULATION AND
C            BACK TRANSFORMATION OF EIGENVECTORS IS TO BE PERFORMED,
C          WR AND WI CONTAIN THE REAL AND IMAGINARY PARTS,
C            RESPECTIVELY, OF THE EIGENVALUES. THE EIGENVALUES
C            ARE UNORDERED EXCEPT THAT COMPLEX CONJUGATE PAIRS
C            OF VALUES APPEAR CONSECUTIVELY WITH THE EIGENVALUE
C            HAVING THE POSITIVE IMAGINARY PART FIRST. IF AN
C            ERROR EXIT IS MADE, THE EIGENVALUES SHOULD BE CORRECT
C            FOR INDICES IERR+1,...,N,
C          IERR IS SET TO
C            ZERO       FOR NORMAL RETURN,
C            J          IF THE J-TH EIGENVALUE HAS NOT BEEN
C            DETERMINED AFTER 30 ITERATIONS.
C     ***** MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C           THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC.
      MACHEP=2.**(-47)
      IERR=0
C     ********** STORE ROOTS ISOLATED BY BALANC **********
      DO 50 I=1,NR
      IF(I.GE.LOW.AND.I.LE.IGH) GO TO 50
      WR(I)=H(I,I)
      WI(I)=0.0
   50 CONTINUE
      EN=IGH
      T=0.
C     **********SEARCH FOR NEXT EIGENVALUES **********
   60 IF(EN.LT.LOW) GO TO 1001
      ITS=0
      NA=EN-1
      ENM2=NA-1
C     ********** LOOK FOR SINGLE SMALL SUB-DIAGONAL ELEMENT
```

```
C                    FOR L=EN STEP -1 UNTIL LOW DO -- **********
   70 CONTINUE
      DO 80 LL=LOW,EN
      L=EN + LOW - LL
      IF(L.EQ.LOW) GO TO 100
      IF(ABS(H(L,L-1)).LE.MACHEP * (ABS(H(L-1,L-1))
     -     + ABS(H(L,L)))) GO TO 100
   80 CONTINUE
C     ********** FORM SHIFT **********
  100 X=H(EN,EN)
      IF(L.EQ.EN) GO TO 370
      Y=H(NA,NA)
      W=H(EN,NA) * H(NA,EN)
      IF(L.EQ.NA) GO TO 380
      IF(ITS.EQ.30) GO TO 1000
      IF(ITS.NE.10.AND.ITS.NE.20) GO TO 130
C     **********FORM EXCEPTIONAL SHIFT **********
      T=T + X
      DO 120 I=LOW,EN
  120 H(I,I)=H(I,I) - X
      S=ABS(H(EN,NA)) + ABS(H(NA,ENM2))
      X=.75*S
      Y=X
      W=-0.4375 * S*S
  130 CONTINUE
      ITS=ITS + 1
C     ********** LOOK FOR TWO CONSECUTIVE SMALL
C                SUB-DIAGONAL ELEMENTS.
C                FOR M=EN-2 STEP -1 UNTIL L DO -- **********
      DO 140 MM=L,ENM2
      M=ENM2 + L -MM
      ZZ=H(M,M)
      R=X-ZZ
      S=Y-ZZ
      P=(R*S - W)/H(M+1,M) + H(M,M+1)
      Q=H(M+1,M+1) -ZZ -R -S
      R=H(M+2,M+1)
      S=ABS(P) + ABS(Q) + ABS(R)
      P=P/S
      Q=Q/S
      R=R/S
      IF(M.EQ.L) GO TO 150
      IF(ABS(H(M,M-1)) * (ABS(Q) + ABS(R)) .LE. MACHEP * ABS(P)
     -  *(ABS(H(M-1,M-1)) + ABS(ZZ) + ABS(H(M+1,M+1)))) GO TO 150
  140 CONTINUE
  150 CONTINUE
      MP2=M+2
      DO 160 I=MP2,EN
      H(I,I-2)=0.
      IF(I.EQ.MP2) GO TO 160
      H(I,I-3)=0.
  160 CONTINUE
C     ********** DOUBLE LU STEP INVOLVING ROWS L TO EN AND
```

```
C                    COLUMNS M TO EN **********
      DO 300 K=M,NA
      NOTLAS=K.NE.NA
      J=MINO(EN,K+3)
      IF(K.EQ.M) GO TO 170
      P=H(K,K-1)
      Q=H(K+1,K-1)
      R=0.0
      IF(NOTLAS) R=H(K+2,K-1)
      X=ABS(P) + ABS(Q) + ABS(R)
      IF(X.EQ.0.0) GO TO 300
      P=P/X
      Q=Q/X
      R=R/X
  170 CONTINUE
C     ********** DETERMINE PIVOT **********
      IPIV=0
      P2=ABS(P)
      Q2=ABS(Q)
      IF(P2.GE.Q2) GO TO 195
      P2=Q2
      P3=Q
      IPIV=1
  195 R2=ABS(R)
      IF(P2.GE.R2) GO TO 197
      P3=R
      R=P
      IPIV=2
  197 IF(IPIV.EQ.0) GO TO 610
      IF(IPIV.EQ.1) Q=P
      P=P3
      IPIV2=IPIV + K
      IF(K.NE.M) H(K,K-1)=H(IPIV2,K-1)
C     ********** INTERCHANGE ROWS K AND IPIV2 **********
      DO 501 I=K,EN
      TEMP=H(IPIV2,I)
      H(IPIV2,I)=H(K,I)
      H(K,I)=TEMP
  501 CONTINUE
C     **********  INTERCHANGE COLUMNS K AND IPIV2 **********
      DO 601 I=L,J
      TEMP=H(I,IPIV2)
      H(I,IPIV2)=H(I,K)
      H(I,K)=TEMP
  601 CONTINUE
  610 CONTINUE
      IF(P.EQ.0.) GO TO 300
      Q=Q/P
      R=R/P
C     ********** ROW MODIFICATION **********
      DO 210 I=K,EN
      P=H(K,I)
      IF(.NOT.NOTLAS) GO TO 200
```

```
      H(K+2,I)=H(K+2,I) -R*P
  200 H(K+1,I)=H(K+1,I) -Q*P
  210 CONTINUE
C     ********** COLUMN MODIFICATION **********
      DO 230 I=L,J
      P=Q*H(I,K+1)
      IF(.NOT.NOTLAS) GO TO 220
      P=P + R*H(I,K+2)
  220 H(I,K)=H(I,K) + P
  230 CONTINUE
  300 CONTINUE
      GO TO 70
C     ********** ONE ROOT FOUND **********
  370 WR(EN)=X + T
      WI(EN)=0.
      EN=NA
      GO TO 60
C     ********** TWO ROOTS FOUND **********
  380 CONTINUE
      P=(Y-X)/2.
      Q=P*P + W
      ZZ=SQRT(ABS(Q))
      X=X+T
      IF(Q.LT.0.) GO TO 420
C     ********** REAL PAIR **********
      ZZ=P + SIGN(ZZ,P)
      WR(NA)=X + ZZ
      WR(EN)=WR(NA)
      IF(ZZ.NE.0.0) WR(EN)=X-W/ZZ
      WI(NA)=0.
      WI(EN)=0.
      GO TO 430
C     ********** COMPLEX PAIR **********
  420 WR(NA)=X+P
      WR(EN)=X + P
      WI(NA)=ZZ
      WI(EN)=-ZZ
  430 CONTINUE
      EN=ENM2
      GO TO 60
C     ********** SET ERROR -- NO CONVERGENCE TO AN
C                EIGENVALUE AFTER 30 ITERATIONS **********
 1000 IERR=EN
 1001 RETURN
C     ********** LAST CARD OF HPLU **********
      END
```

APPENDIX D

Order n^2 Operation Counts

TABLE 5

QR - Operation Counts

FTN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
fetches	13	156	13	156
Boolean	1	5	1	5
stores	5	50	5	50
integer adds	1	5	2	10
compare branch	1	13	1	13
no branch	1	5	1	5
multiplies	5	285	5	285
adds	5	55	5	55
normalize	5	35	5	35

This gives a total of $\frac{1223}{2} n^2$ minor cycles.

RUN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
fetches	27	324	32	384
Boolean	10	50	5	25
stores	7	70	6	60
integer adds	17	85	17	85
compares branch	1	13	1	13
no branch	1	5	1	5

Table 5 Continued.....

RUN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
multiplies	8	456	11	627
adds	5	55	5	55
normalize	5	35	5	35
integer add (a)	1	6	1	6
pack	6	42	12	84
increment	5	30	16	96
count	1	68	1	68

This give a total of $\frac{2782}{2} n^2$ minor cycles.

TABLE 6

LU - Operation Counts

FTN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
fetches	8	96	8	96
Boolean	2	10	1	5
stores	3	30	3	30
integer adds	1	5	1	5
compares branch	1	13	1	13
no branch	1	5	1	5
multiplies	2	114	2	114
adds	2	22	2	22
normalize	2	14	2	14

This gives a total of $\frac{613}{2} n^2$ minor cycles.

RUN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
fetches	15	180	17	204
Boolean	6	30	4	20
stores	6	60	4	40
integer adds	8	40	9	45
compares branch	1	13	1	13
no branch	1	5	1	5
multiplies	5	285	5	285
adds	2	22	2	22

Table 6 Continued.....

RUN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
normalize	2	14	2	14
integer adds (a)	-		-	
pack	6	42	6	42
increments	4	24	9	63
counts	1	68	1	68

This gives a total of $\frac{1604}{2} n^2$ minor cycles.

TABLE 7

FTN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
fetches	2	24	2	24
Boolean	3	15	3	15
stores	3	30	3	30
integer adds	2	10	2	10
compares branch	1	13	1	13

This gives a total (with total from Table 2) of $\frac{797}{2} n^2$ minor cycles.

RUN

Operation	Loop 1 Number	Cycles	Loop 2 Number	Cycles
fetches	9	108	7	84
Boolean	5	25	4	20
stores	5	50	4	40
integer adds	10	50	10	50
compares branch	1	13	1	13
multiplies	2	114	4	228
integer adds (a)	2	12	2	12
packs	4	28	8	56
increment	4	24	6	36

This gives a total (with total from Table 6) of $\frac{2567}{2} n^2$ minor cycles.

TABLE 8

QR - Operation Counts

360/50 H Compiler

Operation	Loop 1 Number	Microseconds	Loop 2 Number	Microseconds
LR	2	5.	2	5.
LE	2	8.	3	12.
LER	5	13.75	4	11.
LCER	2	7.5	2	7.5
STE	3	12.	3	12.
A	2	8.	-	
AE	3	26.19	2	17.46
AER	1	7.97	2	15.94
AR	2	6.5	3	9.75
SR	1	3.25	1	3.25
SER	1	8.75	1	8.75
ME	5	107.5	5	107.5
BXLE branch	1	5.5	1	5.5
no branch	1	4.5	1	4.5

This gives a total of $\frac{444.56}{2} n^2$ microseconds

TABLE 9

LU - Operation Counts

360/50 H Compiler

Operation	Loop 1 Number	Microseconds	Loop 2 Number	Microseconds
L	1	4.	-	
LR	2	5.	2	5.
LE	1	4.	2	8.
LER	2	5.5	1	2.75
LCER	2	7.5	-	
STE	2	8.	1	4.
AE	2	17.46	1	8.73
AER	-		1	7.97
AR	5	16.25	3	8.75
SR	1	3.25	1	3.25
ME	1	21.5	1	21.5
MER	1	20.75	1	20.75
BXLE branch	-		1	5.5
no branch	1	4.5	1	4.5
C	1	3.25	-	
BC	1	3.	-	

This gives a total of $\frac{225.66}{2} n^2$ microseconds

TABLE 10

PLU - Operation Counts

360/50 H Compiler

Operation	Loop 3 Number	Microseconds	Loop 4 Number	Microseconds
LR	1	2.5	1	2.5
LE	2	8.	2	8.
STE	2	8.	2	8.
AR	3	9.75	2	6.5
BXLE branch	1	5.5	1	5.5

This gives a total (with total from Table 9) of $\frac{289.91}{2} n^2$ microseconds

TABLE 12

LU - Operation Counts

Honeywell 437

Operation	Loop 1 Number	Microseconds	Loop 2 Number	Microseconds
Store Floating	7	58.8	4	33.6
Fixed	11	61.6	9	50.4
Load Floating	7	58.8	4	33.6
Fixed	13	72.8	11	61.6
Add Floating	2	24.5	2	24.5
Fixed	12	67.2	9	50.4
Multiply Floating	2	35.7	2	35.7
Fixed	5	75.25	4	60.2
Compare Fixed	1	5.6	1	5.6
Branch Greater	1	2.8	1	2.8
Unconditional	1	2.8	1	2.8
Zero	2	11.2	2	11.2
Exchange A-Q	5	14	4	11.2
Shift Binary	5	22.75	4	18.2
Index	3	8.4	4	11.2
Index Pointer	3	16.8	4	22.4

This gives a total of $\frac{974.0}{2} n^2$ microseconds

TABLE 11

QR - Operation Counts

Honeywell 437

Operation	Loop 1 Number	Microseconds	Loop 2 Number	Microseconds
Store Floating	12	100.8	11	92.4
Fixed	19	106.4	19	106.4
Load Floating	12	100.8	11	92.4
Fixed	21	117.6	21	117.6
Add Floating	5	61.25	2	24.5
Fixed	20	112.	17	95.2
Multiply Floating	5	89.25	5	89.25
Fixed	9	135.45	9	135.45
Compare Fixed	1	5.6	1	5.6
Branch Greater	1	2.8	1	2.8
Unconditional	1	2.8	1	2.8
Zero	2	11.2	2	11.2
Exchange A-Q	9	25.2	9	25.2
Shift Binary	9	40.95	9	40.95
Index	6	16.8	9	25.2
Index Pointer	6	33.6	9	50.4

This gives a total of $\frac{1916.6}{2} n^2$ microseconds

TABLE 13

PLU - Operation Counts

Honeywell 437

Operation	Loop 3 Number	Microseconds	Loop 4 Number	Microseconds
Store Floating	3	25.2	3	25.2
Fixed	9	50.4	9	50.4
Load Floating	3	25.2	2	25.2
Fixed	9	50.4	9	50.4
Add Floating	-			
Fixed	9	50.4	9	50.4
Multiply Floating	-			
Fixed	4	60.2	4	60.2
Compare Fixed	1	5.6	1	5.6
Branch Greater	1	2.8	1	2.8
Unconditional	1	2.8	1	2.8
Exchange A-Q	4	11.2	4	11.2
Shift Binary	4	18.2	4	18.2
Index	4	11.2	4	11.2
Index Pointer	4	22.4	4	22.4

This gives a total (with total from Table 12) of $\frac{1654.4}{2} n^2$ microseconds

APPENDIX E

Order n Operation Counts

FTN Compiler

TABLE 14

Order n 80

Instruction	Number	Minor Cycles
Fetch	15	180
Store	3	30
Compare no branch	2	10
branch	1	13
Integer add	1	5
Floating add	2	22
Floating multiply	4	228
Boolean	7	35
Normalize	2	14
Pack	4	28
Integer add (a)	8	48
Increment	1	6
Mask	1	6
Shift	3	18
No op	2	6

For a total of 649 minor cycles

TABLE 15

Order n 140

Instruction	Number	Minor Cycles
Fetch	43	516
Store	10	100
Compare no branch	2	10
branch	1	13
Floating add	13	143
Floating multiply	16	912
Boolean	22	110
Normalize	15	105
Pack	12	84
Integer add (a)	11	66
Mask	1	6
Shift	10	60
Increment	1	6

For a total of 2131 minor cycles

TABLE 16

Order n 160

Instruction	Number	Minor Cycles
Fetch	3	36
Store	3	30
Compare no branch	1	5
branch	1	13
Integer add	2	10
Integer add (a)	3	18
Boolean	1	5
Mask	2	12
Increment	1	6

For a total of 135 minor cycles

TABLE 17

Order n 260-QR

Instruction	Number	Minor Cycles
Fetch	56	672
Store	19	190
Compare no branch	5	25
branch	1	13
Integer add	15	75
Floating add	5	55
Floating multiply	22	1254
Boolean	19	95
Normalize	5	35
Pack	16	112
Integer add (a)	28	168
Mask	5	30
Shift	6	36
No op	8	24
Increment	9	54
Square root	1	598

For a total of 3436 minor cycles

TABLE 18

Order n 300-LU

Instruction	Number	Minor Cycles
Fetch	46	552
Store	13	130
Compare no branch	4	20
branch	1	13
Integer add	16	80
Floating add	4	44
Floating multiply	14	798
Boolean	15	75
Normalize	4	28
Pack	14	98
Integer add (a)	25	150
Mask	5	30
Shift	4	24
No op	5	15
Increment	8	48

For a total of 2104 minor cycles

TABLE 19

Order n 300-PLU

Instruction	Number	Minor Cycles
Fetch	84	1008
Store	24	240
Compare no branch	7	35
branch	3	39
Integer add	22	110
Floating add	6	66
Floating multiply	26	1482
Boolean	31	155
Normalize	6	42
Pack	27	189
Integer add (a)	48	288
Mask	9	72
Shift	7	42
No op	10	30
Increment	17	102

For a total of 3900 minor cycles

TABLE 20

Loop 2 Order n

QR

Instruction	Number	Minor Cycles
Fetch	9	108
Store	3	30
Boolean	1	5
Integer add	2	10
Compare no branch	1	5
branch	1	13
Floating multiply	3	171
Floating add	3	33
Normalize	3	21

For a total of 396 minor cycles

LU

Instruction	Number	Minor Cycles
Fetch	5	60
Store	2	20
Boolean	1	5
Integer add	1	5
Compare unsuccessful	1	5
successful	1	13
Floating Multiply	1	57
Floating add	1	11
Normalize	1	7

For a total of 183 minor cycles

TABLE 21

Loop 4 Order n-PLU

Instruction	Number	Minor Cycles
Fetch	2	24
Store	3	30
Boolean	3	15
Integer adds	2	10
Compare no branch	1	5

For a total (with Loop 2) of 267

TABLE 22

Square Root Subroutine-SQRT

(Time for one call)

Instruction	Number	Minor Cycles
Fetch	5	60
Boolean	1	5
Integer add	6	30
Compares no branch	2	10
branch	2	26
Floating multiply	3	171
Floating add	7	77
Integer add (a)	1	6
Floating divide	2	114
Increment	2	12
Normalize	3	21
Pack	2	14
Shift	4	24
Unpack	1	7
Return Jump	1	21

For a total of 598 minor cycles

APPENDIX F

Timings

TABLE 23

Timings (in Milliseconds) FTN

Order	HQR Time	%	HLU Time	%	HPLU Time	%	
12	3	100	3		3		a
12	123	100	85	69	88	72	
12	143	100	87	62	91	64	
12	142	100	99	70	103	73	
12 (5.8 x 5.4)	204	100	123	60	145	71	e
12 (5.5 x 5.4)	162	100	98	60	104	64	e
15 (5.20)	158	100	98	62	103	65	e
15 (5.21)	525	100	5 c		350	67	e
15 (5.22)	681	100	5 c		523	77	e
18 (5.12 x 5.1)	623	100	300	48	402	65	de
20 (5.23)	720	100	438 b	61	460	64	e
24	444	100	273	61	333	75	
59	6703	100	783 e		4329	65	
Random Matrices							
12	237	100	124	52	142	60	
25	1247	100	808	65	948	76	
25	1441	100	1095	76	1126	78	
50	7636	100	5614	74	6284	82	
50	7193	100	5219	73	4296	68	
100	53539	100	34497 c		37616	70	

(See Table 1 for references)

TABLE 24

Timings (in milliseconds) RUN

Order	HQR Time	%	HLU Time	%	HPLU Time	%	
12	3		2		3		a
12	213	100	145	68	156	73	
12	256	100	153	60	159	62	
12	257	100	173	67	184	72	
12 (5.8 x 5.4)	361	100	213	59	253	70	e
12 (5.5 x 5.4)	284	100	165	58	177	62	e
15 (5.20)	286	100	179	63	182	64	e
15 (5.21)	970	100	6 c		683	70	e
15 (5.22)	1294	100	5 c		1069	83	e
18 (5.12 x 5.1)	1171	100	535	46	774	66	de
20 (5.23)	1354	100	810 b	60	916	68	e
24	817	100	491	60	599	73	
59	14158	100	1792 c		10099	71	
Random Matrices							
12	447	100	434	97	412	92	
25	2771	100	1750	63	2261	82	
25	2425	100	1815	75	2256	93	
50	15748	100	10929	69	12422	79	
50	18442	100	7781 c		13529	73	
100	122221	100	83467	68	92382	76	

(See Table 1 for references)

TABLE 25

Timings (in seconds) Honeywell 437

Order	HQR Time	%	HLU Time	%	HPLU Time	%
25	.3	100	.2	67	.2	67
25	.3	100	.2	67	.2	67
50	2.0	100	c		1.4	70
50	1.9	100	1.1	58	1.4	74
75	6.3	100	3.6	57	4.7	75
75	6.1	100	3.8	62	4.6	75

(See Table 1 for references)

TABLE 26

Timings (in seconds) IBM 360/50 H Compiler

Order	HQR Time	%	HLU Time	%	HPLU Time	%
25	4.40	100	2.81	64	2.93	66
25	4.50	100	2.38	53	2.87	64
50	24.03	100	c		19.02	79
50	28.67	100	22.73	79	19.73	69
75	71.87	100	c		62.65	73

(See Table 1 for references)

BIBLIOGRAPHY

1. R.T. Gregory and D.L. Karney, A Collection of Matrices for Testing Computational Algorithms, Wiley-Interscience, New York (1969).

2. R.S. Martin, G. Peters and J.H. Wilkinson, The QR Algorithm for Real Hessenberg Matrices, Number. Math. 14, 219-231 (1970).

3. R.S. Martin, and J.H. Wilkinson, Similarity Reduction of a General Matrix to Hessenberg Form, Number. Math. 12, 349-368 (1968).

4. B.N. Parlett, The LU and QR Algorithms, Mathematical Methods for Digital Computers, Vol. 2, A. Ralston and H. Wilf eds., John Wiley, New York (1967).

5. B.N. Parlett, Certain Matrix Eigenvalue Techniques Discussed from a Geometric Point of View, Research Group Report, United Kingdom Atomic Energy Authority, Harwell, Berkshire (1973).

6. B.N. Parlett and C. Reinsch, Balancing a Matrix for Calculations of Eigenvalues and Eigenvectors, Number. Math. 13, 293-304 (1969).

7. B.N. Parlett and Y. Wang, The Influence of the Compiler on the Cost of Triangular Factorization, Computer Science Technical Report 14, University of California, Berkeley (1973).

8. J.H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press, London (1965).

9. Control Data 6400/6500/6600/6700 Computer Systems Reference Manual, Control Data Corporation, St. Paul (1970).

10. Honeywell Series 400 Programmers Reference Manual, Honeywell Information Systems, Inc., Wellesley Hills, Massachusetts (1970).

11. IBM System 360 Model 50 Functional Characteristics, International Business Machines Corporation, New York (1971).