TR-2002014: Iterative Inversion of Structured Matrices

Gianni Codevico

Victor Y. Pan

Marc Van Barel

Xinmao Wang

Iterative Inversion of Structured Matrices [*]

Gianni Codevico[†], Victor Y. Pan[‡], Marc Van Barel[§] and Xinmao Wang[¶]

November 15, 2002

Abstract: Iterative processes for the inversion of structured matrices can be further improved by using a technique for compression and refinement via the least-squares computation. We review such processes and elaborate upon incorporation of this technique into the known frameworks.

Key Words: structured matrices, displacement rank, iterative inversion, least-squares computations

1 Introduction

Structured matrices such as Toeplitz, Hankel, Vandermonde, and Cauchy matrices as well as matrices with a structure generalizing the latter classes are omnipresent in computations for sciences, engineering, and signal processing. Displacement representations of such a matrix enable its fast multiplication by a vector and expression of its inverse via the

[*]The research of the second and fourth author was supported by NSF Grant CCR 9732206 and PSC CUNY Award 66406-0033. The research of the first and third author was supported by the Research Council K.U.Leuven, project OT/00/16 (SLAP: Structured Linear Algebra Package), by the Fund for Scientific Research–Flanders (Belgium), projects G.0078.01 (SMA: Structured Matrices and their Applications), and G.0176.02 (ANCILA: Asymptotic aNalysis of the Convergence behavior of Iterative methods in numerical Linear Algebra), and by the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture, project IUAP V-22 (Dynamical Systems and Control: Computation, Identification & Modelling). The scientific responsibility rests with the authors.

[†]Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven (Heverlee), Belgium (Gianni.Codevico@cs.kuleuven.ac.be),

[‡]Department of Mathematics and Computer Science, Lehman College, City University of New York, Bronx, NY 10468 (vpan@lehman.cuny.edu)

[§]Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven (Heverlee), Belgium (Marc.VanBarel@cs.kuleuven.ac.be)

[¶]Ph.D. Program in Mathematics, Graduate School of CUNY, New York, NY 10016, USA (xwang2@gc.cuny.edu)

solutions of a few linear systems of equations. The latter problems (of inversion and linear system solving) are highly important for the theory and practice of computing.

Some effective direct solution algorithms exploiting the displacement representation can be found , e.g., in [19, 20, 5, 7, 14, 32, 21, 26]. Alternative iterative methods were proposed, e.g., in [22, 23, 25, 24, 33, 27, 3, 28]. The latter methods nontrivially extend some preceding work for general input matrices [2, 34, 30] and can be most effective for well-conditioned inputs.

We briefly survey the state of the art in Sections 2–5. In particular, in Section 5 we cover two policies of keeping matrices compressed during the iterative process. In Section 6, we cover another technique based on the least-squares computation, which in addition enables both compression and refinement of the computed approximations to the inverse (see Theorem 6.1). We elaborate upon incorporation of this technique into the known frameworks for iterative inversion. Section 7 demonstrates the validity of the method by numerical experiments.

Due to the well-known close correlation between computations with structured matrices and with polynomials and rational functions [26], many fundamental algebraic computations such as polynomial multiplication and division, and polynomial and rational interpolation and multipoint evaluation can be reduced to operations with structured matrices. So our work may serve as an example of effective application af numerical methods to solve fundamental problems of algebraic computation.

2 Iterative matrix inversion

Newton's iteration for matrix inversion

$$X_{i+1} = X_i(2I + MX_i),\ i = 0, 1, \dots \tag{2.1}$$

defines a sequence of approximations $X_0, X_1, \dots$ to $-M^{-1}$ with the residuals $I + MX_i$ and $I + X_iM$ squared in each step (2.1). Thus the matrices X_i rapidly converge to $-M^{-1}$ if initially the norms $\|I + MX_i\|$ and/or $\|I + X_iM\|$ are substantially less than 1. In some cases an initial approximation X_0 can be supplied from outside; otherwise it can be generated according to some rules specified in [2, 30, 28, 34] and [26, Chapter 6]. (See also the homotopy/continuation approach in [26, Chapter 6] and [28].) There are certain policies allowing convergence acceleration, such as

$$X_{i+1} = a_iX_i(2I + MX_i),\ i = 0, 1, \dots , \tag{2.2}$$

decreasing the number of steps by a factor of 2 versus (2.1) for scalars a_i specified in [30] (cf. [26, page 191]), and

$$X_{i+1} = X_i(I + R_i + R_i^2 + \dots + R_i^{p-1}), R_i = I + MX_i, i = 0, 1, \dots . \tag{2.3}$$

Note that $R_{i+1} = R_i^p$ under (2.3), multiplication of p matrix pairs is needed per step (2.3), and processes (2.2) and (2.3) turn into (2.1) where $a_i = 1$ for all i and $p = 2$, respectively.

3 Structured matrices

Iterative inversion is most effective for structured matrices, for which matrix-matrix and matrix-vector multiplication can be performed at a low computational cost (using $O(n \log n)$ or $O(n \log^2 n)$ flops, versus the order of n^3 for $n \times n$ general matrices).

Table 3.1 displays the most popular classes of structured matrices. Each of these $n \times n$ matrices is completely defined by $n, 2n-1$, or $2n$ parameters. More generally, many other matrices with similar structures can be represented with $O(n)$ parameters as follows. Associate with a fixed class of structured matrices M a pair of *operator matrices* A and B such that the *Sylvester* and/or *Stein displacements* of M,

$$\triangle_{A,B}(M) = M - AMB = GH^T, \nabla_{A,B}(M) = AM - MB = GH^T, \tag{3.1}$$

respectively, have small rank r (called the *displacement rank* of M). The $n \times n$ matrix M can be effectively expressed via the columns of its *displacement generator* matrices G and H of size $n \times r$. Then operate with structured matrices represented in this compressed form. We trace this important approach back to [18, 16, 15, 13] (cf. also [10, 19]); it has a huge bibliography (cf. [14, 26, 17, 4] for surveys and details). We cover the Sylvester displacement representation referring the reader to [26] and the references therein on the dual Stein displacement representation.

Typical operator matrices are the unit f-circulant matrices,

$$Z_f = (z_{i,j})_{i,j=0}^{n-1} \tag{3.2}$$

where $z_{i+1,i} = 1, i = 0, 1, \ldots, n-2, z_{0,n-1} = f, z_{i,j} = 0$ if $(i-j) \mod n \neq 1$, and diagonal matrices, $D_{\boldsymbol{s}} = \text{diag}(s_i)_{i=0}^{n-1}$ for $\boldsymbol{s} = (s_i)_{i=0}^{n-1}$. Table 3.2 shows operator matrices associated with structured matrices of Table 3.1, the displacement rank, and the cost in flops for multiplication by a vector. For these operator matrices, the arithmetic cost of multiplication of a structured matrix by a vector lies in $O(rn \log n)$ with the operators ∇_{Z_e, Z_f} and ∇_{Z_e, Z_f^T} (that is, in the *Toeplitz/Hankel, T/H, case*) and in $O(rn \log^2 n)$ with the operators $\nabla_{D(t), Z_f^T}$ and $\nabla_{D(s), D(t)}$ (that is, in the *Vandermonde/Cauchy, V/C, case*).

4 Structured iterative inversion via matrix-by-vector multiplication

The acceleration of iterative inversion of structured matrices is achieved by reducing processes (2.1)–(2.3) to matrix-by-vector multiplication. Similarly to (3.1), write

$$\nabla_{B,A}(X_k) = G_k H_k^T \tag{4.1}$$

where G_k, H_k are $n \times l$ matrices, $r \leq l = l(k) \leq n$, and observe that iteration (2.2) can be performed by computing the generators

$$G_{i+1} = (a_{i+1}(2I + X_i M)G_i, a_{i+1}G_i, a_{i+1}X_i G_i), \tag{4.2}$$

Table 3.1: **Four classes of structured matrices**

Toeplitz matrices $(t_{i-j})_{i,j=0}^{n-1}$	Hankel matrices $(h_{i+j})_{i,j=0}^{n-1}$
$\begin{pmatrix} t_0 & t_{-1} & \cdots & t_{1-n} \\ t_1 & t_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{pmatrix}$	$\begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_1 & h_2 & \iddots & h_n \\ \vdots & \iddots & \iddots & \vdots \\ h_{n-1} & h_n & \cdots & h_{2n-2} \end{pmatrix}$
Vandermonde matrices $\left(t_i^j\right)_{i,j=0}^{n-1}$	**Cauchy matrices $\left(\frac{1}{s_i-t_j}\right)_{i,j=0}^{n-1}$**
$\begin{pmatrix} 1 & t_0 & \cdots & t_0^{n-1} \\ 1 & t_1 & \cdots & t_1^{n-1} \\ \vdots & & & \vdots \\ 1 & t_{n-1} & \cdots & t_{n-1}^{n-1} \end{pmatrix}$	$\begin{pmatrix} \frac{1}{s_0-t_0} & \cdots & \frac{1}{s_0-t_{n-1}} \\ \frac{1}{s_1-t_0} & \cdots & \frac{1}{s_1-t_{n-1}} \\ \vdots & & \vdots \\ \frac{1}{s_{n-1}-t_0} & \cdots & \frac{1}{s_{n-1}-t_{n-1}} \end{pmatrix}$

Table 3.2: **Structured matrix operator matrices**

Matrix Class	Pair of Operator Matrices	Displacement Rank	# of Flops for Multiplication by Vector
Toeplitz $(t_{i-j})_{i,j}$	$Z_e, Z_f, e \neq f$	≤ 2	$O(n \log n)$
Hankel $(h_{i+j})_{i,j}$	$Z_e, Z_f^T, ef \neq 1$	≤ 2	$O(n \log n)$
Vandermonde $(t_i^j)_{i,j}$	$D_{\boldsymbol{t}}, Z_f^T$	≤ 1	$O(n \log^2 n)$
Cauchy $(\frac{1}{s_i-t_j})_{i,j}$	$D_{\boldsymbol{s}}, D_{\boldsymbol{t}}$	≤ 1	$O(n \log^2 n)$

$$H_{i+1}^T = \begin{bmatrix} H_i^T \\ H_i^T M X_i \\ H_i^T X_i \end{bmatrix}, \tag{4.3}$$

for X_{i+1}. For $a_{i+1} = 1$, this turns into a compressed version of iteration (2.1), and similar expressions can be derived for process (2.3). Eq. (4.2) and (4.3) reduce step (2.2) (or (2.1)) to multiplication of the matrices M, M^T, X_i, and X_i^T by $l, l, 2l$, and $2l$ vectors, respectively, where $l = l(i)$ is the length of the displacement generator for X_i. This means

$$c_{l,n,r} = O((l + r)^2 n \log^d n) \tag{4.4}$$

flops per step (2.2) where $d = 1$ (in the T/H case) or $d = 2$ (in the V/C case), so it is crucial to bound $l = l(i)$ to make the iteration effective. Typical initial choices of X_0 achieve $l_0 \leq r$, but the process (4.2), (4.3) may inflate this to $l_i = 3^i l_0$, so special care should be taken periodically to keep computations effective.

5 Compression of the iterates via the truncation of singular values or via substitution

To compress the iterates X_{i+1} one should modify (2.1)–(2.3) as follows:

$$\hat{X}_{i+1} = g(X_i, M), \qquad X_{i+1} = f(\hat{X}_{i+1}) \tag{5.1}$$

where $g(X_i, M)$ is the iteration defined by (2.1), (2.2), or (2.3), and the function $f(W)$ defines a compression rule. Unfortunately, already with the first compression step, the theorems in [30] supporting acceleration by twice in (2.2) vs. (2.1) hold no more, so one may either ensure a desired decrease of the residual norm in fewer steps (2.2) by postponing compression (5.1) and thus involving more flops per step or use compression and then risk either divergence or a slow-down of convergence.

The first simple policy of defining $f(\hat{X}_{i+1})$ in (5.1) (proposed in [23], [25, 24] and described as Subroutine R1 in [26, Section 6.4] is to *truncate the smallest singular values of the displacement*

$$\nabla_{A,B}(X_{i+1}) = G_{i+1} H_{i+1}^T. \tag{5.2}$$

Another policy [33, 27] (called *compression by substitution* and described as Subroutine R2 in [26, Section 6.5] is to replace (2.1)–(2.3) by the process

$$G_{i+1} = g(X_i, M)G, \;\; H_{i+1}^T = H^T g(X_i, M), \tag{5.3}$$

requiring about $c_{r,n,r}$ flops per step, that is, about as many flops as in process (4.2), (4.3) for $l = r$. The reader is referred to [23, 25, 27, 29] on the estimates for how much (or how little) the above compression policies slow down the convergence.

6 Compression using a least-squares criterion

The third policy is to compute a least-squares refinement G_{i+1}, H_{i+1} of the displacement generator $\hat{G}_{i+1}, \hat{H}_{i+1}$ of the computed approximation $\hat{X}_{i+1}$ to M^{-1} such that

$$\nabla_{B,A}(\hat{X}_{i+1}) = \hat{G}_{i+1}, \hat{H}_{i+1}^T, \tag{6.1}$$

$$G_{i+1} = \hat{G}_{i+1}Y_{i+1,G},\ H_{i+1} = \hat{H}_{i+1}Y_{i+1,H}, \tag{6.2}$$

and the norms

$$N_G = \|G - M\hat{G}_{i+1}Y_{i+1,G}\|_2 \tag{6.3}$$

and

$$N_H = \|H - M^T\hat{H}_{i+1}Y_{i+1,H}\|_2 \tag{6.4}$$

are minimum over all $l \times r$ matrices $Y_{i+1,G}$ and $Y_{i+1,H}$. The pair G_{i+1}, H_{i+1} is used as a displacement generator representing the matrix

$$X_{i+1} = \nabla_{B,A}^{-1}(G_{i+1}H_{i+1}^T)$$

of (5.2). Besides compression, this policy is also intended to refine the approximation to M^{-1} (see Theorem 6.1).

We consider two applications of this policy of *compression and refinement via least-squares approximation*:

(1) as an alternative for the policy of truncating the smallest singular values, in which case the matrices $\hat{G}_{i+1}^T, \hat{H}_{i+1}^T$ are defined as G_{i+1}^T, H_{i+1}^T in (4.2), (4.3), and

(2) as a complement to the compression by substitution, in which case the matrices $\hat{G}_{i+1}, \hat{H}_{i+1}^T$ are defined as the matrices G_{i+1}, H_{i+1}^T in (5.3).

In case (1), the $\hat{G}_{i+1}, \hat{H}_{i+1}$ are $n \times l$ matrices, for $l \leq 3r$, and are typically rank deficient, so the least-squares computation of $Y_{i+1,G}$ and $Y_{i+1,H}$ should rely on computing the SVD's of $M\hat{G}_{i+1}$ and $M^T\hat{H}_{i+1}$ at the cost of performing $2(2n+11l)l^2$ flops (see [11, pages 257,263]) or maybe on rank revealing QR factorization replacing the SVD's. This clearly dominates the $(4l-2)nr$ flops required for computing the generator G_{i+1}, H_{i+1} in (6.2) but typically (for small r) is dominated by $c_{l,n,r}$ flops in (4.4) involved in the computations in (4.2), (4.3).

In case (2), $\hat{G}_{i+1}$ and $\hat{H}_{i+1}$ are $n \times r$ matrices and typically have full rank; in this case, instead of computing $Y_{i+1,G}$ and $Y_{i+1,H}$ based on the SVD's, one may compute them simply from the normal equations

$$(\hat{G}_{i+1}^T M^T)M\hat{G}_{i+1}Y_{i+1,G} = \hat{G}_{i+1}^T M^T G, \tag{6.5}$$

$$(\hat{H}_{i+1}^T M)M^T\hat{H}_{i+1}Y_{i+1,H} = \hat{H}_{i+1}^T MH. \tag{6.6}$$

This amounts to multiplication of each of the matrices M and M^T by r vectors (that is a fraction of $c_{l,n,r}$ flops of (4.4)), $4r^2(2n-1)$ flops for computing the coefficients of normal equations (6.5), (6.6), and $4r^4/3+3r^3$ flops for solving these equations. Clearly, computations in (6.1)–(6.4) compress the generator $\hat{G}_{i+1}, \hat{H}_{i+1}$ to the smallest length r. Their role for refinement can be seen from the next theorem applied for

$$X_* = X_{i+1},\ G_* = G_{i+1},\ H_* = H_{i+1}. \tag{6.7}$$

Theorem 6.1. *Let A, B, M, and X_* be $n\times n$ matrices and let G, H, G_-, H_-, G_* and H_* be $n\times r$ matrices, $1\le r<n$, such that M is nonsingular,*

$$\nabla_{A,B}(M) = AM - MB = GH^T,\ \operatorname{rank}(GH^T) = r, \tag{6.8}$$

$$\nabla_{B,A}(-M^{-1}) = M^{-1}\nabla_{A,B}(M)M^{-1} = G_-H_-^T, \tag{6.9}$$

$$G = MG_-,\ M^TH_- = H, \tag{6.10}$$

$$\nabla_{B,A}(X_*) = G_*H_*^T,\ X_* = \nabla_{B,A}^{-1}(G_*H_*^T). \tag{6.11}$$

Then

$$\begin{aligned} M\nabla_{B,A}(X_*+M^{-1})M &= -GH^T + MG_*H_*^TM \\ &= -(G-MG_*)H^T - MG_*(H^T - H_*^TM) \\ &= -G(H^T-H_*^TM) - (G-MG_*)H_*^TM \\ &= -G(H^T-H_*^TM) - (G-MG_*)H^T + (G-MG_*)(H^T-H_*^TM). \end{aligned}$$

Proof. *We have $M\nabla_{B,A}(X_*+M^{-1})M = M(-G_-H_-^T+G_*H_*^T)M$ (by (6.9) and (6.11)), so $M\nabla_{B,A}(X_*+M^{-1})M = -GH^T+MG_*H_*^TM$ (by (6.10)).* □

The computations in (6.1)–(6.4) minimize the 2-norms of approximations to G by MG_* and to H by M^TH_*; since $G=MG_-$ and $H=M^TH_-$, this should move G_* closer to G_- and H_* closer to H_-.

By Theorem 6.1, we have

$$\begin{aligned} N = \|M\nabla_{B,A}(X_*+M^{-1})M\|_2 &\le \\ \le \|G\|_2\|H^T-H_*^TM\|_2 &+ \|G-MG_*\|_2\|H_*^TM\|_2. \end{aligned} \tag{6.12}$$

Hence, decreasing the norms $\|H-M^TH_*\|_2$ and $\|G-MG_*\|_2$ leads to decreasing the upper bound on the norm N and consequently on the error norm

$$E = \|X_*+M^{-1}\|_2 \le N\|\nabla_{B,A}^{-1}\|_2\|M^{-1}\|^2.$$

In [26] and [31] quite tight upper and lower bounds on $\|\nabla^{-1}_{B,A}\|_2$ are derived for various often used pairs of A and B. In particular by Corollary 8.10 of [31] we have

$$\|\nabla^{-1}_{Z_e,Z_f}\|_2 \leq \sqrt{r}\frac{|\tilde{e}\tilde{f}|^{\frac{n-1}{n}}}{\min_{i,j}|e^{\frac{1}{n}}\omega^i - f^{\frac{1}{n}}\omega^j|}$$

provided that $\tilde{e} = \max\{|e|, 1/|e|\}$, $\tilde{f} = \max\{|f|, 1/|f|\}$, $\omega = \exp(2\pi\sqrt{-1}/n)$, and the operator is applied on the matrices of rank r.

We also note the respective bounds on the norms of the left and right residuals $R_{left,i} = I + X_iM$ and $R_{right,i} = I + MX_i$:

$$\|R_{left,i}\|_2 \leq E\|M\|_2, \qquad \|R_{right,i}\|_2 \leq E\|M\|_2.$$

Remark 6.1. *For a given displacement $\nabla_{A,B}(M)$ the choice of the generator pair G, H satisfying (3.1) is not unique but this choice does not affect norm N of (6.12), which only depends on GH^T. This follows because $Y_{i+1,G}$,$Y^T_{i+1,H}$ only depends on GH^T as can be seen from (6.5),(6.6) in the full rank case and from QR factorization of $M\hat{G}_{i+1}$ and $M^T\hat{H}_{i+1}$ with column pivoting ([11, Section 5.4.1]) in the rank deficient case.*

7 Numerical experiments

Let us specify a particular invertible displacement operator, which we used to perform the numerical tests. We write $C^+ = Z_{+1}$, $C^- = Z_{-1}$ (cf. (3.2)), and denote with $C^+(\boldsymbol{x}) = \sum_{i=1}^n x_i(C^+)^{i-1}$, $C^-(\boldsymbol{x}) = \sum_{i=1}^n x_i(C^-)^{i-1}$ the $(+1)$-circulant and (-1)-circulant matrices having $\boldsymbol{x}$ as the first column. We recall from [6] that:

$$\begin{aligned}
&C^+(\boldsymbol{x}) = F\,\mathrm{diag}(\boldsymbol{y})F^H,\\
&C^-(\boldsymbol{x}) = DF\,\mathrm{diag}(\hat{\boldsymbol{y}})F^HD^H,\\
&\boldsymbol{y} = \frac{1}{n}F^H\boldsymbol{x}, \hat{\boldsymbol{y}} = \frac{1}{n}F^HD^H\boldsymbol{x}, && (7.1)\\
&F = (\omega^{(i-1)(j-1)})_{i,j=1,\ldots,n}, \qquad \omega = \cos(2\pi/n) + \sqrt{-1}\sin(2\pi/n) = \exp(2\pi\sqrt{-1}/n)\\
&D = \mathrm{diag}(1, \theta, \theta^2, \ldots, \theta^{n-1}), \qquad \theta = \cos(\pi/n) + \sqrt{-1}\sin(\pi/n) = \exp(\pi\sqrt{-1}/n).
\end{aligned}$$

Consider the *invertible* operators

$$\begin{aligned}
\Delta^+(M) = C^+M - MC^-\\
\Delta^-(M) = C^-M - MC^+.
\end{aligned}$$

The following theorem summarizes some well known results (see, e.g., [1, 8, 9, 26]).

Theorem 7.1. *It holds that*

$$\begin{aligned}
&\Delta^+(M) = \sum_{i=1}^k \boldsymbol{u}_i \boldsymbol{v}_i^T \Leftrightarrow M = \frac{1}{2}\sum_{i=1}^k C^+(\boldsymbol{u}_i)C^-(J\boldsymbol{v}_i),\\
&\Delta^-(M) = \sum_{i=1}^k \boldsymbol{u}_i \boldsymbol{v}_i^T \Leftrightarrow M = -\frac{1}{2}\sum_{i=1}^k C^-(\boldsymbol{u}_i)C^+(J\boldsymbol{v}_i),\\
&\Delta^-(M^{-1}) = -M^{-1}\Delta^+(M)M^{-1},
\end{aligned}$$

where J is the permutation matrix having ones on the anti-diagonal. In particular, if $\Delta^+(M) = \sum_{i=1}^k \boldsymbol{u}_i \boldsymbol{v}_i^T$ and $\det M \neq 0$, we obtain

$$M^{-1} = \frac{1}{2}\sum_{i=1}^k C^-(M^{-1}\boldsymbol{u}_i)C^+(JM^{-T}\boldsymbol{v}_i). \tag{7.2}$$

We have implemented the classical approach based on truncating the smallest singular values and the new least squares cutting approach for the displacement operators Δ^+ and Δ^-. The software was written in Matlab[1].

Experimental tests based on this algorithm clearly show that the new *least squares cutting* approach gives more accurate results compared to the classical cutting for well-conditioned matrices. The algorithm is applied to 100 Toeplitz matrices $M = T$ of size 100-by-100 where the entries of each Toeplitz matrix are uniformly random between zero and one. For each of the 100 samples, the starting point is computed as $T^{-1} * (I + \alpha_l * R)$, where the entries of R are uniformly random between -1 and $+1$. The parameter α_l is determined such that the norm of the left residual $\|I + X_0 * T\|$ equals 1 and the norm of the right residual $\|I + T * X_0\|$ is larger than 1. Figure 1 gives a histogram of $\log_{10}(\text{cond}(T))$.

Let $R_i = I + X_i * T$ be the left residual for X_i. Then, the choice of our starting point guarantees that the convergence is reflected in the behaviour of the sequence $\|R_0\|, \|R_1\|, \ldots$ because if we perform the Newton iteration without cutting, we have $\|R_{i+1}\| \leq \|R_i\|^2$.

Figure 2 shows the results for these 100 samples. Each plot shows a histogram of $-\log_{10}\|R_i\|$ (as the x-coordinate) for $i = 0, 3$ and 6. The y-coordinate shows the number of sampled matrices (out of the total of 100) with this value of $\|R_i\|$. The histograms at the left show the results when the classical cutting is used while the histograms at the right are corresponding to the least squares cutting.

In the first iteration the residual norms are the same in all the tests, but already 3 iterations with the least squares cutting generically result in a significantly smaller left residual norm. This is illustrated in the histograms by the fact that the black area in the right figure is shifted more towards the right compared to the left plot. In the 6*th* iteration the difference between the two approaches shows up even more dramatically. For ill-conditioned matrices the classical cutting performs better compared to the new approach.

[1]Matlab is a registered trademark of the MathWorks

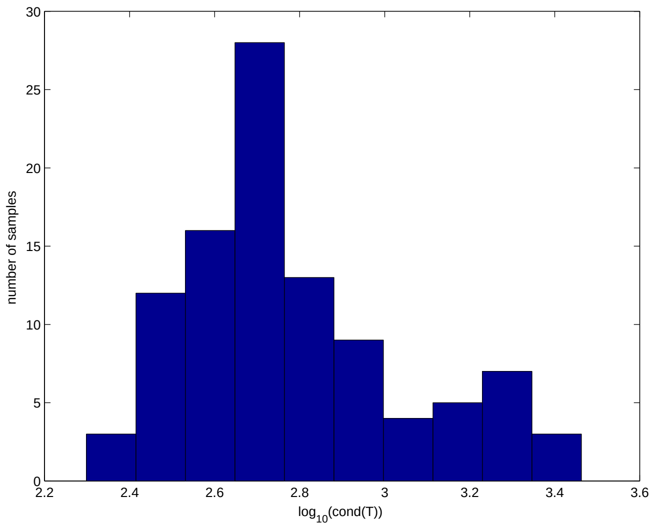

Figure 1: Histogram of $\log_{10}$ of the condition numbers of the 100 Toeplitz matrices.

8 Conclusion

We first recalled Newton's iteration algorithms for the inversion of structured matrices and then presented an alternative compression strategy based on a least-squares criterion. The numerical experiments with inverting random Toeplitz matrices show that for well-conditioned matrices this new compression scheme results in fewer iteration steps to obtain the same accuracy. Our experiments indicate that the approach is less effective for ill-conditioned Toeplitz matrices. In a companion paper [35], we propose an alternative initial iteration step which leads to a much more robust iteration scheme, especially when the matrix is ill-conditioned. To come to a user-friendly, robust and efficient iteration scheme for approximating the inverse of a structured matrix, still a lot of research has to be done. Several other cutting strategies are possible. Finding the right combination of all the possible variants for inverting a specified Toeplitz matrix is not trivial.

References

[1] G.S. Ammar and P. Gader. A variant of the Gohberg-Semencul formula involving circulant matrices. *SIAM Journal on Matrix Analysis and Applications*, 12(3):534–541, 1991.

[2] A. Ben-Israel. A note on iterative method for generalized inversion of matrices. *Mathematics of Computation*, 20:439–440, 1966.

[3] D.A. Bini and B. Meini. Approximate displacement rank and applications. In V. Olshevsky, editor, *Proceedings AMS Conference "Structured Matrices in Operator*

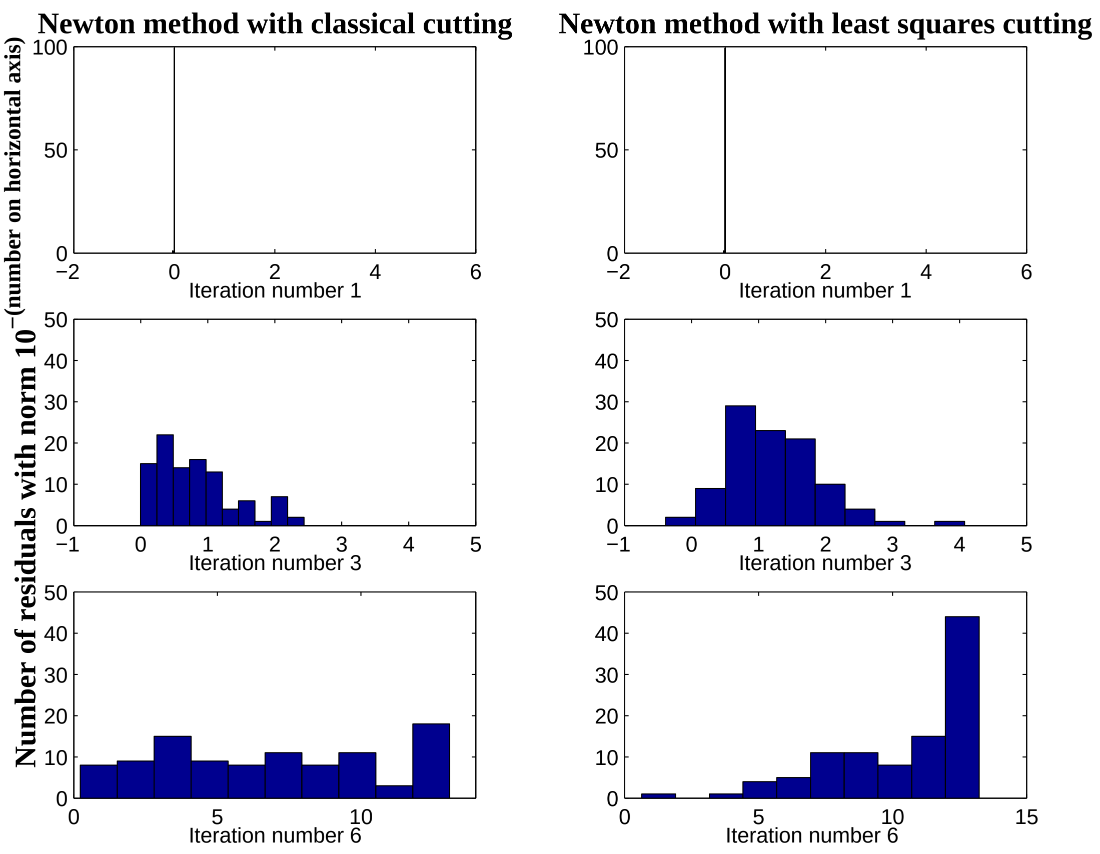

Figure 2: Distribution of the left residual norms in $1, 3$ and 6 iterations in the two approches: the classical truncation of singular values (left) and the least squares cutting (right).

Theory, Numerical Analysis, Control, Signal and Image Pocessing", Boulder, 1999, pages 215–232, Providence, R.I., 2001. American Mathematical Society.

[4] D.A. Bini and V.Y. Pan. *Polynomial and Matrix Computations, vol. 1: Fundamental Algorithms*. Birkhäuser, Boston, 1994.

[5] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra and Its Applications*, 34:103–116, 1980.

[6] W.E. Cline, R.J. Plemmons, and G. Worm. Generalized inverses of certain Toeplitz matrices. *Linear Algebra and Its Applications*, 8:25–33, 1974.

[7] I. Gohberg, T. Kailath, and V. Olshevsky. Fast Gaussian elimination with partial pivoting for matrices with displacement structure. *Mathematics of Computation*, 64(212):1557–1576, 1995.

[8] I. Gohberg and V. Olshevsky. Circulant displacement and decomposition of matrices. *Integral Equations Operator Theory*, 15:730–743, 1992.

[9] I. Gohberg and V. Olshevsky. Complexity of multiplication with vectors for structured matrices. *Linear Algebra and Its Applications*, 202:163–192, 1994.

[10] I. Gohberg and A. Semencul. On the inversion of finite Toeplitz matrices and their continuous analogs. *Matematicheskiie Issledovaniia*, 2:187–224, 1972.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1996.

[12] G. Heinig. *Beitrage zur spektraltheorie von Operatorbushec und zur algebraischen Theorei von Toeplitzmatrizen.* PhD thesis, TH Karl-Marx-Stadt, 1979.

[13] G. Heinig and K. Rost. *Algebraic Methods for Toeplitz-like Matrices and Operators.* Akademie-Verlag, Berlin, and Birkhäuser, Basel/Stuttgart, 1984.

[14] T. Kailath and A. H. Sayed (editors). *Fast Reliable Algorithms for Matrices with Structure.* SIAM Publications, Philadelphia, 1999.

[15] T. Kailath, S. Kung, and M. Morf. Displacement ranks of a matrix. *Bulletin of the American Mathematical Society*, 1:769–773, 1979.

[16] T. Kailath, S.-Y. Kung, and M. Morf. Displacement ranks of matrices and linear equations. *Journal of Mathematical Analysis and Its Applications*, 68:395–407, 1979.

[17] T. Kailath and A. Sayed. Displacement structure: Theory and applications. *SIAM Review*, 37(3):297–386, September 1995.

[18] T. Kailath, A. Vieira, and M. Morf. Inverses of Toeplitz operators, innovations and orthogonal polynomials. *SIAM Rev.*, 20:106–119, 1978.

[19] M. Morf. *Fast Algorithms for Multivariable Systems.* PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, 1974.

[20] M. Morf. Doubling algorithms for Toeplitz and related equations. In *Proceedings of IEEE International Conference on ASSP*, pages 954–959, IEEE Press, Piscataway, New Jersey, 1980.

[21] V. Olshevsky and V. Y. Pan. A unified superfast algorithm for boundary rational tangential interpolation problem. In *Proceedings of the 39th Annual IEEE Symposium Foundations of Computer Science (FOCS'98)*, pages 192–201. IEEE Computer Society Press, Los Alamitos, California, 1998.

[22] V. Y. Pan. Fast and efficient parallel inversion of Toeplitz and block Toeplitz matrices. *Operator Theory: Advances and Applications*, 40:359–389, 1989.

[23] V. Y. Pan. Parallel solution of Toeplitz-like linear systems. *J. Complexity*, 8:1–21, 1992.

[24] V. Y. Pan. Concurrent iterative algorithm for Toeplitz-like linear systems. *IEEE Transactions on Parallel and Distributed Systems*, 4:592–600, 1993.

[25] V. Y. Pan. Decreasing the displacement rank of a matrix. *SIAM Journal on Matrix Analysis and Its Applications*, 14(1):118–121, 1993.

[26] V. Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms.* Birkhäuser/Springer, Boston/New York, 2001.

[27] V. Y. Pan, S. Branham, R. E. Rosholt, and A. Zheng. Newton's iteration for structured matrices and linear systems of equations. In *Fast Reliable Algorithms for Matrices with Structure,(T. Kailath and A. H. Sayed, editors)*, pages 189–210. SIAM Publications, Philadelphia, 1999.

[28] V. Y. Pan, M. Kunin, R. E. Rosholt, and H. Cebecioğlu. Residual correction algorithms for general and structured matrices, 2001. Preprint.

[29] V. Y. Pan, Y. Rami, and X. Wang. Structured matrices and Newton's iteration. *Linear Algebra and Its Applications*, 343-344:233–265, 2002.

[30] V. Y. Pan and R. Schreiber. An improved Newton iteration for the generalized inverse of a matrix, with applications. *SIAM Journal on Scientific and Statistical Computing*, 12(5):1109–1131, 1991.

[31] V. Y. Pan and X. Wang. Inversion of displacement operators. *SIAM Journal on Matrix Analysis and Applications*, 2002. in press.

[32] V. Y. Pan and A. Zheng. Superfast algorithms for Cauchy-like matrix computations and extensions. *Linear Algebra and Its Applications*, 310:83–108, 2000.

[33] V. Y. Pan, A. Zheng, X. Huang, and O. Dias. Newton's iteration for inversion of Cauchy-like and other structured matrices. *Journal of Complexity*, 13(1):108–124, March 1997.

[34] T. Söderström and G.W. Stewart. On the numerical properties of an iterative method for computing the Moore-Penrose generalized inverse. *SIAM Journal on Numerical Analysis*, 11:61–74, 1974.

[35] M. Van Barel and G. Codevico. An adaptation of the Newton iteration method to solve symmetric positive definite Toeplitz systems. Report TW 349, Department of Computer Science, K.U.Leuven, Leuven, Belgium, November 2002.

Appendix

Mathlab Programs for Experimental Computations

```
Row=1;

for n=100:1:100
n
for Beta = 1.00:0.1:1.00
Beta
RCount=0;
while RCount<100
RCount


II=eye(n);
%FF=zeros(n,n);
%ohm=cos(2*pi/n)+sqrt(-1)*sin(2*pi/n);

FF=n*ifft(II);
DD=zeros(n,n);
theta=cos(pi/n)+sqrt(-1)*sin(pi/n);
for i=1:n
DD(i,i)=theta^(i-1);
end

Cp=Cplus(Zeta(n),n);
Cm=Cmin(Zeta(n),n);

% construct random Toeplitz matrix



C=rand(n,1);
R=rand(1,n);

T=toeplitz(C,R);

% compute the inverse of T

Tinv=inv(T);

% some info on T

Tcond=cond(T)

DeltaT=Cp*T-T*Cm;
[UT,ST,VT]=svd(DeltaT);
disp('rank of Delta+ of T')
rank(DeltaT)
UT=UT(:,1:2);
```

```
VT=VT(:,1:2);
ST=ST(1:2,1:2);

UT=UT*sqrt(ST);
VT=VT*sqrt(ST);

% determine a good starting point or the Newton iteration

% alpha=input('alpha? ');
RR=rand(n,n);
lalpha=1.0e-4;
X=Tinv*(II+lalpha*(RR-0.5));
lnorm=norm(II-X*T);
% X=Tinv*(II+alpha*(rand(n,n)-0.5))+alpha*(rand(n,n)-0.5)*norm(Tinv);
ralpha=1;
X=Tinv*(II+ralpha*(RR-0.5));
rnorm=norm(II-X*T);
%X=Tinv*(II+alpha*(rand(n,n)-0.5))+alpha*(rand(n,n)-0.5)*norm(Tinv);


ii=40;
for i=1:ii
alpha=(lalpha+ralpha)/2;
X=Tinv*(II+alpha*(RR-0.5));
DeltaX=Cm*X-X*Cp;
% disp('rank of the perturbed inverse matrix :')
% rank(DeltaX)
% pause
[UX,SX,VX]=svd(DeltaX);
cut=2;
UX=UX(:,1:cut);
VX=VX(:,1:cut);
SX=SX(1:cut,1:cut);
SX2=sqrt(SX);
F=UX*SX2;
G=VX*SX2;
%X=invDeltaMinOld(F,G);

X=invDeltaMin(F,G,FF,DD);
normR=norm(II-X*T);
if normR > Beta
ralpha=alpha;
else
lalpha=alpha;
end;
normR
end
```

```
if(norm(II-T*X)>=1.0)
RCount=RCount+1;
condT(RCount)=Tcond;
end

disp('finito')




DeltaX=Cm*X-X*Cp;
% disp('rank of the perturbed inverse matrix :')
% rank(DeltaX)
% pause
[UX,SX,VX]=svd(DeltaX);
cut=2;
UX=UX(:,1:cut);
VX=VX(:,1:cut);
SX=SX(1:cut,1:cut);
SX2=sqrt(SX);
F=UX*SX2;
G=VX*SX2;
X0=invDeltaMin(F,G,FF,DD);
%X0=invDeltaMinOld(F,G);

% check the accuracy of this starting point

normL1=norm(II-X0*T)
normR1=norm(II-T*X0)

% apply Newton iteration using the traditional cutting approach

it=20;

normL=zeros(1,it+1);
normR=zeros(1,it+1);
normL(1)=normL1;
normR(1)=normR1;
X=X0;
cut=2;

for i=1:it
X=2*X-X*T*X;
DeltaX=Cm*X-X*Cp;
[UX,SX,VX]=svd(DeltaX);
UX=UX(:,1:cut);
VX=VX(:,1:cut);
SX=SX(1:cut,1:cut);
```

```
SX2=sqrt(SX);
F=UX*SX2;
G=VX*SX2;
X=invDeltaMin(F,G,FF,DD);
% X=invDeltaMinOld(F,G);
normL(i+1)=norm(II-X*T);
normR(i+1)=norm(II-T*X);
end;

normLc(Row,:)=normL;
normRc(Row,:)=normR;

% apply Newton iteration using no cutting
it=20;

normL=zeros(1,it+1);
normR=zeros(1,it+1);
normL(1)=normL1;
normR(1)=normR1;
X=X0;

for i=1:it
X=2*X-X*T*X;
normL(i+1)=norm(II-X*T);
normR(i+1)=norm(II-T*X);
end;

normLn(Row,:)=normL;
normRn(Row,:)=normR;

% apply Newton iteration using least squares cutting

it=20;

normL=zeros(1,it+1);
normR=zeros(1,it+1);
normL(1)=normL1;
normR(1)=normR1;
X=X0;

for i=1:it
X=2*X-X*T*X;
DeltaX=Cm*X-X*Cp;
[UX,SX,VX]=svd(DeltaX);
SX2=sqrt(SX);
UX=UX*SX2;
VX=VX*SX2;
UX=UX(:,1:6);
```

```
VX=VX(:,1:6);
Ft=(T*UX)\UT;
Gt=(-T'*VX)\VT;
F=UX*Ft;
G=VX*Gt;
X=invDeltaMin(F,G,FF,DD);
% X=invDeltaMinOld(F,G);
normL(i+1)=norm(II-X*T);
normR(i+1)=norm(II-T*X);
end;

normLl(Row,:)=normL;
normRl(Row,:)=normR;

Row = Row+1;

end

end

end
```

```
Row=1;

for n=100:1:100
n
for Beta = 1.00:0.1:1.00
Beta
RCount=0;
while RCount<100
RCount


II=eye(n);
%FF=zeros(n,n);
%ohm=cos(2*pi/n)+sqrt(-1)*sin(2*pi/n);

FF=n*ifft(II);
DD=zeros(n,n);
theta=cos(pi/n)+sqrt(-1)*sin(pi/n);
for i=1:n
DD(i,i)=theta^(i-1);
end

Cp=Cplus(Zeta(n),n);
Cm=Cmin(Zeta(n),n);

% construct random Toeplitz matrix



C=rand(n,1);
R=rand(1,n);

T=toeplitz(C,R);

% compute the inverse of T

Tinv=inv(T);

% some info on T

Tcond=cond(T)

DeltaT=Cp*T-T*Cm;
[UT,ST,VT]=svd(DeltaT);
disp('rank of Delta+ of T')
rank(DeltaT)
UT=UT(:,1:2);
```

```
VT=VT(:,1:2);
ST=ST(1:2,1:2);

UT=UT*sqrt(ST);
VT=VT*sqrt(ST);

% determine a good starting point or the Newton iteration

% alpha=input('alpha? ');
RR=rand(n,n);
lalpha=1.0e-4;
X=Tinv*(II+lalpha*(RR-0.5));
lnorm=norm(II-X*T);
% X=Tinv*(II+alpha*(rand(n,n)-0.5))+alpha*(rand(n,n)-0.5)*norm(Tinv);
ralpha=1;
X=Tinv*(II+ralpha*(RR-0.5));
rnorm=norm(II-X*T);
%X=Tinv*(II+alpha*(rand(n,n)-0.5))+alpha*(rand(n,n)-0.5)*norm(Tinv);


ii=40;
for i=1:ii
alpha=(lalpha+ralpha)/2;
X=Tinv*(II+alpha*(RR-0.5));
DeltaX=Cm*X-X*Cp;
% disp('rank of the perturbed inverse matrix :')
% rank(DeltaX)
% pause
[UX,SX,VX]=svd(DeltaX);
cut=2;
UX=UX(:,1:cut);
VX=VX(:,1:cut);
SX=SX(1:cut,1:cut);
SX2=sqrt(SX);
F=UX*SX2;
G=VX*SX2;
%X=invDeltaMinOld(F,G);

X=invDeltaMin(F,G,FF,DD);
normR=norm(II-X*T);
if normR > Beta
ralpha=alpha;
else
lalpha=alpha;
end;
normR
end
```

```
if(norm(II-T*X)>=1.0)
RCount=RCount+1;
condT(RCount)=Tcond;
end

disp('finito')




DeltaX=Cm*X-X*Cp;
% disp('rank of the perturbed inverse matrix :')
% rank(DeltaX)
% pause
[UX,SX,VX]=svd(DeltaX);
cut=2;
UX=UX(:,1:cut);
VX=VX(:,1:cut);
SX=SX(1:cut,1:cut);
SX2=sqrt(SX);
F=UX*SX2;
G=VX*SX2;
X0=invDeltaMin(F,G,FF,DD);
%X0=invDeltaMinOld(F,G);

% check the accuracy of this starting point

normL1=norm(II-X0*T)
normR1=norm(II-T*X0)

% apply Newton iteration using the traditional cutting approach

it=20;

normL=zeros(1,it+1);
normR=zeros(1,it+1);
normL(1)=normL1;
normR(1)=normR1;
X=X0;
cut=2;

for i=1:it
X=2*X-X*T*X;
DeltaX=Cm*X-X*Cp;
[UX,SX,VX]=svd(DeltaX);
UX=UX(:,1:cut);
VX=VX(:,1:cut);
SX=SX(1:cut,1:cut);
```

```
SX2=sqrt(SX);
F=UX*SX2;
G=VX*SX2;
X=invDeltaMin(F,G,FF,DD);
% X=invDeltaMinOld(F,G);
normL(i+1)=norm(II-X*T);
normR(i+1)=norm(II-T*X);
end;

normLc(Row,:)=normL;
normRc(Row,:)=normR;

% apply Newton iteration using no cutting
it=20;

normL=zeros(1,it+1);
normR=zeros(1,it+1);
normL(1)=normL1;
normR(1)=normR1;
X=X0;

for i=1:it
X=2*X-X*T*X;
normL(i+1)=norm(II-X*T);
normR(i+1)=norm(II-T*X);
end;

normLn(Row,:)=normL;
normRn(Row,:)=normR;

% apply Newton iteration using least squares cutting

it=20;

normL=zeros(1,it+1);
normR=zeros(1,it+1);
normL(1)=normL1;
normR(1)=normR1;
X=X0;

for i=1:it
X=2*X-X*T*X;
DeltaX=Cm*X-X*Cp;
[UX,SX,VX]=svd(DeltaX);
SX2=sqrt(SX);
UX=UX*SX2;
VX=VX*SX2;
UX=UX(:,1:6);
```

```
VX=VX(:,1:6);
Ft=(T*UX)\UT;
Gt=(-T'*VX)\VT;
F=UX*Ft;
G=VX*Gt;
X=invDeltaMin(F,G,FF,DD);
% X=invDeltaMinOld(F,G);
normL(i+1)=norm(II-X*T);
normR(i+1)=norm(II-T*X);
end;

normLl(Row,:)=normL;
normRl(Row,:)=normR;

Row = Row+1;

end

end

end
```