NUMERICAL METHODS FOR TOEPLITZ MATRICES

by

Douglas Robin Sweet, B.Sc.(Hons), Dip. Comp. Sc.

A thesis submitted for the degree of

Doctor of Philosophy

in the Department of Computing Science, University of Adelaide

May 1982

TABLE OF CONTENTS

VOLUME I

VOLUME II

APPENDIX - PROGRAMS AND RESULTS

SUMMARY

This thesis is mainly concerned with methods for solving Toeplitz linear algebra problems in $O(n^2)$ multiplications/divisions. There are three main aims (i) to find new connexions between known algorithms, and re-derive some of these using different approaches (ii) to derive new results concerning the numerical stability of some algorithms, and to modify these algorithms to improve their numerical performance (iii) to derive fast Toeplitz algorithms for new applications, such as the orthogonal decomposition and singular value decomposition.

In Chapter 2, fast Toeplitz factorization algorithms (FTF's) are re-derived from the Bareiss algorithm for solving Toeplitz systems and also from algorithms for performing rank-1 updates of factors. In Chapter 3, the Bareiss algorithm is related to the Trench Algorithm for Toeplitz inversion.

Several new results regarding the propagation of rounding errors in the Bareiss algorithm are derived in Chapter 4. A pivoting scheme is proposed in Chapter 5 to improve the numerical performance of the Bareiss algorithm.

In Chapter 6, error analyses are performed on FTF's and some pivoting is incorporated to improve their numerical performance. The results of Chapter 3 are used to adapt the Bareiss pivoting procedure to the Trench algorithm.

Methods are proposed in Chapter 7 to compute the QR factorization of a Toeplitz matrix in $O(n^2)$ operations. This algorithm uses the shift-invariance property of the Toeplitz matrix and a known procedure for updating the QR factors of a general matrix. In Chapter 8, methods are described for speeding up the algorithms of the previous chapter, and several extensions are proposed.

In Chapter 9, two algorithms are proposed to compute the singular-value decomposition (SVD) of a Toeplitz matrix in fewer operations than for a general matrix. The first algorithm is $O(n^3)$ but depending on the dimension of the problem requires up to 80% fewer operations than for general SVD algorithms. The second possibly unstable method has complexity $O(n^2 \log n)$. A modification of this method is proposed which may enable the singular values to be calculated stably.

DECLARATION

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

D.R. SWEET

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Dr. Sylvan Elhay, for his encouraging support, his help in planning the thesis, and the many hours he has spent in painstakingly checking the rigour of the mathematics, and the many suggestions he has made to improve the presentation.

To Dr. Jaroslav Kautsky, I give thanks for several stimulating discussions and also for his assistance while my supervisor was on study leave.

Thanks are due to my employer, the Department of Defence, for giving me leave to pursue my studies. To my colleagues at the Defence Research Centre, Salisbury, and especially to Graham Mountford, I am obliged for their encouragement.

The financial support provided by the Australian Government under the Commonwealth Postgraduate Awards Scheme is gratefully acknowledged.

To Tracey Young, special thanks are due for her patience and skill over the last few months in typing the manuscript. I also recognize the assistance of Fiona Adlington in drawing the diagrams, and of Elizabeth Henderson and Dawn Darwent for their help in the typing.

For her patience, encouragement and urging over the past few years I pay tribute to my wife Vanessa.

NOTATION

General Matrix and Vector Notation

Symbol	Explanation
$\underline{a}$	A vector with elements a_1, a_2, ... a_m, where m is the order of $\underline{a}$
A	The matrix with elements a_{ij}, $i=1,\ldots,m;j=1,n$, where m and n are the number of rows and columns respectively in A
$\underline{a}_{i\cdot};\underline{a}_{\cdot j}$	Row i and column j of A
$\underline{a}_{i,j:k};\underline{a}_{h:i;j}$	Elements j to k of $\underline{a}_{i\cdot}$; elements h to i of $\underline{a}_{\cdot j}$
$A_{i:j,k:l}$	Rows i to j of columns k to l of A
$A_{i:j\cdot};A_{\cdot k:l}$	Rows i to j of A; columns k to l of A
A_k	The kth leading submatrix of A (unless otherwise indicated
$\underline{a}^T, A^T$	The transpose of $\underline{a}$, A
A^{-1}	The inverse of A
$\underline{a}^R$	The reverse of $\underline{a}$, i.e. if $\underline{a} = (a_1,\ldots,a_m)^T$, then $\underline{a}^R=(a_m,\ldots,a_1)^T$
$\underline{a}^{RT}$	The reverse-transpose of $\underline{a}$
A^{T2}	The secondary transpose of A (transpose about the secondary diagonal). $A^{T2}=EA^TE$ (see below for E)
$diag\{a_i\}_1^m$	A diagonal matrix with diagonal elements $a_1,a_2,\ldots a_m$
$det\ A$	The determinant of A
$\lVert A\rVert;\lVert\underline{a}\rVert$	A matrix-norm of A; a corresponding subordinate vector-norm of $\underline{a}$ (if applicable)
$\lVert A\rVert_p\ ;\ \lVert\underline{a}\rVert_p$	The p-norm of A and $\underline{a}$

Symbol	Explanation
$A_{k;j}$	The displaced leading submatrix of A with order k and displacement j: for $j\geq 0$, $A_{k;j}=A_{1:k,1+j:k+j}$; for $j<0$, $A_{k;j}=A_{1+\|j\|:k+\|j\|,1:k}$
A^E_k, A^S_k	Same as $A_{k;1}$; $A_{k;-1}$ (E: "East"; S: "South")
$A^E_{k;j}$; $A^S_{k;i}$	$(A_{1:k,1:k-1}, \underline{a}_{1:k,j})$; $A^S_{k;i} = \begin{pmatrix} A_{2:k+1,1:k} \\ \underline{a}_{i,1:k} \end{pmatrix}$
a^r_i , a^c_i	$\\|\underline{a}_{i\cdot}\\|_1$, $\\|\underline{a}_{\cdot i}\\|_1$
A_*	The Toeplitz part of A
a_{max}	$\max_{i,j}\|(A_*)_{ij}\|$
a_j	$j \geq 0$: the jth diagonal above the main diagonal of the Toeplitz part of A; $j < 0$: the $\|j\|$th diagonal below the main diagonal of the Toeplitz part of A.
$\underline{a} \circledast \underline{b}$	The cyclic convolution of $\underline{a}$ and $\underline{b}$
$\prod_{i=1}^{m} A_i$	$A_m A_{m-1} \ldots A_1$, where $\{A_i\}$ are square matrices of the same order (not leading submatrices of A)
$\bar{A}, \bar{a}$	The computed value of A, a
$\delta A, \delta a$	The errors in $\bar{A}, \bar{a}$; $\bar{A}-A$, $a-\bar{a}$ respectively
$rel\ \bar{A}$, $rel\ \bar{a}$	$\max_{i,j}\|\delta a_{ij}\|/\max_{i,j}\|a_{ij}\|$; $\|\delta a/a\|$
$cond\ A$	The condition number of A, defined as $\\|A\\|\\|A^{-1}\\|$
T	A Toeplitz matrix
$\underline{A}$	A block-vector with blocks $A_{(i)}$
A	A block-matrix with blocks $A_{(ij)}$
$\underline{A}_{(i\cdot)}$, $\underline{A}_{(\cdot j)}$	Block-row i and block-column j of A

Symbol	Explanation
$\underset{\sim}{A}_{(i,j:k)}$; $\underset{\sim}{A}_{(h:i,j)}$	Blocks j to k of $\underset{\sim}{A}_{(i\cdot)}$; blocks h to i of $\underset{\sim}{A}_{(\cdot j)}$
$a_{(ij)kl}$	Element k,l of block i,j of A
$A_{(k)}$	kth leading block-submatrix of A
$\underset{\sim}{A}^{BT},\underset{\sim}{A}^{BR},\underset{\sim}{A}^{BRT}$	Block-transpose, block-reverse, block-reverse-transpose of $\underset{\sim}{A}$

Specific Symbols (Latin)

Symbol	Explanation	First Occurrence*
$A^{\hat{i}},A^{\check{i}}$	A after the application of $G(\hat{\phi}_i),G(\check{\phi}_i)$	7.2.1
$\underline{c}^T$	$(T^{-1})_{1,2:n}/(T^{-1})_{11}$	3.2.1
C_p	Cyclic permutation matrix; $(C_p)_{ij}=1$ if $j=i+p(mod\ n)$, where n is the order of C_p; $(C_p)_{ij}=0$ elsewhere	5.2.2
$C^{(\pm i)}$	Cyclic permutation matrix for Bareiss step $(\pm i)$	6.7
D	Diagonal matrix in LDR factorization	2.1
$D,\dot{D},\bar{D},\tilde{D}$	Diagonal scaling matrices for $W,\dot{W},\bar{W},\tilde{W}$	8.4.2
$d_i,\dot{d}_i,\bar{d}_i$	Same as d_{ii}, $\dot{d}_{ii}$, $\bar{d}_{ii}$	8.4.2
$\dot{D}^{\hat{k}},\dot{D}^{\check{k}}$, etc.	$\dot{D}$, etc. after the application of $G(\hat{\phi}_k),G(\check{\phi}_k)$, etc.	8.4.2
$\underline{d}$	$(T^{-1})_{2:n,1}/(T^{-1})_{11}$	3.2.1
$d_{\pm i}$	Denominator of multiplier $m_{\pm i}$	6.7
E	Exchange matrix; $e_{ij}=1$ if $i=n-j+1$; $e_{ij}=0$ elsewhere	2.3
E	Matrix partition	4.2

*chapter/section/sub-section (if applicable)

Symbol	Explanation	First Occurrence
E	Backward error matrix for Toeplitz factorization	6.3.1
$E^{(\pm i)}$	Local error in computing Bareiss iterate $(\pm i)$	4.4
$\underline{e}_i$	ith column of identity matrix	4.2
F	Backward error matrix for Bareiss algorithm	4.4
f_j, $\dot{f}_j$, etc.	Scaling factors for $\underline{g}_{j\cdot}$, $\dot{\underline{g}}_{j\cdot}$, etc.	8.4.3
fl	Floating-point operation	4.4
F	Initialization procedure for rank-1 update	2.4.1
G	Backward error matrix for rank-1 update	6.3.1
$\underline{g}_{j\cdot}$, $\dot{\underline{g}}_{j\cdot}$, etc.	Scaled versions of $\underline{y}_{j\cdot}$, $\dot{\underline{y}}_{j\cdot}$, etc. (e.g. $\underline{y}_{j\cdot}=f_j\underline{g}_{j\cdot}$)	8.4.3
$\underline{g}$	Backward error vector for Bareiss algorithm	4.4
$H(\theta_i)$	Inverse transform with relation θ_i	8.2
J	Backward error matrix for Bareiss algorithm	4.4
K_1, K_2	(Scalars) Bareiss pivoting parameters	5.3.2
L	Unit-lower-triangular matrix in LU factorization	2.1
L	(Scalar) Ratio $t_{max}^{(-k)}/\varepsilon$ (see below)	4.3
$M^{(\pm i)}$	Multiplier matrices for $T^{(\pm i)}$, defined by $M^{(\pm i)}T=T^{(\pm i)}$	3.3.2
$M^{(\pm)}$	Same as $M^{(\pm(n-1))}$	2.3
$m_{\pm i}$	Multipliers for Bareiss recursion $(m_{\pm i}=n_{\pm i}/d_{\pm i})$	2.2
$\underline{\tilde{m}}^{(\pm s)T}$	First row of Toeplitz part of $M^{(\pm s)}$	5.2.2

Symbol	Explanation	First Occurrence
n	Order of Toeplitz matrix T	1.3
$n_{\pm i}$	Numerator for multiplier	6.7
n_A, n_c	Number of A, C-cycles	6.4
$O(n)$, etc.	Order of n, etc.	1.3
$\mathcal{P}$	Initialization procedure for Toeplitz factorization	2.4.2
P_1,P_2	Permutation matrices	6.4
$Q,Q^{(n-1)}$	Orthogonal factor in QR decomposition of T, T_{n-1}	7.1, 7.3.1
$\dot{Q}$, $\tilde{Q}$	Auxiliary orthogonal matrices	7.3.2
$\mathcal{Q}$	Orthogonal block-matrix	8.5.3
$R,R^{(n-1)}$	Triangular factor in QR decomposition of T, $T^{(n-1)}$	7.1, 7.3.1
$\dot{R},\tilde{R},\bar{R}$	Certain upper-triangular + rank-1 matrices	7.3.2
$\tilde{R}^{(i)}$	$\tilde{R}$ after the application of the ith plane rotation	8.2.2
R	Upper-triangular factor in LDR factorization	2.1
R,R_1,R_2,etc.	Certain scalars in error analyses	4.3
$\mathcal{R}$	Recursion relation for rank-1 update	2.4.1
$\mathrm{R},\mathrm{R}^{(n-1)}$	Upper-triangular block-matrices	8.5.3
$\hat{r}$	$\max_i \lvert\tilde{d}_i/d_i\rvert$	6.3.3
$S^T,S^T(\hat{\phi}_i)$, $S^T(\hat{\phi}_i,\check{\phi}_i)$	A complete or partial set of Gill-Golub-Murray-Saunders rotations	7.2.2,7.2.3
$\underline{s}$	A permutation vector	6.4
S_{pq}	A selection matrix $(S_{pq})_{ii}=1$, $p\le i\le q$; $(S_{pq})_{ij}=0$ elsewhere	5.2.2

Symbol	Explanation	First Occurrence
$S^{(\pm i)}$	Selection matrix used in Bareiss step $(\pm i)$	6.7
$S,\tilde{S}$	A set of block Gill-Golub-Murray-Saunders rotations	8.5.3
S	A certain set of submatrices	4.2
T	A Toeplitz matrix	1.1
$T^{(\pm i)}$	Bareiss iterate $(\pm i)$	2.2
$T^{(\pm)}$	Bareiss iterate $(\pm(n-1))$	2.3
$t_j^{(\pm i)}$	Diagonal j (+ above main; - below main) of Toeplitz part of $T^{(\pm i)}$	2.2
$t_{max}^{(\pm k)}$	$\max_{pq}\lvert(T_*^{(\pm k)})_{pq}\rvert$	5.3.1
t, t_0	Same as t_{11}	2.4.1,3.2.1
$t_A^{(\pm i)}, t_B^{(\pm i)}$	Element in first diagonal above, below zero-band of $T_*^{(\pm i)}$	5.4
U	Upper-triangular factor in LU-factorization	2.1
U	Orthogonal matrix in singular-value decomposition	9.1
UT	Upper-triangular matrix	2.1
ULT	Unit-lower-triangular matrix	2.1
UT+R1	Upper-triangular + rank-1 matrix	7.2.1
UH	Upper-Hessenberg matrix	7.2.1
$\underline{u}^T$	$\underline{t}_{1,2:n}$ or $\underline{t}_{1,2:n}/t_{11}$	2.4.1,3.2
V	Orthogonal matrix in singular-value decomposition	9.1
$\underline{v}$	Permutation vector	6.4
$\underline{v}$	$\underline{t}_{2:n,1}$ or $\underline{t}_{2:n,1}/t_{11}$	2.4.1,3.2

Symbol	Explanation	First Occurrence
$W,\dot{W},\widetilde{W},\bar{W}$	Scaled versions of $Y,\dot{Y},\widetilde{Y},\bar{Y}$ ($W=DY$, etc.)	8.4.2
$W^{\hat{j}},W^{\check{j}}$, etc.	$\dot{W}$ after the application of $G(\hat{\phi}_j),G(\check{\phi}_j)$	8.4.2
$Y,\dot{Y},\widetilde{Y}$	$Q^T,\dot{Q}^T,\widetilde{Q}^T$	7.4.1
$\dot{Y}^{\hat{j}},\dot{Y}^{\check{j}}$, etc.	$\dot{Y}$ after the application of $G(\hat{\phi}_j),G(\check{\phi}_j)$ etc.	7.4.3
$\underline{y}_{j\cdot}^{(i)},\dot{\underline{y}}_{j\cdot}^{(i)}$	$\underline{y}_{j\cdot},\dot{\underline{y}}_{j\cdot}$ after certain rotations	7.4.3
$Z_{\pm i}$	Shift matrix; $(Z_{\pm i})$ if $j=k-i$, otherwise 0	2.2

Specific Symbols (Greek)

α	$\lvert t_{-k-1}^{(-k)}/t_{max}^{(-k)}\rvert$	6.7
β,β',β_p	Ratios between norms of certain submatrices of T	Chap.4-6
γ	$\lvert t_{-k-1}^{(-k)}/t_{max}^{(-k)}\rvert$	5.4
Γ	Compressed (2×2) rotation matrix	8.4.3
δ	Pivoting threshold	5.3.1
$\delta a,\delta A$	Error in a, A	4.3
ε	$\lvert t_0^{(-k)}\rvert$	4.3
ε	A small quantity	4.2
λ_i	$1/(T_{i+1}^{-1})_{11}$	3.2.1
λ	$t_{max}^{(-k)}/\varepsilon$	6.2
$\hat{\phi}_i,\check{\phi},\hat{\theta}_i,\check{\theta}_i$	Rotation angles for Gill-Golub-Murray-Saunders upsweeps $(\hat{\ })$ and downsweeps $(\check{\ })$	7.3.3
$\Phi_k(\hat{\phi}_i),\Phi(\hat{\phi}_i)$etc.	Compressed (2×2) fast Givens transform matrices	8.4.2

Symbol	Explanation	First Occurrence
Σ	Diagonal matrix of singular values	9.1
σ_i	Singular values	9.1
$\dot{\sigma}_i, \tilde{\sigma}_i$	$\\|\dot{\underline{r}}_{i:n,1}\\|, \\|\tilde{\underline{r}}_{i:n,1}\\|$	7.3.4
μ	Machine precision	4.3

Special Symbols	Explanation	First Occurrence
$:=$	Definition, e.g. $A := B$ denotes "A is defined by B"	2.2
$=:$	Definition, with the quantity being defined on the right, e.g. $C =: D$ denotes "D is defined by C".	2.3

GLOSSARY OF ALGORITHMS

Name	Algorithm	Algorithm No. (Chap.,Sec.,No.)
ABA	Adapted Bareiss algorithm	Alg. 2.3.1
ABSA	Alternative Bareiss Symmetric Algorithm	Alg. 3.4.1
AGSB	Accelerated Gram-Schmidt Bidiagonalization	Alg. 9.4.2
A-cycles	Bareiss pivoting cycles	Proc. 5.2.2
BNA (B-cycles)	Bareiss nonsymmetric algorithm	Alg. 2.2.1
BPA,BPB,BPC	Backtrack Procedures A,B,C	Proc. 5.2.1, 5.2.2
BPP	Basic Pivoting Procedure	Sec. 5.2.6
BR1A	Bennett Rank-1 update algorithm	Alg. 2.4.2
BSA	Bareiss symmetric algorithm	Alg. 3.3.1
C-cycles	Bareiss pivoting cycles	Proc. 5.2.3
EBSA	Extended Bareiss symmetric algorithm	Alg. 3.3.2
FGT	Fast Givens Transform	Sec. 8.4.1
FTO1	Fast Toeplitz orthogonalization, version 1	Alg. 7.3.1
FTO2	Fast Toeplitz orthogonalization, version 2	Alg. 7.4.1
FTO3	Fast Toeplitz orthogonalization, version 3	Alg. 8.2.3
FTO4	Fast Toeplitz orthogonalization, version 4	Alg. 8.3.2
FTO5	Fast Toeplitz orthogonalization, version 5	Alg. 8.4.2
GGMS	Gill-Golub-Murray-Saunders algorithm	Alg. 7.2.1
GR	Golub-Reinsch Algorithm	Alg. 9.2.1
GRC	Golub-Reinsch-Chan algorithm	Alg. 9.2.2
MBA	Modified Bennett algorithm	Alg. 2.4.4
MGRC	Modified Golub-Reinsch-Chan Algorithm	Alg. 9.3.1
PABSA	Pivoted adapted Bareiss symmetric Algorithm	Proc. 6.8.1

Name	Algorithm	Algorithm No. (Chap.,Sec.,No.)
PABSA-1	Pivoted alternate Bareiss symmetric Algorithm, strategy 1	Alg. 6.8.1
PBA-1,2,3,4	Pivoted Bareiss Algorithm, Strategy 1,2,3,4	Alg. 5.3.1, 5.3.2 etc.
PBSA-1	Pivoted Bareiss Symmetric Algorithm, strategy 1	Alg. 5.5.1
PTZA	Pivoted Trench-Zohar algorithm	Sec. 6.7.8
RBNA (R-cycles)	Reverse Bareiss nonsymmetric algorithm	Proc. 5.2.5
TLDR	Toeplitz LDR factorizer	Alg. 2.4.3
TZA	Trench-Zohar algorithm	Alg. 3.2.1

CHAPTER 1

INTRODUCTION

1. TOEPLITZ MATRICES

A *Toeplitz matrix* is one in which the elements along any northwest-to-southeast diagonals are all equal. As an illustration a 5×5 Toeplitz matrix has the form

$$T = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 & t_4 \\ t_{-1} & t_0 & t_1 & t_2 & t_3 \\ t_{-2} & t_{-1} & t_0 & t_1 & t_2 \\ t_{-3} & t_{-2} & t_{-1} & t_0 & t_1 \\ t_{-4} & t_{-3} & t_{-2} & t_{-1} & t_0 \end{bmatrix} \tag{1.1}$$

Each element of a Toeplitz matrix depends only on the difference of its indices, so we may write t_{ij} as t_{j-i} .

Toeplitz matrices arise in a large number of seemingly unrelated applications of applied mathematics and engineering. Some of these applications are given in the next section.

There are other types of matrices which are related to Toeplitz matrices, and to which Toeplitz theory can be adapted, with suitable modifications and generalizations. Examples are *Hankel* matrices, in which the elements along any *southwest-to-northeast* diagonal are all equal; *circulant* matrices, a subset of the Toeplitz class in which each row is a *right-rotation* of the previous rows; *block*-Toeplitz matrices, where the *blocks* along any northwest-to-southeast *block* diagonal are all equal; block-Hankel and block-circulant matrices; matrices with a 2-level special structure, e.g. a Toeplitz block-structure with circulant blocks. A different generalization of the Toeplitz class is the algebra of α-*Toeplitz* matrices, introduced by Friedlander *et al*, [29]. This algebra is the set of matrices of the form

$$T_\alpha = \sum_{i=1}^{\alpha} L_i U_i$$

where the $\{L_i\}$ and $\{U_i\}$ are upper and lower triangular Toeplitz matrices respectively. Clearly, when $\alpha=2$ and $U_1=L_2=I$, T_α is a Toeplitz matrix. All of the abovementioned types of matrices have important applications, some of which are given below. In this thesis, we concentrate on Toeplitz matrices, and mention only briefly how the results may be extended to related forms, when this is possible. We do not consider special forms such as band or triangular Toeplitz matrices.

2. APPLICATIONS OF TOEPLITZ AND RELATED MATRICES

Toeplitz matrices arise in many applications of time series analysis, image processing, probability and statistics, control theory, and in mathematical techniques such as polynomial, and rational and exponential approximation, solution of elliptic and parabolic partial differential equations and certain Fredholm integral equations. An early reference to the application of Toeplitz forms is that of Grenander and Szegö [42]. Cornyn's thesis [21] gives a very comprehensive list of Toeplitz applications and an extensive bibliography (255 references) up to 1974. For completeness we will repeat most of the applications listed by Cornyn, but only give some representative references, referring the reader to Cornyn for a complete list. A later survey of Toeplitz matrices is given by Roebuck and Barnett [81], who include the important Toeplitz-related work of Kailath, Morf and co-workers, which has appeared since 1974.

We now examine some Toeplitz applications in detail. In the field of time-series analysis, Toeplitz matrices occur in linear filtering or prediction problems [59], [67], [16], [60], [63], [1], [52] when the signal and noise statistics are stationary. At the end of this section, we will illustrate one such application by showing how Toeplitz systems arise in the discrete Wiener filter [59]. Levinson's algorithm for solving this system is the basis for the recently-developed lattice or ladder filter structure, pioneered by Morf, Lee and others [69], [58], [61]. Block-Toeplitz matrices arise in multichannel filtering [88], [4], [65]. In other areas of time-series analysis, Toeplitz matrices arise in maximum-entropy spectral analysis [85], antennas and arrays [82], adaptive beamforming [77], estimates of shaping and matching filters [88] and fast recursive estimation [53].

Matrices with a 2-level special structure (Toeplitz and/or circulant) occur in image processing [2], and in general, 2-dimensional processing [51].

In probability and statistics Toeplitz matrices occur in statistical communication and detection [76], stationary auto-regressive time-series, and the Ising model of statistical mechanics [45]. In control theory, Toeplitz or Hankel matrices occur in the minimal realization problem [24], [65], the minimal design problem [56], [57], and the adaptive estimation of control system parameters by Bayesian methods [5].

In mathematical techniques, Toeplitz systems arise in Padé approximation [39], [38], [10], [14], [15], [13], spectral factorization [79], Prony's method for exponential fitting [46], convolution-type Fredholm equations [73], elliptic and parabolic partial differential equations [62], orthogonal polynomials [49], [55], and canonical matrix fractions [25].

The foregoing illustrates the wide variety of applications where Toeplitz matrices occur. We illustrate one application by deriving the equations for the Discrete Wiener Filter.

The Weiner Filter Equations [59]

We wish to determine the nature of a linear tapped-delay-line filter, which, with input b_k, will have output as close as possible to a desired signal a_k. That is we wish to determine the filter weights $\{w_n\}_0^M$ such that the error

$$\varepsilon_k = a_k - \sum_{n=0}^{M} w_n b_{k-n} \tag{2.1}$$

is as small as possible in the mean-squared sense. To do this, we minimize $\langle\varepsilon_k^2\rangle$ with respect to the w_n. Here $\langle\cdot\rangle$ denotes ensemble average; for a stationary time-series x_k, $\langle x_k\rangle := \lim_{N\to\infty} \frac{1}{2N+1} \sum_{i=-N}^{N} x_{k+i}$. From (2.1), and using the fact that the operator $\langle\cdot\rangle$ is linear,

$$\langle\varepsilon_k^2\rangle = \langle a_k^2\rangle - 2\sum_{n=0}^{M} w_n \langle a_k b_{k-n}\rangle + \langle(\sum_{n=0}^{M} w_n b_{k-n})^2\rangle$$

Differentiating with respect to w_j and setting to zero,

$$\frac{\partial\langle\varepsilon_k^2\rangle}{\partial w_j} = -2\langle a_k b_{k-n}\rangle + 2\sum_{n=0}^{M} w_n\langle b_{k-j} b_{k-n}\rangle = 0 \qquad j=0,\ldots,M. \tag{2.2}$$

Since the processes are stationary the averages in the middle expression of (2.2) are independent of k and depend only on the difference between the indices. In fact $\langle a_k b_{k-n}\rangle = r_{ab}(n)$, the cross-correlation between a_k and b_k, and $\langle b_{k-j} b_{k-n}\rangle = r_{bb}(n-j)$, the autocorrelation function of b_k. This function is even so we also have $\langle b_{k-j} b_{k-n}\rangle = r_{bb}(j-n)$. Hence (1.2) can be written

$$\sum_{n=0}^{M} w_n r_{bb}(j-n) = r_{ab}(n), \quad j=0,\ldots,M. \tag{2.3}$$

Eq. (2.3) is a Toeplitz set of equations; they are the well-known Weiner filter equations.

3. AIMS OF THE THESIS

Note on Operation Counts

Throughout this thesis we will only count the number of multiplications and divisions as a measure of execution time, since these operations normally take considerably more time to execute than additions and subtractions. Moreover, the number of additions and subtractions is, in all the algorithms discussed, either less than or about the same as the number of multiplications. We therefore refer to an *operation* as a multiplication or division.

Most of the applications of section 2 require the solution of a Toeplitz set of linear equations

$$T\underline{x} = \underline{b} \tag{3.1}$$

or the related processes of inverting or factorizing T. Some of the applications require the solution of the full-rank Toeplitz least squares problem, which is to find $\underline{x}$ which minimizes $\|T\underline{x}-\underline{b}\|_2$, where T is $m\times n$ $(m \geq n)$. A few applications require the eigenvalues, the singular values, or the complete singular-value-decomposition (SVD) of T.

In the published literature, much work has been done on fast $(O(n^2)$ operations or less) methods of solving (3.1), and the related problems of factorization and inversion; however, little work has been done hitherto on ways of reducing rounding errors without sacrificing too much execution speed.

On a different topic, few methods exist that exploit the Toeplitz property in techniques of numerical linear algebra such as the QR decomposition, the calculation of the eigenvalues or SVD. We note that many *theoretical* results for the Toeplitz eigen-problem have been obtained [41], but the *computational* side of these problems has been neglected. This thesis therefore aims

(1) To improve the accuracy of solving (3.1), retaining a complexity of $O(n^2)$.

(1a) To derive new connexions between the solution of (3.1) and the problems of factorization and inversion, and to use these results to improve the accuracy of solving the latter two problems.

(2) To develop fast methods $(< O(n^3))$ for linear problems other than that of solving a set of linear equations.

In the next section, we review the main advances in Toeplitz numerical methods, and in the following section, we state the principal results of our investigation of (1) and (2).

4. PREVIOUS WORK ON TOEPLITZ METHODS

In 1947, Levinson [59] was the first to solve a Toeplitz set of equations in $O(n^2)$ operations. His algorithm solves a symmetric positive-definite system in $3n^2$ operations and is the basis for many subsequent Toeplitz solvers and inverters. 12 years later Durbin [26] solved a more specialized system, the Yule-Walker equations, in n^2 operations. In 1969, Bareiss [6], solved an indefinite, nonsingular system in $3\frac{1}{4}n^2$ operations. His approach of eliminating down the diagonals, is quite instructive, and we will extend this approach in developing methods with improved accuracy. From 1968 to 1971, Berlekamp and Massey [8], [64] and Rissanen [78] also developed elimination-type approaches to reduce Toeplitz and Hankel systems respectively to triangular form; these methods also can handle cases where a leading minor is zero, in contrast to many existing Toeplitz methods. Two years later, Wiggins and Robinson [88] improved the operation-count in Levinson's algorithm (for a symmetric positive-definite system) to $2n^2$ operations, and Markel and Gray [63] rederived the Wiggins-Robinson algorithm using orthogonal polynomials. In 1974, Zohar [93] improved the operation-count for a general system to $3n^2$ operations. In 1979, as a result of a co-operative effort between the

US and the USSR, a Toeplitz software package [87] after the style of LINPACK was produced. Also in 1979, Jain [48] reduced the complexity of solving (3.1) from $3n^2$ to $2n^2+8n \log n$, and incorporated iterative improvement, requiring $8n \log n$ operations per iteration. 1980 saw a major advance in that Brent, Gustavson and Yun [13] and Bitmead and Anderson [9] developed Toeplitz solvers with asymptotic complexity $O(n \log^2 n)$. However, Brent [12] has pointed out that n needs to be fairly large for the new algorithm to be faster than $O(n^2)$ methods, so there is still a place for the latter. Bitmead and Anderson also extended their method to α-Toeplitz matrices, as did Morf [66] independently.

Closely related to the problem of solving linear equations is the problem of inverting the system matrix. In 1964, Trench [84] wrote an algorithm which inverted a positive definite symmetric Toeplitz matrix in $3n^2$ operations, and briefly indicated how to extend it to the non-symmetric case. Zohar [92] re-derived Trench's algorithm in a lucid manner; his algorithm applied to all 'strongly nonsingular' Toeplitz matrices, i.e. matrices with no zero leading minors. Justice [49], [50] derived a Toeplitz inversion algorithm from the recursion relations for Szegö polynomials. In 1977, Justice [51] used bivariate Szegö polynomials to derive an efficient inversion algorithm for 2-level Toeplitz matrices.

Also related to the problem of solving linear equations is the problem of factoring T into upper and lower triangular matrices, though these factorizations have other applications, e.g. in Padé approximation [14] and spectral factorization [79]. Rissanen [79] was the first to find an $O(n^2)$ Toeplitz factorizer; Morf [65] independently developed two factorizers, a row-wise and a column-wise factorizer.

Both these factorizers fail when T has a zero leading minor. In 1974, Rissanen [80] developed a pivoting technique for a Hankel factorizer which could handle zero leading minors. In 1977, Gibson [31] interpreted certain intermediate quantities in Toeplitz factorizations as reflection coefficients. Kailath et al [54] derived a comprehensive set of results for the factorization of discrete and continuous α-Toeplitz operators. In 1979 and 1980, Bultheel [14], [15], derived several significant connections between Toeplitz factorization and Padé approximation.

There is a strong connexion between Padé approximations and the solution of Toeplitz systems - i.e. the standard so-called coefficient Padé problem require the solution of a Toeplitz system. Other authors besides Bultheel who have used this connexion include Gragg [38], Graves-Morris [39] and Bose and Basu [10]. Some recent theoretical work on Toeplitz matrices has been done by Delsarte, Genin and Kamp [95]-[97].

Concerning block Toeplitz matrices, Aikake [4] developed the first inversion algorithm for block-Toeplitz matrices in 1973. Agrawal [1] and Agrawal and Jain [2] proposed algorithms to solve Block-Toeplitz and block-circulant systems in 1976 and 1977 for the digital restoration of images and for designing constrained least-squares filters. We have already mentioned Justice's work [51] on 2-level Toeplitz matrices. In 1980, Bose and Basu [11] gave an efficient recursion for the solution of 2-level Hankel matrices arising from 2-D matrix Padé approximants.

There have been a few recent studies on the error analysis of the solution of Toeplitz systems. Cybenko [22] has given a rounding error analysis for Durbin's algorithm, and has extended it [23] to Levinson's and Trench's algorithms. Graves-Morris [39] gave a comprehensive survey of the numerical calculation of Padé Approximants, including Toeplitz methods, stating that the latter are unreliable because of the inability to pivot. De Jong [24] shows that existing $O(n^2)$ recursive methods for solving the so-called realization problem of systems theory are unstable. This problem involves the solution of a *Hankel* system of equations. He proposes a stable method using orthogonal transforms which, however, requires $O(n^3)$ operations.

Turning now to Toeplitz problems other than the solution of a linear set of equations, we find that since 1974, Morf, Kailath and co-workers [68], [53], [58], [28] have developed fast algorithms for least-squares estimation and prediction with stationary or close-to-stationary statistics. These problems essentially require the solution of the Toeplitz (and α-Toeplitz) least-squares problem; and the new methods solve it in $O(pt)$ operations, where p is the order of the problem and t is the number of observations, compared to $O(p^2t)$ operations for the general least-squares problems. In this family of algorithms there are fast Kalman and fast square root estimators; some of the fast algorithms are implemented by the new lattice or ladder filters.

Pye et al [75] discuss a fast method to find the pseudo-inverse of a 2-level circulant matrix, and Hartwig [44] discusses the Drazin inverse of a Toeplitz matrix. Kung and Kailath [56] propose a fast algorithm to find the LQ decomposition of a special Toeplitz matrix, called a generalized resultant matrix, for the minimal design problem. Morf [65] calculates indirectly the R-factor of the Toeplitz QR decomposition by finding the Cholesky decomposition of $T^T T$. Cline, Plemmons and Worm [94] discuss generalized inverses of Toeplitz matrices.

Recently, it has been required to compute the eigenvalues, singular values, and SVD of Toeplitz or Hankel matrices in certain problems of control theory [57] and rational approximation [83]. However, no 'fast' algorithms (complexity less than $O(n^3)$) have yet been developed for these problems.

5. MAIN RESULTS

Most existing Toeplitz algorithms to solve (3.1) break down when a leading submatrix of T is singular. It would be expected that if a leading submatrix of T is ill-conditioned, serious rounding errors may occur in these algorithms. This has been shown by Cybenko [22] for Levinson's [59], Durbin [26] and Trench's [84] algorithms for solving $T\underline{x} = \underline{b}$, and by de Jong [24] for Rissanen's algorithm [78] for Hankel reduction. We show that the possible rounding error at step k is proportional to the condition number of the T_k, the k^{th} leading submatrix of T, for Bareiss's algorithm [6] for solving (3.1) and for Toeplitz factorization. Backward error analyses are also given for Bareiss's algorithm and for Toeplitz factorization algorithms. We then propose a pivoting scheme for Bareiss's algorithm to help overcome the problem of error growth when some leading submatrices of T are ill-conditioned. This pivoting scheme is then extended to Toeplitz factorizers, and to the Trench-Zohar inversion algorithm by using the connexions between the latter two algorithms and Bareiss's algorithm. This pivoting scheme may be considered to be an extension of existing pivoting schemes to handle *singular* leading submatrices such as those of Bareiss, Berlekamp and Massey, and Rissanen [80], and of methods to handle "non-normal" Padé problems [18], which have Toeplitz systems with singular leading submatrices. We note that our pivoting scheme also handles singular leading submatrices, but is not equivalent to Bareiss's method, since the latter technique requires the triangularization of a small non-Toeplitz block, in contrast to the method proposed here.

The pivoting scheme is the main result in respect of our first aim, i.e. to improve the accuracy in solving $T\underline{x} = \underline{b}$. For the second aim, to develop fast methods for other Toeplitz problems, we propose algorithms to find in $O(n^2)$ operations, the factors Q and R of T, where Q is orthogonal and R is upper-triangular. These algorithms are also relevant to the first aim, since solving $T\underline{x} = \underline{b}$ in this way enables singular and ill-conditioned leading submatrices to be handled without significant loss of accuracy. To the author's knowledge, there are no algorithms which calculate Q and R explicitly in $O(n^2)$ operations for a general Toeplitz matrix, though Küng and Kailath's algorithm [56] calculate Q for a special Toeplitz system called a generalized resultant matrix. The present algorithms assume no knowledge of control theory or polynomial matrix theory, as is required by [56], but are formulated in terms of elementary matrix operations.

Also for our second aim, we examine methods that exploit the Toeplitz property in the calculation of the SVD and the eigenvalues. We were unable to find a stable method of finding the Toeplitz SVD in $< O(n^3)$ operations, but we propose an $O(n^3)$ method which is 2 to 5 times as fast as general methods in several important cases. We also propose an $O(n^2 \log n)$ method for the SVD which may be unstable because it is based on the Lanczos algorithm; however, Parlett and Reid [74] have given a method for computing eigenvalues by running the Lanczos algorithm on an iterative basis and using a special method for tracking the eigenvalues. With their method convergence has been attained even for ill-conditioned problems in about $5n$ iterations. We suggest adapting this device to the $O(n^2 log\ n)$ SVD routine, in the hope that the singular values only (or the eigenvalues if T is symmetric) can be calculated stably in $O(n^2 log\ n)$ operations.

6. ORGANIZATION OF THE THESIS

In Chapter 2, some alternative derivations of the Fast Toeplitz Factorizer (FTF) are given; the first derivative relates FTF to the Bareiss algorithm; the second relates FTF to rank-1 undate algorithms for factors. These results will be used in performing the error analysis on, and incorporating pivoting into FTF in Chapter 6. In Chapter 3, the Bareiss algorithm is related to the Trench-Zohar inversion algorithm. This relation will be used to incorporate pivoting into the Trench-Zohar algorithm in Chapter 6. In Chapter 4, an error analysis is performed on the Bareiss algorithm, and in Chapter 5 the pivoting technique is described; the improved numerical performance of the pivoted algorithm is shown. In Chapter 6, error analyses are performed on Toeplitz factorization algorithms and pivoting incorporated into these algorithms and Toeplitz inversion. In Chapter 7 algorithms are described which computes the QR decomposition of a Toeplitz matrix in $O(n^2)$ operations. In Chapter 8, Toeplitz QR algorithms are presented which are faster than those in Chapter 6, though logically more complex. Some extensions and applications of the algorithms are also given. Finally in Chapter 9, two algorithms are given which use the Toeplitz property to accelerate the calculation of the SVD.

Note on the Numbering Scheme. In any chapter, all theorems, lemmas, algorithms, procedures, figures and equations are numbered independently in the form A.B, where A is the section number and B is the order of occurrence of the theorem of lemma or algorithm or procedure or figure or equation within the section. Within the chapter of occurrence, the theorem, etc. is referenced by A.B; otherwise, the theorem, etc. is referenced by C.A.B., where C is the chapter of occurrence.

CHAPTER 2

FAST TOEPLITZ FACTORIZATION ALGORITHMS - ALTERNATIVE DERIVATIONS

1. INTRODUCTION

In this chapter, we consider the decomposition of the Toeplitz matrix T into triangular factors. We mainly consider the form

$$T = LU \tag{1.1}$$

where L is unit-lower-triangular (ULT) and U is upper-triangular (UT). The results carry easily over to other forms; in particular, we will briefly consider the decomposition

$$T = LDR \tag{1.2}$$

where L is ULT, D is diagonal, and R is unit upper-triangular (UUT) at the end of the chapter.

The decomposition (1.1) can be used to solve the system

$$T\underline{x} = \underline{b} \tag{1.3}$$

This can be done by solving

$$L\underline{y} = \underline{b} \tag{1.4}$$

$$\text{and} \qquad U\underline{x} = \underline{y} \tag{1.5}$$

by forward and back-substitution respectively. Other applications of Toeplitz factorization are spectral factorization [79], Padé approximation [14] and minimal design problem of system theory [65].

Rissanen in 1973 [78], and Morf in 1974 [65] independently developed Toeplitz factorization algorithms requiring $2n^2 + O(n)$ operations.

In this chapter, we derive, in two ways, an *LU*-factorization algorithm which is related to Morf's column recursion for the *LDR* factors. For the first derivation, we show that Bareiss's Toeplitz elimination algorithm [6], written in 1969, essentially computes the triangular factors of *T*; we propose a simple modification of Bareiss's algorithm that calculates *L* and *U*. For the second derivation, we show that Toeplitz factorizers may be obtained very simply from algorithms which update the triangular factors of a general matrix by a rank-1 matrix; a Toeplitz *LDR* factorizer may thus be obtained from Bennett's [7] algorithm for updating *LDR* factors, and the Toeplitz *LU* factorizer, derived above, may be obtained from a suitably-modified version of Bennett's algorithm.

The purpose of this chapter is not to present new fast Toeplitz factorizers, (FTF's) but to point out the connexions between FTF's and firstly elimination algorithms (e.g. Bareiss's algorithm), and secondly, rank-1 factor update algorithms. These connexions may be used to derive the FTF's simply, and will also be used in Chapter 6 in our error analysis of Toeplitz factorizers.

In Section 2, we review the Bareiss algorithm in detail; it will be needed in this Chapter to derive FTF algorithms, and also in Chapters 3-6. In section 3, we show how the Bareiss algorithm computes the triangular factors of *T*, and propose a modified algorithm, the adapted Bareiss algorithm (ABA), to calculate *L* and *U*. In Section 4, we show how FTF's may be derived from rank-1 factor-updaters, and derive a Toeplitz *LDR* algorithm from Bennett's algorithm. We then modify Bennett's algorithm to update the *LU* factors, and hence derive a Toeplitz *LU* factorizer, showing it to be the same as the ABA.

2. THE BAREISS ALGORITHM

The Bareiss algorithm solves

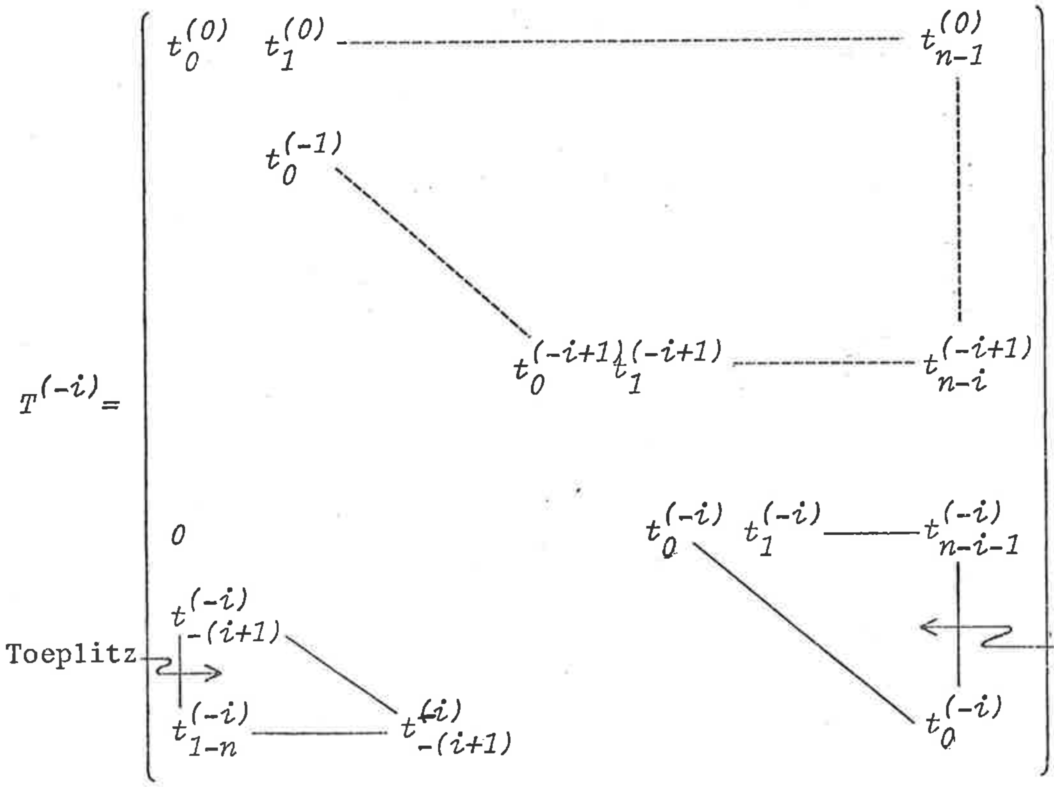
$$T\underline{x} = \underline{b} \tag{1.3}$$

by transforming (1.3) successively into

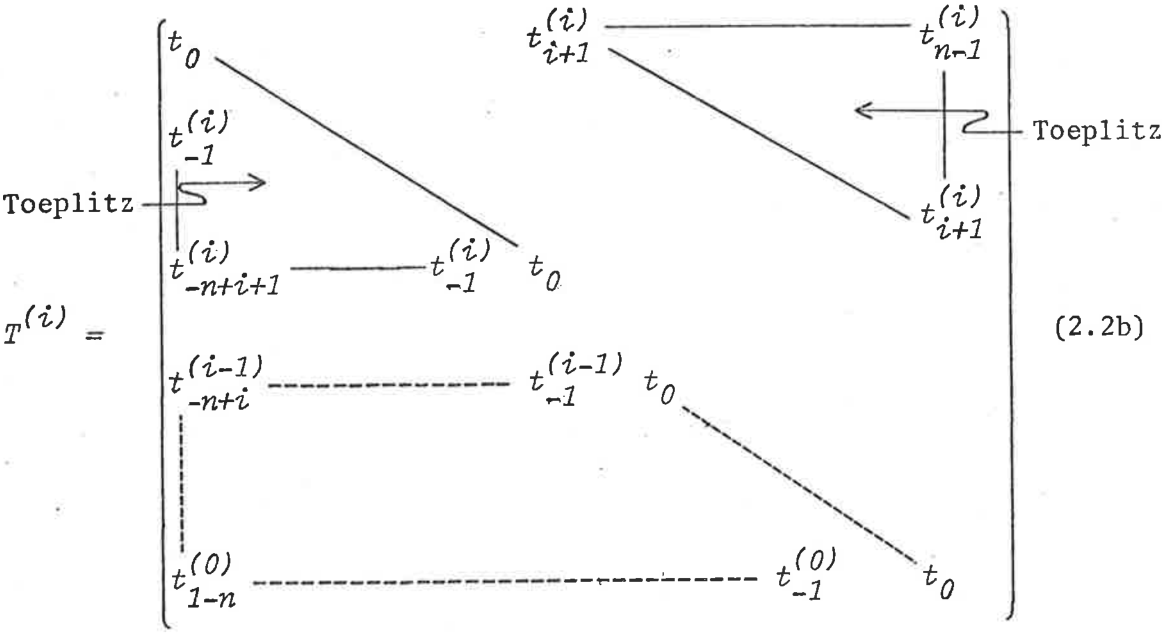
$$T^{(-1)}\underline{x} = \underline{b}^{(-1)} \ , \ \ T^{(1)}\underline{x} = \underline{b}^{(1)} \ , \ T^{(-2)}\underline{x} = \underline{b}^{(-2)} \ ,$$
$$T^{(2)}\underline{x} = \underline{b}^{(2)} \ , \ . \ . \ . \ , \ T^{(1-n)}\underline{x} = \underline{b}^{(1-n)}, \ T^{(n-1)}\underline{x} = \underline{b}^{(n-1)} \tag{2.1}$$

where the matrices $T^{(-i)}$ have zero elements along the i sub-diagonals, the matrices $T^{(i)}$ have zero elements along the i super-diagonals. Thus, $T^{(1-n)}$ is upper-triangular, and $T^{(n-1)}$ is lower-triangular.

Explicitly, $T^{(-i)}$ and $T^{(i)}$ have the forms



(2.2a), and



(2.2b)

respectively, for $i=1,2,\dots,n-1$. We have indicated in (2.2) that rows $i+1$ to n of $T^{(-i)}$, and rows 1 to $n-i$ of $T^{(i)}$, are Toeplitz. Henceforth, we will bound any Toeplitz block with continuous lines, and any non-Toeplitz block with dotted lines, as is done in (2.2). The notation in (2.2) also needs some explanation. The superscripts denote the iteration number and the subscripts denote the distance from the main diagonal - positive if above and negative below. In (2.2b), $t_0 := t_{11}$.

<u>The Bareiss Recursion</u>

We now show how $T^{(-i-1)}$ and $T^{(i+1)}$ are obtained from $T^{(-i)}$ and $T^{(i)}$, and prove (i) that this recursion requires only $4(n-i)$ operations, and (ii) $T^{(-i+1)}$and $T^{(i)}$ have the forms in (2.2), with i replaced by $i+1$, and an extra diagonal of zeros.

We first define the <u>shift matrices</u> Z_{-i} and Z_i :

$$\begin{array}{r l} & \\ \text{row } 1+i \rightarrow & \\ & \end{array} Z_{-i} := \begin{pmatrix} & & 0 \\ 1 & & \\ 0 & \ddots & \\ & & 1 \end{pmatrix} \qquad Z_i := \begin{pmatrix} 1 & & 0 \\ & \ddots & 1 \\ 0 & & \\ & & \end{pmatrix} \begin{array}{l} \\ \leftarrow \text{row } n-i \\ \\ \end{array} = Z_i^T$$

The effect of premultiplying any matrix A by Z_{-i} is to downshift A by i places, and replace the first i rows by zeros; premultiplying A by Z_i upshifts A by i places, replacing the last i rows by zeros.

The Bareiss recursion is:

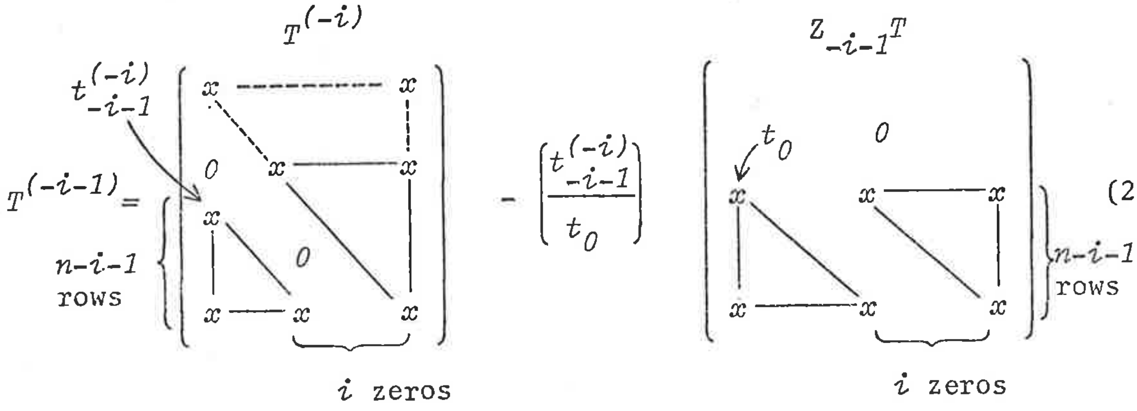
$$T^{(-i-1)} = T^{(-i)} - m_{-i-1}Z_{-i-1}T^{(i)}, \text{ where } m_{-i-1} = t^{(-i)}_{-i-1}/t_0 \qquad (2.3a)$$

$$T^{(i+1)} = T^{(i)} - m_{i+1}\, Z_{i+1}\, T^{(-i-1)}, \text{ where } m_{i+1} = t^{(i)}_{i+1}/t_0^{(i-1)} \qquad (2.3b)$$

The following Theorem shows the validity of the Bareiss recursion. The approach is different from that of Bareiss, who derives his algorithm, rather than stating it in a theorem and proving it; our proof is based on manipulating matrix forms - patterns of zeros and Toeplitz blocks - and it appears to be simpler than Bareiss's derivation. This approach will be useful in later Chapters when we consider more complicated variants of the Bareiss algorithm.

Theorem 2.1 If $T^{(-i)}$ and $T^{(i)}$ are of the forms (2.2a) and (2.2b) respectively, then (i) $T^{(-i-1)}$ and $T^{(i+1)}$ have the same respective form, with i replaced by $i+1$, and an extra diagonal of zeros, (ii) the recursion (2.3) requires $4(n-i)-5$ operations.

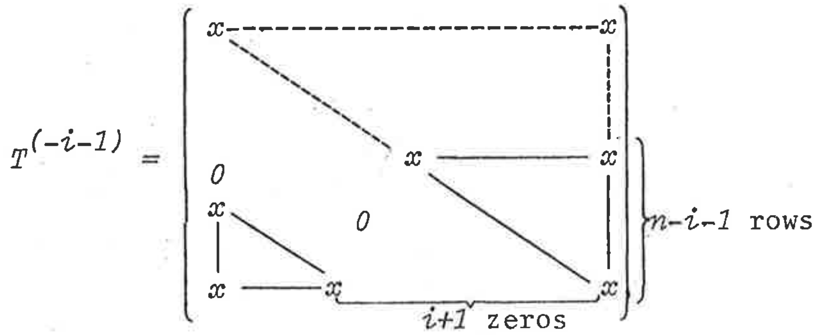
Proof: (i) Writing out the RHS of (2.3a) explicitly, we have



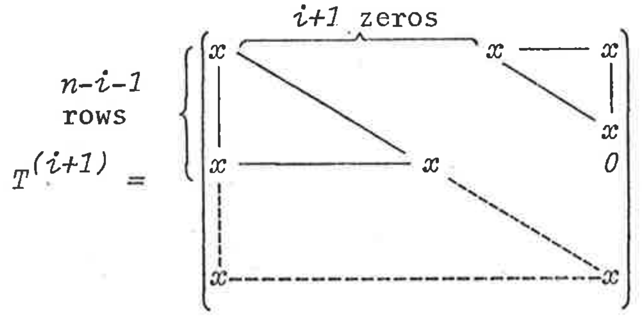
(2.4)

It is clear that $t^{(-i)}_{-i-1}$ is eliminated by t_0 in the operation in (2.4). In addition, the last $n-i-1$ rows of the output matrix of (2.4) are Toeplitz, and diagonals $(-1),(-2),\ldots,(-i)$ are zero.

These three results show that $T^{(-i-1)}$ has the form



It can be similarly shown that $T^{(i+1)}$ has the form



QED. (i).

(ii) Consider (2.4). There are $2(n-i-1)$ non-zero Toeplitz diagonals in the last $n-i-1$ rows $Z_{-i-1}\, T^{(i)}$, so there are as many operations (including the computation of $t^{(-i)}_{-i-1}/t_0$) in computing $T^{(-i-1)}$ using (2.4). Similarly there are $2(n-i)-3$ operations in computing $T^{(i+1)}$. QED. (ii).

The Bareiss Nonsymmetric Algorithm (BNA)

We can use the recursion (2.3) to produce $T^{(1-n)}$ and $T^{(n-1)}$ which have $n-1$ zero diagonals below and above the main diagonal respectively; $T^{(1-n)}$ and $T^{(n-1)}$ are therefore upper and lower triangular respectively. This algorithm is called the Bareiss nonsymmetric algorithm because (2.3b) cannot be obtained from (2.3a) by replacing $-i$ and $-i-1$ by i and $i+1$ respectively. There is also a symmetric version where this can be done.

Algorithm 2.1 - The Bareiss Nonsymmetric Algorithm (BNA)

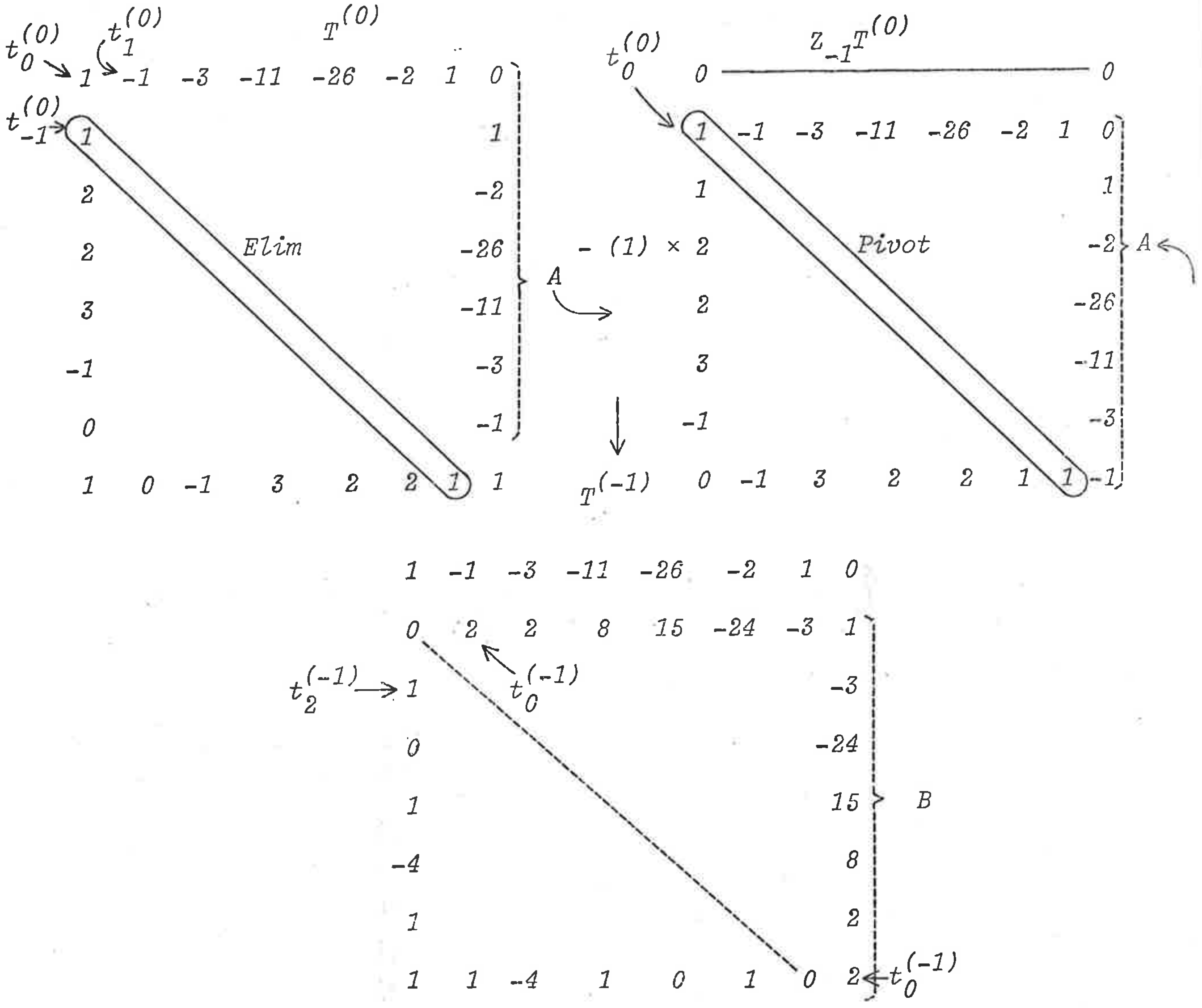
```
1.  Set $T^{(0)} \leftarrow T$

2.  For $i \leftarrow 0$ to $n-2$ do

    2.1  $T^{(-i-1)} \leftarrow T^{(-i)} - m_{-i-1} Z_{-i-1} T^{(i)}$  with  $m_{-i-1} = t_{-i-1}^{(-i)} / t_0$   (2.3a)

    2.2  $T^{(i+1)} \leftarrow T^{(i)} - m_{i+1} Z_{i+1} T^{(-i-1)}$  with  $m_{i+1} = t_{i+1}^{(i)} / t_0^{(-i-1)}$   (2.3b)
```

Operation Count   The number of operations in algorithm 2.1 is

$$\sum_{i=0}^{n-2} [4(n-i)-5] = 2n^2 + O(n) \text{ operations.}$$

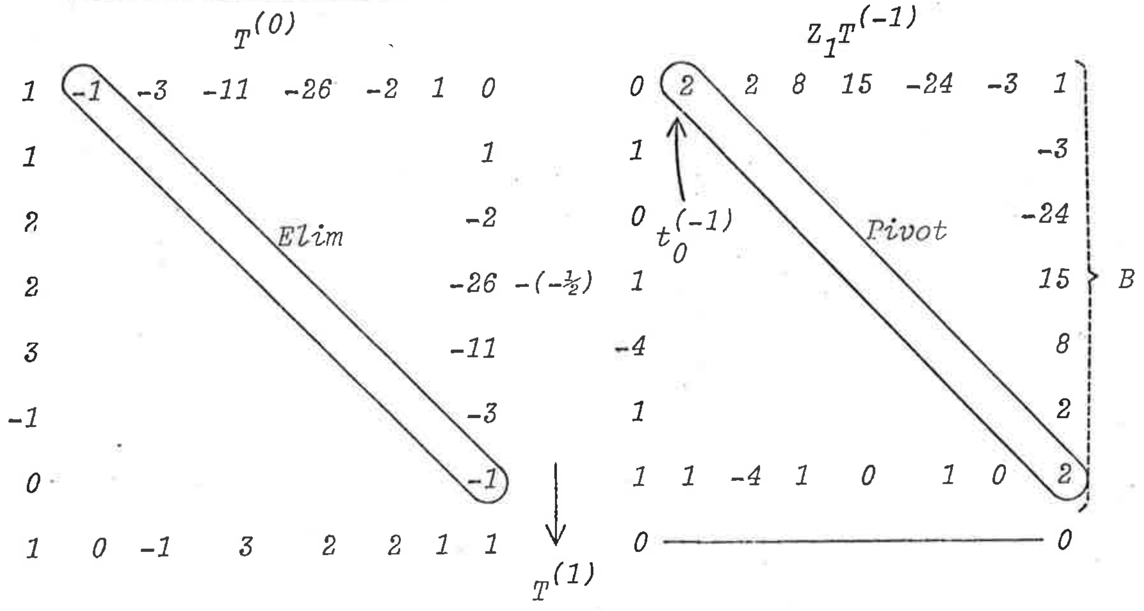
Example 2.1*- Steps (-1), (1) and (-2) of the Bareiss algorithm

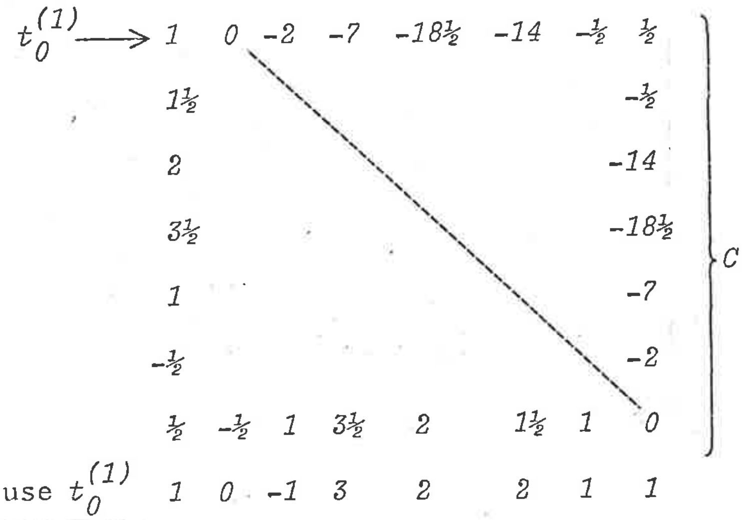
Step (-1) - use $t_0^{(0)}$ to elim $t_{-1}^{(0)}$:



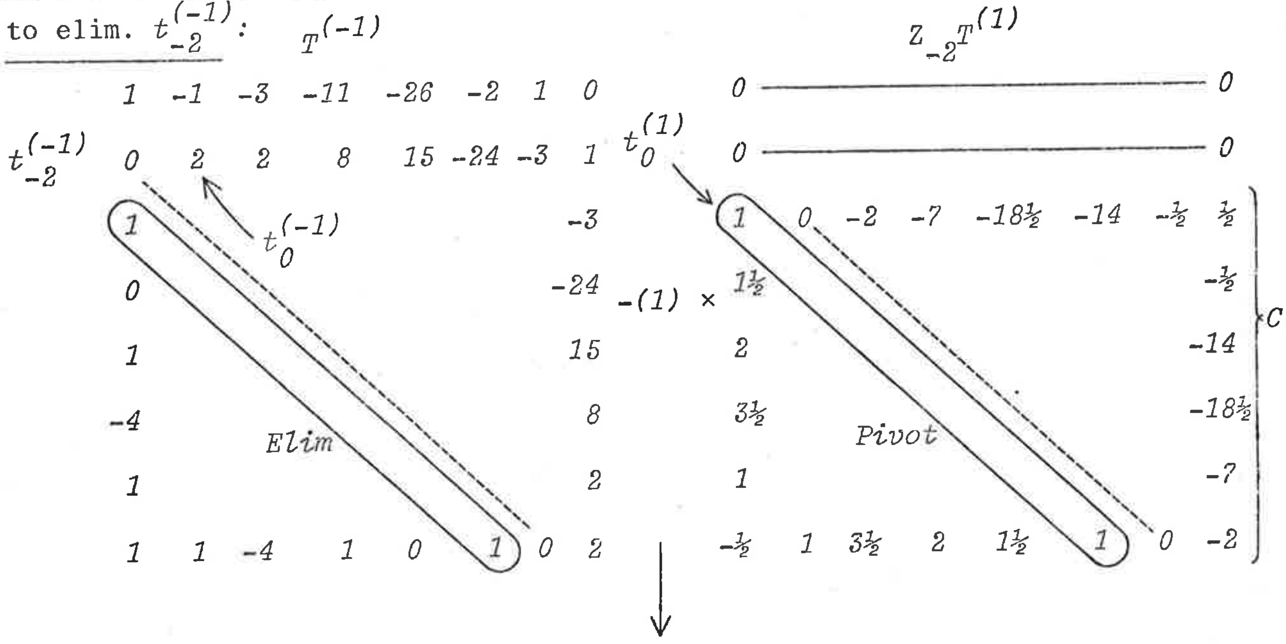
*In these examples, only the boundaries of Toeplitz blocks are shown.

Step (1) - use $t_0^{(-1)}$ to elim. $t_1^{(0)}$;





Step (-2) - use $t_0^{(1)}$ to elim. $t_{-2}^{(-1)}$:



$$\downarrow$$

$$T^{(-2)}$$

$$\begin{array}{cccccccc}
1 & -1 & -3 & -11 & -26 & -2 & -1 & 0 \\
0 & 2 & 2 & 8 & 15 & -24 & -3 & 1 \\
0 & 0 & 4 & 9 & 26\frac{1}{2} & 29 & -23\frac{1}{2} & -3\frac{1}{2} \\
-1\frac{1}{2} & \ddots & \ddots & & & & & -23\frac{1}{2} \\
-1 & & \ddots & \ddots & & & & 29 \\
-7\frac{1}{2} & & & \ddots & \ddots & & & 26\frac{1}{2} \\
0 & & & & \ddots & \ddots & & 9 \\
1\frac{1}{2} & 0 & -7\frac{1}{2} & -1 & -1\frac{1}{2} & 0 & 0 & 4
\end{array}$$

Note on steps and cycles: Bareiss step $-k$ (or k) calculates $T^{(-k)}$ (or $T^{(k)}$). Bareiss cycle k calculates both $T^{(-k)}$ and $T^{(k)}$. Hence, there are two steps per cycle.

3. COMPUTING THE FACTORS OF T USING THE BAREISS ALGORITHM (BNA)

In general, it would be expected that a triangular elimination algorithm would be related to the corresponding triangularization algorithm. For Gaussian elimination, the upper-triangular matrix produced by the reduction is the same as the U-matrix in the LU factorization. We will see this is also true for the Bareiss algorithm. The situation is different for the L-matrix: for Gaussian elimination, the array of multipliers (with the signs inverted) is the L-matrix. This is not true for the Bareiss algorithm. However, the Bareiss algorithm does produce a lower-triangular matrix, $T^{(n-1)}$, which is related to the desired L-matrix.

For compactness, we denote $T^{(1-n)}$ and $T^{(n-1)}$ by $T^{(-)}$ and $T^{(+)}$ respectively. Before we show the relationship between $T^{(\pm)}$ and the triangular factors, we need the following result:

Lemma 3.1 The upper-triangular matrix $T^{(-)}$ is related to T by

$$T^{(-)} = M^{(-)}T \tag{3.1}$$

where $M^{(-)}$ is unit lower-triangular; and the lower-triangular matrix $T^{(+)}$ is related to T by

$$T^{(+)} = M^{(+)}T \tag{3.2}$$

where $M^{(+)}$ is upper-triangular.

Proof: Bareiss [6] shows that $\underline{t}_{i.}^{(-)}$ is a linear combination of rows 1 to i of T, with the co-efficient of $\underline{t}_{i.}$ being unity, i.e. for some μ_{ij}, $j=1,\ldots,i-1$

$$\underline{t}_{i.}^{(-)} = (\mu_{i1},\mu_{i2},\ldots,\mu_{i,i-1},1)\begin{pmatrix}\underline{t}_{1.}\\ \vdots\\ \underline{t}_{i.}\end{pmatrix}$$

$$= (\mu_{i1},\mu_{i2},\ldots,\mu_{i,i-1},1,0,\ldots,0)T$$

$$\text{So } T^{(-)} = \begin{bmatrix} 1 & & & \\ \mu_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ \mu_{n1} & \cdots & \mu_{n,n-1} & 1 \end{bmatrix} T =: M^{(-)}T,$$

where $M^{(-)}$ is unit lower triangular. This proves (3.1). The proof of (3.2) follows similarly. QED

We now prove the main result of this section:

Theorem 3.1 Let $T=LU$, where L is unit lower-triangular and U is upper-triangular, and let $T^{(-)}$ and $T^{(+)}$ be the reduced matrices produced by the Bareiss algorithm (BNA). Then

$$U = T^{(-)} \tag{3.3a}$$

$$\text{and} \quad L = t_0^{-1}\, T^{(+)T2}, \tag{3.3b}$$

where $\cdot^{T2}$ denotes transposition about the secondary diagonal, or just secondary transpose.

Proof: From Lemma 3.1, $T^{(-)} = M^{(-)}T$

i.e. $T = (M^{(-)})^{-1}\, T^{(-)}$

Now $(M^{(-)})^{-1}$ is ULT because $M^{(-)}$ is, and $T^{(-)}$ is UT, so, by the uniqueness of this factorization, $(M^{(-)})^{-1}=L$ and $T^{(-)}=U$. This proves (3.3a).

For the second part, we have

$$T^{(+)} = M^{(+)}T, \tag{3.4}$$

where $M^{(+)}$ is upper-triangular by the above lemma. From (3.4),

$$T = M^{(+)-1}T^{(+)} \tag{3.5}$$

where $(M^{(+)})^{-1}$ is upper-triangular because $M^{(+)}$ is. In addition, by equation (2.2b), it can be seen that by putting $i=n-1$, $T^{(+)}$ is lower-triangular with t_0's along the diagonal.

Taking the secondary transpose of (3.5), and using the fact that $T^{T2} = T$ because T is Toeplitz,

$$T = (M^{(+)-1}T^{(+)})^{T2} \;;$$

but it can easily be shown that for any two matrices X and Y

$$(XY)^{T2} = Y^{T2}X^{T2}$$

(Use the fact that $X^{T2} = (EXE)^{T}$, where E is a matrix with 1's along the secondary diagonal and zeros elsewhere; the relation $EE = I$ is also needed);

$$\text{hence} \quad T = T^{(+)T2}(M^{(+)-1})^{T2}$$

$$= t_0^{-1}\, T^{(+)T2}\, t_0\, (M^{(+)-1})^{T2} \tag{3.6}$$

Now $T^{(+)T2}$ is lower-triangular with t_0's along the diagonal because $T^{(+)}$ has this form, so $t_0^{-1}\, T^{(+)T2}$ is unit lower-triangular. As well, $(M^{(+)-1})^{T2}$ is upper-triangular because $M^{(+)-1}$ is, hence $t_0(M^{(+)-1})^{T2}$ is. Thus in (3.6)

$$\left.\begin{aligned} L &= t_0^{-1}\, T^{(+)T2} \\ \text{and} \quad U &= t_0(M^{(+)-1})^{T2} \end{aligned}\right\} \tag{3.7}$$

The first equation in (3.7) is the same as (3.3b). QED.

Using the Bareiss algorithm to obtain L and U.

Theorem 3.1 indicates that the L and U factors may be obtained by executing the Bareiss algorithm and applying eqs. (7.3).

As mentioned in the previous section in the remark on operation counts, $2n^2 + O(n)$ operations are required to calculate $T^{(-)}$ and $T^{(+)}$; since $\frac{1}{2}n^2 + O(n)$ operations are required in (3.3b) to multiply $T^{(+)T2}$ by t_0^{-1}, $\frac{5}{2}n^2 + O(n)$ operations are required in all. This can be reduced to $2n^2 + O(n)$ operations by adapting the Bareiss algorithm to calculate L and U directly.

Adapting the Bareiss algorithm to calculate L and U directly.

If we modify Bareiss's algorithm to calculate $T^{(-)}$ and $t_0^{-1}T^{(+)}$ instead of $T^{(-)}$ and $T^{(+)}$, then by (3.3), the modified algorithm calculates L and U directly. Theorem 3.2 shows that this can be done by a very simple modification of Bareiss's algorithm:

Theorem 3.2 Let t_0, $T^{(+)}$ and $T^{(-)}$ be as produced by BNA (algorithm 2.1). Then $T^{(-)}$ and $t_0^{-1}T^{(+)}$ may be produced by the following modified procedure:

Algorithm 3.1 - The adapted Bareiss algorithm (ABA)

```
1.  Set $\tilde{T}^{(-0)} \leftarrow T$ and $\tilde{T}^{(+0)} \leftarrow t_0^{-1}T$ $\{\therefore\ t_0^{(+0)}=1\}$                        (3.8a,b)
2.  $\tilde{T}^{(-1)} \leftarrow \tilde{T}^{(-0)} - \tilde{m}_{-1} Z_{-1} \tilde{T}^{(0)}$, where $\tilde{m}_{-1} = \tilde{t}_{-1}^{(-0)}$                      (3.9a,b)
3.  $\tilde{T}^{(1)} \leftarrow \tilde{T}^{(+0)} - \tilde{m}_1 Z_1 \tilde{T}^{(-1)}$, where $\tilde{m}_1 = \tilde{t}_1^{(+0)}/\tilde{t}_0^{(-1)}$                  (3.10a,b)
4.  For $i \leftarrow 1$ to $n-2$ do
    4.1  $\tilde{T}^{(-i-1)} \leftarrow \tilde{T}^{(-i)} - \tilde{m}_{-i-1} Z_{-i-1} \tilde{T}^{(i)}$, where $\tilde{m}_{-i-1} = \tilde{t}_{-i-1}^{(-i)}$        (3.11a,b)
    4.2  $\tilde{T}^{(i+1)} \leftarrow \tilde{T}^{(i)} - \tilde{m}_{i+1} Z_{i+1} \tilde{T}^{(-i-1)}$, where $\tilde{m}_{i+1} = \tilde{t}_{i+1}^{(i)}/\tilde{t}_0^{(-i-1)}$   (3.12a,b)
```

Proof: Easy - show by induction that for $i=0,\ldots,n-1$, $\tilde{T}^{(-i)} = T^{(-i)}$, $\tilde{T}^{(i)} = t_0^{-1}T^{(i)}$.

Compact Version of ABA

The whole of the $\tilde{T}^{(\pm i)}$ do not have to be stored at each stage of ABA, since by eqs. (2.1a), only $\underline{\tilde{t}}_{i+1,i+1:n}^{(-i)}$ and $\underline{\tilde{t}}_{i+2:n,1}^{(-i)}$, need be calculated to specify $\tilde{T}^{(-i)}$, and only $\underline{\tilde{t}}_{1,i+1:n}^{(i)}$ and $\underline{\tilde{t}}_{1:n-i,1}^{(i)}$ need be calculated to specify $\tilde{T}^{(i)}$. Hence Algorithm 3.1 can be written in the following compact form using the fact that $\underline{\tilde{t}}_{i+2,i+2:n}^{(-i)} = \underline{\tilde{t}}_{i+1,i+1:n-1}^{(-i)}$ (by Toeplicity) in steps 2(a) and 4.1(a).

Algorithm 3.2 - ABA (compact form)

```
1.  $\underline{\tilde{t}}_{1.}^{(-0)} \leftarrow \underline{t}_{1.}$ ; $\underline{\tilde{t}}_{2:n,1}^{(-0)} \leftarrow \underline{t}_{2:n,1}$. ; $\underline{\tilde{t}}_{1,2:n}^{(+0)} \leftarrow \underline{t}_{1,2:n}/t_0$; $\underline{\tilde{t}}_{.1}^{(+0)} \leftarrow \underline{t}_{.1}/t_0$

2.  (a) $\underline{\tilde{t}}_{2,2:n}^{(-1)} \leftarrow \underline{\tilde{t}}_{1,1:n-1}^{(-0)} - \tilde{m}_{-1}\,\underline{\tilde{t}}_{1,2:n}^{(+0)}$    }
                                                               } where $\tilde{m}_{-1} = \tilde{t}_{-1}^{(-0)}$
    (b) $\underline{\tilde{t}}_{3:n,1}^{(-1)} \leftarrow \underline{\tilde{t}}_{3:n,1}^{(-0)} - \tilde{m}_{-1}\,\underline{\tilde{t}}_{2:n-1,1}^{(+0)}$  }

3.  (a) $\underline{\tilde{t}}_{3:n,1}^{(1)} \leftarrow \underline{\tilde{t}}_{3:n,1}^{(+0)} - \tilde{m}_1\,\underline{\tilde{t}}_{3:n,1}^{(-1)}$    }
                                                               } where $\tilde{m}_1 = \tilde{t}_1^{(+0)}/\tilde{t}_0^{(-1)}$
    (b) $\underline{\tilde{t}}_{1:n-1,1}^{(1)} \leftarrow \underline{\tilde{t}}_{1:n-1,1}^{(+0)} - \tilde{m}_1\,\underline{\tilde{t}}_{2:n,1}^{(-1)}$  }
```

```
4. For $i \leftarrow 1$ to $n-2$ do:
    4.1 (a) $\underline{\tilde{t}}^{(-i-1)}_{i+2,i+2:n} \leftarrow \underline{\tilde{t}}^{(-i)}_{i+1,i+1:n-1} - \tilde{m}_{-i-1}\underline{\tilde{t}}^{(i)}_{1,i+2:n}$
        (b) $\underline{\tilde{t}}^{(-i-1)}_{i+3:n,1} \leftarrow \underline{\tilde{t}}^{(-i)}_{i+3:n,1} - \tilde{m}_{-i-1}\underline{\tilde{t}}^{(i)}_{2:n-i-1,1}$
        where $\tilde{m}_{-i-1} = \tilde{t}^{(-i)}_{-i-1}$
    4.2 (a) $\underline{\tilde{t}}^{(i+1)}_{1,i+3:n} \leftarrow \underline{\tilde{t}}^{(i)}_{1,i+3:n} - \tilde{m}_{i+1}\underline{\tilde{t}}^{(-i-1)}_{i+2,i+3:n}$
        (b) $\underline{\tilde{t}}^{(i+1)}_{1:n-i-1,1} \leftarrow \underline{\tilde{t}}^{(i)}_{1:n-i-1,1} - \tilde{m}_{i+1}\underline{\tilde{t}}^{(-i-1)}_{i+2:n,1}$
        where $\tilde{m}_{i+1} = \tilde{t}^{(i)}_{i+1}/\tilde{t}^{(-i-1)}_{0}$
```

Calculation of L and U using the Adapted Bareiss Algorithm (ABA)

Theorems 3.1 and 3.2 enable us to calculate L and U as follows:

Algorithm 3.3 - Calculation of L and U using ABA

```
1. Calculate $\tilde{T}^{(-)}$ and $\tilde{T}^{(+)}$ using ABA.
2. Then $L=\tilde{T}^{(+)T2}$ and $U=\tilde{T}^{(-)}$.
```

Operation Counts

It is easy to see that ABA requires $2n^2 + O(n)$ operations to calculate $\tilde{T}^{(-)}$ and $\tilde{T}^{(+)}$, and hence L and U. To solve a Toeplitz system with k RHS's, ABA therefore requires $(2+k)n^2 + O(n)$ operations. By contrast, Bareiss's algorithm requires $(2 + \frac{5}{4}k)n^2 + O(n)$ operations.

4. DERIVATION OF FTF*ALGORITHMS FROM RANK-1 UPDATERS

In this section we show how to derive FTF algorithms by relating them simply to rank-1 update algorithms. We first show how LU-FTF algorithms can be derived from LU-updaters, then show how LDR-FTF algorithms can be derived form LDR updaters. We then apply these results by deriving an LDR-FTF algorithm from Bennett's LDR updater, and an LU-FTF from a

*Fast Toeplitz factorization

modified version of Bennett's algorithm.

4.1 Relation Between *LU* Rank-1 Updaters and Toeplitz *LU* Factorizers

The rank-1 update problem often has the form: suppose $A=LU$, where L is unit lower-triangular and U is upper-triangular. Find $\tilde{L}$ unit lower-triangular and $\tilde{U}$ upper-triangular such that

$$\tilde{L}\tilde{U} = A + \underline{x}\underline{y}^T = LU + \underline{x}\underline{y}^T \tag{4.1}$$

Several algorithms requiring only $O(n^2)$ operations are available to find $\tilde{L}$ and $\tilde{U}$. Some of these can be written in the form

1. $\{\underline{x}^{(1)}, \underline{y}^{(1)}\} \leftarrow F\{\underline{x}, \underline{y}\}$ (4.2a)

2. For $i \leftarrow 1$ to n do $\{\tilde{\underline{l}}_{\cdot i}, \tilde{\underline{u}}_{i\cdot}, \underline{x}^{(i+1)}, \underline{y}^{(i+1)}\} \leftarrow R\{\underline{l}_{\cdot i}, \underline{u}_{i\cdot}, \underline{x}^{(i)}, \underline{y}^{(i)}\}$ (4.2b)

where the $\{\underline{x}^{(i)}, \underline{y}^{(i)}\}$ are auxiliary vectors, F is an initialization procedure, and R is a set of recursion relations requiring $O(n)$ operations.

The following theorem shows how an FTF algorithm can be very simply derived from a rank-1 updater of the type (4.2).

Theorem 4.1 Let T be an $n \times n$ Toeplitz matrix, and let $\hat{L}$, a unit lower-triangular matrix, and $\hat{U}$, an upper-triangular matrix, be such that

$$T = \hat{L}\hat{U} \quad ; \tag{4.3}$$

let F and R specify a rank-1 updater of the type (2.2). Then the following algorithm may be used to find $\hat{L}$ and $\hat{U}$ in $O(n^2)$ operations:

Algorithm 4.1

1. $\hat{\underline{l}}_{\cdot 1} \leftarrow \underline{t}_{\cdot 1}/t_{11}$; $\hat{\underline{u}}_{1\cdot} \leftarrow \underline{t}_{1\cdot}$ (4.4a)

2. $\{\underline{x}^{(1)}, y^{(1)}\} \leftarrow F\{\underline{t}_{2:n,1}, -\underline{t}^T_{1,2:n}/t_{11}\}$ (4.4b)

3. For $i \leftarrow 1$ to $n-1$ do
   $\{\hat{\underline{l}}_{2:n,i+1}, \hat{\underline{u}}_{i+1,2:n}, \underline{x}^{(i+1)}, \underline{y}^{(i+1)}\} \leftarrow R\{\hat{\underline{l}}_{1:n-1,i}, \hat{\underline{u}}_{i,1:n-1}, \underline{x}^{(i)}, y^{(i)}\}$
   (4.4c)

<u>Proof:</u> Let $\dot{T}$, $\dot{L}$ and $\dot{U}$ be the $(n-1)$th order leading submatrices of T, $\hat{L}$ and $\hat{U}$ respectively. Then,

$$\dot{T} = \dot{L}\dot{U} \tag{4.5}$$

Let $\mathring{T}$, $\mathring{L}$ and $\mathring{U}$ be the $(n-1)$th order trailing submatrices of T, $\hat{L}$ and $\hat{U}$ respectively, let $t{:=}t_{11}$, $\underline{v}{:=}\underline{t}_{2:n,1}$ and $\underline{u}^T{:=}\underline{t}_{1,2:n}$. Because T is Toeplitz, $\mathring{T}=\dot{T}$, so (4.3) can be written as follows in partitioned form:

$$T = \left(\begin{array}{c|c} t & \underline{u}^T \\ \hline \underline{v} & \dot{T} \end{array}\right) = \left(\begin{array}{c|c} 1 & \underline{0}^T \\ \hline t^{-1}\underline{v} & \mathring{L}\diagdown \end{array}\right) \left(\begin{array}{c|c} t & \underline{u}^T \\ \hline \underline{0} & \diagdown\mathring{U} \end{array}\right) = \hat{L}\hat{U} \tag{4.6}$$

The trailing submatrices on both sides of (4.6) are

$$\dot{T} = \mathring{L}\mathring{U} + t^{-1}\underline{v}\underline{u}^T \tag{4.7}$$

(4.5) and (4.7) together yield

$$\mathring{L}\mathring{U} = \dot{L}\dot{U} - t^{-1}\underline{v}\underline{u}^T \tag{4.8}$$

(4.8) is a rank-1 update problem, so by (4.2) $\mathring{L}$ and $\mathring{U}$ are determined from $\dot{L}$ and $\dot{U}$ by the procedure

```
1.  {$\underline{x}^{(1)}, y^{(1)}$} ← F{$\underline{v}$, $-\underline{u}/t$}                (4.9a)

2.  For $i$ ← $1$ to $n-1$ do
        {$\mathring{\underline{l}}_{\cdot i}, \mathring{\underline{u}}_{i\cdot}, \underline{x}^{(i+1)}, \underline{y}^{(i+1)}$}←R{$\dot{\underline{l}}_{\cdot i}, \dot{\underline{u}}_{i\cdot}; \underline{x}^{(i)}, y^{(i)}$}    (4.9b)
```

From the definitions of $\dot{L}$ and $\mathring{L}$, (4.9b) can be rewritten

```
2.  For $i$ ← $1$ to $n-1$ do
    {$\hat{\underline{l}}_{2:n,i+1}, \hat{\underline{u}}_{i+1,2:n}, \underline{x}^{(i+1)}, \underline{y}^{(i+1)}$}←R{$\hat{\underline{l}}_{1:n-1,i}, \hat{\underline{u}}_{i,1:n-1}, \underline{x}^{(i)}, \underline{y}^{(i)}$}
                                                                  (4.10)
```

Now, eq.(4.10) in fact constitutes a recursion for calculating the columns of $\hat{L}$ and the rows of $\hat{U}$ in $O(n^2)$ operations. $\underline{x}^{(1)}$ and $\underline{y}^{(1)}$ can be initialized by (4.9a), and $\hat{\underline{l}}_{\cdot 1}$ and $\hat{\underline{u}}_{1\cdot}$ can be initialized by (4.4a), the proof of which is trivial. Hence, (4.4a),(4.9a) and (4.10) constitute the desired algorithm for factorizing T. QED.

4.2 Relation Between *LDR* Rank-1 Updaters and Toeplitz *LDR* Factorizers

A result similar to Theorem 4.1 enables *LDR* Toeplitz factorization algorithms to be derived from a certain class of *LDR* rank-1 updaters:

Theorem 4.2 Consider the rank-1 update problem

$$\widetilde{LDR} = LDR + \sigma \underline{x}\underline{y}^T \tag{4.11}$$

Suppose the rank-1 updater for (4.11) has the form

```
(1)  Initialize $\{\underline{x}^{(1)},\underline{y}^{(1)},\sigma^{(1)}\} \leftarrow P\{\underline{x},\underline{y},\sigma,\underline{l}_{\cdot 1},d_1,\underline{r}_{1\cdot}\}$
(2)  For $i \leftarrow 1$ to $n$ do $\{\tilde{\underline{l}}_{\cdot i},\tilde{d}_i,\tilde{\underline{r}}_{i\cdot},\underline{x}^{(i+1)},\underline{y}^{(i+1)},\sigma^{(i+1)}\} \leftarrow R\{\underline{l}_{\cdot i},d_i,\underline{r}_{i\cdot},\underline{x}^{(i)},\underline{y}^{(i)},\sigma^{(i)}\}$
                                                                                    (4.12a-b)
```

then the FTF algorithm for T has the form

```
(1)  Initialize $\underline{l}_{\cdot 1},d_1,\underline{r}_{1\cdot}$
(2)  $\underline{x} \leftarrow \underline{t}_{2:n-1,1}/t_{11}$;  $\underline{y} \leftarrow \underline{t}_{1,2:n-1}/t_{11}$;   $\sigma \leftarrow -t_{11}$
(3)  $\{\underline{x}^{(1)},\underline{y}^{(1)},\sigma^{(1)}\} \leftarrow P\{\underline{x},\underline{y},\sigma,\underline{l}_{\cdot 1},d_1,\underline{r}_{1\cdot}\}$
(4)  For $i \leftarrow 1$ to $n-1$ do $\{\underline{l}_{2:n,i+1},\underline{r}_{i+1,2:n},d_{i+1},\underline{x}^{(i+1)},\underline{y}^{(i+1)},\sigma^{(i+1)}\}$
                    $\leftarrow R\{\underline{l}_{1:n-1,i},\underline{r}_{i,1:n-1},d_i,\underline{x}^{(i)},\underline{y}^{(i)},\sigma^{(i)}\}$      (4.13a-d)
```

Proof: Analogous to that for Theorem 4.1.

4.3 Derivation of a Toeplitz LDR-Factorizer From the Bennett Algorithm

Bennett's algorithm calculates a rank-p update of the LDR factors, i.e. it finds $\tilde{L}, \tilde{D}$ and $\tilde{R}$ such that

$$\tilde{L}\tilde{D}\tilde{R} = LDR + XCY^T$$

where C is $p{\times}p$, and X and Y are $n{\times}p$. We consider only the rank-1 update: find $\tilde{L}$, $\tilde{D}$ and $\tilde{R}$ such that

$$\tilde{L}\tilde{D}\tilde{R} = LDR + c\underline{x}\underline{y}^T. \qquad (4.14)$$

The algorithm in this case is [7]:

Algorithm 4.2 - Bennett's Rank-1 algorithm (BR1A)

```
1.  Set $c^{(1)} \leftarrow c$, $\underline{x}^{(1)} \leftarrow \underline{x}$ and $\underline{y}^{(1)} \leftarrow \underline{y}$          (4.15a-c)
2.  Set $i \leftarrow 1$
{Main loop}
3.  Repeat step 3.1 - 3.10, stopping at 3.3 when $i=n$.
    3.1  $p^{(i)} \leftarrow c^{(i)} x_i^{(i)}$                                        (4.16a)
    3.2  $\tilde{d}_{ii} \leftarrow d_{ii} + p^{(i)} y_i^{(i)}$                          (4.16b)
    3.3  Stop if $i=n$
    3.4  $q^{(i)} \leftarrow c^{(i)} y_i^{(i)}$                                        (4.16c)
    3.5  $\underline{x}^{(i+1)} \leftarrow \underline{x}^{(i)} - x_i^{(i)} \underline{l}_{\cdot i}$              (4.16d)
    3.6  $\underline{y}^{(i+1)} \leftarrow \underline{y}^{(i)} - y_i^{(i)} \underline{r}_{i\cdot}$.             (4.16e)
    3.7  $\tilde{\underline{l}}_{\cdot i} \leftarrow \underline{l}_{\cdot i} + \tilde{d}_{ii}^{-1} q^{(i)} \underline{x}^{(i+1)}$       (4.16f)
    3.8  $\tilde{\underline{r}}_{i\cdot} \leftarrow \underline{r}_{i\cdot} + \tilde{d}_{ii}^{-1} p^{(i)} \underline{y}^{(i+1)}$       (4.16g)
    3.9  $c^{(i+1)} \leftarrow c^{(i)} - \tilde{d}_{ii}^{-1} q^{(i)} p^{(i)}$                (4.16h)
    3.10 Increment $i$, and go to 3.1.
```

It can be checked that BR1A requires $2n^2 + 4n-4$ operations.

A Toeplitz *LDR* factorizer may be derived from Algorithm 4.2 by applying Theorem 4.2. In this case, eqs. (4.15) constitute the initialization procedure P, and eqs. (4.16) constitute the recursion relation R.

<u>Algorithm 4.3</u> - Toeplitz *LDR* factorizer (TLDR)

```
1.  $\underline{l}_{\cdot 1} \leftarrow \underline{t}_{\cdot 1}/t_0$ ; $\underline{r}_{1\cdot} \leftarrow \underline{t}_{1\cdot}/t_0$ ; $d_1 \leftarrow t_0$

2.  $\underline{x} \leftarrow \underline{t}_{2:n-1,1}/t_0$ ; $\underline{y} \leftarrow \underline{t}_{1,2:n}/t_0$ ; $c \leftarrow -t_0$

3.  $c^{(1)} \leftarrow \underline{c}$ ; $\underline{x}^{(1)} \leftarrow \underline{x}$ ; $\underline{y}^{(1)} \leftarrow \underline{y}$

4.  Set $i \leftarrow 1$
{Main loop}
5.  Repeat steps 5.1-5.10 stopping at 5.3 when $i=n-1$.
    5.1  $p^{(i)} \leftarrow c^{(i)} x_i^{(i)}$

    5.2  $d_{i+1} \leftarrow d_i + p^{(i)} y_i^{(i)}$

    5.3  Stop if $i=n-1$.

    5.4  $q^{(i)} \leftarrow c^{(i)} y_i^{(i)}$

    5.5  $\underline{x}^{(i+1)} \leftarrow \underline{x}^{(i)} - x_i^{(i)} \; \underline{l}_{1:n-1,i}$

    5.6  $\underline{y}^{(i+1)} \leftarrow \underline{y}^{(i)} - y_i^{(i)} \; \underline{r}_{i,1:n-1}$

    5.7  $\underline{l}_{2:n,i+1} \leftarrow \underline{l}_{1:n-1,i} + d_{i+1}^{-1} \, q^{(i)} \; \underline{x}^{(i+1)}$

    5.8  $\underline{r}_{i+1,2:n} \leftarrow \underline{r}_{i,1:n-1} + d_{i+1}^{-1} \, p^{(i)} \; \underline{y}^{(i+1)}$

    5.9  $c^{(i+1)} \leftarrow c^{(i)} - d_{i+1}^{-1} q^{(i)} p^{(i)}$
    5.10 Increment $i$, and go to 5.1.
```

It can be checked that TLDR requires $2n^2 + 6n - 4$ operations.

4.4 Modification of Bennett's Algorithm to Update the *LU*, Rather than the LDR Factorization

If we make the transformations $\tilde{U} = \tilde{D}\tilde{R}$, $L = \tilde{L}$ and $\underline{z} = \underline{cy}$, eq. (4.14) can be written

$$\tilde{L}\tilde{U} = LU + \underline{xz}^T \tag{4.17}$$

which is the *LU* update problem. By observing that

$$u_{ii} = d_{ii} \tag{4.18}$$

$$\tilde{u}_{ii} = \tilde{d}_{ii} \tag{4.19}$$

and making the transformations

$$\underline{u}_{i\cdot} = d_{ii}\underline{r}_{i\cdot} \tag{4.20}$$

$$\tilde{\underline{u}}_{i\cdot} = \tilde{d}_{ii}\,\tilde{\underline{r}}_{i\cdot} \tag{4.21}$$

$$\underline{z}^{(i)} = c^{(i)}\underline{y}^{(i)}, \tag{4.22}$$

we can express all the quantities in algorithm (4.2) in forms of $\underline{l}_{i\cdot}$, $\underline{u}_{i\cdot}$, $\tilde{\underline{l}}_{i\cdot}$, $\tilde{\underline{u}}_{i\cdot}$, $\underline{x}^{(i)}$ and $\underline{y}^{(i)}$, yielding an *LU*-update algorithm. We now do this.

Putting (4.22) in (4.15c) the latter becomes

$$\underline{z}^{(1)} = \underline{z} \tag{4.23}$$

Putting (4.19), (4.18) and (4.16a) in (4.16b), the latter becomes

$$\begin{aligned}\tilde{u}_{ii} &= u_{ii} + c^{(i)}x_i^{(i)}y_i^{(i)}\\ &= u_{ii} + x_i^{(i)}z_i^{(i)},\end{aligned} \tag{4.24}$$

using (4.22).

Putting (4.19), (4.16c) and (4.22) in (4.16f), the latter becomes

$$\tilde{\underline{l}}_{\cdot i} = \underline{l}_{\cdot i} + \tilde{u}_{ii}^{-1}\,z_i^{(i)}\underline{x}^{(i+1)} \tag{4.25}$$

Putting (4.16e) in (4.16g), multiplying through by $\tilde{d}_{ii}$ and using (4.21), (4.16g) becomes

$$\underline{\tilde{u}}_{i.} = \tilde{d}_{ii}\, \underline{r}_{i.} + p^{(i)}(\underline{y}^{(i)} - y_i^{(i)} \underline{r}_{i.}),$$

which, with (4.16b) becomes

$$\underline{\tilde{u}}_{i.} = d_{ii}\underline{r}_{i.} + p^{(i)}\underline{y}^{(i)}$$

$$= \underline{u}_{i.} + x_i^{(i)} \underline{z}^{(i)}, \text{ using (4.20), (4.16a) and (4.11).} \qquad (4.26)$$

Multiplying (4.16e) through by $c^{(i+1)}$ and using (4.16h), (4.16c) and (4.16a), (4.16e) becomes

$$\underline{z}^{(i+1)} = c^{(i)}(1 - \tilde{d}_{ii}^{-1}\, x_i^{(i)} z_i^{(i)})(\underline{y}^{(i)} - y_i^{(i)} \underline{r}_{i.})$$

$$= (1 - \tilde{d}_{ii}^{-1}\, x_i^{(i)} z_i^{(i)})(\underline{z}^{(i)} - z_i^{(i)} \underline{r}_{i.}) \text{ using (4.22)}$$

By (4.16b) the first bracketed term is $d_{ii}/\tilde{d}_{ii}$, so

$$\underline{z}^{(i+1)} = (1 - \tilde{d}_{ii}^{-1} x_i^{(i)} z_i^{(i)})\underline{z}^{(i)} - \tilde{d}_{ii}^{-1} z_i^{(i)} \underline{u}_{i.},$$

$$= \underline{z}^{(i)} - \tilde{d}_{ii}^{-1}\, z_i^{(i)} \underline{\tilde{u}}_{i.} \quad \text{(using (4.26))}$$

$$= \underline{z}^{(i)} - \tilde{u}_{ii}^{-1}\, z_i^{(i)} \underline{\tilde{u}}_{i.} \qquad (4.27)$$

Eqs. (4.15a), (4.23), (4.16d), (4.26), (4.27) and (4.25) form the basis of an *LU*-updating algorithm. Assembling these and replacing *z* by *y*, we get

<u>Algorithm 4.4</u> - Modified Bennett algorithm to solve (4.1) (MBA)

```
1.  $\underline{x}^{(1)} \leftarrow \underline{x}$, $\underline{y}^{(1)} \leftarrow \underline{y}$                      (4.28a,b)
2.  Set $i \leftarrow 1$
3.  Repeat 3.1 - 3.10, stopping at 3.2 when $i=n$
    3.1  $\tilde{u}_{ii} \leftarrow u_{ii} - x_i^{(i)} y_i^{(i)}$                    (4.28c)
    3.2  Stop if $i = n$
    3.3  $\underline{x}^{(i+1)} \leftarrow \underline{x}^{(i)} - x_i^{(i)} \underline{l}_{\cdot i}$                    (4.28d)
    3.4  $\underline{\tilde{u}}_{i\cdot} \leftarrow \underline{u}_{i\cdot} + x_i^{(i)} \underline{y}^{(i)}$                    (4.28e)
    3.5  $\underline{y}^{(i+1)} \leftarrow \underline{y}^{(i)} - \tilde{u}_{ii}^{-1} y_i^{(i)} \underline{\tilde{u}}_{i\cdot}$                    (4.28f)
    3.6  $\underline{\tilde{l}}_{\cdot i} \leftarrow \underline{l}_{\cdot i} + \tilde{u}_{ii}^{-1} y_i^{(i)} \underline{x}^{(i+1)}$                    (4.28g)
    3.7  Replace $i$ by $i+1$ ; go to (4.28c).
```

Operation Counts

The number of operations required to execute algorithm (4.28) is $2n^2 + n - 1$ compared to $2n^2 + 4n - n$ for the Bennett algorithm as described by equations (4.16); hence the present algorithm is slightly quicker (by $3n-3$ operations) than the Bennett algorithm for this updating problem.

4.5 Derivation of Toeplitz *LU* Factorizer from the Bennett Algorithm

A Toeplitz *LU*-factorizer may be derived from algorithm 4.4 by applying Theorem 4.1. Here, eqs. (4.28a,b) constitute the initialization procedure P, and eqs. (4.28c - 4.28g) constitute the recursion R.

Algorithm 4.5 - Toeplitz *LU*-factorizer

```
1.  $\underline{l}_{\cdot 1} \leftarrow \underline{t}_{\cdot 1}/t_0$ ; $\underline{u}_{1\cdot} \leftarrow \underline{t}_{1\cdot}$                    (4.29a,b)
2.  $\underline{x}^{(1)} \leftarrow \underline{t}_{2:n,1}$ ; $y^{(1)} \leftarrow -\underline{t}^T_{1,2:n}/t_0$                    (4.29c,d)
3.  For $i \leftarrow 1$ to $n-1$ do:
    3.1  $\underline{x}^{(i+1)} \leftarrow \underline{x}^{(i)} - x_i^{(i)} \underline{l}_{1:n-1,i}$                    (4.29e)
    3.2  $\underline{u}_{i+1,2:n} \leftarrow \underline{u}_{i,1:n-1} + x_i^{(i)} \underline{y}^{(i)}$                    (4.29f)
    3.3  $\underline{y}^{(i+1)} \leftarrow \underline{y}^{(i)} - u^{-1}_{i+1,i+1} y_i^{(i)} \underline{u}_{i+1,2:n}$                    (4.29g)
    3.4  $\underline{l}_{2:n,i+1} \leftarrow \underline{l}_{1:n-1,i} + u^{-1}_{i+1,i+1} y_i^{(i)} \underline{x}^{(i+1)}$                    (4.29h)
```

Relation between Algorithm 4.5 and ABA

By making the transformations $\underline{u}_{i+1\cdot} \rightarrow (\underline{0}^T, \underline{\tilde{t}}^{(-i)}_{i+1,i+1:n})$, $\underline{l}_{\cdot i+1} \rightarrow (\underline{0}^T, \underline{\tilde{t}}^{(i)}_{1:n-i,1})^T$, $\underline{x}^{(i+1)} \rightarrow (\underline{0}^T, \underline{\tilde{t}}^{(-i)T}_{i+2:n,1})^T$ and $\underline{y}^{(i+1)} \rightarrow (\underline{0}^T, -\underline{\tilde{t}}^{(i)}_{1,i+2:n})$, algorithm 4.5 can be transformed to algorithm 3.2, the ABA. Hence the ABA and the Toeplitz *LU*-factorizer are equivalent.

5. CONCLUSION

It is shown that Toeplitz factorization algorithms have been derived in two ways: from Bareiss's elimination algorithm, and from rank-1 update algorithms. A Toeplitz *LU*-factorizer derived from a modified version of Bennett's rank-1 update algorithm is shown to be the same as that derived from Bareiss's algorithm.

CHAPTER 3

THE RELATION BETWEEN TOEPLITZ ELIMINATION AND INVERSION ALGORITHMS

1. INTRODUCTION

In this chapter we derive a relationship between one of Bareiss's Toeplitz elimination algorithms [6] and the well-known Toeplitz inversion algorithm of Trench [84] and Zohar [92].

We describe Zohar's formulation in Section 2, noting a modification by Jain [48] which accelerates the Trench-Zohar algorithm (TZA) when the order of the Toeplitz matrix T is greater than 16. In Section 3, we review Bareiss's symmetric algorithm (BSA). (Note that the term "symmetric" refers to the structure of the algorithm, and does not mean that BSA can only be applied to symmetric matrices.) We then derive an extended BSA (EBSA) which calculates two upper-triangular and two lower-triangular matrices, and show that if L and U are the triangular factors of T, then the two upper-triangular matrices produced by EBSA are closely related to U and U^{-1}, and the two lower-triangular matrices are closely related to L and L^{-1}. EBSA is similar to an algorithm by Rissanen [79], but calculates L column-by-column and U row-by-row, whereas Rissanen's algorithm calculates L row-by-row and U column-by-column.

In Section 4, we modify EBSA, giving the Alternative Bareiss Symmetric Algorithm (ABSA), which calculates L^{-1} and U^{-1} only (in fewer operations than for EBSA). We then derive a relationship between ABSA and TZA. In a later chapter (Chapter 5), a form of pivoting will be incorporated into ABSA to improve its numerical performance. This relationship allows pivoting to be applied to TZA.

Some related work has been done by Bultheel [14], [15], in which a duality between Toeplitz factorization and inversion algorithms is shown; the approach is via a matrix interpretation of Padé approximation algorithms. In the present work, the approach is to extend the Toeplitz elimination algorithm of Bareiss in a straightforward manner: no mention of Padé theory is made.

2. THE TRENCH-ZOHAR ALGORITHM

We state without proof the Trench-Zohar algorithm (TZA) for Toeplitz inversion. Some changes of notation have been made to conform with the notation in the Bareiss algorithm (introduced later).

Let the input $n \times n$ Toeplitz $T = [t_{j-i}]_{i,j=0}^{n-1}$ be partitioned as follows,

$$T =: t_0 \left[\begin{array}{c|c} 1 & \underline{u}^T \\ \hline \underline{v} & t_0^{-1} T_{n-1} \end{array}\right] ,$$

let B, the inverse of T, be partitioned as follows,

$$B =: \frac{1}{t_0 \lambda_{n-1}} \left[\begin{array}{c|c} 1 & \underline{c}^T \\ \hline \underline{d} & M \end{array}\right] .$$

Now, define T_{i+1} as the *(i+1)*th leading submatrix of T, and let $B_{i+1} := T_{i+1}^{-1}$ $(i=1, \ldots, n-1)$ be partitioned as follows,

$$B_{i+1} =: \frac{1}{t_0\lambda_i} \left(\begin{array}{c|c} 1 & \underline{c}_i^T \\ \hline \underline{d}_i & M^{(i)} \end{array}\right) ; \tag{2.1}$$

then B may be calculated as follows:

<u>Algorithm 2.1</u> (Trench-Zohar algorithm (TZA))

```
{Initialization}

1.    $\lambda_1 \leftarrow 1 - u_1 v_1$

2.    $\underline{c}_1 \leftarrow (-u_1)$

3.    $\underline{d}_1 \leftarrow (-v_1)$

{Phase I, main loop:  in each pass, calculate $\underline{c}_{i+1}$, $\underline{d}_{i+1}$ and $\lambda_{i+1}$}

4.    For $i \leftarrow 1$ to $n-2$ do

      4.1   $\eta_i \leftarrow -(u_{i+1} + \underline{c}_i^T \underline{u}_i^R)$      {$\underline{u}_i, \underline{v}_i$: first $i$ elements of $\underline{u}, \underline{v}$}

      4.2   $\gamma_i \quad -(v_{i+1} + \underline{v}_i^T \underline{d}_i^R)$

      4.3   $\underline{c}_{i+1} \leftarrow \begin{pmatrix} \underline{c}_i + \frac{\eta_i}{\lambda_i} \underline{d}_i^R \\ \eta_i/\lambda_i \end{pmatrix}$

      4.4   $\underline{d}_{i+1}^R \leftarrow \begin{pmatrix} \eta_i/\lambda_i \\ \underline{d}_i^R + \frac{\gamma_i}{\lambda_i} \underline{c}_i \end{pmatrix}$

      4.5   $\lambda_{i+1} \leftarrow \lambda_i - \eta_i \gamma_i/\lambda_i$
```

```
{Phase II - calculate the rest of $B$ from its first row and column}

5.  $b_{11} \leftarrow 1/t_0\lambda_{n-1}$

6.  $\underline{b}_{1,2:n} \leftarrow \underline{c}/t_0\lambda_{n-1}$

7.  $\underline{b}_{2:n,1} \leftarrow \underline{d}/t_0\lambda_{n-1}$

8.  For $i \leftarrow 2$ to $n$ do {calculate all $b_{ij}$ above the secondary diagonal}

    8.1  for $j \leftarrow 2$ to $n-i+1$ do $b_{ij} \leftarrow b_{i-1,j-1} + \frac{1}{t_0\lambda_{n-1}} (\underline{d}\,\underline{c}^T - \underline{c}^R\,\underline{d}^{RT})_{i-1,j-1}$

9.  $b_{ij} \leftarrow b_{n+1-j,n+1-i}$ $\forall$ $i,j$ below the secondary diagonal.
```

Operation Counts

Phases I and II of the Trench-Zohar algorithm require $2n^2 + O(n)$ and $n^2 + O(n)$ operations respectively.

Jain's Modification

To solve the system $T\underline{x} = \underline{b}$, one would normally find T^{-1} by the above algorithm, then evaluate $T^{-1}\underline{b}$. These steps require $4n^2 + O(n)$ operations in all. Jain's modification [48] enables $T\underline{x} = \underline{b}$ to be solved in $2n^2 + 8n\ log\ n + O(n)$ operations. In Jain's method, Phase I of the Trench-Zohar algorithm is executed to find $\underline{c}$, $\underline{d}$ and λ_{n-1}. Then the following formula by Gohberg and Semencul [33]

$$B = \frac{1}{t_0\lambda_{n-1}} \left\{ \begin{pmatrix} 1 & & 0 \\ d_1 & \ddots & \\ \vdots & \ddots & \ddots \\ d_{n-1} \cdots d_1 & & 1 \end{pmatrix} \begin{pmatrix} 1 & c_1 \cdots & c_{n-1} \\ & \ddots & \vdots \\ & \ddots & c_1 \\ 0 & & 1 \end{pmatrix} - \begin{pmatrix} 0 & & 0 \\ d_n & \ddots & \\ \vdots & \ddots & \ddots \\ d_1 \cdots d_{n-1} & & 0 \end{pmatrix} \begin{pmatrix} 0 & c_{n-1} \cdots & c_1 \\ & \ddots & \vdots \\ & \ddots & c_{n-1} \\ & & 0 \end{pmatrix} \right\} \tag{2.2}$$

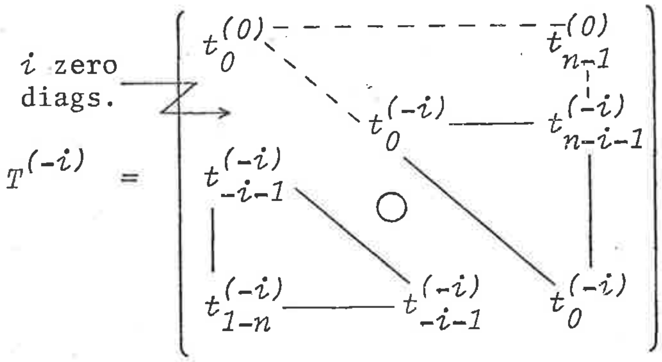
may be used to evaluate $\underline{x} = B\underline{b}$. The main work is in evaluating four matrix-vector products, where the matrices are triangular Toeplitz. It may be shown [3] that such a product may be embedded in a circular convolution, which may be evaluated in $O(n \log n)$ operations by FFT [20].

3. THE EXTENDED BAREISS SYMMETRIC ALGORITHM

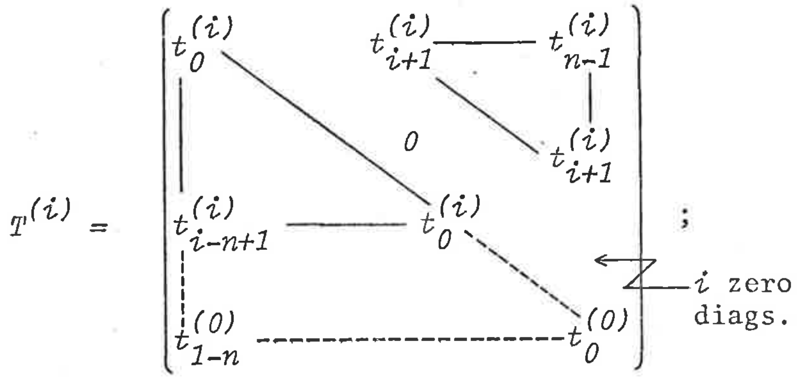
We first describe the Bareiss symmetric algorithm (BSA), which reduces a Toeplitz matrix to upper and lower triangles in $2n^2 + O(n)$ operations. We then extend BSA to calculate two upper and two lower triangles, which are closely related to U, U^{-1}, L and L^{-1}, where L and U are the triangular factors of T.

3.1 Bareiss Symmetric Algorithm (BSA)

BSA [6] is very similar to Bareiss's Nonsymmetric algorithm, described in the last Chapter. BSA generates a sequence of n-1 pairs of matrices, the i-th pair of which has the form



(3.1a)



; (3.1b)

the subscripts in the elements of (3.1a) and (3.1b) denote the distance above (positive) or below (negative) the main diagonal, and blocks which are bounded by continuous lines are Toeplitz.

The basic recursion in BSA may be written in the following matrix form:

Algorithm 3.1 (Bareiss Symmetric algorithm (BSA))

```
1.  $T^{(0)} \leftarrow T$                                                    (3.2)

2.  For $i \leftarrow 1$ to $n-1$ do:

    2.1  $T^{(-i)} \leftarrow T^{(1-i)} - m_{-i} Z_{-i} \quad T^{(i-1)}$        (3.3a)

         where $m_{-i} = t_{-i}^{(1-i)}/t_0^{(i-1)}$ and $(Z_{-i})_{kl} = \begin{cases} 1 & (l=k-i) \\ 0 & \text{elsewhere} \end{cases}$   (3.3b,c)

    2.2  $T^{(i)} \leftarrow T^{(i-1)} - m_i Z_i \quad T^{(1-i)}$               (3.4a)

         where $m_i = t_i^{(i-1)}/t_0^{(1-i)}$ and $(Z_i)_{kl} = \begin{cases} 1 & (l=k+i) \\ 0 & \text{elsewhere} \end{cases}$   (3.4b,c)
```

In (2.6a), the effect of Z_{-i} is to downshift $T^{(i-1)}$ by i rows and replace the first i rows by zeros. Similarly, Z_i upshifts a matrix by i rows and replaces the last i rows by zeros. If $T^{(1-i)}$ and $T^{(i-1)}$ have the forms shown in (3.1a) and (3.1b) (with $i-1$ replacing i) it can be checked that $T^{(-i)}$ and $T^{(i)}$ as produced by (3.3) and (3.4) have

i null sub- and super-diagonals respectively. So the $(n-1)$ iterates in the BSA, $T^{(1-n)}$ and $T^{(n-1)}$, are upper and lower triangular respectively, and it will be seen later that they are simply related to L and U. BSA is called "symmetric" because the recursion for the negative-index iterate (step 2.1) is the same as that for the positive-index iterate (step 2.2), with all indices changed in sign. In Chapter 2, we described Bareiss's non-symmetric algorithm which does not have this property, but is slightly more efficient.

3.2 The Extended Bareiss Symmetric Algorithm

We wish to extend BSA to calculate L^{-1} and U^{-1} as well as L and U. We now introduce the multiplier matrices, defined by

$$M^{(-i)}T = T^{(-i)} \qquad (3.5)$$

$$M^{(i)}T = T^{(i)} \qquad (3.6)$$

T is assumed nonsingular, so $M^{(-i)}$ and $M^{(i)}$ are unique. Substituting (3.5) and (3.6) into (3.2)-(3.4), and postmultiplying by T^{-1} yields a recursion for the $M^{(\pm i)}$

$$M^{(0)} = I \qquad (3.7)$$

$$M^{(-i)} = M^{(1-i)} - m_{-i}\, Z_{-i}\, M^{(i-1)} \qquad (3.8)$$

$$M^{(i)} = M^{(i-1)} - m_i\, Z_i\, M^{(1-i)} \qquad (3.9)$$

Equations (3.7)-(3.9), together with algorithm 3.1, enable us to calculate both the $T^{(\pm i)}$ and the $M^{(\pm i)}$:

Algorithm 3.2 (Extended BSA (EBSA))

1. $T^{(0)} = T$ (3.10)

2. $M^{(0)} = I$ (3.11)

3. For $i \leftarrow 1$ to $n-1$ do:

3.1 $m_{-i} = t_{-i}^{(-i+1)} / t_{0}^{(i-1)}$ (3.12)

3.2 $T^{(-i)} = T^{(-i+1)} - m_{-i}\, Z_{-i}\, T^{(i-1)}$ (3.13)

3.3 $M^{(-i)} = M^{(-i+1)} - m_{-i}\, Z_{-i}\, M^{(i-1)}$ (3.14)

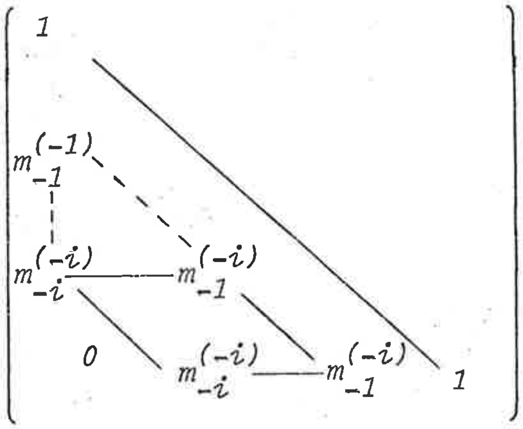
3.4 $m_{i} = t_{-i}^{(i-1)} / t_{0}^{(-i+1)}$ (3.15)

3.5 $T^{(i)} = T^{(i-1)} - m_{i}\, Z_{i}\, T^{(-i+1)}$ (3.16)

3.6 $M^{(i)} = M^{(i+1)} - m_{i}\, Z_{i}\, M^{(-i+1)}$ (3.17)

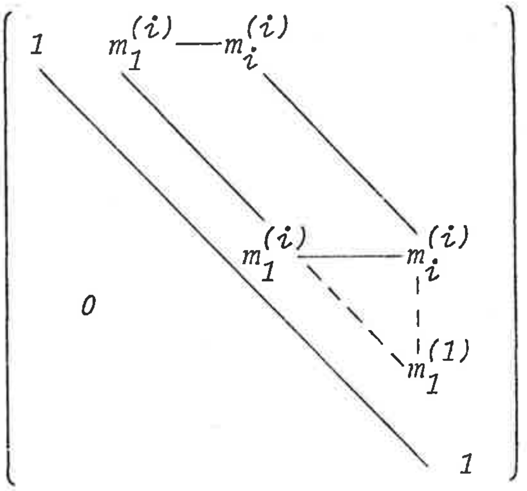
It may be shown by an induction, that $M^{(-i)}$ and $M^{(i)}$ have the form (Toeplitz blocks are bounded by continuous lines)

$M^{(-i)} =$



(3.18)

$M^{(i)} =$



, (3.19)

whence it may be shown that (3.14) and (3.17) require only i independent operations each; recall that (3.13) and (3.16) required only $2(n-i)$ independent operations each, so EBSA requires $\sum_{i=1}^{n-1} i+2(n-i) = 3n^2+O(n)$ operations to execute.

In the theorem below, we show how the triangular factors of T and their inverses are related to the output of EBSA.

Theorem 3.1 Let $T=LU$, where T is a nonsingular Toeplitz matrix, L is unit lower-triangular, and U is upper-triangular. Let $T^{(+)}:=T^{(n-1)}$, $T^{(-)}:=T^{(1-n)}$, $M^{(+)}:=M^{(n-1)}$ and $M^{(-)}:=M^{(1-n)}$ be the reduced and multiplier matrices produced by applying EBSA to T.

Then:

$$U = T^{(-)} \tag{3.20}$$

$$L = [\text{diag}\{1/t_{ii}^{(+)}\}T^{(+)}]^{T2} \tag{3.21}$$

$$L^{-1} = M^{(-)} \tag{3.22}$$

$$U^{-1} = [\text{diag}\{1/t_{ii}^{(+)}\}M^{(+)}]^{T2} \tag{3.23}$$

Proof: By definition, $M^{(-)}T = T^{(-)}$

$$T = M^{(-)-1}T^{(-)} \tag{3.24}$$

Now from (3.18), setting $i=n-1$, $M^{(-)}$ is unit lower-triangular, so $M^{(-)-1}$ is also. $T^{(-)}$ is upper-triangular. So (3.24) is an LU factorization of T, therefore

$$L = M^{(-)-1}$$

$$U = T^{(-)} ,$$

because the LU factorization is unique for L unit triangular. This proves (3.20) and (3.22).

By definition $M^{(+)}T = T^{(+)}$

i.e. $$T = M^{(+)-1}T^{(+)} = (M^{(+)-1}\text{diag}\{t_{ii}^{(+)}\})(\text{diag}\{1/t_{ii}^{(+)}\}T^{(+)}) \tag{3.25}$$

Recall from Chapter 2 that for any matrices X and Y,

$$(XY)^{T2} = Y^{T2}\, X^{T2} \,.$$

Taking the secondary transpose of (3.25) and using the above identity,

$$T^{T2}=T=(\mathrm{diag}\{1/t_{ii}^{(+)}\}T^{(+)})^{T2}(M^{(+)-1}\mathrm{diag}\{t_{ii}^{(+)}\})^{T2} \qquad (3.26)$$

Now $T^{(+)}$ is lower-triangular, so $\mathrm{diag}\{1/t_{ii}^{(+)}\}T^{(+)}$ and hence $(\mathrm{diag}\{1/t_{ii}^{(+)}\} \times \times T^{(+)})^{T2}$ is also unit lower-triangular. Also from (3.19), setting $i=n-1$, $M^{(+)}$ is upper-triangular, hence so is the right-hand factor of (3.26). So (3.26) is again the (unique) LU factorization of T, therefore

$$L \;=\; (\mathrm{diag}\{1/t_{ii}^{(+)}\}T^{(+)})^{T2} \qquad (3.27)$$

$$U \;=\; (M^{(+)-1}\mathrm{diag}\{t_{ii}^{(+)}\})^{T2} \qquad (3.28)$$

(3.27) is the same as (3.21), and (3.23) follows by inverting (3.28).

QED.

4. THE ALTERNATIVE BAREISS SYMMETRIC ALGORITHM (ABSA) - TOEPLITZ INVERSION

We have shown that EBSA calculates L, U, L^{-1} and U^{-1} in $3n^2+O(n)$ operations. We now modify EBSA to calculate L^{-1} and U^{-1} in only $2n^2+O(n)$ operations. We then show how the resulting algorithm, the alternative Bareiss symmetric algorithm (ABSA) is related to the Trench-Zohar algorithm.

From equations (3.22) and (3.23), we require $M^{(-)}$, $M^{(+)}$ and $\mathrm{diag}\{t_{ii}^{(+)}\}$ to generate L^{-1} and U^{-1}. If we omit steps 3.2 and 3.5 from EBSA (algorithm 3.2), we will have an algorithm to calculate the desired quantities if we can calculate $t_{-i}^{(-i+1)}$, $t_0^{(i-1)}$, $t_i^{(i-1)}$ and $t_0^{(-i+1)}$ without needing to calculate the rest of $T^{(-i)}$ and $T^{(i)}$. To this end, we prove the following theorem.

Theorem 4.1 Let $T^{(\pm i)}$, $M^{(\pm i)}$ be as in Algorithm 3.2. Let $t_j^{(\pm i)}$, be as in equations (3.1a,b). Then the following relations hold:

$$t_{-i}^{(-i+1)} = \underline{m}_{i+1.}^{(-i+1)}\ \underline{t}_{.1} \tag{4.1}$$

$$t_0^{(i-1)} = t_0^{(i-2)} - m_{i-1}\ t_{-i+1}^{(-i+2)} \tag{4.2}$$

$$t_i^{(i-1)} = \underline{m}_{1.}^{(i-1)}\underline{t}_{.i+1} \tag{4.3}$$

$$t_0^{(-i+1)} = t_0^{(-i+2)} - m_{-i+1}\ t_{i-1}^{(i-2)} \tag{4.4}$$

$$= t_0^{(i-1)} \tag{4.5}$$

Proof: (4.1) and (4.3) follow from the definitions of $M^{(\pm i)}$ (eqs. (3.5) and (3.6)), and (4.2) follows from writing out the (i,i) element of (3.16) and using (3.1a,b). (4.4) follows from writing out the $(1,1)$ element of (3.13) and using (3.1a,b). Alternatively, (4.2) and (4.4) follow directly from Bareiss's equations (3.1b). (4.5) is shown by Bareiss.

QED.

If we omit steps 3.2 and 3.5 in EBSA and insert (4.1)-(4.3), (4.5) to calculate the elements of $T^{(\pm i)}$ needed for $m_{\pm i}$, we have the Alternative Bareiss Symmetric Algorithm, which calculates $M^{(-)}$, $M^{(+)}$ and $t_0^{(\pm i)}$, and, by (3.22) and (3.23), L^{-1} and U^{-1}:

Algorithm 4.1 (Alternative Bareiss Symmetric Algorithm (ABSA))

```
1.  $t_0^{(0)} \leftarrow t_0$
2.  $M^{(0)} \leftarrow I$
3.  For $i \leftarrow 1$ to $n-1$ do:
    3.1  $t_{-i}^{(1-i)} \leftarrow \underline{m}_{i+1.}^{(1-i)} \; \underline{t}_{\cdot 1}$          {eq.(4.1)}
    3.2  $m_{-i} \leftarrow t_{-i}^{(1-i)} / t_0^{(i-1)}$
    3.3  $M^{(-i)} \leftarrow M^{(1-i)} - m_{-i} \, Z_{-i} \, M^{(i-1)}$
    3.4  $t_i^{(i-1)} \leftarrow \underline{m}_{1.}^{(i-1)} \; \underline{t}_{\cdot i+1}$          {eq.(4.3)}
    3.5  $m_i \leftarrow t_i^{(i-1)} / t_0^{(1-i)}$
    3.6  $M^{(i)} \leftarrow M^{(i-1)} - m_i \, Z_i M^{(1-i)}$
    3.7  $t_0^{(-i)} \leftarrow t_0^{(1-i)} - m_{-i} t_i^{(i-1)}$;  $t_0^{(i)} \leftarrow t_0^{(-i)}$     {eqs.(4.4),(4.5)}
```

Algorithm 4.1 may be written in a more compact form using (3.18) and (3.19). We observe that the first i rows of $M^{(-i)}$ are the same as those of $M^{(1-i)}$, and the last i rows of $M^{(i)}$ are the same as those of $M^{(i-1)}$. Hence $\underline{m}_{i+1.}^{(-i)}$ and $\underline{m}_{1.}^{(i)}$ only need be calculated to specify $M^{(-i)}$ and $M^{(i)}$ respectively. So, writing out row $i+1$ and 1 of steps 3.3 and 3.6 respectively, and noting from (3.18) that $\underline{m}_{i+1,1:i+1}^{(1-i)} = (0, \underline{m}_{i,1:i}^{(1-i)})$, we obtain:

Algorithm 4.1a (ABSA)

```
1.  $t_0^{(0)} \leftarrow t_0$
2.  $m_{11}^{(0)} \leftarrow 1$
```

```
3.  For $i \leftarrow 1$ to $n-1$ do:
    3.1  $t_{-i}^{(1-i)} \leftarrow \underline{m}_{i,1:i}^{(1-i)} \, \underline{t}_{2:i+1,1}$
    3.2  $m_{-i} \leftarrow t_{-i}^{(1-i)}/t_{0}^{(i-1)}$
    3.3  $\underline{m}_{i+1,1:i+1}^{(-i)} \leftarrow (0, \underline{m}_{i,1:i}^{(1-i)}) - m_{-i}(\underline{m}_{1,1:i}^{(i-1)}, 0)$
    3.4  $t_{i}^{(i-1)} \leftarrow \underline{m}_{1,1:i}^{(i-1)} \quad \underline{t}_{1:i,i+1}$
    3.5  $m_{i} \leftarrow t_{i}^{(i-1)}/t_{0}^{(1-i)}$
    3.6  $\underline{m}_{1,1:i+1}^{(i)} \leftarrow (\underline{m}_{1,1:i}^{(i-1)}, 0) - m_{i}(0, \underline{m}_{i,1:i}^{(1-i)})$
    3.7  $t_{0}^{(-i)} \leftarrow t_{0}^{(1-i)} - m_{-i} \, t_{i}^{(i-1)}$
4.  $\underline{m}_{1:j+1,j+1}^{(-)} = \underline{m}_{1:j+1,j+1}^{(-j)}$, $\underline{m}_{n-j,n-j:n}^{(+)} = \underline{m}_{n-j,n-j:n}^{(j)}$, $j=0,\ldots,n-1$
```

The relation between ABSA and the Trench-Zohar Algorithm

The connection between ABSA (algorithm 4.1a and the Trench-Zohar algorithm (algorithm 2.1)) may be found by using the following results [48]:

$$(B_{i+1})_{1.} = (U_{i+1}^{-1})_{.i+1}^{RT} \tag{4.6}$$

$$(B_{i+1})_{.1} = (U_{i+1}^{-1})_{i+1,i+1} \, (L_{i+1}^{-1})_{i+1.}^{RT} \tag{4.7}$$

where $\qquad T_{i+1} = L_{i+1}U_{i+1}$

We now wish to get the quantities in the Trench-Zohar algorithm in terms of the quantities in ABSA. Using (3.22) and (3.23) (applied to T_{i+1}), (4.6) and (4.7) become

$$(B_{i+1})_{1.} = \underline{m}_{1,1:i+1}^{(i)}/t_{0}^{(i)} \tag{4.8}$$

$$(B_{i+1})_{.1}^{T} = \underline{m}_{i+1,1:i+1}^{(-i)}/t_{0}^{(-i)} = m_{i+1,1:i+1}^{(-i)}/t_{0}^{(i)} \tag{4.9}$$

The first component of (4.8) ⇒ $(B_{i+1})_{11} = \dfrac{1}{t_0^{(i)}}$, i.e., using (2.1),

$$t_0 \, \lambda_i \;=\; t_0^{(i)} \tag{4.10}$$

Again using (2.1), (4.8) and (4.9) can be written (components 2 to $i+1$):

$$\underline{c}_i^T \;=\; \underline{m}^{(i)}_{1,2:i+1} \tag{4.11}$$

$$\underline{d}_i^{RT} \;=\; \underline{m}^{(-i)}_{i+1,1:i} \tag{4.12}$$

Putting (4.11) in algorithm 2.1, step 4.1, we get

$$\eta_i = -(t_{i+1}/t_0 + \underline{m}^{(i)}_{1,2:i+1} \, \underline{u}^R_{1:i})$$

$$\therefore \quad = \underline{m}^{(i)}_{1,1:i+1} \, \underline{t}_{.i+1}/t_0 = \frac{-t^{(i)}_{i+1}}{t_0} \, , \tag{4.13}$$

using algorithm 4.1a, step 3.4.

Similarly, it can be shown that

$$\gamma_i \;=\; -t^{(-i)}_{-i-1}/t_0 \tag{4.14}$$

Now, by (4.10)-(4.14), algorithms 2.1 and 4.1a calculate the same quantities. Only the notation is different. Thus, if we make the transformations

$$\begin{array}{ll} t_0^{(i)} \rightarrow t_0\lambda_i & \qquad t^{(i)}_{i+1} \rightarrow -t_0\eta_i \\ \underline{m}_{1,2:i+1} \rightarrow \underline{c}_i & \qquad t^{(-i)}_{-i-1} \rightarrow -t_0\gamma_i \, , \\ \underline{m}^{(-i)}_{i+1,1:i} \rightarrow \underline{d}^R_i & \end{array}$$

then algorithm 4.1a should transform to phase 1 of algorithm 1. This is easily verified. Hence ABSA is equivalent to phase 1 of the Trench-Zohar algorithm.

Derivation of a variant of the Trench-Zohar Algorithm using the Bareiss nonsymmetric algorithm (BNA)

In a manner similar to that above, we can extend BNA [6] (which also reduces T to upper and lower triangles, but with $t_{ii}^{(+)}$ normalised to t_0) to calculate L,U,L^{-1} and U^{-1}; from the extended BNA, we can derive an alternative BNA (ABNA) which calculates L^{-1} and U^{-1} in $2n^2 + O(n)$ operations. By using the relations between $L_i^{-1},U_i^{-1},(B_i)_{1.}$ and $(B_i)_{.1}$, ABNA can again be used to calculate $\underline{b}_{.1}$ and $\underline{b}_{1.}$, hence it can be shown that ABNA is equivalent to a variant of the Trench-Zohar algorithm in which the $\underline{b}_{.i}$ are scaled by t_0, rather than $\lambda_i t_0$, in phase 1.

5. CONCLUSION

An extended form of the Bareiss symmetric algorithm (BSA) is presented to calculate L and U, the triangular factors of a Toeplitz matrix, together with L^{-1} and U^{-1}, in $3n^2 + O(n)$ operations. From this, an alternative BSA has been derived to calculate L^{-1} and U^{-1} only in $2n^2 + O(n)$ operations. Using relations connecting L^{-1},U^{-1} and the first rows and columns of the submatrices $\{T_i\}$ of T, it was shown that the alternative BSA and the Trench-Zohar algorithm were equivalent. An alternative Bareiss non-symmetric algorithm may be similarly derived, which is equivalent to a variant of the Trench-Zohar algorithm.

CHAPTER 4

ERROR ANALYSIS OF BAREISS'S ALGORITHM

1. INTRODUCTION

We now present rounding-error analyses of Bareiss's algorithm BNA. The main purpose of this chapter is to show that the rounding-error can increase without limit if a leading submatrix approaches singularity, showing that BNA is unstable. In the next chapter, we propose a pivoting scheme for BNA which gives satisfactory results when a leading submatrix is singular or close to singular. To the author's knowledge, there is no error analysis of BNA as yet available in the literature, though Cybenko [22] gives an error analysis of TZA (which, as we saw could be derived from BSA), and DeJong [24], shows that Rissanen's algorithm [78] for the triangularization of a Hankel matrix is unstable. Rissanen's algorithm, however is not equivalent to BNA; to see this, note that if T is converted to a Hankel matrix by premultiplying by E, the exchange matrix, Rissanen's algorithm find the factorization

$$ET = \tilde{L}\tilde{D}\tilde{L}^T \Rightarrow T = E\tilde{L}\tilde{D}\tilde{L}^T \tag{1.1}$$

whereas BNA finds the factorization

$$T = LDU \tag{1.2}$$

The factorizations (1.1) and (1.2) are different because (a) EL is a different shape from L and (b) $U \neq L$ in general.

In Section 2, we derive several bounds on intermediate quantities produced by BNA. These bounds are functions of the condition numbers of certain submatrices of T. The condition number of a nonsingular matrix is a measure of its closeness to singularity, and is defined by

$$cond\ A := \|A\|\|A^{-1}\|. \tag{1.3}$$

In Section 3, we give a forward error analysis of one stage of BNA, and show that the errors in iterate *(k)* may be *L* times the errors in the previous iterate, where *L* is proportional to cond T_{k+1}. In Section 4, we give a backward error analysis of BNA, showing that if second-order quantities are neglected the computed solution $\underline{x}_c$ exactly solves a perturbed system $(T + \delta T)\underline{x}_c = \underline{b} + \delta\underline{b}$, where δT and $\delta\underline{b}$ are computable in terms of quantities calculated during the execution of BNA.

2. BAREISS ALGORITHM - MISCELLANEOUS RESULTS

The results of this section will be used in Section 3, where a forward error analysis of one step of BNA is performed. We first consider what happens when one or more leading submatrices of *T* are singular.

<u>Lemma 2.1</u> Let *A* be any order *k+1* matrix (not necessarily Toeplitz). Suppose A_k and A_{k+1} are singular. Define $A_k^E := A_{1:k,2:k+1}$ and $A_k^S := A_{2:k+1,1:k}$. Then either A_k^E or A_k^S is singular.

<u>Proof:</u> Assume det $A_k^E \neq 0$. Then, since det $A_k = 0$, $\exists\alpha_i$ not all zero, $i=1,\ldots,k$ such that $\sum_{i=1}^{k} \alpha_i\, \underline{a}_{i,1:k} = \underline{0}^T$. Consider separately the cases $\alpha_1 = 0$ and $\alpha_1 \neq 0$. If $\alpha_1 = 0$, setting $\alpha_{k+1} = 0$ gives $\sum_{i=2}^{k+1} \alpha_i\, \underline{a}_{i,1:k'} = \underline{0}^T \Rightarrow$ det $A_k^S = 0$.

If $\alpha_1 \neq 0$, then $\underline{a}_{1,1:k} + \sum_{i=2}^{k} (\alpha_i/\alpha_1)\, \underline{a}_{i,1:k} = \underline{0}^T$, so, by a determinental identity,

$$\det A_{k+1} = \det \left[\begin{array}{c} \underset{\sim}{a}_{1,1:k+1} + \sum_{i=2}^{k} (\alpha_i/\alpha_1)\, \underset{\sim}{a}_{i,1:k+1} \\ \hdashline A_{2:k+1,1:k+1} \end{array} \right]$$

$$= \det \left[\begin{array}{cc} \underset{\sim}{0}^T & \beta \\ \hdashline A_k^S & \underset{\sim}{a}_{1:k+1,k+1} \end{array} \right] = 0, \qquad (2.1)$$

where $\beta := a_{1,k+1} + \sum_{i=2}^{k} (\alpha_i/\alpha_1)\, a_{i,k+1} \neq 0$ because $\det A_k^E \neq 0$. Hence by (2.1) $\det A_k^S = 0$. QED.

For the next Lemma, and for the discussion in the next chapter, the following definition will be useful:

Definition For any matrix A, the displaced leading submatrix of order i and displacement j, denoted by $A_{i;j}$, is defined by:

$$A_{i;j} := A_{1:i,j+1:j+i} \qquad j \geq 0 \qquad (2.2a)$$

$$A_{i;j} = A_{|j|+1:|j|+i,1:i} \qquad j \leq 0 \qquad (2.2b)$$

Eq. (2.2a) shows that for $j \geq 0$, $A_{i;j}$ is the matrix contained in an $i \times i$ box with its top border at the top of A and its left border displaced j places from the left border of A. Eq. (2.2b) shows that for $j \leq 0$, $A_{i;j}$ is the matrix contained in an $i \times i$ box with its left border at the left of A and its top border displaced $|j|$ places down from the top of A.

Lemma 2.2 Let T be a Toeplitz matrix, and let $\det T_k \neq 0$, $\det T_{k+1;i} = 0$, $i = 0,\dots,r$. Then $\det T_{k;i} \neq 0$, $i=1,\dots,r+1$.

Proof: By induction on i. For $i=1$, suppose $det\ T_{k;1} := det\ T^E_k=0$. Then by Lemma 2.1, $det\ T_{2:k+1,2:k+1} = 0 \Rightarrow det\ T_k = 0$ (by Toeplicity), which is a contradiction, so $det\ T_{k;1} \neq 0$.

Suppose $det\ T_{k;i} \neq 0$. Then, if we suppose that $det\ T_{k;i+1}=0$, we can get a contradiction by the same argument as above, so det $T_{k;i+1} \neq 0$. QED.

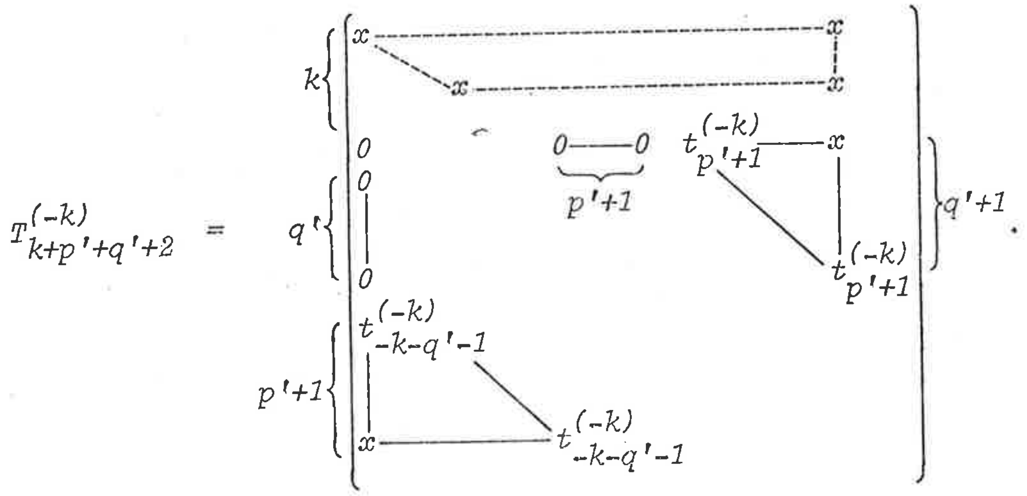
In Theorem 2.1 below, we show that if $s + 1$ leading submatrices of T are singular, then $s + 1$ extra zero-diagonals adjacent to the eliminated diagonals will appear. Bareiss notes this fact without proof. However, Theorem 2.1 states precisely when and in what form the zero diagonals will occur. We believe that the proof is not trivial:

Theorem 2.1 Suppose $det\ T_{k+i} = 0 (i=1,\ldots,s+1)$ and $det\ T_k \neq 0$. Then for some $p,q \geq 0$ $p + q \geq s$, $t^{(-k)}_j = 0$. $j=0,\ldots,p;-k-1,\ldots,q$.

Proof: By contradiction. Suppose

$$t^{(-k)}_j = 0,\ j=0,\ldots,p';-k-1,\ldots,-k-q', \tag{2.3}$$

where $p'+q' < s$ and $t^{(-k)}_{p'+1}$, $t^{(-k)}_{-k-q'-1} \neq 0$. Consider $T_{k+p'+q'+2}$, which by (2.2) and (2.3) has the form



(2.4)

From (2,4)

$$\det T^{(-k)}_{k+p'+q'+2} = (t^{(-k)}_{-k-q'-1})^{p'+1}(t^{(-k)}_{p'+1})^{q'+1} \det T^{(-k)}_{k;p'+1} \quad (2.5)$$

Now [6] $\det T^{(-h)}_{h;j} = \det T_{h;j}$, $h=0,\ldots,n-1$; $j=0,\ldots,n-h$ (2.6)

So by (2.6) and (2.4)

$$\det T_{k+1;j} = \det T^{(-k)}_{k+1;j} = 0, \; j=0,\ldots,p'+1.$$

∴ By Lemma 2.2,

$\det T^{(-k)}_{k;p'+1} = \det T_{k;p'+1} \neq 0 \Rightarrow$ by (2.5) $\det T^{(-k)}_{k+1'+q'+2} \neq 0$,

which contradicts

$\det T_{k+i} = 0$ $(i=1,\ldots,s+1)$, since $p'+q'+2 \leq s+1$.

QED.

Corollary: Suppose $\det T_{k+i} = 0 (i=1,\ldots,s+1)$, but $\det T_k \neq 0$. Then BNA breaks down after stage $(-k)$.

Proof: By Theorem 2.1, $t^{(-k)}_{k+1,k+1} = 0$, so eq. (2.2.3b) will break down.

QED.

Remark Bareiss [6] makes a statement equivalent to this corollary in his Corollary 2, so the above corollary is not new, but is included here for completeness.

We next consider what happens to $T^{(-k)}$ when the $\{T_{k+i}\}$ are close to singular in the sense that the distance between T_{k+i} and a singular matrix is small compared to $\|T_{k+i}\|$. A useful index of "closeness to singularity" is the condition number, defined for any non-singular matrix A in (1.3).

It can be shown [19] that an equivalent definition is

$$\mathit{cond}\; A := \max_{B \text{ singular}} \frac{\|A\|}{\|B-A\|} \quad (2.7)$$

Hence (2.7) says that the condition number is the inverse of the "relative closeness to singularity". Thus if the $\{T_{k+i}\}$ are close to singularity, they have large condition numbers. Theorem 2.1 stated that if the $\{T_{k+i}\}$ were singular, several extra diagonals of $T^{(-k)}$ will be zero. By a continuity argument, it would be expected that if the $\{T_{k+i}\}$ were close to singular (had large condition numbers) these extra diagonals would be "small" compared to the other elements of $T^{(-k)}$.

It would be useful to find a generalized version of Theorem 2.1 that shows that if $cond\ \{T_{k+i}\}_1^s$, were all greater than some M, then s of the diagonals $t_j^{(-k)}$ would be less than some bound δ. Such bounds for $s \leq 2$ are shown in Theorem 2.2, but for $s \geq 3$, the bounds rapidly became wildy pessimistic. The converse statement - that s of the diagonals $t_j^{(-k)}$ are less than δ only if $cond\ \{T_{k+i}\}_1^s$ are greater than M - is easy to show, and this is done in Theorem 2.3.

The following five Lemmas ((2.3)-(2.7)) are required before Theorem 2.2.

Lemma 2.3 Let A be a $(k+1) \times (k+1)$ matrix with $a_{k+1,k+1}$, $a^{-1}_{k+1,k+1} \neq 0$. Then

$$\left.\begin{array}{l} \|(A^{-1})_{1:k,k+1}\|/|(A^{-1})_{k+1,k+1}| \\ \\ \|(A^{-1})_{k+1,1:k}\|/|(A^{-1})_{k+1,k+1}| \end{array}\right\} \leq \beta\ cond\ A_k \qquad (2.8a,b)$$

where $\beta := \|A_{k+1}\|/\|A_k\|$.

Proof: Partition

$$A := \left(\begin{array}{c|c} A_k & \underline{b} \\ \hline \underline{c}^T & d \end{array}\right), \qquad A^{-1} = \left(\begin{array}{c|c} E & \underline{f} \\ \hline \underline{g}^T & h \end{array}\right) \qquad (2.9)$$

then $$\left(\begin{array}{c|c} A_k & \underline{b} \\ \hline \underline{c}^T & \underline{d} \end{array}\right) \left(\begin{array}{c|c} E & \underline{f} \\ \hline \underline{g}^T & h \end{array}\right) = I \qquad (2.10)$$

$$\Rightarrow A_k\underline{f} + h\underline{b}^T = 0 \;\Rightarrow\; \frac{\underline{f}}{h} = -A_k^{-1}\underline{b} \qquad (2.11)$$

so $\|\frac{\underline{f}}{h}\| = \|(A^{-1})_{1:k,k+1}\|/|(A^{-1})_{k+1,k+1}|$

$$\leq \|A_k^{-1}\|\|\underline{b}\| \leq \|A_k^{-1}\|\|A_{k+1}\| = \beta\|A_k^{-1}\|\|A_k\| = \beta \; cond\; A_k$$

which is (2.8a). (2.8b) can be shown similarly.

QED.

Lemma 2.4 Let A be a $(k+1)\times(k+1)$ matrix. Then

$$\|A_{k+1}^{-1}\| \leq |(A^{-1})_{k+1,k+1}|(1 + \beta\; cond\; T_k)^2 + \|A_k\|$$

where

$$\beta := \|A_{k+1}\|/\|A_k\|$$

Proof: Partition A, A^{-1} as in (2.9). Then, from (2.10),

$$A_kE + \underline{b}\underline{g}^T = I$$

so $E = A_k^{-1} - A_k^{-1}\underline{b}\underline{g}^T = A_k^{-1} + \frac{\underline{f}}{h}\underline{g}^T$ using (2.11)

so $$A_{k+1}^{-1} = \left(\begin{array}{c|c} A^{-1} & \underline{0} \\ \hline \underline{0}^T & 0 \end{array}\right) + h\begin{pmatrix} \frac{\underline{f}}{h} \\ 1 \end{pmatrix}[\underline{g}^T \; 1]$$

so $\|A_{k+1}^{-1}\| \leq \|A_k\| + |A_{k+1,k+1}^{-1}|(1 + \beta\; cond\; T_k)^2$ using Lemma 2.3.

QED.

<u>Lemma 2.5</u> Define $T^E_{k+1;j} := (T_{1:k+1,1:k}, \underline{t}_{1:k+1,j})$

and $T^s_{k+1;i} := \begin{pmatrix} T_{2:k+1,1:k+1} \\ \underline{t}_{i,1:k+1} \end{pmatrix}$,

and $\beta := max\ \{\|T^E_{k+1;j}\|, \|T^S_{k+1;j}\|\}/\|T_k\|$

Then, after step $(-k)$ of the Bareiss algorithm

$$\frac{|t_j^{(-k)}|}{\|T_{k+1+j}\|} \leq \frac{(1 + \beta\ cond\ T_k)^2}{cond\ T^E_{k+1;k+1+j} - \beta\ cond\ T_k} \qquad j \geq 0 \qquad (2.12)$$

$$\frac{|t_{-i}^{(-k)}|}{\|T_{i+1}\|} \leq \frac{(1 + \beta\ cond\ T_k)^2}{cond\ T^s_{k+1;i+1} - \beta\ cond\ T_k} \qquad i \geq k+1 \qquad (2.13)$$

<u>Proof:</u> It is easy to show that for $j \geq 0$

$$t_j^{(-k)} = 1/(T^E_{k+1;k+1+j})^{-1}_{k+1,k+1} \qquad (2.14)$$

From Lemma 2.4 and eq. (2.14),

$$\|(T^E_{k+1;k+1+j})^{-1}\| \leq \|T_k\| + (1 + \beta\ cond\ T_k)^2/|t_j^{(-k)}|$$

$$\therefore \qquad |t_j^{(-k)}| \leq (1 + \beta\ cond\ T_k)^2/\{\|(T^E_{k+1;j+k+1})^{-1}\| - \|T_k\|\}$$

from which (2.12) follows. The proof of (2.13) is similar.

QED.

<u>Lemma 2.6</u> Let F be any $n \times n$ matrix, let $G=F^{-1}$ and let g_{st} be the element of G with largest magnitude. Then

$$\underline{f}_{t\cdot} = \sum_{\substack{l=1 \\ l\neq t}}^{n} \alpha_l \underline{f}_{l\cdot} + \underline{\varepsilon}, \text{ where } \alpha_l := \frac{g_{sl}}{g_{st}} \Rightarrow |\alpha_l| \leq 1$$

$$\text{and} \quad \underline{\varepsilon}^T := \underline{e}_s^T/g_{st} \qquad (2.15a\text{-}c)$$

Proof: Expand $\underline{g}_{s\cdot}\,\underline{f}_{\cdot j} = \delta_{sj}$, divide through by g_{st}, and take f_{tj} to the left-hand-side. This shows (2.15) for the jth component. QED.

Lemma 2.7 Let $M:=\min(\mathrm{cond}\ T_k,\ \mathrm{cond}\ T_{k+1})$,

$$\beta := \max(\|T_{k+1}\|/\|T_k\|, \|T_{k+1}\|/\|T^s_{k;k+1}\|, \|T_{k+1}\|/\|T^E_{k;k+1}\|)$$

where

$$T^s_{k;k+1} := T_{2:k+1,1:k} \text{ and } T^E_{k;k+1} = (T_{1:k,1:k-1}, \underline{t}_{1:k,k+1}).$$

Then

$$\mathrm{cond}\ T^E_{k;k+1}\ \mathrm{cond}\ T^s_{k;k+1} \geq M/[\tfrac{3}{2}\,\beta^2(3k+1)].$$

Proof: Let $\underline{t}_i^T := \underline{t}_{i,1:k}$ and $\underline{u}_i^T := \underline{t}_{i,1:k+1}$, $i=1,\ldots,k+1$ (2.16a,b)

Let $(T^{-1}_{k+1})_{ji}$ be the element of T^{-1}_{k+1} with largest magnitude. Then, applying Lemma 2.6 to T_{k+1}, we get the following relation between $(k+1)$-column vectors

$$\underline{u}_i^T = \sum_{\substack{l=1\\ l\neq i}}^{k+1} \alpha_l \underline{u}_l^T + \underline{\xi}^T, \quad |\alpha_l|\leq 1, \quad \underline{\xi}^T := \underline{e}_j^T/(T^{-1}_{k+1})_{ji}. \qquad (2.17)$$

Removing the last column of (2.17), and using (2.16), we get the following relation between k-column vectors

$$\underline{t}_i^T = \sum_{\substack{l=1\\ l\neq 1}}^{k+1} \alpha_l \underline{t}_l^T + \underline{\varepsilon}^T, \text{ where } \underline{\varepsilon}^T := \underline{\xi}^T_{1:k} \qquad (2.18)$$

Let $(T^{-1}_k)_{qp}$ be the element of T^{-1}_k with largest magnitude. Applying Lemma 2.6 to T_k, we have

$$\underline{t}_p^T = \sum_{\substack{l=1\\ l\neq p}}^{k} \beta_l\, \underline{t}_l^T + \underline{\delta}^T, \quad |\beta_l|\leq 1, \quad \underline{\delta}^T = \underline{e}_q^T/(T^{-1}_k)_{qp} \qquad {}^*(2.19)$$

If we replace the term $\underline{t}_p^T$ in (2.18) by the RHS of (2.19), (2.18) can be written

$$\underline{t}_i^T = \sum_{\substack{l=1\\ l\neq p}}^{k} \alpha'_l\, \underline{t}_l^T + \alpha_{k+1}\underline{t}_{k+1}^T + \underline{\varepsilon}^T + \alpha_j\underline{\delta}^T, \quad \alpha'_l := \alpha_i+\alpha_p\beta_p \Rightarrow |\alpha'_l|\leq 2. \qquad (2.20)$$

*β_l is not related to the β in the Lemma statement.

Taking $\underline{t}_1^T$ to the left of (2,20), and substituting the resulting expression for $\underline{t}_1^T$ into (2,19), we get

$$\underline{t}_p^T = \sum_{\substack{l=2\\ l\neq p}}^{k} \beta_l \underline{t}_l^T + \underline{\delta} + \frac{\beta_1}{\alpha_1'}\{\underline{t}_i^T - \sum_{\substack{l=2\\ l\neq p}}^{k} \alpha_l' \underline{t}_l^T - \alpha_{k+1}\underline{t}_{k+1}^T - \underline{\varepsilon}^T - \alpha_j \underline{\delta}^T\} \tag{2.21}$$

We consider separately the cases $\frac{\beta_1}{\alpha_1'} \leq 1$ and $\frac{\beta_1}{\alpha_1} > 1$.

<u>Case 1. $\frac{\beta_1}{\alpha_1'} \leq 1$</u>

(2.21) shows that $T^s_{k;k+1}$ can be made singular by adding $\underline{\delta} + \frac{\beta_1}{\alpha_1'}(-\underline{\varepsilon} - \alpha_j \underline{\delta})$ to $\underline{t}_p$, so the distance between $T^s_{k;k+1}$ and a singular matrix is $\leq \|\underline{\varepsilon}\| + \|2\underline{\delta}\|$. Hence, applying (2.7)

$$\begin{aligned} cond\ T^s_{k;k+1} &\geq \|T^s_{k;k+1}\|/\{\|\underline{\varepsilon}\| + \|2\underline{\delta}\|\} \qquad (2.22)\\ &\geq \|T^s_{k;k+1}\|/\{|(T^{-1}_{k+1})_{ji}|^{-1} + 2|(T^{-1}_k)_{qp}|^{-1}\} \end{aligned}$$

by the definition of $\underline{\varepsilon}$ and $\underline{\delta}$.

Now $|(T^{-1}_{k+1})_{ji}| \geq \|T^{-1}_{k+1}\|/(k+1)$ and $|(T^{-1}_k)_{qp}| \geq \|T^{-1}_k\|/k$, so this becomes

$$\begin{aligned} cond\ T^s_{k;k+1} &\geq \|T^s_{k;k+1}\|/\{\frac{k+1}{\|T^{-1}_{k+1}\|} + \frac{2k}{\|T^{-1}_k\|}\}\\ &= \|T^s_{k;k+1}\|/\{\frac{(k+1)\|T_{k+1}\|}{cond\ T_{k+1}} + \frac{2k\|T_k\|}{cond\ T_k}\}\\ &\geq \frac{\|T^s_{k;k+1}\|}{\|T_{k+1}\|}\{\frac{k+1}{cond\ T_{k+1}} + \frac{2k}{cond\ T_k}\}^{-1}\\ &\geq \frac{1}{\beta}\{\frac{k+1}{M} + \frac{2k}{M}\}^{-1} = \frac{M}{\beta(3k+1)} \qquad (2.23) \end{aligned}$$

Case 2. $\frac{\beta_1}{\alpha'_1} > 1$

Rearranging (2.21) so that $\underline{t}_{k+1}$ is on the left, we see that $T^{s}_{k;k+1}$ can be made singular by adding $\frac{1}{\alpha_{k+1}} (-\underline{\varepsilon}-\alpha_{j}\underline{\delta}+\frac{\alpha'_1}{\beta_1}\underline{\delta})$ to $\underline{t}_{k+1}$.

Then, as for (2.22)

$$cond\ T^{s}_{k;k+1} \geq |\alpha_k| \| T^{s}_{k;k+1} \| / \{ \| \underline{\varepsilon} \| + 2 \| \delta \| \} \geq \frac{|\alpha_{k+1}| M}{\beta(3k+1)} , \qquad (2.24)$$

using (2.22) and (2.23).

Define $\underline{v}^T_i := (\underline{t}_{i,1:k-1}, t_{i,k+1})$ and $\dot{\underline{\varepsilon}}^T := (\underline{\xi}^T_{1:k-1}, \xi_{k+1})$.

Then, from (2.17)

$$\underline{v}^T_i = \sum_{\substack{l=1 \\ l\neq 1}}^{k} \alpha_l \, \underline{v}^T_l + \alpha_{k+1} \, \underline{v}^T_{k+1} + \dot{\underline{\varepsilon}}^T \qquad (2.25)$$

Eq. (2.25) shows that $T^{E}_{k;k+1}$ may be made singular by adding $\alpha_{k+1} \, \underline{v}^T_{k+1} + \dot{\underline{\varepsilon}}^T_{k+1}$ to $\underline{v}^T_i$, so

$$cond\ T^{E}_{k;k+1} \geq \frac{\| T^{E}_{k;k+1} \|}{|\alpha_{k+1}| \| \underline{u}_{k+1} \| + \| \underline{\varepsilon} \|} \qquad (2.26)$$

We now consider two sub-cases: $\| \alpha_{k+1} \, \underline{u}_{k+1} \| \leq 2 \| \underline{\xi} \|$ and $\| \alpha_{k+1} \, \underline{u}_{k+1} \| > 2 \| \underline{\xi} \|$.

Case 2(a). $\| \alpha_{k+1} \, \underline{u}_{k+1} \| \leq 2 \| \xi \|$

(2.26) becomes

$$cond\ T^{E}_{k;k+1} \geq \| T^{E}_{k;k+1} \| / (3 \| \underline{\xi} \|).$$

By a development similar to that in which (2.23) was derived from (2.22), this becomes

$$cond\ T^{E}_{k;k+1} \geq M/[3\beta(k+1)] \qquad (2.27)$$

Case 2(b). $\|\alpha_{k+1}\ \underset{\sim}{u}_{k+1}\| \geq 2\|\underset{\sim}{\xi}\|$

From (2.26)

$$\frac{\|T^E_{k;k+1}\|}{cond\ T^E_{k;k+1}} \leq \|\alpha_{k+1}\ \underset{\sim}{u}_{k+1}\| + \|\xi\| \leq \frac{3}{2}\ \|\alpha_{k+1}\ \underset{\sim}{u}_{k+1}\|$$

$$\leq \frac{3}{2}|\alpha_{k+1}|\,\|T_{k+1}\| \leq \frac{3}{2}|\alpha_{k+1}|\,\beta\|T^E_{k;k+1}\| \qquad (2.28)$$

(2.28) and (2.24) give

$$cond\ T^{\varepsilon}_{k;k+1}\ cond\ T^E_{k;k+1} \geq M/\{\tfrac{3}{2}\ \beta^2(3k+1)\} \qquad (2.29)$$

Eqs. (2.23), (2.24) and (2.29) cover all possible cases. It is clear that the bound in (2.29) is less than or equal to the bounds in (2.23) and (2.24), so (2.29) is true in all cases.

QED.

We can now bound some of the diagonals $\{t^{(-k)}_j\}$ if the condition number of T_{k+1}, T_{k+2} are bounded by M.

Theorem 2.2 Let $M := min(cond\ T_{k+1}, cond\ T_{k+2})$. Then

(i) $$\frac{t^{(-k)}_0}{\|T_{k+1}\|} \leq \frac{(1 + \beta\ cond\ T_k)^2}{cond\ T_{k+1} - \beta\ cond\ T_k} \qquad (2.30)$$

(ii) $$\frac{t^{(-k)}_1}{\|T_{k+2}\|} \leq \frac{(1 + \beta\ cond\ T_k)^2}{\dfrac{M}{\frac{3}{2}\ cond\ T^{\varepsilon}_{k+1;k+2}\ \beta^2(3k+1)} - \beta\ \ cond\ T_k} \qquad (2.31)$$

or $$\frac{t^{(-k)}_{-k-1}}{\|T_{k+2}\|} \quad \frac{(1 + \beta\ cond\ T_k)^2}{\dfrac{M}{\frac{3}{2}\ cond\ T^E_{k+1;k+2}\ \beta^2(3k+1)} - \beta\ \ cond\ T_k} \qquad (2.32)$$

where β is as in Lemma 2.5.

Proof: (2.30) follows directly from Lemma 2.5. (2.31) and (2.32) follow from Lemmas 2.5 and 2.7. QED.

Remarks Theorem 2.2 shows that if T_{k+1} is ill-conditioned, $|t_0^{(-k)}|/\|T_{k+1}\|$ is small. If T_{k+2} is ill-conditioned as well, then M will be large, and Lemma 2.7 shows that either or both of $T^S_{k+1;k+2}$ and $T^E_{k+1;k+2}$ will be ill-conditioned. If $T^S_{k+1;k+2}$ is ill-conditioned, but $T^E_{k+1;k+2}$ is well-conditioned, Theorem 2.2 shows that $t_1^{(-k)}/\|T_{k+2}\|$ is small. If $T^E_{k+1;k+2}$ is well-conditioned and $T^S_{k+1;k+2}$ is ill-conditioned, then $t_{-k-1}^{(-k)}/\|T_{k+2}\|$ is small. If $T^S_{k+1;k+2}$ and $T^E_{k+1;k+2}$ are both ill-conditioned, then (2.12), (2.13) and lemma 2.7 show that the product

$$\left(\frac{t_1^{(-k)}}{\|T_{k+2}\|}\cdot\frac{t_{-k-1}^{(-k)}}{\|T_{k+2}\|}\right) \text{ is small.}$$

We showed above that if $cond\ T_{k+1}$ and $cond\ T_{k+2}$ were bounded by M, we could put an upper bound on $t_0^{(-k)}$, and $t_1^{(-k)}$ or $t_{-k-1}^{(-k)}$, which is inversely proportional to M. We conjecture that this holds in general, i.e. if $cond\ T_{k+i} \geq M$, $i=1,\ldots,s$, then $\exists$ an upper bound for diagonals $t_{-k-1-q}^{(-k)},\ldots,t_{-k-1}^{(-k)};t_0^{(-k)},\ldots,t_p^{(-k)}$, where $p+q+1=s$. We have been unable to find realistic bounds, but our limited experience suggests that if $T_{k+1},\ldots,T_{k+s}$ are ill-conditioned, s of the diagonals of $T^{(-k)}$ adjoining and including $t_0^{(-k)}$, are small compared to the largest Toeplitz diagonal of $T^{(-k)}$.

It is easy, however, to prove a slight generalization of the converse of this conjecture. We can show that if s of the $t_j^{(-k)}$ are small, then $T_{k+1},\ldots,T_{k+s}$ are ill-conditioned; in fact, for a given

$k + 1 \le i \le k + s$, there are s ill-conditioned submatrices, including the leading submatrix. In theorem 2.3, $T_{k+1;j}$, etc. are as defined in eq.(2.2).

<u>Theorem 2.3</u> Let $|t_j^{(-k)}| \le \varepsilon$, $j=-k-1-q,\dots,-k-1;\ 0,\dots,p$ (2.33)

Then *cond* $T_{k+1;j} \ge \|T_{k+i;j}\|/[\varepsilon(p+q+1)]$,

$$i,j \in S := \{i,j: 1 \le i \le p+q+1, -q \le j \le p\}$$

<u>Proof</u> (outline) By writing out $T^{(-k)}$ explicitly using (2.2.2), it can be seem that at least one row $(T^{(-k)}_{k+i;j})_{r\cdot}$, $i,j \in S$, $k+1 \le r \le k+i$, consists solely of zeros and elements in the set $\{t_l^{(-k)}\}$ of (2.33).Hence, by (2.33) $\|(T^{(-k)}_{k+i;j})_{r\cdot}\| \le \varepsilon(p+q+1)$. From Bareiss [6], $(T^{(-k)}_{k+i;j})_{r\cdot} = (T_{k+i;j})_{r\cdot}$ + linear combination of $\{(T_{k+i;j})_{u\cdot}\}_{r-k}^{r-1}$, hence $T - \underline{e}_r(T^{(-k)}_{k+i;j})_{r\cdot}$ is singular, where $\underline{e}_r^T = (0,\dots 0,\overset{\text{position } r}{\downarrow 1},0,\dots 0)$. So, by (2.7)

$$cond\ T_{k+i;j} \ge \|T_{k+i;j}\|/\|(T^{(-k)}_{k+i,j})_{r\cdot}\| \ge \|T_{k+i;j}\|/[\varepsilon(p+q+1)].$$

QED.

3. BAREISS ALGORITHM - INCREASE IN ERROR BOUND

We now bound the increase in relative error from step $(-k)$ to step (k) of BNA, and show that this increase is bounded only by a quantity proportional to *cond* T_{k+1}. Computed quantities, will be denoted by bars, the error $\bar{a}-a$ by δa, and the relative error $|\delta a/a|$ by rel a $(a \ne 0)$.

We then define the <u>relative error of a computed matrix $\bar{A}$</u>, denoted rel $\bar{A}$, by

$$\text{rel } \bar{A} := \max_{i,j}|\delta a_{ij}|/\max_{i,j}|a_{ij}|, \text{ where } \delta a_{ij} := \bar{a}_{ij} - a_{ij}.$$

We also denote $T^{(-i)}_{i+1:n}$ and $T^{(i)}_{1:n-i}$, the Toeplitz parts of $T^{(-i)}$ and $T^{(i)}$ respectively, by $T^{(-i)}_*$ and $T^{(i)}_*$ respectively, for $i=0,\dots,n-1$. Throughout chapters 4-6, μ represents the machine precision. The result of this section is

Theorem 3.1 Let $\varepsilon := |t_0^{(-k)}|$, and let R be such that* rel $\bar{T}_*^{(-k)} \leq \mu R$ and rel $\bar{T}_*^{(k-1)} \leq \mu R$, and let β be as in Lemma 2.5. (3.1a-d)

Then* $$rel\ \bar{T}_*^{(k)} \leq \mu L R_3 \tag{3.2}$$

where $L := t^{(-k)}_{max}/\varepsilon$, with*

$$\frac{\|T\|}{\|T_{k+1}\|} \cdot \frac{cond\ T_{k+1} - \beta\ cond\ T_k}{cond\ T(1+\beta\ cond\ T_k)^2 \cdot (n-k)} \leq L \leq cond\ T_{k+1} \times$$

$$\times \max_j \|\underline{t}_{1:k+1,j}\| / \|T_{k+1}\| \tag{3.3a,b}$$

and R_3 is given by

$$R_3 := R_2 \left\{ \frac{1 + \frac{R}{R_2}\beta' cond\ T_k^E/L^2}{1 - \beta' cond\ T_k^E/L} \right\}, \text{ where } \beta' := \max_j \frac{\|\underline{t}_{1:k,j}\|}{\|T_k^E\|},$$

$$R_2 := R_1(1 + \frac{R}{R_1 L}) \text{ and } R_1 := R(1 + \frac{\beta'\ cond\ T_k^E}{L}) \tag{3.4a-d}$$

Proof: By writing out element rs of the Bareiss recursion (2.2.3), setting $j=s-r$, $t_j^{(\cdot)} = t_{rs}^{(\cdot)}$ and replacing exact quantities by computed quantities, we get

$$\bar{t}_j^{(k)} = \bar{t}_j^{(k-1)} - \bar{m}_k\ \bar{t}_{j-k}^{(-k)}, \text{ where } \bar{m}_k = \bar{t}_k^{(k-1)}/t_0^{(-k)} \tag{3.5a,b}$$

By (3.1) $$|\delta t_j^{(k-1)}| \leq \mu R t_{max}^{(k-1)}, \text{ where} \tag{3.6}$$

$$t_{max}^{(\cdot)} = \max_{u,v} (T_*^{(\cdot)})_{uv} .$$

*L, R, R_1, etc. are all scalars here.

Using the assumption $rel\ (\bar{x}/\bar{y}) \le rel\bar{x} + rel\bar{y}$, we get from (3.5b)

$$rel\ \bar{m}_k \le rel\ \bar{t}_k^{(k-1)} + rel\ \bar{\varepsilon} \le \mu R\left[\frac{t_{max}^{(k-1)}}{|t_k^{(k-1)}|} + \frac{t_{max}^{(-k)}}{\varepsilon}\right] \quad \text{(by 3.1b,c)}$$

$$= \mu R\left[\frac{t_{max}^{(k-1)}}{|t_k^{(k-1)}|} + L\right]. \qquad (3.7)$$

It can be easily shown that

$$t_j^{(k-1)} = t_k^{(k-1)} (T_k^E)^{-1}_{k\cdot}\ \underline{t}_{1:k,j+1}$$

$$\Rightarrow t_{max}^{(k-1)} \le \beta' |t_k^{(k-1)}|\ cond\ T_k^E \quad \text{where } \beta' := \max_j \|\underline{t}_{1:k,j}\|/\|T_k^E\|$$

(3.8a-c)

So (3.7) becomes $rel\ \bar{m}_k \le \mu R_1 L$,

$$\text{where } R_1 := R\left[1 + \frac{cond\ T_k^E}{L} \cdot \max_j \frac{\|\underline{t}_{1:k,j}\|}{\|T_k^E\|}\right]. \qquad (3.9a,b)$$

Now, using (3.1b), (3.9a) and the assumption $rel(\bar{x}\bar{y}) \le rel\bar{x} + rel\bar{y}$,

$$|\delta(m_k t_{j-k}^{(-k)})| \le |m_k t_{j-k}^{(-k)}|\{\mu R_1 L + \mu R t_{max}^{(-k)}/|t_{j-k}^{(-k)}|\}$$

$$\le |m_k t_{max}^{(-k)}|\{\mu R_1 L + \mu R\}$$

$$= |m_k t_{max}^{(-k)}|\mu R_2 L, \text{ where } R_2 := R_1(1 + \frac{R}{R_1 L})$$

(3.10a,b)

From (3.5a), (3.6), (3.10a), and the assumption

$|\delta(x+y)| \leq |\delta x| + |\delta y|$

$|\delta t_j^{(k)}| \leq \mu(Rt_{max}^{(k-1)} + |m_k t_{max}^{(-k)}| LR_2)$, which, with (3.8b),

$\leq \mu(Rt_k^{(k-1)} \beta' cond\ T_k^E + |m_k t_{max}^{(-k)}| LR_2)$, which, with

(3.5b), (3.3a)

$$\leq \mu L/|m_k\ t_{max}^{(-k)}|(R\beta' cond\ T_k^E/L^2 + R_2) \qquad (3.11)$$

Also, using (3.5a),

$t_{max}^{(k)} \geq |m_k t_{max}^{(-k)}| - |t_{max}^{(k-1)}|$, which, with (3.8b),

$\geq |m_k t_{max}^{(-k)}| - |t_k^{(k-1)} \beta' cond\ T_k^E|$, which, with (3.5b) and

(3.3a)

$$\geq |m_k t_{max}^{(-k)}|(1 - \beta' cond\ T_k^E/L) \ . \qquad (3.12)$$

(3.11) and (3.12) yield $rel\ \bar{T}_*^{(k)} = \mu LR_3$, where

$$R_3 := R_2(1 + \frac{R\beta'}{R_2} cond\ T_k^E/L^2)/(1-\beta' cond\ T_k^E/L) \qquad (3.13a,b).$$

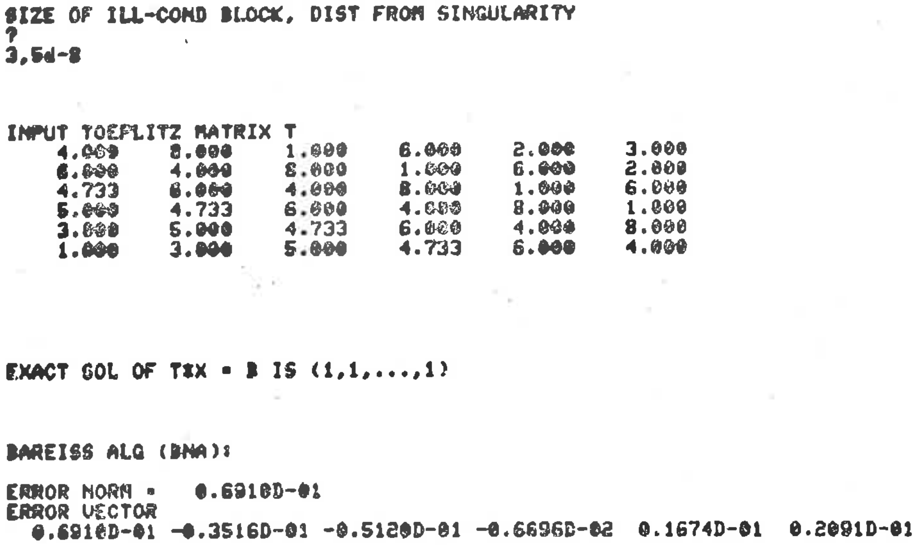
(3.13a), (3.8c), (3.9b), (3.10b) and (3.13b) together give the result (3.2).

The left-hand bound in (3.3a) may be found by applying Theorem 2.2 to bound ε above, and (2.7) to bound $t_{max}^{(-k)}$ below. The right-hand bound in (3.3a) may be found by using (2.7) to bound ε below. QED.

Theorem 3.1 shows that if T_{k+1} approaches singularity, the relative error bound for $T_*^{(k)}$, and hence the elements of subsequent iterates, may increase without limit.

Computer Results deomonstrating Loss of Accuracy

Table 3.1 shows a 6 x 6 Toeplitz matrix, A, selected such that $det\ A_3$ is small. ($det\ A_3=0$ if $a_{31}=\frac{71}{15}$). The right-hand side is such that the exact solution is $(1,1,\ldots,1)$. It is seen that the sizes of errors in the solution elements range from 0.007 to 0.069, and the ration $\|\delta\underline{x}\|_\infty/\|\underline{x}\|_\infty$ is 0.69, which is pathological considering the relative machine precision $\mu \doteq 10^{-16}$. This therefore bears out the above arguments.



```
SIZE OF ILL-COND BLOCK, DIST FROM SINGULARITY
?
3,5d-8


INPUT TOEPLITZ MATRIX T
     4.000     8.000     1.000     6.000     2.000     3.000
     6.000     4.000     8.000     1.000     6.000     2.000
     4.733     6.000     4.000     8.000     1.000     6.000
     5.000     4.733     6.000     4.000     8.000     1.000
     3.000     5.000     4.733     6.000     4.000     8.000
     1.000     3.000     5.000     4.733     6.000     4.000


EXACT SOL OF T*X = B IS (1,1,...,1)


BAREISS ALG (BNA):

ERROR NORM =    0.6910D-01
ERROR VECTOR
  0.6910D-01 -0.3516D-01 -0.5120D-01 -0.6696D-02  0.1674D-01  0.2091D-01
```

Table 3.1 : Application of BNA to matrix with an
ill-conditioned leading submatrix of order 3.

4. BAREISS ALGORITHM - BACKWARD ERROR ANALYSIS

Here, we show that the computed solution $\underset{\sim}{x}_c$ is the exact solution of the perturbed system;

$$(T + \delta T)\underline{x} = \underline{b} + \delta\underline{b} \ .$$

Neglecting second-order quantities, $\|\delta T\|$ and $\|\delta\underline{b}\|$ can be bounded in terms of quantities computed during BNA. Theorem 4.1 gives δT and $\delta\underline{b}$ in terms of certain error matrices F and J, and error vectors $\underline{g}$ and $\underline{k}$. Theorem 4.2 gives bounds for F, J, $\underline{g}$ and $\underline{k}$.

We first recall the Bareiss recursion

$$T^{(-i)} = T^{(1-i)} - m_{-i}Z_{-i}T^{(i-1)} \tag{4.1}$$

$$T^{(i)} = T^{(i-1)} - m_i Z_i T^{(-i)} \tag{4.2}$$

$$\underline{b}^{(-i)} = \underline{b}^{(1-i)} - m_{-i}Z_{-i}\underline{b}^{(i-1)} \tag{4.3}$$

$$\underline{b}^{(i)} = \underline{b}^{(i-1)} - m_i Z_i \underline{b}^{(-i)}, \tag{4.4}$$

and define $\hat{T}^{(-i)}$, $\hat{T}^{(i)}$, $\hat{\underline{b}}^{(-i)}$ and $\hat{\underline{b}}^{(i)}$ as the result of applying recursion (4.1)-(4.4) without rounding error, but with the $\{m_{\pm i}\}$ replaced by $\{\bar{m}_{\pm i}\}$. Also, assume T is nonsingular, and define the <u>multiplier matrices</u> by

$$M^{(\pm i)}T = T^{(\pm i)} \tag{4.5}$$

$$\hat{M}^{(\pm i)}T = \hat{T}^{(\pm i)} \tag{4.6}$$

Putting (4.5) in (4.1) and (4.2) yields

$$M^{(-i)} = M^{(1-i)} - m_{-i}Z_{-i}M^{(i-1)}$$

$$M^{(i)} = M^{(i-1)} - m_i Z_i M^{(-i)}$$

It can then be shown by induction, using (4.5) and (4.6) that $M^{(-i)}$ is ULT and $M^{(i)}$ is UT. Similarly $\hat{M}^{(-i)}$ and $\hat{M}^{(i)}$ can be shown to be ULT and UT respectively.

We then have

<u>Theorem 4.1</u> Assume w.l.o.g.* that $t_0 \neq 0$. Then the solutions $\bar{\underline{x}}_u$ and $\bar{\underline{x}}_l$ of the upper and lower-triangular systems

$$\bar{T}^{(1-n)}\bar{\underline{x}}_u = \bar{\underline{b}}^{(1-n)} \tag{4.7}$$

$$\bar{T}^{(n-1)}\bar{\underline{x}}_l = \bar{\underline{b}}^{(n-1)} \tag{4.8}$$

are the exact solutions respectively of the perturbed systems

$$[T + (L + \delta L)F]\bar{\underline{x}}_u = \underline{b} + (L + \delta L)\underline{g} \tag{4.9}$$

$$[T + t_0^{-1}(U + \delta U)^{T2}J]\bar{\underline{x}}_l = \underline{b} + t_0^{-1}(U + \delta U)^{T2}\underline{k} \tag{4.10}$$

where L is ULT and U is UT such that $T = LU$; $\delta L = (\hat{M}^{(1-n)})^{-1} - (M^{(1-n)})^{-1}$;

$$\delta U := (\hat{M}^{(n-1)})^{-1} - (M^{(n-1)})^{-1} \;;$$

$$F := \bar{T}^{(1-n)} - \hat{T}^{(1-n)} \tag{4.11}$$

$$\underline{g} := \bar{\underline{b}}^{(1-n)} - \hat{\underline{b}}^{(1-n)} \tag{4.12}$$

$$J := \bar{T}^{(n-1)} - \hat{T}^{(n-1)} \tag{4.13}$$

$$\underline{k} := \bar{\underline{b}}^{(n-1)} - \hat{\underline{b}}^{(n-1)} \tag{4.14}$$

<u>Proof:</u> It is clear from the definition of $\hat{\underline{b}}^{(\pm i)}$ that $\hat{M}^{(\pm i)}\underline{b} = \hat{\underline{b}}^{(\pm i)}$

(4.15)

Using the definition of F and $\underline{g}$ in (4.7),

$$(\hat{T}^{(1-n)} + F)\bar{\underline{x}}_u = \hat{\underline{b}}^{(1-n)} + \underline{g}$$

i.e. $(\hat{M}^{(1-n)}T + F)\bar{\underline{x}}_u = \hat{M}^{(1-n)}\underline{b} + \underline{g}$ (using (4.6) and (4.15)).

or** $(T + (\hat{M}^{(1-n)})^{-1}F)\bar{\underline{x}}_u = \underline{b} + (\hat{M}^{(1-n)})^{-1}\underline{g}$ (4.16)

*Otherwise BNA fails.

**$\hat{M}^{(1-n)}$ cannot be singular since it is ULT.

In Chapter 2 it was shown that (Theorem 2.3.1)

$$(M^{(1-n)})^{-1} = L,$$

so $$(\hat{M}^{(1-n)})^{-1} = L + \delta L,$$

where $$\delta L := (\hat{M}^{(1-n)})^{-1} - (M^{(1-n)})^{-1}. \tag{4.17}$$

Putting (4.17) in (4.16) yields (4.9). The proof of (4.10) is similar. QED.

For the relations (4.9), (4.10) to be useful, we must find bounds for $\|F\|$, $\|\underline{g}\|$, $\|J\|$ *and* $\|\underline{k}\|$. We first need the following Lemma:

<u>Lemma 4.1</u>

$$F = \sum_{p=1}^{n-1} (N_p^{(1-n)} E^{(p)} + N_{-p}^{(1-n)} E^{(-p)}) \tag{4.18}$$

$$\underline{g} = \sum_{p=1}^{n-1} (N_p^{(1-n)} \underline{h}^{(p)} + N_{-p}^{(1-n)} \underline{h}^{(-p)}) \tag{4.19}$$

$$J = \sum_{p=1}^{n-1} (N_p^{(n-1)} E^{(p)} + N_{-p}^{(n-1)} E^{(-p)}) \tag{4.20}$$

$$\underline{k} = \sum_{p=1}^{n-1} (N_p^{(n-1)} \underline{h}^{(p)} + N_{-p}^{(n-1)} \underline{h}^{(-p)}) , \tag{4.21}$$

where $E^{(\pm p)}$ and $\underline{h}^{(\pm p)}$, the <u>local</u> errors committed in calculating $\bar{T}^{(\pm p)}$ and $\underline{\bar{b}}^{(\pm p)}$, are given by

$$\bar{T}^{(-p)} = \bar{T}^{(1-p)} - \bar{m}_{-p} Z_{-p} \bar{T}^{(p-1)} + E^{(-p)} \tag{4.22}$$

$$\bar{T}^{(p)} = \bar{T}^{(p-1)} - \bar{m}_p Z_p \bar{T}^{(-p)} + E^{(p)} \tag{4.23}$$

$$\underline{\bar{b}}^{(-p)} = \underline{\bar{b}}^{(1-p)} - \bar{m}_{-p} Z_{-p} \underline{\bar{b}}^{(p-1)} + \underline{h}^{(-p)} \tag{4.24}$$

$$\underline{\bar{b}}^{(p)} = \underline{\bar{b}}^{(p-1)} - \bar{m}_p Z_p \underline{\bar{b}}^{(-p)} + \underline{h}^{(p)} \tag{4.25}$$,

and the $N_{\pm p}^{\pm(n-1)}$ are given by the following recursions:

$$N_p^{(-p)} = 0 \; ; \; N_p^{(p)} = I \tag{4.26}$$

$$\left.\begin{aligned} N_p^{(-i)} &= N_p^{(1-i)} - \bar{m}_{-i}\, Z_{-i} N_p^{(i-1)} \\ N_p^{(i)} &= N_p^{(i-1)} - \bar{m}_i Z_i N_p^{(-i)} \end{aligned}\right\} \; i=p+1,\ldots,n-1 \tag{4.27a,b}$$

$$N_{-p}^{(p-1)} = 0; \quad N_{-p}^{(-p)} = I \tag{4.28}$$

$$\left.\begin{aligned} N_{-p}^{(i)} &= N_{-p}^{(i-1)} - \bar{m}_i Z_i N_{-p}^{(1-i)} \\ N_{-p}^{(-1-i)} &= N_{-p}^{(-i)} - \bar{m}_{-1-i} Z_{-1-i} N_{-p}^{(i)} \end{aligned}\right\} \; i=p,\ldots,n-2 \tag{4.29a,b}$$

$$N_{-p}^{(n-1)} = N_{-p}^{(n-2)} - \bar{m}_{-i} Z_{-i} N_{-p}^{(1-n)} \tag{4.30}$$

(note that (4.30) is just (4.29) with $i=n-1$).

<u>Proof</u>: We prove by induction on k that

$$\left.\begin{aligned} \bar{T}^{(-k)} - \hat{T}^{(-k)} &= \sum_{p=1}^{k} (N_p^{(-k)} E^{(p)} + N_{-p}^{(-k)} E^{(-p)} \\ \bar{T}^{(k)} - \hat{T}^{(k)} &= \sum_{p=1}^{k} (N_p^{(k)} E^{(p)} + N_{-p}^{(k)} E^{(p)} \end{aligned}\right\} \; k=1,\ldots,n-1, \tag{4.31,4.32}$$

and then, setting $k=n-1$ will yield (4.18) and (4.20).

We first show (4.31) for $k=1$. From (4.22)

$$\bar{T}^{(-1)} = T^{(0)} - \bar{m}_{-1} Z_{-1} T^{(0)} + E^{(-1)} \;, \tag{4.33}$$

and from the definition,

$$\hat{T}^{(-1)} = T^{(0)} - \bar{m}_{-1} Z_{-1} T^{(0)} \tag{4.34}$$

Equations (4.33) and (4.34) $\Rightarrow T^{(-1)} - \hat{T}^{(1)} = E^{(-1)}$. (4.35)

The RHS of (4.31), with $k=1$ is, using (4.26a) and (4.28b)

$$N_1^{(-1)}E^{(1)} + N_{-1}^{(-1)}E^{(-1)} = E^{(-1)} \qquad (4.36)$$

The result follows from (4.35) and (4.36).

Now, consider (4.32) for $k=1$. The LHS of (4.32), using (4.26b), (4.28a) and (4.29a) to evaluate the N's, is

$$N_1^{(1)}E^{(1)} + N_{-1}^{(1)}E^{(-1)} = E^{(1)} - \bar{m}_1 Z_1 E^{(-1)}. \qquad (4.37)$$

From (4.23),

$$\begin{aligned}\bar{T}^{(1)} &= T^{(0)} - \bar{m}_1 Z_1 \bar{T}^{(-1)} + E^{(1)}\\ &= T^{(0)} - \bar{m}_1 Z_1 \hat{T}^{(-1)} - \bar{m}_1 Z_1 E^{(-1)} + E^{(1)} \qquad (4.38)\end{aligned}$$

using (4.35).

By definition $\hat{T}^{(1)} = T^{(0)} - \bar{m}_1 Z_1 \hat{T}^{(-1)}$ (4.39)

Eqs. (4.37) and (4.39) prove (4.32) for $k=1$.

Assume now that (4.31) and (4.32) are true also for

$l = 2,\dots,k$.

From (4.22);

$$\begin{aligned}\bar{T}^{(-k-1)} &= \bar{T}^{(-k)} - \bar{m}_{-k-1} Z_{-k-1} \bar{T}^{(k)} + E^{(-k-1)}\\ &= \hat{T}^{(-k)} + \sum_{p=1}^{k} (N_p^{(-k)}E^{(p)} + N_{-p}^{(-k)}E^{(-p)}) - \bar{m}_{-k-1} Z_{-k-1} \hat{T}^{(k)} +\\ &+ \sum_{p=1}^{k} (-\bar{m}_{-k-1} Z_{-k-1} N_p^{(k)} E^{(p)} - \bar{m}_{-k-1} Z_{-k-1} N_{-p}^{(k)} E^{(-p)}) + E^{(-k-1)}\end{aligned}$$

using (4.31) and (4.32).

So

$$\begin{aligned}\bar{T}^{(-k-1)} &= \hat{T}^{(-k)} - \bar{m}_{-k-1} Z_{-k-1} \hat{T}^{(k)} +\\ &+ \sum_{p=1}^{k} \{N_p^{(-k)} - \bar{m}_{-k-1} Z_{-k-1} N_p^{(k)})E^{(p)} +\\ &+ (N_{-p}^{(-k)} - \bar{m}_{-k-1} Z_{-k-1} N_{-p}^{(k)})E^{(-p)})\} + E^{(-k-1)}\end{aligned}$$

Using (4.27a) and (4.29b), this becomes

$$\bar{T}^{(-k-1)} = \hat{T}^{(-k)} - \bar{m}_{-k-1} Z_{-k-1} \hat{T}^{(k)} +$$

$$+ \sum_{p=1}^{k} (N_p^{(-k-1)} E^{(p)} + N_{-p}^{(-k-1)} E^{(-p)}) + E^{(-k-1)},$$ which,

with (4.26a,b)

$$= \hat{T}^{(-k)} - \bar{m}_{-k-1} Z_{-k-1} \hat{T}^{(k)} +$$

$$+ \sum_{p=1}^{k+1} (N_p^{(-k-1)} E^{(p)} + N_{-p}^{(-k-1)} E^{(-p)}) \qquad (4.40)$$

But, by definition,

$$\hat{T}^{(-k-1)} = \hat{T}^{(-k)} - \bar{m}_{-k-1} Z_{-k-1} \hat{T}^{(k)} \qquad (4.41)$$

and, subtracting (4.41) from (4.40) yields (4.31) with k replaced by $k+1$. The proof for (4.32) is similar. This completes the proof of (4.18) and (4.20). The proof of (4.19) and (4.21) is analogous, with references to $E^{(\pm p)}$ being replaced everywhere by references to $\underline{h}^{(\pm p)}$. QED.

Upper Bounds on $F, J, \underline{g}$ and $\underline{k}$

We now use Lemma 4.1 to get bounds on the elements of $F, J, \underline{g}$ and $\underline{k}$.

Theorem 4.2 Assume that the result of any of the floating-point operations +, −, × or ÷ satisfies

$$fl(x \; op \; y) = (1 + \varepsilon)(x \; op \; y), \; |\varepsilon| \le \mu \qquad (4.42)$$

where μ is the machine precision. Let $\tau := \max_{i,j} |t_{ij}|$ and $\eta := \max_{i} |b_i|$.

Then $\forall_{i,j}$. $|f_{ij}| \le 4(n-1)\mu\alpha\tau$, $|j_{ij}| \le 4(n-1)\mu\alpha\tau$, $|g_i| \le 4(n-1)\mu\alpha\eta$ $|k_i| \le 4(n-1)\mu\alpha\eta$

where $$\alpha := \prod_{1-n}^{n-1} (1 + |m_l|). \qquad (4.43a\text{-}e)$$

<u>Proof:</u> We first evaluate a bound for $|f_{ij}|$. Consider the term $N_p^{(1-n)}E^{(p)}$ in (2.31). From (4.23)

$$e_{ij}^{(p)} = \bar{t}_{ij}^{(p)} - (\bar{t}_{ij}^{(p-1)} - \bar{m}_p \bar{t}_{i+p,j}^{(-p)}). \qquad (4.44)$$

Using (4.42), we can write the floating-point version of the Bareiss recursion (4.2) as

$$\bar{t}_{ij}^{(p)} = (1+\varepsilon)\{\bar{t}_{ij}^{(p-1)} - (1+\varepsilon')\bar{m}_p \bar{t}_{i+p,j}^{(-p)}\}, \quad |\varepsilon|, |\varepsilon'| \le \mu \qquad (4.45a,b)$$

Putting (4.45a) in (4.44) and neglecting second order quantities,

$$e_{ij}^{(p)} = \varepsilon \bar{t}_{ij}^{(p-1)} - (\varepsilon+\varepsilon')\bar{m}_p \bar{t}_{i+p,j}^{(-p)} \qquad (4.46)$$

From BNA, eqs. (4.1) and (4.2), it may be shown by induction on p that, neglecting second-order quantities,

$$|\bar{t}_{ij}^{(p-1)}| \le \tau \prod_{l=1-p}^{p-1} (1+|m_l|) \qquad (4.47)$$

$$|\bar{t}_{ij}^{(-p)}| \le \tau \prod_{l=-p}^{p-1} (1+|m_l|). \qquad (4.48)$$

Putting (4.47) and (4.48) in (4.46), and using (4.45b) we get

$$|e_{ij}^{(p)}| \le 2\mu\tau \prod_{l=-p}^{p} (1+|m_l|). \qquad (4.49)$$

It may be similarly shown by induction on p that

$$(N_p^{(1-n)}E^{(p)})_{ij} \le \max_{ij} |e_{ij}^{(p)}| \prod_{l=-p-1}^{1-n} (1+|m_l|) \prod_{l=p+1}^{n-1} (1+|m_l|) \qquad (4.50)$$

where for convenience we define $m_l=0$ if $|l| > n-1$.

(4.49) and (4.50) together give

$$(N_p^{(1-n)}E^{(p)})_{ij} \le 2\mu\tau \prod_{1-n}^{n-1} (1+|m_l|) \qquad (4.51)$$

We can similarly bound

$$(N_{-p}^{(1-n)}E^{(-p)})_{ij} \le 2\mu\tau \prod_{1-n}^{n-1} (1+|m_l|) \tag{4.52}$$

Putting (4.51) and (4.52) in (4.18) yields the result for $|f_{ij}|$. The results for $|j_{ij}|$, $|g_i|$ and $|k_i|$ are similar. QED.

Remarks

If we ignore the second-order quantities $\delta L.F$ and $\delta L.\underline{g}$ in (4.9) we see that the ∞-norm of the equivalent perturbation in T is bounded by $4n(n-1)\mu\tau\|L\|_\infty\alpha$. This bound is pessimistic because of the approximations used in Theorem 4.2. However, the factor α can be compared with Wilkinson's "growth factor" g [89] which, for partial pivoting, can be bounded only by 2^{n-1}. In fact, if the multipliers $|m_l| \le 1$, then $\alpha \le 2^{(2n-2)}$, the square of the bound on g.

The bounds (2.52) also show that BNA is unstable, because as, say, T_{k+1} approaches singularity, $|m_{-k-1}|$ and hence, $\|F\|$, $\|J\|$, $\|\underline{g}\|$ and $\|\underline{k}\|$ increase without bound.

5. CONCLUSION

After several preliminary results were derived regarding the size of elements of Bareiss iterates as a function of the condition number of certain submatrices of T, forward and backward error analyses were performed on BNA. Both these analyses, though yielding only pessimistic error bounds, show that BNA is unstable, in the sense that the error increases without bound as the condition number of any leading submatrix approaches infinity.

CHAPTER 5

THE PIVOTED BAREISS ALGORITHM

1. INTRODUCTION

As we have seen in the previous Chapter, the appearance of a large multiplier is a warning of a possible loss in accuracy. A large multiplier results when a pivot element is small. In BNA, a small pivot can only occur in calculating a positive-index iterate, as the pivot for a negative-index iterate (NII) is always t_0 (see equation 2.2.2b). In Lemma 4.2.5 we saw that the pivot for calculating $T^{(k)}$, namely, $t_0^{(-k)}$ is small when T_{k+1}, is ill-conditioned. In either case we must modify BNA to select another pivot which is not small. The new algorithm is the pivoted BNA (PBNA). We will also briefly describe a pivoted BSA (PBSA).

As mentioned above, Bareiss [6] proposes a pivoting scheme to cater for zero leading minors, but it requires the triangularization of a non-Toeplitz submatrix. Our algorithm will avoid the need to do this. In lecture notes, Morf states that the Berlekamp-Massey algorithm [8] can be modified to handle zero leading minors, but not in a numerically-robust manner. Rissanen [80] describes a pivoting scheme to handle zero leading minors in Hankel matrices. To the author's knowledge, there are no previous pivoting schemes which treat non-zero leading principal minors in Toeplitz matrices for the purpose of improving numerical performance.

In section 2, we describe the various pivoting procedures which, in section 3, are combined in a number of strategies which attempt to improve the numerical performance of the Bareiss algorithm. Also in section 3, it is pointed out that the pivoting procedures are equivalent to the calculation of elements along certain paths in the Padé table.

In the Padé literature, algorithms for some of these paths, such as rows and diagonals are known. The pivoting viewpoint is new, and many of the known Padé algorithms follow simply from this approach.

We showed in Chapter 4, that when a leading submatrix is ill-conditioned, serious error growth may occur if BNA is used. In section 4, we analyze the simplest strategy, PBA-1, showing that when this is used, no such error growth can occur, provided the pivots actually used are not small. We conjecture that this result is also true for any pivoting strategy.

In section 5, we indicate how to introduce pivoting into the Bareiss Symmetric Algorithm (BSA).

2. PIVOTING PROCEDURES FOR THE BAREISS ALGORITHM

Here we explain, with the aid of "shape" diagrams, how pivoting may be incorporated in BNA. We will leave until later the discussion of the pivoting strategy, which involves selection of the pivot elements to get the best numerical performance.

Note on Shape Diagrams

A shape diagram indicates the areas of a matrix that are Toeplitz, non-Toeplitz and null. Elements of the matrix are at the intersection of the grid lines. Toeplitz areas are bounded by solid lines, non-Toeplitz areas are bounded by dotted lines, and all other areas are null. Extra information is put on the diagram as needed. So in Figure 2.1, areas A and B are Toeplitz, areas C and D are non-Toeplitz, and areas E and F and null.

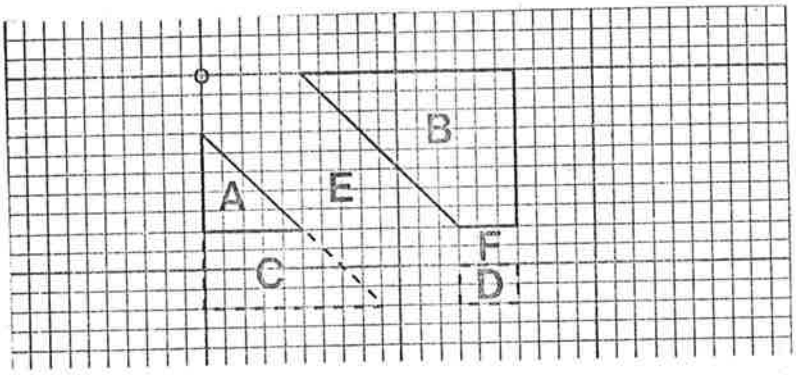


Fig. 2.1 - Example of a shape diagram

2.1 The Basic Pivoting Procedure

We henceforth denote the Toeplitz part of any Bareiss iterate $T^{(\pm i)}$ by $T_*^{(\pm i)}$, an element in the first non-zero diagonal above the zero-band of $T_*^{(\pm i)}$ by $t_A^{(\pm i)}$, and an element in the first non-zero diagonal below the zero-band by $t_B^{(\pm i)}$. At step $(-k)$, BNA uses $t_B^{(k-1)}$ as pivot and eliminates $t_B^{(1-k)}$, as is shown by the dashed arrow in Figure 2.2. It is, however possible to use $t_A^{(k-1)}$ as pivot to eliminate $t_A^{(1-k)}$.

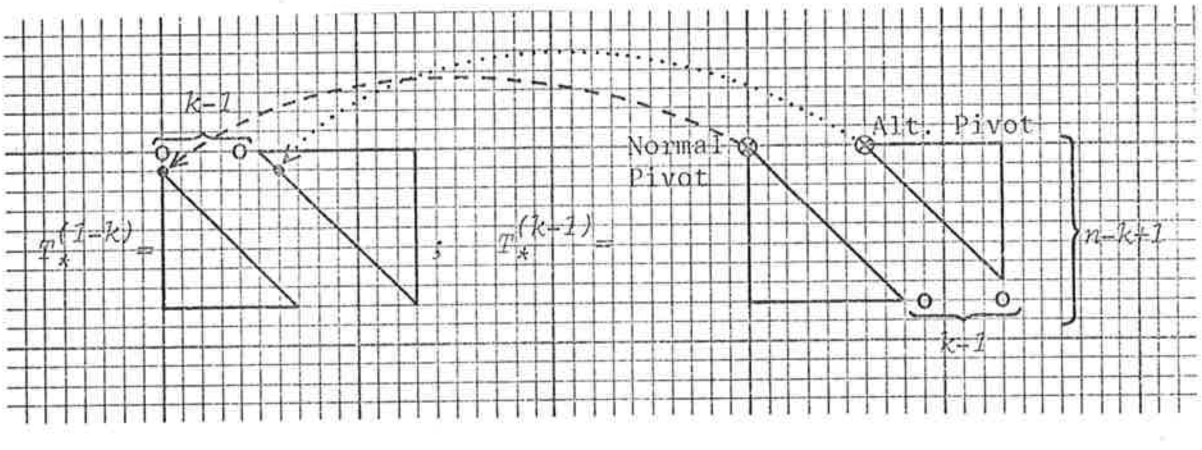


Fig. 2.2 - Form of $T_*^{(1-k)}$ and $T_*^{(k-1)}$ in BNA, with pivot choices $t_B^{(k-1)}$, $t_A^{(k-1)}$.

(i) BNA pivot (dashed arrow)

$$T_*^{(-k)} \leftarrow T_{*2:n-k+1\cdot}^{(1-k)} - m_{-k}\, T_{*1:n-k\cdot}^{(k-1)} \quad \text{where } m_{-k} = t_B^{(1-k)}/t_B^{(k-1)} \qquad (2.1a,b)$$

(ii) Alternative pivot (dotted arrow)

$$T_*^{(-k)} \leftarrow T_{*2:n-k+1\cdot}^{(1-k)} - m_{-k}\, T_{*1:n-k\cdot}^{(k-1)} \quad \text{where} \quad m_{-k} = t_A^{(1-k)}/t_A^{(k-1)} \qquad (2.2a,b)$$

At step *(k)*, BNA uses $t_A^{(-k)}$ as pivot and eliminates $t_A^{(k-1)}$, as is shown by the dashed arrow in Figure 2.3. It is, however, possible to use $t_B^{(-k)}$ as pivot to eliminate $t_B^{(k-1)}$.

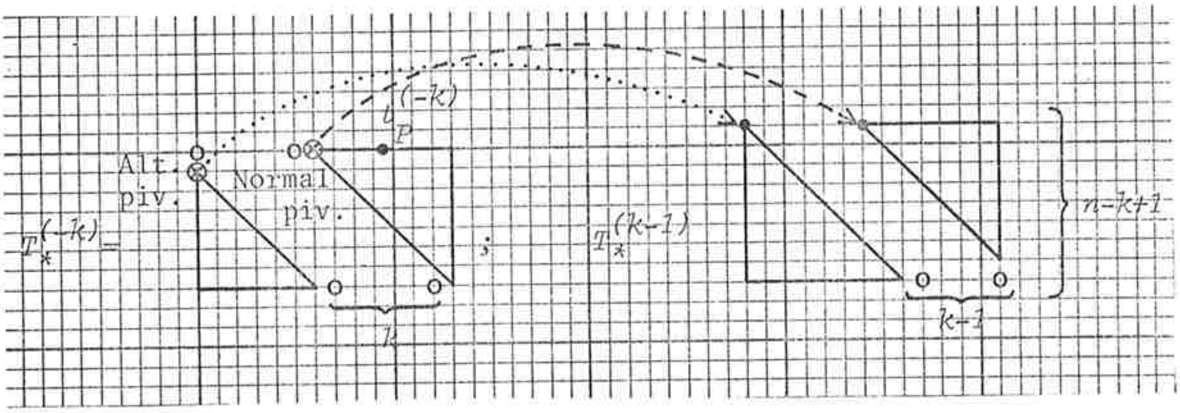


Fig. 2.3 - Form of $T_*^{(-k)}$ and $T^{(k)}$, with pivot choices $t_A^{(-k)}$, $t_B^{(-k)}$.

The two alternative operations in Figure 2.3 are given by:

(i) BNA pivot (dashed arrow)

$$T_*^{(k)} \leftarrow T_{*1:n-k\cdot}^{(k-1)} - m_k\, T_*^{(-k)}, \quad \text{where} \quad m_k = t_A^{(k-1)}/t_A^{(-k)} \qquad (2.3a,b)$$

(ii) Alternative pivot (dotted arrow)

$$T_*^{(k)} \leftarrow T_{*1:n-k-1\cdot}^{(k-1)} - m_k\, T_{2:n-k\cdot}^{(-k)}, \quad \text{where } m_k = t_B^{(k-1)}/t_B^{(-k)} \qquad (2.4a,b)$$

Reversed-order BNA

Normally in BNA, we calculate $T^{(-k)}$ first, then $T^{(k)}$. However, we could calculate $T^{(k)}$ before $T^{(-k)}$. This will be useful in one of the pivoting procedures to be described later (C-cycles). The diagrams and equations corresponding to the reversed-order BNA are similar to those above.

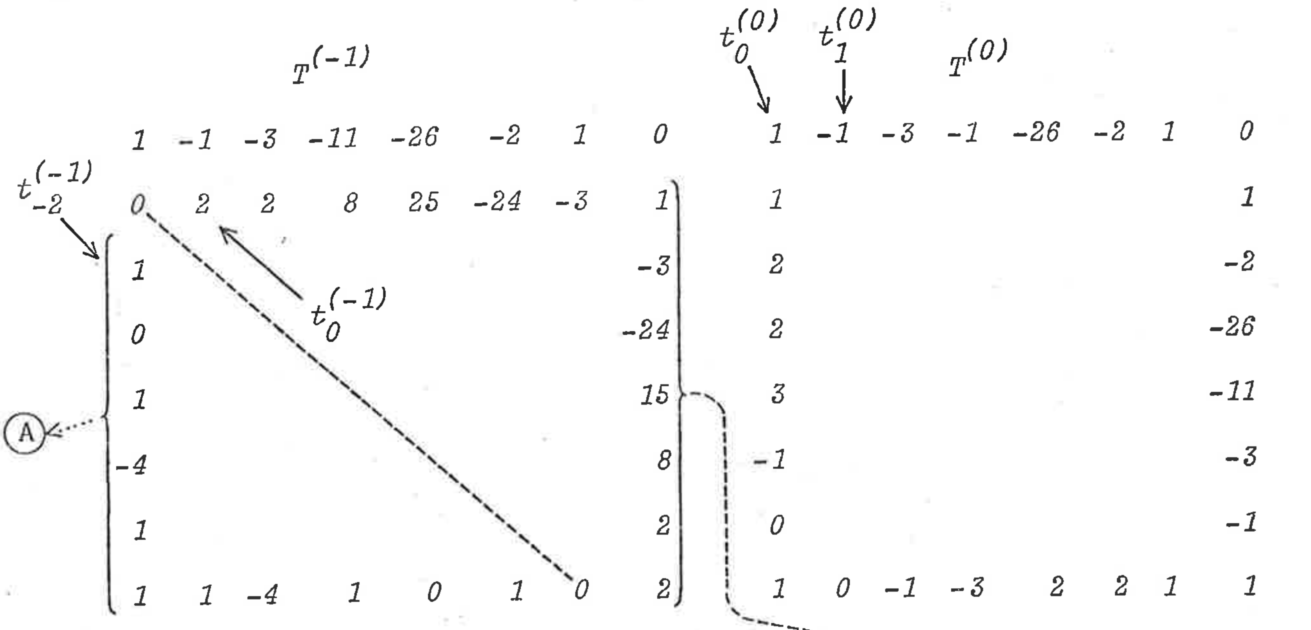
Basic Pivoting Operations - Summary

At each cycle k, we have the choice of whether to perform step *(-k)* or step *(k)* first; and at each step, we have the choice of whether to eliminate above or below the zero-band. All pivoting procedures discussed below are obtained from these basic pivoting operations.

Example 2.1:

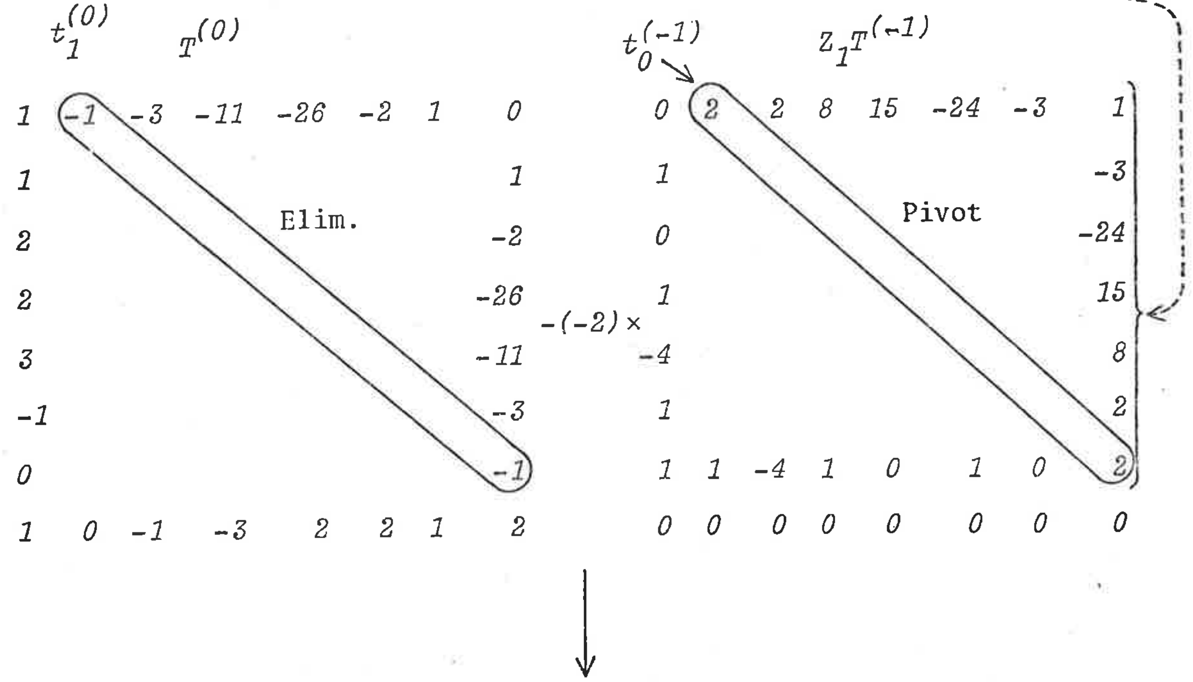
Basic Pivoting Procedure

Suppose after step *(-1)* of BNA we have



$$\begin{array}{cccccccc}
1 & -1 & -3 & -11 & -26 & -2 & 1 & 0 \\
0 & 2 & 2 & 8 & 25 & -24 & -3 & 1 \\
1 & & & & & & & -3 \\
0 & & & & & & & -24 \\
1 & & & & & & & 15 \\
-4 & & & & & & & 8 \\
1 & & & & & & & 2 \\
1 & 1 & -4 & 1 & 0 & 1 & 0 & 2
\end{array}
\qquad
\begin{array}{cccccccc}
1 & -1 & -3 & -1 & -26 & -2 & 1 & 0 \\
1 & & & & & & & 1 \\
2 & & & & & & & -2 \\
2 & & & & & & & -26 \\
3 & & & & & & & -11 \\
-1 & & & & & & & -3 \\
0 & & & & & & & -1 \\
1 & 0 & -1 & -3 & 2 & 2 & 1 & 1
\end{array}$$

Step *(1)* of the normal BNA-use $t_0^{(-1)}$ to elim. $t_1^{(0)}$



$$\begin{array}{cccccccc}
1 & -1 & -3 & -11 & -26 & -2 & 1 & 0 \\
1 & & & & & & & 1 \\
2 & & & & & & & -2 \\
2 & & & & & & & -26 \\
3 & & & & & & & -11 \\
-1 & & & & & & & -3 \\
0 & & & & & & & -1 \\
1 & 0 & -1 & -3 & 2 & 2 & 1 & 2
\end{array}
\quad -(-2)\times \quad
\begin{array}{cccccccc}
0 & 2 & 2 & 8 & 15 & -24 & -3 & 1 \\
1 & & & & & & & -3 \\
0 & & & & & & & -24 \\
1 & & & & & & & 15 \\
-4 & & & & & & & 8 \\
1 & & & & & & & 2 \\
1 & 1 & -4 & 1 & 0 & 1 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

$$\downarrow$$

$$T^{(1)}$$

$$\begin{array}{cccccccc}
1 & 0 & -1 & -7 & -18\frac{1}{2} & -14 & -1 & \frac{1}{2} \\
1\frac{1}{2} & & & & & & & -\frac{1}{2} \\
2 & & & & & & & -14 \\
3\frac{1}{2} & & & & \ddots & & & -18\frac{1}{2} \\
1 & & & & & & & -7 \\
-\frac{1}{2} & & & & & & & -2 \\
\frac{1}{2} & -\frac{1}{2} & 1 & 3\frac{1}{2} & 2 & 1\frac{1}{2} & 1 & 0 \\
1 & 0 & -1 & -3 & 2 & 2 & 1 & 1
\end{array}$$

Step *(1)*, alternative pivot - use $t_{-2}^{(-1)}$ to elim. $t_0^{(0)}$:

$$t_0^{(0)} \quad , \quad T^{(0)} \qquad\qquad\qquad t_2^{(-1)} \quad Z_2T^{(0)}$$

$$\begin{array}{cccccccc}
1 & -1 & -3 & -11 & -26 & -2 & 1 & 0 \\
1 & & & & & & & 1 \\
2 & & & \text{Elim.} & & & & -2 \\
2 & & & & & & & -26 \\
3 & & & & & & & -11 \\
-1 & & & & & 1 & & -3 \\
0 & & & & & & & -1 \\
1 & 0 & -1 & -3 & 2 & 2 & 1 & 1
\end{array}
\quad -(-1)\times \quad
\begin{array}{cccccccc}
1 & 0 & 2 & 2 & 8 & 15 & -24 & -3 \\
0 & & & & & & & -24 \\
1 & & & & \text{Pivot} & & & 15 \\
-4 & & & & & & & 8 \\
1 & & & & & & & 2 \\
1 & 1 & -4 & 1 & 0 & 1 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\\1\end{array}}\right\} \leftarrow \text{Ⓐ}$$

$$\downarrow$$

$$T^{(1)}$$

$$\begin{array}{cccccccc}
0 & 1 & -5 & -13 & -34 & -17 & 25 & 3 \\
1 & & & & & & & 25 \\
1 & & & & & & & -17 \\
-6 & & & \ddots & & & & -34 \\
2 & & & & & & & -13 \\
-2 & 2 & -6 & 1 & 1 & 0 & -1 & -5 \\
0 & -1 & -3 & 2 & 2 & 1 & 1 & -1 \\
1 & 0 & -1 & -3 & 2 & 2 & 1 & 1
\end{array}$$

Remark: Note that the Toeplitz part of $T^{(1)}$ above consists of a triangle and a trapezoid, whereas in BNA, the Toeplitz part always consists of two triangles (see e.g. the fifth matrix in the above example) - we call this latter form the Bareiss form. In general, the Bareiss form will be lost whenever non-Bareiss pivots are used. We may wish to restore the Bareiss form to enable BNA to be resumed. This is discussed later.

2.2 The Extended Pivoting Procedure: Backtracking

It sometimes happens that at step k of BNA, neither $t_A^{(-k)}$ nor $t_B^{(-k)}$ (see Figure 2.3) are satisfactory pivots. Within certain constraints, it is possible to select a pivot different from $t_A^{(-k)}$ $t_B^{(-k)}$. Suppose we wish to use $t_P^{(-k)}$ (indicated in Figure 2.3) as pivot, and $t_P^{(-k)}$ is p places above $t_A^{(-k)}$. This cannot be done immediately, because several of zero-diagonals in $T^{(k-1)}$ will be destroyed. $t_P^{(-k)}$ can, however, be made available for pivoting by backtracking in a way that, in principle, moves the zero-band of $T^{(-k)}$ up p places so that it adjoins $t_P^{(-k)}$. The zero-band of $T^{(k-1)}$ will also be moved up p places, so that the new $T_*^{(-k)}$ and $T_*^{(k-1)}$ look as follows:

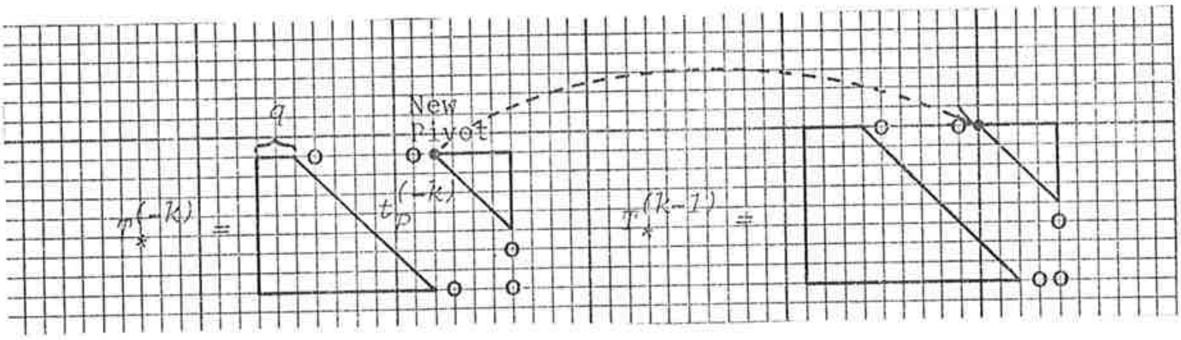


Fig. 2.4 - As for Fig. 2.3, but with zero-band moved up p places $(p=3)$.

$t_p^{(-k)}$ can then be used as pivot, as is shown in Figure 2.4. Note again that the Toeplitz blocks above are not in Bareiss form, i.e. do not have two Toeplitz triangles each. After the new pivot is used, it may be desired to recover the Toeplitz triangles so that BNA may be resumed. This so-called restoration is discussed later.

Note that we may wish to move the zero-band down instead of up. The methods discussed in the following allow the zero-band to be moved in either direction.

Suppose we wish to move the zero-band p places ($p>0$ up: $p<0$: down). There are two cases to consider, $|p| \geqslant k$ and $|p| < k$. If $|p| \geqslant k$, we must go back to the beginning, and modify BNA to eliminate the desired diagonals. We call this operation Backtrack Procedure A (BPA). If $|p| \geqslant k$, we have a choice of two methods, Backtrack Procedure B (BPB) or Backtrack Procedure C (BPC). In BPB, we go back $|p|$ cycles and then use the basic pivoting procedure to eliminate the desired diagonals. In BPC, we repeat the following procedure $|p|$ times: go back one cycle by running the normal BNA in reverse, then go forward one cycle, selecting the pivots to move the zero-band one place in the desired direction. Whether BPB or BPC is to be preferred depends on the pivoting sequence and will be discussed in the section on strategy.

We explain the backtrack procedures in detail below.

2.2.1 Backtrack Procedure A (BPA)

Let p be the desired displacement of the zero-band (positive for upshift, negative for downshift). We proceed as in BNA except that we start by eliminating t_{p-1} and t_{p+1} instead of t_{-1} and t_1, respectively.

When $p>0$ the first two steps of Backtrack Procedure A can be shown as follows:

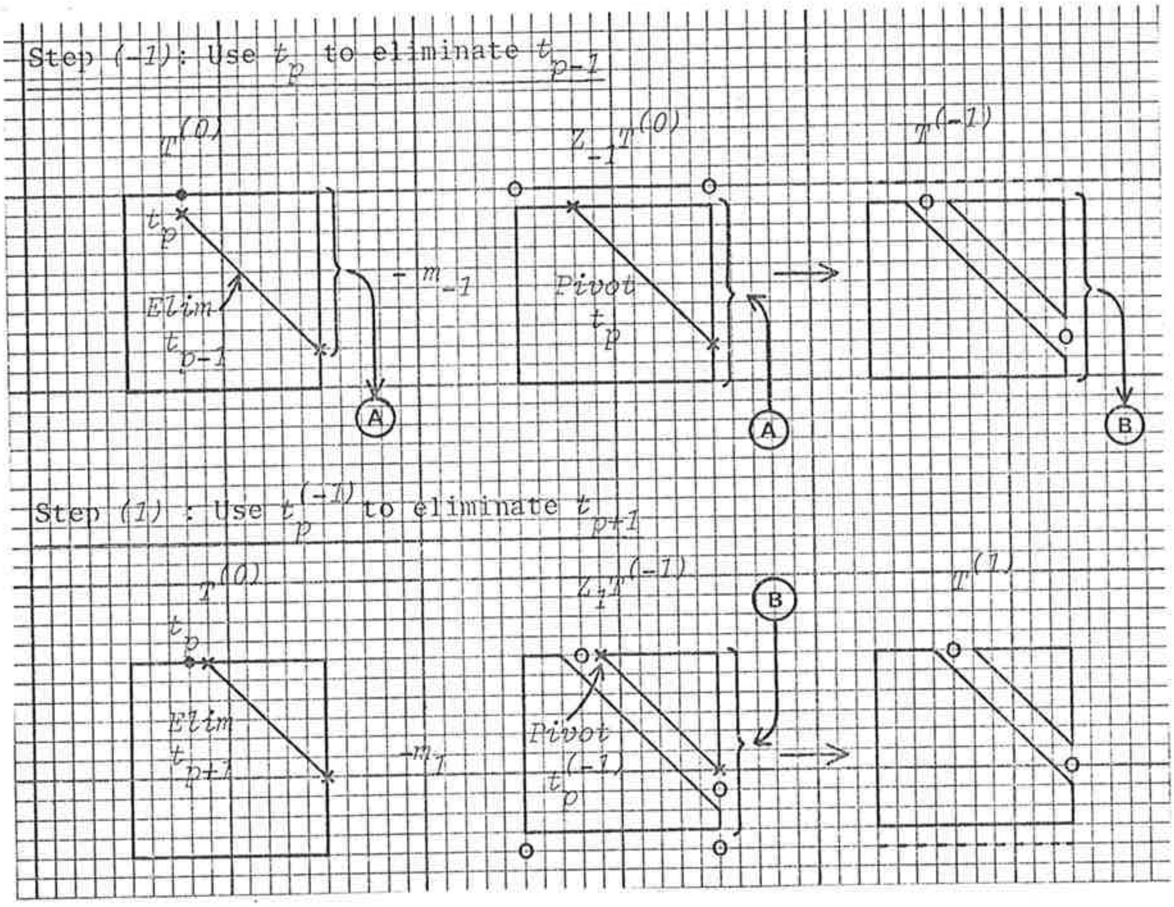


Fig. 2.5 - Steps (-1) and (1) of Backtrack Procedure A.

We continue eliminating below diagonal p in the negative-index iterates and above diagonal p in the positive-index iterates. After step $-k$, the matrices will have the form of Figure 2.4. Backtrack Procedure A may be stated formally as follows, where p is the desired shift:

Procedure 2.1 - Backtrack Procedure A {l,r : row pointers; i : cycle counter}

1. $T^{(0)} \leftarrow T$; $l \leftarrow 2$; $r \leftarrow n-1$; $i \leftarrow 1$ (3.5a-c)

2. $T^{(-i)}_{l:n.} \leftarrow T^{(1-i)}_{l:n.} - m_{-i}\ T^{(i-1)}_{1:r.}$, where $m_{-i} = t^{(1-i)}_{p-i}/t^{(i-1)}_{p}$ (3.6a,b)

3. Stop if $i = k$.

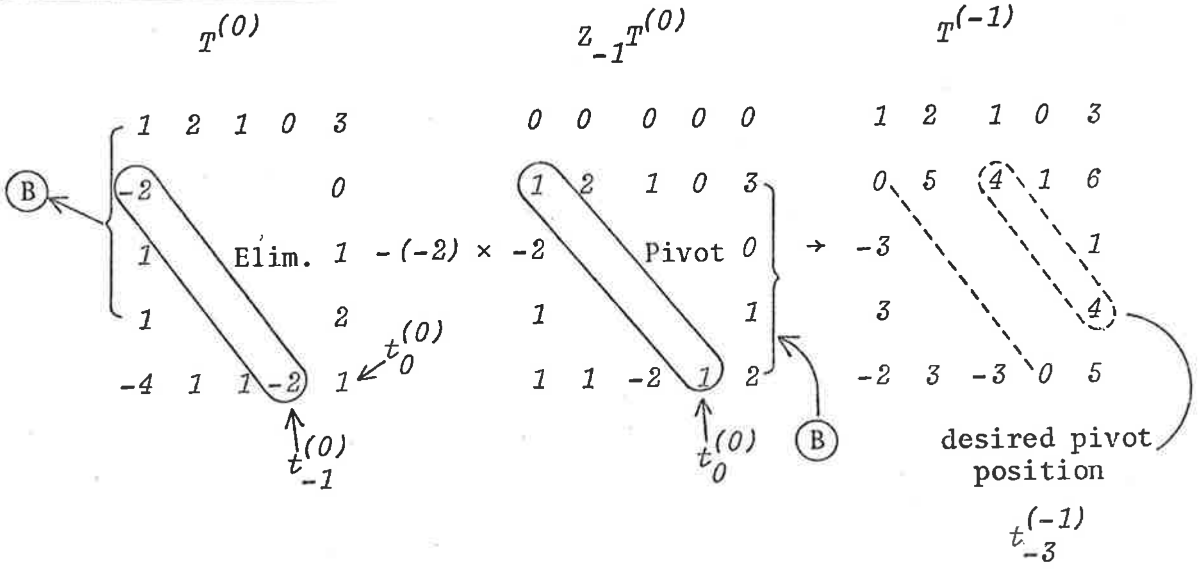
4. $T^{(i)}_{1:r.} \leftarrow T^{(i-1)}_{1:r.} - m_i T^{(-i)}_{l:n.}$, where $m_i = t^{(i-1)}_{p+i} / t^{(-i)}_{p}$ (3.7a,b)

5. $i \leftarrow i + 1$; $l \leftarrow l + 1$; $r \leftarrow r - 1$; go to 2.

Note that the procedure is the same as BNA, if p is set to zero.

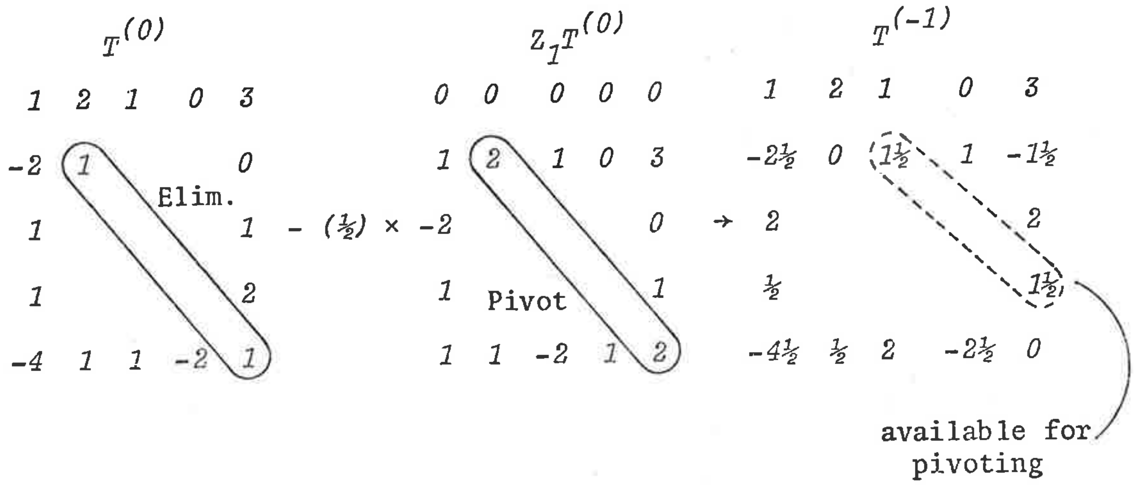
Example 2.2

Normal Bareiss - Use $t^{(0)}_0$ to elim. $t^{(0)}_1$:



$$T^{(0)} = \begin{array}{ccccc} 1 & 2 & 1 & 0 & 3 \\ -2 & & & & 0 \\ 1 & & & & 1 \\ 1 & & & & 2 \\ -4 & 1 & 1 & -2 & 1 \end{array} \quad - (-2) \times \quad Z_{-1}T^{(0)} = \begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 3 \\ -2 & & & & 0 \\ 1 & & & & 1 \\ 1 & 1 & -2 & 1 & 2 \end{array} \rightarrow T^{(-1)} = \begin{array}{ccccc} 1 & 2 & 1 & 0 & 3 \\ 0 & 5 & 4 & 1 & 6 \\ -3 & & & & 1 \\ 3 & & & & 4 \\ -2 & 3 & -3 & 0 & 5 \end{array}$$

Backtrack Procedure A - use $t^{(0)}_1$ to elim. $t^{(0)}_0$:



$$T^{(0)} = \begin{array}{ccccc} 1 & 2 & 1 & 0 & 3 \\ -2 & 1 & & & 0 \\ 1 & & & & 1 \\ 1 & & & & 2 \\ -4 & 1 & 1 & -2 & 1 \end{array} \quad - (\tfrac{1}{2}) \times \quad Z_{1}T^{(0)} = \begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 3 \\ -2 & & & & 0 \\ 1 & & & & 1 \\ 1 & 1 & -2 & 1 & 2 \end{array} \rightarrow T^{(-1)} = \begin{array}{ccccc} 1 & 2 & 1 & 0 & 3 \\ -2\tfrac{1}{2} & 0 & 1\tfrac{1}{2} & 1 & -1\tfrac{1}{2} \\ 2 & & & & 2 \\ \tfrac{1}{2} & & & & 1\tfrac{1}{2} \\ -4\tfrac{1}{2} & \tfrac{1}{2} & 2 & -2\tfrac{1}{2} & 0 \end{array}$$

Note that $t_1^{(-1)}$ in BPA is different from $t_1^{(-1)}$ in BNA, as is shown by the example above. The strategy described later aims to keep this difference small.

2.2.2 Backtrack Procedure B

In this case, we go back to iterates $-(k-|p|)$ and $(k-|p|)$, and select pivots in such a manner as to move the zero-band in the desired direction. To move the zero-band up, we perform a series of A-cycles; to move the zero-band down, we perform a series of C-cycles. A and C-cycles are described below.

A-cycles.

(i) Step $(-i-1)$. With $t_A^{(i)}$ as pivot, eliminate $t_A^{(-i)}$.

(ii) Step $(i+1)$. With $t_A^{(-i-1)}$ as pivot, eliminate $t_A^{(i)}$.

Figure 2.6 shows a typical A-cycle. The initial zero-band displacement (ZBD) is p, and the final ZBD is $p+1$. It can be checked from Figure 2.6 that the operations in an A-cycle are:

Procedure 2.2 : A-cycle

Input: $T^{(\pm i)}$; h, j : indices of first and last rows of Toeplitz part of $T^{(-i)}$;

r, s : indices of first and last rows of Toeplitz part of $T^{(i)}$.

```
1.  $T^{(-i-1)} \leftarrow T^{(-i)} - m_{-i-1} C_{r-h-1} S_{r,s-1} T^{(i)}$,  where $m_{-i-1} = t_A^{(-i)}/t_A^{(i)}$  (2.8)

2.  $T^{(i+1)} \leftarrow T^{(i)} - m_{i+1} C_{h-r} S_{h+1,j} T^{(-i-1)}$,  where $m_{i+1} = t_A^{(i)}/t_A^{(-i-1)}$  (2.9)

3.  $h \leftarrow h+1$;  $r \leftarrow r+1$.
```

Note: C_j is a cyclic permutation matrix, defined by $(C_j)_{pq} = \delta_{p-q-j \pmod n}$; S_{jk} is a selection matrix: $(S_{jk})_{pq} = 0$ except for $j \le p = q \le k$, when $(S_{jk})_{pq} = 1$.

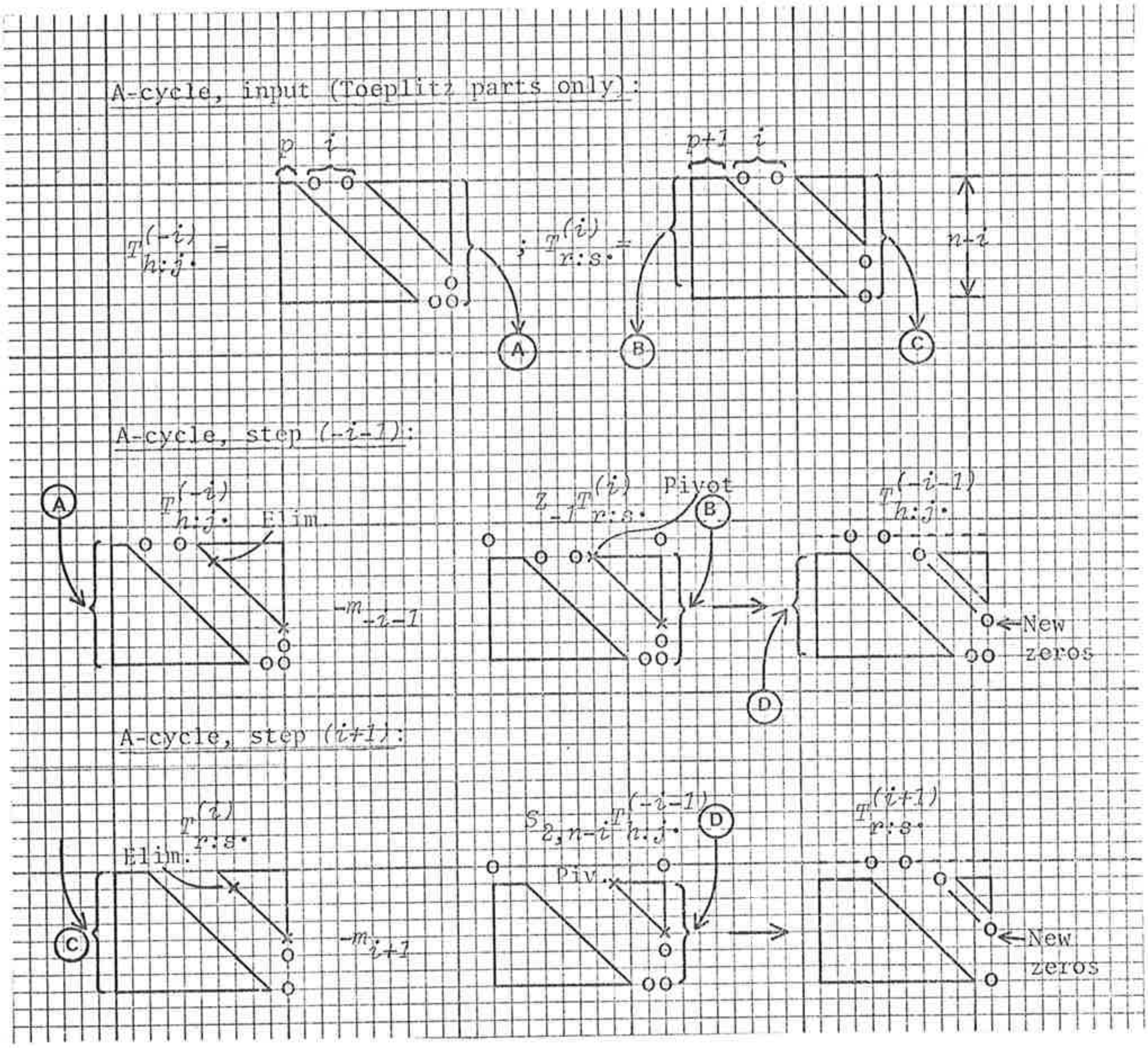
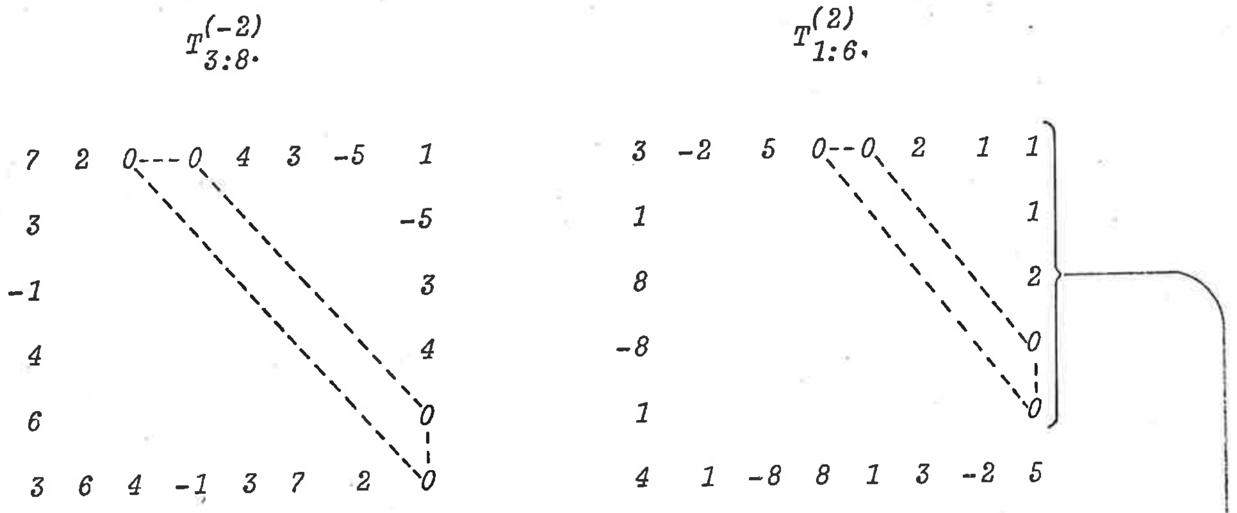


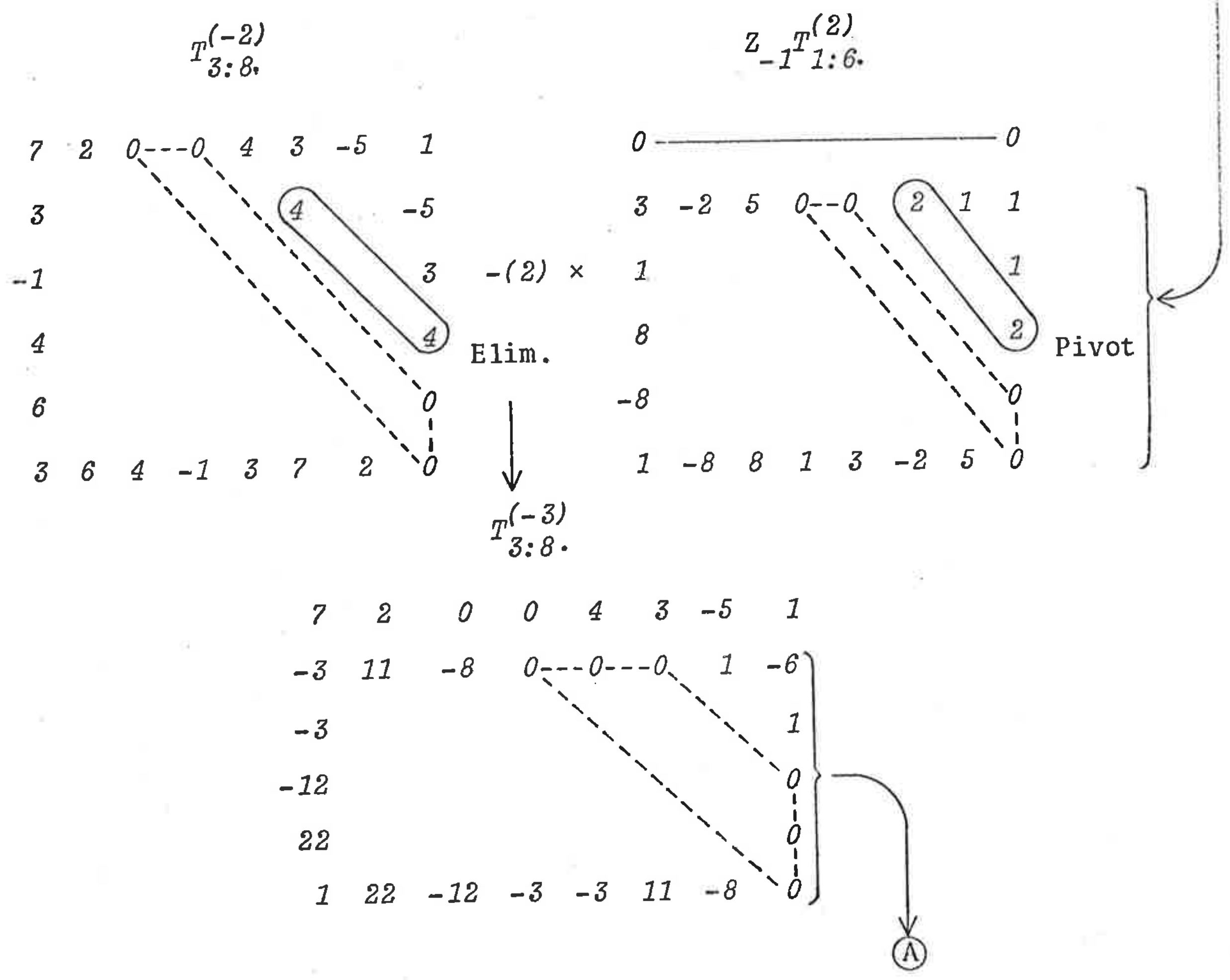
Fig. 2.6 - Steps *(-i-1)* and *(i+1)* of a typical A-cycle.

Example 2.3 - A cycle

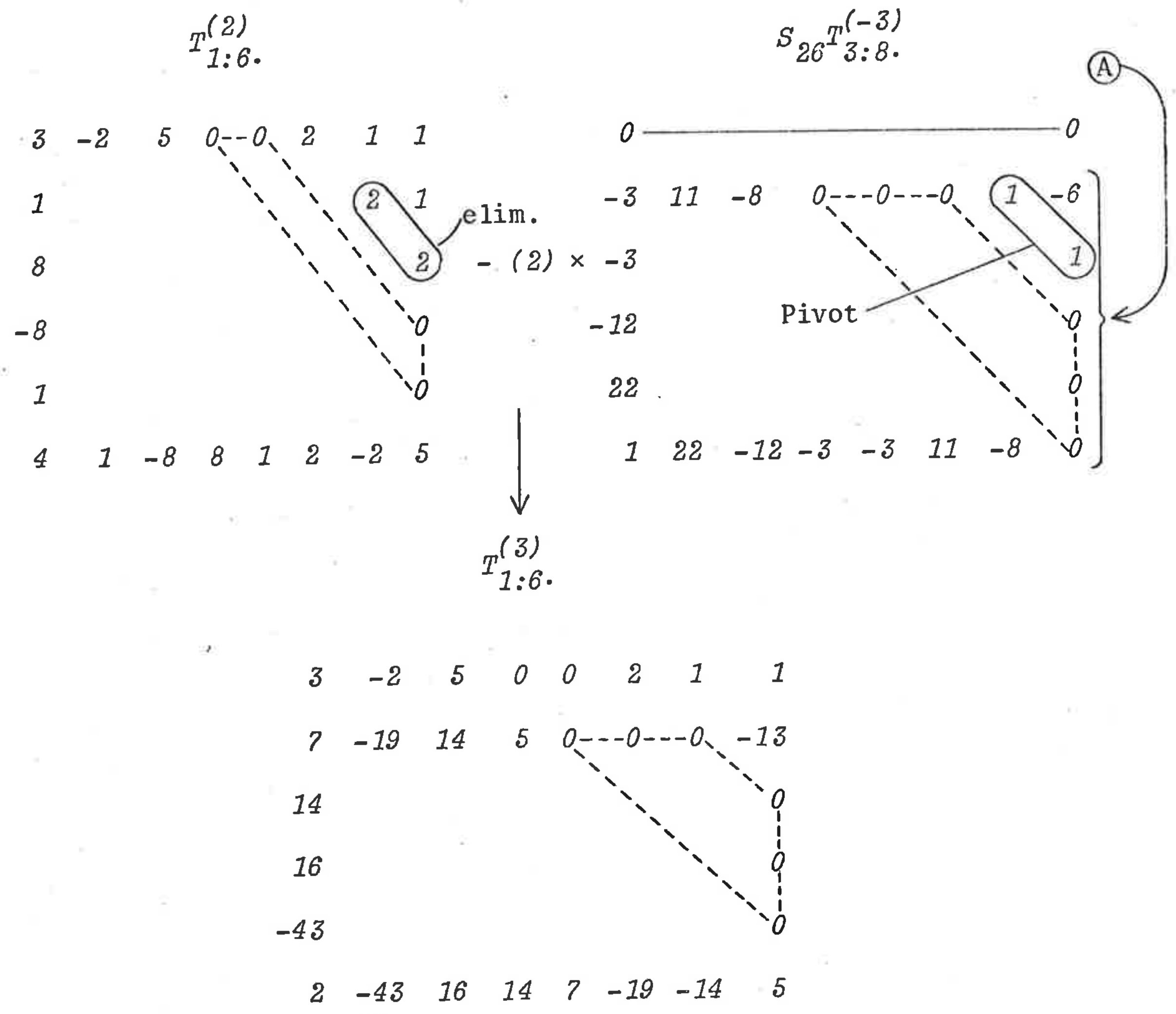
The inputs are the Toeplitz parts of $T^{(-2)}$ and $T^{(2)}$ with a zero-band displacement of 2. The example shows the working of an A-cycle.



A-cycle, step (-3) : use $t_A^{(2)}$ to eliminate $t_A^{(-2)}$:



A-cycle, step (3) : use $t_A^{(-3)}$ to eliminate $t_A^{(2)}$:



C-cycles:

Recall that a C-cycle moves the zero-band down. Cycle $(i+1)$ is:

(i) Step $(i+1)$. With $t_B^{(-i)}$ as pivot, eliminate $t_B^{(i)}$.

(ii) Step $(-i-1)$. With $t_B^{(i+1)}$ as pivot, eliminate $t_B^{(-i)}$.

Figure 2.7 shows a typical C-cycle. The initial zero-band displacement (ZBD) is $-q$ and the final ZBD is $-q-1$. It can be checked from Figure 2.7 that the operations in the C-cycle are:

Procedure 2.3 - C-cycle

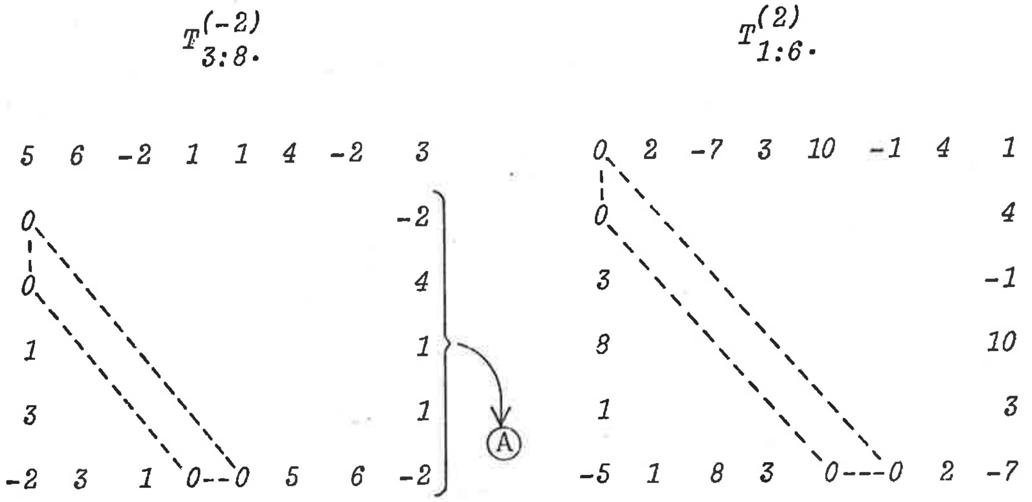
Input: As for Procedure 2.2.

1. $$T^{(i+1)} \leftarrow T^{(i)} - m_{i+1}C_{h+1-r}S_{h+1,j}T^{(-i)}, \quad \text{where } m_{i+1} = t_B^{(i)}/t_B^{(-i)} \tag{2.10}$$

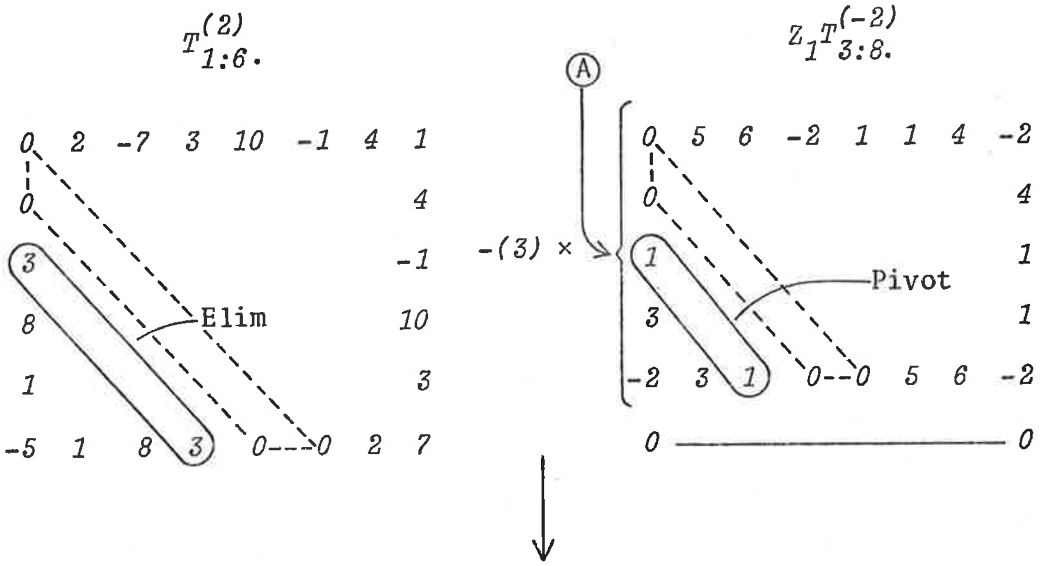
2. $$T^{(-i-1)} \leftarrow T^{(-i-1)} - m_{-i-1}C_{r-h}S_{r,s-1}T^{(i+1)}, \quad \text{where } m_{-i-1} = t_B^{(-i)}/t_B^{(i+1)} \tag{2.11}$$

Example 2.4

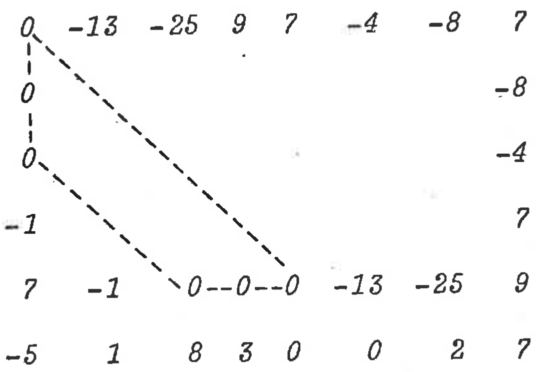
The inputs are the Toeplitz parts of $T^{(-2)}$ and $T^{(2)}$ with a zero-band displacement of *-2*. The example shows the working of a C-cycle.



C-cycle, step (3) : Use $t_B^{(-2)}$ to eliminate $t_B^{(2)}$:

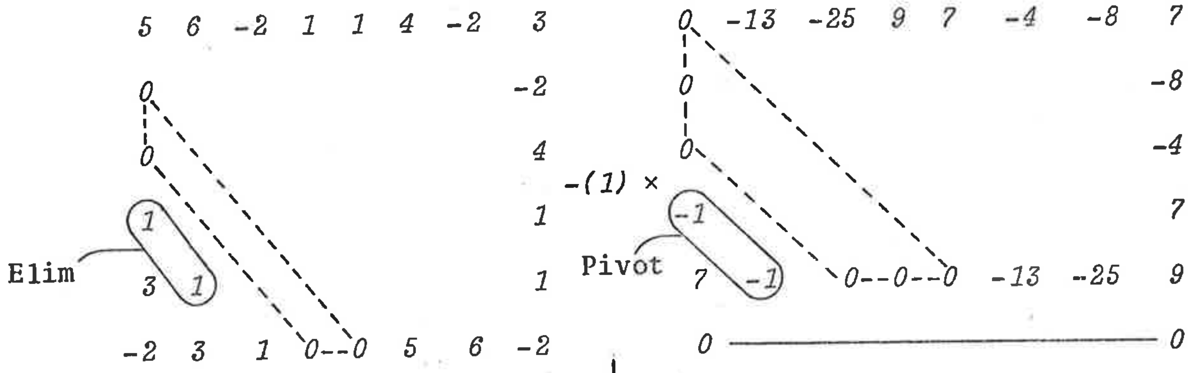


$T^{(3)}_{1:6\cdot}$

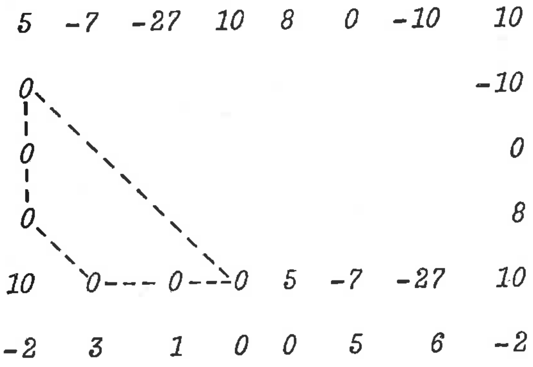


C-cycle, step (-3) : Use $t_B^{(3)}$ to eliminate $t_B^{(-2)}$

$T^{(-2)}_{3:8\cdot}$ $S_{25}\ T^{(3)}_{1:6\cdot}$



$T^{(-3)}_{3:8\cdot}$



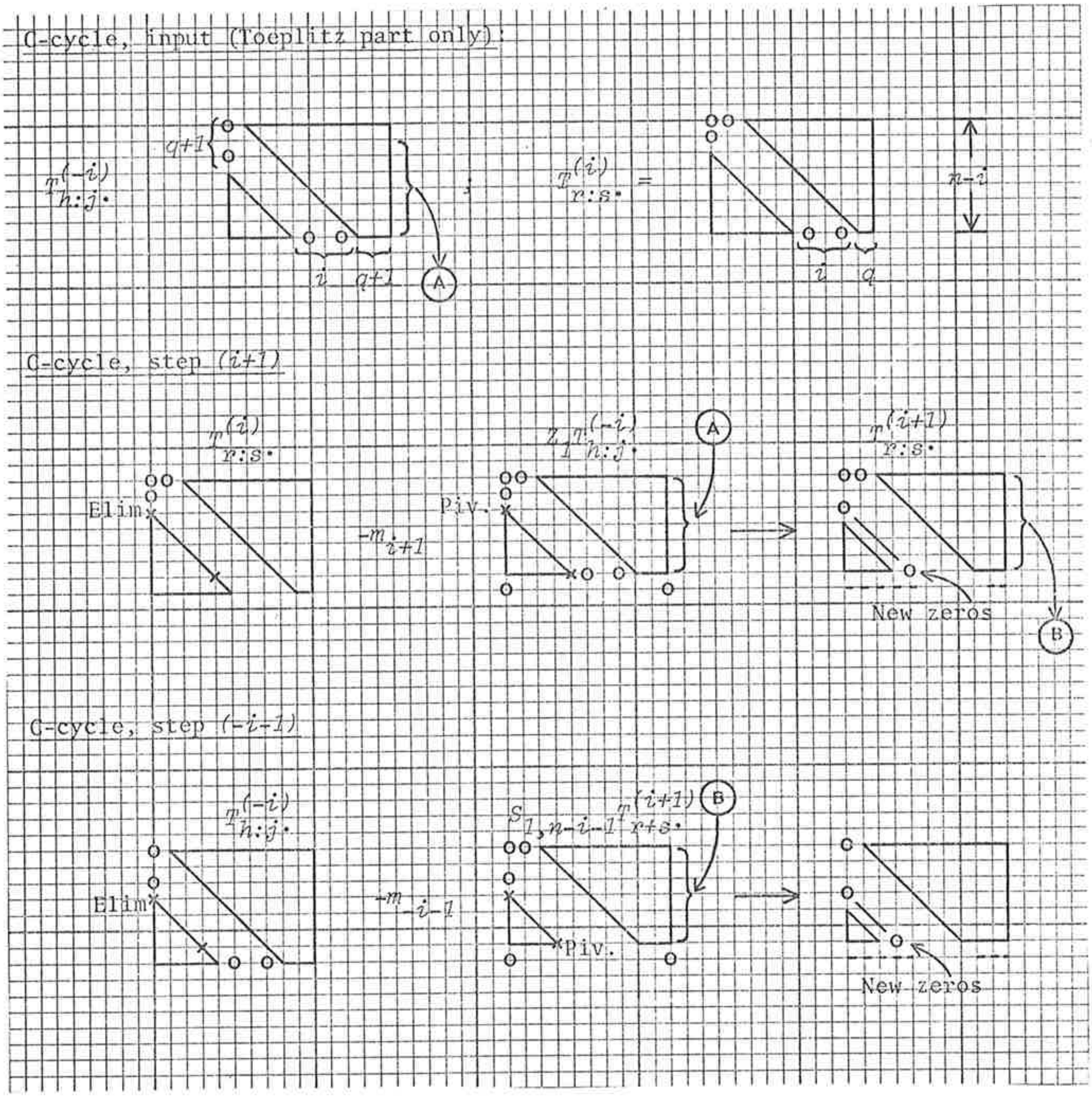
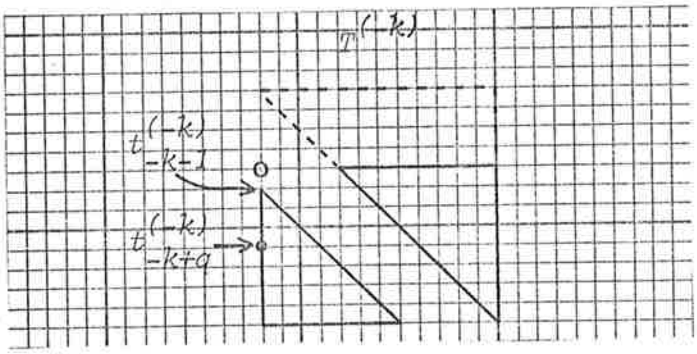


Fig. 2.7 - Steps *(i+1)* and *(-i-1)* of a C-cycle

Special Note on the use of C-cycles for BPB

We indicated before that when the desired pivot, $t_p^{(-k)}$ is above the zero-band, we make it available for pivoting by executing p A-cycles to move the zero-band up p places. If the desired pivot is below the zero-band, say in position $t_{-k-1-q}^{(-k)}$, as shown below,

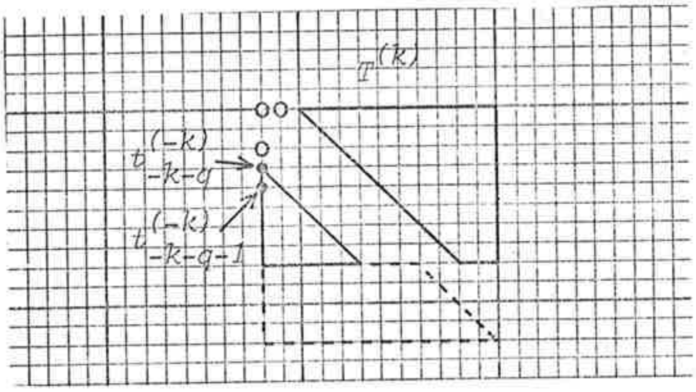


then we must move the zero-band down not q places, as might be supposed from the diagram, but $q+1$ places. This is a consequence of the following two facts:

(a) In any C-cycle, $T^{(i)}$ is calculated before $T^{(-i)}$.

(b) It can be shown that for any C-cycle (and also for A and BNA cycles) $T_*^{(i)}$ is $T_*^{(-i)}$ with a scaling factor applied and the zero-band shifted up one place by any backtrack procedure.

To see that $q+1$ shifts are required, observe that if the zero-band is moved down only q places, we get, at step k, ignoring the constant scaling factor,



then, at step $(-k)$, we must use $t_{-k-q}^{(-k)}$ as pivot; however, this frustrates our intention to use $t_{-k-q-1}^{(-k)}$ as pivot; therefore, we must move the zero-band down $q+1$, rather than q places. In short, we need an extra shift when doing C-cycles because the desired pivot now appears in the positive index iterate $T^{(k)}$, rather than the negative index iterate $T^{(-k)}$.

BPB - Summary

Suppose at step k we wish to use a pivot which is q places above or below the zero-band. We execute

Procedure 2.4 - Backtrack Procedure B

1. If the pivot is above the zero-band, set $p=q$, otherwise set $p=-q-1$.
2. Get $T^{-(k-|p|)}$, $T^{(k-|p|)}$.
3. If $p > 0$, execute p A-cycles, yielding $T^{(-k)}$ and $T^{(k)}$ with ZBD = p.
   else execute $|p|$ C-cycles, yielding $T^{(-k)}$ and $T^{(k)}$ with ZBD = p.

Suppose $t_p^{(-k)}$ is made available for pivoting by the execution of BPB. In general, this will be different from the $t_p^{(-k)}$ produced by BNA. The strategy described later aims to keep this difference small.

2.2.3 Backtrack Procedure C (BPC)

This is an alternative method to BPB for moving the zero-band in a Toeplitz iterate. Here, BNA has to be run in reverse, and for this, the Toeplitz parts of the multiplier matrices $M^{(\pm s)}$, defined by $M^{(\pm s)}T := T^{(\pm s)}$, are required. These can be calculated by a recursion similar to that in steps 3.3 and 3.6 of ABSA. We now describe the reverse BNA (RBNA).

The Reverse BNA : R-cycles

Let A_* be the Toeplitz part of any matrix A. Suppose $T_*^{(-s)}$, $M_*^{(-s)}$, $T_*^{(s)}$ and $M_*^{(s)}$ are given, and we wish to calculate $T_*^{(1-s)}$, $M_*^{(1-s)}$, $T_*^{(s-1)}$ and $M_*^{(s-1)}$.

Let $M_*^{(1-s)} = M^{(1-s)}_{h:j\cdot}$ and $M_*^{(s-1)} = M^{(s-1)}_{u:v\cdot}$ (2.12a,b)

Then $M_*^{(-s)} = M^{(-s)}_{h+1,j\cdot} = M^{(1-s)}_{h+1,j\cdot} - m_{-s}M^{(s-1)}_{u:v-1\cdot}$ (2.13)

$$M_*^{(s)} = M^{(s)}_{u:v-1\cdot} = M^{(s-1)}_{u:v-1\cdot} - m_s M^{(-s)}_{h+1:j\cdot} \qquad (2.14)$$

Now the $M_*^{(\pm i)}$ are completely specified by their first row, denoted $\tilde{\underline{m}}^{(\pm i)T}$. It can be easily shown from (2.13) and (2.14) that

$$\tilde{\underline{m}}^{(-s)T} = \tilde{\underline{m}}^{(1-s)T}C_{-1}^{T} - m_{-s}\tilde{\underline{m}}^{(s-1)T} \qquad (2.15)$$

$$\tilde{\underline{m}}^{(s)T} = \tilde{\underline{m}}^{(s-1)T} - m_s\tilde{\underline{m}}^{(-s)T} \qquad (2.16)$$

Equations (2.15) and (2.16) may be rearranged to calculate m_s, m_{-s}, $\tilde{\underline{m}}^{(s-1)}$ and $\tilde{\underline{m}}^{(1-s)}$ from $\underline{m}^{(s)}$ and $\underline{m}^{(-s)}$. The last nonzero component of (2.16) is

$$\tilde{m}^{(s)}_{s+1} = -m_s\tilde{m}^{(-s)}_{s+1} \Rightarrow m_s = -\tilde{m}^{(s)}_{s+1}/\tilde{m}^{(-s)}_{s+1} \qquad (2.17)$$

Equation (2.16) may be re-arranged to give

$$\tilde{\underline{m}}^{(s-1)T} = \tilde{\underline{m}}^{(s)T} + \tilde{m}_s\underline{m}^{(-s)T} \qquad (2.18)$$

The first component of equation (2.15) is

$$\tilde{m}^{(-s)}_1 = -m_{-s}\tilde{m}^{(s-1)}_1 \Rightarrow m_{-s} = \tilde{m}^{(-s)}_1/\tilde{m}^{(s-1)}_1 \qquad (2.19)$$

Equation (2.15) can be rearranged to give

$$\tilde{\underline{m}}^{(1-s)T} = \left(\tilde{\underline{m}}^{(-s)T} + m_{-s}\tilde{\underline{m}}^{(s-1)T}\right)C_{-1} \qquad (2.20)$$

Equations (2.15) - (2.20) are the desired recursion for m_{-s}, m_s and the Toeplitz parts of $M^{(s-1)}$ and $M^{(1-s)}$.

Note that (2.12) ⇒ $T_*^{(1-s)} = T^{(1-s)}_{h:j\cdot}$ and $T_*^{(s-1)} = T^{(s-1)}_{u:v\cdot}$, and that (2.13) and (2.14) imply

$$T_*^{(-s)} = T^{(-s)}_{h+1:j\cdot} = T^{(1-s)}_{h+1:j\cdot} - m_{-s} T^{(s-1)}_{u:v-1\cdot} \tag{2.21}$$

$$T_*^{(s)} = T^{(s)}_{u:v-1\cdot} = T^{(s-1)}_{u:v-1\cdot} - m_s T^{(-s)}_{h+1:j\cdot} \tag{2.22}$$

respectively, so from (2.21) and (2.22),

$$T^{(s-1)}_{u:v-1\cdot} = T^{(s)}_{u:v-1\cdot} + m_s T^{(-s)}_{j+1:j\cdot} \tag{2.23}$$

$$T^{(1-s)}_{h+1:j\cdot} = T^{(-s)}_{h+1:j\cdot} + m_{-s} T^{(s-1)}_{u:v-1\cdot} \tag{2.24}$$

$\underline{t}_h^{(1-s)}$ is also part of $T_*^{(1-s)}$. The first $n-1$ elements are, by Toeplicity

$$\underline{t}^{(1-s)}_{h,1:n-1\cdot} = \underline{t}^{(1-s)}_{h,2:n} , \tag{2.25}$$

and using the definition of the multiplier,

$$t^{(1-s)}_{hn} = \underline{\tilde{m}}^{(1-s)T} \underline{t}_{\cdot n} \tag{2.26}$$

Similarly, the last row of $T_*^{(s-1)}$ is given by

$$\underline{t}^{(s-1)}_{v,2:n} = \underline{t}^{(s-1)}_{v-1,1:n-1} \tag{2.27}$$

$$\begin{aligned} t^{(s-1)}_{v,1} &= \underline{m}^{(s-1)}_{v\cdot} \underline{t}_{\cdot 1} \\ &= \underline{\tilde{m}}^{(s-1)T} C^T_{u-v} \underline{t}_{\cdot 1} \text{ , by Toeplicity.} \end{aligned} \tag{2.28}$$

Equations (2.17) - (2.20), (2.23) - (2.28) constitute the desired recursion for RBNA. We call this an R-cycle, summarized in the following procedure:

Procedure 2.5 : R-cycle

Input: $T_*^{(-s)} = T_{h+1:j\cdot}^{(-s)}$, $\underline{\tilde{m}}^{(-s)T}$, $T_*^{(s)} = T_{u:v-1\cdot}^{(s)}$, $\underline{\tilde{m}}^{(s)T}$.

Operations: Execute in order equations (2.17) - (2.20), (2.23) - (2.28).

Output: $T_*^{(1-s)} = T_{h:j\cdot}^{(1-s)}$, m_{-s}, $\underline{\tilde{m}}^{(1-s)T}$, $T_*^{(s-1)} = T_{u:v\cdot}$, m_s, $\underline{\tilde{m}}^{(s-1)T}$.

BPC - Description

Having described RBNA, we can now give BPC. Suppose we have calculated $T^{(-k)}$ and wish to use $t_{-k-1-q}^{(-k)}$ as pivot. As for BPB, in the case we must move the zero-band down $q+1$ places. The procedure is:

Procedure 2.6a : BPC:zero-band downshift

(i) Recover $T_{k:n\cdot}^{(1-k)}$ (this requires only $2n$ storage locations).

(ii) For $i \leftarrow 1$ to q do:

(a) Run an R-cycle of RBNA to get $T_*^{(2-k)}$ and $T_*^{(k-2)}$.

(b) Run a C-cycle to give $T_*^{(1-k)}$ and $T_*^{(k-1)}$ with the zero-bands moved down one place. It may be readily checked that the recursions for the $\underline{\tilde{m}}^{(\pm i)T}$ in the C-cycle are:

$$\underline{\tilde{m}}^{(k-1)T} = \underline{\tilde{m}}^{(k-2)T} - m_{k-1}\underline{\tilde{m}}^{(2-k)T} C_{-1}^{T}$$

$$\underline{\tilde{m}}^{(1-k)T} = \underline{\tilde{m}}^{(2-k)T} - m_{1-k}\underline{\tilde{m}}^{(k-1)T}$$

(c) Calculate $T^{(-k)}$, $T^{(k)}$ by a further C-cycle (which results in the zero-band being moved down the $q+1^{\underline{st}}$ place).

Similarly, if we wish to use $t_p^{(-k)}$ $(p>0)$ as pivot, we must move the zero-band up p places; we do this as follows:

Procedure 2.6b : BPC - Zero-band upshift

(i) Recover $T_{k:n}^{(1-k)}$.

(ii) For $i \leftarrow 1$ to $p-1$ run R and A-cycles alternately.

(iii) Calculate $T^{(-k)}$, $T^{(k)}$ by a further A-cycle.

Fig. 2.8 shows the working of one cycle of BPC.

Example 2.5

Example 2.1 of Chapter 2 shows the working of the normal BNA. Consider iterate (-2) of that example

$$T^{(-2)}$$

$$\begin{array}{l|cccccccc}
 & 1 & -1 & -3 & -11 & -26 & -2 & -1 & 0 \\
 & 0 & 2 & 2 & 8 & 15 & -24 & -3 & 1 \\
 & 0 & 0 & 4 & 9 & 26\frac{1}{2} & 29 & -23\frac{1}{2} & -3\frac{1}{2} \\
\text{Desired Pivot} & -1\frac{1}{2} & & & \text{Normal BNA pivot} & & & & -23\frac{1}{2} \\
\text{Position } t_{-4}^{(-2)} \rightarrow & -1 & & & & & & & 29 \\
 & -7\frac{1}{2} & & & & & & & 26\frac{1}{2} \\
 & 0 & & & & & & & 9 \\
 & 1\frac{1}{2} & 0 & -7\frac{1}{2} & -1 & -1\frac{1}{2} & 0 & 0 & 4
\end{array}$$

Suppose we wish to make $t_{-4}^{(-2)}$ accessible by using BPC. We first recover $T^{(-1)}$, $T^{(1)}$ from memory (see example 2.1 of Chapter 2), $\underline{\tilde{m}}^{(-1)T}$ and $\underline{\tilde{m}}^{(1)T}$ (working not shown):

$$T^{(-1)} \qquad\qquad T^{(1)}$$

$$\begin{array}{cccccccc}
1 & -1 & -3 & -11 & -26 & -2 & 1 & 0\\
0 & 2 & 2 & 8 & 15 & -24 & -3 & 1\\
1 & \ddots & & & & & & -3\\
0 & & \ddots & & & & & -24\\
1 & & & \ddots & & & & 15\\
-4 & & & & \ddots & & & 8\\
1 & & & & & \ddots & & 2\\
1 & 1 & -4 & 1 & 0 & 1 & 0 & 2
\end{array}
\qquad
\begin{array}{cccccccc}
1 & 0 & -2 & -7 & -18\frac{1}{2} & -14 & -\frac{1}{2} & \frac{1}{2}\\
1\frac{1}{2} & & \ddots & & & & & -\frac{1}{2}\\
2 & & & \ddots & & & & -14\\
3\frac{1}{2} & & & & \ddots & & & -18\frac{1}{2}\\
1 & & & & & \ddots & & -7\\
-\frac{1}{2} & & & & & & \ddots & -2\\
\frac{1}{2} & -\frac{1}{2} & 1 & 3\frac{1}{2} & 2 & 1\frac{1}{2} & 1 & 0\\
1 & 0 & -1 & 3 & 2 & 2 & 1 & 1
\end{array}$$

$$\underline{\tilde{m}}^{(-1)T}_{1:2} = (-1,1) \qquad\qquad \underline{\tilde{m}}^{(1)T}_{-1,1:2} = (\tfrac{1}{2},\tfrac{1}{2})$$

Now, by equation (2.17) $\underline{m}_1 = -(\frac{1}{2})/1 = -\frac{1}{2}$

by equation (2.18) $\underline{\tilde{m}}^{(0)}_{-1,1:1} = (\frac{1}{2}) + (-\frac{1}{2})(-1) = (1)$

by equation (2.19) $m_{-1} = -1/1 = -1$

by equation (2.20) $\underline{\tilde{m}}^{(0)}_{1:1} = (1) + (-1)(0) = (1)$

Applying (2.23) we get

$$T^{(1)}_{1:7\cdot} \qquad\qquad T^{(-1)}_{2;8\cdot}$$

$$\begin{array}{cccccccc}
1 & 0 & -2 & -7 & -18\frac{1}{2} & -14 & -\frac{1}{2} & \frac{1}{2}\\
1\frac{1}{2} & & \ddots & & & & & -\frac{1}{2}\\
2 & & & \ddots & & & & -14\\
3\frac{1}{2} & & & & \ddots & & & -18\frac{1}{2}\\
1 & & & & & \ddots & & -7\\
-\frac{1}{2} & & & & & & \ddots & -2\\
\frac{1}{2} & -\frac{1}{2} & 1 & 3\frac{1}{2} & 2 & 1\frac{1}{2} & 1 & 0
\end{array}
+(-\tfrac{1}{2})
\begin{array}{cccccccc}
0 & 2 & 2 & 8 & 15 & -24 & -3 & 1\\
1 & \ddots & & & & & & -3\\
0 & & \ddots & & & & & -24\\
1 & & & \ddots & & & & 15\\
-4 & & & & \ddots & & & 8\\
1 & & & & & \ddots & & 2\\
1 & 1 & -4 & 1 & 0 & 1 & 0 & 2
\end{array}$$

$$\downarrow$$

$$\downarrow$$

$$T^{(0)}_{1;7}$$

$$\begin{array}{cccccccc}
1 & -1 & -3 & -11 & -26 & -2 & 1 & 0 \\
1 & & & & & & & 1 \\
2 & & & & & & & -2 \\
2 & & & & & & & -26 \\
-3 & & & & & & & -11 \\
-1 & & & & & & & -3 \\
0 & -1 & -3 & 2 & 2 & 1 & 1 & -1
\end{array}$$

Applying (2.27), $t^{(0)}_{-8,2:8} = (0\ -1\ -3\ 2\ 2\ 1\ 1)$

Applying (2.28), $t^{(0)}_{81} = (1) \times (1) = 1$

Hence the Toeplitz part of $T^{(0)}$ (in fact, the whole of $T^{(0)}$) has been reconstructed. The Toeplitz part of $T^{(-0)}(=T=T^{(0)})$ can be similarly reconstructed*. Next we perform:

BPC, Step (1) - use $t^{(0)}_{-1}$ to elim $t^{(0)}_{0}$ (C-cycle)

$$\begin{array}{cccccccc}
\multicolumn{8}{c}{T^{(0)}} \\
1 & -1 & -3 & -11 & -26 & -2 & 1 & 0 \\
1 & & & & & & & 1 \\
2 & & & \text{Elim.} & & & & -2 \\
2 & & & & & & & -26 \\
-3 & & & & & & & -11 \\
-1 & & & & & & & -3 \\
0 & & & & & & & -1 \\
1 & 0 & -1 & 3 & 2 & 2 & 1 & 1
\end{array}
\quad -(1)\times \quad
\begin{array}{cccccccc}
\multicolumn{8}{c}{Z_1T^{(0)}} \\
1 & 1 & -1 & -3 & -11 & -26 & -2 & 1 \\
2 & & & & & & & -2 \\
2 & & & & \text{Pivot} & & & -26 \\
-3 & & & & & & & -11 \\
-1 & & & & & & & -3 \\
0 & & & & & & & -1 \\
1 & 0 & -1 & -3 & 2 & 2 & 1 & 1 \\
0 & \multicolumn{6}{c}{\hrulefill} & 0
\end{array}$$

$$\downarrow$$

*The first iteration of step 2.1 of BPC (Procedure 2.6a) requires the recovery of $T^{(\pm(k-1))}$ by an R-cycle. We could instead, at any cycle k, store $T^{(\pm(k-1))}$ and $T^{(\pm(k-2))}$. This requires only $4n$ locations.

↓

$$T^{(1)}$$

$$\left.\begin{array}{cccccccc}
0 & -2 & -2 & -8 & -15 & 24 & 3 & -1 \\
-1 & & & & & & & 3 \\
0 & & & & & & & -24 \\
-1 & & & & & & & -15 \\
4 & & & & & & & -8 \\
-1 & & & & & & & -2 \\
-1 & -1 & 4 & -1 & 0 & -1 & 0 & -2
\end{array}\right\}\;\textcircled{A}\rightarrow$$

$$\begin{array}{cccccccc}
1 & 0 & -1 & 3 & 2 & 2 & 1 & 1
\end{array}$$

By (2.29) $\tilde{m}^{(1)T}_{-1:2}=(1\ 0)-(1)(0\ 1)=(1\ -1)$

BPC, step (-1): use $t^{(1)}_{-1}$ to elim. $t^{(0)}_{-1}$

$$T^{(0)} \qquad\qquad\qquad\qquad S_{1,7}T^{(1)}$$

$$\begin{array}{cccccccc}
1 & -1 & -3 & -11 & -26 & -2 & 1 & 0 \\
1 & & & & & & & 1 \\
2 & & & & & & & -2 \\
2 & & & & & & & -26 \\
-3 & & & & & & & -11 \\
-1 & & & & & & & -3 \\
0 & & & & & & \cdot & -1 \\
1 & 0 & -1 & 3 & 2 & 2 & 1 & 1
\end{array}
\;-(-1)\times\;
\begin{array}{cccccccc}
0 & -2 & -2 & -8 & -15 & 24 & 3 & -1 \\
-1 & & & & & & & 3 \\
0 & & & & & & & -24 \\
-1 & & & & & & & -15 \\
4 & & & & & & & -8 \\
-1 & & & & & & & -2 \\
-1 & -1 & 4 & -1 & 0 & -1 & 0 & -2 \\
0 & \multicolumn{6}{c}{\text{———————}} & 0
\end{array}\left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\\1\\1\end{array}}\right\}\;\textcircled{A}$$

↓

$$T^{(-1)}$$

$$\begin{array}{cccccccc}
1 & -3 & -5 & -19 & -41 & 22 & 4 & -1 \\
0 & & & & & & & 4 \\
2 & \ddots & & & & & & 22 \\
1 & & \ddots & & & & & -41 \\
-1 & & & \ddots & & & & -19 \\
-2 & & & & \ddots & & & -5 \\
-1 & -2 & -1 & 1 & 2 & 0 & 1 & -3 \\
1 & 0 & -1 & 3 & 2 & 2 & 1 & 1
\end{array}$$

By (2.30), $\tilde{m}^{(-1)T}_{-1;2} = (1\ 0)-(-1)(1\ -1) = (2\ -1)$

BPC, step (2): use $t^{(-1)}_{-2}$ to eliminate $t^{(1)}_{-1}$

$$T^{(1)} \qquad\qquad\qquad\qquad\qquad\qquad C_1S_{2,7}T^{(-1)}$$

$$\begin{array}{cccccccc}
0 & -2 & -2 & -8 & -15 & 24 & 3 & -1 \\
-1 & \ddots & & & & & & 3 \\
0 & & \ddots & & & & & -24 \\
-1 & & & \ddots & & & & -15 \\
4 & & & & \ddots & & & -8 \\
-1 & & & & & \ddots & & -2 \\
-1 & -1 & 4 & -1 & 0 & -1 & 0 & -2 \\
1 & 0 & -1 & 3 & 2 & 2 & 1 & 1
\end{array}
\quad -(-\tfrac{1}{2})\times \quad
\begin{array}{cccccccc}
0 & 1 & -3 & -5 & -19 & -41 & 22 & 4 \\
2 & \ddots & & & & & & 22 \\
1 & & \ddots & & & & & -41 \\
-1 & & & \ddots & & & & -19 \\
-2 & & & & \ddots & & & -5 \\
-1 & -2 & -1 & 1 & 2 & 0 & 1 & -3 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0
\end{array}$$

Elim.

Pivot

$$\downarrow$$

$$\downarrow$$

$$T^{(2)}$$

$$\begin{array}{l|cccccccc}
 & 0 & -1\tfrac{1}{2} & -3\tfrac{1}{2} & -10\tfrac{1}{2} & -24\tfrac{1}{2} & 3\tfrac{1}{2} & 14 & 1 \\
t_{-4}^{(2)} & 0 & \ddots & & & & & & 14 \\
\text{(scaled)} \rightarrow & \tfrac{1}{2} & & \ddots & & & & & 3\tfrac{1}{2} \\
 & -1\tfrac{1}{2} & & & \ddots & & & & -24\tfrac{1}{2} \\
 & 3 & & & & \ddots & & & -10\tfrac{1}{2} \\
 & -\tfrac{1}{2} & 3 & -1\tfrac{1}{2} & \tfrac{1}{2} & 0 & 0 & -1\tfrac{1}{2} & -3\tfrac{1}{2} \\
 & -1 & -1 & 4 & -1 & 0 & -1 & 0 & -2 \\
 & 1 & 0 & -1 & 3 & 2 & 2 & 1 & 1
\end{array}$$

Recall from the special note on C-cycles that $T_*^{(2)}$ above is a scaled version of $T_*^{(-2)}$ with a zero-band displacement of (-2), so $t_{31}^{(2)}$ (indicated above) is a scaled version of $t_{-4}^{(-2)}$. Recall also that as in BPA and BPB, $t_{-4}^{(-1)}$ will in general be different form $t_{-4}^{(-2)}$ in BNA. The strategy later aims to keep this difference small.

Cost of Backtracking

It is clear that backtracking requires $O(|p|n)$ operations, where p is the zero-band shift, so $\Sigma|p_i|$ must not be more than about $2n$ for a pivoting procedure to be viable. In the strategy described later, which caters for ill-conditioned leading submatrices, we aim to keep $\Sigma|p_i|\sim n$.

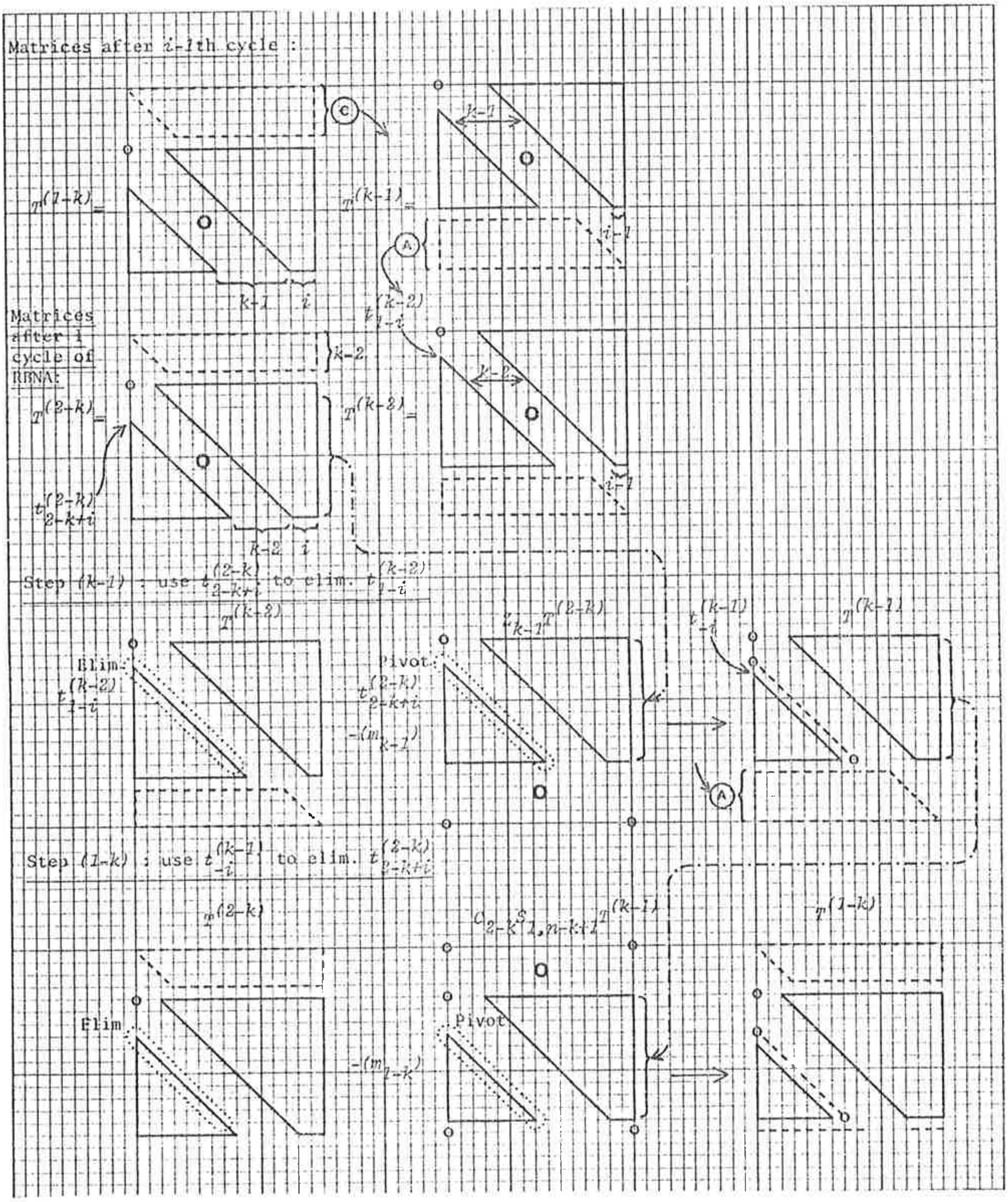


Fig. 2.8 - The ith cycle of BPC.
]

2.3 Restoration of Bareiss Form

We have discussed above two types of pivoting; the basic pivoting procedure, which allows a 2-way choice at each step, and backtracking, which allows the zero-band to be moved to enable other elements to be used as pivot. We noted above that both the basic pivoting procedure and backtracking result in the Toeplitz part of the iterates losing their Bareiss form. To illustrate, Fig. 2.9 shows Toeplitz blocks in Bareiss form and non-Bareiss form.

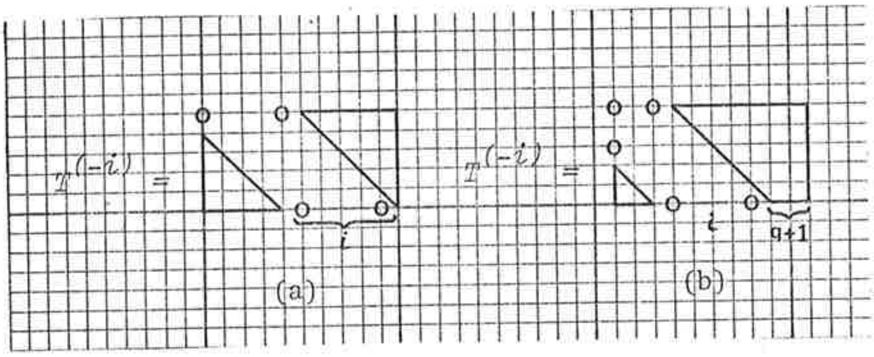


Fig. 2.9 - Toeplitz block with (a) Bareiss form
(b) non-Bareiss form

After a series of pivoting operations, we wish to convert the Toeplitz parts of the iterates back to Bareiss form to enable the Bareiss algorithm to be resumed. We call this operation restoration to Bareiss form (RBF). RBF can be performed by executing BPC to move the zero-band back to the Bareiss position, but a more economical method, requiring only half the number of operations per cycle, is to eliminate the diagonals in such a manner as to reduce the Toeplitz trapezoids to Toeplitz triangles. When the Toeplitz trapezoid is above the zero-band, as is shown in Fig. 2.9a, we can restore the Toeplitz block to Bareiss form by executing q A-cycles, where q is the modulus of the zero-band displacement (ZBD). ZBD = -2 in Fig. 2.9b. Each A-cycle raises the zero-band one place.

Similarly, if the Toeplitz trapezoid is below the zero-band, we can restore the Toeplitz block to Bareiss form by executing q C-cycles. In summary, we have

Procedure 2.7 - Restoration of Bareiss Form (RBF)

Input: $T^{(\pm i)}$, with zero-band displacement p.

Operations: If $p>0$, execute p C-cycles

else execute $|p|$ A-cycles.

Output: $T^{(\pm(i+|p|))}$, with no zero-band displacement.

Non-completion of RBF

It can be seen from the above procedure that if $n-i<|p|$, RBF will not run for $|p|$ cycles, and the resulting matrices will not be in Bareiss form, but will have non-Toeplitz trapezoids with base-lengths $|p|-n-i$ and $|p|-n-i+1$, which will require $O(|p|-n-i)^3$ operations to reduce. One method of overcoming this problem is to run RBF as long as possible (i.e. to cycle $n-1$) and then do the following procedure:

(i) augment the Toeplitz matrix, i.e. increase its order to $n+m$ and add m extra diagonals (say of zeros) above and below the main diagonal (fig. 2.10 shows a typical augmented Toeplitz matrix).

(ii) find the Toeplitz part of the augmented $T^{(1-n)}$ and $T^{(n-1)}$ using $M^{(1-n)}$ and $M^{(n-1)}$.

(iii) use BPC to move the zero-band back to its Bareiss position.

Note: There is no restriction on the displacement of the zero-band when BPC is used; however, recall that BPC takes twice as many operations per cycle as does RBF.

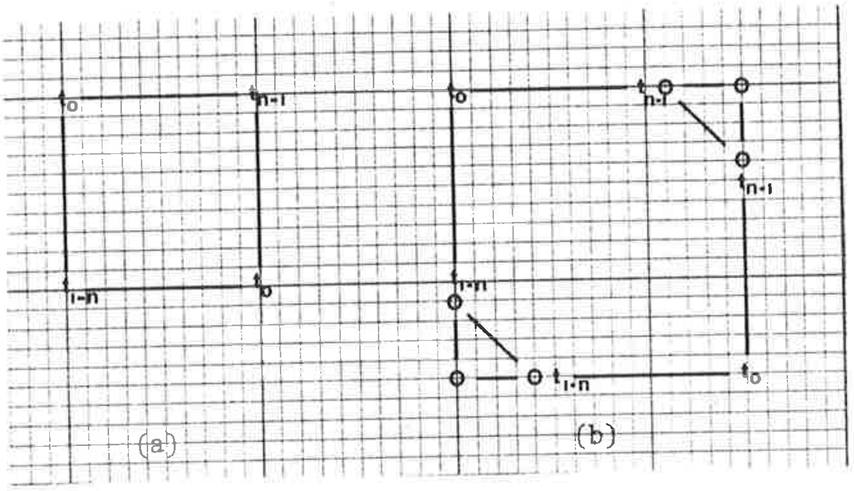


Fig. 2.10 - (a) Original Toeplitz Matrix
(b) Augmented Toeplitz Matrix

Solution of Reduced System of Equations

The pivoted BNA will not produce upper and lower triangles, as BNA does; we can, however, get a column-permuted upper triangle from the output of BNA as follows:

Procedure 2.8

(i) For normal BNA-cycles, C-cycles and BPC-cycles, save the first row of $T^{(-i)}$ which has i zeros in it.

(ii) For A-cycles, save the first row of both $T^{(-i)}$ and $T^{(i)}$ which has i zeros in it.

To see how to get a permuted upper-triangle, it can be checked that sequences of the four different pivoting procedures produce the following shapes when steps (i) and (ii) are carried out:

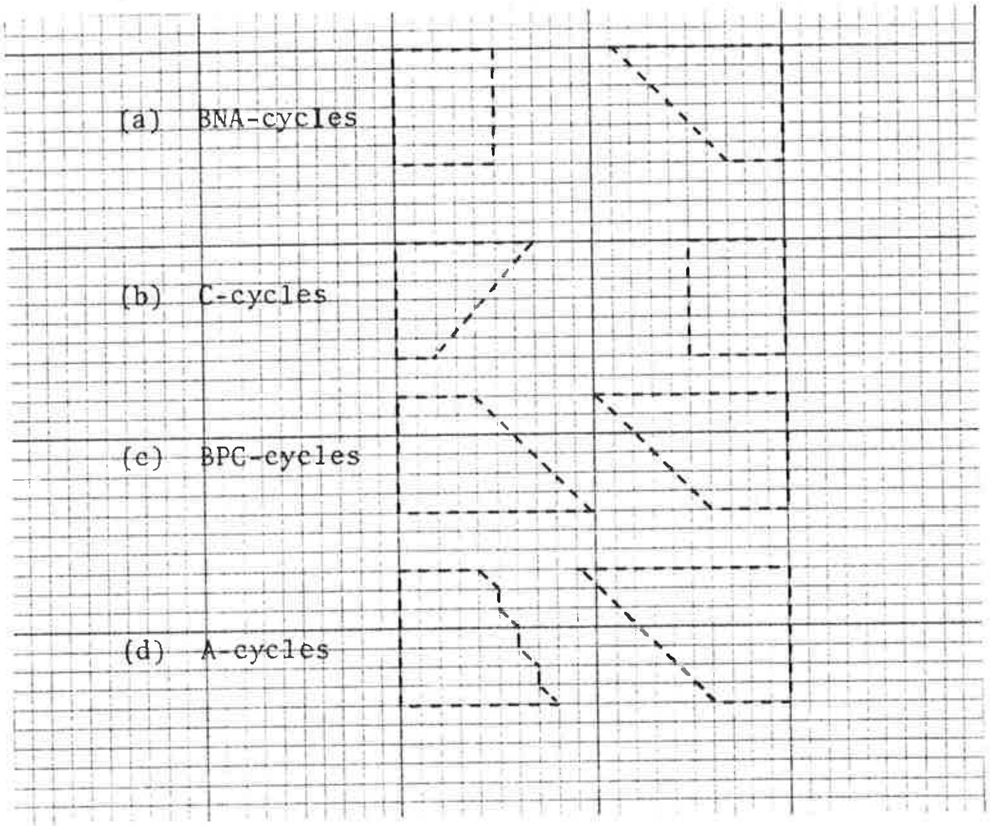


Fig. 2.11 - Shapes produced by various pivoting procedures

The complete matrix is a combination of these shapes, such as in Fig. 2.12:

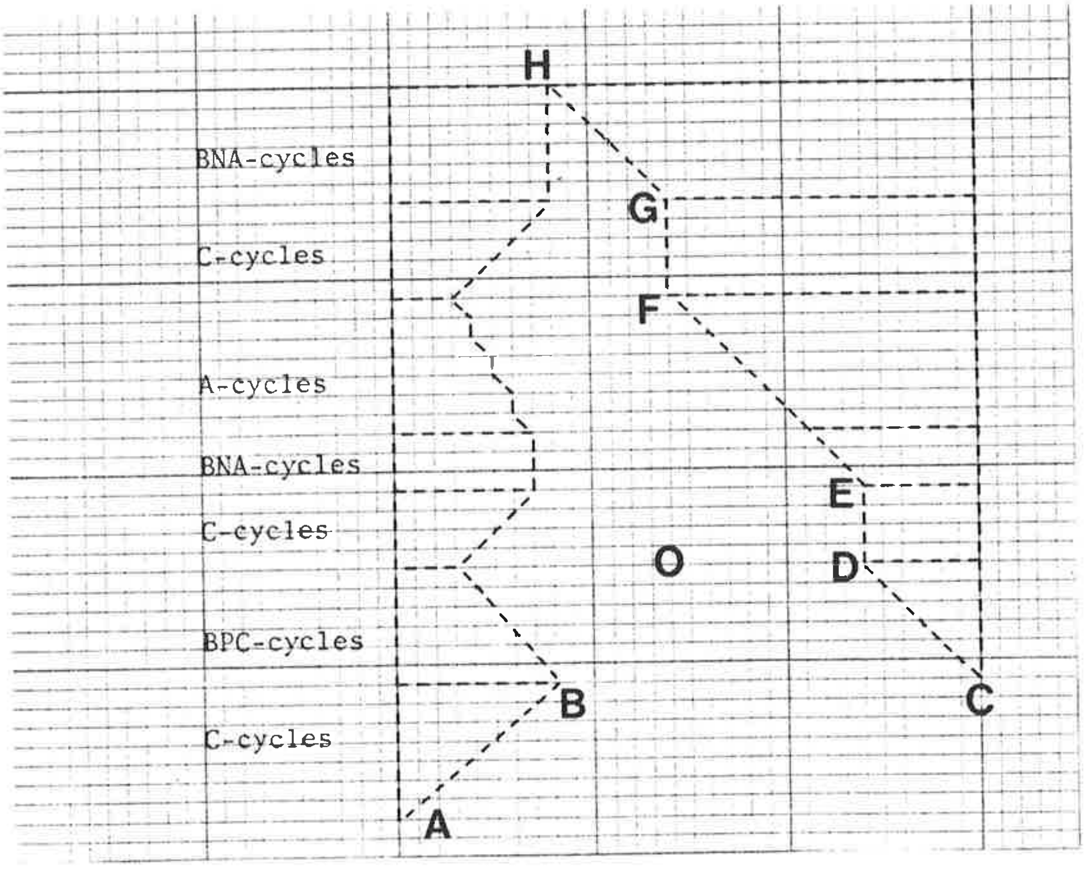


Fig. 2.12 - Typical matrix produced by Procedure 2.8

The n rows selected by taking rows labelled H to G, F to E, D to C and B to A are a column permuted UT. This can be seen by putting columns labelled A to B, C to D, E to F and G to H in columns n, $n-1,\ldots,1$ respectively.

2.4 Pivoting Operations - Summary

We have described three pivoting operations: the basic pivoting operation, backtracking and RBF; the last two operations, in fact are also made up from combinations of the basic pivoting operations (viz. A, C and R-cycles) executed according to a particular sequence. There is also a close relation between particular sequences of the basic pivoting procedure and algorithms for finding Padé approximants on a particular path through the Padé table. This relation will be examined later.

We now wish to incorporate pivoting operations in a strategy that gets the best possible numerical performance from the Bareiss algorithm while keeping the backtracking overheads to $O(n^2)$ operations. Such a problem has connexions with the problem of optimizing the path through the Padé table to the desired approximant, from a numerical point of view, but this is an interesting and separate study, and we will not got into it here. The strategy we will describe in the next subsection is not optimal, but is designed to handle cases where several leading submatrices are ill-conditioned; as we have seen above, BNA may suffer a serious loss of accuracy in these cases.

3. PIVOTING STRATEGIES

We describe a simple strategy which caters for the case when a single submatrix is ill-conditioned, and then some more general strategies for the case in which when several nested submatrices are ill-conditioned.

3.1 Simple Pivoting Strategy

According to theorem 4.2.3, the appearance of a small pivot $t_0^{(-k)}$ in BNA indicates that T_{k+1} is ill-conditioned, and by Theorem 4.3.1, the relative error of the elements of the computed matrices may increase by a factor of $|t_{max}^{(-k)}/t_0^{(-k)}|$, where $t_{max}^{(-k)} := \max_j |t_j^{(-k)}|$. The basic pivoting procedure allows the choice between two pivots, $t_0^{(-k)}$ and $t_{-k-1}^{(-k)}$, so if $t_0^{(-k)}$ is small compared to $t_{max}^{(-k)}$ but $t_{-k-1}^{(-k)}$ is not, selection of the latter as pivot might be expected to give better results than selection of the former, as done by BNA. After using $t_{-k-1}^{(-k)}$ as pivot in step (k) we should then eliminate $t_0^{(-k)}$ in step $(-k-1)$. This step should be quite accurate, as $t_0^{(-k)}$ is small and therefore the multiplier m_{-k-1} would normally be small. The above remarks suggest the use of the following pivot strategy:

Algorithm 3.1 - Pivoted Bareiss Algorithm, Strategy 1 (PBA-1)

1. Execute BNA, until for some k, $|t_0^{(-k)}/t_{max}^{(-k)}|<\delta$, where δ is some predetermined small quantity, or until $k=n-1$, in which case, exit.
2. If $|t_{-k-1}^{(-k)}| \leq |t_0^{(-k)}|$, we can do not better by selecting $t_{-k-1}^{(-k)}$, so continue with BNA; else do

   2.1 With $t_{-k-1}^{(-k)}$ as pivot, eliminate $t_0^{(k-1)}$

   2.2 With $t_k^{(k)}$ as pivot, eliminate $t_0^{(-k)}$

2.3 Restore Bareiss form by using $t_1^{(-k-1)}$ to eliminate $t_k^{(k)}$.

2.4 Go to 1.

PBA-1 is programmed on p.A1.

Selection of δ

If there is a-priori knowledge of the condition number of T_{k+1}, δ should be set such that pivoting occurs iff* T_{k+1} is ill-conditioned, e.g. suppose it is known that $cond\ T_{k+1} > M$ and $cond\ T_i < m$, $i \neq k+1$, $m << M$. Then, limited experience has shown that $t_0^{(-k)}$ is of the order of $(m/M)t_i^{(-k)}$ is $> \sim t_{max}^{(-k)}/m$. Hence, setting $\delta = 1/\sqrt{M}$, the geometric mean of these quantities, should maximize the chance that pivoting will occur iff necessary. In a test example, $cond\ T_k \sim 10^6$ and the other $\{T_i\}$ had condition number of 10 to 100, so we set $\delta = 10^{-3}$.

If there is no a-priori knowledge of condition number, δ should not be made too close to unity, otherwise the multiplier arising from step 2.2 will no longer be small; the best we can then do in step 2.2 is to select the larger of the two elements as pivot. This modified procedure, however will cause the zero-band, to be shifted from its normal 'Bareiss' position; and we observed above that if it is shifted too far (i.e. if the number of shifts, q, is greater than $n-k$), RBF cannot be completed, and the matrix must be augmented and BPC used. In this case, the strategy is much more involved, and the operation count is at least 50% greater (because of the need to accumulate the multipliers for BPC). We could alternatively limit the zero-band displacement to $n-k$. However, at that stage, there will be no further choice of pivots, so if any of these pivots are small, there may be a loss of accuracy. If we are willing to accept this, or reduce a non-Toeplitz matrix, as will be the case when $q > n-k$, we can select $\delta = 1$, i.e. we select the larger pivot each time.

*If and only if.

If there is no a-priori knowledge of condition number, but we wish to limit the increase in error-bound to a factor of L per step, we can proceed as follows. We will see later that the increase in error at each step is bounded by $\sim |t^{(\pm i)}_{max}/t^{(\pm i)}_{pivot}|$, so we should select the pivot satisfying $|t^{(\pm i)}_{max}/t^{(\pm i)}_{pivot}| \leq L$. It may happen that neither pivot satisfies this condition, but this is a limitation of the simple pivoting strategy.

Another way to proceed is to note that the total increase in error is bounded by $\sim \prod(t^{(\pm i)}_{max}/t^{(\pm i)}_{pivot})$, so if the desired accuracy is ε and the machine precision is μ, we should select each pivot $t^{(-i)}_{pivot}$ as follows: select $t^{(-i)}_{pivot}$ to move the zero-band back to the Bareiss position if

$$\prod_{j=1}^{i} (t^{(\pm j)}_{max}/t^{(\pm j)}_{pivot}) \leq (\varepsilon/\mu)^{i/n} \qquad (3.1)$$

otherwise, select the larger pivot. This strategy attempts to keep $\prod(t^{(\pm i)}_{max}/t^{(\pm i)}_{pivot}) \geq \mu/\varepsilon$ whilst minimizing the displacement of the zero-band: however, for n even of moderate size, the condition (3.1) will seldom be satisfied, and the strategy will select the larger pivot each time, so that there is a risk of RBF being unable to terminate. This is also a limitation of the simple pivoting procedure.

Effectiveness of the Simple Procedure

We remarked above that the simple procedure works well if $t^{(-k)}_{0}$ is small compared to $t^{(-k)}_{max}$, but $t^{(-k)}_{-k-1}$ is not. Theorem 4.2.3 states that, in this case, T_{k+1} is ill-conditioned, and that if T_{k+2} is well-conditioned, neither $t^{(-k)}_{-k-1}$ nor $t^{(-k)}_{1}$ will be small. Hence the simple procedure works well when a single submatrix is ill-conditioned.

The result on page A.7 bear this out. In the 6×6 input matrix, T_3 is badly-conditioned (it would be singular if $t_{31}={}^{71}/15$). When BNA is run, the error in the solution is 0.69, compared to a machine precision of 10^{-16}. When PBA-1 is run, the error is a modest multiple of the machine precision. When several of the $t_j^{(-k)}$ about $t_0^{(-k)}$, as well as $t_0^{(-k)}$, are small (e.g. this will occur when several leading submatrices are ill-conditioned), the simple procedure may not give good results. The more general strategy described below caters for this case.

3.2 General Pivoting Strategy

By Theorem 4.2.3, the appearance of several small diagonals $t_j^{(-k)}$, $j=0,\ldots,p; -k-1,\ldots,-k-q$ indicates that $\{\{T_{i;j}\}_{j=-q}^{p}\}_{i=k+1}^{k+1+q}$ are all ill-conditioned; Theorem 4.3.1 shows that for $j=0$, the relative error in the elements of the matrices arrived at by using $t_j^{(-k)}$ as pivot may increase by a factor of $|t_{max}^{(-k)}/t_j^{(-k)}|$ and we conjecture that this is also true for non-zero j. Hence, to reduce the loss of accuracy, we must choose pivots in such a manner as to avoid the small pivots that arise from the reduction of ill-conditioned submatrices of T in our reduction sequence.

3.2.1 Submatrix charts

Before we describe our pivoting strategy, it is instructive to illustrate the reduction sequences - the sequences of submatrices of T triangularized when BNA and the various pivoting procedures are executed. These reduction sequences can be shown on a submatrix chart, illustrated below;

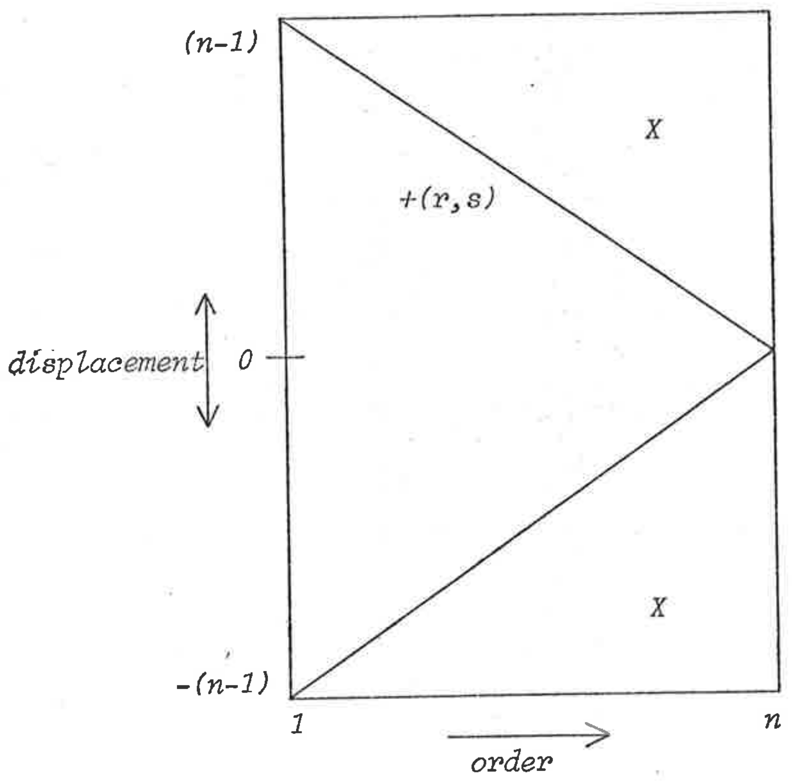


Fig. 3.1 Submatrix chart

A submatrix chart provides a framework for illustrating the n^2 unique contiguous submatrices that can be selected from a Toeplitz matrix. Numbers on the x-axis represents the order of the submatrix, and numbers on the y-axis, its displacement from the main diagonal. Thus the matrix in the (r,s) position is $T_{r;s}$. The areas denoted "X" are not feasible submatrices, since $|displacement| > (order\ of\ T) - 1$ in the area; such a matrix would include some elements outside the $n{\times}n$ Toeplitz matrix.

Fig. 3.2 shows the reduction sequence for BNA. In it, $T_1, T_2, \ldots, T_n$ are successively reduced.

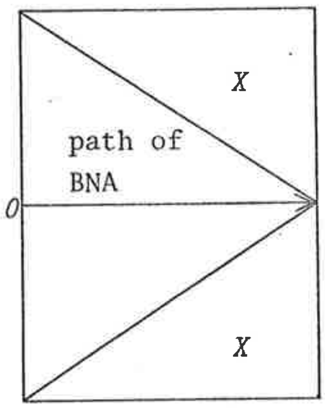


Fig. 3.2 Reduction sequence for BNA

Fig. 3.3 shows the possible paths for the basic pivoting procedure. Selecting both pivots above the zero band results in path A; path B results when there is one pivot above and one below; and path C results when both pivots are below. In fact, A and C cycles produce paths A and C respectively, and BNA cycles produce path B.

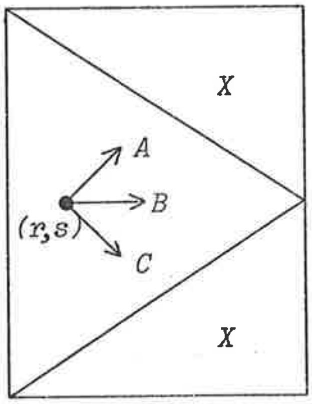


Fig. 3.3 Alternatives - basic pivoting procedure

Fig. 3.4 shows a typical path for Backtrack Procedure A. When the zero-band for $T^{(-1)}$ is in diagonal $(-q-1)$, submatrix $T_{2;-q}$ has been reduced.

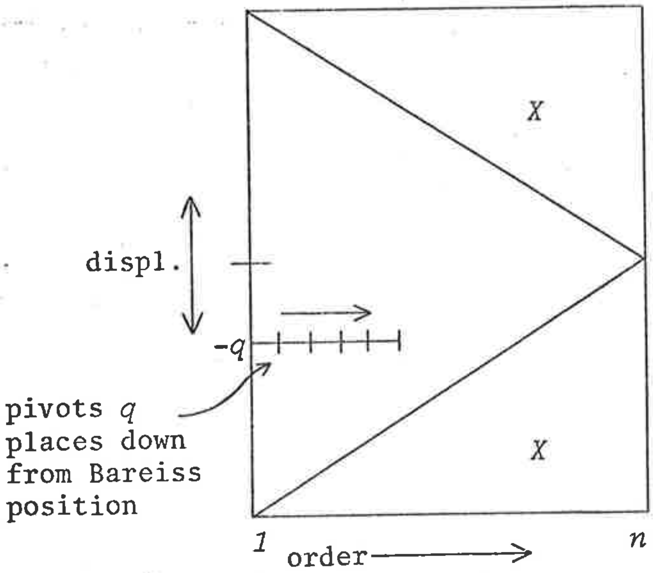


Fig. 3.4 Reduction sequence for BPA

Fig. 3.5 shows a typical path for BPB. To move the zero-band down q places, we go back q steps, and take the down-path for q cycles.

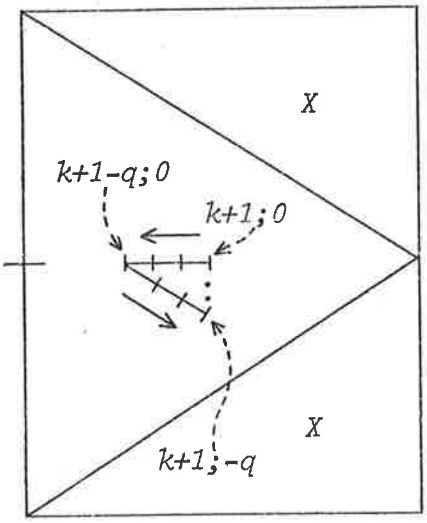


Fig. 3.5 Reduction sequence for BPB

Fig. 3.6 shows a typical path for BPC. To move the zero-band down q places, we go back one step then follow the sawtooth path shown; each "tooth" consists of a back-step (resulting from a cycle of RBNA) and a downward forward step (resulting from a cycle where both pivots are selected below the zero-band).

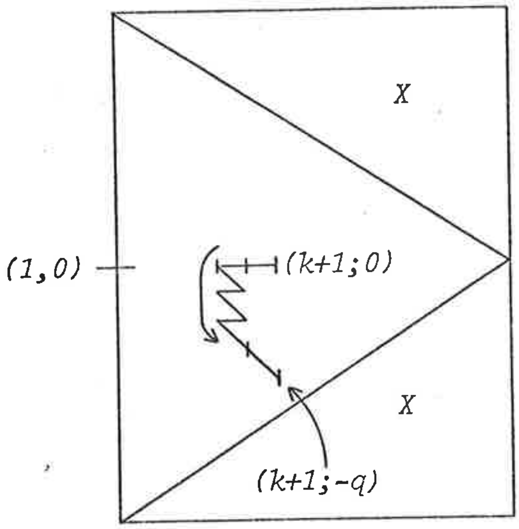


Fig. 3.6 Reduction sequence for BPC

Fig. 3.7 shows a path in the non-feasible region. This normally occurs if the zero-band is shifted too far in the pivoting. If the Toeplitz matrix is augmented (say, by diagonals, there will be a new, larger feasible region, which provides a wider choice in reaching $(n,0)$.

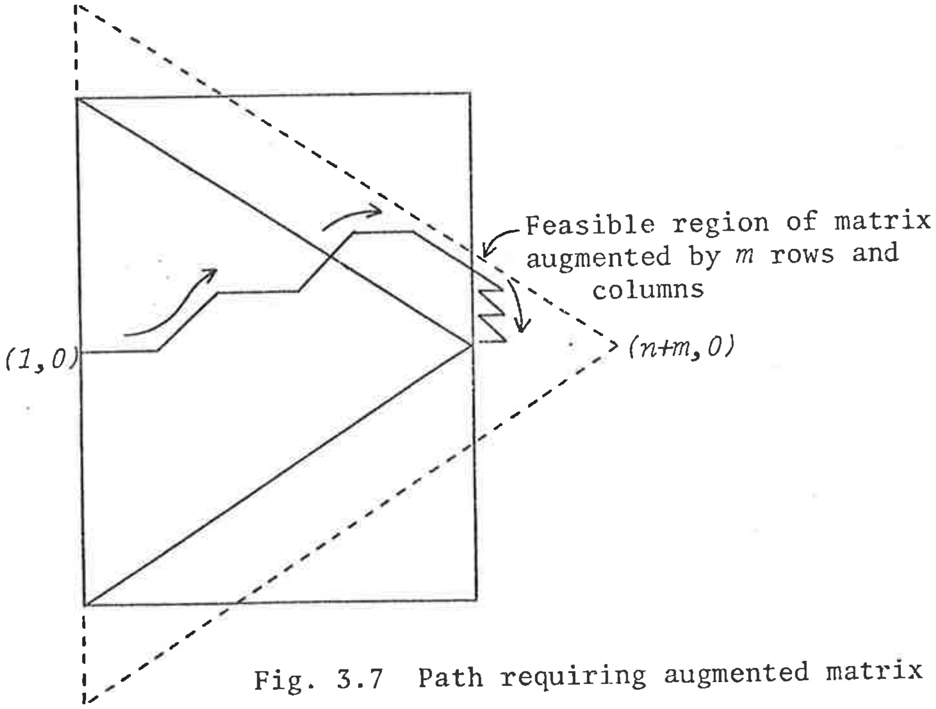


Fig. 3.7 Path requiring augmented matrix

Connexions with Padé Approximants

Bultheel [14] shows that the essential system to be solved in finding the $[i/j]$ Padé approximant is $T_{i;j}\underline{x} = \alpha\underline{e}_1$. Therefore, the reduction sequence in a submatrix diagram is equivalent to a corresponding path in the Padé table. Therefore, there is a close relationship between some of the reduction sequences derived above and algorithms in the literature on computing various paths in the Padé table. Bultheel [14] gives an excellent survey of the algorithms available, and shows that (i) row-paths compute the triangular factors of a Toeplitz matrix and (ii) diagonal and antidiagonal paths compute the triangular factors of a Hankel matrix Thus, the application of the BPP to calculate a row or diagonal path yields algorithms which are the same as Bultheel's; however, the basic pivoting procedure is an alternative way of looking at Padé paths, and essentially generalizes Bultheel's Toeplitz and Hankel algorithms - both are special cases of particular sequences of the basic pivoting procedure. We can apply this observation by noting that the path AB in Fig. 3.8 produces the factors of a Hankel matrix with principal submatrices $T_{1;n-1}, T_{2;n-2},\ldots,T_{n;0}$.

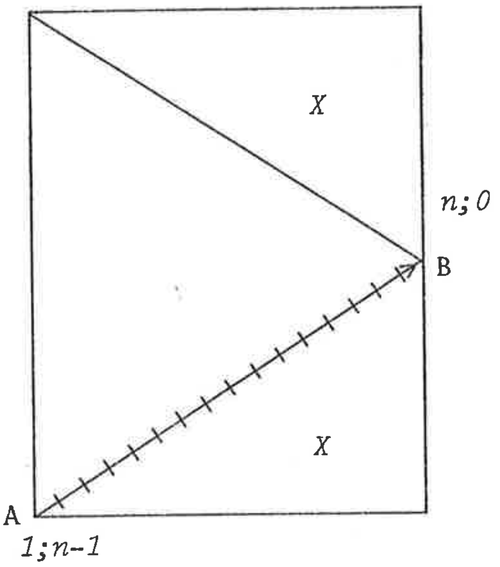


Fig. 3.8 Path to factorize a Hankel matrix

*Basic pivoting procedure.

We also believe that the concept introduced in BPC, i.e. reversing the Bareiss algorithm, is new. There is one related reference in the Toeplitz literature; Huang and Cline [47] proposes an algorithm to reverse the Trench algorithm, but [47] makes no mention of the related concept of going backward in the Padé table, which is in fact what RBNA does, as is shown in Fig. 3.9(a). Pivoting can also be introduced easily into RBNA (by reversing the basic pivoting procedure) so that as well as going left in the submatrix (or Padé) table, we can also go back along the diagonals or anti-diagonals, as shown in Fig. 3.9(b).

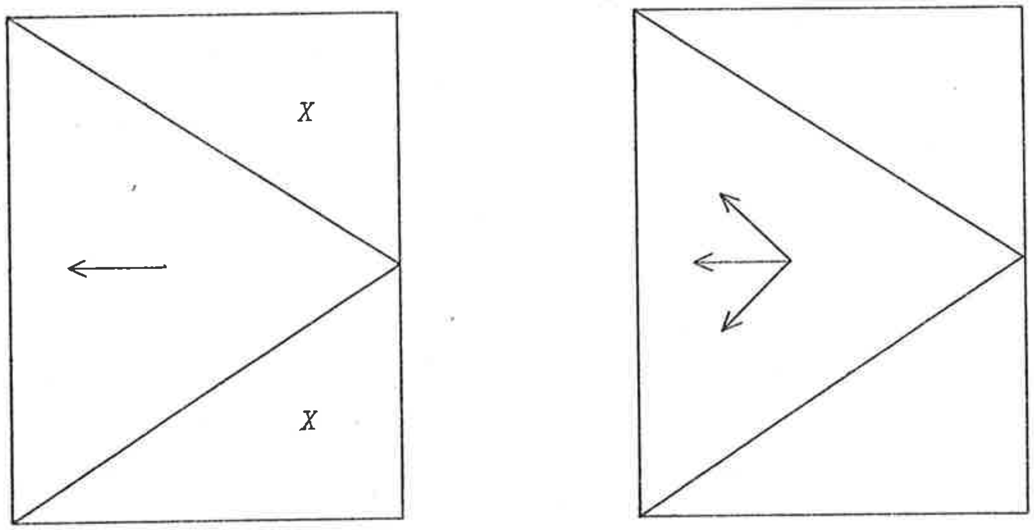


Fig. 3.9(a) Path of RBNA

(b) Possible paths of pivoted RBNA

3.2.2 Strategy Incorporating BPB, BNA and RBF

We now consider a strategy which aims to avoid ill-conditioned submatrices in the reduction sequence. Fig. 3.10 illustrates what to do in the case where $t_{-k-q}^{(-k)},\ldots,t_{p}^{(-k)}$ all small, (implying that all submatrices with order $k+1$ to $k+2+p+q$ and displacement $-q$ to p are all ill-conditioned).

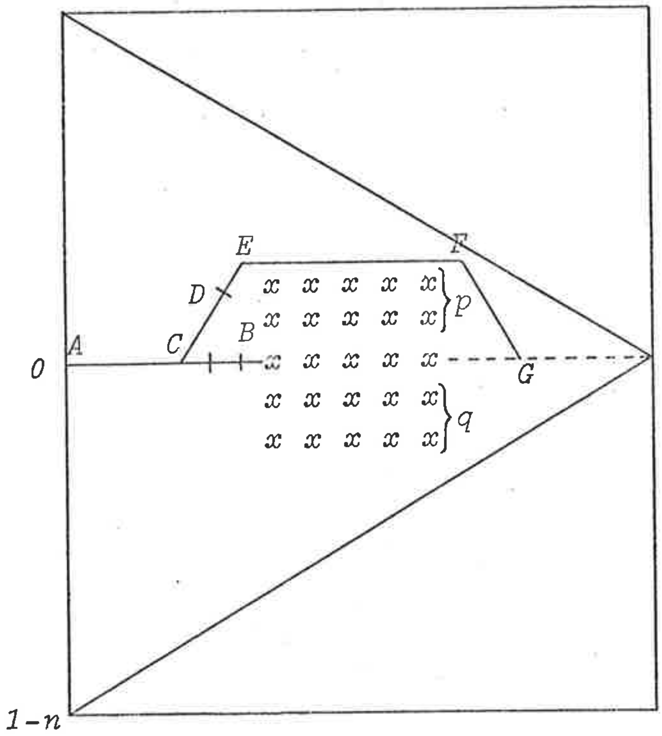


Fig. 3.10 Reduction path around an ill-conditioned block

In terms of the pivoting procedures, what happens is as follows. We execute the normal BNA (path AB) until the ratio $|t_0^{(-k)}/t_{max}^{(-k)}|$ is smaller than a preset value, δ (we leave discussion of the selection of δ until later). This indicates that $cond\ T_{k+1} > \|T_{k+1}\|/(\delta t_{max}^{(-k)})$. We then look at the pivots on either side of $t_0^{(-k)}$. If $t_{-k-q}^{(-k)},\ldots,t_p^{(-k)}$ are also less than δ, then all the matrices illustrated by x's are ill-conditioned, and we must avoid using pivots from the bottoms of the lower-triangles derived from these matrices. In Fig. 3.10, we used BPB (path BCD) to slide the zero-band up p places to make $t_p^{(-k)}$ available for pivoting. If the block of x's were closer to the left boundary, and $q < k$, we would use BPA instead to get to point D. We then use $t_p^{(-k)}$ as pivot (point E), and continue to do so for another $p + q$ cycles (path EF), then restore to Bareiss form. This pivoting strategy can be summarized as follows:

Algorithm 3.2 - Pivoted Bareiss Algorithm, Strategy 2 (PBA-2);

Using BPB for Backtracking and RBF

1. Execute BNA, until for some k, $|t_0^{(-k)}/t_{max}^{(-k)}| < \delta$, or until $k=n-1$.

   If $k=n-1$, exit, else do:

2. Find p such that $|t_p^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}, |t_j^{(-k)}| < \delta$, $j=0,\ldots,p-1$

   Find q such that $|t_{-k-1-q}^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$, $|t_j^{(-k)}| < \delta$, $j=-k-1,\ldots,-k-q$

3. If $p \leq q+1$, move the zero-band up p places to make $t_p^{(-k)}$ available for pivoting, else move the zero-band down $q+1$ places to make $t_{-k-q-1}^{(-k)}$ available for pivoting (see procedure 2.3, BPB).

4. Do $p+q$ BNA-cycles

5. Execute RBF.

6. Go to 1.

PBA-2 is programmed on p.A.8.

Problems with PBA-2

PBA-2 works well if there is a block of ill-conditioned submatrices and all other submatrices are well-conditioned. If there is a wide range of condition numbers, or if there are several blocks* close together, the situation is less clear-cut.

Problem 1 - Pivot drift

The appearance of several small pivots $t_{-k-q}^{(-k)},\ldots,t_p^{(-k)}$ indicate that $T_{k+1;-q},\ldots,T_{k+1;p}$ are all ill-conditioned. However, the converse does not necessarily hold, i.e. if $t_p^{(-k)}$, the chosen pivot is not small, $T_{k+1;p+1}$ (at point "E" on Fig. 3.10) is not necessarily well-conditioned. If it is ill-conditioned, this will be revealed after BPB is performed - the new $t_p^{(-k)}$ will be small and possibly unsuitable as a pivot i.e. $< \delta \cdot t_{max}^{(-k)}$. This change in $t_p^{(-k)}$ we call pivot drift. There are several

*Blocks of ill-conditioned submatrices on the submatrix diagram.

ways we can try to overcome this problem. One method is to search until $t_p^{(-k)}$ is considerably larger than all of the elements between $t_p^{(-k)}$ and the zero-band. The following Lemma then shows that, in general, the ratio $t_p^{(-k)}/t_{max}^{(-k)}$ will not be changed much after backtracking.

Lemma 3.1 Let $t_j^{(-k)}$ be the output of step $(-k)$ of BNA, and let $\tilde{t}_j^{(-k)}$ be the output of any pivoting procedure which moves the zero-band to displacement p, so that

$$\tilde{t}_0^{(-k)}=\tilde{t}_1^{(-k)}=\ldots=\tilde{t}_{p-1}^{(-k)}=0,\ p>0,\ \text{or}\ \tilde{t}_{-k-1}^{(-k)}=\tilde{t}_{-k-2}^{(-k)}=\ldots=\tilde{t}_{-k+p}^{(-k)}=0,\ p<0.$$

Let $j(max)$ be such that $t_{max}^{(-k)}=|t_{j(max)}^{(-k)}|$. Then

$$\frac{\tilde{t}_j^{(-k)}}{\tilde{t}_{j(max)}^{(-k)}} = \frac{t_j^{(-k)}}{t_{j(max)}^{(-k)}} \quad \frac{(1+\varepsilon_j)}{(1+\varepsilon_{j(max)})}, \tag{3.2}$$

where for any i,

$$\varepsilon_i \le \begin{cases} \beta_p \; condT_{k;p} \; \| (\underset{-}{0}^T, t_0^{(-k)}, \ldots, t_{p-1}^{(-k)})/t_i^{(-k)} \|, & p>0 \\ \beta_p \; condT_{k;p} \| (\tilde{t}_{-k+p}^{(-k)}, \tilde{t}_{-k+p+1}^{(-k)}, \ldots, \tilde{t}_{-k-1}^{(-k)}, \underset{-}{0}^T)^T / t_i^{(-k)} \|, & p<0 \end{cases}$$

and $\beta_p := \|T\|/\|T_{k;p}\|$

Proof: We prove the Lemma only for the case $p>0$. The proof for the case $p<0$ is similar. It was shown by Bareiss, that in BNA, $\underset{-}{t}_{l\cdot}^{(-k)}$ is a linear combination of $\underset{-}{t}_{l-k\cdot}, \ldots, \underset{-}{t}_{l\cdot}$, and that the coefficient of $\underset{-}{t}_{l\cdot}$ in the linear combination is unity. Hence for some $\{\alpha_i\}$ and $k+1 \le l \le n$,

$$\underline{t}^{(-k)}_{l\cdot} = \sum_{i=1}^{k} \alpha_i \underline{t}_{i-l-1-k+i\cdot} + \underline{t}_{k+1\cdot} \xrightarrow{l=k+1} \underline{t}^{(-k)}_{k+1,1+p;k+p} =$$

$$= \sum_{i=1}^{k} \alpha_i \underline{t}_{i,1+p:k+p} + \underline{t}_{k+1,1+p;k+p} \qquad (3.3a,b)$$

Similarly, for any backtrack procedure $\exists \{\tilde{\alpha}_i\}$ not all zero such that for $k+1 \le l \le n$,

$$\underline{\tilde{t}}^{(-k)}_{l\cdot} = \sum_{i=1}^{k} \tilde{\alpha}_i \underline{t}_{l-1-k+i\cdot} + \tilde{\alpha}_{k+1} \underline{t}_{l\cdot} \xrightarrow{l=k+1} \underline{\tilde{t}}^{(-k)}_{k+1,1+p:k+p} =$$

$$= \sum_{i=1}^{k} \tilde{\alpha}_i \underline{t}_{i,1+p:k+p} + \tilde{\alpha}_{k+1} \underline{t}_{k+1,1+p:k+p}. \qquad (3.4a,b)$$

Now $\underline{\tilde{t}}^{(-k)}_{k+1,1+p:k+p} = \underline{0}^T$ and $\tilde{\alpha}_{k+1} \ne 0$, because since $T_{k;p} := T_{1:k,1+p:k+p}$ is nonsingular, the summation term in (3.4b) is non-zero. Hence (3.4b) can be written

$$\underline{0}^T = \sum_{i=1}^{k} \alpha'_i \underline{t}_{i,1+p:k+p} + \underline{t}_{k+1,1+p:k+p},$$

$$\text{where} \quad \alpha'_i := \tilde{\alpha}_i / \tilde{\alpha}_{k+1} \qquad (3.5a,b)$$

Subtracting (3.5a) from (3.3b),

$$\underline{g}^T := \underline{t}^{(-k)}_{k+1,1+p:k+p} = \sum_{i=1}^{k} (\alpha_i - \alpha'_i) \underline{t}_{i,1+p:k+p}$$

$$= \underline{\delta\alpha}^T T_{k;p}, \text{ where } (\underline{\delta\alpha})_i := \alpha_i - \alpha'_i$$

$$\therefore \quad \underline{\delta\alpha}^T = \underline{g}^T T^{-1}_{k;p} \qquad (3.6)$$

Subtracting (3.3a) from $\tilde{\alpha}_{k+1}^{-1}$ × eq.(3.4a), and taking the mth component, we get

$\tilde{\alpha}_{k+1}^{-1}\,\tilde{t}_{lm}^{(-k)} - t_{lm}^{(-k)} = -\underline{\delta\alpha}^{T}(t_{l-k,m},\ldots,t_{l-1,m})$, which

with (3.6) $= -\underline{g}^{T}T_{k;p}^{-1}\,(t_{l-k,m},\ldots,t_{l-1,m})$ (3.7)

Replacing $\tilde{t}_{lm}^{(-k)}$ by $\tilde{t}_{j}^{(-k)}$, etc, where $j=m-l$, (3.7) becomes

$$\tilde{\alpha}_{k+1}^{-1}\,\tilde{t}_{j}^{(-k)} - t_{j}^{(-k)} = \varepsilon_j t_j^{(-k)},$$

where $\varepsilon_j = -\underline{g}^{T}T_{k;p}^{-1}(t_{l-k,m},\ldots,t_{l-1,m})/t_j^{(-k)}$. (3.8a,b)

Taking norms and using the definition of $\underline{g}^{T}$

(3.8) ⇒ $|\varepsilon_j| \le \beta_p\ \mathit{cond}T_{k;p}\|(\underline{0}^{T},t_0^{(-k)},\ldots,t_{p-1}^{(-k)})/t_j^{(-k)}\|$ (3.9)

where $\beta_p := \|T\|/\|T_{k;p}\|$.

Let $j(max)$ be such that $t_{max}^{(-k)} = |t_{j(max)}^{(-k)}|$. Then (3.2) follows from (3.8a). QED.

Remarks

We see from the above Lemma that if $\{t_j^{(-k)}\}_0^{p-1}$ are small compared to $t_p^{(-k)}$, then the ratio $\tilde{t}_p^{(-k)}$ $\tilde{t}_{max}^{(-k)}$ will be little changed from $t_j^{(-k)}/t_{max}^{(-k)}$ unless $T_{k;p}$ is badly-conditioned. In the absence of any knowledge of $\mathit{cond}T_{k;p}$, a reasonable way to proceed is to find p such that $|t_p^{(-k)}|\ge K_1 t_i^{(-k)}$, $i=1,\ldots,p-1$ (where K_1 is some large number) as well as being $\ge \delta\cdot t_{max}^{(-k)}$. Alternatively, we could simply search until $|t_p^{(-k)}|\ge K_2\delta\cdot t_{max}^{(-k)}$, where K_2 is some constant > 1. Then, assuming arbitrary pivot drift with a uniform probability density function, the probability that the new $|t_p^{(-k)}|\le \delta\cdot t_{max}^{(-k)}$ is $\le 1/K_2$, in the absence of any knowledge of $cond\ T_{k;p}$. If we use one of these strategies and the new pivot is still $< \delta \cdot t_{max}^{(-k)}$, we must use BPC to move the zero-band down until $|t_p^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$ (note that we wish to use BPB whenever possible since it is more efficient than BPC).

Problem 2

Running into ill-conditioned blocks during backtracking and RBF. Fig. 3.11(a) and (b) show these respective situations:

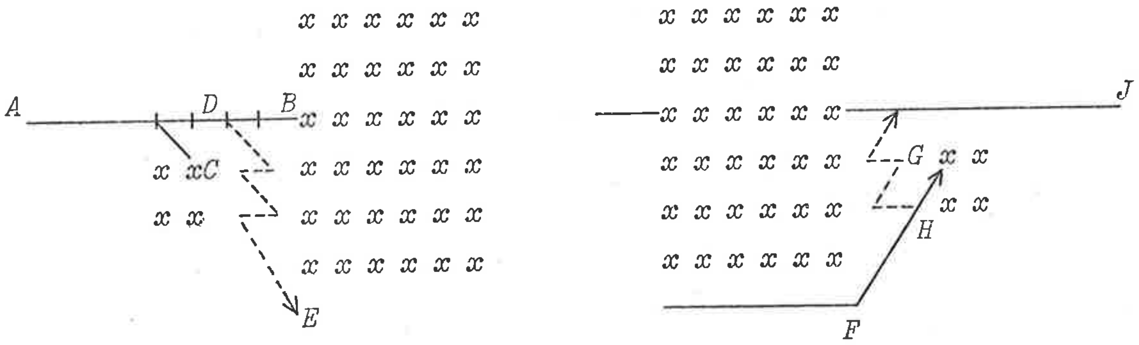


Fig. 3.11 - Running into ill-conditioned blocks on (a) BPB (b) RBF

In fig. 3.11(a), AB is the Bareiss path and BC is the path* of the attempted BPC. To avoid the block at C (evidenced by a small pivot) we can use BPC instead of BPB, following the path DE. We could also use a hybrid strategy, using BPB wherever possible (because it is faster than BPC) and using BPC to avoid any blocks encountered.

Fig. 3.11(b) shows FG, the attempted RBF path. To avoid the block at G, we could use BPC instead of RBF, following the path FH. Alternatively, we could use a hybrid strategy as before.

*Note that the previous iterates $T^{(\pm i)}$ should be stored for use by BPB. To store $n-1$ iterates requires $2n^2$ locations.

3.2.3 Strategies incorporating BPB, BPC, BNA and RBF

We now present some modified strategies in view of the discussion above.

Algorithm 3.3 - Pivoted Bareiss Algorithm, strategy 3 (PBA-3):

Using BPC for Backtracking and RBF (Programmed on p.A.21).

1. Execute BNA, until for some k, $|t_0^{(-k)}/t_{max}^{(-k)}| < \delta$, or until $k=n-1$.

   If $k=n-1$ exit, otherwise do

2. Find p such that $|t_p^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$, $|t_j^{(-k)}| < \delta$, $j=0,\dots,p-1$

   Find q such that $|t_{-k-1-q}^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$, $|t_j^{(-k)}| < \delta$, $j=-k-1,\dots,-k-q$

3. If $p \leq q+1$, move the zero-band up using BPC, until the available pivot, $t_p^{(-k)}$, satisfies $|t_{p'}^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$ (Note that p' may be greater than p), else move the zero-band down using BPC until the available pivot, $t_{-k-1-q'}^{(-k)}$, satisfies $|t_{-k-1-q'}^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$ (Note that q' may be $\geq q$).

4. If $p \leq q+1$, set $s=p'+q$ otherwise set $s=p+q'$. s is the new block-size. Skirt the block by executing s BNA-cycles.

5. Restore to Bareiss form using BPC.

6. Go to 1.

Algorithm 3.4 - Pivoted Bareiss Algorithm, strategy 4 (PBA-4):

Using a hybrid strategy for backtracking and RBF

1. Execute BNA, until for some k, $|t_0^{(-k)}/t_{max}^{(-k)}| < \delta$, or until $k=n-1$.

   If $k=n-1$, exit, otherwise do

2. Find p such that $|t_p^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$ and $|t_p^{(-k)}| \geq K_1 max\{|t_j^{(-k)}|\}_0^{p-1}$;

   Find q such that $|t_{-k-1-q}^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$ and $|t_{-k-1-q}^{(-k)}| \geq K_1\ max\{|t_{-k-j}^{(-k)}|\}_1^q$

3. (a) If $p \leq q{+}1$, move the zero-band up p places using BPB. If any block is encountered (as evidenced by a pivot $< \delta t_{max}^{(-i)}$), go around it using BPC. At the end of the hybrid procedure, check if $|t_p^{(-k)}| \geq \delta t_{max}^{(-k)}$. If not, slide the zero-band up further using BPC, until a p' is found such that $|t_{p'}^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$. If this does not occur, select p' to maximize $|t_{p'}^{(-k)}/t_{max}^{(-k)}|$— in this case, T is ill-conditioned.

   (b) As for (a), but searching for a q' <u>below</u> the zero-band such that $|t_{-k-1-q'}^{(-k)}| \geq \delta \cdot t_{max}^{(-k)}$.

4. If $p \leq q{+}1$, set $s{=}p'{+}q$, otherwise set $s{=}p{+}q'$. s is the new block-size. Skirt the block by executing s BNA-cycles.

5. Execute RBF. If any block is encountered (as evidenced by a pivot $< \delta t_{max}^{(-i)}$ in magnitude), go around it using BPC.

6. Go to 1.

<u>Algorithm 3.4a</u> - Pivoted Bareiss Algorithm, Strategy 4a (PBA-4a): modified pivot selection strategy

Same as for PBA-4, except for step 2, which is replaced by:

2. Find p such that $|t_p^{(-k)}| \geq K_2 \delta \cdot t_{max}^{(-k)}$, $|t_j^{(-k)}| < K_2 \delta \cdot t_{max}^{(-k)}$, $j{=}0,\ldots,p{-}1$

   Find q such that $|t_{-k-1-q}^{(-k)}| \geq K_2\ \delta t_{max}^{(-k)}$, $|t_j^{(-k)}| < K_2\ \delta t_{max}^{(-k)}$, $j{=}{-}k{-}1,\ldots,{-}k{-}p$

<u>Selection of δ</u>

If there is a-priori knowledge of the condition numbers of the submatrices of T, δ should be set such that pivoting occurs if an ill-conditioned submatrix is encountered, e.g. suppose it is known that a few submatrices have condition number greater than M, and the rest have condition numbers less than m, where $m{<<}M$. Then, limited experience has shown that during BNA, the ill-conditioned blocks will be evidenced by pivots $t_j^{(-k)}$ of order $\frac{m}{M}t_{max}^{(-k)}$, and the well-conditioned blocks will be evidenced by pivots > about $t_{max}^{(-k)}/m$. Hence, setting

$$\delta = 1/\sqrt{M} \tag{3.10}$$

the geometric mean of these quantities, should maximise the chance that pivoting will occur iff necessary. In a test example, some submatrices had a condition number of 10^4-10^6 and the rest had condition numbers of 10 to 100, so we set $\delta = 10^{-3}$. One case of interest where there is a-priori knowledge is the non-normal Padé problem, where some submatrices are numerically singular. In this case, $M = O(\mu^{-1})$, where μ^{-1} is the machine precision. A reasonable value for δ is $\mu^{\frac{1}{2}}$.

If there is no knowledge of condition number but we wish to limit the increase in relative error per step to some quantity ξ, we should, as for PBA-1, find a pivot satisfying $|t_{pivot}^{(i)}/t_{max}^{(i)}| \geq 1/\xi$, instead of specifying a fixed value for δ. (Recall our conjecture at the beginning of subsection 3.2 that the increase in relative error is bounded by a factor of $|t_{max}^{(i)}/t_{pivot}^{(i)}|$). As for the simple pivoting strategy, the total increase in relative error is then bounded by a factor of $\prod(t_{max}^{(\pm i)}/t_{pivot}^{(\pm i)})$. We should therefore select each pivot $t_{pivot}^{(i)}$ to satisfy

$$\prod_{j=1}^{i} (t_{pivot}^{(\pm j)}/t_{max}^{(\pm j)}) \geq (\mu/\varepsilon)^{(i/n)} \tag{3.11}$$

With this strategy and for n of even moderate size, the maximum allowable value of $|t_{max}^{(i)}/t_{pivot}^{(i)}|$ is about unity, and this will result in extensive backtracking; in the worst case, the algorithm will search most of the available pivots at every stage, resulting in an $O(n^3)$ operation count. To keep the $O(n^2)$ complexity, we could limit the number of pivots searched to some (small) integer l, but occasionally some very small pivots that violate the condition (3.11) will then be used. There is essentially a tradeoff between accuracy and complexity, but the exact nature of the tradeoff depends on the particular problem, and much more work is needed on this topic.

Selection of K_1

This is required in step 2 of PBA-4a. Again, if we have some a-priori knowledge of the condition numbers of the submatrices, we can use Lemma 3.1 to select K_1 to ensure the pivot drift is less than a given quantity α. Usually, we have no such a-priori knowledge, so we must assume $T_{k;q}$ is well-conditioned (say $cond\ T_{k;q} \sim k$) and select K_1 accordingly.

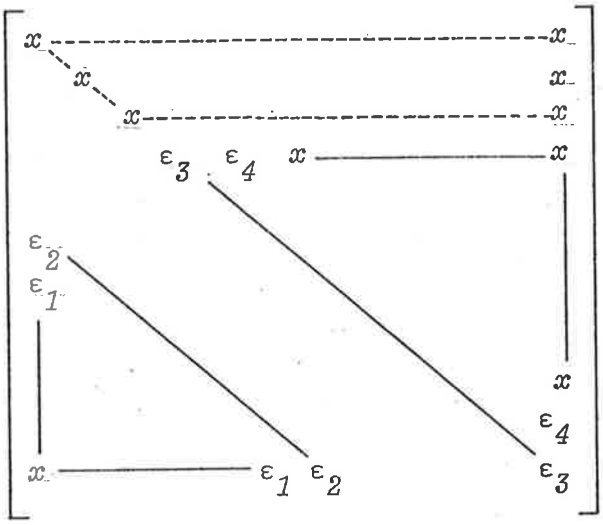
Selection of K_2

This is required at step 2 of PBA-4b. In this case, we assume that if $|t_p^{(-k)}| \geq K_2 \delta \cdot t_{max}^{(-k)}$, the probability of it drifting to be less than $\delta \cdot t_{max}^{(-k)}$ is $\frac{1}{K_2}$. Hence K_2 need only be a moderate integer, say 10, for $|t_p^{(-k)}|$ to remain $\geq \delta \cdot t_{max}^{(-k)}$ after backtracking in most cases. Where this is not so, we then simply use BPC to find an appropriate pivot.

Results

The 13×13 Toeplitz matrix on p.A.19 has $A_{1:4,1:4}$, $A_{2:5,1:4}$, $A_{3:6,1:4}$ and $A_{1:4,2:5}$, all ill-conditioned, a change of only 2×10^{-3} being required to make the singular; thus $A_4, \ldots, A_7$ are also ill-conditioned. Execution of BNA produces at step (-3) the form

$T^{(-3)} =$



where $|\varepsilon_i| < 10^{-4} t_{max}^{(-3)}$. The machine precision is 10^{-16}, and the results on p.A.19 shows that the relative error in the solution of $T\underline{x}=\underline{b}$ is 0.026. Using PBA-2 and PBA-3, the relative errors in the solution of $T\underline{x}=\underline{b}$ are 4×10^{-14} and 12×10^{-14} respectively, still moderate multiples of the machine precision.

4. INCREASE IN ERROR BOUND - SIMPLE PIVOTING STRATEGY

We show in the next theorem that if T_{k+1} is badly-conditioned, but $T_{k;1}$ and $T_{k;-1}$ are well-conditioned, the increase in error bound will not be large if PBA-1 is used, whereas Theorem 4.3.1 shows that when BNA is used, the error bound increases by a factor proportional to $cond\ T_{k+1}$.

Theorem 4.1 Let ε, R, R_1, μ, β', $\bar{T}_*^{(-k)}$, $\bar{T}_*^{(k-1)}$ be as in Theorem 4.3.1. Also, let $\gamma := |t_{-k-1}^{(-k)}/t_{max}^{(-k)}|$ and assume *that $\gamma\neq 0$. Then if PBA-1 is executed, and at cycle k, $t_{-k-1}^{(-k)}$ is selected as pivot,

*Otherwise PBA-1 fails

(i) $$**rel\ \bar{T}_*^{(k)} \le \begin{cases} \mu R_4\ \gamma^{-1}(1 + \frac{R\beta}{R_1}\gamma^2\ cond\ T_k)/(1-\beta\ \gamma\ cond\ T_k), \gamma < {}^1/_\beta\ condT_k \\ \mu\beta' R_3 cond\ T_k^E/\gamma^2,\ \gamma \ge {}^1/_\beta\ cond\ T_k\ , \end{cases}$$

where $R_4 := R(1 + \delta)$, $R_3 := R_4(1 + \gamma^2\ R/R_4)$

(ii) $rel\ \bar{T}^{(-k-1)} \le \mu R_8 \gamma^{-1} cond\ T_k^E$,

where $R_8 := R_7/\{(1-\beta' L^{-1} condT_k^E(1 + \gamma^{-1})\}$, $R_7 := R_6(1 + \gamma/cond\ T_k^E)$,

$R_6 := \beta' R_5(1 + \gamma)$, $R_5 := R_1(1 + rel\ \bar{T}_*^{(k)}/(\mu R_1 L))$

Proof: By following the errorsthrough as in Theorem 4.3.1, we get, in analogy to the equation after (4.3.10).

$$|\delta t_j^{(k)}| \le \mu\{Rt_{max}^{(k-1)} + |m_k t_{max}^{(-k)}|R_4/\gamma\},\text{ where } R_4 = R(1 + \gamma) \tag{4.1}$$

It is easily shown, in analogy to eq. (4.3.8), that

$$t_{max}^{(k-1)} \le \beta|t_0|\ cond\ T_k \tag{4.2}$$

Continuing the analogy to Theorem 4.3.1, and using (4.2), we get (cf. (4.3.13))

$$rel\ \bar{T}_*^{(k)} \le \mu\gamma^{-1}\ R_2\ ,$$

$$\text{where } R_2 = R_4(1 + \frac{R\beta}{R_1}\gamma^2\ condT_k)/(1 - \beta\ \gamma\ condT_k). \tag{4.3}$$

This bound cannot be used if $\gamma \ge {}^1/_\beta\ condT_k$. An alternative bound can be derived by using

$$t_{max}^{(k-1)} \le \beta'|t_k^{(k-1)}|cond\ T_k^E \tag{4.3.8}$$

and $|t_0| \le \beta'|t_k^{(k-1)}|\ cond\ T_k^E$ (a special case of 4.3.8) (4.4)

**Defined as in Theorem 4.3.1.

in (4.1), where

$$|\delta t_j^{(k)}| \leq \mu(R\beta'|t_k^{(k-1)}|\mathit{cond}\ T_k^E + R_4\beta'|t_k^{(k-1)}|/\mathit{cond}T_k^E/\gamma^2 \quad (4.5)$$

But $|t_{max}^{(k)}| \geq |t_k^{(k)}| = |t_k^{(k-1)}|$, so (4.6)

$$\mathit{rel}\ \bar{T}_*^{(k)} \leq \mu\beta'\mathit{cond}\ T_k^E(R + R_4/\gamma^2) = \mu\beta'\mathit{cond}T_k^E\ R_3/\gamma^2$$

where $R_3 := R_4(1 + \frac{\gamma^2}{R_4} R)$ QED (i)

Eqs. (4.3) and (4.6) prove the first part of the theorem.

We now trace the errors through step $(-k-1)$. Observe that in PBA-1

$$m_{-k-1} = t_0^{(-k)}/t_k^{(k)} = t_0^{(-k)}/t_k^{(k-1)} \quad (4.7)$$

is the inverse of m_{-k} used in BNA, so its relative error is the same. Thus from (4.3.9a)

$$\mathit{rel}\ \bar{m}_{-k-1} \leq \mu R_1\ L, \quad \text{where } L := t_{max}^{(-k)}/\varepsilon \quad (4.8)$$

Now,

$$\begin{aligned} |\delta(\bar{m}_{-k-1}t_{j+k-1}^{(k)})| &\leq |m_{-k-1}t_{j+k-1}^{(k)}|\{\mathit{rel}\ \bar{m}_{-k-1} + \mathit{rel}\ t_{j+k-1}^{(k)}\} \\ &\leq |m_{-k-1}t_{max}^{(k)}|\mathit{rel}\ \bar{m}_{-k-1} + |m_{-k-1}\delta t_{max}^{(k)}| \\ &\leq |m_{-k-1}t_{max}^{(k)}|\mathit{rel}\ \bar{m}_{-k-1}\{1+\mathit{rel}\bar{T}_*^{(k)}/\mathit{rel}\bar{m}_{-k-1}\} \end{aligned} \quad (4.9)$$

which with (4,8) and (4.7) is

$$\leq \mu R_5 t_{max}^{(-k)}\ |t_{max}^{(k)}/t_{k-1}^{(k)}|, \text{ where } R_5 := R_1(1+\mathit{rel}\bar{T}_*^{(k)}/(\mu R_1 L)) \quad (4.10)$$

Now from PBA-1

$$t_{max}^{(k)} \leq t_{max}^{(k-1)} + |m_k t_{max}^{(-k)}| = t_{max}^{(k-1)} + \gamma^{-1}|t_0|,$$

since $m_k = t_0/t_{-k-1}^{(-k)}$.

So $|\delta(\bar{m}_{-k-1}t^{(k)}_{j+k+1})| \leq \mu R_5 t^{(-k)}_{max}(t^{(k-1)}_{max} + \gamma^{-1}|t_0|)/|t^{(k-1)}_k|$,

which, with (4.3.8)

$$\leq \mu R_6 \gamma^{-1} cond\ T^E_k\ t^{(-k)}_{max}, \qquad R_6 = \beta' R_5(1+\gamma) \qquad (4.11)$$

So from PBA-1

$$|\delta t^{(-k-1)}_j| \leq |\delta t^{(-k)}_j| + |\delta m_{-k-1} t^{(+k)}_{j+k-1}| \leq \mu R_7 \gamma^{-1} cond T^E_k t^{(-k)}_{max},$$

where $R_7 = R_6(1 + \dfrac{\gamma}{cond\ T^E_k} \cdot \dfrac{R}{R_6})$

So
$$\begin{aligned} t^{(-k-1)}_{max} &\geq t^{(-k)}_{max} - |m_{-k-1}|t^{(k)}_{max} \\ &= t^{(-k)}_{max} - \left|\frac{\varepsilon}{t^{(k)}_k}\right| (t^{(k-1)}_{max} + \gamma^{-1}|t_0|) \\ &= t^{(-k)}_{max}(1-\beta' L^{-1} cond\ T^E_k(1+\gamma^{-1})) \end{aligned}$$

so $rel\ T^{(-k-1)} \leq \mu R_8 \gamma^{-1} cond\ T^E_k$,

$$\text{where} \qquad R_8 = R_7/\{(1-\beta' L^{-1}\ cond\ T^E_k(1+\gamma^{-1}))\} \qquad (4.12)$$

QED.

A slightly tighter bound than that given by Theorem 4.1 can be obtained by observing that in (4.11) we majorized $|t^{(-k)}_{max}/t^{(k-1)}_k|$ by $\beta'(1+\gamma^{-1})cond\ T^E_k$. If this is not done, we get the following result:

Theorem 4.2 Let μ, R, R_5 and β be as in Theorem 4.1. Then

$$rel\ \bar{T}^{(-k-1)}_* \leq \mu R_9(|t^{(k)}_{max}/t^{(k)}_k|,\ R_9 := R_5(1+\gamma)/\{1-\beta' L^{-1} cond\ T^E_k(1+\gamma^{-1})\}$$

Proof: Elementary.

Discussion

For typical values of $\gamma (10^{-1}$ to $10^{-2})$ and for T_k and T_k^E well-conditioned, it can be seen that $R_9 \doteq R_8 \doteq \ldots \doteq R$. Hence Theorem 4.1(i) and Theorem 4.2 show that the error bound increase is approximately $|t_{max}^{(\pm i)}/t_{pivot}^{(\pm i)}|$ at each step. Hence the simple pivoting strategy will work providing γ is not too small.

This analysis can be extended to a more general pivoting strategy, but the working is tedious. We conjecture that the same conclusion will result, i.e. the error bound increase will be approximately $|t_{max}^{(\pm i)}/t_{pivot}^{(\pm i)}|$ at each step.

5. INTRODUCTION OF PIVOTING INTO THE BAREISS SYMMETRIC ALGORITHM (BSA)

We can get a pivoted BSA by replacing all BNA-steps by BSA-steps, but leaving the pivoting steps unchanged. So, the pivoted BSA (single strategy) could run as follows:

Algorithm 5.1 (PBSA-1)

1. Execute BSA, until for some k, $|t_0^{(-k)}/t_{max}^{(k)}| < \delta$, or until $k=n-1$. If $k=n-1$, exit, else do
2. Perform step 2 of PBA-1, and go to 1.

6. CONCLUSION

Methods have been proposed to introduce pivoting into the Bareiss algorithm, and a simple error analysis and some examples suggest that an improved numerical performance results.

CHAPTER 6

NUMERICAL ASPECTS OF TOEPLITZ FACTORIZATION & INVERSION

1. INTRODUCTION

In the previous two chapters, we discussed (i) the propagation of rounding errors and (ii) pivoting in the Bareiss algorithm for Toeplitz elimination. In this chapter, we discuss these topics in two related Toeplitz problems - Toeplitz factorization by the adapted Bareiss algorithm (ABA), and Toeplitz inversion by the Trench-Zohar algorithm (TZA). In Section 2, we use the forward error analysis of the Bareiss non-symmetric algorithm (derived in Chapter 4) to show that ABA is unstable, that is, we show that after step k the error bounds increase by a factor proportional to cond T_{k+1}. As mentioned in Chapter 4, De Jong [24] has shown that Rissanen's algorithm for triangularizing a *Hankel* matrix is unstable; we observed that Rissanen's algorithm was not equivalent to ABA.

In Section 3, we perform a backward error analysis of ABA and other Toeplitz factorizers by relating the backward errors of the latter to the backward errors of rank-1 update algorithms for factors. In section 4, we derive a factorization interpretation of the pivoted Bareiss algorithm(PBA), enabling pivoting to be incorporated into Toeplitz factorization. In section 5, we use the forward error analysis of PBA (derived in Chapter 5) to show that the possible large increase in the error bound of ABA is avoided in the pivoted factorizer.

Cybenko's [22] error analysis of TZA shows that the error bound at step k increases without limit as cond $T_{k+1} \to \infty$. In section 6, we give a simple example that demonstrates this. We then use the connexion between BSA and TZA, derived in Chapter 3, to introduce pivoting in TZA. By modifying our error analysis, of the pivoted Bareiss algorithm, we can show that the instability which occurs when cond $T_{k+1} \to \infty$ is now avoided.

2. TOEPLITZ FACTORIZATION - INCREASE IN ERROR BOUND

Recall from Chapter 2 that the adapted Bareiss algorithm (ABA) could be used to find the LU-factorization of T. ABA is the same as BNA except for the initial matrices. In BNA the initial matrix for both the PI and NI iterates was T, but in ABA, the initial matrices for the PI and NI iterates, denoted $T^{(-0)}$ and $T^{(+0)}$, were T and $t_0^{-1}T$ respectively.

At the conclusion of ABA, we had

$$U = T^{(1-n)} \qquad (2.1)$$

and

$$L = T^{(n-1)T2} \qquad (2.2)$$

where L is ULT, was UT, and $LU = T$.

We now consider the propagation of rounding errors in ABA from step $(-k)$ to step (k). The operations in step (k) are exactly the same as for BNA, and the error analysis is the same, so Theorem 4.3.1 may be applied directly. Recall the definitions:

for any scalar quantity a, $\delta a := \bar{a} - a$, where $\bar{a}$ is the computed value of a,

for any matrix A, $rel\ \bar{A} := \max_{i,j}|\delta a_{ij}|/\max_{i,j}|a_{ij}|$,

$A_* :=$ Toeplitz part of A.

Then we have

<u>Theorem 2.1</u> Let $t_j^{(\pm i)}$ be the exact quantities produced by ABA, let $\varepsilon : = |t_0^{(-k)}|$, $t_{max}^{(-k)} : = \max_{i,j} |(T_*^{(-k)})_{ij}|$, $\lambda : = t_{max}^{(-k)}/\varepsilon$, and let μ, β, R and R_3 be as in Theorem 4.3.1. Then (i) $rel\ \bar{T}_*^{(k)} \leqslant \mu\lambda R_3$,

$$\text{(ii)} \quad |\delta\ell_{i,k+1}| \leqslant \mu\lambda R_3 t_{max}^{(k)} \qquad (2.3a,b)$$

$$\text{(iii)} \quad \frac{\|T\|}{\|T_{k+1}\|} \cdot \frac{cond\ T_{k+1} - \beta\ cond\ T_k}{cond\ T(1+\beta\ cond\ T_k)^2(n-k)} \leqslant \lambda \leqslant cond\ T_{k+1} \max_j \left(\frac{\|\underline{t}_{1:k+1,j}\|}{\|T_{k+1}\|}\right),$$

where the $\ell_{i,k+1}$ of (2.3b) is an element of the matrix L in (2.2).

<u>Proof:</u> Parts (i) and (iii): as in Theorem 4.3.1. Part (ii): Eq(2.3b) follows from (2.3a), the definition of $rel\ \bar{T}_*^{(k)}$, and eq.(2.2). Theorem 2.1 shows that ABA is unstable, i.e. as $cond\ T_k \to \infty$, $\lambda \to \infty \Rightarrow rel\ \bar{T}_*^{(k)}$ is unbounded. Example 3.1, given later, shows that the error in ABA may be arbitrarily large.

3. TOEPLITZ FACTORIZATION - BACKWARD ERROR ANALYSIS

In this section, we derive bounds on the perturbation matrix, E, defined by

$$\bar{L}\bar{U} =: T + E.$$

Recall from Chapter 2 that FTF's could be derived from rank 1 factor updaters (R1FU's). We will show that the perturbation matrices for FTF's are simply related to the perturbation matrices for the associated R1FU's, so previous error analyses of R1FU's can be used in the analysis of FTF's.

This section is organized as follows: In subsection 3.1, a simple relation (Theorem 3.1) is proved between the perturbation matrices of MBA, the modified Bennett algorithm(MBA) and the adapted Bareiss Algorithm(ABA). In subsections 3.2 and 3.3, Fletcher and Powell's error analysis is modified to calculate the perturbation matrices for MBA in the general and symmetric positive-definite cases. In subsection 3.4, Theorem 3.1 and the results for MBA is used to analyze ABA - a-posteriori bounds for E are derived for the general case, and a-priori bounds are derived for the positive-definite case. In subsection 3.5, a variant of Theorem 3.1 is used to analyze a Toeplitz *LDR* factorizer.

3.1 Relation Between the Backward Errors for MBA and ABA

The relationship is contained in the following theorem:

Theorem 3.1 Let $\bar{L}$ and $\bar{U}$ be the factors of T computed by ABA. Let E, the *backward error matrix* for ABA, be such that

$$\bar{L}\bar{U} = T + E; \tag{3.1}$$

let $\bar{L}'$ and $\bar{U}'$ be the $(n-1)th$ leading principal submatrices of $\bar{L}$ and $\bar{U}$ respectively, let $\bar{\underline{x}}$ be the computed value of $\underline{t}_{2:n,1}/t_{11}$, let $\bar{\underline{y}} := -\underline{t}^T_{1,2:n-1}$, and let $\bar{L}^*$ and $\bar{U}^*$ be factors of $\bar{L}'\bar{U}' + \bar{\underline{x}}\bar{\underline{y}}^T$ computed using MBA. Let G, the backward error for MBA, be such that

$$\bar{L}^*\bar{U}^* = \bar{L}'\bar{U}' + \bar{\underline{x}}\bar{\underline{y}}^T + G, \tag{3.2}$$

and let μ be the machine precision.

Then (i) $e_{1j} = 0, \quad j=1,\ldots,n;$ (3.3)

(ii) $\exists|\varepsilon_{i1}| \le \mu, \quad i=2,\ldots,n$ such that $e_{i1} = \varepsilon_{i1}t_{i1}$ (3.4)

(iii) $e_{i+1,j+1} = e_{ij} + g_{ij}, \quad i=2,\ldots,n; \; j=2,\ldots,n$. (3.5)

Proof: (i) From (3.1), $(T+E)_{1j} = \bar{\ell}_{11}\bar{u}_{1j} = 1.\bar{u}_{1j} = t_{1j}$

using (2.4.29b) $\Rightarrow e_{1j} = 0$ (3.6)

(ii) For floating-point calculations, we may assume [89] that

$$fl(x \circ y) = (1+\varepsilon)(x \circ y), \quad |\varepsilon| \le \mu, \qquad (3.7)$$

where fl denotes the result of a floating-point computation and $\circ$ is $+$, $-$, $\times$, or $/$.

For $i=2,\ldots,n$, (3.1) gives

$$(T+E)_{i1} = \bar{\ell}_{i1}\bar{u}_{11} = \bar{\ell}_{i1}t_{11}, \text{ which using (2.4.29a),}$$

$$= (1+\varepsilon_{i1})(t_{i1}/t_{11})t_{11}, \quad \varepsilon_{i1} \le \mu \qquad (3.8)$$

QED(ii).

(iii) Consider algorithm 2.4.4 with the substitutions $L \to \bar{L}'$, $U \to \bar{U}'$, $\underline{x} \to \underline{\bar{x}}$, $y \to \underline{\bar{y}}$, $\tilde{L} \to \bar{L}^*$, $\tilde{U} \to \bar{U}^*$, and compare it with algorithm 2.4.5 with the substitutions $L \to \bar{L}$ and $U \to \bar{U}$. We show inductively that

$$\bar{L}^* = \bar{L}_{2:n,2:n} \; ; \; \bar{U}^* = \bar{U}_{2:n,2:n} \; . \qquad (3.9)$$

To do this, denote the $\{\underline{x}^{(i)}\}$ and $\{\underline{y}^{(i)}\}$ computed by algorithm 2.4.4 as $\{\underline{\bar{x}}^{*(i)}\}$ and $\{\underline{\bar{y}}^{*(i)}\}$, and those computed by algorithm 2.4.5 as $\{\underline{\bar{x}}^{(i)}\}$ and $\{\underline{\bar{y}}^{(i)}\}$. It is clear from (2.4.28c,d) and (2.4.29c,d) that $\underline{\bar{x}}^{\prime(1)} = \underline{\bar{x}}^{(1)}$ and $\underline{\bar{y}}^{*(1)} = \underline{\bar{y}}^{(1)}$. Suppose $\underline{\bar{x}}^{*(i)} = \underline{\bar{x}}^{(1)}$ and $\underline{\bar{y}}^{*(i)} = \underline{\bar{y}}^{(i)}$. Recall from the theorem statement that we defined $\underline{\bar{\ell}}'_{.i} := \underline{\bar{\ell}}_{1:n-1,i}$ and $\underline{\bar{u}}'_{i.} := \underline{\bar{u}}_{i,1:n-1}$. Hence the inputs to step 2, iteration i of algorithm 2.4.4 are the same as those to step 3, algorithm 2.4.5. Since the operations are the same, the outputs will be the same in rounded calculations. Hence $\underline{\bar{\ell}}^*_{.i} = \underline{\bar{\ell}}_{2:n,i+1}$, $\underline{\bar{u}}^*_{i.} = \underline{\bar{u}}_{i+1,2:n}$, $\underline{\bar{x}}^{*(i+1)} = \underline{\bar{x}}^{(i+1)}$ and $\underline{\bar{y}}^{*(i+1)} = \underline{\bar{y}}^{(i+1)}$. This completes the inductive proof of (3.9).

The (i,j) element of (3.2) yields

$$\underline{\bar{\ell}}^*_{i.}\underline{\bar{u}}^*_{.j} = \underline{\bar{\ell}}'_{i.}\underline{\bar{u}}'_{.j} + \bar{x}_i\bar{y}_j + g_{ij} \ .$$

Using (3.9), the definitions of $\bar{L}'$, $\bar{U}'$, $\underline{\bar{x}}$, and $\underline{\bar{y}}$, and (2.4.29a,b),

$$\underline{\bar{\ell}}_{i+1,2:n}\underline{\bar{u}}_{2:n,j+1} = \underline{\bar{\ell}}_{i.}\underline{\bar{u}}_{.j} - \bar{\ell}_{i+1,1}\bar{u}_{1,j+1} + g_{ij}$$

i.e. $$\underline{\bar{\ell}}_{i+1.}\underline{\bar{u}}_{.j+1} = \underline{\bar{\ell}}_{i.}\underline{\bar{u}}_{.j} + g_{ij} \ . \qquad (3.10)$$

The (i,j) element of (3.1) yields

$$\underline{\bar{\ell}}_{i.}\underline{\bar{u}}_{.j} = t_{ij} + e_{ij} \qquad (3.11)$$

Similarly

$$\begin{aligned}\underline{\bar{\ell}}_{i+1.}\underline{\bar{u}}_{.j+1} &= t_{i+1,j+1} + e_{i+1,j+1} \\ &= t_{ij} + e_{i+1,j+1} \quad (T \text{ Toeplitz}) \qquad (3.12)\end{aligned}$$

Putting (3.12) and (3.13) in (3.10) yields

$e_{i+1,j+1} = e_{ij} + g_{ij}$. QED(iii).

<u>Corollary 2.1</u> Let $E, G, T, \varepsilon_{i1}$ and μ be as defined above. Then, the elements of E are given explicitly by

$$e_{ij} = \begin{cases} \sum_{k=1}^{i-1} g_{i-k,j-k} & i \le j \\ \varepsilon_{i-j+1,1}t_{i-j+1} + \sum_{k=1}^{j-1} g_{i-k,j-k}, & \\ \qquad |\varepsilon_{i-j+1,1}| \le \mu, & i > j \ . \end{cases} \qquad (3.13)$$

<u>Proof:</u> Trivial.

Thus, to find bounds for E, we find bounds for G by adapting the error analysis of Fletcher and Powell [27] for rank-one updates, then apply (3.13).

3.2 Bounds for the Backward Errors of MBA

The approach follows that of [27], but some details are different, since a different rank-1 updater, the composite-*t* method, is considered. Also, [27] considers only the positive-definite case, whereas we consider the indefinite case, and then in the next sub-section, specialize the analysis to the symmetric positive-definite case. It will be found that in the indefinite case, stability cannot be guaranteed (in the sense that $\exists$ cases where the error is unbounded - see example 3.1), and only a-posteriori bounds can be derived. The symmetric positive-definite case is stable, and a-priori bounds can be derived.

Throughout this sub-section, we refer to MBA, given in algorithm 2.4.4. The inputs to the algorithm, $L,U,\underline{x},\underline{y}$, and the outputs from the algorithm, $\tilde{L}$ and $\tilde{U}$ are related by

$$\tilde{L}\tilde{U} = LU + \underline{x}\underline{y}^T .$$

It can be shown by induction that the first $i-1$ elements of $\underline{\ell}_{.i}$ and $\underline{u}_{i.}$ are null, $(i=1,\ldots,n)$. It is also easily checked that the quantities involved in the i^{th} basic step satisfy

$$\tilde{\underline{\ell}}_{.i}\tilde{\underline{u}}_{i.} + \underline{x}^{(i+1)}\underline{y}^{(i+1)T} = \underline{\ell}_{.i}\underline{u}_{i.} + \underline{x}^{(i)}\underline{y}^{(i)T} \qquad (3.14)$$

Consider the errors incurred in the i^{th} basic step. For ease of notation, we rewrite $\underline{\ell}_{.i},\underline{u}_{i.},\underline{x}^{(i)},\underline{y}^{(i)},\tilde{\underline{\ell}}_{.i},\tilde{\underline{u}}_{i.},\underline{x}^{(i+1)}$ and $\underline{y}^{(i+1)}$ as $\underline{\ell},\underline{u}^T,\underline{x},\underline{y},\underline{\ell}^*,\underline{u}^{*T},\underline{x}^*$ and $\underline{y}^*$ respectively. Eq. (3.14) can then be rewritten

$$\underline{\ell}^*\underline{u}^{*T} + \underline{x}^*\underline{y}^{*T} = \underline{\ell}\underline{u} + \underline{x}\underline{y}^T \tag{3.15}$$

Computed quantities will be denoted by bars. We wish to bound the error $G^{(i)}$ in carrying out (3.15), defined by

$$\bar{\underline{\ell}}^*\bar{\underline{u}}^{*T} + \bar{\underline{x}}^*\bar{\underline{y}}^{*T} = \underline{\ell}\underline{u} + \underline{x}\underline{y}^T + G^{(i)} \tag{3.16}$$

The quantities on the left of (3.16) are the exact data for the next basic step. It therefore follows that the complete computed factorization is an exact factorization of a perturbation of the original problem given by

$$\tilde{A} = \tilde{L}\tilde{U}^T + \underline{x}\underline{y}^T + G \tag{3.17a}$$

$$\text{where} \qquad G := \Sigma G^{(i)}, \tag{3.17b}$$

so bounds on G may be obtained from bounds on the $G^{(i)}$. Note that $\tilde{A}$ is now a *computed* matrix.

Following Wilkinson [89], we find that the following calculations are made when the i^{th} step of MBA is executed with rounding errors:

$$\beta = x_i \tag{3.18a}$$

$$\bar{u}^*_j = (1+e_{1j})(u_j-(1+e_{2j})\beta y_j), \quad j \geqslant i \tag{3.18b}$$

$$\bar{x}^*_j = (1+e_{3j})(x_j-(1+e_{4j})\beta\ell_j), \quad j \geqslant i+1 \tag{3.18c}$$

$$\bar{\gamma} = (1+e_1)y_i/\bar{u}^*_i \tag{3.18d}$$

$$\bar{\ell}^*_j = (1+e_{5j})(\ell_j-(1+e_{6j})\bar{\gamma}\bar{x}^*_j), \quad j \geqslant i+1 \tag{3.18e}$$

$$\bar{y}^*_j = (1+e_{7j})(y_j-(1+e_{8j})\bar{\gamma}\bar{u}^*_j), \quad j \geqslant i+1, \tag{3.18f}$$

where e_1 and $e_{1j}, e_{2j}, \ldots$ each represent separate errors bounded by

μ, the relative machine precision. Following the method of [27] we can manipulate eqs. (3.18), eventually obtaining

$$\begin{aligned} g_{jk}^{(i)} &:= \bar{\ell}_j^* \bar{u}_k^* + \bar{x}_j^* \bar{y}_k^* - \ell_j u_k - x_j y_k \\ &= \bar{\ell}_j^* \bar{u}_k^* (e_1 + e_{1k} + e_{3j} + e_{5j} + e_{6j} - e_{1i} - e_{2i} + O(\mu^2)) \\ &+ \ell_j \bar{u}_k^* (e_{1i} + e_{2i} - e_1 - e_{3j} - e_{4j} - e_{6j} + O(\mu^2)) \\ &- \ell_j u_k (-e_{4j}) + \bar{x}_j^* \bar{y}_k^* (e_{1i} + e_{2k} + e_{3j} - e_1 - e_{1k} - e_{7k} - e_{8k} + O(\mu^2)) \\ &+ \bar{x}_j^* y_k (e_{1i} + e_{2i} - e_1 - e_{1k} - e_{2k} - e_{8k} + O(\mu^2)) \end{aligned} \qquad (3.19)$$

We now find first-order bounds on G, as is done in [27]. Neglecting the $O(\mu^2)$ terms, we have from (3.19)

$$|g_{jk}^{(i)}| \le \mu\{7|\bar{x}_j^* \bar{y}_k^*| + 6|\bar{x}_j^* y_k| + 7|\bar{\ell}_j^* \bar{u}_k^*| + 6|\ell_j \bar{u}_k^*| + |\ell_j u_k|\} \qquad (3.20)$$

From (3.17b) and (3.20) and writing $\bar{x}_j^*$ as $x_j^{(i+1)}$, $\bar{\ell}_j^*$ as $\tilde{\ell}_{ji}$, etc.,

$$\begin{aligned} |g_{jk}| &\le \sum_{i=1}^{n} |g_{jk}^{(i)}| \\ &\le \mu \sum_{i=1}^{n} \{7|x_j^{(i+1)}||y_k^{(i+1)}| + 6|x_j^{(i+1)}||y^{(i)}| + 7|\tilde{\ell}_{ji}||\tilde{u}_{ik}| \\ &\quad + 6|\ell_{ji}||\tilde{u}_{ik}| + |\ell_j||u_k|\}, \end{aligned} \qquad (3.21)$$

where all the quantities on the RHS of (3.21) are *computed* quantities, and $x_j^{(i+1)} = y_k^{(i+1)} = 0$.

Define $X := [\underline{x}^{(1)}, \underline{x}^{(2)}, \ldots, \underline{x}^{(n)}]$ and $Y := \begin{pmatrix} \underline{y}^{(1)T} \\ \underline{y}^{(2)T} \\ \vdots \\ \underline{y}^{(n)T} \end{pmatrix}$

Then (3.21) may be written completely in matrix notation:

$$|G| \leq \mu\{7|X||Y|-7|\underline{x}^{(1)}||\underline{y}^{(1)T}|+6|X|Z|Y| \\ + 7|\tilde{L}||\tilde{U}|+6|L||\tilde{U}|+|L||U|\} \qquad (3.22)$$

where for any matrix A, $|A| = [|a_{ij}|]$, and Z is the shift matrix

$$Z = \begin{bmatrix} 0 & & & \\ 1 & \ddots & & \text{O} \\ & \ddots & \ddots & \\ \text{O} & & 1 & 0 \end{bmatrix} .$$

Now from (3.22)

$$|G| \leq \mu\{7|X||Y|+6|X|Z|Y|+7|\tilde{L}||\tilde{U}|+7|L||\tilde{U}|+|L||U|+|\tilde{L}||U|\}$$

$$= \mu|X|(7I+6Z)|Y|+\mu(|\tilde{L}|+|L|)(7|\tilde{U}|+|U|) \qquad (3.23)$$

Eq. (3.23) provides first-order a-posteriori bounds on the elements of G. Unfortunately, (3.23) requires $O(n^3)$ operations to evaluate compared to $O(n^2)$ operations for MBA itself. A cruder bound which requires only $O(n^2)$ operations to evaluate may be obtained by observing that for any two vectors $\underline{a}$ and $\underline{b}$,

$$|\underline{a}^T\underline{b}| \leq \|\underline{a}\|_2\|\underline{b}\|_2 \leq \|\underline{a}\|_1\|\underline{b}\|_1 = (\Sigma|a_i|)(\Sigma|b_i|)$$

Let $$a_i^r := \|\underline{a}_{i.}\|_1 \text{, and } a_i^c := \|\underline{a}_{.i}\|_1 \qquad (3.24)$$

Then from (3.23)

$$|G| < \mu\underline{x}^r(7I+6Z)\underline{y}^{cT}+\mu(\tilde{\underline{\ell}}^r+\underline{\ell}^r)(7\tilde{\underline{u}}^{cT}+\underline{u}^{cT}) \qquad (3.25)$$

This bound is of the type derived by Wilkinson [89], which relates the backward error to the largest element encountered during the algorithm.

Remark on the Bound

Only a-posteriori bounds for G can be derived in the indefinite case, and stability cannot be guaranteed, since we can find examples where the errors are unbounded, such as:

Example 3.1

Take

$$L = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} -1-2\varepsilon & 1 \\ 0 & -1 \end{bmatrix}, \quad \underset{\sim}{x}^T = \underset{\sim}{y} = [(1+\varepsilon), 0]$$

where $|\varepsilon| < \mu^{½}$, where μ is the machine precision.

Here

$$A = LU = \begin{bmatrix} -1-2\varepsilon & 1 \\ 1+2\varepsilon & 0 \end{bmatrix} \quad \text{and} \quad \tilde{A} = A+\underset{\sim}{xy}^T = \begin{bmatrix} \varepsilon^2 & 1 \\ 1+2\varepsilon & 0 \end{bmatrix}$$

have spectral condition numbers bounded by $(2+4\varepsilon)^2$ and $(1+|\varepsilon|^2)$ respectively, so both are well-conditioned. Executing MBA, we get

$$\underset{\sim}{x}^{(1)} \leftarrow \begin{pmatrix} 1+\varepsilon \\ 0 \end{pmatrix}; \quad \underset{\sim}{y}^{(1)} \leftarrow -\begin{pmatrix} 1+\varepsilon \\ 0 \end{pmatrix}$$

$$\beta \leftarrow 1+\varepsilon$$

$$\underset{\sim}{\tilde{u}}_{1.} \leftarrow \begin{pmatrix} -1-2\varepsilon \\ 1+2\varepsilon \end{pmatrix} + (1+\varepsilon)\begin{pmatrix} 1+\varepsilon \\ 0 \end{pmatrix} = \begin{cases} \begin{pmatrix} \varepsilon^2 \\ 1+2\varepsilon \end{pmatrix} & \text{exact} \\ \begin{pmatrix} 0 \\ 1+2\varepsilon \end{pmatrix} & \text{computed} \end{cases}$$

Since the computed value of $\tilde{u}_{11}$ is zero, the algorithm will fail at step 7 (eq.(2.4.28f)), i.e. the error will be unbounded. (In exact arithmetic, $\gamma = -\frac{1+\varepsilon}{\varepsilon^2}$).

3.3 Bounds for the backward errors of MBA:positive-definite-symmetric case

Continuing our development in analogy with [27], we observe that when the matrices are symmetric the terms involved in the i^{th}

step are related by

$$\bar{\underline{y}}^* = \bar{\sigma}^*\underline{x}^*, \quad \text{where} \quad \bar{\sigma}^* := \bar{y}^*_{i+1}/\bar{x}^*_{i+1} \tag{3.26}$$

$$\underline{y} = \sigma\underline{x}, \quad \text{where} \quad \sigma := y_i/x_i \tag{3.27}$$

$$\bar{\underline{u}}^* = \bar{d}^*\bar{\underline{\ell}}^*, \quad \text{where} \quad \bar{d}^* := \bar{u}^*_{ii} \tag{3.28}$$

$$\underline{u} = d\underline{\ell}, \quad \text{where} \quad d := u_{ii}, \tag{3.29}$$

so that (3.20) can be written

$$|g^{(i)}_{jk}| \le \mu\{7|\bar{\sigma}^*\bar{x}^*_j\bar{x}^*_k| + 6|\sigma\bar{x}^*_j x_k| + 7|\bar{d}^*\bar{\ell}^*_j\bar{\ell}^*_k| + 6|\bar{d}^*\ell_j\bar{\ell}^*_k| + |d\ell_j u_k|\} \tag{3.30}$$

It can be shown [27] that

$$\frac{\bar{d}^*}{d} = \frac{\sigma}{\bar{\sigma}^*} \,. \tag{3.31}$$

Define

$$a^*_j := |\bar{d}^*|^{1/2}|\bar{\ell}^*_j|, \; a_j := |d|^{1/2}|\ell_j|, \; w^*_j := |\bar{\sigma}^*|^{1/2}|\bar{x}^*_j|,$$

$$w_j := |\sigma|^{1/2}|x_j| \quad \text{and} \quad r := |\bar{d}^*/d|^{1/2} \tag{3.32a-e}$$

Then (3.30) can be written, using (3.31)

$$|g^{(i)}_{jk}| \le \mu\{7w^*_j w^*_k + 6rw^*_j w_k + 7a^*_j a^*_k + 6ra_j a^*_k + a_j a_k\} \tag{3.33}$$

Following [27], we can use the Cauchy-Schwartz inequality on the last three terms in (3.33) to obtain

$$|g^{(i)}_{jk}| \le \mu\{w^*_j(7w^*_k+6rw_k)+[7a^{*2}_j+(6r+1)a^2_j]^{1/2}[(6r+7)a^{*2}_k+a^2_k]^{1/2}\}. \tag{3.34}$$

In the symmetric case, the computed matrix $\tilde{A}$ in (3.17a) satisfies the equation

$$\tilde{A} = \sum_{p=1}^{i-1}\tilde{\underline{\ell}}_{\cdot p}\tilde{d}_p\tilde{\underline{\ell}}^T_{\cdot p} + \sum_{p=i}^{n}\underline{\ell}_{\cdot p}d_p\underline{\ell}^T_{\cdot p} + \sigma_i\underline{x}^{(i)}\underline{x}^{(i)T} + \sum_{p=i}^{n}G^{(p)}$$

where $\sigma_i := y_i^{(i)}/x_i^{(i)}$ and is analogous to the definition in (3.27). In the positive-definite case, all the $\tilde{d}_p$ and d_p are positive. The diagonal elements of this matrix equation then give the inequality

$$|\sigma_i|^{\frac{1}{2}}|x_j^{(i)}| \le (\tilde{a}_{jj}+h_{jj})^{\frac{1}{2}} \qquad (3.35)$$

where $$H := \Sigma|G^{(i)}| \; .$$

Now the definition of H, expression (3.34), definitions (3.32a-e), the Cauchy-Schwartz inequality and expression (3.35) give the bounds (recall that $\bar{d}^*=\tilde{d}_i$, $d=d_i$ etc.):

$$h_{kj} = h_{jk} = \sum_{i=1}^{j}|g_{jk}^{(i)}| \qquad (j \le k)$$

$$\le \mu\{ \sum_{i=1}^{j} |\sigma_{i+1}|^{\frac{1}{2}}|x_j^{(i+1)}|[7|\sigma_{i+1}|^{\frac{1}{2}}|x_k^{(i+1)}| + 6\hat{r}|\sigma_i|^{\frac{1}{2}}|x_k^{(i)}|] +$$

$$+ [7\sum_{i=1}^{j}|\tilde{d}_i\tilde{\ell}_{ji}^2| + (6\hat{r}+1)\sum_{i=1}^{j}|d_i\ell_{ji}^2|]^{\frac{1}{2}}[(6\hat{r}+7)\sum_{i=1}^{j}|\tilde{d}_i\tilde{\ell}_{ki}^2|+|d_il_{ki}^2|]^{\frac{1}{2}} \qquad (3.36a,b)$$

$$\text{where} \quad \hat{r} := \max_i|\tilde{d}_i/d_i|.$$

so with (3.35)

$$h_{jk} \le \mu\{(7+6\hat{r})j\sqrt{(\tilde{a}_{jj}+h_{jj})(\tilde{a}_{kk}+h_{kk})} + \sqrt{[7\tilde{a}_{jj}+(6\hat{r}+1)a_{jj}][(6\hat{r}+7)\tilde{a}_{kk}+a_{kk}]}\} \qquad (3.37)$$

By letting $j = k$, it follows that

$$h_{jj} \le (6\hat{r}+7)\mu[a_{jj}+\tilde{a}_{jj}]/(1-(7+6\hat{r})j\mu).$$

We consider separately the cases $\sigma > 0$ and $\sigma < 0$, where σ is defined as in (3.27). For the case $\sigma > 0$, we get, when $\delta := (7+6\hat{r})nu < \frac{1}{2}$

$$|g_{jk}| \le \mu(7+6\hat{r})j\sqrt{(\tfrac{3}{2}\tilde{a}_{jj}+\tfrac{1}{2}a_{jj})(\tfrac{3}{2}\tilde{a}_{kk}+\tfrac{1}{2}a_{kk})}+\sqrt{(7\tilde{a}_{jj}+(6\hat{r}+1)a_{jj})((6\hat{r}+7)\tilde{a}_{kk}+a_{kk})}$$

Also $\tilde{a}_{jj} > a_{jj}$ when $\sigma > 0$, so

$$|g_{jk}| \le \mu(2j+1)(6\hat{r}+8)\sqrt{\tilde{a}_{jj}\tilde{a}_{kk}}, \quad \delta \le \tfrac{1}{2} ; \tag{3.38}$$

when $\delta > \frac{1}{2}$, we must use (3.23) or (3.25).

When $\sigma < 0$, we get a very satisfactory result, since $|\frac{\tilde{d}_i}{d_i}| < 1 \Rightarrow \hat{r} < 1$ so $h_{jj} \le \{13\mu+O(\mu^2)\}(a_{jj}+\tilde{a}_{jj})$, and the h_{jj} term in (3.37) is of second order, and may be neglected in a first-order analysis, as is done by Fletcher and Powell. Eq. (3.37) then becomes, using $\hat{r} < 1$ and $\tilde{a}_{jj} < a_{jj}$

$$|g_{jk}| \le 14(\mu+1)\sqrt{a_{jj}a_{kk}} \qquad \sigma < 0 \tag{3.39}$$

Notice that (3.39) is entirely a-priori, so for $\sigma < 0$, stability is guaranteed. For $\sigma > 0$, the update algorithm can be modified in a manner similar to that indicated in [27], which will give a-priori bounds. We do not pursue this point here, for in the case of interest which arises from Toeplitz matrices, $\sigma < 0$.

3.4 Bounds for the backward errors in the adapted Bareiss algorithm for Toeplitz factorization

Let E be the backward error matrix as defined in (3.1).

Then from (3.13)

$$|e_{ij}| \le \mu\zeta_{i-j}|t_{i+j-1}| + \sum_{k=1}^{min(i-1,j-1)} |g_{i-k,j-k}| \tag{3.40}$$

$$\zeta_{i-j} = \begin{cases} 0, & i \le j \\ 1, & i > j \end{cases}$$

where from Theorem 3.1, G is the backward error in computing

$$\bar{L}^*\bar{U}^* = \bar{L}'\bar{U}' + \underline{\bar{x}}\,\underline{\bar{y}}^T + G , \tag{3.41}$$

where $\bar{L}'$ and $\bar{U}'$ are the leading submatrices of the computed upper and lower triangles, $\bar{\underline{x}}$ is the computed value of $-\underline{t}_{2:n,1}/t_{11}$ and $\bar{\underline{y}} := \underline{t}_{1,2:n}$.

We now evaluate E for the general case. Applying (3.25) for the problem (3.41),

$$|G| < \mu\bar{\underline{x}}^{*r}(7I+6Z)\bar{\underline{y}}^{*cT} + \mu(\bar{\underline{\ell}}^{*r}+\bar{\underline{\ell}}'^{r})(7\bar{\underline{u}}^{*cT}+\bar{\underline{u}}'^{cT}) \qquad (3.42)$$

where the operators $\cdot^{r}$ and $\cdot^{c}$ are as in (3.24), and $\bar{X}*$ and $\bar{Y}*$ are the matrices of auxiliary vectors produced by MBA. We showed in Theorem 3.1 that $\bar{X}*$ and $\bar{Y}*$ were the same as $\bar{X}$ and $\bar{Y}$, the matrix of auxiliary vectors produced by ABA, and also that $\bar{L}* = \bar{L}_{2:n,2:n}$ and $\bar{U}* = \bar{U}_{2:n,2:n}$ where $\bar{L}$ and $\bar{U}$ are the computed Toeplitz factors. Also by definition $\bar{L}' = \bar{L}_{n-1}$ and $\bar{U}' = \bar{U}_{n-1}$, so the (p,q) elements of (3.42) can be written

$$|g_{pq}| < \mu\bar{x}^{r}_{p}(7\bar{y}^{r}_{q}+6\bar{y}^{r}_{q}) + \mu(\bar{\ell}^{r}_{p}+\bar{\ell}^{r}_{p+1})(7\bar{u}^{c}_{q}+\bar{u}^{c}_{q+1}) \qquad (3.43)$$

Putting (3.43) in (3.40) yields

$$|e_{ij}| \le \mu\zeta_{i-j}|t_{i+j-1}| + \mu \sum_{k=1}^{min(i-1,j-1)} \{\bar{x}^{r}_{i-k}(7\bar{y}^{r}_{j-k}+6\bar{y}^{r}_{j-k+1}) +$$

$$+ (\bar{\ell}^{r}_{i-k}+\bar{\ell}^{r}_{i-k+1})(7\bar{u}^{c}_{j-k}+\bar{u}^{c}_{j-k+1}) \qquad (3.44)$$

The bounds (3.44) require $O(n^2)$ operators to evaluate.

When T is symmetric positive-definite (PDS), (3.41) becomes $\bar{L}*\bar{U}* = \bar{L}'\bar{U}' - \bar{\underline{x}}*(\alpha\bar{\underline{x}}*)^T$, where $\alpha > 0$, so we can use (3.39) and (3.40) to give

$$\left.\begin{aligned} |e_{ij}| &\le \mu \sum_{k=1}^{i-1} 14(k+1)|t_{i+1-j,1}| && (i \le j) \\ &= 7\mu|t_{i+1-j,1}|i(i+1) && (i \le j) \\ \text{and} \qquad |e_{ij}| &\le \mu|t_{i+1-j,1}|+|e_{ij}| && (i > j) \end{aligned}\right\} \qquad (3.45)$$

This is an a-priori bound and indicates that the FTF algorithm is stable in the PDS case.

3.5 Other FTF algorithms

As we saw in Chapter 2, other Toeplitz *LU* algorithms, and Toeplitz *LDR* algorithms, may be derived from different rank-1 updaters by using theorems 2.4.1 and 2.4.2. The error analysis of the *LU* updaters may be obtained using (3.13). For *LDR* factorizers, the analysis of (3.13) is the slightly modified form

$$e_{ij} = \varepsilon_{i-j+1}t_{i-j+1} + \sum_{k=1}^{j-1} g_{i-k,j-k}, \quad |\varepsilon_{i-j+1,1}| \le \mu. \tag{3.46}$$

We illustrate by analyzing algorithm 2.4.5 , the Toeplitz *LDR* factorizer based on Bennett's algorithm. Fletcher and Powell give an error analysis of the latter, so using these bounds (their eq. (5.30)) and (3.46) we obtain

$$|e_{ij}| \le \mu|t_{i+j-1,1}|(\tfrac{5}{2}j^2 + \tfrac{49}{2}j - 26). \tag{3.47}$$

This completes the discussion on the backward error analysis of FTF algorithms.

4. RELATION BETWEEN THE PIVOTED BAREISS ALGORITHM AND TOEPLITZ FACTORIZATION.

It can be seen from eqs. (2.1) and (2.2) that the triangles produced by the Bareiss algorithm are simply related to the Toeplitz factors. It would be expected that the reduced matrices produced by the Pivoted Bareiss algorithm (PBA) are related to some factorization of the Toeplitz matrix. In this section, such a relationship is proved for any PBA where BPA and/or BPB are used, i.e. only A, C and BNA-cycles are executed. Further work is required to find a factorization representation where BPC is used.

Note: We will henceforth refer to BNA-cycles as simply B-cycles. This is convenient because A,B and C cycles are variant of the Bareiss recursion that move the zero-band up 1,0 and -1 places respectively.

The following algorithm calculates two matrices, one ULT and the other UT, from the output of the PBA. It will be shown presently that the product of these matrices is a row and column-permuted version of T.

Algorithm 4.1 (Programmed on p.A.31)

Let $\{T^{(\pm i)}\}$ be the output of any PBA containing only A,C and B-cycles defined by procedures 5.2.2 , 5.2.3 and eq. (2.2.3). respectively. Let n_A and n_C be the number of A and C-cycles respectively. Then, carry out the following steps:

```
1.  Form the two permutation vectors $\underline{s}$ and $\underline{v}$ as follows:
    1.1  $s_1 \leftarrow n_A + 1;\ v_1 \leftarrow n_C + 1$
    1.2  For $k \leftarrow 1$ to $n-1$ do 1.2A, 1.2B or 1.2C if cycle $k$ is
         an A,B or C-cycle respectively:
         1.2A  $s_{k+1} \leftarrow min\{s_i\}_1^k-1;\ v_{k+1} \leftarrow max\{v_i\}_1^k+1$
         1.2B  $s_{k+1} \leftarrow max\{s_i\}_1^k+1;\ v_{k+1} \leftarrow max\{v_i\}_1^k+1$
         1.2C  $s_{k+1} \leftarrow max\{s_i\}_1^k-1;\ v_{k+1} \leftarrow min\{v_i\}_1^k+1$
```

2. Let P_1 be the permutation *matrix* which on premultiplication moves rows $s_1,s_2,\ldots,s_n$ to rows $1,2,\ldots,n$ respectively. Let P_2 be the permutation *matrix* which on premultiplication moves rows $v_1,v_2,\ldots,v_n$ to rows $1,2,\ldots,n$ respectively.

Note: If PA is a certain re-ordering of the rows of A, then the same re-ordering of the columns of A is $(PA^T)^T$, or AP^T. Note also that any permutation matrix P is orthogonal.

3. Set $g \leftarrow n_A$, $h \leftarrow n_C$

4. $\underline{\ell}'_{.1} \leftarrow P_1\underline{t}^{RT}_{n-n_C}$; $\underline{\ell}_{.1} \leftarrow \underline{\ell}'_{.1}/\ell'_{11}$; $\underline{u}_{1.} \leftarrow \underline{t}_{1+n_A.}P_2^T$

5. For $k \leftarrow 1$ to $n-1$ do 5A, 5B or 5C if cycle k is an A, B, or C-cycle respectively:

   5A. $\underline{\ell}'_{.k+1} \leftarrow P_1\underline{t}^{(-k)RT}_{\rho(g).}$; $\underline{\ell}_{.k+1} \leftarrow \underline{\ell}'_{.k+1}/\ell'_{k+1,k+1}$; $\underline{u}_{k+1.} \leftarrow \underline{t}^{(-k)}_{\lambda(h).}P_2^T/\tilde{m}^{(-k)}_{k+1}$;

   $g \leftarrow g-1$, where $\lambda(i)$ and $\rho(i)$ are the rows with i elements to the left and right of the zero-band respectively.

   5B. $\underline{\ell}'_{.k+1} \leftarrow P_1\underline{t}^{(k)RT}_{\rho(g).}$; $\underline{\ell}_{.k+1} \leftarrow \underline{\ell}'_{.k+1}/\ell'_{k+1,k+1}$; $\underline{u}_{k+1.} \leftarrow \underline{t}^{(-k)}_{\lambda(h).}P_2^T/\tilde{m}^{(-k)}_{k+1}$.

   5C. $\underline{\ell}'_{.k+1} \leftarrow P_1\underline{t}^{(k)RT}_{\rho(g).}$; $\underline{\ell}_{.k+1} \leftarrow \underline{\ell}'_{.k+1}/\ell'_{k+1,k+1}$; $\underline{u}_{k+1.} \leftarrow \underline{t}^{(k)}_{\lambda(h)}P_2^T/\tilde{m}^{(k)}_{k+1}$;

   $h \leftarrow h-1$.

We now show that L and U are the desired triangular factors:

Theorem 4.1 Let P_1, P_2, L and U be as produced by algorithm 4.1. Then:

(i) L and U are ULT and UT respectively.

(ii) $LU = P_1TP_2^T$.

Proof: (i) Let g_k and h_k be the value of g and h respectively after cycle k of algorithm 4.1 and let $n_A(k)$ and $n_C(k)$ be the number of A and C cycles respectively after cycle k of the PBA. It is easily seen from loops 1.2 and 5 that

$$g_k = n_A - n_A(k) \text{ and } s_{min}(k+1) := min\{s_i\}_1^{k+1} = n_A+1-n_A(k) \quad (4.1)$$

$$h_k = n_c - n_c(k) \text{ and } v_{min}(k+1) := min\{v_i\}_1^{k+1} = n_C+1-n_C(k) \quad (4.2)$$

It is also clear from loop 1.2 that

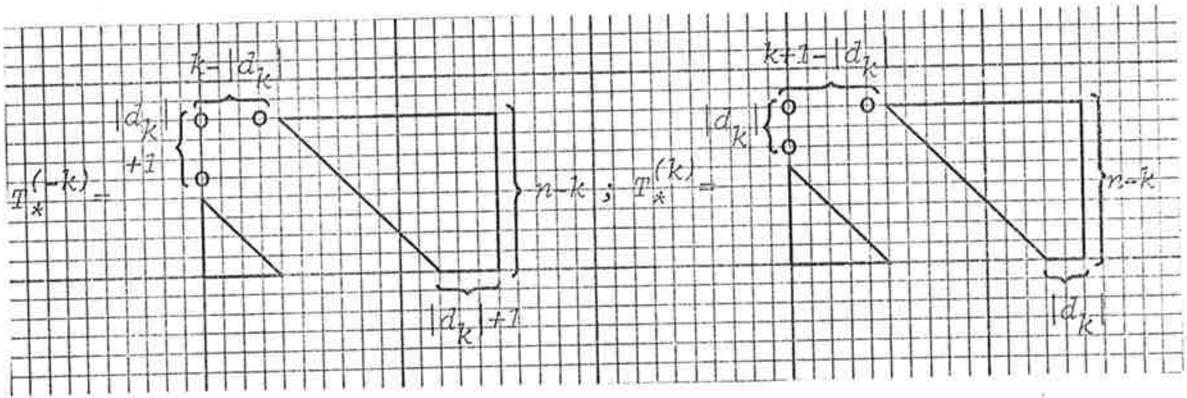
$$\{s_i\}_1^k = perm^*\{s_{min}(k), \ldots, (s_{min}(k))+k-1\}, \text{ which with (4.1)}$$
$$= perm\{g_{k-1}+1, \ldots, g_{k-1}+k\} \quad (4.3)$$

*permutation

However, by definition $\{t^{(-k)}_{\rho(g_{k-1}),j}\}^{n-g_{k-1}-k+1}_{n-g_{k-1}}$ are all zero $\Rightarrow \{(\underline{t}^{(-k)RT}_{\rho(g_{k-1}).})_i\}^{g_{k-1}+k}_{g_{k-1}+1}$ are all zero $\Rightarrow \{(P_1 t^{(-k)}_{\rho(g_{k-1}).})_i\}^{k}_{1}$ are all zero $\Rightarrow$ L is upper-triangular. With steps 5A(ii), 5B(ii) and 5C(iii), we have L is ULT. The proof that U is UT is similar to the proof that L is LT.

QED(i)

(ii) Let d_k be the displacement of the zero-band from its Bareiss-position at step k (+: above; -: below). Recall from Chapter 5, fig. (2.9) that the Toeplitz parts of $T^{(-k)}$ and $T^{(k)}$ have the forms (for $d_k < 0$):



It can be shown by induction on k that $\underline{t}^{(-k)}_{*i.}$ and $\underline{t}^{(k)}_{*i.}$ are linear combinations of $\{\underline{t}_{\ell.}\}^{i+k}_{i}$. This fact, together with the shapes of $T^{(-k)}_*$ and $T^{(k)}_*$ implies that

$$\underline{t}^{(-k)}_{\lambda(h_{k-1}).} = l.c.^* \{\underline{t}_{i.}\}^{h_{k-1}-d_k+1+k}_{h_{k-1}-d_k+1}$$

$$\text{and} \qquad \underline{t}^{(k)}_{\lambda(h_{k-1}).} = l.c.\{\underline{t}_{i.}\}^{h_{k-1}-d_k+1}_{h_{k-1}-d_k} . \tag{4.4a,b}$$

Since d_k increments with A-cycles and d_k decrements with C-cycles, we must have

$$d_k = d_0+n_A(k)-n_C(k) = n_C-n_A+n_A(k)-n_C(k).$$

*linear combination

So if cycle k was an A or B-cycle,

$$h_{k-1}-d_k+1 = h_k-d_k+1 = n_A(k)-n_A(k)+1 \quad \text{(using 4.2)}$$
$$= s_{min}(k+1), \quad \text{(using 4.1)}.$$

So by (4.4a),

$$\underline{t}^{(-k)}_{\lambda(h_{k-1}).} = l.c.\{\underline{t}_{i.}\}_{s_{min}(k+1)}^{s_{min}(k+1)+k} = l.c.\{\underline{t}_{i.}\}_{s_1}^{s_{k+1}} \qquad (4.5)$$

With steps 5A(ii) and 5B(ii), and the definition of $\tilde{\underline{m}}^{(\pm k)}$.

$$\text{Eq. (4.5)} \Rightarrow \underline{u}_{k+1.}P_2 = n.l.c.^*\{\underline{t}_{i.}\}_{s_1}^{s_{k+1}} \Rightarrow \underline{u}_{k+1.}P_2$$

$$= n.l.c.\{(P_1T)_{i.}\}_1^{k+1}$$

$$\Rightarrow \underline{u}_{k+1.} = n.l.c.\{(P_1TP_2^T)_{i.}\}_1^{k+1} \Rightarrow \underline{u}_{k+1.} \text{ is row } k$$

k of the unique U-factor of $P_1TP_2^T$. The proof when cycle k is a C-cycle is similar. We can similarly show that $\underline{\ell}_{.k+1}$ is column k of the unique L-factor of $P_1TP_2^T$.

QED.

Corollary Algorithm 4.1 can be used to compute the following permuted factorization of T:

$$T = P_1^TLUP_2 \ . \qquad (4.6)$$

5. PIVOTED TOEPLITZ FACTORIZATION - AVOIDANCE OF INCREASE IN ERROR BOUND

We noted in the last chapter that in the pivoted Bareiss algorithm PBA, the error bound increase for step $(-k)$ to step $(-k-1)$ is proportional to $|t_{max}^{(-k)}/t_{piv}^{(-k)}|$, where $t_{piv}^{(-k)}$ is the pivot element. Algorithm 4.1 is merely a rearrangement of the outputs from PBA, so the error

* normalized l.c.: normalized such that the last coefficient is 1.

propagation is the same. Hence the possible large increase in error bound which occurs in the normal Toeplitz factorizer can be avoided by a judicious pivoting strategy.

6. TOEPLITZ INVERSION - INCREASE IN ERROR BOUND

In this Chapter, we have so far discussed the numerical aspects of Toeplitz factorization. We next consider error propagation in the Trench-Zohar algorithm (TZA) for Toeplitz inversion - later, we will propose a pivoting scheme to improve its numerical performance.

Cybenko [22] has performed an error analysis on TZA, and we do not repeat it here. He has found that for an indefinite Toeplitz matrix, TZA is unstable, i.e. the increase in error bound at any cycle k is unbounded. The following simple example illustrates this point. We work through TZA (Algorithm 3.2.1) in both exact arithmetic (unbarred quantities) and rounded arithmetic (barred quantities). Note that the matrix T being inverted is well-conditioned $(cond_{\infty}T = 9+O(\varepsilon)$, $\varepsilon \leq |\varepsilon_i|)$.

Example 6.1

$$T = \begin{bmatrix} 1 & 1-\varepsilon_1 & 0 \\ 1+\varepsilon_2 & 1 & 1-\varepsilon_1 \\ 0 & 1+\varepsilon_2 & 1 \end{bmatrix}, \qquad \begin{array}{l} \varepsilon_1-\varepsilon_2 = \alpha\mu,\ \varepsilon_1\varepsilon_2 = \beta\mu,\ |\beta| < 1. \\ |\varepsilon_1|^2 < 1,\ |\varepsilon_2|^2 < 1,\ \mu = \text{machine precision} \end{array}$$

$$\Rightarrow t_0 = 1,\ \underline{u}^T = [1-\varepsilon_1, 0], \underline{v}^T = [1+\varepsilon_2, 0]$$

Step 1. $\lambda_1 = \varepsilon_1-\varepsilon_2+\varepsilon_1\varepsilon_2 = (\alpha+\beta)\mu;\ \bar{\lambda}_1 = \varepsilon_1-\varepsilon_2 = \alpha\mu$, since $|\beta| < 1$.

Step 2. $\underline{c}_1 = \bar{\underline{c}}_1 = [\varepsilon_1-1]$.

Step 3. $\underline{d}_1 = \bar{d}_1 = [-1-\varepsilon_2]$.

Step 4.1, $i=1$. $\eta_1 = (1-\varepsilon_1)^2$; $\bar{\eta}_1 = (1-2\varepsilon_1)$.

Step 4.2, $i=1$. $\gamma_1 = (1+\varepsilon_2)^2$; $\bar{\gamma}_1 = (1+2\varepsilon_2)$.

Step 4.3, $i=1$. $\eta_1/\lambda_1 = (1-\varepsilon_1)^2/(\alpha+\beta)\mu$; $\overline{\eta_1/\lambda_1} = (1-2\varepsilon_1)/\alpha\mu$

$$\underline{e}_2 = \begin{bmatrix} \varepsilon_1-1 - \dfrac{(1-\varepsilon_1)^2(1+\varepsilon_2)}{(\alpha+\beta)\mu} \\ \\ (1-\varepsilon_1)^2/(\alpha+\beta)\mu \end{bmatrix}; \quad \bar{\underline{e}}_2 = \begin{bmatrix} \varepsilon_1-1 - \dfrac{(1-2\varepsilon_1)(1+\varepsilon_2)}{\alpha\mu} \\ \\ (1-2\varepsilon_1)/\alpha\mu \end{bmatrix}$$

Step 4.4, $i=1$. $\gamma_1/\lambda_1 = (1+\varepsilon_2)^2/(\alpha+\beta)\mu$; $\overline{\gamma_1/\lambda_1} = (1+2\varepsilon_2)/\alpha\mu$

$$\underline{d}_2^R = \begin{bmatrix} (1+\varepsilon_2)^2/(\alpha+\beta)\mu \\ \\ -1-\varepsilon_2 + \dfrac{(1+\varepsilon_2)^2(\varepsilon_1-1)}{(\alpha+\beta)\mu} \end{bmatrix}; \quad \bar{\underline{d}}_2^R = \begin{bmatrix} (1+2\varepsilon_2)/\alpha\mu \\ \\ -1-\varepsilon_2 + \dfrac{(1+2\varepsilon_2)(\varepsilon_1-1)}{(\alpha+\beta)\mu} \end{bmatrix}$$

Step 4.5, $i=1$. $\lambda_2 = (\alpha+\beta)\mu - \dfrac{(1-\varepsilon_1)^2(1+\varepsilon_2)^2}{(\alpha+\beta)\mu}$; $\bar{\lambda}_2 = \alpha\mu - \dfrac{(1-2\varepsilon_1)(1+2\varepsilon_2)}{\alpha\mu}$

So, ignoring second-order terms,

$$rel\ \bar{\lambda}_2 := \left|\frac{\bar{\lambda}_2-\lambda_2}{\lambda_2}\right| \doteq |\beta/\alpha| \doteq \overline{rel(T^{-1})}_{11}, \text{ since } (T^{-1})_{11} = 1/\lambda_2. \qquad (6.1)$$

Eq. (6.1) shows that as $\alpha \to 0$, $\overline{rel(T^{-1})}_{11} \to \infty$, i.e. the error at cycle 1 of step 4 is unbounded.

7. PIVOTED EXTENDED BAREISS SYMMETRIC ALGORITHM

Recall from Chapter 3, section 3 that the EBSA (algorithm 3.3.2) was derived from BSA (algorithm 3.3.1) by appending the calculation of the *multiplier matrices* $M^{(\pm i)}$, defined in eq. (3.3.5,6). Similarly, a *pivoted* EBSA (PEBSA) may be derived from PBSA (algorithm 5.5.1) by appending the calculation of the multiplier matrices (if BPC is used, the multiplier matrices are calculated anyway). All steps in PBSA (in fact, in any PBA) except reverse BNA steps have the form

$$T^{(\pm i)} \leftarrow T^{(\pm(i-1))} - m_{\pm i} C^{(\pm i)} S^{(\pm i)} T^{(\mp(i-\ell))}, \tag{7.1}$$

where $C^{(\pm i)}$ is a cyclic permutation matrix, $S^{(\pm i)}$ is a selection matrix, i.e. a diagonal matrix of the form $s_{jj}^{(\pm i)} = 1,\ j = j_1, \ldots, j_2$ and $s_{jj}^{(\pm i)} = 0$ elsewhere, and $\ell = 0$ or 1. Using $T^{(\pm i)} =: M^{(\pm i)}T$ in (7.1) and postmultiplying by T^{-1}, we get

$$M^{(\pm i)} \leftarrow M^{(\pm(i-1))} - m_{\pm i} C^{(\pm i)} S^{(\pm i)} M^{(\mp(i-\ell))} \tag{7.2}$$

So to convert PBSA to PEBSA, we simply add a step of the form (7.2) after every step of the form (7.1).

8. THE PIVOTED ALTERNATIVE BAREISS SYMMETRIC ALGORITHM AND PIVOTED TRENCH-ZOHAR ALGORITHM

In Chapter 3, we derived the alternative Bareiss Symmetric algorithm (ABSA, algorithm 3.4.1) which calculates the $m_{\pm i}$ and not the $T^{(\pm i)}$, from EBSA as follows:

(a) Drop all steps with $T^{(\pm i)}$ on the left-hand-side, i.e. steps of type (7.1).

(b) Let $n_{\pm i}$ and $d_{\pm i}$ be the numerator and denominator of $m_{\pm i}$ respectively.

Calculate $n_{\pm i}$ and $d_{\pm i}$ as follows:

(i) Check if $n_{\pm i}$ or $d_{\pm i}$ is available from the previous step.

(ii) Calculate $n_{\pm i}$ or $d_{\pm i}$ using the Bareiss recursion, if possible.

(iii) Otherwise, use the identity $\underline{m}_{p.}^{(\pm i)} \underline{t}_{.q} = t_{pq}^{(\pm i)}$. (8.1)

We can use the same technique to get a pivoted ABSA (PABSA) from PEBSA. Step (b) needs further explanation, as different operators are required for different types of pivoting steps.

<u>Calculation of $n_{\pm i}$, $d_{\pm i}$</u>

Let $t_A^{(\pm i)}$, $t_B^{(\pm i)}$ be elements in the first non-zero diagonal respectively above and below the zero-band of $T_*^{(\pm i)}$, the Toeplitz part of $T^{(\pm i)}$. Then, the following can easily be checked from the diagrams in Chapter 5:

	BSA	PBA-1 (steps 2.1 & 2.2)	A-cycles	B-cycles	C-cycles*
n_i	$t_B^{(1-i)}$	$t_A^{(1-i)}$	$t_A^{(1-i)}$	$t_B^{(1-i)}$	$t_B^{(1-i)}$
d_i	$t_B^{(i-1)}$	$t_A^{(i-1)}$	$t_A^{(i-1)}$	$t_B^{(i-1)}$	$t_B^{(i)}$
n_i	$t_A^{(i-1)}$	$t_B^{(i-1)}$	$t_A^{(i-1)}$	$t_A^{(i-1)}$	$t_B^{(i-1)}$
d_i	$t_A^{(1-i)}$	$t_B^{(-i)}$	$t_A^{(-i)}$	$t_A^{(-i)}$	$t_B^{(1-i)}$

Table 8.1: Numerators and Denominators of Multipliers for Various Pivoting Operations.

Table 8.2 shows how $t_A^{(\pm i)}$ and $t_B^{(\pm i)}$ can be calculated (the relation can be checked from the recursions in Chapter 5):

*For C-cycles, step (i) precedes step $(-i)$.

	BSA	PBA-1 (steps 1.1 & 1.2)	A-cycles	B-cycles	C-cycles	R-cycles
$t_A^{(-i)}$	$t_A^{(1-i)}-m_i t_A^{(i-1)}$	use (8.1)	use (8.1)	$t_A^{(1-i)}-m_{-i} t_A^{(i-1)}$	$t_A^{(1-i)}$	$t_A^{(-1-i)} m_{-1-i} t_A^{(i)}$
$t_B^{(-i)}$	use (8.1)	$t_B^{(1-i)}-m_{-i} t_B^{(i-1)}$	$t_B^{(1-i)}-m_{-i} t_B^{(i-1)}$	use (8.1)	use (8.1)	$m_{-i-1} t_B^{(i)}$
$t_A^{(i)}$	use (8.1)	$t_A^{(i-1)}$	use (8.1)	use (8.1)	$t_A^{(i-1)}-m_i t_A^{(1-i)}$	$m_{i+1} t_A^{(-i-1)}$
$t_B^{(i)}$	$t_B^{(i-1)}-m_i t_B^{(1-i)}$	use (8.1)	$t_B^{(i-1)}$	$t_B^{(i-1)}$	use (8.1)	$t_B^{(i+1)}$

Table 8.2: Calculation of $t_A^{(\pm i)}$ and $t_B^{(\pm i)}$ for various pivoting operations.

Tables 8.1 and 8.2 can be used to calculate the $n_{\pm i}$ and $d_{\pm i}$ at each step.

Derivation of PABSA directly from PBSA

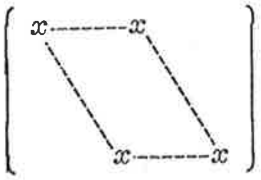
It is clear that we can combine the two derivations (PBSA → PEBSA and ΓEBSA → PABSA) as follows:

(a) *Replace* all steps of type (7.1) with steps of type (7.2).

(b) Calculate $n_{\pm i}$, $d_{\pm i}$ using Tables 8.1 and 8.2; $m_{\pm i} = n_{\pm i}/d_{\pm i}$.

PABSA - Compact Form

Note that in (7.2) the rows of $M^{(\pm i)}$ being changed make up a *Toeplitz block* of the form



,

so the first row completely specifies the block. Thus (7.2) can be written in terms of $\underline{\tilde{m}}^{(\pm i)T}$, the first row of the Toeplitz part of $M^{(\pm i)}$. If $C^{(\pm i)} = C_s$ and $S^{(\pm i)} = S_{ab}$, where C_s and S_{ab} are as defined in Chapter 5, eq. (5.2.8c,d), it can be checked that (7.2) can be written as follows:

$$\underline{\tilde{m}}^{(\pm i)T} = \underline{\tilde{m}}^{(\pm(i-1))T} C^T_{r_{\pm}-a+s} - m_{\pm i}\underline{\tilde{m}}^{(\mp(i-\ell))T} C^T_{r_{\mp}-a} , \qquad (8.2)$$

where $r_{\pm}$ is the index of the first row of the Toeplitz part of $M^{(\pm i)}$. The procedure for converting PBSA to the compact PABSA is now as follows:

Procedure 8.1

(a) Replace all steps of type (7.1) with steps of type (8.2).

(b) Calculate $n_{\pm i}$, $d_{\pm i}$ using Tables (8.1) and (8.2); $m_{\pm i}=n_{\pm i}/d_{\pm i}$.

2-choice Pivoted ABSA

We now apply procedure 8.1 to get a PABSA which corresponds to PBSA-1 (algorithm 5.5.1). PABSA-1 allows a choice of two pivots at each step *(k)*. We postpone discussion of the pivot selection strategy until section 10.

Algorithm 8.1 - PABSA-1 (Programmed on p.A.41)

```
1.  $\tilde{\underline{m}}^{(\pm 0)T} \leftarrow \underline{e}_1;\ t_0^{(0)} \leftarrow t_0;\ t_1^{(0)} \leftarrow t_1;\ t_{-1}^{(0)} \leftarrow t_{-1}$ .        (8.5a-d)
    {Normal ABSA loop, with pivot selection test}
2.  For $k \leftarrow 1$ to $n-1$ do
    2.1 $m_{-k} \leftarrow t_{-k}^{(1-k)}/t_0^{(k-1)}$                                   (8.6)
    2.2 $\tilde{\underline{m}}^{(-k)T} \leftarrow \tilde{\underline{m}}^{(1-k)T}C_{-1}-m_{-k}\tilde{\underline{m}}^{(k-1)T}$                (8.7)
    2.3 $t_0^{(-k)} \leftarrow t_0^{(1-k)}-m_{-k}t_k^{(k-1)}$; if $k \neq n-1$, $t_{-k-1}^{(-k)} \leftarrow \tilde{\underline{m}}^{(-k)}C_{-1}\underline{t}_{.1}$   (8.8)
    2.4 $m_k \leftarrow t_k^{(k-1)}/t_0^{(1-k)}$                                      (8.9)
    2.5 Select $t_0^{(-k)}$ or $t_{-k-1}^{(-k)}$ as pivot.  For the latter, do
        pivoting procedure $P$.
    2.6 $\tilde{\underline{m}}^{(k)T} \leftarrow \tilde{\underline{m}}^{(k-1)T}-m_k\tilde{\underline{m}}^{(1-k)T}C_{-1}$                  (8.10)
    2.7 $t_0^{(k)} \leftarrow t_0^{(k-1)}$; if $k \neq n-1$, $t_{k+1}^{(k)} \leftarrow \underline{m}^{(k)T}\underline{t}_{.k+2}$          (8.11)
    {Simple pivoting procedure $P$ - corresponds to step 2 of PBA-1}
        P1. $m_k \leftarrow t_0^{(k-1)}/t_{-k-1}^{(-k)}$                               (8.12)
        P2. $\tilde{\underline{m}}^{(k)T} \leftarrow \tilde{\underline{m}}^{(k-1)T}-m_k\tilde{\underline{m}}^{(-k)T}C_{-1}$; $t_{-1}^{(k)} \leftarrow \tilde{\underline{m}}^{(k)T}C_{-1}\underline{t}_{.1}$;
            $t_k^{(k)} \leftarrow t_k^{(k-1)}$                                       (8.13a-c)
        P3. $m_{-k-1} \leftarrow t_0^{(-k)}/t_k^{(k)}$                                 (8.14)
        P4. $\tilde{\underline{m}}^{(-k-1)T} \leftarrow \tilde{\underline{m}}^{(k)T}-m_{-k-1}\tilde{\underline{m}}^{(k)T}$; $t_{-k-1}^{(-k-1)} \leftarrow t_{-k-1}^{(-k)}-m_{-k-1}t_{-1}^{(k)}$;
            $t_{-1}^{(-k-1)} \leftarrow \tilde{\underline{m}}^{(-k-1)T}\underline{t}_{.k+2}$                       (8.15a-c)
            {RBF}
        P5. $m_{k+1} \leftarrow t_k^{(k)}/t_{-1}^{(-k-1)}$                              (8.16)
```

P6. $\underline{\tilde{m}}^{(k+1)T} \leftarrow \underline{\tilde{m}}^{(k)T}C_{-1}-m_{k+1}\underline{\tilde{m}}^{(-k-1)}$; $t_{-1}^{(k+1)} \leftarrow t_{-1}^{(k)}$;

$t_{k+1}^{(k+1)} \leftarrow \underline{m}^{(k+1)T}\underline{t}_{\cdot,k+3}$ (8.17a-c)

{Move $T_*^{(-k-1)}$ and $T_*^{(k+1)}$ to the bottom and top respectively.}

P7. $t_{-k-2}^{(-k-1)} \leftarrow t_{-k-1}^{(-k-1)}$; $t_0^{(-k-1)} \leftarrow t_1^{(-k-1)}$; $t_0^{(k+1)} \leftarrow t_{-1}^{(k+1)}$;

$t_{k+2}^{(k+1)} \leftarrow t_{k+1}^{(k+1)}$. (8.18a-d)

P8. Set $k \leftarrow k+2$, and return to step 2.1.

Multi-choice Pivoted ABSA

More general pivoting schemes, involving BPA, BPB and possibly BPC, may be incorporated by using procedures 7.1 to adapt the corresponding PBSA. We do not go into detail here.

Pivoted Trench-Zohar Algorithm (PTZA) (Programmed on p.A.41)

As remarked in Chapter 3, ABSA is just TZA with different notation. Hence PABSA is a pivoted version of TZA. PABSA can be formulated as a pivoted TZA by using the notational transformations in Chapter 3 (eqs. 3.4.15), with suitable modifications in the pivoting procedure. We will not elaborate this here.

Results

Table 8.3 show the results of applying (i) TZA and (ii) the 2-choice PTZA on a Toeplitz matrix with T_3 close to singular (adding $-5\times \times 10^{13}$ to t_{31} would make T_3 singular). TZA gives very inaccurate results, whereas the error in the 2-choice TZA is a moderate multiple of the machine precision.

```
SIZE OF ILL-COND BLOCK, DIST FROM SINGULARITY
?
3,5d-13


INPUT TOEPLITZ MATRIX T
     8.000     4.000     1.000     6.000     2.000     3.000
     4.000     8.000     4.000     1.000     6.000     2.000
   -34.000     4.000     8.000     4.000     1.000     6.000
     5.000   -34.000     4.000     8.000     4.000     1.000
     3.000     5.000   -34.000     4.000     8.000     4.000
     1.000     3.000     5.000   -34.000     4.000     8.000

*** RESULTS FOR TRENCH-ZOHAR ALG.***


INVERSE OF T
     0.042    -0.010    -0.022    -0.001    -0.004     0.004
     0.011     0.018    -0.002    -0.025    -0.002    -0.004
     0.002     0.030     0.005     0.002    -0.025    -0.001
     0.034    -0.001     0.010     0.005    -0.002    -0.022
    -0.131     0.173    -0.001     0.030     0.018    -0.010
     0.214    -0.131     0.034     0.002     0.011     0.042

ERROR MATRIX = T*T.INVERSE - IDENTITY
   -0.320D-01  -0.316D-01  -0.135D-01  -0.116D-01  -0.679D-02  -0.330D-02
   -0.605D-01  -0.160D-02  -0.110D-01  -0.979D-02  -0.575D-02  -0.122D-01
   -0.935D-01   0.224D-01   0.835D-02   0.397D-03  -0.371D-03  -0.123D-01
   -0.179D+00   0.956D-02   0.754D-01   0.215D-01   0.178D-01  -0.157D-01
    0.416D-01  -0.109D+00  -0.602D-01   0.871D-01   0.185D-01   0.250D-01
    0.126D+00  -0.118D+00  -0.929D-01  -0.832D-01   0.739D-01   0.250D-01

INFINITY-NORM OF ERROR MATRIX =     0.520D+00

*** RESULTS FOR PIVOTED TRENCH-ZOHAR ALG., STRATEGY 1 ***


INVERSE OF T
     0.041    -0.008    -0.021     0.000    -0.004     0.004
     0.008     0.018     0.000    -0.025    -0.001    -0.904
     0.006     0.028     0.003     0.004    -0.025     0.000
     0.039    -0.003     0.009     0.003     0.600    -0.021
    -0.120     0.170    -0.003     0.028     0.018    -0.008
     0.213    -0.120     0.039     0.006     0.008     0.041

ERROR MATRIX = T*T.INVERSE - IDENTITY
   -0.278D-16   0.222D-15   0.0          0.162D-15   0.651D-16   0.278D-16
    0.291D-15  -0.472D-15   0.111D-15  -0.226D-16   0.128D-15   0.971D-16
    0.444D-15  -0.208D-15   0.222D-15   0.165D-16  -0.173D-16   0.971D-16
   -0.167D-15  -0.555D-15   0.141D-15   0.0          0.295D-16  -0.668D-16
   -0.301D-14   0.555D-15   0.860D-15   0.191D-15   0.444D-15  -0.278D-15
    0.178D-14  -0.197D-14  -0.888D-15   0.112D-14   0.694D-16   0.666D-15

INFINITY-NORM OF ERROR MATRIX =     0.649D-14
READY
```

Table 8.3 - Inversion of a Toeplitz matrix with ill-conditioned leading submatrix of order 3 using TZA and 2-choice PTZA.

9. AVOIDANCE OF ERROR GROWTH IN PABSA FOR PTZA.

We perform an analysis similar to that for PBNA (Theorem 5.4.1) to show that the possible serious error growth in TZA, as evidenced by example 6.1, is avoided in PTZA (or PABSA). The result is:

Theorem 9.1 Let $rel\ \underline{\tilde{m}}^{(-k)}$, $rel\ \underline{\tilde{m}}^{(k-1)} rel\ t_0^{(k-1)}$ and $rel\ t_k^{(k-1)}$ be $\le \mu R$, $\varepsilon = |t_0^{(-k)}|$, $\alpha = |t_{-k-1}^{(-k)}|$, $\theta = \tilde{m}_{max}^{(-k)} \|T\|$. (9.1a-g)

Then: (i) $rel\ \underline{\tilde{m}}^{(k)} \le \mu R_8 \theta/\alpha$, where $R_8 := (1+\beta\frac{R}{R_1} cond\ T_k(\alpha/\theta)^2)/(1-\beta cond\ T_k \alpha/\theta)$

where $R_1 := R(1+2\alpha/\theta)$ and $\beta := \|T\|/\|T_k\|$

(ii) $rel\ \underline{\tilde{m}}^{(-k-1)} \le \mu\beta' R_6\ cond\ T_k^E \theta/\alpha$, where

$R_6 := R_5/(1 - \frac{\varepsilon\beta'}{\theta}\ cond\ T_k^E(\beta\ cond\ T_k + \theta/\alpha))$,

$R_5 := R_4(1 + \frac{R}{R_4} \cdot \frac{\alpha}{\theta} \cdot \frac{1}{\beta' cond T_k^E})$

$R_4 := R_3(1 + \beta cond T_k \alpha/\theta)$, $R_3 := R_2(1 + \varepsilon rel\ \underline{\tilde{m}}^{(k)}/\mu R_2 \theta)$,

$\beta' := \|T_k\|/\|T_k^E\|$, $R_2 := R(1 + \varepsilon/\theta)$

(iii) $rel\ t_1^{(-k-1)} \le \mu\beta^2\beta' R_7\ cond\ T_k^E(\frac{\theta}{\alpha}) \cdot cond\ T_k\ cond\ T_{k+2}$

where $R_7 := R_5/(1 - \beta\beta'\ cond\ T_k^E\ cond\ T_k\ \varepsilon/\alpha)$

Proof: We proceed by following the errors through in the same way as for PBNA (Theorem 5.4.1).

(i) From (7.8b) $\delta\alpha \le \sum_{i=1}^{k+1} \delta(\tilde{m}_i^{(-k)} t_{-i}) \le \mu R\theta$, where

$\theta := m_{max}^{(-k)} \|T\|$ and $\tilde{m}_{max}^{(-k)} := \max_i |\tilde{m}_i^{(-k)}|$ (9.2a-c)

Similarly $\delta\varepsilon \le \mu R\theta$ (9.2d)

Note that θ is an upper-bound for $t_{max}^{(-k)}$.

Applying a forward error analysis to (8.13a), and using (9.1b), (8.12), (9.1c), (9.1d) and (9.1a), we get

$$|\delta\tilde{m}_j^{(k)}| \le \mu(R\tilde{m}_{max}^{(k-1)} + |m_k \tilde{m}_{max}^{(-k)}| R_1\theta/\alpha) \qquad (9.3)$$

where $R_1 := R(1+2\alpha/\theta)$.

It is easily shown that $(T_k^{-1})_{1.} = \underline{\tilde{m}}^{(k-1)T}/t_0^{(k-1)}$, so

$$\tilde{m}_{max}^{(k-1)} \leq |t_0^{(k-1)}| cond\ T_k/\|T_k\| \qquad (9.4)$$

Also, using (9.2b) and (8.12)

$$|m_k \tilde{m}_{max}^{(-k)}| = \frac{|t_0^{(k-1)}\theta|}{\alpha\|T\|} \qquad (9.5)$$

Putting (9.4) and (9.5) in (9.3), we get

$$|\delta\tilde{m}_j^{(k)}| \leq \mu R_1(\theta/\alpha)|m_k \tilde{m}_{max}^{(-k)}|(1+\beta(R/R_1)\ cond\ T_k(\alpha/\theta)^2),$$

$$\text{where} \quad \beta. := \|T\|/\|T_k\| \qquad (9.6a,b)$$

Also from (8.13a)

$$\tilde{m}_{max}^{(k)} \geq |m_k \tilde{m}_{max}^{(-k)}| - \tilde{m}_{max}^{(k-1)} \qquad (9.7)$$

Putting (9.4) and (9.5) in (9.7) yields

$$\tilde{m}_{max}^{(k)} \geq |m_k \tilde{m}_{max}^{(-k)}|(1-\beta cond T_k(\alpha/\theta)) \qquad (9.8)$$

Result (i) follows from (9.6) and (9.8). QED(i)

(ii) We continue following the errors through step $(-k-1)$ as we did in Theorem 5.4.1, replacing the $t_j^{(\pm i)}$ by $m_j^{(\pm i)}$, L by θ/ε and δ by α/θ.

$$\delta(m_{-k-1}\tilde{m}_j^{(k)}) \leq |m_{-k-1}\tilde{m}_{max}^{(k)}| rel\ \bar{m}_{-k-1} + |m_{-k-1}\tilde{m}_j^{(k)}|\,|\delta\tilde{m}^{(k)}|_{max}/\tilde{m}_j^{(k)}$$

$$= |m_{-k-1}\tilde{m}_{max}^{(k)}| rel\ \bar{m}_{k-1}(1+rel\ \underline{\tilde{m}}^{(k)}/rel\ \bar{m}_{-k-1}) \qquad (9.9)$$

Eqs. (8.14), (9.2d) and (9.1d) give

$$rel\ \bar{m}_{-k-1} \leq \mu R_2\theta/\varepsilon, \ R_2 = R(1+\varepsilon/\theta) \qquad (9.10)$$

Putting (9.10) and (8.14) in (9.9), and using the fact (eq. 5.4.4) that $|t_k^{(k)}|=|t_k^{(k-1)}|\geq|t_0^{(k-1)}|/\beta' cond T_k^E$,

$$\beta' := \frac{\|T_k\|}{\|T_k^E\|} ,$$

$$\delta(m_{-k-1}\tilde{m}_j^{(k)}) \leq \mu R_3\beta'\theta\left(\frac{\tilde{m}^{(k)}_{max}}{|t_0^{(k-1)}|}\right) cond\ T_k^E ,$$

$$\text{where } R_3 := R_2(1+\varepsilon\ rel\ \underline{\tilde{m}}^{(k)}/\mu R_2\theta). \qquad (9.11a,b)$$

Using (9.4), (8.13a) and (8.12) we get

$$\tilde{m}^{(k)}_{max} \leq |t_0^{(k-1)}|\left(\frac{cond\ T_k}{\|T_k\|} + \frac{\tilde{m}^{(-k)}_{max}}{|\alpha|}\right) \qquad (9.12)$$

Putting this in (9.11a) and using (9.2b) we then get

$$\delta(m_{-k-1}\tilde{m}_j^{(k)}) \leq \mu R_4\beta'\ cond\ T_k^E\ \tilde{m}^{(-k)}_{max}\theta/\alpha ,$$

$$\text{where } R_4 := R_3(1+\beta\ cond\ T_k\alpha/\theta) \qquad (9.13)$$

So from (8.15a) and (9.13),

$$|\delta\tilde{m}_j^{(-k-1)}| \leq \mu R_5\beta'\ cond\ T_k^E\tilde{m}^{(-k)}_{max}(\theta/\alpha),$$

$$\text{where } R_5 := R_4(1 + \frac{R}{R_4}\cdot\frac{\alpha}{\theta}\cdot\frac{1}{\beta' cond T_k^E}) \qquad (9.14)$$

Also from (8.15a),

$$\tilde{m}^{(-k-1)}_{max} \geq \tilde{m}^{(-k)}_{max} - |m_{-k-1}\tilde{m}^{(k)}_{max}|$$

Using the definition of m_{-k-1} and majorizing $\tilde{m}^{(k)}_{max}$ using (9.12), and using (5.44) and (9.2b), we get

$$\tilde{m}^{(-k-1)}_{max} \geq \tilde{m}^{(-k)}_{max}(1-\frac{\varepsilon\beta'}{\theta}cond\ T_k^E(\beta\ cond\ T_k + \theta/\alpha)) \qquad (9.15)$$

Eqs. (9.14) and (9.15) give result (ii). QED(ii)

(iii) Using (8.15c), $\delta t_{-1}^{(-k-1)} \le \max_{j} |\delta \tilde{m}_j^{(-k-1)}| \; \Sigma t_{i,k+2}$

which, using (9.14),

$$\le \mu R_5 \beta' \, \mathit{cond}\, T_k^E \, \tilde{m}_{max}^{(-k)} (\theta/\alpha) \| T \| \text{, which, by lemma (4.2.3)}$$

$$\le \mu R_5 \beta\beta' \, \mathit{cond}\, T_k^E \; \mathit{cond}\, T_k \; (\theta/\alpha) \| T \| \qquad (9.16)$$

It can be seen from (4.2.7) that

$$|t_{-1}^{(-k-1)}|^{-1} \le \mathit{cond}\, T_{k+2} / (\tilde{m}_{k+1}^{(-k-1)} \| T_{k+2} \|) \qquad (9.17)$$

By working through algorithm 8.1, we have

$$\tilde{m}_{k+1}^{(-k-1)} = 1 + (t_0^{(k-1)} t_0^{(-k)}) / (\alpha t_k^{(k-1)}) \tilde{m}_k^{(-k)} \qquad (9.18)$$

By Lemma 4.2.3, $|\tilde{m}_k^{(-k)}| \le \beta \; \mathit{cond}\; T_k$, so eqs. (9.16), (9.17), (9.18) and (5.4.4) together give

$$\mathit{rel}\; t_{-1}^{(-k-1)} \le \mu \beta^2 \beta' R_7 \mathit{cond} T_k^E \mathit{cond} T_k \mathit{cond} T_{k+2} (\tfrac{\theta}{\alpha}),$$

$$\text{where } R_7 := R_5 / (1 - \frac{\varepsilon\beta\beta'}{\alpha} \mathit{cond}\; T_k^E \; \mathit{cond}\; T_k) \qquad \text{QED(iii)}$$

Discussion

As an example consider the case where T_k is well-conditioned but T_{k+1} is ill-conditioned, with $\mathit{cond}\; T_k = 10$, $\mathit{cond}\; T_k^E = 10$, $\mathit{cond}\; T_{k+1}^E = 10^5$, $\alpha/\theta = \frac{1}{20}$ and $\varepsilon/\theta = 10^{-5}$. Suppose also that β and β' are about unity. This will be true for well-balanced matrices. Thus $R_8 \doteq R_7 \doteq \ldots \doteq R_1 \doteq R$ (in general R_8 will be a modest multiple of R), so the *increase* in relative error bound from step $(-k)$ to step (k) is about $\theta/\alpha \doteq 20$, and from step $(-k)$ to step $(-k-1)$ is $\gamma/\alpha \cdot \mathit{cond}\; T_k^E = 200$. For TZA, the increase in relative error bound would be expected to be about $\theta/\varepsilon = 10^5$.

10. PIVOT SELECTION STRATEGY

Recall from Chapter 5 we selected a pivot $t_j^{(-k)}$ satisfying $|t_j^{(-k)}/t_{max}^{(-k)}| \geq \delta$. However, $t_{max}^{(-k)}$ is no longer available in PABSA. θ (defined in 9.1e) is a bound for $t_{max}^{(-k)}$, but it may not be very tight. If, we use *either* the 2-choice scheme *or* a multi-choice scheme with BPC for backtracking, we can use the *multipliers* in our pivot strategy. From the backward error analysis of BNA, we know that the growth in the perturbation matrix at each step $(\pm i)$ is bounded by $(1+|m_{\pm i}|)$. This result suggests that the following heuristic will work for the case in which a few $\{T_i\}$ are very ill-conditioned, and the rest are well-conditioned.

Procedure 10.1

```
1. Let $g$ be the target upper-bound on the growth in the
   perturbation matrix at each step.
2. Execute ABSA, until at step $(k)$, $(1+|m_k|) > g$, where
   $m_k = t_k^{(k-1)}/t_0^{(-k)}$.
3. For a 2-choice pivoting strategy, do 3A, else do 3B
   3A. Select $t_0^{(-k)}$ or $t_{-k-1}^{(-k)}$ to minimize the multiplier
   3B. For the more general pivoting procedure, execute
       the following loop
       3B.1 Set $p=0$, $q=0$.
       3B.2 Repeat
            3B.2.1 $p \leftarrow p+1$; $q \leftarrow q+1$
            3B.2.2 If $p=1$, move zero-bands up one place
                   using BPC. If $p > 1$, move zero-bands
                   of outputs of previous execution of
                   3B.2.2 up one place using BPC.
```

```
    3B.2.3 If $p=1$, move zero-bands of original
           $T^{(-k)}, T^{(k-1)}$ down a place using BPC.
           If $p > 1$, move zero-bands of outputs
           of previous execution of 3B.2.3 down a
           place using BPC.
    Until $|t_p^{(-k)}/t_{k+p}^{(k)}| \geq g-1$   or $|t_{-k-p}^{(-k)}/t_{-p}^{(k)}| \geq g-1$
                    or  $p = n-k$                    (10.1a-c)
3B.3 If (10.1a) is true, use $t_p^{(-k)}$, as pivot; if
     (10.1b) is true, use $t_{-p-1}^{(-k)}$ as pivot;
     otherwise use the pivot that minimizes the
     multiplier.
```

11. CONCLUSION

Error analyses have been performed on Toeplitz factorization algorithms showing that they are unstable; a simple example is given to show that Toeplitz inversion is also unstable. Pivoting has been incorporated into these algorithms, and error analyses and results show that the pivoted algorithms have a better numerical performance than the unpivoted algorithms.

CHAPTER 7

FAST TOEPLITZ ORTHOGONALIZATION

1. INTRODUCTION

We have so far considered ways of improving the numerical performance of $O(n^2)$ methods to solve the Toeplitz set of equations

$$T\underset{\sim}{x} = \underset{\sim}{b} \; . \tag{1.1}$$

We now pursue a different approach, which, however, has application to the accurate solution of (1.1). In Chapter 2, we solved (1.1) by first computing the *LU*-decomposition of *T*. Because of the structure of *T*, it might be expected that other common matrix decompositions could be calculated in $O(n^2)$ operations. In this chapter, we consider the orthogonal decomposition

$$T = QR \tag{1.2}$$

where Q is orthogonal, i.e. $Q^TQ = QQ^T = I$, and R is upper-triangular; we develop algorithms to calculate Q and R in $O(n^2)$ operations. The *QR*-decomposition of a Toeplitz matrix has received little attention in the literature, though some related work has been done [69] which solves the least-squares Toeplitz problem in $O(n^2)$ operations using lattice or ladder recursions. For the minimal design problem of control theory, Kung and Kailath [56] consider the case where T has the special form

$$T = \begin{bmatrix} \underline{t}_1 & \underline{t}_2 & \cdots & \underline{t}_k & & & \\ & \ddots & & & \ddots & & 0 \\ 0 & & \ddots & & & \ddots & \\ & & & \underline{t}_1 & \underline{t}_2 & \cdots & \underline{t}_k \end{bmatrix} \tag{1.3}$$

where the $\underline{t}_i$'s are vectors, and effectively compute the LQ decomposition (L lower-triangular), though L does not appear to be calculated explicitly, but via the relation

$$L = TQ^T$$

The matrix in (1.3) is part of a "vector circulant matrix". The present algorithms can be extended to block-Toeplitz matrices (Chapter 8) and we conjecture that they can be extended to vector-Toeplitz (and circulant) matrices as well. The present algorithm is described in terms only of elementary matrix operations, assumes no knowledge of control theory or polynomial matrix theory, which is required by [56]. Morf [65], in considering minimal realizations, calculates R (but not Q) for a general (block) Toeplitz matrix by performing a "fast" Cholesky factorization of T^TT. To see this, let $\bar{R}$ be the upper-triangular Cholesky factor of T, so that

$$T^TT = \bar{R}^T\bar{R} \tag{1.4}$$

But from (1.2) $T^TT = R^TQ^TQR = R^TR$ (1.5)

so that from (1.4), (1.5) and the uniqueness of the Cholesky factorization, $R=\bar{R}$. The algorithm described below calculates R(& Q) without forming T^TT. The present algorithms can of course be used to solve (1.1) and tests have shown that it is more stable than methods not involving orthogonal transformations. Other applications include the least-squares solution of Toeplitz systems which arise in the Covariance Method of linear prediction [60], [67], and in Prony's method for functional approximation [46].

In this chapter, Section 2 describes an algorithm by Gill, Golub, Murray and Saunders, which is used to develop the orthogonalization algorithms. In Section 3, the first orthogonalization algorithm, FT01 (FTO: fast Toeplitz orthogonalization) is presented. FT01 calculates only R explicitly; a method will then be given to solve (1.1) using R only. This method is a particular case of a technique for general matrices which Paige [71] has shown to be numerically stable. The second orthogonalization technique, FT02, which calculates Q and R explicitly, is presented in Section 4. (1.1) can easily be solved when T is factorized in this manner.

The next chapter will describe orthogonalization algorithms that are about one-third faster than FT01 and FT02, but are logically more complex. It will also be shown how to halve the operation count in all algorithms by using fast plane rotations [30] instead of the normal plane rotations which are the basis of the FTO algorithms.

2. THE GILL-GOLUB-MURRAY-SAUNDERS ALGORITHM (GGMS)

This algorithm [32] is needed for the derivation of the orthogonalization algorithms in Sections 3 and 4. The GGMS algorithm reduces a matrix which is the sum of an upper-triangular to a rank-1 matrix *(UT+R1)* to upper-triangular *(UT)* form. We first describe the algorithm, then give some properties of the algorithm, which will be needed later.

2.1 Description of the GGMS Algorithm

The GGMS algorithm reduces a *UT+R1* matrix to *UT* form in two phases: first from *UT+R1* to upper-Hessenberg *(UH)*, then from *UH* to *UT*. The process may be illustrated as follows:

GGMS Algorithm, Phase I: *UT+R1* → *UH*

The rows below the subdiagonal are eliminated from the last to the third using plane rotations. A typical stage of the reduction for a 7x7 matrix is

$$\begin{matrix} x & x & x & x & x & x & x \\ + & x & x & x & x & x & x \\ + & + & x & x & x & x & x \\ + & + & + & x & x & x & x \\ \textcircled{+} & + & + & + & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & x \end{matrix}$$

Here, +'s denote the elements of a rank-1 block. If we choose a rotation in the (4,5) plane to null the (5,1) element, clearly the (5,2) and (5,3) elements will also be nulled by the same plane rotation since they are part of a rank-1 block. Thus after this plane rotation, we have

$$\begin{array}{ccccccc} x & x & x & x & x & x & x \\ + & x & x & x & x & x & x \\ + & + & x & x & x & x & x \\ + & + & + & x & x & x & x \\ 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & x \end{array}$$

Thus rows 7,6,...,3 may be eliminated below the subdiagonal by suitably-chosen plane rotations in the (6,7),(5,6),...,(2,3) planes respectively.

GGMS Algorithm, Phase II: *UH* → *UT*

The subdiagonal elements are eliminated from the top-left to the bottom-right, using plane rotations. A typical stage of the reduction for a 7x7 matrix is

$$\begin{array}{ccccccc} x & x & x & x & x & x & x \\ 0 & x & x & x & x & x & x \\ & 0 & {}^{\curvearrowright}x & x & x & x & x \\ & & \textcircled{x} & x & x & x & x \\ & & & x & x & x & x \\ & & & & x & x & x \\ & & & & & x & x \end{array}$$

The (4,3) element can be eliminated by a suitable rotation in the (3,4) plane. Thus, elements (2,1),(3,2),...,(7,6) may be eliminated by appropriate rotations in the (1,2),(2,3),...,(6,7) planes respectively. The general GGMS algorithm is stated in Algorithm 2.1.

Algorithm 2.1 - The GGMS Algorithm

Notation - We denote $G(\hat{\phi}_i)$ as the plane rotation matrix applied in the $(i,i+1)$ plane in phase I, and $G(\check{\phi}_i)$ as the corresponding plane rotation matrix in phase II. We denote $A^{\hat{i}}$ and $A^{\check{i}}$ as the transformed matrices after the application of $G(\hat{\phi}_i)$ and $G(\check{\phi}_i)$ respectively. This terminology is similar to Wilkinson's $A^{(i)}$ in Gaussian elimination, [89] which is the transformed matrix after the ith column has been eliminated.

```
Input      The $n \times n$ matrix $A$ of the form $UT+R1$.
Procedure  {Phase I - $UT+R1 \rightarrow UH$}
    1. Set $A^{\hat{n}} \leftarrow A$.
    2. For $i \leftarrow n-1$ downto $2$ do {elim row $i+1$ below subdiagonal using plane rotation $G(\hat{\phi}_i)$}
        2.1 $\cos \hat{\phi}_i \leftarrow a_{i1}^{\hat{i+1}}/r$ ; $\sin \hat{\phi}_i \leftarrow a_{i+1,1}^{\hat{i+1}}/r$ ,
            where $r \leftarrow \sqrt{(a_{i1}^{\hat{i+1}})^2+(a_{i+1,1}^{\hat{i+1}})^2}$
        2.2 $A^{\hat{i}} \leftarrow G(\hat{\phi}_i)A^{\hat{i+1}}$
    {Phase II - $UH \rightarrow UT$}
    3. Set $A^{\check{0}} \leftarrow A^{\hat{2}}$
    4. For $i \leftarrow 1$ to $n-1$ do {eliminate $(i+1,i)$ element using plane rotation $G(\check{\phi}_i)$}
        4.1 $\cos \check{\phi}_i \leftarrow a_{ii}^{\check{i-1}}/r$ ; $\sin \check{\phi}_i \leftarrow a_{i+1,i}^{\check{i-1}}/r$ ,
            where $r \leftarrow \sqrt{(a_{ii}^{\check{i-1}})^2 + (a_{i+1,i}^{\check{i-1}})^2}$
        4.2 $A^{\check{i}} \leftarrow G(\check{\phi}_i)\ A^{\check{i-1}}$
```

Remark   It may be seen that the GGMS algorithm requires $(2n-3)$ plane rotations in all. These rotations require $4n^2 + O(n)$ operations to execute.

2.2 Miscellaneous results - GGMS algorithm

The following properties of the GGMS algorithm will be required later:

Lemma 2.1 The product of the plane rotations that triangularizes A in the GGMS algorithm is

$$S^T := \prod_{i=1}^{n-1} G(\check{\phi}_i) \prod_{j=n-1}^{2} G(\hat{\phi}_j)$$

where the products indicated by Π are defined by

$$\prod_{i=1}^{k} A_i = A_k A_{k-1} \cdots A_1$$

Proof Trivial

Definition Define the partial products

$$S^T(\hat{\phi}_i) := \prod_{j=n-1}^{i} G(\hat{\phi}_j)$$

$$S^T(\check{\phi}_i) := \prod_{j=2}^{i} G(\check{\phi}_j)\, S^T(\hat{\phi}_2) \,.$$

$S^T(\hat{\phi}_i)$ and $S^T(\check{\phi}_i)$ may be considered to be the product of all the plane rotations up to and including $G(\hat{\phi}_i)$ and $G(\check{\phi}_i)$ respectively.

Clearly $S^T(\check{\phi}_{n-1}) = S^T$, $S^T(\hat{\phi}_i)A = A^{\hat{i}}$ and $S^T(\check{\phi}_i) = A^{\check{i}}$.

The following may then be shown:

Lemma 2.2

$$a^{\hat{i}}_{i1} = \| \underline{a}_{i:n,1} \| \qquad (2.1)^*$$

Proof The plane rotations $G(\hat{\phi}_{n-1}), G(\hat{\phi}_{n-2}), \ldots, G(\hat{\phi}_i)$ modify only rows i to n, so the norm of these rows is invariant, i.e.

$$\| \underline{a}^{\hat{i}}_{i:n,1} \| = \| \underline{a}_{i:n,1} \| \qquad (2.2)$$

* $\underline{x}_{i:j,k}$ denotes elements i to j of the kth column of X.

Also in GGMS, $G(\hat{\phi}_{n-1}),\ldots,G(\hat{\phi}_i)$ respectively eliminate elements $n,\ldots,i+1$ of the first column (see comment on Algorithm 2.1, step 1), so

$$a_{k1}^{\hat{i}} = 0, \qquad k=i+1,\ldots,n$$

$$\therefore \quad \| \underline{a}_{i:n,1}^{\hat{i}} \| = a_{i1}^{\hat{i}} \quad (positive) \tag{2.3}$$

and (2.2) and (2.3) ⇒ (2.1). QED.

Lemma 2.3 $\cos\hat{\phi}_i = a_{i1}/\|\underline{a}_{i:n,1}\|$; $\sin\hat{\phi}_i = \|\underline{a}_{i+1:n,1}\|/\|\underline{a}_{i:n,1}\|$

Proof In step 2.1 of the GGMS algorithm, iteration i,

$$a_{i1}^{\hat{i+1}} = a_{i1} \tag{2.4}$$

because $G(\hat{\phi}_{n-1}),\ldots,G(\hat{\phi}_{i+1})$ do not modify row i. Also, from Lemma 2.2,

$$a_{i+1,1}^{\hat{i+1}} = \|a_{i+1:n,1}\|, \tag{2.5}$$

and $\sqrt{(a_{i1}^{\hat{i+1}})^2+(a_{i+1,1}^{\hat{i+1}})^2} = \sqrt{a_{i1}^2+\|\underline{a}_{i+1:n,1}\|^2} = \|\underline{a}_{i:n,1}\|$,
(2.6)

using (2.4) and (2.5). Putting (2.4)-(2.6) into step 1 of algorithm 2.1 yields the result. QED.

Lemma 2.4 $a_{ij}^{\hat{i}} = \dfrac{a_{ij}}{a_{i1}} \|\underline{a}_{i:n,1}\|$, $\quad i > j$

Proof The plane rotations $G(\hat{\phi}_{n-1}),\ldots,G(\hat{\phi}_i)$ only modify rows i to n, so the norm of these rows is invariant, i.e.

$$\|\underline{a}_{i:n,j}^{\hat{i}}\| = \|\underline{a}_{i:n,j}\|$$

$$= \frac{a_{ij}}{a_{i1}} \|\underline{a}_{i:n,1}\| , \tag{2.7}$$

because the elements of A below the diagonal are part of a rank-1 matrix. Also in GGMS $G(\hat\phi_{n-1}), G(\hat\phi_{n-2}), \ldots, G(\hat\phi_i)$ successively eliminate elements $n, n-1, \ldots, i+1$ in the jth column $(i>j)$ (see comment in step 1), so

$$a^{\hat{i}}_{kj} = 0, \qquad k=i+1,\ldots,n$$

so $$\|\underline{a}^{\hat{i}}_{i:n,j}\| = a^{\hat{i}}_{ij}\ , \tag{2.8}$$

and the result follows from (2.7) and (2.8). QED.

Lemma 2.5 $$S^T\underline{a}_{\cdot j} = \prod_{i=1}^{j} G(\check\phi_i) \prod_{i=j}^{2} G(\hat\phi_i) \begin{bmatrix} \underline{a}_{1:j,j} \\ \alpha \\ \underline{0} \end{bmatrix},$$

where $$\alpha = \frac{a_{j+1,j}}{a_{j1}}\ \|\underline{a}_{j+1:n,1}\|$$

Proof $$\underline{a}^{\hat{j+1}}_{\cdot j} := S^T(\hat\phi_{j+1})\underline{a}_{\cdot j} = \begin{bmatrix} \underline{a}_{1:j,j} \\ \alpha \\ 0 \end{bmatrix}, \tag{2.9}$$

because (i) the plane rotations that make up $S^T(\hat\phi_{j+1})$ do not modify rows 1 to j (ii) $\alpha = a^{\hat{j+1}}_{j+1,j}$ by Lemma 2.4 (iii) the plane rotations in $S^T(\hat\phi_{j+1})$ respectively eliminate elements $n, n-1, \ldots, j+2$.

Also $$a^{\check{i}}_{j+1,j} = 0 \tag{2.10}$$

because in step 4.1 of algorithm 2.1, $\check\phi_j$ is selected to eliminate the $j+1,j$ element,

and $$\underline{a}^{\check{j}}_{j+2:n} = \underline{a}^{\hat{j}}_{j+2:n}$$

because the plane rotations $G(\hat\phi_{j-1}), \ldots, G(\check\phi_{j-1})$ do not modify rows $j+1$ to n

$$= \underline{0} \text{ using (2.9)} \qquad (2.11)$$

so because of (2.10) and (2.11),

$$S^T \underline{a}_{\cdot j} = S^T(\check{\phi}_{n-1})\underline{a}_{\cdot j} = \underline{a}^{\check{j}}_{\cdot j} =: S^T(\check{\phi}_j)\underline{a}_{\cdot j} \qquad (2.12)$$

$$= \prod_{i=2}^{j} G(\check{\phi}_i) \prod_{i'=j}^{2} G(\hat{\phi}_{i'}) \, S^T(\hat{\phi}_{j+1}) \, \underline{a}_{\cdot j}$$

$$= \prod_{i=2}^{j} G(\check{\phi}_i) \prod_{i=j}^{2} G(\hat{\phi}_{i'}) \begin{pmatrix} \underline{a}_{1:j,j} \\ \alpha \\ 0 \end{pmatrix} \text{, using (2.9)}$$

QED.

3. FAST TOEPLITZ ORTHOGONALIZATION, VERSION 1 (FTO 1)

Recall that we wish to find Q orthogonal and R upper-triangular such that

$$T = QR \qquad (3.1)$$

In the algorithm to be described in this section, only R is calculated explicitly. It will then be shown how $A\underline{x} = \underline{b}$ may be solved using R only. In the next section it will be shown how to generate Q in $O(n^2)$ operations, if this is desired.

3.1 Outline of the Algorithm

We give a brief outline of FTO 1, pointing out how R may be calculated in $O(n^2)$ operations. We present only the main idea in this subsection, leaving the mathematical details to the following subsections.

Let T have order n, and denote the leading principal submatrix of T as T_{n-1}. Let $T_{n-1} = Q^{(n-1)}R^{(n-1)}$, where $Q^{(n-1)}$ is orthogonal and $R^{(n-1)}$ is upper-triangular. Then, the following will be shown later:

(a) If T is Toeplitz, a certain matrix of the form

$$\left(\begin{array}{c|ccc} x & x & --- & x \\ \hline x & & & \\ \cdot & & & \\ \cdot & & R^{(n-1)} & \\ \cdot & & & \\ x & & & \end{array}\right)$$

may be converted to the upper-triangular matrix R by the application of the GGMS algorithm, which uses $(2n-3)$ plane rotations.

(b) $R+C$, where C is a certain rank-1 matrix, may be converted to a matrix of the form

$$\left(\begin{array}{c|c} & x \\ R^{(n-1)} & \cdot \\ & \cdot \\ & \cdot \\ & x \\ \hline \underline{0}^T & x \end{array}\right)$$

by another application of the GGMS algorithm.

The operations in (a) and (b) may be written compactly as

$$\left(\begin{array}{c|ccc} x & x & --- & x \\ \hline x & & & \\ \cdot & & & \\ \cdot & & R^{(n-1)} & \\ \cdot & & & \\ x & & & \end{array}\right) \xrightarrow[\text{rotations}]{\substack{\text{GGMS}\\ \text{plane}}} R \xrightarrow[\substack{C \\ \text{(rank-1)}}]{\text{add}} R+C \xrightarrow[\text{rotations}]{\substack{\text{GGMS}\\ \text{plane}}} \left(\begin{array}{c|c} & x \\ R^{(n-1)} & x \\ & \cdot \\ & \cdot \\ & \cdot \\ \hline & x \end{array}\right) \tag{3.2}$$

Now the transformations in (3.2) require $O(n^2)$ operations, or $O(n)$ operations per column. The essential property is that because T is Toeplitz, a matrix with $R^{(n-1)}$ as the trailing submatrix can be converted to another with $R^{(n-1)}$ as the leading submatrix in $O(n^2)$ operations. This property enables us to calculate $R^{(n-1)}$ and (as a by product) R in $O(n^2)$ operations: writing out only the kth column*

*$\underline{x}_{\cdot k}$ denotes the kth column of a matrix X.

of (3.2),

$$\begin{bmatrix} \underline{x} \; - \\ \\ \underline{r}_{\cdot k-1}^{(n-1)} \end{bmatrix} \xrightarrow[\text{rotations}]{\text{GGMS plane}} \underline{r}_{\cdot k} \xrightarrow[\underline{c}_{\cdot k}]{\text{add}} (R+C)_{\cdot k} \xrightarrow[\text{rotations}]{\text{GGMS plane}} \begin{bmatrix} \underline{r}_{\cdot k}^{(n-1)} \\ \\ \overline{0} \end{bmatrix}, \qquad (3.3)$$

we see that (3.3) suggests a recursion for calculating $\underline{r}_{\cdot k}^{(n-1)}$ from $\underline{r}_{\cdot k-1}^{(n-1)}$. The plane rotations in (3.3) require only $O(n)$ operations, so all of $R^{(n-1)}$, and as a by-product, R, may be calculated in $O(n^2)$ operations. The complete procedure is summarized below:

1. Initialize $\underline{r}_{\cdot 1}$ and $\underline{r}_{\cdot 1}^{(n-1)}$ (easily done)
2. $k \leftarrow 2$

{Main loop}

3. Convert $\begin{bmatrix} \underline{x} \; - \\ \underline{r}_{\cdot k-1}^{(n-1)} \end{bmatrix}$ to $\underline{r}_{\cdot k}$ by plane rotations
4. Stop if $k{=}n$.
5. Add $\underline{c}_{\cdot k}$ to $\underline{r}_{\cdot k}$ (C is a rank-1 matrix)
6. Convert $(R{+}C)_{\cdot k}$ to $\begin{bmatrix} \underline{r}_{\cdot k}^{(n-1)} \\ 0 \end{bmatrix}$ by plane rotations.
7. $k \leftarrow k{+}1$
8. Go to step 3.

Remark Several details, such as the calculation of the rotation matrices, the 'x's in step 3 and $\underline{c}_{\cdot k}$ in step 5 have not been discussed here. This will be done in the following subsections, when the algorithm is derived in detail. It will be seen that the computation of these quantities is only a minor part of the total work of the algorithm.

3.2 Preliminary Results

The following results are basic to FTO. Here, the matrices and transformations in (3.2) are derived.

Define $\dot{Q} := \left(\begin{array}{c|c} 1 & \underline{0}^T \\ \hline \underline{0} & Q^{(n-1)} \end{array}\right)$

and let T be partitioned

$$T =: \left(\begin{array}{c|c} t & \underline{u}^T \\ \hline \underline{v} & \dot{T}_{n-1} \end{array}\right)$$

$$= \left(\begin{array}{c|c} t & \underline{u}^T \\ \hline \underline{v} & T_{n-1} \end{array}\right) \quad \text{because } T \text{ is Toeplitz} \tag{3.4}$$

Then

$$\dot{Q}^T T = \left(\begin{array}{c|c} t & \underline{u}^T \\ \hline Q^{(n-1)T}\underline{v} & R^{(n-1)} \end{array}\right) =: \dot{R} \tag{3.5}$$

$\dot{R}$ is the matrix on the left of (3.2). Since $\dot{R}$ has the form $UT{+}R1$, the GGMS algorithm can be used to transform it to R, an UT matrix, using $2n-3$ plane rotations. Let $\dot{S}^T$ be the product of these plane rotations. Then

$$\boxed{\dot{S}^T\dot{R} = R} \tag{3.6}$$

The GGMS algorithm was described in the last section. We now assert that R is the upper-triangular matrix required in (3.1), because from (3.5)

$$T = \dot{Q}\dot{R} = (\dot{Q}\dot{S})R\ ,$$

an orthogonal decomposition of T. The transformation (3.6) is the first operation in (3.2).

Next define
$$\tilde{Q} := \left(\begin{array}{c|c} Q^{(n-1)} & \underline{0} \\ \hline \underline{0}^T & 1 \end{array}\right).$$

Because T is Toeplitz, it may be partitioned
$$T = \left(\begin{array}{c|c} T_{n-1} & \underline{u}^R \\ \hline \underline{v}^{RT} & t \end{array}\right),$$

where $\underline{u}$, $\underline{v}$ and t are as in (2.4) and $\underline{x}^R$ denotes $\underline{x}$ with its elements reversed.

Then
$$\tilde{Q}^T T = \left(\begin{array}{c|c} R^{(n-1)} & Q^{(n-1)^T}\underline{u}^R \\ \hline \underline{v}^{RT} & t \end{array}\right)$$

$$= \bar{R} + \underline{e}_n(\underline{v}^{RT},0), \text{ where } \underline{e}_n := [0,\ldots,0,1]$$

$$\text{and} \quad \bar{R} := \left(\begin{array}{c|c} R^{(n-1)} & Q^{(n-1)^T}\underline{u}^R \\ \hline \underline{0}^T & t \end{array}\right)$$

so
$$\bar{R} = \tilde{Q}^T T - \underline{e}_n(v^{RT},0) = \tilde{Q}^T(T - \tilde{Q}\underline{e}_n(\underline{v}^{RT},0))$$
$$= \tilde{Q}^T(T - \underline{e}_n(\underline{v}^{RT},0)) = \tilde{Q}^T(QR - \underline{e}_n(\underline{v}^{RT},0))$$
$$= \tilde{Q}^T Q(R - Q^T\underline{e}_n(\underline{v}^{RT},0)) = \tilde{Q}^T Q(R - \underline{q}^T_{n.}(\underline{v}^{RT},0))$$
$$= \tilde{Q}^T Q(R+C), \text{ where } C := -\underline{q}^T_{n.}(\underline{v}^{RT},0) \qquad (3.7)$$

Thus, (3.7) states that $R+C$, where C is a rank-one matrix, may be transformed to $\bar{R}$ by an orthogonal transformation, since $\tilde{Q}$ and Q are both orthogonal. The GGMS algorithm can be used again to transform $R+C$ to $\bar{R}$ using $2n-3$ plane rotations. If $\tilde{S}^T$ is the product of these plane rotations,

$$\boxed{\tilde{S}^T\tilde{R} = \bar{R}} \quad \text{where} \quad \tilde{R} := R+C = R - \underline{q}_{n.}^T\,(\underline{v}^{RT},0) \qquad (3.8)$$

The rank-1 matrix C is that added in the second operation in (2.2), and the transformation (3.7) is the third operation in (3.2). Rewriting (3.2) using (3.5) and (3.7), we get

$$\underbrace{\left[\begin{array}{c|c} t & \underline{u}^T \\ \hline Q^{(n-1)T}\underline{v} & R^{(n-1)} \end{array}\right]}_{\dot{R}} \xrightarrow[\dot{S}^T]{\text{GGMS plane rotations}} \left[R\right] \xrightarrow[C=-\underline{q}_{n.}^T\,(\underline{v}^{RT},0)]{\text{add}} \underbrace{\left[R+C\right]}_{\tilde{R}} \xrightarrow[\tilde{S}^T]{\text{GGMS plane rotations}} \underbrace{\left[\begin{array}{c|c} R^{(n-1)} & Q^{(n-1)}\underline{u}^R \\ \hline \underline{0}^T & t \end{array}\right]}_{\bar{R}} \qquad (3.9)$$

The operations in (3.9) form the basis of FTO.

3.3 FTO1 - The Main Recursion

In subsection 3.1, we showed that writing out the kth column of (3.2) yielded a recursion for the columns of $R^{(n-1)}$. (3.9) is a detailed version of (3.2), so writing out the kth column of (3.9), we have

$$\underbrace{\begin{bmatrix} u_{k-1} \\ \\ \underline{r}_{\cdot k-1}^{(n-1)} \end{bmatrix}}_{\dot{\underline{r}}_{\cdot k}} \xrightarrow[\dot{S}^T]{\text{GGMS rotations}} \begin{bmatrix}\underline{r}_{\cdot k}\end{bmatrix} \xrightarrow[\underline{c}_{\cdot k}=-\underline{q}_{n.}^T\, v_{n-k}]{\text{add}} \underbrace{\begin{bmatrix}\underline{r}_{\cdot k}+\underline{c}_{\cdot k}\end{bmatrix}}_{\tilde{\underline{r}}_{\cdot k}} \xrightarrow[\tilde{S}^T]{\text{GGMS rotations}} \underbrace{\begin{bmatrix} \underline{r}_{\cdot k}^{(n-1)} \\ \\ 0 \end{bmatrix}}_{\bar{\underline{r}}_{\cdot k}} \qquad (3.10)$$

The transformations in (3.10) constitute a recursion by which each column of $R^{(n-1)}$ and R can be calculated in $O(n)$ operations. However, (3.10) cannot be used in its present form for Toeplitz orthogonalization, because the GGMS plane rotations and the elements of $\underline{q}_{n.}$ cannot all be calculated with the available information. However, by using Lemma 2.5, (3.10) can be transformed to a form which uses a subset of the GGMS plane rotations and elements of $\underline{q}_{n.}$ that can be easily calculated from known quantities.

Doing this, we let $\{\hat{\phi}_i\}_{n-1}^{2}$, $\{\check{\phi}_i\}_{1}^{n-1}$ be the plane rotations that make up $\dot{S}^T$. Applying Lemma 2.5, the first transformation in (3.10) can be written

$$\underline{r}_{\cdot k} \leftarrow \prod_{i=1}^{k} G(\check{\phi}_i) \prod_{i=k}^{2} G(\hat{\phi}_i) \overbrace{\begin{bmatrix} u_{k-1} \\ \underline{r}_{\cdot k}^{(n-1)} \end{bmatrix}}^{\dot{\underline{r}}_{\cdot k}}$$

$$=: S^T(\hat{\phi}_k, \check{\phi}_k) \begin{bmatrix} u_{k-1} \\ \underline{r}_{\cdot k}^{(n-1)} \end{bmatrix} , \qquad (3.11a)$$

where $S^T(\hat{\phi}_k, \check{\phi}_k)$ denotes the product of all the plane rotations from $G(\hat{\phi}_k)$ to $G(\check{\phi}_k)$.

Now, let $\{\hat{\theta}_i\}_{n-1}^{2}$, $\{\check{\theta}_i\}_{1}^{n-1}$ be the plane rotations that make up $\tilde{S}^T$. Again applying Lemma 2.5, the last transformation in (3.10) can be written

$$\begin{bmatrix} \overbrace{\underline{r}_{\cdot k}^{(n-1)}}^{\bar{\underline{r}}_{\cdot k}} \\ \\ 0 \end{bmatrix} \leftarrow \prod_{i=1}^{k} G(\check{\theta}_i) \prod_{i=k}^{2} G(\hat{\theta}_i) \begin{bmatrix} \underline{r}_{1:k,k} - v_{n-k}\, \underline{q}_{n,1:k}^{T} \\ \alpha_k \\ 0 \end{bmatrix} ,$$

where $\alpha_k := -v_{n-k} \| \underline{q}_{n,k+1:n} \| = -v_{n-k}\sqrt{1-\| \underline{q}_{n,1:k}\|^2}$.

As in (3.11a), we rewrite this transformation as

$$\begin{bmatrix} \overbrace{\underline{r}_{\cdot k}^{(n-1)}}^{\bar{\underline{r}}_{\cdot k}} \\ \\ 0 \end{bmatrix} \leftarrow S^T(\hat{\theta}_k, \check{\theta}_k) \begin{bmatrix} \underline{r}_{1:k,k} - v_{n-k}\, \underline{q}_{n,1:k}^{T} \\ \alpha_k \\ 0 \end{bmatrix} \qquad (3.11b)$$

where $S^T(\hat{\theta}_k, \check{\theta}_k)$ denotes the product of all the plane rotations from $G(\hat{\theta}_k)$ to $G(\check{\theta}_k)$.

The transformations (3.11) can be written compactly as

$$\overbrace{\begin{bmatrix} u_k \\ \\ \underline{r}_{\cdot k}^{(n-1)} \end{bmatrix}}^{\underline{r}_{\cdot k}} \xrightarrow[S^T(\hat{\phi}_k,\check{\phi}_k)]{\text{plane rotations}} \left(\underline{r}_{\cdot k}\right) \xrightarrow[(-v_{n-k}\,\underline{q}_{n,1:k}^T,\alpha_k,\underline{0}^T)^T]{\text{add}} \begin{bmatrix} \underline{r}_{1:k,k}-v_{n-k}\,\underline{q}_{n,1:k}^T \\ \alpha_k \\ 0 \end{bmatrix} \longrightarrow$$

$$\xrightarrow[S^T(\hat{\theta}_k,\check{\theta}_k)]{\text{plane rotations}} \overbrace{\begin{bmatrix} \underline{r}_{\cdot k}^{(n-1)} \\ \\ \underline{0} \end{bmatrix}}^{\bar{\underline{r}}_{\cdot k}} \qquad (3.12)$$

The transformations in (3.12) again constitute a recursion for the columns of $R^{(n-1)}$ and R, but only $\{q_{ni}, \hat{\phi}_i, \check{\phi}_i, \hat{\theta}_i, \check{\theta}_i\}_{i\leqslant k}$ are required, rather than all of the GGMS plane rotations as needed by (3.10). This subset of the $\{q_{ni}, \hat{\phi}_i, \check{\phi}_i, \hat{\theta}_i, \check{\theta}_i\}$ is easily calculated from known quantities. This is most conveniently done by presenting FT01, and proving the relevant formulae.

3.4 FT01 - The Algorithm

The algorithm in outline is, as we have seen:

1. Initialize $\underline{r}_{\cdot 1}$, $\underline{r}_{\cdot 1}^{(n-1)}$, $\check{\phi}_1$, $\check{\phi}_1$ and q_{n1}. (Note - there is no $\hat{\phi}_1$ and $\hat{\theta}_1$).

2. For $k \leftarrow 2$ to n do

   2.1 Calculate $\hat{\phi}_k$, $\check{\phi}_k$, $\hat{\theta}_k$, $\check{\theta}_k$ and q_{nk} from known quantities.

   2.2 Calculate $\underline{r}_{\cdot k}$ and $\underline{r}_{\cdot k}^{(n-1)}$. using the recursion (3.12) {end of outline}.

The algorithm in detail is presented and proved in Theorem 3.1.

Theorem 3.1 Let T be an $n \times n$ Toeplitz matrix, and let Q orthogonal and R upper triangular such that $T=QR$. Then R may be calculated in $O(n^2)$ operationsby the following algorithm, where all intermediate quantities (except $\sigma_i := \|\dot{\underline{r}}_{i:n,1}\|$, $\tilde{\sigma}_i := \|\underline{q}_{i:n,1}\|$) are as defined in the foregoing:

Algorithm 3.1 - FT01 (Programmed on p.A.48)

```
{Initialization}
1.   $r_{11}^{(n-1)} \leftarrow \|\underline{t}_{1:n-1,1}\|$
2.   $r_{11} \leftarrow \sqrt{r_{11}^{(n-1)2} + t_{n1}^2}$
3.   $\dot{r}_{11} \leftarrow t_{11}$                    {$\dot{R}$ is as in eq.(3.5)}
4.   $\dot{\sigma}_1 \leftarrow r_{11}$                  {$\dot{\sigma}_i := \|\dot{\underline{r}}_{i:n,1}\|$}
5.   $\dot{\sigma}_2 \leftarrow \sqrt{r_{11}^2 - \dot{r}_{11}^2}$
6.   $\cos \check{\phi}_1 \leftarrow \dot{r}_{11}/\dot{\sigma}_1$ ; $\sin \check{\phi}_1 \leftarrow \dot{\sigma}_2/\dot{\sigma}_1$
7.   $q_{n1} \leftarrow t_{n1}/r_{11}$
8.   $\tilde{\sigma}_2 \leftarrow \sqrt{1-q_{n1}^2}$            {$\tilde{\sigma}_i := \|\underline{q}_{i:n,1}\|$}
9.   $\tilde{r}_{11} \leftarrow r_{11} - q_{n1}\, t_{n1}$
10.  $\cos \check{\theta}_1 \leftarrow \tilde{r}_{11}/r_{11}^{(n-1)}$; $\sin \check{\theta}_1 \leftarrow t_{11}\tilde{\sigma}_2/r_{11}^{(n-1)}$

{Main loop - calculate $\underline{r}_{\cdot j}$ and $\underline{r}_{\cdot j}^{(n-1)}$ from $\underline{r}_{\cdot j-1}$ and $\underline{r}_{\cdot j-1}^{(n-1)}$}
11.  $j \leftarrow 2$

{Phase I - calculate $\underline{r}_{\cdot j}$ using eq. (3.11a)}
12.1 $\dot{\underline{r}}_{1:j,j} \leftarrow \begin{pmatrix} t_{1j} \\ \underline{r}_{1:j-1,j-1}^{(n-1)} \end{pmatrix}$
```

```
12.2 if $j \neq n$ do {calculate and apply $G(\hat{\phi}_j)$}
    12.2.1 $\dot{r}_{j1} \leftarrow (z_{j-1} - \sum_{i=1}^{j-2} r^{(n-1)}_{i,j-1} \dot{r}_{i+1,1})/r^{(n-1)}_{j-1,j-1}$,
           where $\underline{z} := T^{T}_{n-1} \underline{t}_{2:n,1}$
    12.2.2 $\dot{\sigma}_{j+1} \leftarrow \sqrt{\dot{\sigma}_j^2 - \dot{r}_{j1}^2}$        $\{\dot{\sigma}_{j+1} := \|\dot{\underline{r}}_{j+1:n,1}\|\}$
    12.2.3 cos $\hat{\phi}_j \leftarrow \dot{r}_{j1}/\dot{\sigma}_j$ ; sin $\hat{\phi}_j = \dot{\sigma}_{j+1}/\dot{\sigma}_j$
    12.2.4 $\dot{\underline{r}}_{\cdot j} \leftarrow G(\hat{\phi}_j)\dot{\underline{r}}_{\cdot j}$
12.3 if $j \neq 2$ do {rest of GGMS upsweep}
    12.3.1 for $i \leftarrow j-1$ downto $2$ do $\dot{\underline{r}}_{\cdot j} \leftarrow G(\hat{\phi}_i)\dot{\underline{r}}_{\cdot j}$
{GGMS downsweep, excluding $G(\check{\phi}_j)$}
12.4 for $i \leftarrow 1$ to $j-1$ do $\dot{\underline{r}}_{\cdot j} \leftarrow G(\check{\phi}_i)\dot{\underline{r}}_{\cdot j}$
{if $j \neq n$, calculate and apply $G(\check{\phi}_j)$}
12.5A if $j=n$ then $\underline{r}_{\cdot j} \leftarrow \dot{\underline{r}}_{\cdot j}$ ; stop
12.5B else 12.5B.1 $r_{jj} \leftarrow \sqrt{\dot{r}_{jj}^2 + \dot{r}_{j+1,j}^2}$ ; cos $\check{\phi}_j \leftarrow \frac{\dot{r}_{jj}}{r_{jj}}$ ; sin $\check{\phi}_j \leftarrow \frac{\dot{r}_{j+1,j}}{r_{jj}}$ ;
           12.5B.2 $\underline{r}_{\cdot j} \leftarrow \begin{pmatrix} \dot{\underline{r}}_{1:j-1,j} \\ r_{jj} \\ \underline{0} \end{pmatrix}$
```

```
{Phase II - calculation of $\underline{r}_{\cdot j}^{(n-1)}$ using eq. (3.11b)}
{Calculate & apply $G(\hat{\phi}_j)$}

12.6   $q_{nj} \leftarrow (t_{nj} - \sum_{i=1}^{j-1} q_{ni}\, r_{ij})/r_{jj}$

12.7   $\tilde{\sigma}_{j+1} \leftarrow \sqrt{\tilde{\sigma}_j^2 - q_{nj}^2}$

12.8   $\cos\hat{\theta}_j \leftarrow q_{nj}/\tilde{\sigma}_j$ ; $\sin\hat{\theta}_j \leftarrow \tilde{\sigma}_{j+1}/\tilde{\sigma}_j$

12.9   $\underline{\tilde{r}}'_j \leftarrow \begin{pmatrix} \underline{r}_{1:j,j} - t_{nj}\,\underline{q}^T_{n,1:j} \\ -t_{nj}\tilde{\sigma}_{j+1} \\ \underline{0} \end{pmatrix}$

12.10  $\underline{\tilde{r}}'_j \leftarrow G(\hat{\theta}_j)\underline{\tilde{r}}'_j$

{Rest of GGMS upsweep}
12.11  if $j \neq 2$ do
        12.11.1  for $i \leftarrow j-1$ downto 2 do $\underline{\tilde{r}}'_j \leftarrow G(\hat{\theta}_i)\underline{\tilde{r}}'_j$

{GGMS downsweep, excluding $G(\check{\theta}_j)$}
12.12  for $i \leftarrow 1$ to $j-1$ do $\underline{\tilde{r}}'_j \leftarrow G(\check{\theta}_i)\underline{\tilde{r}}'_j$

{Calculate and apply $G(\check{\theta}_j)$}
12.13  $r_{jj}^{(n-1)} \leftarrow \sqrt{\tilde{r}'^2_{jj} + \tilde{r}'^2_{j+1,j}}$ ; $\cos\check{\theta}_j \leftarrow \tilde{r}'_{jj}/r_{jj}^{(n-1)}$ ; $\sin\check{\theta}_j \leftarrow \tilde{r}'_{j+1,j}/r_{jj}^{(n-1)}$;

12.14  $\underline{r}_{\cdot j}^{(n-1)} \leftarrow \begin{pmatrix} \underline{\tilde{r}}'_{1:j-1,j} \\ r_{jj}^{(n-1)} \\ \underline{0} \end{pmatrix}$
```

```
12.15  $j \leftarrow j+1$

12.16  Go to 12.1
       {End of main loop}
```

The proof for each step of algorithm 3.1 is indicated in the table below.

Step	Proof
1	$r_{11}^{(n-1)} = \Vert\underline{r}_{\cdot 1}^{(n-1)}\Vert = \Vert Q^{(n-1)T}\underline{t}_{1:n-1,1}\Vert = \Vert\underline{t}_{1:n-1,1}\Vert$.
2	Similar to 1.
3	Defined in eq. (3.5)
4	Definition - See comment on step 4.
5	Use the definition $\dot{\sigma}_i := \Vert\underline{r}_{i:n,1}\Vert$
6	From Lemma 2.1.
7	Equate $(n,1)$ element on either side of $T=QR$
8	Use $\tilde{\sigma}_i := \Vert\underline{q}_{i:n,1}\Vert$ (see comment)
9	Defined in eq. (3.8).
10	If GGMS is applied to $\tilde{R}$, we get by step 3, algorithm 2.1: $\tilde{r}_{11}^{\check{0}} = \tilde{r}_{11}^{\hat{2}}$ (3.13a) $= \tilde{r}_{11}$ because $G(\hat{\phi}_{n-1}),\ldots,G(\hat{\phi}_2)$ do not modify row 1. Also $\tilde{r}_2^{\check{0}} = \tilde{r}_2^{\hat{2}}$ (3.13b) $= \Vert\underline{\tilde{r}}_{2:n,1}\Vert = \tilde{\sigma}_2 t_{11}$ using Lemma 2.2 and the definition for $\tilde{\sigma}_i$. Then step 10 follows if (3.13a) and (3.13b) are substituted in step 4.1, Algorithm 2.1.
11	j is the row counter.

Step	Proof
12.1	Defined in eq. (3.5)
12.2.1	By eq. (3.5), $\dot{\underline{r}}_{2:n,1} = Q^{(n-1)T}\underline{t}_{2:n,1}$; but $T_{n-1} = Q^{(n-1)T}R^{(n-1)}$, so $Q^{(n-1)T} = R^{(n-1)-T}T^T_{n-1}$, hence $\dot{\underline{r}}_{2:n,1} = R^{(n-1)^{-T}} T^T_{n-1}\, \underline{t}_{2:n,1}$ $= R^{(n-1)^{-T}}\underline{z}$, so $\dot{\underline{r}}_{2:n}$ may be obtained by back-substituting in $R^{(n-1)T}\, \dot{\underline{r}}_{2:n,1} = \underline{z}$.
12.2.2	Use the definition $\dot{\sigma}_{j+1} := \lVert\underline{r}_{j+1:n,1}\rVert$.
12.2.3	From Lemma 2.
12.2.4, 12.3, 12.5A, 12.5B.1	These steps constitute the premultiplication of $\dot{\underline{r}}_{\cdot j}$ by $\prod_{i=1}^{k} G(\check{\phi}_i) \prod_{i=j}^{2} G(\hat{\phi}_i)$, hence by eq. (3.11a) the output is $\underline{r}_{\cdot j}$.
12.5B.1	The output of step 12.4 is $\dot{\underline{r}}^{\dot{j}\check{-}1}_{\cdot j}$. Substituting this in step 4.1, algorithm 2.1 yields 12.5B.1.
12.6	Equate (n,j) element on either side of $T = QR$.
12.7	Definition $\tilde{\sigma}_{j+1} := \lVert\underline{q}_{n,j+1:n}\rVert$
12.8	Follows from Lemma 2.2 and the definition of $\dot{\underline{r}}_{\cdot 1}$ (eq.3.9)
12.9, 12.10, 12.11, 12.12, and 12.14	These steps constitute the premultiplication of the vector on the right of eq. (3.11b) by $$\prod_{i=1}^{j} G(\check{\theta}_i) \quad \prod_{i=j}^{2} G(\hat{\theta}_i)$$ hence by (3.11b) the output is $\begin{pmatrix} \underline{r}^{(n-1)}_{\cdot j} \\ 0 \end{pmatrix}$.

Step	Proof
12.13	By eq. (2.9), the RHS of step 12.9 is $\tilde{\underline{r}}^{\hat{j}+1}_{\cdot j}$, so the output of step 13.12 is $$\prod_{i=1}^{j-1} G(\check{\theta}_{\cdot i}) \prod_{i=j}^{2} G(\hat{\theta}_{\cdot i}) \, \tilde{\underline{r}}^{\hat{j}+1}_{\cdot j} =: \tilde{\underline{r}}^{\check{j}-1}_{\cdot j}$$ Substituting this in step 4.1, algorithm 2.1 yields 12.13. QED.

3.5 Solution of the System $T\underline{x}=\underline{b}$ using R only.

T can be decomposed in the form

$$T = LW \tag{3.14}$$

where L is lower-triangular and W is orthogonal. Then

$$T\underline{x} = \underline{b}$$

$$\Rightarrow \quad LW\underline{x} = \underline{b}$$

$$\text{so} \quad \underline{x} = W^T L^{-1} \underline{b} \tag{3.15}$$

It has been shown by Paige [71] that the system (3.15) can be solved stably* using L only by substituting $W = L^{-1}T$ (obtainable from 3.14) in (3.15), yielding

$$\underline{x} = T^T L^{-T} L^{-1} \underline{b} \tag{3.16}$$

It is easily shown that $L^T = R$, where R is the output when FT01 is performed on T^T.

*If the accuracy of L is comparable to that obtained using normal orthogonalization techniques, then $rel\underline{x}=O(\mu cond T)$, rather than $O(\mu cond^2 T)$.

4. CALCULATION OF BOTH Q AND R EXPLICITLY : FTO 2

In this section, we extend FTO 1 to calculate Q explicitly, as well as R. Some preliminary results are given in subsection 4.1. A column recursion for Q^T is given in subsection 4.2, and a row recursion for Q^T is given in subsection 4.3. In subsection 4.4, the row recursion for Q^T is combined with FTO 1 to give FTO 2, which calculates the columns of R and rows of Q^T (i.e. columns of Q) together.

4.1 Preliminary Results

The notation is as in section 2.2, and the results are analogous to eqs. (3.6), (3.8) and (3.10).

Rewriting (3.5), $\dot{Q}^T T = \dot{R}$

so $\dot{S}^T \dot{Q}^T T = \dot{S}^T \dot{R} = R$ using (3.6)

therefore $T = \dot{Q}\dot{S}R \Rightarrow Q = \dot{Q}\dot{S}$ or $Q^T = \dot{S}^T \dot{Q}^T$ (4.1)

We also have $Q^T(T-\underline{e}_n(\underline{v}^{RT},0)) = R - \underline{q}_{n.}^T(\underline{v}^{RT},0) =: \tilde{R}$

so $\tilde{S}^T Q^T(T - \underline{e}_n(\underline{v}^{RT},0)) = \tilde{S}^T \tilde{R} = \bar{R}$ using (3.8)

or $T - \underline{e}_n(\underline{v}^{RT},0) = (Q\tilde{S})\bar{R}$. (4.2)

But $\tilde{Q}^T(T - \underline{e}_n(\underline{v}^{RT},0)) = \bar{R}$

or $T - \underline{e}_n(\underline{v}^{RT},0) = \tilde{Q}\bar{R}$, (4.3)

and (4.2) and (4.3) $\Rightarrow \tilde{Q} = Q\tilde{S}$ or $\tilde{Q}^T = \tilde{S}^T Q^T$ (4.4)

For convenience denote

$Y := Q^T$, $Y^{(n-1)} = Q^{(n-1)T}$, $\dot{Y} = \dot{Q}^T$ and $\tilde{Y} = \tilde{Q}^T$ (4.5)

then (4.1) and (4.4) become

$$\boxed{\begin{aligned} Y &= \dot{S}^T \dot{Y} \\ \tilde{Y} &= \tilde{S}^T Y \end{aligned}} \qquad (4.6a,b)$$

Note that these results are analogous to (3.6) and (3.8). Writing $\dot{Y}$ and $\tilde{Y}$ in (4.6a) and (4.6b) explicitly, we have the transformations

$$\underbrace{\left[\begin{array}{c|c} 1 & \\ \hline & Y^{(n-1)} \end{array}\right]}_{\dot{Y}} \xrightarrow[\dot{S}^T]{\text{GGMS rotations}} \left[\, Y \,\right] \xrightarrow[\tilde{S}^T]{\text{GGMS rotations}} \underbrace{\left[\begin{array}{c|c} Y^{(n-1)} & \\ \hline & 1 \end{array}\right]}_{\tilde{Y}} \qquad (4.7)$$

This is another transformation of the type in (3.2) or (3.10), in which a matrix with $Y^{(n-1)}$ in the trailing submatrix is transformed to one with $Y^{(n-1)}$ in the leading submatrix in $O(n^2)$ operations. This property can be used as before to calculate $Y^{(n-1)}$ (and, as a by-product, Y) in $O(n^2)$ operations. We present a column recursion which can be used only when all the GGMS rotations have been previously calculated (say by FTO 1), and a row recursion in which only $\hat{\phi}_j$ $\check{\phi}_j$, $\hat{\theta}_j$ and $\check{\theta}_j$ are required to calculate $\underline{y}_{j\cdot}$ and $\underline{y}_{j\cdot}^{(n-1)}$. The row recursion is required if FTO 1 is to be extended to calculate Q explicitly, since not all the GGMS rotations are available during the execution of FTO 1.

4.2 The column recursion for Y

Writing out the jth column of (4.7) we have

$$\underbrace{\begin{bmatrix} 0 \\ \\ \underline{y}_{\cdot j-1}^{(n-1)} \end{bmatrix}}_{\dot{\underline{y}}_{\cdot j}} \xrightarrow[\dot{S}^T]{\text{GGMS rotations}} \left[\, \underline{y}_{\cdot j} \,\right] \xrightarrow[\tilde{S}^T]{\text{GGMS rotations}} \underbrace{\begin{bmatrix} \underline{y}_{\cdot j}^{(n-1)} \\ \\ 0 \end{bmatrix}}_{\tilde{\underline{y}}_{\cdot j}} \qquad (4.8)$$

which is the desired recursion for the columns of $Y^{(n-1)}$ and Y. As noted before, all the GGMS rotations are required, so FTO 1 must be run before the recursion (4.8) can be executed. The algorithm is

Algorithm 4.1 - Column recursion for Y

```
1.  $\underline{y}_{\cdot 1} \leftarrow \underline{e}_1^T := [1, 0, \ldots, 0]^T$
2.  $j \leftarrow 2$
{Main loop - one pass for each column of $Y$ and $\tilde{Y}$ calculated}
3.  $\underline{y}_{\cdot j} \leftarrow \dot{S}^T \underline{y}_{\cdot j}$
4.  Stop if $j=n$.
5.  $\tilde{\underline{y}}_{\cdot j} \leftarrow \tilde{S}^T \underline{y}_{\cdot j}$
6.  $\underline{y}_{\cdot j}^{(n-1)} \leftarrow \tilde{y}_{1:n-1,j}$
7.  $j \leftarrow j+1$
8.  $\underline{y}_{\cdot j} \leftarrow \begin{bmatrix} 0 \\ \underline{y}_{\cdot j-1}^{(n-1)} \end{bmatrix}$
9.  Go to 3.
```

Proof Step 1 is clear from the L.H. matrix of (4.7), and the main loop is simply an implementation of the recursion (4.8).

QED.

4.3 The row recursion for Y

The calculations are essentially the same as in the column recursion, but they are re-arranged to calculate Y and $Y^{(n-1)}$ row-by-row, and only $\hat{\phi}_j$, $\check{\phi}_j$, $\hat{\theta}_j$ and $\check{\theta}_j$ are required to calculate the jth row of these matrices. Before presenting the row-recursion, we need some preliminary definitions and results.

Preliminaries

Consider the plane rotations that convert $\dot{Y}$ to Y in (4.7).

Define
$$\dot{Y}^{\hat{j}} := \prod_{k=n-1}^{j} G(\hat{\phi}_j)\dot{Y} \tag{4.9}$$

$$Y^{\check{j}} := \prod_{k=1}^{j} G(\check{\phi}_j) \prod_{k=n-1}^{2} G(\hat{\phi}_j)\ \dot{Y} \tag{4.10}$$

$\dot{Y}^{\hat{j}}$ and $\dot{Y}^{\check{j}}$ can be considered to be Y after the application of $G(\hat{\phi}_j)$

and $G(\check{\phi}_j)$ respectively. The definition for $Y^{\hat{j}}$ and $Y^{\check{j}}$ are analogous to definitions for $A^{\hat{i}}$ and $A^{\check{i}}$ in the GGMS algorithm.

Next, define $\dot{\underline{y}}^{(1)}_{j.} := \dot{\underline{y}}^{\hat{j}}_{j.}$, $\dot{\underline{y}}^{(2)}_{j.} := \dot{\underline{y}}^{\widehat{j-1}}_{j.}$ and $\dot{\underline{y}}^{(3)}_{j.} := \dot{\underline{y}}^{\widecheck{j-1}}_{j.}$ (4.11a-c)

The significance of the definitions (4.11) is as follows. Consider what happens to the jth row as the GGMS rotations are applied to $\dot{Y}$ on the L.H.S. of (4.7). Row j will only be changed by $G(\hat{\phi}_j)$, $G(\hat{\phi}_{j-1})$, $G(\check{\phi}_{j-1})$ and $G(\check{\phi}_j)$. So $\dot{\underline{y}}^{(1)}_{j.}$, $\dot{\underline{y}}^{(2)}_{j.}$ and $\underline{y}^{(3)}_{j.}$ are the results of the first, second and third changes to row j. The result of the fourth change is $\underline{y}_{j.}$.

If we now consider the plane rotations that convert Y to $\tilde{Y}$ in (4.7), we may define $Y^{\hat{j}}$, $Y^{\check{j}}$, $\underline{y}^{(1)}_{j.}$, $\underline{y}^{(2)}_{j}$ and $\underline{y}^{(3)}_{j.}$ in a manner analogous to definitions (4.9), (4.10), (4.11a), (4.11b) and (4.11c) respectively.

Define the <u>compressed rotation matrix for angle ψ</u> by

$$\Gamma(\psi) := \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix}$$

Then the following results may be easily shown:

<u>Lemma 4.1</u> (i)

$$\begin{bmatrix} \dot{\underline{y}}^{(1)}_{j.} \\ \dot{\underline{y}}^{(2)}_{j+1.} \end{bmatrix} = \Gamma(\hat{\phi}_j) \begin{bmatrix} \dot{\underline{y}}_{j.} \\ \dot{\underline{y}}^{(1)}_{j+1.} \end{bmatrix} \qquad (4.12)$$

(ii)

$$\begin{bmatrix} \underline{y}_{j.} \\ \dot{\underline{y}}^{(3)}_{j+1.} \end{bmatrix} = \Gamma(\check{\phi}_j) \begin{bmatrix} \dot{\underline{y}}^{(3)}_{j.} \\ \dot{\underline{y}}^{(2)}_{j+1.} \end{bmatrix} \qquad (4.13)$$

Lemma 4.2 (i)

$$\begin{pmatrix} \underline{y}^{(1)}_{j.} \\ \underline{y}^{(2)}_{j+1.} \end{pmatrix} = \Gamma(\hat{\theta}_j) \begin{pmatrix} \underline{y}_{j.} \\ \underline{y}^{(1)}_{j+1.} \end{pmatrix} \tag{4.14}$$

(ii)

$$\begin{pmatrix} \tilde{\underline{y}}_{j.} \\ \underline{y}^{(3)}_{j+1.} \end{pmatrix} = \Gamma(\check{\theta}_j) \begin{pmatrix} \underline{y}^{(3)}_{j.} \\ \underline{y}^{(2)}_{j+1.} \end{pmatrix} \tag{4.15}$$

The recursion

By re-arranging eqs. (4.12)-(4.15), we obtain a recursion for $\dot{\underline{y}}^{(1)}_{j+1.}$, $\dot{\underline{y}}^{(2)}_{j+1.}$, $\underline{y}_{j.}$, $\underline{y}^{(3)}_{j+1.}$, $\underline{y}^{(1)}_{j+1.}$, $\underline{y}^{(2)}_{j+1.}$, $\tilde{\underline{y}}_{j.}$ and $\underline{y}^{(3)}_{j+1}$.

Writing the first equation in (4.12) with $\dot{\underline{y}}^{(1)}_{j+1.}$ on the L.H.S.,

$$\dot{\underline{y}}^{(1)}_{j+1.} = (\dot{\underline{y}}^{(1)}_{j.} - \dot{\underline{y}}_{j.} \cos \hat{\phi}_j)/\sin \hat{\phi}_j \tag{4.16}$$

Writing the second equation in (4.12) and both equations in (4.13) explicitly:

$$\dot{\underline{y}}^{(2)}_{j+1} = -\dot{\underline{y}}_{j.} \sin \hat{\phi}_j + \dot{\underline{y}}^{(1)}_{j+1.} \cos \hat{\phi}_j \tag{4.17}$$

$$\underline{y}_{j.} = \dot{\underline{y}}^{(3)}_{j.} \cos \check{\phi}_j + \dot{\underline{y}}^{(2)}_{j+1} \sin \check{\phi}_j \tag{4.18}$$

$$\dot{\underline{y}}^{(3)}_{j+1} = -\dot{\underline{y}}^{(3)}_{j.} \sin \check{\phi}_j + \dot{\underline{y}}^{(2)}_{j+1.} \cos \check{\phi}_j \tag{4.19}$$

Writing the first equation in (4.14) with $\underline{y}^{(1)}_{j+1}$. on the L.H.S.,

$$\underline{y}^{(1)}_{j+1.} = (\underline{y}^{(1)}_{j.} - \underline{y}_{j.} \cos \hat{\theta}_j)/\sin \hat{\theta}_j \tag{4.20}$$

Writing the second equation in (4.14) and both equations in (4.15) explicitly:

$$\underline{y}^{(2)}_{j+1.} = \underline{y}_{j.} \sin \hat{\theta}_j + \underline{y}^{(1)}_{j+1.} \cos \hat{\theta}_j \tag{4.21}$$

$$\underline{y}_j = \underline{y}^{(3)}_{j.} \cos \check{\theta}_j + \underline{y}^{(2)}_{j+1.} \sin \check{\theta}_j \tag{4.22}$$

$$\underline{y}^{(3)}_{j+1.} = -\underline{y}^{(3)}_{j.} \sin \check{\theta}_j + \underline{y}^{(2)}_{j+1.} \cos \check{\theta}_j \tag{4.23}$$

Using the definition of $\tilde{Y}$ and $\dot{Y}$,

$$\dot{\underline{y}}_{j+1.} = (0, \tilde{\underline{y}}_{j,1:n-1}) \tag{4.24}$$

The set of equations (4.17) - (4.24) is fully recursive, thus we have our desired recursion for the rows of Y and $\tilde{Y}$ using $\hat{\phi}_j$, $\check{\phi}_j$, $\hat{\theta}_j$ and $\check{\theta}_j$ only. Some initialization is required, but we postpone this to the next subsection in which the row-recursion for Y is combined with FTO 1 to produce an algorithm which calculates recursively the rows of Y and the columns of R.

4.4 Calculation of R and Y in $O(n^2)$ operations - FTO 2

We simply present the algorithm and prove the steps involving the calculation of Y. We do not repeat the steps which are the same as in FTO 1, but refer to them by step number in algorithm 2.

Theorem 4.1 A Toeplitz matrix T may be factored in the form $T=QR$, where Q is orthogonal and R is upper-triangular, in $O(n^2)$ operations by the following algorithm. All notation is as in the foregoing.

Algorithm 4.2 - FTO 2 (Programmed on p.A.54)

```
{Initialization}
1 - 10.  Initialize $r_{11}^{(n-1)}$, $r_{11}$, $\dot{r}_{11}$, $\dot{\sigma}_1$, $r_{11}$, $\dot{\sigma}_2$, $\check{\phi}_1$, $q_{n1}(=y_{1n})$, $\tilde{\sigma}_2$,
         $\tilde{r}_{11}$ and $\check{\theta}_1$, as in steps 1-10 of FTO 1.
11.      $\underline{y}_{1.}^{(n-1)} \leftarrow \underline{t}_{1:n-1,1}^T / r_{11}^{(n-1)}$ ; $\dot{\underline{y}}_{2.} \leftarrow (0, \underline{y}_{1.}^{(n-1)})$ ; $\tilde{\underline{y}}_{1\cdot} \leftarrow (\underline{y}_{1.}^{(n-1)}, 0)$ ;
         $\underline{y}_{1.} \leftarrow \frac{\underline{t}_{\cdot 1}^T}{r_{11}}$ ; $\dot{\underline{y}}_{1.} = \underline{e}_1^T$
12.      $\dot{\underline{y}}_{2.}^{(1)} \leftarrow (\underline{y}_{1.} - \dot{\underline{y}}_{1.} \cos \check{\phi}_1)/\sin \check{\phi}_1$ ; $\dot{\underline{y}}_{2.}^{(3)} \leftarrow -\dot{\underline{y}}_{1.} \sin \check{\phi}_1 + \underline{y}_{2.}^{(1)} \cos \check{\phi}_1$
13.      $\underline{y}_{2.}^{(1)} \leftarrow (\tilde{\underline{y}}_{1.} - \underline{y}_{1.} \cos \check{\theta}_1)/\sin \check{\theta}_1$ ; $\underline{y}_{2.}^{(3)} \leftarrow -\underline{y}_{1.} \sin \check{\theta}_1 + \underline{y}_{2.}^{(1)} \cos \check{\theta}_1$
```

```
{Main loop - calculate $\underline{r}_{\cdot j}$, $\underline{r}_{\cdot j}^{(n-1)}$, $\underline{y}_{j}$ and $\underline{y}_{j\cdot}^{(n-1)}$}

14.           $j \leftarrow 2$
              {Phase I : calc. $\underline{r}_{\cdot j}$ using eq.(3.11a), and $\underline{y}_{j\cdot}$ using eqs. (4.16)
               -(4.19)}

15.1-15.5 Same as steps 12.1-12.5 of FTO 1, but replacing step 12.2.1
              of FTO 1 by
                              $\dot{r}_{j1} \leftarrow \underline{y}_{j-1\cdot}^{(n-1)} \, \underline{t}_{2:n,1}$

{Calculation of $\underline{y}_{\cdot j}$}

15.6          $\dot{\underline{y}}_{j\cdot} \leftarrow (0, \underline{y}_{j-1\cdot}^{(n-1)})$

15.7          $\dot{\underline{y}}_{j+1\cdot}^{(1)} \leftarrow (\dot{y}_{j\cdot}^{(1)} - \underline{y}_{j\cdot} \cos \hat{\phi}_j)/\sin \hat{\phi}_j$ ; $\dot{\underline{y}}_{j+1\cdot}^{(2)} \leftarrow -\dot{\underline{y}}_{j\cdot} \sin \hat{\phi}_j + \dot{\underline{y}}_{j+1\cdot}^{(1)} \cos \hat{\phi}_j$

15.8          $\begin{bmatrix} \underline{y}_{j\cdot} \\ \dot{\underline{y}}_{j+1\cdot}^{(3)} \end{bmatrix} \leftarrow \Gamma(\check{\phi}_j) \begin{bmatrix} \dot{\underline{y}}_{j\cdot}^{(3)} \\ \dot{\underline{y}}_{j+1\cdot}^{(2)} \end{bmatrix}$

15.9          If $j=n$, set $\underline{y}_{n\cdot} = \underline{y}_{n\cdot}^{(3)}$ and stop

{Phase II - calc. $\underline{r}_{\cdot j}^{(n-1)}$ using eq.(3.11b) and $\underline{y}_{j\cdot}^{(n-1)}$ using eqs.(4.20)-(4.23)}

{Calculation of $\underline{r}_{\cdot j}^{(n-1)}$}

15.10 -       Same as steps 12.6-12.14 of FTO 1, but replacing step 12.6 of
15.18         FTO 1 by
                              $q_{nj} \leftarrow y_{jn}$

{Calculation of $\underline{y}_{j\cdot}^{(n-1)}$}

15.19         $\underline{y}_{j+1\cdot}^{(1)} \leftarrow (\underline{y}_{j\cdot}^{(1)} - \underline{y}_{j\cdot} \cos \hat{\theta}_j)/\sin \hat{\theta}_j$ ; $\underline{y}_{j+1\cdot}^{(2)} \leftarrow -\underline{y}_{j\cdot} \sin \check{\theta}_j + \underline{y}_{j+1\cdot}^{(1)} \cos \check{\theta}_j$

15.20         $\begin{bmatrix} \tilde{y}_{j\cdot} \\ \underline{y}_{j+1\cdot}^{(3)} \end{bmatrix} \leftarrow \Gamma(\check{\theta}_j) \begin{bmatrix} \underline{y}_{j\cdot}^{(3)} \\ \underline{y}_{j+1\cdot}^{(2)} \end{bmatrix}$
```

```
15.21    $\underline{y}^{(n-1)}_{j.} \leftarrow \tilde{\underline{y}}_{j,1:n-1}$ ;

15.22    $j \leftarrow j+1$

15.23    Go to 15.1  {End of main loop}
```

Proof

Step	Proof
1-10	Proved in FTO 1
11(i)	Equate column 1 on either side of $T_{n-1} = Y^{(n-1)T}R^{(n-1)}$
11(ii),(iii),(v)	Definitions - eq. (4.5)
11(iv)	Equate column 1 on either side of $T = Y^T R$
12(i)	Consider the effect of the GGMS rotations on row 1 in the first operation of (4.7). Row 1 is only changed by $G(\check{\phi}_1)$, so $\underline{y}_{1.} = \dot{\underline{y}}_{1.} \cos\check{\phi}_1 + \dot{\underline{y}}^{(1)}_{2.} \sin\check{\phi}_1 \Rightarrow$ step 12(i)
12(ii)	Because there is no $G(\hat{\phi}_1)$ $\dot{\underline{y}}^{(3)}_{2.} = \dot{\underline{y}}^{(2)}_{2.} = -\dot{\underline{y}}_{1.} \sin\check{\theta}_1 +$ $\dot{\underline{y}}^{(1)}_{2.} \cos\check{\theta}_1$, by Lemma 4.1.
13	Proof is analogous to that for step 12.
14	Row and column counter.
15.1-15.5	Proved in FTO 1. The relation $r_{j1} = \underline{y}^{(n-1)}_{j-1.}\underline{t}_{2:n-1}$ follows from the definition (eq.(3.5)).
15.6	Definition (eq.(4.5))
15.7-15.20	This is the recursion expressed by eqs. (4.16)-(4.23).
15.21	Definition (eq.(4.5)).

4.5 Solution of $T\underline{x} = \underline{b}$ using Q and R

If $$T\underline{x} = \underline{b}$$

then $$QR\underline{x} = \underline{b}$$

$$R\underline{x} = Q^T\underline{b}$$

which is easily solved.*

5. CONCLUSION

Algorithms have been presented which calculate (i) R only (ii) both Q and R, where the orthogonal decomposition of a Toeplitz matrix T is $T=QR$. The methods are based on an algorithm by Gill, Golub, Murray and Saunders, which triangularizes a matrix of the form (upper-triangular+rank-1) using plane rotations, and the shift-invariance property of T is used to generate an algorithm requiring $O(n^2)$ operations. The actual operations counts are for an order-n matrix: (i) $9\frac{1}{2}n^2+0(n)$ operations to calculate R only (ii) $25n^2+0(n)$ operations to calculate R and Q. In the next Chapter, another algorithm is presented which requires only $7\frac{1}{2}n^2+0(n)$ and $19n^2$ operations in cases (i) and (ii) respectively, but is logically more complex. It will also be shown how to use modified or 'fast' plane rotations to approximately halve all of those operation counts.

* FT02 can be used to solve $T\underline{x}=\underline{b}$ in $O(n)$ space, by finding the QR decomposition of T^T (LQ decomposition of T).

CHAPTER 8

FAST TOEPLITZ ORTHOGONALIZATION II

1. INTRODUCTION

In the last chapter, we described algorithms to calculate, in $O(n^2)$ operations, the orthogonal decomposition

$$T = QR \tag{1.1}$$

of a Toeplitz matrix R, where Q was orthogonal and R was upper-triangular. Algorithm FTO1 calculated R in $9\frac{1}{2}n^2$ operations* and algorithm FTO2 calculated both Q and R in $25n^2$ operations. In this chapter, we describe several extensions to these results. In section 2, we describe an algorithm, FTO3, which is related to FTO1, but requires only $7\frac{1}{2}n^2$ operations to calculate R. In section 3, we describe an algorithm FTO4, which is related to FTO2, but requires only $19n^2$ operations to calculate Q and R. In section 4, we indicate how fast Givens transforms can be used to reduce the operation counts in FTO3 and FTO4 by almost half. Only one of the accelerated algorithms, FTO5 (corresponding to FTO3) is given in detail, but the other is easily generated by adding suitable initialization and housekeeping to the fast recursions.

The accelerated routines are logically more complex than the original routines because of the need to keep track of scaling factors.

In section 5, we describe some extensions to the work on orthogonalization of rectangular Toeplitz matrices, solution of the Toeplitz least-squares problem and orthogonalization of block Toeplitz matrices.

*As in previous chapters, we count only multiplications and divisions. The lower-order terms are omitted in the operation counts.

The algorithms described in the following only apply when T is nonsingular. The algorithms have been extended to cater for some singular cases, but this work is not yet complete and will not be presented here.

2. FAST TOEPLITZ ORTHOGONALIZATION VERSION 3 (FTO3)

In this section we describe an algorithm which calculates R in $7\frac{1}{2}n^2$ operations, compared to $9\frac{1}{2}n^2$ operations for FTO1. The new algorithm FTO3, is logically more complex than FTO1, but the derivation is analogous.

2.1 Outline of the Algorithm

Recall from section 3.1 of the last chapter, eq.(3.2), that for T Toeplitz we could perform the following transofrmations

$$\left(\begin{array}{c|c} x & x\;-\;-\;-\;x \\ \hline x & \\ \cdot & \\ \cdot & R^{(n-1)} \\ \cdot & \\ x & \end{array}\right) \xrightarrow[\mathit{rotations}]{\substack{(2n-3)\\ \mathit{GGMS}}} R \xrightarrow[C]{\mathit{add}} R + C \xrightarrow[\mathit{rotations}]{\substack{(2n-3)\\ \mathit{GGMS}}} \left(\begin{array}{c|c} & x \\ & \cdot \\ R^{(n-1)} & \cdot \\ & \cdot \\ & x \\ \hline & x \end{array}\right) \tag{2.1}$$

where $R^{(n-1)}$ is the upper-triangular factor in $T_{n-1} = Q^{(n-1)}R^{(n-1)}$ ($Q^{(n-1)}$ orthogonal), C is a rank-1 matrix and the GGMS rotations are the rotations used in the Gill-Golub-Murray-Saunders orthogonal-factor update algorithm [32]. The GGMS algorithm was described in detail in the last chapter.

There is a more efficient alternative to (2.1) which can be written

$$\left[\begin{array}{c|ccc} x & x & - - - & x \\ \hline x & & & \\ \cdot & & R^{(n-1)} & \\ \cdot & & & \\ \cdot & & & \\ x & & & \end{array}\right] \xrightarrow[\textit{rotations}]{\substack{(2n-3)\\ GGMS}} R \xrightarrow[\textit{last row}]{\textit{replace}} \begin{bmatrix} R_{1:n-1}, & \\ \underline{v}^{RT}, & t \end{bmatrix} \xrightarrow[\textit{transforms}]{\substack{(n-1)\\ \textit{Inverse}}} \left[\begin{array}{ccc|c} & & & x \\ & & & \cdot \\ & R^{(n-1)} & & \cdot \\ & & & \cdot \\ & & & x \\ \hline x & - - - & x & x \end{array}\right] \tag{2.2}$$

where inverse transforms are related to Givens rotations and have the same operation counts. They will be defined in the next sub-section. Eq. (2.2) requires $3n + O(1)$ transforms (GGMS or inverse) compared to $4n + O(1)$ transforms for 2.1, so the former procedure requires 25% fewer operations than the latter. Here, as for FT01, we can get a recursion for the columns of $R^{(n-1)}$ and (as a by-product) R by writing out the kth column of (2.2)

$$\begin{bmatrix} \underline{x} \\ \\ \underline{r}^{(n-1)}_{\cdot k-1} \end{bmatrix} \xrightarrow[\textit{rotations}]{GGMS} \underline{r}_{\cdot k} \longrightarrow \begin{bmatrix} \underline{r}_{1:n-1,k} \\ \\ \overline{v}_{n-k} \end{bmatrix} \xrightarrow[\textit{transforms}]{\textit{Inverse}} \begin{bmatrix} \underline{r}^{(n-1)}_{\cdot k} \\ \\ \overline{x} \end{bmatrix} \tag{2.3}$$

The recursion suggested by (2.3) is:

1. Initialize $\underline{r}_{\cdot 1}$ and $\underline{r}^{(n-1)}_{\cdot 1}$
2. $k \leftarrow 2$

{Main loop}

3. Convert $\begin{bmatrix} x \\ \underline{r}^{(n-1)}_{\cdot k-1} \end{bmatrix}$ to $\underline{r}_{\cdot k}$ by GGMS rotations
4. Stop if $k=n$
5. Replace $r_{nk}(=0)$ with v_{n-k}.
6. Convert $\underline{r}_{\cdot k}$ to $\begin{bmatrix} \underline{r}^{(n-1)}_{\cdot k} \\ \overline{x} \end{bmatrix}$ by inverse transforms

7. $k \leftarrow k+1$

8. Go to 3.

Remarks: Step 3 is performed exactly as in FT01. The details of step 5 will be given in the next subsection. The recursion (2.3) calculates R and $R^{(n-1)}$ with 25% fewer transforms than are needed by FT01.

2.2 Preliminary Results

In this subsection, we derive a method for converting

$$\begin{pmatrix} R_{1:n-1\cdot} \\ \underline{v}^{RT}, t \end{pmatrix}$$

to a matrix with $R^{(n-1)}$ as its leading submatrix, as required in the second operation in (2.2). We proceed by first describing a method to convert the matrix $\tilde{R} := \left(\begin{array}{c|c} R^{(n-1)} & Q^{(n-1)T}\underline{u}^{R} \\ \hline \underline{v}^{RT} & t \end{array}\right)$ to R by plane rotations.

We then re-arrange this method so that given R and $\underline{r}_{n\cdot}=(\underline{v}^{RT},t)$, we can calculate the rest of $\tilde{R}$.

Calculation of R from $\tilde{R}$.

Define, as in the last chapter,

$$\tilde{Q} := \left(\begin{array}{c|c} Q^{(n-1)} & \underline{0} \\ \hline \underline{0}^{T} & 1 \end{array}\right) \qquad (2.4)$$

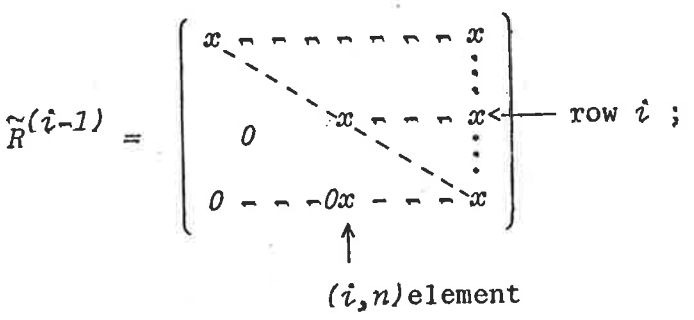
Because T is Toeplitz, it may be partitioned in the following two ways:

$$T =: \left(\begin{array}{c|c} T_{n-1} & \underset{\sim}{u}^R \\ \hline \underset{\sim}{v}^{RT} & t \end{array}\right) = \left(\begin{array}{c|c} t & \underset{\sim}{u}^T \\ \hline \underset{\sim}{v} & T_{n-1} \end{array}\right) \tag{2.5}$$

Then

$$\tilde{Q}^T T = \left(\begin{array}{c|c} R^{(n-1)} & Q^{(n-1)T}\underset{\sim}{u}^R \\ \hline \underset{\sim}{v}^{RT} & t \end{array}\right) =: \tilde{R} \tag{2.6}$$

$\tilde{R}$ has the form $\begin{pmatrix} x & \cdots & x \\ & \ddots & \vdots \\ x & \overset{0}{\cdots} & x \end{pmatrix}$, and may be converted to upper-triangular form by applying $n-1$ plane rotations which successively eliminate $n-1$ elements of the bottom row from left to right. To see this, suppose after the $(i-1)$th plane rotation, we have the matrix $\tilde{R}^{(i-1)}$ which has the form



then the (i,n) element can be eliminated by applying a suitable rotation in the (i,n) plane. The complete process is:

Algorithm 2.1

```
1.  $\tilde{R}^{(0)} \leftarrow \tilde{R}$
2.  For $i \leftarrow 1$ to $n-1$ do:
      2.1  $\rho_i \leftarrow \sqrt{\tilde{r}_{ii}^{(i-1)2} + \tilde{r}_{ni}^{(i-1)2}}$; $\cos\theta_i \leftarrow \tilde{r}_{ii}^{(i-1)}/\rho_i$; $\sin\theta_i \leftarrow \tilde{r}_{ni}^{(i-1)}/\rho_i$
                                                                                   (2.7a-c)
```

2.2 $$\begin{pmatrix} \tilde{\underline{r}}^{(i)}_{i\cdot} \\ \tilde{\underline{r}}^{(i)}_{n\cdot} \end{pmatrix} \leftarrow \begin{pmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{pmatrix} \begin{pmatrix} \tilde{\underline{r}}^{(i-1)}_{i\cdot} \\ \tilde{\underline{r}}^{(i-1)}_{n\cdot} \end{pmatrix} \tag{2.8}$$

3. $R \leftarrow \tilde{R}^{(n-1)}$

Denote $G(\theta_i)$ as the ith rotation matrix. Then, by algorithm 2.1,

$$R = S^T\tilde{R}, \quad \text{where } S^T := \prod_{1}^{n-1} G(\theta_i) \quad , \tag{2.9a-c}$$

where $\prod_{i=p}^{q} A_i$ denotes $A_q \; A_{q-1} \dots A_p$ for any set of matrices A_i. We used this notation in Chapter 7.

The following results, which can be easily shown from Algorithm 2.1, will be required later:

$$\underline{r}^{(p)}_{i\cdot} = \tilde{\underline{r}}_{i\cdot}, \; 1 \le i \le n-1, \; 0 \le p < i \tag{2.10a}$$

$$\underline{r}^{(p)}_{i\cdot} = \underline{r}_{i\cdot}, \; 1 \le i \le n-1, \; i \le p \le n-1 \tag{2.10b}$$

Calculation of $\tilde{R}_{1:n-1\cdot}$ from R and $\tilde{\underline{r}}_{n\cdot}$

We proceed by re-arranging the operations in Algorithm 2.1. We use an inductive argument. Suppose $\underline{r}_{i\cdot}$ and $\tilde{\underline{r}}^{(i-1)}_{n\cdot}$ are known. Then, we can calculate $\tilde{\underline{r}}_{i\cdot}$, $\tilde{\underline{r}}^{(i)}_{n\cdot}$ and θ_i as follows. It is clear that in (2.7a), $\rho_i = r_{ii}$, and by (2.10a) and (2.10b) $\tilde{r}^{(i-1)}_{ii} = \tilde{r}_{ii}$ and $\tilde{r}^{(i)}_{ii} = r_{ii}$, so (2.7) can be re-arranged to give

$$\tilde{r}_{ii} = \sqrt{r^2_{ii} - \tilde{r}^{(i-1)2}_{ni}}; \; \sec\theta_i = r_{ii}/\tilde{r}_{ii}; \; \tan\theta_i = r^{(i-1)}_{ni}/\tilde{r}_{ii} \tag{2.11a-c}$$

Also from (2.10a) and (2.10b), $\tilde{\underline{r}}^{(i-1)}_{i\cdot} = \tilde{\underline{r}}_{i\cdot}$ and $\tilde{\underline{r}}^{(i)}_{i\cdot} = \underline{r}_{i\cdot}$, so (2.8) can be rearranged to give

*The orthogonal decomposition of an arbitrary matrix using plane rotations and/or reflections is stable even for singular matrices. The present algorithm may be unstable because the functions *sec*θ, and *tan*θ are unbounded. However, it can be shown that *sec*θ_i, and *tan*θ_i are bounded by $(\|T\|/\|T_{n-1}\|)$ *cond*T_{n-1}. The algorithm can be run on an augmented matrix, so that *sec*θ_i, and *tan*θ_i are bounded by $(\|T_{n+1}\|/\|T_n\|)$ *condT*. In other words, *sec*θ_i and *tan*θ_i are large only if T is ill-conditioned.

$$\begin{bmatrix}\underline{\tilde{r}}_{i\cdot}\\ \\ \underline{\tilde{r}}^{(i)}_{n\cdot}\end{bmatrix} = \begin{bmatrix}\sec\theta_i & -\tan\theta_i\\ \\ -\tan\theta_i & \sec\theta_i\end{bmatrix}\begin{bmatrix}\underline{r}_{i\cdot}\\ \\ \underline{\tilde{r}}^{(i-1)}_{n\cdot}\end{bmatrix} \qquad (2.12)$$

Eqs. (2.11) and (2.12) are the desired relation for calculating θ_i, $\underline{\tilde{r}}_{i\cdot}$ and $\underline{\tilde{r}}^{(i)}_{n\cdot}$ from $\underline{r}_{i\cdot}$ and $\underline{\tilde{r}}^{(i-1)}_{n\cdot}$ and from the basis of a recursion for calculating $\tilde{R}_{1:n-1\cdot}$ from R and $\underline{\tilde{r}}_{n\cdot}$. We call the transformation an inverse transform, since it effectively reverses the effect of (2.8).* The complete recursion is:

* See facing page

Algorithm 2.2 - Conversion of R to $\tilde{R}$ by Inverse Transforms

1. $\underline{\tilde{r}}^{(0)}_{n\cdot} \leftarrow [\underline{v}^{RT}, t]$ (2.13)

2. For $i \leftarrow 1$ to $n-1$ do:

   2.1 $\tilde{r}_{ii} \leftarrow \sqrt{r^2_{ii} - \tilde{r}^{(i-1)2}_{ni}}$; $\sec\theta_i = r_{ii}/\tilde{r}_{ii}$; $\tan\theta_i = \tilde{r}^{(i-1)}_{ni}/\tilde{r}_{ii}$ (2.14a-c)

   2.2 $$\begin{bmatrix}\underline{\tilde{r}}_{i\cdot}\\ \\ \underline{\tilde{r}}^{(i)}_{n\cdot}\end{bmatrix} \leftarrow \begin{bmatrix}\sec\theta_i & -\tan\theta_i\\ \\ -\tan\theta_i & \sec\theta_i\end{bmatrix}\begin{bmatrix}\underline{r}_{i\cdot}\\ \\ \underline{\tilde{r}}^{(i-1)}_{n\cdot}\end{bmatrix} \qquad (2.15)$$

Denote $H(\theta_i)$ as the ith inverse transform matrix. Then, by algorithm 2.2, and noting that $\underline{\tilde{r}}^{(n-1)}_{n\cdot} = \underline{r}_{n\cdot}$

$$\begin{bmatrix}\tilde{R}_{1:n-1\cdot}\\ \\ \underline{r}_{n\cdot}\end{bmatrix} = \tilde{S}\begin{bmatrix}R_{1:n-1\cdot}\\ \hline \underline{v}^{RT},\ t\end{bmatrix} \quad \text{where } \tilde{S} = H(\theta_{n-1})H(\theta_{n-2})\ldots H(\theta_1) =: \prod_{i=1}^{n} H(\theta_i) \qquad (2.16)$$

Now, recall from the previous Chapter that the matrix on the left of (2.2) is

$$\dot{R} := \left[\begin{array}{c|c} t & \underline{u}^T\\ \hline \\ Q^{(n-1)T}\underline{v} & R^{(n-1)}\end{array}\right] \qquad (2.17)$$

and that $\dot{S}^{T}\dot{R} = R$,

$$\text{where } \dot{S}^T := \prod_{i=1}^{n-1} G(\check{\phi}_i) \prod_{i=n-1}^{2} G(\hat{\phi}_i) \text{ is the product of Givens}$$

rotations. (2.18a,b)

Rewriting (2.2) using (2.17), (2.18) and (2.16) we get

$$\underbrace{\left(\begin{array}{c|c} t & \underline{u}^T \\ \hline Q^{(n-1)T}\underline{v} & R^{(n-1)} \end{array}\right)}_{\dot{R}} \xrightarrow[\dot{S}^T]{\text{GGMS rotations}} \left(\begin{array}{c} R \end{array}\right) \longrightarrow \left(\begin{array}{c} R_{1:n-1} \\ \hdashline \underline{v}^{RT}, \ t \end{array}\right) \longrightarrow$$

$$\xrightarrow[\tilde{S}]{\text{inverse transf.}} \left(\begin{array}{c|c} R^{(n-1)} & Q^{(n-1)T}\underline{u}^R \\ \hline \underline{0}^T & r_{nn} \end{array}\right) \qquad (2.19)$$

The operations in (2.19) are basic to FT03.

2.3 FT03 - The Main Recursion

In section 2.1, we saw that writing out the kth column of (2.2) yielded a recursion for the columns of $R^{(n-1)}$. (2.19) is a detailed version of (2.2), so writing out the kth column of (2.19), we have for $1 < k < n$

$$\underbrace{\begin{pmatrix} u_{k-1} \\ \\ \underline{r}^{(n-1)}_{\cdot k-1} \end{pmatrix}}_{\dot{\underline{r}}_{\cdot k}} \xrightarrow[\dot{S}^T]{\text{GGMS rotations}} \begin{pmatrix} \underline{r}_{\cdot k} \end{pmatrix} \longrightarrow \begin{pmatrix} \underline{r}_{1:n-1,k} \\ \\ v_{n-k} \end{pmatrix} \xrightarrow[\text{inverse transf. } =: \tilde{S}_k]{\text{First } k} \begin{pmatrix} \underline{r}^{(n-1)}_{\cdot k} \\ \\ \overline{0} \end{pmatrix} \qquad (2.20)$$

Notice in (2.20) we use only the <u>first k</u> inverse transforms

$$S_k := \prod_{i=1}^{k} H(\theta_i) \qquad (2.21)$$

since it is clear that the last $n - 1 - k$ transforms have no further effect on column k. Following Chapter 7, (eq. 3.12), we can rewrite the first transformation in (2.20), yielding

$$\begin{bmatrix} u_{k-1} \\ \\ \underline{r}^{(n-1)}_{\cdot k-1} \end{bmatrix} \xrightarrow[S^T(\hat{\phi}_k,\check{\phi}_k)]{\substack{plane \\ rotations}} \begin{bmatrix} \\ \underline{r}_{\cdot k} \\ \\ \end{bmatrix} \longrightarrow \begin{bmatrix} \underline{r}_{1:n-1,k} \\ \\ v_{n-k} \end{bmatrix} \longrightarrow$$

$$\xrightarrow[\substack{inverse \\ transf. \\ \tilde{S}_k}]{first\ k} \begin{bmatrix} \underline{r}^{(n-1)}_{\cdot k} \\ \\ 0 \end{bmatrix}, \quad 1 < k < n \qquad (2.22)$$

where $S^T(\hat{\phi}_k,\check{\phi}_k)$ is defined as in eq. (7.3.11b).

The transformations (2.22) constitute a recursion for the columns of $R^{(n-1)}$ and R, and only $\{\check{\phi}_i, \hat{\phi}_i, \theta_i\}_{i\leq k}$ are required at step k, and these can easily be calculated from known quantities. This is most conveniently done by presenting FTO3, and proving the relevant formulae.

2.4 FTO3 - The Algorithm

The algorithm (in outline) is:

1. Initialize $\underline{r}_{\cdot 1}$, $\underline{r}^{(n-1)}_{\cdot 1}$, $\check{\phi}_1$. (Note - there is no $\hat{\phi}_1$).

2. Initialize θ_1.

3. For $k \leftarrow 2$ to n do

   3.1 Calculate $\hat{\phi}_k$ and $\check{\phi}_k$ from known quantities.

   3.2 Calculate $\underline{r}_{\cdot k}$ using the first part of (2.22).

   3.3 Calculate $\underline{r}^{(n-1)}_{\cdot k}$ using the rest of (2.22)

   3.4 Calculate θ_k from known quantities.

The algorithm in detail is presented and proved in Theorem 2.1. Note that the steps corresponding to steps 1, 3.1 and 3.2 of the Outline are exactly the same as in FTO1, and in fact are FTO1 steps 1-6 and 12.1-12.4.

Note also that we use j instead of k as the column pointer.

Theorem 2.1 Let T be an $n{\times}n$ Toeplitz matrix and let Q orthogonal and R upper-triangular such that $T=QR$. Then R may be calculated in $O(n^2)$ operations by the following algorithm:

Algorithm 2.3 - FTO3 (Programmed on p.A.72)

```
{Initialization - steps 1-6 are the same as in FTO1}
1.  $r_{11}^{(n-1)} \leftarrow \|\underline{t}_{1:n-1,1}\|_2$
2.  $r_{11} \leftarrow \sqrt{r_{11}^{(n-1)2} + t_{n1}^2}$
3.  $\dot{r}_{11} \leftarrow t_{11}$
4.  $\dot{\sigma}_1 \leftarrow r_{11}$
5.  $\dot{\sigma}_2 \leftarrow \sqrt{r_{11}^2 - \dot{r}_{11}^2}$
6.  $\cos \check{\phi}_1 \leftarrow \dot{r}_{11}/\dot{\sigma}_1$ ; $\sin \check{\phi}_1 \leftarrow \dot{\sigma}_2/\dot{\sigma}_1$
7.  $\sec\theta_1 \leftarrow r_{11}/r_{11}^{(n-1)}$; $\tan\theta_1 \leftarrow v_{n-1}/r_{11}^{(n-1)}$
{Main loop - calculate $\underline{r}_{\cdot j}$ and $\underline{r}_{\cdot j}^{(n-1)}$ from $\underline{r}_{\cdot j-1}$ and $\underline{r}_{\cdot j-1}^{(n-1)}$}
8.  $j \leftarrow 2$
{Phase I - calculate $\dot{\underline{r}}_{\cdot j}$ using first transformation of (2.22).  Steps
 are same as for FTO1, Phase I}
9.1 $\dot{\underline{r}}_{1:j,j} \leftarrow \begin{bmatrix} t_{1j} \\ \underline{r}^{(n-1)}_{1:j-1,j-1} \end{bmatrix}$
9.2 if $j \neq n$ do {calculate and apply $G(\hat{\phi}_j)$}
    9.2.1 $\dot{r}_{j1} \leftarrow (z_{j-1} - \sum_{i=1}^{j-2} r^{(n-1)}_{i,j-1}\, \dot{r}_{i+1,1})/r^{(n-1)}_{j-1,j-1}$,
                  where $\underline{z} := T^T_{n-1}\, \underline{t}_{2:n,1}$
```

```
        9.2.2  $\dot{\sigma}_{j+1} \leftarrow \sqrt{\dot{\sigma}_j^2 - \dot{r}_{j1}^2}$

        9.2.3  $\cos\hat{\phi}_j \leftarrow \dot{r}_{j1}/\dot{\sigma}_j$ ; $\sin\hat{\phi}_j \leftarrow \dot{\sigma}_{j+1}/\dot{\sigma}_j$

        9.2.4  $\dot{\underline{r}}_{\cdot j} \leftarrow G(\hat{\phi}_j)\dot{\underline{r}}_{\cdot j}$

    9.3  if $j \neq 2$ do {rest of GGMS upsweep}

        9.3.1  for $i \leftarrow j - 1$ downto $2$ do $\dot{\underline{r}}_{\cdot j} \leftarrow G(\hat{\phi}_i)\dot{\underline{r}}_{\cdot j}$

    {GGMS downsweep, excluding $G(\check{\phi}_j)$}

    9.4  for $i \leftarrow 1$ to $j-1$ do $\dot{\underline{r}}_{\cdot j} \leftarrow G(\check{\phi}_j)\dot{\underline{r}}_{\cdot j}$

    {if $j \neq n$, calculate and apply $G(\check{\phi}_j)$}

    9.5A  if $j=n$, then $\underline{r}_{\cdot j} \leftarrow \dot{\underline{r}}_{\cdot j}$ ; stop

    9.5B  else 9.5B.1  $r_{jj} \leftarrow \sqrt{\dot{r}_{jj}^2 + \dot{r}_{j+1,j}^2}$ ; $\cos\check{\phi}_j \leftarrow \frac{\dot{r}_{jj}}{r_{jj}}$ ; $\sin\check{\phi}_j \leftarrow \frac{\dot{r}_{j+1,j}}{r_{jj}}$ ;

               9.5B.2  $\underline{r}_{\cdot j} \leftarrow \begin{pmatrix} \dot{\underline{r}}_{1:j-1,j} \\ r_{jj} \\ \underline{0} \end{pmatrix}$

    {Phase II - calculation of $\underline{r}_{\cdot j}^{(n-1)}$ using rest of (2.22)}

    9.6  $\tilde{r}_{nj}^{(0)} \leftarrow v_{n-j}$

    {First $j$-$1$ inverse transf. $H(\theta_1),\ldots,H(\theta_{j-1})$}

    9.7  for $i \leftarrow 1$ to $j-1$ do $\begin{pmatrix} r_{ij}^{(n-1)} \\ \tilde{r}_{nj}^{(i)} \end{pmatrix} \leftarrow \begin{pmatrix} \sec\theta_i & -\tan\theta_i \\ -\tan\theta_i & \sec\theta_i \end{pmatrix} \begin{pmatrix} r_{ij} \\ \tilde{r}_{nj}^{(i-1)} \end{pmatrix}$

    {Calculate and apply $H(\theta_k)$}

    9.8  $r_{jj}^{(n-1)} \leftarrow \sqrt{r_{jj}^2 - \tilde{r}_{nj}^{(j-1)2}}$ ; $\sec\theta_j \leftarrow r_{jj}/r_{jj}^{(n-1)}$; $\tan\theta_j \leftarrow \tilde{r}_{nj}^{(j-1)}/r_{jj}^{(n-1)}$.

    {End of Main Loop}
```

Proof: Steps 1-6 and 9.1-9.5 are the same as in FTO1 and were proved there. Step 7 follows by putting (2.12) and the definition of $\tilde{R}$ into (2.13b,c). Step 9.6 follows from (2.12), steps 9.7, 9.8a carry out the application of

$$\{H(\theta_i)\}_1^j \quad \text{to} \quad \begin{pmatrix} \underline{r}_{1:n-1,j} \\ v_{n-j} \end{pmatrix}, \quad \text{so are true by the last part}$$

of (2.21). Steps 9.8b,c follow from (2.13b,c) and the defintion of $\tilde{R}$. QED.

3. CALCULATION OF BOTH Q AND R EXPLICITLY: FTO4

In this section, we extend FTO3 to calculate Q explicitly, as well as R. Some preliminary results are given in subsection 3.1. A column recursion for Q^T is given in subsection 3.2, and a row recursion for Q^T is given in subsection 3.4 In subsection 3.5, the row recursion for Q^T is combined with FTO3 to give FTO4, which calculates the columns of R and rows of Q^T (i.e. columns of Q) together. It will be seen that the recursions have two phases, one involving GGMS rotations and the other involving inverse rotations, introduced in the last section. The phase involving GGMS rotations is exactly the same as for FTO2.

3.1 Preliminary Results

As in Chapter 6, define

$$Y := Q^T, \; Y^{(n-1)} := Q^{(n-1)T}, \quad \dot{Y} := \left[\begin{array}{c|c} 1 & \underline{0}^T \\ \hline \underline{0} & Q^{(n-1)T} \end{array}\right]$$

and $$\tilde{Y} := \left(\begin{array}{c|c} Q^{(n-1)T} & \underline{0} \\ \hline \underline{0}^T & 1 \end{array}\right) = \tilde{Q}^T. \qquad (3.1)$$

We showed in Chapter 7 that

$$Y = \dot{S}^T \dot{Y} \qquad (3.2)$$

where $\dot{S}^T$ is the product of GGMS rotations used in FT01, and defined in (7.2.18).

Now we rewrite (1.1) $\quad T = QR \qquad (3.3)$

and from (2.6) $\quad T = \tilde{Q}\tilde{R}$, which with (2.9) yields

$$T = \tilde{Q}SR \qquad (3.4)$$

If T is nonsingular and $r_{ii} > 0$ (as was the case in FT01-FT03), the decomposition $T = QR$ is unique, so

eqs.(3.3) and (3.4) $\Rightarrow Q = \tilde{Q}S \Rightarrow Y = S^T\tilde{Y} \qquad (3.5)$

When T has rank $p < n$, then $R_{p+1:n.} = 0$ and $R_{1:p.}$ and $Q_{.1:p}$ are unique if R has the following form, where $j(i)$ is the index of the first non-zero element of $\underline{r}_{i.}$:

$$r_{i,j(i)} > 0, \quad 1 \le i \le p$$

$$j(i) > j(i-1), \quad 2 \le i \le p$$

(This form is related to the row echelon form, in which case the first equation above is replaced by $r_{i,j(i)} = 1$).

The procedure described below has been modified to cater for some cases of singular T. We conjecture that it can be generalized to cater for any T.

Recall from section 2 that $R = S^T\tilde{R}$ (2.9)..

Eq. (2.9) could be rearranged to yield

$$\begin{pmatrix} \tilde{R}_{1:n-1\cdot} \\ \\ \underline{t}_{n\cdot} \end{pmatrix} = \tilde{S} \begin{pmatrix} R_{1:n-1} \\ \\ \underline{\tilde{r}}_{n\cdot} \end{pmatrix}$$

where $\tilde{S}$, the product of inverse transforms, is as in (2.16). Similarly, (3.5) may be re-arranged to yield

$$\begin{pmatrix} \tilde{Y}_{1:n-1\cdot} \\ \\ \underline{y}_{n\cdot} \end{pmatrix} = \tilde{S} \begin{pmatrix} Y_{1:n-1\cdot} \\ \\ \underline{\tilde{y}}_{n\cdot} \end{pmatrix}$$

$$\text{i.e.} \quad \begin{pmatrix} \tilde{Y}_{1:n-1\cdot} \\ \\ \underline{y}_{n\cdot} \end{pmatrix} = \tilde{S} \begin{pmatrix} Y_{1:n-1\cdot} \\ \\ \underline{e}_{n}^{T} \end{pmatrix} \tag{3.6}$$

Eqs. (3.2) and (3.6) may be written together as

$$\left(\begin{array}{c|c} 1 & \underline{0}^T \\ \hline \underline{0} & Y^{(n-1)} \end{array}\right) \xrightarrow[\dot{S}^T]{\substack{GGMS \\ rotations}} \left(Y \right) \longrightarrow \begin{pmatrix} Y_{1:n-1\cdot} \\ \\ \underline{e}_n^T \end{pmatrix} \xrightarrow[\tilde{S}]{\substack{inverse \\ transf.}} \left(\begin{array}{c|c} Y^{(n-1)} & \underline{0} \\ \hline \underline{y}_{n,1:n-1} & y_{nn} \end{array}\right) \tag{3.7}$$

This is another transformation of the type (2.19), in which a matrix with $Y^{(n-1)}$ as the trailing submatrix is transformed to one with $Y^{(n-1)}$ as the leading submatrix in $O(n^2)$ operations. This property can be used to calculate $Y^{(n-1)}$ (and, as a by-product, Y) in $O(n^2)$ operations. We present a column recursion which can be used only when all of the GGMS rotations and inverse rotations have previously been calculated and saved (say, by FT03), and a row-recursion in which only $\hat{\phi}_j$, $\check{\phi}_j$ and θ_j are required to calculate $\underline{y}_{j\cdot}$ and $\underline{y}_{j\cdot}^{(n-1)}$. The row recursion is required if FTO3 is to be extended to calculate Q explicitly, since not all of the GGMS rotations are available during the execution of FTO3.

3.2 The Column Recursion for Y

Writing out the jth column of (3.7), we have

$$\underbrace{\begin{bmatrix}0\\ \\ \underline{y}^{(n-1)}_{\cdot j-1}\end{bmatrix}}_{\dot{\underline{y}}_{\cdot j}} \xrightarrow[\dot{S}^T]{\substack{GGMS\\ rotations}} \begin{bmatrix}\\ \underline{y}_{\cdot j}\\ \\ \end{bmatrix} \longrightarrow \begin{bmatrix}\underline{y}_{1:n-1,j}\\ \\ 0\end{bmatrix} \xrightarrow[\tilde{S}]{\substack{inverse\\ transf.}} \begin{bmatrix}\underline{y}^{(n-1)}_{\cdot j}\\ \\ y_{nj}\end{bmatrix} \tag{3.8}$$

The column recursion for Y can be written down immediately from (3.8). Note that FTO3 must have been previously run to supply the GGMS and inverse transforms.

Algorithm 3.1 - Column Recursion for Y

```
1. $\dot{\underline{y}}_{\cdot 1} \leftarrow \underline{e}_1$

2. $j \leftarrow 1$

{Main loop - one pass for each column of $Y$ and $Y^{(n-1)}$ calculated}

3. $\underline{y}_{\cdot j} \leftarrow \dot{S}^T \dot{\underline{y}}_{\cdot j}$

4. Stop if $j=n$

5. $\begin{bmatrix}\underline{y}^{(n-1)}_{\cdot j}\\ y_{nj}\end{bmatrix} \leftarrow \tilde{S} \begin{bmatrix}\underline{y}_{1:n-1,j}\\ 0\end{bmatrix}$

6. $\dot{\underline{y}}_{\cdot j+1} \leftarrow \begin{bmatrix}0\\ \underline{y}^{(n-1)}_{\cdot j}\end{bmatrix}$

7. $j \leftarrow j + 1$

8. Go to 3.
```

3.3 The Row Recursion for Y

Eq. (3.7) shows that there are two main phases in the Y recursions in the GGMS rotations and the inverse transforms. Phase I of the row recursion is the same as phase I of the row recursion of Chapter 7 - eqs. (7.4.16)-(7.4.19). We rewrite these equations here, and note that $\dot{\underline{y}}_{2\cdot}^{(1)}$ and $\dot{\underline{y}}_{2\cdot}^{(3)}$ must be initialized as in Chapter 7.

$$\dot{\underline{y}}_{j+1\cdot}^{(1)} = (\dot{\underline{y}}_{j\cdot}^{(1)} - \dot{\underline{y}}_{j\cdot}\ cos\hat{\phi}_j)/sin\hat{\phi}_j \tag{3.9}$$

$$\dot{\underline{y}}_{j+1\cdot}^{(2)} = -\dot{\underline{y}}_{j\cdot}\ sin\hat{\phi}_j + \dot{\underline{y}}_{j+1\cdot}^{(1)}\ cos\hat{\phi}_j \tag{3.10}$$

$$\underline{y}_{j\cdot} = \dot{\underline{y}}_{j\cdot}^{(3)}\ cos\check{\phi}_j + \dot{\underline{y}}_{j+1\cdot}^{(2)}\ sin\check{\phi}_j \tag{3.11}$$

$$\dot{\underline{y}}_{j+1\cdot}^{(3)} = -\dot{\underline{y}}_{j\cdot}^{(3)}\ sin\check{\phi}_j + \dot{\underline{y}}_{j+1\cdot}^{(2)}\ cos\check{\phi}_j \tag{3.12}$$

Phase II of the row recursion may be written down directly from the inverse transforms without any re-arrangement:

$$\tilde{\underline{y}}_{n,1:n-1}^{(0)} = \underline{0}^T \tag{3.13}$$

$$\begin{bmatrix} \underline{y}_{j\cdot}^{(n-1)} \\ \tilde{\underline{y}}_{n,1:n-1}^{(j)} \end{bmatrix} = \begin{bmatrix} sec\theta_j & -tan\theta_j \\ -tan\theta_j & sec\theta_j \end{bmatrix} \begin{bmatrix} \underline{y}_{j,1:n-1} \\ \tilde{\underline{y}}_{n,1:n-1}^{(j-1)} \end{bmatrix} \tag{3.14}$$

and from the definition $\dot{\underline{y}}_{j+1\cdot} = (0,\ \underline{y}_{j\cdot}^{(n-1)})$ (3.15)

Eqs. (3.9) - (3.12), (3.14) and (3.15) constitute the desired row recursion for Y. using $\hat{\phi}_j$, $\check{\phi}_j$ and θ_j only. Some initialization is required, but this is postponed to the next subsection in which the row-recursion for Y is combined with FT03 to give an algorithm which calculates recursively the rows of Y and columns of R with 75% of the transforms required in FT02.

3.4 Calculation of R and Y in $19n^2$ operations - FT04

FT04 is presented in Theorem 3.1. All the steps involving R have been proved in FT03, all the steps involving Phase I of the calculation of Y have been proved in FT02, and the steps involving Phase II of the calculation of Y follow directly from (3.13) and (3.14), hence no proof is necessary. For brevity, the steps involving R which are the same as in FT03 have not been repeated, but referenced by their FT03 step numbers.

Theorem 3.1 A nonsingular* Toeplitz matrix T may be factored in the form $T=QR$, where Q is orthogonal and R is upper-triangular, in $19n^2$ operations by the following algorithm:

Algorithm 3.2 - FT04 (Programmed on p.A.78)

```
{Initialization}
1 - 7.  Initialize $r_{11}^{(n-1)}$, $r_{11}$, $\dot{r}_{11}$, $\dot{\sigma}_1$, $\dot{\sigma}_2$, $\check{\phi}_1$ and $\theta_1$ as in steps 1-7
        of FT03.
8.    $\underline{y}_{1.}^{(n-1)} \leftarrow \underline{t}_{1:n-1,1}^{T}/r_{11}^{(n-1)}$; $\dot{\underline{y}}_{2.} \leftarrow (0, \underline{y}_{1.}^{(n-1)})$; $\underline{y}_{1.} \leftarrow \underline{t}_{.1}^{T}/r_{11}$; $\dot{\underline{y}}_{1.} = \underline{e}^{T}$
      {same as FT02, step 11}
9.    $\dot{\underline{y}}_{2.}^{(1)} \leftarrow (\underline{y}_{1.} - \dot{\underline{y}}_{1.}\cos\check{\phi}_1)/\sin\check{\phi}_1$; $\dot{\underline{y}}_{2.}^{(3)} \leftarrow -\dot{\underline{y}}_{1.}\sin\check{\phi}_1 + \underline{y}_{2.}^{(1)}\cos\check{\phi}_1$ {same as FT02,
      step 12}
10.   $\tilde{\underline{y}}_{n,1:n-1}^{(0)} = \underline{0}^{T}$;  $\tilde{\underline{y}}_{n,1:n-1}^{(1)} = -\underline{y}_{1,1:n-1}\tan\theta_1$  {eqs. (3.13, 3.14)}
{Main loop - calculates $\underline{r}_{.j}$, $\underline{r}_{.j}^{(n-1)}$, $\underline{y}_{j.}$ and $\underline{y}_{j.}^{(n-1)}$}
11.   $j \leftarrow 2$
```

*FT04 has been modified to cater for some singular cases. We conjecture that it can be modified to handle any T.

```
{Phase I - Calculate $\underline{r}_{.j}$ using eq.(2.22a), and $\underline{y}_{j.}$ using eqs.(3.9)-(3.12)}
{Calculation of $\underline{r}_{.j}$}
12.1-12.5   Same as steps 9.1 to 9.5 of FT03, but replacing steps 9.2.1
            of FT03 by $\dot{r}_{j1} \leftarrow \underline{y}_{j-1.}^{(n-1)} \underline{t}_{2:n,1}$ and adding "$\underline{y}_{n.} \leftarrow \dot{\underline{y}}_{n.}^{(3)}$" before the
            "stop" in 9.5A.
{Calculation of $\underline{y}_{.j}$}
12.6    $\dot{\underline{y}}_{j.} \leftarrow (0,\ \underline{y}_{j-1.}^{(n-1)})$
12.7    $\dot{\underline{y}}_{j+1.}^{(1)} \leftarrow (\underline{y}_{j.}^{(1)} - \dot{\underline{y}}_{j.}\cos\hat{\phi}_j)/\sin\hat{\phi}_j$; $\dot{\underline{y}}_{j+1.}^{(2)} \leftarrow -\dot{\underline{y}}_{j.}\sin\hat{\phi}_j + \dot{\underline{y}}_{j+1.}^{(1)}\cos\hat{\phi}_j$
12.8    $\underline{y}_{j.} \leftarrow \dot{\underline{y}}_{j.}^{(3)}\cos\check{\phi}_j + \dot{\underline{y}}_{j+1.}^{(2)}\sin\check{\phi}_j$; $\dot{\underline{y}}_{j+1.}^{(3)} \leftarrow -\dot{\underline{y}}_{j.}^{(3)}\sin\check{\phi}_j + \dot{\underline{y}}_{j+1.}^{(2)}\cos\check{\phi}_j$
{Phase II - Calculate $\underline{r}_{.j}^{(n-1)}$ using (3.22c) and $\underline{y}_{j.}^{(n-1)}$ using (3.14)}
{Calculation of $\underline{r}_{.j}^{(n-1)}$}
12.9-12.11  Same as steps 9.6-9.8 of FT03.
{Calculation of $\underline{y}_{j.}^{(n-1)}$}
12.12   $\underline{y}_{j.}^{(n-1)} \leftarrow \underline{y}_{j,1:n-1}\sec\theta_j - \tilde{\underline{y}}_{n,1:n-1}^{(j-1)}\tan\theta_j$;
        $\tilde{\underline{y}}_{n,1:n-1}^{(j)} = -\underline{y}_{j,1:n-1}\tan\theta_j + \tilde{\underline{y}}_{n,1:n-1}^{(j-1)}\sec\theta_j$   {eq.(3.14)}
12.13   $j \leftarrow j+1$
12.14   Go to 12.1  {End of Main Loop}
```

4. THE USE OF FAST GIVENS TRANSFORMS IN TOEPLITZ ORTHOGONALIZATION

In this section, we review fast Givens transforms (FGT's). We then incorporate FGT's into FTO3, and describe an algorithm, FTO5, which is almost twice as fast as FTO3. We indicate how to incorporate FGT's into FTO4 as well, but do not state the resulting algorithm in detail, as it is quite long, and can be easily generated by adding suitable initialization and housekeeping steps to the fast recursions.

4.1 Review of Fast Givens Transforms

FGT's were first introduced by Gentleman [30], but Hammarling [43] gives a clear exposition. A sequence of FGT's can be applied without square-roots until the last step, at which point, n square-roots are required to recover the correctly-scaled result. In addition, each FGT requires $2n$ operations, compared to $4n$ operations and a square root for a normal Givens transform. In incorporating FGT's into Toeplitz orthogonalization, we are unable to avoid the square-root computation at each FGT, but we can retain the $2n$ operation count per transform.

The basic idea of the FGT is as follows. Suppose the operand rows $\underline{x}_1^T$ and $\underline{x}_2^T$ are, represented in scaled form, i.e. as $d_1\underline{y}_2^T$ and $d_2\underline{y}_2^T$ respectively. Applying the rotation θ, we have [90]

$$\begin{bmatrix} \underline{x}_1'^T \\ \underline{x}_2'^T \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} d_1 & \\ & d_2 \end{bmatrix} \begin{bmatrix} \underline{y}_1^T \\ \underline{y}_2^T \end{bmatrix} \tag{4.1}$$

$$= \begin{pmatrix} d_1 \cos\theta & d_2 \sin\theta \\ -d_1 \sin\theta & d_2 \cos\theta \end{pmatrix} \begin{pmatrix} \underline{y}_1^T \\ \underline{y}_2^T \end{pmatrix}$$

$$= \begin{pmatrix} d_1 \cos\theta & \\ & d_2 \cos\theta \end{pmatrix} \begin{pmatrix} 1 & \tan\theta(d_2/d_1) \\ -\tan\theta\ (d_1/d_2) & 1 \end{pmatrix} \begin{pmatrix} \underline{y}_1^T \\ \underline{y}_2^T \end{pmatrix}$$

$$=: \begin{pmatrix} d_1' \underline{y}_1'^T \\ d_2' \underline{y}_2'^T \end{pmatrix} \quad \text{where} \quad d_1' = d_1 \cos\theta,\ d_2' = d_2 \cos\theta$$

$$\text{and} \quad \begin{pmatrix} \underline{y}_1'^T \\ \underline{y}_2'^T \end{pmatrix} := \begin{pmatrix} 1 & \tan\theta(d_2/d_1) \\ -\tan\theta(d_1/d_2) & 1 \end{pmatrix} \begin{pmatrix} \underline{y}_1^T \\ \underline{y}_2^T \end{pmatrix} =: \tilde{G} \begin{pmatrix} \underline{y}_1^T \\ \underline{y}_2^T \end{pmatrix}, \text{ say.} \qquad (4.2\text{a-e})$$

Eq. (4.2) states that to calculate the scaled rows after the rotation θ, we premultiply the scaled rows by

$$\begin{pmatrix} 1 & (\tan\theta)/\rho \\ -\rho \tan\theta & 1 \end{pmatrix}, \text{ where } \rho := d_1/d_2,$$

then multiply the scaling factors by $\cos\theta$. Clearly (4.2) requires only $2n + O(n)$ operations.

If $\tan\theta$, $\cos\theta$ and the scaling factors are not required explicitly, then $\tilde{g}_{12}$ and $\tilde{g}_{21}$ may be calculated without square-roots as follows. The analysis follows Wilkinson [90]. Suppose we wish to annihilate $x_{2t} = d_2 y_{2t}$. Then from (4.1), we have

$$-(\sin\theta)\ d_1 y_{1t} + (\cos\theta) d_2 y_{2t} = 0 \qquad (4.3)$$

By manipulating (4.3) and (4.2b,c) the following may be shown;

$$\left.\begin{aligned} \tilde{g}_{12} &:= tan\theta({}^{d}2/d_1) = d_2^2\, y_{2t}/(d_1^2 y_{1t}) \\ \tilde{g}_{21} &:= -tan\theta({}^{d}1/d_2) = -y_{2t}/y_{1t} \\ d_1'^2 &= d_1^2[(d_1^2 y_{1t}^2)/(d_1^2 y_{1t}^2 + d_2^2 y_{2t}^2)] \\ d_2'^2 &= d_2^2[(d_1^2 y_{1t}^2)/(d_1^2 y_{1t}^2 + d_2^2 y_{2t}^2)] \end{aligned}\right\} \qquad (4.4a\text{-}d)$$

Notice only the squares of the scaling factors are used in eqs. (4.4). Eqs. (4.4a-d) constitute the normal FGT. Thus the FGT requires no square-roots during the reduction process, but if correct scaling of the reduced matrix is required n square roots are needed at the end. In our algorithm, $tan\theta$, $cos\theta$ and the scaling factors are required explicitly, so we must use the normal Givens method for calculating $cos\theta$ and $sin\theta$, then apply (4.1a-e).

4.2 Incorporation of FGT's into FT03 - Algorithm FT05

The basic recursion of FT03 is eq. (2.22):

$$\underbrace{\begin{bmatrix} u_{k-1} \\ \\ \underline{r}^{(n-1)}_{.k-1} \end{bmatrix}}_{\dot{\underline{r}}_{.k}} \xrightarrow[S^T(\hat{\phi}_k,\check{\phi}_k)]{rotn's} \underline{r}_{.k} \longrightarrow \underbrace{\begin{bmatrix} \underline{r}_{1:n-1,k} \\ \\ v_{n-k} \end{bmatrix}}_{\underline{r}'_{.k}} \longrightarrow$$

$$\xrightarrow[\substack{inverse\ transf.\\ \tilde{S}_k}]{first\ k} \underbrace{\begin{bmatrix} \underline{r}^{(n-1)}_{.k} \\ \\ 0 \end{bmatrix}}_{\bar{\underline{r}}_{.k}} \qquad (4.5)$$

We will develop two procedures by which the plane rotations and inverse transforms in (4.5) can be done in two instead of the normal four operations that are normally required. These procedures will be the basis for FT05.

Consider the first transformation in (4.5). Suppose $\dot{\underline{r}}_{.k}$ is represented as

$$\dot{\underline{r}}_{.k} = \begin{bmatrix} 1 & & & \\ & \dot{d}_{2k} & & \\ & & \ddots & \\ & & & \dot{d}_{kk} \end{bmatrix} \begin{bmatrix} u_k \\ \\ \underline{w}^{(n-1)}_{.k} \end{bmatrix} =: \dot{D}_k \dot{\underline{w}}_{.k} \qquad (4.6)$$

and $\underline{r}_{.k}$ is represented as

$$\underline{r}_{.k} = \begin{bmatrix} d_{1k} & & \\ & \ddots & \\ & & d_{kk} \end{bmatrix} \qquad \underline{w}_{.k} = D_k \underline{w}_{.k} \qquad (4.7)$$

Then by (4.2), the plane rotations $S^T(\hat{\phi}_k, \check{\phi}_k)$ may be carried out by FGT's as in procedure 4.1. The superscript notation used in $W^{\hat{k}}$, $W^{\check{k}}$, etc. is the same as that used in Chapter 7 (Notational remark, algorithm 2.1 - $W^{\hat{k}}$ is the result of applying all the transforms up to rotation $\hat{\phi}_k$ to W.

Procedure 4.1

1. $\dot{w}^{\hat{k}}_{kk} \leftarrow \dot{w}_{kk}$ (4.8)

2. $\dot{d}^{\hat{k}}_{kk} \leftarrow \dot{d}_{kk} \cos \hat{\phi}_k$ (4.9)

3. $\dot{r}^{\hat{k}}_{k+1,k} \leftarrow -\dot{r}_{kk} \sin \hat{\phi}_k$ (4.10)

4. For $i \leftarrow k-1$ <u>downto</u> 2 do

4.1 $$\Phi_k(\hat{\phi}_i) \leftarrow \begin{bmatrix} 1 & (\tan\hat{\phi}_i)\,d_{i+1,k}^{\widehat{i+1}}/\dot{d}_{ik} \\ -(\tan\hat{\phi}_i)\,\dot{d}_{ik}/d_{i+1,k}^{\widehat{i+1}} & 1 \end{bmatrix} \tag{4.11}$$

4.2 $$\begin{bmatrix} \dot{w}_{ik}^{\hat{i}} \\ \dot{w}_{i+1,k}^{\hat{i}} \end{bmatrix} \leftarrow \Phi_k(\hat{\phi}_i) \begin{bmatrix} \dot{w}_{ik} \\ \dot{w}_{i+1,k}^{\widehat{i+1}} \end{bmatrix} \tag{4.12}$$

4.3 $$\dot{d}_{i+1,k}^{\hat{i}} \leftarrow d_{i+1,k}^{\widehat{i+1}} \cos\hat{\phi}_i \; ; \; \dot{d}_{ik}^{\hat{i}} \leftarrow \dot{d}_{ik} \cos\hat{\phi}_i \tag{4.13a,b}$$

5. $$\dot{w}_{1k}^{\check{0}} \leftarrow \dot{w}_{1k} \; ; \; \dot{d}_{1k}^{\check{0}} \leftarrow 1 \; ; \; \dot{w}_{2k}^{\check{0}} \leftarrow \dot{w}_{2k}^{\hat{2}} \; ; \; \dot{d}_{2k}^{\check{1}} \leftarrow \dot{d}_{2k}^{\hat{2}} \tag{4.14a-d}$$

6. For $i \leftarrow 1$ to $k-1$ do

6.1 $$\Phi_k(\hat{\phi}_i) \leftarrow \begin{bmatrix} 1 & (\tan\check{\phi}_i)\,\dot{d}_{i+1}^{\hat{i}}/\dot{d}_{i}^{\check{i}-1} \\ -(\tan\check{\phi}_i)\,\dot{d}_{i}^{\check{i}-1}/\dot{d}_{i+1}^{\hat{i}} & 1 \end{bmatrix} \tag{4.15}$$

6.2 $$\begin{bmatrix} w_{ik} \\ \dot{w}_{i+1,k}^{\check{i}} \end{bmatrix} \leftarrow \Phi_k(\hat{\phi}_i) \begin{bmatrix} \dot{w}_{ik}^{\check{i}-1} \\ \dot{w}_{i+1,k}^{\hat{i}} \end{bmatrix} \tag{4.16}$$

6.3 $$d_{ik} \leftarrow d_{ik}^{\check{i}-1} \cos\check{\phi}_i ; \; d_{i+1,k}^{\check{i}} \leftarrow d_{i+1,k}^{\hat{i}} \cos\check{\phi}_i \tag{4.17a,b}$$

7. $$\dot{r}_{kk}^{\check{k}-1} \leftarrow d_{kk}^{\check{k}-1}\, w_{kk}^{\check{k}-1} ; \; r_{kk} \leftarrow [(\dot{r}_{kk}^{\check{k}-1})^2 + (r_{k+1,k}^{\hat{k}})^2]^{\frac{1}{2}} ;$$

$$\cos\check{\phi}_k \leftarrow r_{kk}^{\check{k}-1}/r_{kk} ; \; d_{kk} \leftarrow d_{kk}^{\check{k}-1} \cos\phi_k ; \; w_{kk} \leftarrow r_{kk}/d_{kk} \tag{4.18a-e}$$

It will be shown later (Theorem 4.1) that the scaling factors $\dot{d}_{ik}$, etc. are the same in all columns, so the $\Phi_k(\hat{\phi}_i)$ and $\Phi_k(\check{\phi}_i)$, which depend only on the $\hat{\phi}_i$, $\check{\phi}_i$ and the scaling factors, are also the same in all columns. Hence we may write $\Phi_k(\hat{\phi}_i) = \Phi(\hat{\phi}_i)$, etc., and $\dot{d}^{\hat{i}}_{i+1,k} = \dot{d}^{\hat{i}}_{i+1}$, etc. The steps 4.1, 4.3, 6.1 and 6.3, in which the $\Phi_k(\hat{\phi}_i)$ and $\dot{d}^{\hat{i}}_{i+1,k}$ etc. are calculated, may therefore be omitted except when $i=k-1$, i.e. when $\Phi(\hat{\phi}_i)$, $\dot{d}^{\hat{i}}_{i+1}$ etc. are first calculated for that value of i.

Procedure 4.1 may be replaced by the following more efficient procedure.

Procedure 4.2

```
1.  $\dot{w}^{\hat{k}}_{kk} \leftarrow \dot{w}_{kk}$                                              (4.19)

2.  $\dot{d}^{\hat{k}}_{k} \leftarrow \dot{d}_k \cos\hat{\phi}_k$                                       (4.20)

3.  $\dot{r}^{\hat{k}}_{k+1,k} \leftarrow -\dot{r}_{kk} \sin\hat{\phi}_k$                                   (4.21)

4.  $\Phi(\hat{\phi}_{k-1}) \leftarrow \begin{pmatrix} 1 & (\tan\hat{\phi}_{k-1})\dot{d}^{\hat{k}}_{k}/\dot{d}_{k-1} \\ -(\tan\hat{\phi}_{k-1})\dot{d}_{k-1}/\dot{d}^{\hat{k}}_{k} & 1 \end{pmatrix}$;

    $\dot{d}^{\widehat{k-1}}_{k} \leftarrow \dot{d}^{\hat{k}}_{k} \cos\hat{\phi}_{k-1}$; $\dot{d}^{\widehat{k-1}}_{k-1} \leftarrow \dot{d}_{k-1} \cos\hat{\phi}_{k-1}$; $\begin{pmatrix} \dot{w}^{\widehat{k-1}}_{k-1,k} \\ \dot{w}^{\widehat{k-1}}_{kk} \end{pmatrix} \leftarrow \Phi(\hat{\phi}_{k-1}) \begin{pmatrix} \dot{w}_{k-1,k} \\ \dot{w}^{\hat{k}}_{kk} \end{pmatrix}$

                                                                        (4.23a-d)

5.  For $i \leftarrow k-2$ downto $2$ do $\begin{pmatrix} w^{\hat{i}}_{ik} \\ w^{\hat{i}}_{i+1,k} \end{pmatrix} \leftarrow \Phi(\hat{\phi}_i) \begin{pmatrix} \dot{w}_{ik} \\ w^{\widehat{i+1}}_{i+1,k} \end{pmatrix}$      (4.24)

6.  For $i \leftarrow 1$ to $k-2$ do $\begin{pmatrix} w_{ik} \\ \dot{w}^{\check{i}}_{i+1,k} \end{pmatrix} \leftarrow \Phi(\check{\phi}_i) \begin{pmatrix} w^{\widecheck{i-1}}_{ik} \\ \dot{w}^{\hat{i}}_{i+1,k} \end{pmatrix}$          (4.25)
```

7. $$\Phi(\check{\phi}_{k-1}) \leftarrow \begin{bmatrix} 1 & (\tan\check{\phi}_{k-1})\dot{d}_k^{\hat{k}-1}/\dot{d}_{k-1}^{\check{k}-2} \\ -(\tan\check{\phi}_{k-1})\dot{d}_{k-1}^{\check{k}-2}/\dot{d}_k^{\hat{k}-1} & 1 \end{bmatrix}; \qquad (4.26a\text{-}d)$$

$$\dot{d}_k^{\check{k}-1} \leftarrow \dot{d}_k^{\hat{k}-1}\ \cos\check{\phi}_{k-1}; \begin{bmatrix} w_{k-1,k} \\ w_{kk}^{\check{k}-1} \end{bmatrix} \leftarrow \Phi(\check{\phi}_{k-1}) \begin{bmatrix} \dot{w}_{k-1,k}^{\check{k}-2} \\ \dot{w}_{kk}^{\hat{k}-1} \end{bmatrix}$$

8. $$\dot{r}_{kk}^{\check{k}-1} \leftarrow \dot{d}_k^{\check{k}-1}\dot{w}_{kk}^{\check{k}-1};\ r_{kk} \leftarrow \sqrt{(\dot{r}_{kk}^{\check{k}-1})^2+(\dot{r}_{k+1,k}^{\hat{k}})^2}; \qquad (4.27a,b)$$

$$\cos\check{\phi}_k \leftarrow \dot{r}_{kk}^{\check{k}-1}/r_{kk};\ \tan\check{\phi}_k \leftarrow \dot{r}_{k+1,k}^{\hat{k}}/\dot{r}_{kk}^{\check{k}-1} \qquad (4.28a,b)$$

$$d_k \leftarrow d_k^{\check{k}-1}\ \cos\check{\phi}_k;\ w_{kk} \leftarrow r_{kk}/d_k \qquad (4.29a,b)$$

It is easy to see that procedure 4.2 requires only $4k+O(1)$ operations, compared to $8(k) + O(1)$ operations to compute $S^T(\hat{\phi}_k,\check{\phi}_k)\dot{\underline{r}}_{.k}$ using Givens transforms.

Consider the last transformation in (4.5). We first need a fast inverse transform (F.I.T.) procedure. This can be derived from the fast Givens procedure (eqs. 4.2b,c,d) by observing that in a F.I.T. d_1, $\underline{y}_1^T$, d_2' and $\underline{y}_2'^T$ are required and everything else is known. Rearranging (4.2b-d) yields

$$d_1 = d_1'/\cos\theta\ ;\quad d_2' = d_2\cos\theta\ ; \qquad (4.30a,b)$$

$$\left.\begin{aligned} \underline{y}_1^T &= \underline{y}_1^T - \underline{y}_2^T \tan\theta \; (d_2/d_1) \\ \underline{y}_2^T &= -\underline{y}_1^T \tan\theta (d_1/d_2) + \underline{y}_2^T \end{aligned}\right\} \qquad (4.31)$$

Eqs. (4.31) constitute a F.I.T. Applying (4.30) for the first $k-1$ inverse transforms in (4.5), calculating and applying the last inverse transform as in FT03, step 9.7 and assuming that the scaling factors are the same in all columns (proved in Theorem 4.1), we get Procedure 4.3, which transforms $\underline{r}'_{.k}$ to $\bar{\underline{r}}_{.k}$ in $2k+O(1)$ operations compared to $4k + O(1)$ operations using inverse transforms. Note that $\tilde{\underline{r}}_{n.}^{(k)} = d_n^{(k)}\tilde{\underline{w}}_{n.}^{(k)}$ in this procedure:

Procedure 4.3

```
1.  $\tilde{w}_{nk}^{(0)} = v_{n-k}$                                        (4.32)

{First $k-1$ fast inverse transforms}
2.  For $i \leftarrow 1$ to $k-1$ do
{$C_1, C_2$: FIT coefficients, defined in 4.38}
    2.1  $\bar{w}_{ik} = w_{ik} - C_1(\theta_i)w_{nk}^{(i-1)}$                   (4.33)
    2.2  $\tilde{w}_{nk}^{(i)} = -C_2(\theta_i)\bar{w}_{ik} + \tilde{w}_{nk}^{(i-1)}$           (4.34)

{Calculate and apply last inverse transform}
3.  $\tilde{r}_{nk}^{(k-1)} \leftarrow \tilde{d}_n^{(k-1)}\,\tilde{w}_{nk}$ ; $\bar{r}_{kk} \leftarrow \sqrt{r_{kk}^2 - \tilde{r}_{nk}^{(k-1)2}}$;     (4.35a,b)
    $\cos\theta_k \leftarrow \tilde{r}_{kk}/r_{kk}$; $\tan\theta_k \leftarrow \tilde{r}_{nk}^{(k-1)}/\bar{r}_{kk}$          (4.36a,b)
    $\tilde{d}_n^{(k)} \leftarrow \tilde{d}_n^{(k-1)}\cos\theta_k$; $\bar{d}_k \leftarrow d_k/\cos\theta_k$; $\bar{w}_{kk} \leftarrow \bar{r}_{kk}/\bar{d}_k$     (4.37a,b)
    $C_1(\theta_k) \leftarrow (\tilde{d}_n^{(k-1)}/\bar{d}_k)\tan\theta_k$; $C_2(\theta_k) \leftarrow (\bar{d}_k/\tilde{d}_n^{(k-1)})\tan\theta_k$   (4.38a,b)
```

We now prove that the scaling factors produced when recursion (4.5) is implemented with FGT's, are the same in all columns.

Theorem 4.1 Let $\dot{d}_{ik}$, $\dot{d}^{\widehat{i+1}}_{ik}$, $\dot{d}^{\hat{i}}_{ik}$, $\dot{d}^{\check{i}-1}_{ik}$ and d_{ik} be the scaling factors of row i when the first transformation in (4.2) is implemented by procedure 4.1, and let $\bar{d}^{(i)}_{ik}$ and $\tilde{d}^{(i)}_{nk}$ be the scaling factors of rows i and n when the last recursion in (4.2) is implemented as in FT03, but with fast inverse transforms. Then these scaling factors are the same in all columns, and are given by the recursion

$$\dot{d}_{1k} = \dot{d}_1 := 1 \; ; \; \tilde{d}^{(0)}_{nk} = \tilde{d}^{(0)}_n := 1 \qquad (4.39a,b)$$

$$d_{1k} = d_1 := cos\check{\phi}_1 \qquad (4.40)$$

$$\bar{d}_{1k} = \bar{d}_1 := d_1 \; cos\theta_1 \qquad (4.41)$$

$$\dot{d}_{ik} = \dot{d}_i := \bar{d}_{i-1} \qquad (4.42)$$

$$\dot{d}^{\hat{i}}_{ik} = \dot{d}^{\hat{i}}_i := \dot{d}_i \; cos \; \hat{\phi}_i \qquad (4.43)$$

$$\dot{d}^{\hat{i}-1}_{ik} = \dot{d}^{\hat{i}-1}_i := \dot{d}^{\hat{i}}_i \; cos\hat{\phi}_{i-1} \qquad (4.44)$$

$$d^{\check{i}-1}_{ik} = d^{\check{i}-1}_i := d^{\hat{i}-1}_i \; cos\check{\phi}_{i-1} \qquad (4.45)$$

$$d_{ik} = d_i := d_i^{\check{i}-1} \cos\check{\phi}_i \qquad (4.46)$$

$$\bar{d}_{ik} = \bar{d}_i := d_i/\cos\theta_i \qquad (4.47)$$

$$\tilde{d}_{nk}^{(i)} = \tilde{d}_n^{(i)} := \tilde{d}_n^{(i-1)} \cos\theta_i \qquad (4.48)$$

Proof: By (4.3), $d_{1k} = d_1 = 1$ which proves (4.39a).

The proof of (4.39b) is similar.

By (4.17a), with $i=1$, $d_{ik}=d_{1k}^{\check{0}}\cos\check{\phi}_1 = \cos\check{\phi}_1$ using (4.14b).

This proves (4.40). From (4.2),

$$\bar{r}_{1k} = r'^{\check{1}}_{1k}, \text{ so } \bar{d}_{1k} = d'^{\check{1}}_{1k} = d'_{1k} \cos\check{\theta}_1 \text{ using (4.1b)}$$

$$= d_{1k} \cos\check{\theta}_1 \text{ using (4.28)}$$

$$= d_1 \cos\check{\theta}_1, \text{ using (4.40). This proves (4.41).}$$

We now show (4.37)-(4,45) by induction on i, with the result for all columns k being proved in parallel. Assume now that $\bar{d}_{i-1,k} = \bar{d}_{i-1}$, $k=1,\dots,n$.

From (4.5) $\dot{r}_{ik} = \bar{r}_{i-1,k-1}$ so $\dot{d}_{ik} = \bar{d}_{i-1,k-1} = \bar{d}_{i-1}$ which proves (4.42).

From (4.13b), $\dot{d}_{ik}^{\hat{i}} = \dot{d}_{ik} \cos\hat{\phi}_i = \dot{d}_i \cos\hat{\phi}_i$ using (4.42).

This proves (4.43).

From (4.13a), $\dot{d}_{ik}^{\hat{i}-1} = \dot{d}_{ik} \cos\hat{\phi}_{i-1} = \dot{d}_i \cos\hat{\phi}_{i-1}$ using (4.43).

This proves (4.44).

From (4.17b), $\dot{d}_{ik}^{\check{i}-1} = \dot{d}_{ik}^{\hat{i}-1} \cos\check{\phi}_{i-1} = \dot{d}_i^{\hat{i}-1} \cos\check{\phi}_{i-1}$ using (4.44).

This proves (4.45).

From (4.17a), $d_{ik} = \check{d}_{ik}^{i-1} \cos\check{\phi}_i = \check{d}_i^{i-1} \cos\check{\phi}_i$ using (4.45).

This proves (4.46).

The proof of (4.47),(4.48) is similar. QED.

Outline of FTO5

Procedures 4.2 and 4.3 from the basis of a recursion by which R and $R^{(n-1)}$ may be calculated in approximately half the number of operations required by FTO3. The algorithm is, in outline

1. Initialize $\check{\phi}_1$, w_{11}, d_1, θ_1, $\bar{w}_{11}$ and $\bar{d}_1$.

2. For $k \leftarrow 2$ to $n-1$ do 2.1-2.4, then for $k=n$ do 2.1 and 2.2:

   2.1 Calculate $\hat{\phi}_k$ as in FTO3.

   2.2 Calculate $\Phi(\hat{\phi}_{k-1})$, $\underline{w}_{\cdot k}$, d_k, $\Phi(\check{\phi}_{k-1})$ and $\hat{\phi}_k$ as in procedure 4.2.

   2.3 Calculate θ_k as in FTO3.

   2.4 Calculate F.I.T. for θ_k, $\underline{\bar{r}}_{\cdot k}$, $\bar{d}_k$ and $\tilde{d}_n^{(k)}$ as in procedure 4.3.

FTO5 - The Algorithm

All of the steps have been previously proved (in FTO3) or are trivial to show, except for step 9.2. Step 9.2 is proved below.

Algorithm 4.1 - Fast Toeplitz Orthogonalization, Version 5 (FTO5)

(Programmed on p.A.84)

{Initialization}

1. $\dot{\sigma}_2 \leftarrow \|\underline{t}_{2:n,1}\|$

2. $\bar{r}_{11} \leftarrow \|\underline{t}_{1:n-1,1}\|$

3. $r_{11} \leftarrow (\bar{r}_{11}^2 + t^2)^{\frac{1}{2}}$ {recall that $t := t_{11}$}

```
4.  $\tan\check{\phi}_1 \leftarrow \dot{\sigma}_2/t$ ; $d_1 \leftarrow \cos\check{\phi}_1 \leftarrow t/r_{11}$   $\dot{d}_1 = 1$; $\dot{\check{d}}_1^0 = 1$

5.  $w_{11} \leftarrow r_{11}/d_1$

6.  $\cos\theta_1 \leftarrow \bar{r}_{11}/r_{11}$; $\tan\theta_1 \leftarrow v_{n-1}/\bar{r}_{11}$

7.  $\bar{d}_{11} \leftarrow d_1/\cos\theta_1$

8.  $\bar{w}_{11} = \bar{r}_{11}/\bar{d}_{11}$

{Main loop}

9.  For $k \leftarrow 2$ to $n-1$ do

    9.1  $\dot{\underline{w}}_{1:k,k} \leftarrow \begin{pmatrix} u_{k-1} \\ \bar{\underline{w}}_{1:k-1,k-1} \end{pmatrix}$ ; $\dot{d}_k \leftarrow \bar{d}_{k-1}$

    9.2  if $k \neq n$ do {calculate $\hat{\phi}_k$}

             $s_{k-1} \leftarrow (z_{k-1} - \sum_{i=1}^{k-2} \bar{w}_{i,k-1}\, s_i)/\bar{w}_{k-1,k-1}$, where $\underline{z} := T_{n-1}^T\, \underline{t}_{2:n,1}$;

             $\dot{r}_{k1} \leftarrow s_{k-1}/\bar{d}_{k-1}$

    9.3  $\dot{\sigma}_{k+1} \leftarrow (\dot{\sigma}_k^2 - \dot{r}_{k1}^2)^{\frac{1}{2}}$

    9.4  $\tan\hat{\phi}_k \leftarrow \dot{\sigma}_{k+1}/\dot{r}_{k1}$ ;  $\cos\hat{\phi}_k \leftarrow \dot{r}_{k1}/\dot{\sigma}_k$

    9.5  Calculate $\Phi(\hat{\phi}_{k-1})$, $\Phi(\check{\phi}_{k-1})$, $\underline{w}_{.k}$, $d_k$ and $\check{\phi}_k$ using procedure 4.2.
```

9.6 Calculate $\bar{\underline{r}}_{.k}$, $\bar{d}_k$, $\tilde{d}_n^{(k)}$ and $\check{\theta}_k$ using procedure 4.3.

Proof of step 9.2

We showed in Chapter 7, FTO2, that

$$\dot{\underline{r}}_{2:n,1} = R^{(n-1)-T}\underline{z}, \text{ where } \underline{z} := T^T\underline{t}_{2:n,1} \tag{4.49}$$

or $\quad R^{(n-1)T}\dot{\underline{r}}_{2:n-1} = \underline{z}$,

so $\quad \bar{R}^T_{n-1}\dot{\underline{r}}_{2:n-1} = \underline{z}$, which by (4.7) becomes $\qquad$ (4.50)

$$(\bar{D}_{n-1}\bar{W}_{n-1})^T\dot{\underline{r}}_{2:n-1} = \underline{z}$$

or $\quad \bar{W}^T_{n-1}\ \underline{s} = \underline{z}$, where $\underline{s} := \bar{D}_{n-1}\dot{\underline{r}}_{2:n-1}$ $\qquad$ (4.51)

(4.51) may be solved by back-substitution which is step 9.2.

Operation Count

The operation count for FTO1A is $4\frac{1}{2}n^2 + O(n)$, compared to $7\frac{1}{2}n^2 + O(n)$ for FTO3.

4.3 Incorporation of FGT's into FTO4

Recall that FTO4 calculates Q (and $Q^{(n-1)}$) as well as R (and $R^{(n-1)}$). FTO3 is essentially FTO4 plus a recursion which calculates $Y := Q^T$ and $Y^{(n-1)} := Q^{(n-1)T}$ row-by-row. If we incorporate FGT's and FIT's into the Y, $Y^{(n-1)}$ recursion, and append the new recursion to FTO3, we have an algorithm which calculates Q and R in approximately half the number of operations of FTO4.

Incorporation of FGT's into the first part of Y-recursion

The Y-recursion is given in eqs. (3.19)-(3.12),(3.14),(3.15). Eqs. (3.9) and (3.10) may be rearranged to give

$$\begin{bmatrix} \dot{\underline{y}}^{(1)}_{j+1.} \\ \\ \dot{\underline{y}}^{(2)}_{j+1.} \end{bmatrix} = \begin{bmatrix} -cot\hat{\phi}_j & cosec\hat{\phi}_j \\ \\ -cosec\hat{\phi}_j & cot\hat{\phi}_j \end{bmatrix} \begin{bmatrix} \dot{\underline{y}}_{j.} \\ \\ \dot{\underline{y}}^{(1)}_{j.} \end{bmatrix} \tag{4.52}$$

We now derive a 'fast' form for (4.52), in analogy to the way the FGT was derived in (4.2). Suppose $\dot{\underline{y}}_j = \dot{f}_j\,\dot{\underline{g}}_{j.}$ and $\dot{\underline{y}}^{(1)}_{j.} = \dot{f}^{(1)}_j\,\dot{\underline{g}}^{(1)}_{j.}$, where $\dot{f}_j$ and $\dot{f}^{(1)}_j$ are scaling factors. Then (4.52) can be written

$$\begin{bmatrix} \dot{\underline{y}}^{(1)}_{j+1.} \\ \\ \dot{\underline{y}}^{(2)}_{j+1} \end{bmatrix} = \begin{bmatrix} -cot\hat{\phi}_j & cosec\hat{\phi}_j \\ \\ -cosec\hat{\phi}_j & cot\hat{\phi}_j \end{bmatrix} \begin{bmatrix} \dot{f}_j & 0 \\ \\ 0 & \dot{f}^{(1)}_j \end{bmatrix} \begin{bmatrix} \dot{\underline{g}}_{j.} \\ \\ \dot{\underline{g}}^{(1)}_{j.} \end{bmatrix}$$

$$= \begin{bmatrix} -\dot{f}_j cot\hat{\phi}_j & \dot{f}^{(1)}_j\,cosec\hat{\phi}_j \\ \\ -\dot{f}_j cosec\hat{\phi}_j & \dot{f}^{(1)}_j\,cot\hat{\phi}_j \end{bmatrix} \begin{bmatrix} \dot{\underline{g}}_{j.} \\ \\ \dot{\underline{g}}^{(1)}_{j.} \end{bmatrix}$$

$$= \begin{bmatrix} \dot{f}_j cot\hat{\phi}_j & 0 \\ \\ 0 & \dot{f}^{(1)}_j cot\hat{\phi}_j \end{bmatrix} \begin{bmatrix} -1 & (sec\hat{\phi}_j)\dot{f}^{(1)}_j/\dot{f}_j \\ \\ -(sec\hat{\phi}_j)\dot{f}_j/\dot{f}^{(1)}_j & 1 \end{bmatrix} \begin{bmatrix} \dot{\underline{g}}_{j.} \\ \\ \dot{\underline{g}}^{(1)}_{j.} \end{bmatrix},$$

so $\dot{\underset{\sim}{y}}^{(1)}_{j+1.}$ and $\dot{\underset{\sim}{y}}^{(2)}_{j+1.}$ can be represented as $\dot{f}^{(1)}_{j+1}\,\dot{\underset{\sim}{g}}^{(1)}_{j+1.}$ and $\dot{f}^{(2)}_{j+1}\,\dot{\underset{\sim}{g}}^{(2)}_{j+1.}$, where

$$\dot{f}^{(1)}_{j+1} = \dot{f}_j \cot\hat{\phi}_j \; ; \; \dot{f}^{(2)}_{j+1} = \dot{f}^{(1)}_j \cot\hat{\phi}_j \qquad (4.53)$$

and

$$\begin{pmatrix} \dot{\underset{\sim}{g}}^{(1)}_{j+1.} \\ \\ \dot{\underset{\sim}{g}}^{(2)}_{j+1.} \end{pmatrix} = \begin{pmatrix} -1 & (\sec\hat{\phi}_j)\dot{f}^{(1)}_j/\dot{f}_j \\ \\ -(\sec\hat{\phi}_j)\dot{f}_j/\dot{f}^{(1)}_j & 1 \end{pmatrix} \begin{pmatrix} \dot{\underset{\sim}{g}}_{j.} \\ \\ \dot{\underset{\sim}{g}}^{(1)}_{j.} \end{pmatrix} \qquad (4.54)$$

Eqs. (4.53) and (4.54) are the desired 'fast' form for (4.52), requiring $2n + O(1)$ operations, compared to (4.52).

We next write out the fast form of (3.11) and (3.12). Let $\dot{\underset{\sim}{y}}^{(3)}_{j.} = \dot{f}^{(3)}_j\,\underset{\sim}{g}^{(3)}_{j.}$, where $\underset{\sim}{g}^{(3)}_{j.}$ is a scaling factor. Eqs. (3.11) and (3.12) constitute a Givens transform, so we can write down the corresponding FGT using (4.2):

$$f_j = \dot{f}^{(3)}_j \cos\check{\phi}_j \; ; \; \dot{f}^{(3)}_{j+1} = \dot{f}^{(2)}_{j+1}\cos\check{\phi}_j \qquad (4.55)$$

$$\begin{pmatrix} \underset{\sim}{g}_{j.} \\ \\ \dot{\underset{\sim}{g}}^{(3)}_{j+1.} \end{pmatrix} = \begin{pmatrix} 1 & (\tan\check{\phi}_j)\dot{f}^{(2)}_{j+1}/\dot{f}^{(3)}_j \\ \\ -(\tan\check{\phi}_j)\dot{f}^{(3)}_j/\dot{f}^{(2)}_{j+1} & 1 \end{pmatrix} \begin{pmatrix} \dot{\underset{\sim}{g}}^{(3)}_{j.} \\ \\ \dot{\underset{\sim}{g}}^{(2)}_{j+1.} \end{pmatrix} \qquad (4.56)$$

To accelerate (3.14), we note that they constitute an inverse transform with angle θ. We may write down the corresponding fast inverse transform directly using (4.31). This, combined with (4.53-4.56), constitute a recursion by which the $f_{j.}$ and $\underset{\sim}{g}_{j.}$ and therefore the $\underset{\sim}{y}_{j.}\,(=f_j\underset{\sim}{g}_{j.})$ can be calculated in approximately half the number of operations as in FT04. Combining FT05 with this recursion therefore gives an algorithm that calculates Q and R in approximately half the number of operations as FT04. We do not give the detailed algorithm here.

5. TOEPLITZ ORTHOGONALIZATION - SOME EXTENSIONS

We now describe three extensions to the FTO algorithms, which hitherto have been used to orthogonalize a square Toeplitz matrix. We consider (i) the orthogonalization of a rectangular Toeplitz matrix, (ii) the related Toeplitz least-squares problem, and (iii) orthogonalization of block Toeplitz matrices.

5.1 Rectangular Toeplitz Matrices

The orthogonalization of a rectangular Toeplitz matrix is described by the next two theorems.

Theorem 5.1 Let T be a full-rank $m \times n$ Toeplitz matrix, $m \geq n$. Let

$$T = QR \tag{5.1}$$

where Q has n orthogonal columns (of length m) and R is an $n \times n$ upper triangle. Then:

(i) R may be calculated by applying n cycles of FTO1 or FTO3 to T^*, where T^* is any $m \times m$ Toeplitz extension of T.

(ii) Q and R may be calculated by applying n cycles of FTO2 or FTO4 to T^*.

Proof: Let

$$T^* = Q^* R^* . \tag{5.2}$$

Applying the first n cycles of FTO1 or FTO3 to T^* yields $R^*_{.1:k}$. From (5.2), $T^*_{1:k.} = T = Q^* R^*_{.1:k} = Q^*_{.1:k} R^*_{1:k,1:k}$ which is a factorization of the form (5.1), so $R = R^*_{1:k,1:k}$. This proves (i). The proof of (ii) is similar. QED.

Remark: The method can be modified to handle some cases where rank $T < n$. We conjecture that it can be extended to handle any T.

Theorem 5.2 Let T be an $m{\times}n$ Toeplitz matrix. Let $T = \dot{Q}\dot{R}$, where $\dot{Q}$ is orthogonal $(m{\times}m)$ and $\dot{R}$ is an $m{\times}n$ matrix with $\dot{r}_{ij}=0$, $i>j$. Then;

(i) $\dot{R}$ alone may be calculated by applying FTO1 or FTO3 to T^*, a Toeplitz matrix with T as its first n columns, and

(ii) $\dot{Q}$ and $\dot{R}$ may be calculated by applying FTO2 or FTO4 to T^*.

Proof: Let $T^* = Q^*R^* \Rightarrow T = T^*_{.1;k} = Q^*R^*_{.1;k}$ which has the form $T=\dot{Q}\dot{R}$, so applying FTO1 or FTO3 to T^* yields the desired matrix R, and applying FTO2 or FTO4 to T^* yields Q and R.

QED.

5.2 Solution of the Toeplitz Least Squares Problem

Let T be an $m{\times}n$ rectangular Toeplitz matrix with $m > n$. The least-squares problem seeks that $\underline{x}$ which minimizes

$$\underline{e} = \|T\underline{x} - \underline{b}\|_2 , \tag{5.4}$$

where $\underline{x}$ has length n and $\underline{b}$ has length m. One example is the covariance method of linear predictive coding. Morf et al [67], has proposed the efficient solution of the normal equation

$$T^T T\underline{x} = T^T\underline{b} \tag{5.5}$$

by making use of the 'close-to-Toeplitz' structure of $T^T T$. Lee et al. proposed methods to minimize $\|\underline{e}\|_2$ by various ladder (or lattice) recursions [58]. The Toeplitz least-squares problem may also be solved by making use of R, or both R and Q as calculated by FTO.

Finding $\underline{x}$ using R only

Let $T = QR$ where Q is $m{\times}n$ and R is $n{\times}n$. Then (5.5) becomes

$$R^T Q^T QR\underline{x} = T^T\underline{b}$$

$$\Rightarrow \quad \underline{x} = R^{-1}R^{-T}T^T\underline{b} \tag{5.6}$$

which enables $\underline{x}$ to be calculated in $O(mn)$ operations. Since (5.6) is based on the normal equations (5.5), the condition number of the problem is $cond^2 T$, which may cause difficulties when $cond\, T$ is large. A better approach, first proposed by Golub [34] for general matrices is described next.

Finding $\underline{x}$ using Q and R

Golub [34] shows that $\underline{x}$ is the solution of

$$R\underline{x} = Q^T\underline{b} \tag{5.7}$$

The error term may be found by direct calculation, or alternatively, if $\dot{Q}$ (an orthogonal matrix with Q as its first n columns) is calculated as in Theorem 5.2, then $\|\underline{e}\|_2 = \|\dot{Q}^T_{.n+1:m}\,\underline{b}\|_2$. (5.8)

5.3 Block Toeplitz Matrices

The development of the recursion for block Toeplitz matrices ia analogous to that for scalar Toeplitz matrices. Let p be the block-size. The basic block-operation is performed by applying a sequence of p Householder transforms, instead of the Givens transform used in the scalar case. In analogy to eq. (7.3.9) we can show that*

$$\underbrace{\left[\begin{array}{c|c} T & \underline{U}^T \\ \hline Q^{(n-1)T}\underline{V} & R^{(n-1)} \end{array}\right]}_{\dot{R}} \xrightarrow[\dot{S}^T]{\text{Block GGMS ops}} \underbrace{R \to R - \underline{Q}^T_{(n.)}(\underline{V}^{BRT},0)}_{\tilde{R}} \longrightarrow$$

$$\xrightarrow[\tilde{S}^T]{\text{Block GGMS ops}} \underbrace{\left[\begin{array}{c|c} R^{(n-1)} & Q^{(n-1)R}\underline{U}^{BR} \\ \hline \underline{0}^T & T \end{array}\right]}_{\bar{R}} , \tag{5.9}$$

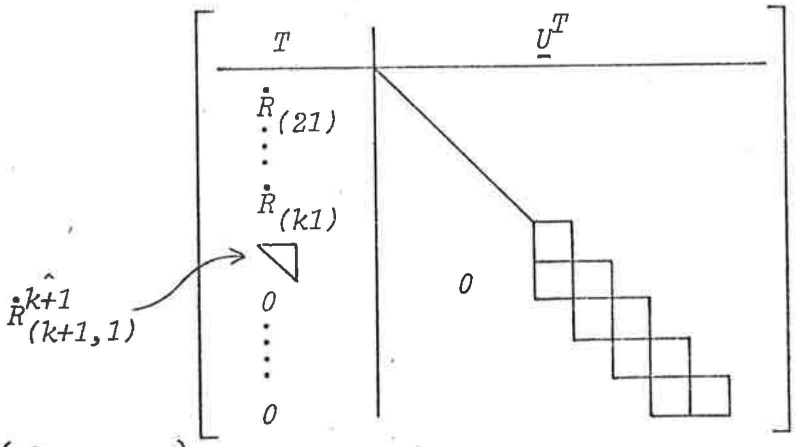
*Block matrices are denoted by script letters, block vectors by underlined capitals. BR and BRT are block-reverse and block-reverse-transpose respectively. $\underline{Q}_{(n\cdot)}$ is the nth block-row of Q.

where T, the block-Toeplitz matrix with block-order n and block-size p is partitioned

$$T := \left(\begin{array}{c|c} T & \underline{U}^T \\ \hline \underline{V} & T_{(n-1)} \end{array}\right) = \left(\begin{array}{c|c} T_{(n-1)} & \underline{U}^{BR} \\ \hline \underline{V}^{BRT} & T \end{array}\right)$$

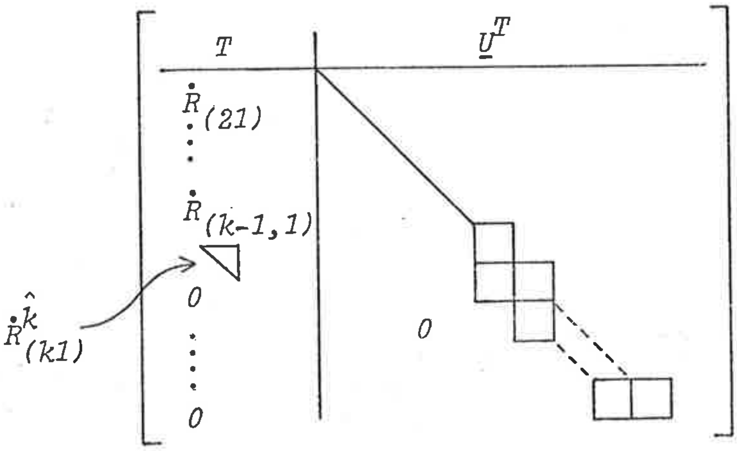
T is the $(1,1)$ block of T, $\underline{Q}_{(n,)}$ is the nth block-row of Q and BR denote block-reversal. At a typical stage in phase I the block GGMS reduction of $\dot{R}$, we have the form*

$\dot{R}^{\widehat{k+1}} =$



The matrix $\begin{pmatrix} \dot{R}_{(k1)} \\ \dot{R}^{\widehat{k+1}}_{(k+1,1)} \end{pmatrix}$ can be reduced to upper-triangular form by p householder transforms, yielding

$\dot{R}^{\hat{k}} =$



*$A_{(ij)}$ is the i,jth block-element of A.

The basic GGMS block-operation is performed by applying this sequence of p householder transforms. In Phase II of the block GGMS algorithm, a block upper-Hessenberg matrix is reduced to an upper-triangle. This can be done, in analogy to the scalar case, by $n-1$ GGMS block-operations starting at the top of the block-diagonal and working to the bottom.

From (3.9), the basic recursion can be developed, in analogy to (7.3.12)*

$$\underbrace{\begin{pmatrix} U_{(k-1)} \\ \\ \dot{\underline{R}}^{(n-1)}_{(.k)} \end{pmatrix}}_{\underline{R}^{(\widehat{k+1})}_{(.k)}} \xrightarrow[S_1^T(\hat{k},\check{k})]{block\ ops} \underline{R}_{(.k)} \rightarrow \underbrace{\begin{pmatrix} \underline{R}_{(1:k,k)} - V_{(n-k)}\underline{Q}^T_{(n,1:k)} \\ A_{(k)} \\ \underline{0} \end{pmatrix}}_{\underline{R}^{\widehat{k+1}}_{(.k)}} \longrightarrow$$

$$\xrightarrow[S_2^T(\hat{k},\check{k})]{block\ ops} \underbrace{\begin{pmatrix} \underline{R}^{(n-1)}_{(.k)} \\ \\ \underline{0} \end{pmatrix}}_{\bar{\underline{R}}_{(.k)}} \tag{5.10}$$

where $S_1^T(k,k)$ and $S_2^T(k,k)$ are the products of the block operations from the kth upsweep operation to the kth downsweep operation. The recursion (5.10) requires $4k$ block-operations to execute, so the whole algorithm requires $2k$ block-operations.

Here, we will only indicate how the block-operations in the upsweeps of the block FTO, are calculated. All other steps are straightforward and analogous to FTO1.

*$\underline{A}_{(.k)}$ is the kth block-column of A; $\underline{A}_{(1:k,k)}$ consists of the first k block-elements of $\underline{A}_{(.k)}$.

Calculation of Upsweep Block-Operations (UBO's)

It is sufficient to show how to calculate the UBO in the first GGMS transformation in (5.9). The UBO's in the second GGMS are calculated in the same way. At the beginning of step k, we have $\dot{R}_{(k1)}$ and $\dot{R}^{\hat{k}}_{(k1)}$. We wish to calculate $\dot{R}^{\widehat{k+1}}_{(k+1,1)}$ and the set of p to Householder transforms by which

$$\begin{pmatrix} \dot{R}_{(k1)} \\ \\ \dot{R}^{\widehat{k+1}}_{(k+1,1)} \end{pmatrix} \text{ is transformed to } \begin{pmatrix} \dot{R}^{\hat{k}}_{(k1)} \\ \\ 0 \end{pmatrix}.$$

It is sufficient to show what to do for the first Householder transforms and first row of $\dot{R}^{\widehat{k+1}}_{(k+1,1)}$ - the other desired quantities are calculated similarly.

By norm-invariance, we have*

$$\dot{r}^{\hat{k}}_{(k1)11} = [\|\dot{\underline{r}}_{(k1).1}\|^2 + (\dot{r}^{\widehat{k+1}}_{(k+1,1)11})^2]^{\frac{1}{2}}$$

$$\text{so} \quad \dot{r}^{\widehat{k+1}}_{(k+1,1)11} = [(\dot{r}^{\hat{k}}_{(k1)11})^2 - \|\dot{\underline{r}}_{(k1).1}\|^2]^{\frac{1}{2}} \qquad (5.11)$$

Let

$$A := \begin{pmatrix} \dot{R}_{(k1)} \\ \\ \dot{\underline{r}}^{\widehat{k+1}}_{(k+1,1)1.} \end{pmatrix}$$

Recall that the blocks are of order p.
$\underline{a}_{p+1.}$ is unknown, except for $a_{p+1,1} = \dot{r}^{\widehat{k+1}}_{(k+1,1)11}$.

In the Householder method $\underline{a}_{2:p+1,1}$ is eliminated by forming

*$a_{(ij)kl}$ is the klth element of the i,j block of A.

$$A^{(1)} = (I - \underline{u}\underline{u}^T/2K^2)A, \text{ where } \underline{u} = [a_{11} \pm s_1\ a_{21}, \ldots a_{p+1,1}]^T,$$

where $s := \|\underline{a}_{.1}\|$

so $$A^{(1)} = A - \frac{\underline{u}\underline{u}^T A}{2K^2} = A - \frac{\underline{u}\underline{c}^T}{2K^2}, \text{ where } \underline{c}^T := \underline{u}^T A. \qquad (5.12)$$

therefore $\underline{a}^{(1)}_{1.} = \underline{a}_{1.} - \dfrac{u_1 \underline{c}^T}{2K^2}$,

$$\underline{c}^T = \frac{2K^2}{u_1}(\underline{a}_{1.} - \underline{a}^{(1)}_{1.}) = \frac{2K^2}{u_1}\ (\dot{r}_{(k,1)1.} - \dot{\underline{r}}^{\hat{k}}_{(k,1)1.}) \qquad (5.13)$$

which can be calculated from known quantities.

From (5.12), therefore $\underline{a}^{(1)}_{i.}$, $i=2,\ldots,p$ can be calculated from known quantities:

$$\underline{a}^{(1)}_{1.} = \underline{a}_{i.} - \frac{u_i \underline{c}^T}{2K^2}, \quad i=2,\ldots,p \qquad (5.14)$$

and from (5.12) $$\dot{\underline{r}}^{\widehat{k+1}}_{(k+1,1)1.} = \underline{a}_{p+1.} = (\underline{c}^T - \sum_{i=1}^{p} u_i\ \underline{a}_{i.})/u_{p+1} \qquad (5.15)$$

Eqs. (5.11), (5.13), (5.14) and (5.15) enable us to calculate the first Householder transform and the first row of $\dot{R}^{\widehat{k+1}}_{(k+1,1)}$. The other Householder transforms and the rest of $\dot{R}^{\widehat{k+1}}_{(k+1,1)}$ can be calculated in the same way.

Block FTO - Concluding Remarks

A block FTO analogous to FTO1 can be written from the recursion in (5.10). The block-operations in the upsweeps are calculated by the method outlined above. The total operation count for the block FTO is $3\tfrac{1}{2}p^3n^2$. Other block FTO algorithms analogous to FTO1 to FTO4 can be developed.

6, CONCLUSION

We have described two ways to accelerate FTO algorithms developed in Chapter 7. By using a logically more complex recursion, we can reduce the computing load by about 25%. By incorporating Fast Givens Transforms, we can reduce the computing load by a further 50% (approximately). We have also described some sundry extensions to the FTO algorithms - orthogonalization of rectangular Toeplitz matrices, solution of the Toeplitz least-squares problem, and orthogonalization of block-Toeplitz matrices.

CHAPTER 9

THE SINGULAR-VALUE DECOMPOSITION OF TOEPLITZ MATRICES

1. INTRODUCTION

Let A be a real $m\times n$ matrix, $m\geq n$. It is well-known [37] that $\exists$ U,Σ and V with dimensions $m\times n$, $n\times n$ and $n\times n$ respectively such that

$$A = U\Sigma V^T \tag{1.1}$$

where $U^TU = V^TV = VV^T = I_n$ and $\Sigma = diag(\sigma_1,\ldots,\sigma_n)$. The matrix U consists of the n orthonormalized eigenvectors associated with the n largest eigenvalues of AA^T, and the matrix V consists of the orthonormalized eigenvectors of A^TA. The diagonal elements of Σ are the non-negative square roots of the eigenvalues of A^TA; they are called the singular values. We shall assume that

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0.$$

The decomposition (1.1) is called the singular value decomposition (SVD).

There are alternative representations [37] to that given by (1.1); however (1.1) is the most useful for computational purposes, so we only consider this form here.

The SVD provides a very convenient and stable method of solving several common problems of linear algebra. Some of these are [35]:

(i) solving a set of linear equations with a nonsingular matrix, particularly when the system is ill-conditioned;

(ii) more generally, analyzing the system $A\underline{x}=\underline{b}$ where A is a general $m\times n$ matrix: the SVD can be used to determine whether the system is consistent, whether the solution is unique, and what the general form of the solution is;

(iii) solving the least-squares problem: find $\underline{x}$ such that $\| A\underline{x}-\underline{b} \|_2$ and $\| \underline{x} \|_2$ are minimized, where A is the general $m\times n$ matrix with $m>n$;

(iv) finding pseudo-inverses [35]; and

(v) approximating matrices by matrices of lower rank.

Recently it has been found desirable to calculate the SVD of Toeplitz or Hankel matrices. Two such applications are rational Chebyshev approximation on the unit disk [83] and in Hankel-norm reductions of system theory [57].

In such applications, an SVD algorithm which takes advantage of the Toeplitz structure to reduce the number of arithmetic operations would be useful. To the author's knowledge such an algorithm is not at present available. In section 2, we review the SVD algorithm of Golub-Reinsch [37] and a modification thereof by Chan [17]. In sections 3 and 4 we present two algorithms which are further modifications of the Golub-Reinsch algorithm and take advantage of the Toeplitz structure to reduce the computing load by 50-80% or more, depending on the parameters of the problem.

2. THE GOLUB-REINSCH ALGORITHM

The Golub-Reinsch algorithm (GR) [37] is a very stable method for finding the SVD. GR has two phases - a direct phase in which A is reduced to a bidiagonal matrix $J^{(0)}$, and an iterative phase in which $J^{(0)}$ is reduced to the diagonal matrix Σ. Both the techniques described in this report use the Toeplitz property to "speed-up" the first phase of the Golub-Reinsch algorithm. The second phase is unchanged, except for re-orderings in the sequence of computation of certain matrix products. We describe GR below, and later a modification by Chan [17] which is faster when $m>2n$.

Algorithm 2.1 - The Golub-Reinsch Algorithm (GR)

Phase I (direct) - reduction to bidiagonal form:

A is transformed by two sequences of Householder Transforms [89] $\{P^{(k)}\}_1^n$, *and* $\{Q^{(k)}\}_1^{n-2}$, such that

$$P^{(n)}\text{---}P^{(1)}AQ^{(1)}\text{---}Q^{(n-2)} = \left(\begin{array}{cccc} x & x & & 0 \\ & \ddots & \ddots & \\ & & & x \\ 0 & & & x \\ \hline & & 0 & \end{array}\right) = J^{(0)}, \tag{2.1}$$

an upper-bidiagonal matrix. If we let $A^{(1)}=A$ and define

$$A^{(k+\frac{1}{2})} = P^{(k)}A^{(k)} \qquad (k=1,2,\dots n)$$

$$A^{(k+1)} = A^{(k+\frac{1}{2})}Q^{(k)} \qquad (k=1,2,\dots,n-2)$$

then $P^{(k)}$ is determined such that $a_{ik}^{(k+\frac{1}{2})}=0,\ (i=k+1,\dots,m)$

and $Q^{(k)}$ is determined such that $a_{kj}^{(k+\frac{1}{2})}=0,\ (j=k+2,\dots,n)$

It can be checked that no previously-nulled elements are filled at any stage.

The singular values of $J^{(0)}$ are the same as those of A. Thus if the SVD of

$$J^{(0)} = G\Sigma H^T \tag{2.2}$$

then $$A = PG\Sigma H^T Q^T$$

so that $U=PG$, $V=QH$ with $P:=P^{(1)}\dots P^{(n)}$, $Q:=Q^{(1)}\dots Q^{(n-2)}$.

Phase II (iterative) - reduction to diagonal form:

$J^{(0)}$ is iteratively diagonalized by a variant of the QR method so that

$$J^{(0)} \to J^{(1)} \to --- \to \Sigma \tag{2.4}$$

where $$J^{(i+1)} = S^{(i)T}J^{(i)}T^{(i)}$$

where $S^{(i)}$ and $T^{(i)}$ are products of plane rotations and are therefore orthogonal. The $\{T^{(i)}\}$ are chosen so that the sequence $M^{(i)}=J^{(i)T}J^{(i)}$ converges to a diagonal matrix while the matrices $S^{(i)}$ are chosen so that all $J^{(i)}$ are of bidiagonal form. The products of the $T^{(i)}$'s and $S^{(i)}$'s are the matrices H^T and G^T respectively in (2.2). Details are given in [37].

Operation Counts:

In discussing operation counts, we count only the number of multiplications and divisions, as on most computers, the addition/subtraction time is much less than the multiplication or division time. Moreover, it can be checked that the number of additions/subtractions is about the same as the number of multiplications/divisions in all the algorithms discussed herein. Thus the number of multiplications and divisions is a reasonable measure of computing time. We also ignore all lower-order terms in operation counts, e.g. $3mn^2 + O(mn) + O(n^2)$ will be written as $3mn^2$.

In GR, Chan [17] gives the following operation counts for the various parts of the algorithm:

I. Bidiagonalization

Reduction from A to $J^{(0)}$	$2(mn^2-n^3/3)$	operations
Accumulation of $P=P^{(1)}\ldots P^{(n)}$	$mn^2-n^3/3$	operations
Accumulation of $Q=Q^{(1)}\ldots Q^{(n-2)}$	$2n^3/3$	operations

II. Diagonalization - operations per iteration

Accumulation of $S^{(i)}$ on P	$2mn$ operations
Accumulation of $T^{(i)}$ on Q	$2n^2$ operations
Reduction from $J^{(0)}$ to Σ	$0(n)$ operations

It is reported in [37] that the average number of iterations, k, required for II is usually less than $2n$. Chan has run the Golub-Reinsch algorithm with $k=2n$ and has found it quite accurate.

The Golub-Reinsch Algorithm (GR) can be speeded up by using "fast" Givens transforms [30] in Phase II. In this case, all the operation counts in this phase are halved.

Table 2.1 shows the total operation count for GR, assuming that $k=2n$, with and without accumulation of U and V, using slow and fast Givens transforms in Phase II:

	Explicit Calc. of U & V	No explicit Calc. of U & V
Slow Givens Transforms	$7mn^2 + 4n^3$	$2(mn^2 - \frac{n^3}{3})$
Fast Givens Transforms	$5mn^2 + 2n^3$	$2(mn^2 - \frac{n^3}{3})$

Table 2.1 - Operation Counts for Golub-Reinsch Algorithm assuming that $k=2n$.

Chan's Modification of GR (GRC)

Chan proposes a modification of GR which is more efficient than GR when m/n is greater than about 2, and is about twice as efficient when m/n is about 10. His procedure is:

Algorithm 2.2 - The Golub-Reinsch-Chan Algorithm (GRC)

1. Convert A to upper-triangular form R using Householder Transforms, so that

$$Q^T A = R$$

where Q is the (orthogonal) product of the Householder Transforms (Q is not calculated explicitly).

2. Find the SVD of R using GR such that

$$R = W\Sigma Y^T$$

3. Then the SVD of A is

$$A = U\Sigma V^T,$$

where
$$U = Q \begin{bmatrix} X \\ \hline 0 \end{bmatrix} \Big\}m$$
$$\underbrace{}_{n}$$

and
$$V = Y.$$

GRC has an extra phase at the beginning, viz. triangularization of A, which requires $mn^2 - \frac{n^3}{3}$ operations. However, many more operations than these are saved in step 2 when $\frac{m}{n}$ is large, because the algorithm then operates on an upper-triangular matrix with $\frac{n}{2}(n+1)$ non-zero elements instead of a full $m\times n$ matrix. Table 2.2 shows the operation counts for GRC, assuming that $k=2n$:

	Explicit Calc. of U & V	No Explicit Calc. of U & V
Slow Givens Transforms	$3mn^2 + 9\frac{1}{3}n^3$	$mn^2 + n^3$
Fast Givens Transforms	$3mn^2 + 5\frac{1}{3}n^3$	$mn^2 + n^3$

Table 2.2 - Operation Counts for Golub-Reinsch Algorithm as modified by Chan, assuming that $k=2n$.

3. TOEPLITZ SVD BY FAST TOEPLITZ ORTHOGONALIZATION

To calculate the SVD of a Toeplitz matrix A, GRC may be accelerated by using Fast Toeplitz Orthogonalization (FTO), together with two other modifications, as described below:

(1) In step (1) of GRC, convert A to R and calculate $Q._{1:n}$ using* Fast Toeplitz Orthogonalization (FTO) rather than Householder Transforms. This will require $O(mn)$ operations rather than $mn^2-n^3/3$ operations in GRC.

(2) If U and V are not required explicitly, replace phase I of step 2 (bidiagonalization of R) by the following procedure: Use fast Givens transforms to zero out the elements of R above the superdiagonal $\{r_{ij}\}_{j=i+1}$ either row-by-row or column-by-column, starting with r_{1n}. To illustrate, suppose we are eliminating the $\{r_{ij}\}$ row-by-row and we wish to eliminate r_{pq}. We perform two fast Givens transforms as follows:

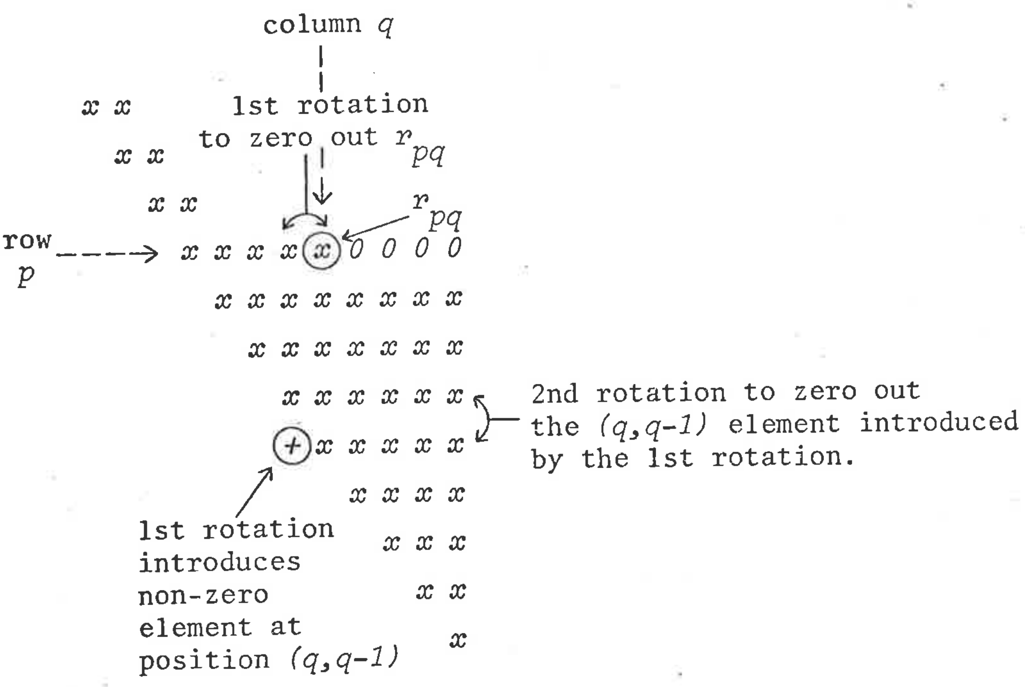

column q
1st rotation to zero out r_{pq}
r_{pq}
row p
2nd rotation to zero out the $(q,q-1)$ element introduced by the 1st rotation.
1st rotation introduces non-zero element at position $(q,q-1)$
x
0
+

Figure 3.1 - Elimination of r_{pq}

*$Q._{1:n}$ denotes columns 1 to n of Q.

This procedure requires $\frac{2}{3}n^3$ operations, compared to the $\frac{4}{3}n^3$ operations required by GR in bidiagonalizing R without taking advantage of its structure. This procedure, using slow Givens transforms was suggested by Chan, but rejected because there was no computational advantage over GR. The simple modification of using fast Givens transforms saves the $\frac{2}{3}n^3$ operations here.

(3) If U is required explicitly, calculate the product $Q\begin{pmatrix} x \\ -- \\ 0 \end{pmatrix}$ directly instead of using Householder transforms, as in GRC. This is possible since $Q_{\cdot 1:n}$ is available from step 1. This procedure requires mn^2 operations, rather than the $2mn^2-n^3$ operations required by GRC.

We present the modified procedure:

Algorithm 3.1 - Modified Golub-Reinsch-Chan Algorithm (MGRC)

1. Perform the decomposition $A=Q_{\cdot 1:n}R$ using Fast Toeplitz Orthogonalization.
2. If U and V are not both required explicitly do step 2A; otherwise do step 2B:

   2A. Use fast Givens transforms to zero out the elements of R above the superdiagonal row-by-row or column-by-column, starting with r_{1n}. The elimination is performed as in Fig.3.1.

   2B. Transform R to upper-bidiagonal form using Householder transforms, as in *GR*. Accumulate the row-transforms on X^T, and the column transforms on Y^T.
3. Diagonalize the bidiagonal matrix as in GR.
4. If U is required explicitly, compute $U=Q_{\cdot 1:n}X$. Note that $V=Y$.

Operation Counts

We compare operation counts for GR, GRC and MGRC in Table 3.1. Fast Givens transforms are used throughout in the bidiagonalization phase:

Calc. of U & V Algorithm	Explicit Calculation of U and V	No Explicit Calculation of U and V
GR	$5mn^2 + 2n^3$	$2(mn^2 - \frac{n^3}{3})$
GRC	$3mn^2 + 5\frac{1}{3}n^3$	$mn^2 + n^3$
MGRC	$mn^2 + 6\ n^3$ $+ 0(mn)$	$\frac{2n^3}{3} + 0(mn)$

Table 3.1 - SVD of a Toeplitz matrix: Comparison of Operation Counts for GR, GRC and MGRC, where Fast Givens Transforms are used and $k=2n$.

Discussion

For $m>>n$, a very common situation, MGRC requires one-third of the work of GRC and one-fifth of the work of GR when U and V are both required explicitly; when U and V are not required explicitly, MGRC is as order of magnitude faster than both GR and GRC because there is no mn^2 term.

For $m=n$, MGRC is slightly faster than GRC and slightly slower than GR when U and V are both required (however MGRC is faster than GR as well when $\frac{m}{n} > \frac{7}{6}$); this is the one case where MGRC is slightly worse than GR, (though better than GRC) and arises because accumulation of X followed by multiplication by $Q._{1:n}$ is not optimal here; it would be better to accumulate X <u>on</u> $Q._{1:n}$ (i.e. premultiply $Q^T._{1:n}$ by the Householder and Givens transforms yielded by steps 2B and 3). However, the asymptotic operation count would still only be equal to that of GR.

For $m=n$ and when U and V are not required explicitly, MGRC requires one-third of the work of GRC and one-half of the work of GRC.

In summary, therefore, if A is Toeplitz, MGRC is 2-5 times as fast as GR or GRC except in the following cases:

- $*m=n$, U,V both required (worst case) - MGRC, with a slight modification requires about the same work as GR and GRC.
- $*m>>n$, U,V not required (best case) - MGRC is an order of magnitude faster than both GR and GRC.

4. TOEPLITZ SVD BY GRAM-SCHMIDT BIDIAGONALIZATION

The second Toeplitz technique involves replacing phase I of GR by a Gram-Schmidt-type (GS) bidiagonalization technique, and using the Toeplitz form of A to accelerate the slowest part of the GS bidiagonalization. Phase II of GR is accelerated (as was done by Chan) by re-arranging the accumulation of U.

We first state and outline the proof of Algorithm 4.1, the GS bidiagonalization technique. The method was first proposed by Golub [35].

Algorithm 4.1 - GS Bidiagonalization

Input Matrix $A(m\times n;\ m\geq n)$

Output Matrices $W(m\times n)$, $B(n\times n)$ and $X(n\times n)$ such that

$$A = WBX^T \tag{4.1}$$

W has orthogonal columns, B is upper-bidiagonal, and X is orthogonal.

```
Procedure  1.* Set $\underline{x}_{.1} \leftarrow [1,0,\ldots,0]^T$
           2.  $\hat{\underline{w}}_{.1} \leftarrow A\underline{x}_{.1}$ ; $b_{11} \leftarrow \pm\|\hat{\underline{w}}_{.1}\|$ ; $\underline{w}_{.1}$  $\hat{\underline{w}}_{.1}/b_{11}$          (4.2a-c)
           3.  For $j \leftarrow 2$ to $n$ do
               3.1 $\hat{\underline{x}}^T_{.j} \leftarrow \underline{w}^T_{.j-1}A - b_{j-1,j-1}\,\hat{\underline{x}}^T_{.j-1}$;
                   $b_{j,j-1} \leftarrow \pm\|\hat{\underline{x}}^T_{.j}\|$; $\underline{x}^T_{.j} \leftarrow \hat{\underline{x}}^T_{.j}/b_{j,j-1}$          (4.3a-c)
               3.2 $\hat{\underline{w}}_{.j} \leftarrow A\underline{x}_{.j} - b_{j-1,j}\,\underline{w}_{.j-1}$; $b_{jj} \leftarrow \pm\|\hat{\underline{w}}_{.j}\|$; $\underline{w}_{.j} \leftarrow \hat{\underline{w}}_{.j}/b_{jj}$
                                                                                       (4.4a-c)
```

$*\underline{x}_{.j}$ denotes the jth column of X.

Proof (Outline) It is clear from (4.2b,c), (4.3b,c) and (4.4b,c) that

$$\|\underline{x}_{\cdot j}\| = \|\underline{w}_{\cdot j}\| = 1, \quad j = 1,\ldots,n. \tag{4.5}$$

In addition, (4.2)-(4.4) respectively yield

$$b_{11}\, \underline{w}_{\cdot 1} = A\underline{x}_{\cdot 1} \tag{4.6}$$

$$b_{j,j-1}\, \underline{x}^T_{\cdot j} = \underline{w}^T_{\cdot j-1} A - b_{j-1,j-1}\, \underline{x}^T_{\cdot j-1} \tag{4.7}$$

$$b_{jj}\, \underline{w}_{\cdot j} = A\underline{x}_{\cdot j} - b_{j-1,j}\, \underline{w}_{\cdot j-1} \tag{4.8}$$

And it is easy to show that (4.5)-(4.7) imply that

$$AX = WB \tag{4.9}$$

and

$$W^T A = BX^T \tag{4.10}$$

are satisfied by W,X and B.

It can be proved by induction that

$$\left.\begin{array}{ll} \underline{x}^T_{\cdot j}\, \underline{x}_{\cdot k} = 0, & (4.11) \\ \underline{w}^T_{\cdot j}\, \underline{w}_{\cdot k} = 0; & (4.12) \end{array}\right\} \begin{array}{l} k < j \\ j = 2,\ldots,n \end{array}$$

(4.11) and (4.12) are easily shown with j=2. For j>2, assume that (4.11) and (4.12) hold for j'<j, then calculate $\underline{x}^T_{\cdot j}\, \underline{x}_{\cdot k}$, using (4.7) to substitute for $\underline{x}^T_{\cdot j}$ and $\underline{x}_{\cdot k}$; and the resultant terms have factors of the type $\underline{w}^T_{\cdot p}\, \underline{w}_{\cdot q}$, $p<j$, $q<p \Rightarrow \underline{x}^T_{\cdot j}\, \underline{x}_{\cdot k}=0$. (4.12) may be similarly shown.

Thus (4.5),(4.11) and (4.12) show that W has orthogonal columns and X is orthogonal; this last fact combined with (4.9) yields (4.1).

QED

Acceleration of GS Bidiagonalization when A is Toeplitz

The main work of Algorithm 4.1 is the computation of $A\underline{x}_{\cdot 1}$, $\underline{w}^T_{\cdot j-1}A$ and $A\underline{x}_{\cdot j}$ in steps 4.1a, 4.2a and 4.3a respectively. Each of these calculations normally requires nm operations. However, when A is Toeplitz the complexity of these calculations can be reduced to $O(m \log m)$ by observing that the multiplication of an m×n Toeplitz matrix A by a vector $\underline{z}$ can be embedded in a circular convolution which can be done by FFT [20] (or by even faster recent methods [91]).

To see this, let $a_{i-j} := a_{ij}$,

$$\underline{a}^T = [a_{-(n-1)}, a_{-(n-2)}, \dots, a_{-1}, a_0, a_1, \dots, a_{m-1}, \overbrace{0, \dots, 0}^{n-1}] \tag{4.13}$$

and $$\underline{z}_f^T = [z_1, z_2, \dots z_n, \overbrace{0, \dots\dots\dots\dots 0}^{m+n-2}]. \tag{4.14}$$

Now if $\underline{c} = \underline{a} \circledast z_f$ the circular convolution of $\underline{a}$ and $\underline{z}_f$,

then $$c_i = \sum_{j=1}^{n} a_{j+i-1} \, z_{n-j+1}, \quad i=1,\dots,m$$

$$= (A\underline{z})_{m-i+1}, \quad i=1,\dots,n \tag{4.16}$$

Thus to calculate $A\underline{z}$ in $O(n \log n)$ operations, we simply select in reverse order elements 1 to m of $\underline{a} \circledast \underline{z}_f$, where $\underline{a}$ and $\underline{z}_f$ are as defined above. To calculate $\underline{a} \circledast \underline{z}_f$, we FFT $\underline{a}$ and $\underline{z}_f$, giving the complex spectra $\underline{f}(\underline{a})$ and $\underline{f}(\underline{z}_f)$ respectively, then inverse-FFT $\underline{f}(\underline{c}) := [f_i(\underline{c})]$, where $f_i(\underline{c}) = f_i(\underline{a}) f_i(\underline{z}_f)$.

Acceleration of Phase II of GR

This can be accelerated (as was done by Chan) by re-arranging the calculation of U. Recall that in GR, the $S^{(i)}$ were accumulated on P, requiring αkmn operations (see sec. 2) where $\alpha=2$ for slow Givens transforms. and $\alpha=1$ for fast Givens transforms. In the present algorithm, we accumulate the $S^{(i)}$ on the $n\times n$ identity matrix I_n, yielding an $n\times n$ matrix S, then compute

$$U = WS \tag{4.17}$$

where W is as produced by the (accelerated) GS bidiagonalization.

With this modification, the calculation of U requires $\alpha kn^2 + mn^2$ operations. Assuming that $k=2n$ and $\alpha=1$, this modification halves the work in calculating U when $m>>n$, and is faster than GR when $\frac{m}{n} > 2$. Thus if $\frac{m}{n} \leq 2$, GR is used.

The Algorithm

We now incorporate the accelerated GS Bidiagonalization and the accelerated GR Phase II in our second Toeplitz SVD:

Algorithm 4.2 - Toeplitz SVD by Accelerated Gram-Schmidt Bidiagonalization

```
{Phase I - Bidiagonalization :  $A=WBX^T$}
1.  Set $\underline{x}_{\cdot 1} \leftarrow \underbrace{[1,0,\ldots,0]}_{n} =: \underline{e}_1^T$
2.  Set $\underline{a} \leftarrow [a_{1n}, a_{1,n-1} \cdots a_{11}, a_{21} \cdots a_{m1}, \overbrace{0,\ldots,0}^{n-1}]$
    and $\underline{a}' \leftarrow [a_{m1}, \ldots, a_{11}, a_{12}, \ldots, a_{1n}, \overbrace{0,\ldots,0}^{m-1}]$
3.  $\hat{w}_{m-i+1,i} \leftarrow \{\underline{t} \circledast [\underline{x}^T_{\cdot 1}, 0,\ldots,0]^T\}_i, \quad i=1,\ldots,m$
4.  $b_{11} \leftarrow \|\hat{\underline{w}}_{\cdot 1}\|; \ \underline{w}_{\cdot 1} \leftarrow \hat{\underline{w}}_{\cdot 1}/b_{11}$
{Main Bidiagonalization Loop}
5.  For $j \leftarrow 2$ to $n$ do:
    5.1  $\underline{c} \leftarrow \underline{a}' \circledast [\underline{W}^T_{\cdot j-1}, \overbrace{0,\ldots,0}^{m+n-2}]^T; \ x_{ij} \leftarrow c_{n-i+1} - b_{j-1,j-1}x_{i,j-1}, \quad i=1,\ldots,n;$
         $b_{j,j-1} \leftarrow \|\underline{x}^T_{\cdot j}\|; \ \underline{x}^T_{\cdot j} \leftarrow \hat{\underline{x}}^T_{\cdot j}/b_{j,j-1}$
    5.2  $\underline{c} \leftarrow \underline{a} \circledast [\underline{x}^T_{\cdot j}, \overbrace{0,\ldots,0}^{m+n-2}]^T; \ w_{ij} \leftarrow c_{m-i+1} - b_{j-1,j}\, w_{i,j-1}, \quad i=1,\ldots,m;$
         $b_{jj} \leftarrow \|\hat{w}_{\cdot j}\|; \ \underline{w}_{\cdot j} \leftarrow \hat{\underline{w}}_{\cdot j}/b_{jj}$
```

```
{Phase II - Diagonalization}
6.  If $\frac{m}{n} \le 2$ then do
        6A.   Diagonalize B and calculate U and V using GR, Phase II
    else do
        6B.1  Diagonalize B as in GR, Phase II, accumulating the $\{S^{(i)}\}$ on $I_n$
              to produce $S$, and accumulating the $\{T^{(i)}\}$ on $X$ to produce $V$.
        6B.2  $U = WS$
```

Operation Counts

In Phase I, the main work is in performing the convolutions in steps 5.1 and 5.2. If this is done by FFT, these will require $(4m+2n-4)\log(2m+n-2)$ and $(4n+2m-4)\log(2n+m-2)$ operations respectively. (Note that $\underline{a}$ and $\underline{a}'$ only have to be FFT'd once). Hence Phase I requires $2n\{(2m+n-2)\log(2m+n-2)+(2n+m-2)\log(2m+n-2)\}$ operations.

In phase II, assuming fast Givens Transforms were used and $k=2n$, we can show that mn^2+4n^3 operations are required to calculate U and V and (as before) $O(n^2)$ operations are required to calculate Σ. We summarize the results in Table 4.1 (note that $(2m+n-2)$ and $(2n+m-2)$ have been replaced by $2m+n$ and $2n+m$ respectively, since the correction terms are an order of magnitude lower):

Calc. of U and V Algorithm	Explicit Calc.	No Explicit Calc.
GR	$5mn^2+2n^3$	$2(mn^2-\frac{n^3}{3})$
GRC	$3mn^2+5\frac{1}{3}\ n^3$	mn^2+n^3
AGSB	$2n\{(2m+n)\log(2m+n)$ $+(m+2n)\log(m+2n)\}$ $+mn^2+4n^3$	$2n\{(2m+n)\log(2m+n)$ $+(m+2n)\log(m+2n)\}$

Table 4.1 - SVD of a Toeplitz matrix: Comparison of Operation Counts for GR, GRC and AGSB where Fast Givens Transforms are used and $k=2n$.

Discussion

For $m>>n>>1$, it can be seen that AGSB is 3 times as fast as GRC and 5 times as fast as GR when U and V are required explicitly; when U and V are not required, AGSB is an order of magnitude faster than GR and GRC, because there is no mn^2 term. These comparisons are similar to those of MGRC (Table 3.1).

For $m=n>>1$, AGSB is slightly faster ($\frac{7}{5}$ to $\frac{5}{3}$ times) than both GR and GRC when U and V are both required - note that this is better than the result for MGRC, which was slightly slower than GR; when U and V are not required, AGSB is an order of magnitude faster than GR and GRC, because AGSB has complexity $O(n^2 log\ n)$ rather than $O(n^3)$. This is better than MGRC (Table 3.1), where the "speed-up" is a factor of 3.

If m and n are not both large, AGSB could be slower than any of GR, GRC and MGRC, because the $n(2m+n)log(2m+n)$ terms could be greater than the mn^2 and n^3 terms. Thus, AGSB is preferable when m and n are both large.

Possible numerical instability of AGSB

It has been shown by Golub [35] that Gram-Schmidt bidiagonalization is related to the Lanczos tridiagonalization of a symmetric matrix. The Lanczos procedure is known to be unstable [89] without re-orthogonalization, hence the same problem may arise in AGSB, though the excellent numerical properties of the FFT mean that errors will accumulate more slowly in this particular Gram-Schmidt procedure. It is therefore possible that AGSB is of only theoretical interest in many cases. Further work is needed to pursue this point.

Calculation of the Singular Values by the Iterative Lanczos Algorithm

Even though the Lanczos algorithm may give inaccurate results at step n, Paige and others [72], [86], have shown that continuing the Lanczos algorithm on an iterative basis, produces a set of tridiagonal matrices T_j (each the leading submatrix of its successor) whose sets of eigenvalues contain improving approximations to more and more of the eigenvalues of A. Parlett and Reid [74] propose a method of tracking the convergence of these eigenvalues for A symmetric. For a very difficult case (their POIS 992), the complete spectrum is obtained in about $5n$ iterations. More typically $2n$ iterations are required.

The main work in the Lanczos algorithm (as for GSB) is the calculation of the matrix-vector product $A\underline{v}_j$. As for GSB, this can be accelerated using fast convolution. Hence, the Parlett-Reid algorithm may determine the singular values (i.e. eigenvalues) of a symmetric Toeplitz matrix more stably. This observation is summarized in the next algorithm. We do not give details of Parlett and Reid's tracking procedure here, except to say that it estimates the eigenvalues of T_j by approximating the zeros and poles of a rational function $\delta(x)$ that depends on the entries of T_j. This approximation procedure requires $O(j)$ operations.

Algorithm 4.3 - Accelerated Lanczos Algorithm for Singular Values of a Symmetric Toeplitz Matrix

```
1.  Initialize Lanczos algorithm, and set $j=1$.
2.  Repeat
    2.1  Do step $j$ of the Lanczos algorithm [74] using fast convolution
         for matrix-vector product $A\underline{v}_j$. This generates a tridiagonal
         matrix $T_j$.
    2.2  Estimate the eigenvalues of $T_j$ by approximating the poles and
         zeros of $\delta(x)$.
    2.3  $j \leftarrow j+1$
    until Parlett and Reid's termination condition is satisfied.
```

Singular Values of a Non-symmetric Toeplitz Matrix

One way to proceed is to find the eigenvalues of $A^T A$ using the Parlett-Reid algorithm, using two fast convolutions for the matrix-vector products $A^T A\underset{-}{v}_j$. Alternatively, it can be shown, that AGSB, if continued on an iterative basis, calculates bidiagonal matrices B_j such that

$$B_j^T B_j = T_j ,$$

where the T_j are the tridiagonal matrices produced by running the Lanczos algorithm on $A^T A$. Hence we can track the square-roots of the eigenvalues of the T_j (which converge to the singular values of A) by approximating the zeros and poles of $\delta(x^2)$ by Parlett and Reid's method. This suggests the following algorithm:

Algorithm 4.4 - Iterative AGSB for the Singular Values of a Toeplitz Matrix

```
1.  Initialize AGSB and set $j=1$.
2.  Repeat
    2.1  Do step $j$ of AGSB, using fast convolution for the matrix-vector
         products. This generates a bidiagonal matrix $B_j$.
    2.2  Evaluate $T_j = B_j^T B_j$
    2.3  Estimate the square-roots of the eigenvalues of $T_j$ by approximating
         the poles and zeros of $\delta(x^2)$
    2.4  $j \leftarrow j+1$
    until Parlett and Reid's termination condition is satisfied.
```

Remark These two algorithms have not yet been tested, and further work is required in this regard. There is reason for optimism, however, since there seems to be no reason to suppose in general that the Lanczos algorithm converges less quickly for non-sparse matrices than for sparse matrices.

5. CONCLUSION

Two algorithms have been presented which take advantage of the structure of a Toeplitz matrix in calculating its SVD. Both can be an order of magnitude faster than general SVD routines, depending on the values of m and n, and whether U and V are required explicitly. In most cases of interest, one or other of the routines is several times as fast as general SVD routines. The one exception is when n and m are "small" (<30 say) and U and V are both required - in this case there does not seem to be much advantage in using either of the Toeplitz SVD routines. It is pointed out that the second algorithm may be unstable in some cases of interest, and more work is required to investigate this problem. However, by modifying it to make it iterative and using a recently-developed tracking procedure, it may be possible to calculate at least the singular values stably.

CHAPTER 10

CONCLUSION

The solution of Toeplitz linear systems, and the related operations of Toeplitz inversion and factorization, have many applications in engineering and applied mathematics. We have derived several connexions between known $O(n^2)$ algorithms in the field, relating then to Bareiss's Toeplitz elimination algorithm and to rank-1 update procedures. We have presented new results on the numerical performance of some of these algorithms, and propose a pivoting scheme which should improve the performance of Toeplitz solvers, factorizers and inverters in all indefinite cases, but especially when some leading submatrices of the system matrix are ill-conditioned.

More recently, it has been found useful to perform other operations on Toeplitz matrices such as the QR decomposition and the singular value decomposition (SVD). We have proposed several algorithms to compute the QR decomposition in $O(n^2)$ operations. The fastest method of calculating R by orthogonal transforms requires slightly more than twice the number of operations required to find the triangular factors of T. We have extended the techniques to rectangular and block-Toeplitz matrices. We have proposed methods of accelerating the SVD when the system matrix is Toeplitz, including an $O(n^2 \log n)$ algorithm which may, however, be unstable, and have suggested a modification which may calculate the singular values stably in $O(n^2 \log n)$ operations, but further work is required to test the proposal. It may also be possible to reduce the SVD complexity to $O(n^2)$.

Further work can be done on extending these results to matrices which are related to Toeplitz, such as α-Toeplitz matrices and multi-level matrices. In another direction, it may be possible to develop $O(n\ log^2\ n)$ methods such as in [13] for Toeplitz QR and SVD problems. As was remarked in the Introduction, the $O(n^2)$ methods will still be useful for small to moderate Toeplitz problems.

REFERENCES

[1] B.P. Agrawal, "An algorithm for designing constrained least-squares fitters", IEEE Trans. Acoust., Speech, Sig. Processing, vol. ASSP-25, no. 5, pp.410-414, Oct. 1977.

[2] B.P. Agrawal and V.K. Jain, "Digital restoration of images in the presence of correlated noise", Comput. and Elec. Eng., vol. 3, pp.65-74, 1976.

[3] A.V. Aho, J.E. Hopcroft and J.D. Ullman, "The design and analysis of computer algorithms", Reading, MA: Addison-Wesley, 1974.

[4] H. Akaike, "Block Toeplitz matrix inversion", SIAM J. Appl. Math., vol. 24, no.2, March 1975.

[5] M. Aoki, "Optimization of stochastic systems", New York: Academic Press, 1967.

[6] E.H. Bareiss, "Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices", Numer. Math., vol. 13, pp.404-424, 1969.

[7] J.M. Bennett, "Triangular factors of modified matrices", Numer. Math., vol. 7, pp.217-221, 1965.

[8] E.R. Berlekamp, "Algebraic coding theory", New York: McGraw-Hill, chap.7, 1968.

[9] R.R. Bitmead and B.D.O. Anderson, "Asymptotically fast solution of Toeplitz and related systems of equations", Lin. Alg. and Appl., vol. 34, pp.103-116, 1980.

[10] N.K. Bose and S. Basu, "Theory and recursive computation of 1-D matrix Padé approximants", IEEE Trans. Circ. Sys. vol. CAS-27, no.4, pp.323-325, April 1980.

[11] N.K. Bose and S. Basu, "Two-dimensional matrix Padé approximants - existence, nonuniqueness and recursive computation", IEEE Trans. Auto. Control, vol. AC-25, no.3, pp.509-514, June 1980.

[12] R.P. Brent, Private Communication, 1979.

[13] R.P. Brent, F.G. Gustavson and D.Y.Y. Yun, "Fast solution of Toeplitz systems of equations and computation of Padé approximants", J. Algorithms, vol. 1, pp.259-295, 1980.

[14] A. Bultheel, "Recursive algorithms for the Padé table: two approaches". In "Padé approximation and its applications", ed. L. Wuytack, Berlin: Springer-Verlag, pp.211-230, 1979.

[15] A. Bultheel, "Recursive algorithms for the matrix Padé problem", Math. Comp. vol. 35, no.151, pp.875-892, July 1980.

[16] G. Carayannis, "An alternative formulation for the recursive solution of the covariance and autocorrelation equation", IEEE Trans. Acoust., Speech, Sig. Processing, vol. ASSP-25, no.6, pp.574-577, Dec. 1977.

[17] T.F.C. Chan, "On computing the singular-value decomposition", Tech. Rept. STAN-CS-77-588, Computer Science Dept., Stanford University, Feb. 1977.

[18] G. Claessens and L. Waytack, "On the computation of non-normal Padé approximants", J.Comp.Appl.Maths., vol. 5, no.4, pp.283-289, 1979.

[19] S.D. Conte and C. de Boor, "Elementary numerical analysis - an algorithmic approach", Chapter 4, New York: McGraw-Hill, 1980.

[20] J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier series", Math. Comp., vol. 19, pp.297-301, Apr. 1965.

[21] J.J. Cornyn, Jr., "Direct methods for solving systems of linear equations involving Toeplitz or Hankel matrices", Masters thesis, NRL Memorandum Report 2920, Oct. 1974.

[22] G. Cybenko, "Round-off error propagation in Durbin's, Levinson's and Trench's algorithms", Proc. Int. Conf. on Acoust., Speech and Sig. Processing, Washington DC, April 1979.

[23] G. Cybenko, "Error analysis of Durbin's algorithm", Tech. Rept., Dept. of Mathematics, Tufts University, 1980.

[24] L.S. De Jong, "Numerical aspects of recursive realization algorithms", SIAM J.Contr.Opt., vol. 16, pp.646-659, 1978.

[25] B.W. Dickinson, T. Kailath and M. Morf, "Canonical matrix fraction and state-space description for deterministic and stochastic linear systems", IEEE Trans.Auto. Control, vol. AC-19, pp.656-667, Dec. 1974.

[26] J. Durbin, "The fitting of time-series models", Rev. Int. Stat. Inst., vol. 28, pp.229-249, 1959.

[27] R. Fletcher and M.J.D. Powell, "On the modification of LDL^T factorizations", Math. Comp. vol. 28, pp.1067-1087, Oct. 1974.

[28] B. Friedlander, T. Kailath, M. Morf and L. Ljung, "Extended and Chandrasekhar equations for general discrete-time estimation problems", IEEE Trans.Auto.Control, vol. 23, no.4, pp.653-659, 1978.

[29] B. Friedlander, M. Morf, T. Kailath and L. Ljung, "New inversion formulas for matrices classified in terms of their distance from Toeplitz matrices", Linear Alg. Applics., vol. 27, pp.31-60, 1979.

[30] W.M. Gentleman, "Least-squares computation by Givens' transformations without square roots", J. Inst. Maths. Applics., vol. 12, pp.329-336, 1973.

[31] J.D. Gibson, "On reflection co-efficients and the Cholensky decomposition", IEEE Trans. Acoust., Speech, Sig. Processing, vol. ASSP-25, no.1, pp.93-96, Feb. 1977.

[32] P.E. Gill, G.H. Golub, W. Murray and M.A. Saunders, "Methods for modifying matrix factorizations", Math. Comp., vol. 28, no.126, pp.505-535, 1974.

[33] I.C. Gohberg and A.A. Semencul, "On the inversion of finite Toeplitz matrices and their continuous analogues", Mat. Issled. (Russian) no. 2, pp.201-233, 1972.

[34] G.H. Golub, "Linear least-squares solutions by Householder transforms", Numer. Math., vol. 7, H.B. Series Linear Algebra, pp.269-276, 1965.

[35] G.H. Golub, "Least squares, singular values and matrix approximations", Aplikace Matematiky, vol. 13, pp.41-51, 1965.

[36] G.H. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix", J. SIAM Numer. Anal., Ser.B, Vol. 2, no.2, pp.205-224, 1965.

[37] G.H. Golub and C. Reinsch, "Singular-value decomposition and least-squares solutions", Numer. Math., vol. 14, pp.403-420, 1970.

[38] W.B. Gragg, "The Padé table and its relation to certain algorithms of numerical analysis", SIAM Rev., vol. 14, pp.1-62, 1972.

[39] P.R. Graves-Morris, "The numerical calculation of Padé approximants", in "Padé approximation and its applications", ed. L. Waytack, Berlin: Springer-Verlag, pp.231-245, 1979.

[40] R.M. Gray, "Toeplitz and circulant matrices: a review", Tech. Rept. 6502-1, Inf. Syst. Laboratory, Center for Syst. Research, Stanford University, June 1971.

[41] R.M. Gray, "Toeplitz and circulant matrices: II", Tech. Rept. 6504-1, Information Systems Lab., Stanford University, April 1977.

[42] U. Grenander and G. Szegö, "Toeplitz forms and their applications", Berkeley, CA: University of California Press, 1958.

[43] S. Hammarling, "A note on modifications to the Givens' plane rotation", J. Inst. Maths. Applics., vol. 13, pp.215-218, 1974.

[44] R.E. Hartwig, "Schur's theorem and the Drazin inverse", Pac. J. Math., vol. 78, no.1, pp.133-138, 1978.

[45] R.E. Hartwig and M.E. Fisher, "Asymptotic behaviour of Toeplitz matrices and determinants", Archive for rational Mech. and Analysis, vol. 32, p.190, 1969.

[46] F.B. Hildebrand, "Introduction to numerical analysis", New York: McGraw-Hill, pp.378-382, 1956.

[47] N.M. Huang and R.E. Cline, "Inversion of persymmetric matrices having Toeplitz inverses", J. ACM, vol. 19, p.437, 1972.

[48] J.R. Jain, "An efficient algorithm for a large Toeplitz set of linear equations", IEEE Trans. Acoust., Speech, Sig. Processing, vol. 27, no.2, pp.612-615, 1979.

[49] J.H. Justice, "An algorithm for inverting positive-definite Toeplitz matrices", SIAM J. Appl. Math. vol. 23, no.3, pp.289-291, Nov. 1972.

[50] J.H. Justice, "The Szegö recursion relation and inverses of positive-definite Toeplitz matrices", SIAM J. Math. Anal., vol. 5, no. 3, pp.503-508, May 1974.

[51] J.H. Justice, "A Levinson-type algorithm for two-dimensional Wiener filtering using bivariate Szegö polynomials", Proc. IEEE, vol. 65, pp.882-886, 1977.

[52] T. Kailath, "A view of three decades of linear filtering theory", IEEE Trans. Inf. Theory, vol. IT-20, pp.146-181, Mar. 1974.

[53] T. Kailath, "Some alternatives in recursive estimation", Int. J. Control, vol. 32, no.2, pp.311-328, 1980.

[54] T. Kailath, B. Levy, L. Ljung and M. Morf, "The factorization and representation of operators in the algebra generated by Toeplitz operators", SIAM J. Appl. Math., vol. 37, no. 3, pp.467-484, 1979.

[55] T. Kailath, A Vieira and M. Morf, "Inverses of Toeplitz operators, innovations and orthogonal polynomials", SIAM Rev. vol. 20, pp.106-119, 1978.

[56] S. Kung and T. Kailath, "Fast projection methods for minimal design problems in linear systems theory", Automatica, vol. 16, pp.399-403, 1980.

[57] S. Kung and D.W. Lin, "Optimal Handel-norm reductions: Multivariable systems", IEEE Trans. Auto. Control, vol. AC-26, no.4, pp.832-852, 1981.

[58] D.T. Lee, M. Morf and B. Friedlander, "Recursive least-squares ladder algorithms", IEEE Trans. Acoust., Speech, Sig. Processing, vol. ASSP-29, no.3, June 1981.

[59] N. Levinson, "The Wiener rms (root-mean-square) error criterion in filter design and prediction", J. Math. Phys., vol. 25, pp.261-278, January 1947.

[60] J. Makhoul, "Linear prediction: A tutorial review", Proc. IEEE, vol. 63, pp.561-580, Apr. 1975.

[61] J. Makhoul, "Stable and efficient lattice methods for linear prediction", IEEE Trans. Acoust, Speech, Sig. Processing, vol. ASSP-25, pp.423-428, Oct. 1977.

[62] M.A. Malcolm and J. Palmer, "A fast method for solving a class of tridiagonal linear systems", Comm. ACM, vol. 17, pp.14-17, 1974.

[63] J.D. Markel and A.H. Gray, "On autocorrelation equations, as applied to speech analysis", IEEE Trans. Audio Electroacoust., vol. AU-21, pp.69-79, Apr. 1973.

[64] J.L. Massey, "Shift-register synthesis and BCH decoding", IEEE Trans. Inf. Theory, vol. IT-15, pp.122-127, Jan. 1969.

[65] M. Morf, "Fast algorithms for multivariable systems", Ph.D. thesis, Dept. of Elec. Eng., Stanford University, Stanford, CA 1974.

[66] M. Morf, "Doubling algorithms for Toeplitz and related equations", Lecture at Information Systems Laboratory, Stanford University, Stanford, CA, April 17, 1979.

[67] M. Morf, B. Dickinson, T. Kailath and A. Vieira, "Efficient solution of covariance equations for linear prediction", IEEE Trans. Acoust., Speech, Sig. Processing, vol. ASSP-25, pp.429-433, Oct. 1977.

[68] M. Morf and T. Kailath, "Square-root algorithms for linear least-squares estimation", IEEE Trans. Auto. Control, vol. 20, no.4, pp.487-497, 1975.

[69] M. Morf and D. Lee, "Recursive least-squares ladder forms for fast parameter tracking", Proc. IEEE Conf. on Decision and Control, (San Diego, CA) pp.1362-1367, Jan. 1979.

[70] M. Morf, A. Vieira and D.T. Lee, "Ladder forms for identification and speech processing", in Proc. 1977 Conf. Decision and Control (New Orleans LA), pp.1074-1078, Dec. 1977.

[71] C.C. Paige, "An error analysis of a method for solving matrix equations", Math. Comp., vol. 27, no.122, pp.355-359, 1973.

[72] C.C. Paige, "Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix", J. Inst. Maths. Applics., vol. 18, pp.341-349, 1976.

[73] B.S. Pariiski, "An economical method for the numerical solution of convolution equations", USSR Comp. Math. Phys. pp.208-211, 1978.

[74] B.N. Parlett and J.K. Reid, "Tracking the progress of the Lanczos algorithm for large symmetric eigenproblems", IMA J. Numer. Anal., vol. 1, pp.135-155, 1981.

[75] W.C. Pye, T.L. Boullion and T.A. Atchison, "The pseudoinverse of a composite matrix of circulants", SIAM J. Appl. Math., vol. 24, no. 4, pp.552-555, 1973.

[76] W.D. Ray, "The inverse of a finite Toeplitz matrix", Technometrics, vol. 12, pp.153-156, 1970.

[77] F.L. Reed, L.W. Brooks and W. Norr, "Adaptive beanforming and filtering", Notes form a course given by Technology Service Corp., pp.107-108, May 15-17, 1973.

[78] J. Rissanen, "Recursive identification of linear systems", SIAM J. Contr. Opt., vol. 9, pp.420-430, 1971.

[79] J. Rissanen, "Algorithms for the triangular decomposition of block Hankel and Toeplitz matrices with application to factoring positive matrix polynomials", Math. Comp., vol. 27, pp.147-154, 1973.

[80] J. Rissanen, "Solution of linear equations with Hankel and Toeplitz matrices", Numer. Math., vol. 22, pp.361-366, 1974.

[81] P.A. Roebuck and S. Barnett, "A survey of Toeplitz and related matrices", Int. J. Systems Sci., vol. 9, no.8, pp.921-934, 1978.

[82] D.H. Sinnott, "Matrix analysis of linear antenna arrays of equally-spaced elements", IEEE Trans. on antennas and prop., vol. AP-21, pp.385-386, 1973.

[83] L.N. Trefethen, "Rational Chebyshev approximation on the unit disk", Numer. Math., vol. 37, pp.297-320, 1981.

[84] W.F. Trench, "An algorithm for the inversion of finite Toeplitz matrices", SIAM J. Appl. Math., vol. 12, pp.515-522, 1964.

[85] T.J. Ulrych, D.E. Smylic, O.G. Jensen and G.K.C. Clarke, "Predictive filtering and smoothing of short records by using maximum entropy", J. Geophys. Res., vol. 78, pp.4959-4964, Aug. 10, 1973.

[86] J.M. van Kats and H.A. van der Vorst, "Automatic monitoring of Lanczos-schemes for symmetric and skew-symmetric generalized eigenvalue problems", Report TR-7, Academic Computer Centre, Utrecht, 1977.

[87] V.V. Voevodin, E.E. Tirtishnikov, O.B. Arushnian, M.K. Samarin, J.M. Boyle, W.R. Cowell, K.W. Dritz and B.S. Garbow, "The Toeplitz package users' guide - preliminary version", Argonne National Laboratory, May 1979.

[88] R.A. Wiggins and E.A. Robinson, "Recursive solution to the multichannel filtering problem", J. Geophys. Res., vol. 70, no.8, pp.1885-1891, 1965.

[89] J.H. Wilkinson, The algebraic eigenvalue problem", Oxford: Clarendon Press, 1965.

[90] J.H. Wilkinson, "Some recent advances in numerical linear algebra", in "The State of the art in numerical analysis", 1977.

[91] S. Winograd, "On computing the discrete Fourier transform", Math. Comp. vol. 32, pp175-199, Jan. 1978.

[92] S. Zohar, "Toeplitz matrix inversion: the algorithm of W.F. Trench", J. ACM, Vol. 16, pp.592-601, 1969.

[93] S. Zohar, "The solution of a Toeplitz set of linear equations", J. ACM. vol. 21, pp.272-276, 1974.

* Supplementary references:
See over

Supplementary References

[94] R. Cline, R. Plemmons and G. Worm, "Generalized inverses of Toeplitz matrices", Lin. Alg. and Appl., vol. 8, pp.25-33, 1974.

[95] P. Delsarte, Y. Genin and Y. Kamp, "Schur parameterization of positive-definite block-Toeplitz systems", SIAM J. Appl. Maths., vol. 36, pp.34-45, 1979.

[96] P. Delsarte, Y. Genin and Y. Kamp, "A polynomial approach to the Levinson algorithm", IEEE. Trans. Inf. Theory, to appear.

[97] P. Delsarte, Y. Genin and Y. Kamp, "On the Toeplitz embedding of an arbitrary matrix", Lin. Alg. and Appl., to appear.