NUMERICAL ANALYSIS PROJECT
MANUSCRIPT NA-92-01

JANUARY 1992

A Look-Ahead Algorithm for the Solution of General Hankel Systems

by

Roland W. Freund
Hongyuan Zha

NUMERICAL ANALYSIS PROJECT
COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

A Look-Ahead Algorithm for the Solution of General Hankel Systems

Roland W. Freund*
RIACS, Mail Stop T041-5
NASA Ames Research Center
Moffett Field, CA 94035

Hongyuan Zha†
Scientific Computing and Computational Mathematics
Stanford University
Stanford, CA 94305

Summary. The solution of linear systems of equations with Hankel coefficient matrices can be computed with only $\mathcal{O}(n^2)$ arithmetic operations, as compared to $\mathcal{O}(n^3)$ operations for the general case. However, the classical Hankel solvers require the nonsingularity of all leading principal submatrices of the Hankel matrix. The known extensions of these algorithms to general Hankel systems can handle only exactly singular submatrices, but not ill-conditioned ones, and hence they are numerically unstable. In this paper, a stable procedure for solving general nonsingular Hankel systems is presented, using a look-ahead technique to skip over singular or ill-conditioned submatrices. The proposed approach is based on a look-ahead variant of the nonsymmetric Lanczos process that was recently developed by Freund, Gutknecht, and Nachtigal. We first derive a somewhat more general formulation of this look-ahead Lanczos algorithm in terms of formally orthogonal polynomials, which then yields the look-ahead Hankel solver as a special case. We prove some general properties of the resulting look-ahead algorithm for formally orthogonal polynomials. These results are then utilized in the implementation of the Hankel solver. We report some numerical experiments for Hankel systems with ill-conditioned submatrices.

Mathematics Subject Classifications (1991): 65F05, 15A57, 42C05

*The research of the first author was supported by DARPA via Cooperative Agreement NCC 2-387 between NASA and the Universities Space Research Association (USRA).

†The research of the second author was supported in part by NSF grant DRC-8412314 and Cooperative Agreement NCC 2-387 between NASA and the Universities Space Research Association (USRA).

1 Introduction

Many important applications lead to linear systems of equations

$$H_n x_n = b_n \tag{1.1}$$

with real or complex coefficient matrices of the special form

$$H_n = [\, h_{j+k} \,]_{j,k=0,\ldots,n} = \begin{bmatrix} h_0 & h_1 & h_2 & \cdots & h_n \\ h_1 & & \iddots & \iddots & \vdots \\ h_2 & & \iddots & & \vdots \\ \vdots & \iddots & & & h_{2n-1} \\ h_n & \cdots & \cdots & h_{2n-1} & h_{2n} \end{bmatrix} \tag{1.2}$$

A matrix of the type (1.2) is called a *Hankel matrix*, and in the sequel, we refer to the corresponding linear system (1.1) as a ***Hankel system.*** Note that Hankel matrices are always symmetric, but they are non-Hermitian if complex entries occur. For example, Hankel systems arise in connection with orthogonal polynomials [5, 8], Padé approximation [13, 3], the minimal realization problem in systems theory [23, 22, 21, 16], and the Berlekamp-Massey algorithm for decoding Reed-Solomon and BCH codes [1, 27, 7, 20].

It is well known that the special structure of H_n can be exploited when solving Hankel systems, and there are algorithms that require only $\mathcal{O}(n^2)$ arithmetic operations for the solution of (1.1), as compared to $\mathcal{O}(n^3)$ operations for general systems of order $n + 1$. These algorithms are based on a factorization of H_n of the type

$$U_n^T H_n U_n = D_n, \tag{1.3}$$

where

$$U_n = [\, u_{jk} \,]_{j,k=0,\ldots,n} = \begin{bmatrix} 1 & u_{01} & \cdots & u_{0n} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \cdots & 0 & 1 \end{bmatrix} \tag{1.4}$$

is a unit upper triangular matrix, and

$$D_n = \operatorname{diag}(\, \delta^{(0)}, \delta^{(1)}, \ldots, \delta^{(l(n))}) \tag{1.5}$$

is a nonsingular block diagonal matrix. Furthermore, (1.3) and the solution x_n of (1.1) are computed recursively by updating the decompositions $U_m^T H_m U_m = D_m$ and the solutions x_m of those leading $(m + 1) \times (m + 1)$ subsystems $H_m x_m = b_m$ of (1.1) for which $m < n$ and H_m is nonsingular. A classical method of this type is due to Trench [31]. His algorithm requires that H_n is ***strongly regular,*** i.e., all leading principal submatrices H_m, $m = 0, 1, \ldots, n$, of H_n are nonsingular. It generates a factorization of the form (1.3), where D_n is a scalar diagonal matrix, i.e., $l(n) = n$ in (1.5). However, strong regularity is only guaranteed in special cases, such as Hermitian positive definite H_n, and generally, singular principal submatrices H_m, m < ***n,*** cannot be excluded. Rissanen [30] and others (see [19] and the references therein) have proposed extensions of Trench's algorithm to general nonsingular Hankel systems, which can handle singular leading principal submatrices. These algorithms again generate factorizations of the form (1.3), where, in general, D_n is now block diagonal. Furthermore, in (1.5), each block $\delta^{(k)}$ of size $s_k > 1$ corresponds to $s_k - 1$ successive singular leading principal submatrices of H_n.

In order to obtain a numerically stable Hankel solver, it is crucial to skip not only over exactly singular, but also over nonsingular, yet ill-conditioned leading principal submatrices. However, all known extensions of Trench's algorithm to general Hankel systems are designed to handle only singular submatrices exactly. At best, they can handle ill-conditioned submatrices approximately by treating a nearly singular submatrix as an exactly singular one. The purpose of this paper is to present an extension of Trench's algorithm for solving general nonsingular Hankel matrices, designed to handle singular, as well as ill-conditioned leading submatrices exactly.

It is well known (see, e.g., [29, 14]) that Hankel matrices and their factorizations **(1.3)** are intimately connected with the nonsymmetric Lanczos process [25] for tridiagonalizing general non-Hermitian square matrices. In particular, singular or ill-conditioned leading principal submatrices correspond to exact or near-breakdowns in the classical form of the Lanczos algorithm. Due to the problem of breakdowns, the nonsymmetric Lanczos method did not receive much attention in the past, although it has been known for quite some time that exact breakdowns (see [14, 24, 16]) and even near-breakdowns [28] can be overcome by using so-called look-ahead techniques. In recent years, there has been a true revival of the nonsymmetric Lanczos process; we refer the reader to [17, 9] and the references given there. Freund, Gutknecht, and Nachtigal [10] have developed a look-ahead Lanczos algorithm that can handle exact and near-breakdowns in the classical process. Here, we first derive a somewhat more general formulation of this look-ahead Lanczos algorithm, from which we then immediately obtain our look-ahead Hankel solver as a special case.

The remainder of this paper is organized as follows. In Section 2, we present a look-ahead Lanczos process for formally orthogonal polynomials and give elementary proofs of some of its properties. In Section 3, we describe our look-ahead algorithm for solving general Hankel systems. In Section 4, it is shown how Trench's method and other Hankel solvers can be obtained as special cases from the look-ahead algorithm. In Section 5, we report results of numerical experiments with Hankel matrices that have various kinds of ill-conditioned submatrices. Finally, in Section 6, we make some concluding remarks.

Throughout the paper, all vectors and matrices are allowed to have real or complex entries. As usual, M^T and M^H denote the transpose and conjugate transpose of a matrix M, respectively. The smallest and largest singular value of M is denoted by $\sigma_{\min}(M)$ and $\sigma_{\max}(M)$, respectively. The vector norm $\|x\| = \sqrt{x^H x}$ is the Euclidean norm, and $\|M\| = \sigma_{\max}(M)$ is the corresponding matrix norm. Moreover, for square matrices M of size > 1, we denote by $\kappa(M) = \|M\| \cdot \|M^{-1}\|$ the Euclidean condition number, and, if M is 1×1, we set $\kappa(M) = 1/|M|$. Whenever we call a matrix ill-conditioned, it is with respect to this condition number. We denote by

$$\mathcal{P}_n = \{\varphi(\lambda) \equiv \sigma_0 + \sigma_1\lambda + \cdots + \sigma_n\lambda^n \mid \sigma_0, \sigma_1, \ldots, \sigma_n \in \mathbb{C}\}$$

the set of all complex polynomials of degree at most n and by $\mathcal{P}$ the set of all complex polynomials. The symbol 0 will be used for the number zero, the zero matrix, and the polynomial $\varphi(\lambda) \equiv 0$; its actual meaning and, in the case of the zero matrix, its dimension will be apparent from the context. Finally, we denote by I the identity matrix; its dimension will always be evident.

2 A Look-Ahead Lanczos Algorithm for Polynomials

In this section, we derive a look-ahead Lanczos process for ***formally orthogonal polynomials*** (referred to as FOPs hereafter) and present elementary proofs of some of its properties.

2.1 Formally Orthogonal Polynomials

Let $\{h_j\}_{j=0}^{\infty}$ be a given infinite sequence[1] of real or complex numbers, and let H_n, $n = 0, 1, \ldots$, be the associated family of Hankel matrices (1.2). We define a symmetric bilinear form

$$\langle\psi, \varphi\rangle, \quad \psi, \varphi \in \mathcal{P}, \tag{2.1}$$

by setting

$$\langle\psi, \varphi\rangle := v^T H_n u \tag{2.2}$$

for any two polynomials

$$\varphi(\lambda) \equiv [1 \ \lambda \ \cdots \ \lambda^n]u, \quad \psi(\lambda) \equiv [1 \ \lambda \ \cdots \ \lambda^n]v, \quad u, \ v \in \mathbb{C}^{n+1}, \tag{2.3}$$

of degree at most n, $n = 0, 1, \ldots$. Note that

$$h_j = \langle\lambda^j, 1\rangle, \quad j = 0, 1, \ldots,$$

and hence h_j is just the jth moment associated with $\langle\cdot, \cdot\rangle$.

A polynomial $\varphi_n \in \mathcal{P}_n$, $\varphi_n \not\equiv 0$, that satisfies

$$\langle\psi, \varphi_n\rangle = 0 \quad \text{for all } \psi \in \mathcal{P}_{n-1}, \tag{2.4}$$

is called a FOP (with respect to the bilinear form (2.1)) of degree n *(see, e.g.,* [8, 17]). Note that, in general, the bilinear form (2.1) does not define a positive definite inner product. In particular, in contrast to polynomials that are orthogonal with respect to a true inner product, FOPs φ_n need not exist or need not be unique for every degree n *(see* [17] for

[1]One can always assume that the sequence is infinite, by simply setting $h_j := 0$, $j = n+1, n+2, \ldots$, if only a finite sequence $h_0, h_1, \ldots, h_n$ is given.

details). A FOP φ_n is called ***regular*** if it is uniquely determined by (2.4) up to a scalar factor. Regular FOPs φ_n have maximal degree ***n,*** and in the following we will always assume that regular FOPs are normalized to be monic, i.e.,

$$\equiv u_{0n} \quad u_{1n}\lambda \quad \cdots \quad u_{n-1,n}\lambda^{n-1} \quad \lambda^n.$$

Using (2.4), (2.3), and (1.2), one readily verifies that the coefficients $u_{0n}, u_{1n}, \ldots, u_{n-1,n}$ of φ_n are given as the solution of the special Hankel system

(2.5)
$$H_{n-1}\begin{bmatrix} u_{0n} \\ u_{1n} \\ \vdots \\ u_{n-1,n} \end{bmatrix} = -\begin{bmatrix} h_n \\ h_{n+1} \\ \vdots \\ h_{2n-1} \end{bmatrix}.$$

Furthermore, in view of (2.5), we have the following result.

Lemma ***2.1 A regular FOP of degree n exists if, and only if, H_{n-1} is nonsingular.***

2.2 The Algorithm

Let $\{\varphi_{n_j}\}_{j=0}^{J}$ denote the sequence of all existing regular FOPs. Here $J = \infty$ or $J < \infty$; note that, in view of Lemma **2.1,** the latter case occurs if, and only if, there exists an integer $n_\star$ $(=: n_J)$ such that $H_{n_\star-1}$ is nonsingular and H_n is singular for all $\boldsymbol{n} \geq \boldsymbol{n}_\star$. It is well known (see, e.g., [8, **16,** 17]) that the existing regular FOPs are connected via three-term recurrences of the form

(2.6)
$$\begin{aligned} \varphi_{n_{j+1}}(\lambda) &= \mathrm{A},(\lambda)\varphi_{n_j}(\lambda) - \gamma_{n_j}\varphi_{n_{j-1}}(\lambda), \\ &\text{where } \psi_{n_j} \in \mathcal{P}_{n_{j+1}-n_j}, \quad \gamma_{n_j} \in \mathbf{C}. \end{aligned}$$

The relation (2.6) is the basis for the modified Lanczos algorithms given in [24, 16, 17], which skip over exact breakdowns in the classical Lanczos method [25] and generate all existing regular FOPs. In particular, if a regular FOP φ_n exists for each n, then (2.6) reduces to a standard three-term recurrence, which is the basis of the classical Lanczos process. We recall (see, e.g., [10]) that an exact breakdown occurs in step n of the standard Lanczos algorithm if no regular FOP of degree ***n*** exists.

Of course, in finite precision arithmetic, a numerically stable procedure also needs to skip over near-breakdowns in the classical Lanczos method. Therefore, instead of generating all existing regular FOPs, one attempts to compute only a subsequence $\{\varphi_{n_{j_k}}\}_{k=0}^{K} \subseteq \{\varphi_{n_j}\}_{j=0}^{J}$ of "well-defined" regular FOPs; for simplicity, we will set $n_k := n_{j_k}$ in the sequel (cf. (2.7)). After introducing some further notation, we will present a sketch of such a look-ahead Lanczos process for FOPs.

The algorithm generates a sequence of polynomials $\{\varphi_n\}_{n=0}^{\infty}$, where each polynomial φ_n is monic and of exact degree n. We use the indices

(2.7)
$$0 =: n_0 < n_1 < \cdots < n_k < \cdots,$$

to mark those polynomials that are generated as regular FOPs φ_{n_k}, $k = 0, 1, \ldots$; the remaining polynomials are called ***inner.*** Note that the condition (2.4) is void for $n = 0$, and hence $\varphi_0 = 1$ is a regular FOP of degree 0. Furthermore, for each fixed n, *we* define $l = l(n)$ by

$$n_l \leq n < n_{l+1},$$

so that $l = l(n)$ always denotes the number of the last regular FOP with degree $\leq n$.

Then, the first $n + 1$ polynomials $\varphi_0, \varphi_1, \ldots, \varphi_n$ can be grouped into $l+1$ blocks:

$$\begin{aligned} \Phi^{(k)} &:= \left[\varphi_{n_k} \quad \varphi_{n_k+1} \quad \cdots \quad \varphi_{n_{k+1}-1}\right], \quad k = 0, 1, \ldots, l-1, \\ \Phi^{(l)} &:= \left[\varphi_{n_l} \ \varphi_{n_l+1} \cdots \varphi_n\right]. \end{aligned} \tag{2.8}$$

Note that the first polynomial φ_{n_k} in each block is a regular FOP, while the remaining polynomials in each block are inner polynomials. We remark that the lth block is complete if $n = n_{l+1} - 1$; in this case, the next polynomial φ_{n+1} is constructed as a regular FOP. Otherwise, if $n < n_{l+1} - 1$, the lth block is still incomplete, and φ_{n+1} is added to the lth block as an inner polynomial.

So far, we have not specified how to actually construct the inner polynomials. The point is that the inner polynomials can be chosen such that the φ_n's from blocks corresponding to different indices k are still orthogonal to each other. More precisely, as we will prove in Section 2.3, the following block orthogonality relations hold:

$$\left\langle \Phi^{(k)}, \Phi^{(j)} \right\rangle = \begin{cases} 0 & \text{if } j \neq k, \\ \delta^{(k)} & \text{if } j = k, \end{cases} \quad \text{for all } j, k = 0, 1, \ldots, l, \tag{2.9}$$

where

$$\begin{aligned} &\delta^{(k)} \quad \text{is nonsingular,} \quad k = 0, 1, \ldots, l-1, \\ \text{and} \quad &\delta^{(l)} \quad \text{is nonsingular if } n = n_{l+1} - 1. \end{aligned} \tag{2.10}$$

Here and in the sequel, we have used the notation

$$\langle \Phi, \Psi \rangle := \begin{bmatrix} \langle \phi_0, \psi_0 \rangle & \cdots & \langle \phi_0, \psi_k \rangle \\ \vdots & & \vdots \\ \langle \phi_j, \psi_0 \rangle & \cdots & \langle \phi_j, \psi_k \rangle \end{bmatrix} \in \mathbf{C}^{(j+1)\times(k+1)},$$

where

$$\Phi = [\phi_0 \quad \phi_1 \quad \cdots \quad \phi_j] \quad \text{and} \quad \Psi = [\psi_0 \quad \psi_1 \quad \cdots \quad \psi_k]$$

are row vectors of polynomials in P.

After these preliminaries, the look-ahead Lanczos process for FOPs can be sketched as follows.

Algorithm 2.2 (Look-ahead Lanczos process for FOPs)

```
0) Set $\varphi_0 = 1$, $\Phi^{(0)} = \varphi_0$, $\delta^{(0)} = \langle \Phi^{(0)}, \Phi^{(0)} \rangle$.
   Set $n_0 = 0$, $l = 0$, $\Phi^{(-1)} = 0$.
```

For $n = 0, 1, \ldots,$ *do :*

1) *Decide whether to construct* φ_{n+1} *as a regular FOP or as an inner polynomial and go to 2) or 3), respectively.*

2) *(Regular step.) Compute*

(2.11) $$\alpha_n = (\delta^{(l)})^{-1}\langle\Phi^{(l)}, \lambda\varphi_n\rangle, \quad \beta_n = (\delta^{(l-1)})^{-1}\langle\Phi^{(l-1)}, \lambda\varphi_n\rangle,$$

(2.12) $$\varphi_{n+1} = \lambda\varphi_n - \Phi^{(l)}\alpha_n - \Phi^{(l-1)}\beta_n,$$

set $n_{l+1} = n + 1$, $l = l + 1$, $\Phi^{(l)} = \emptyset$, *and go to 4).*

3) *(Inner step.) Compute*

(2.13) $$\beta_n = (\delta^{(l-1)})^{-1}\langle\Phi^{(l-1)}, \lambda\varphi_n\rangle,$$

(2.14) $$\varphi_{n+1} = \lambda\varphi_n - \sum_{m=n_l}^{n}\xi_m\varphi_m - \Phi^{(l-1)}\beta_n,$$

4) *Set*

$$\Phi^{(l)} = [\,\Phi^{(l)} \quad \varphi_{n+1}\,], \quad \delta^{(l)} = \langle\Phi^{(l)}, \Phi^{(l)}\rangle.$$

We remark that, by (2.10), the matrices $\delta^{(l)}$ and $\delta^{(l-1)}$ in (2.11) and (2.13) are guaranteed to be nonsingular.

Moreover, note that, in (2.14), $\xi_m \in \mathbb{C}$, $m = 0, 1, \ldots,$ are arbitrary recurrence coefficients. For the Hankel solver, we will chose $\xi_m \equiv 0$ (cf. (2.26)).

Finally, we remark that Algorithm 2.2 reduces to the classical Lanczos process [25], if only regular steps 2) are performed.

2.3 Properties

In this subsection, we prove some properties of Algorithm 2.2. We will make use of the relation

(2.15) $$\langle\psi, \lambda\varphi\rangle = \langle\lambda\psi, \varphi\rangle \text{ for all } \varphi, \psi \in \mathcal{P},$$

which is an immediate consequence of (2.2) and the Hankel structure of the matrices (1.2).

First, we show that the polynomials $\varphi_0, \varphi_1, \ldots, \varphi_n$ generated by Algorithm 2.2 do indeed satisfy the orthogonality relations (2.9) and (2.10).

Theorem **2.3** *Let* $n \in \{0, 1, \ldots\}$ *and* $l = l(n)$. *Then:*

(2.16) $$\langle\Phi^{(k)}, \varphi_n\rangle = 0 \quad \text{for all} \quad k = 0, 1, \ldots, l-1,$$

and

(2.17) $$\delta^{(l-1)} := \langle\Phi^{(l-1)}, \Phi^{(l-1)}\rangle \text{ is nonsingular.}$$

Proof. The **proof** is by induction on n. For $n = 0$, by (2.7), $l = 0$, and thus both conditions (2.16) and (2.17) are void.

Now let $n \geq 0$, $l = l(n)$, and assume that (2.16) and (2.17) hold for all polynomials $\varphi_0, \varphi_1, \dots, \varphi_n$. Hence, using the notation (2.8), we have

$$\langle \Phi^{(k)}, \Phi^{(j)} \rangle = 0 \quad \text{for all} \quad j \neq k,\ j, k = 0, 1, \dots, l. \tag{2.18}$$

We need to show that

$$\langle \Phi^{(k)}, \varphi_{n+1} \rangle = 0 \quad \text{for all} \quad k = 0, 1, \dots, l' - 1, \tag{2.19}$$

and that

$$\delta^{(l'-1)} := \langle \Phi^{(l'-1)}, \Phi^{(l'-1)} \rangle \quad \text{is nonsingular,} \tag{2.20}$$

where

$$l' = l(n+1) = \begin{cases} l & \text{if } \varphi_{n+1} \text{ is inner,} \\ l + 1 & \text{if } \varphi_{n+1} \text{ is regular.} \end{cases} \tag{2.21}$$

First, we remark that

$$\langle \Phi^{(k)}, \lambda\varphi_n \rangle = 0 \quad \text{for all} \quad k = 0, 1, \dots, l-2, \tag{2.22}$$

or, equivalently,

$$\langle \varphi_j, \lambda\varphi_n \rangle = 0 \quad \text{for all} \quad j = 0, 1, \dots, n_{l-1} - 1.$$

Indeed, if φ_j is a regular FOP, then, with (2.15) and (2.12) (with n replaced by j), one obtains the relations

$$\begin{aligned} \langle \varphi_j, \lambda\varphi_n \rangle &= \langle \lambda\varphi_j, \varphi_n \rangle \\ &= \langle \varphi_{j+1}, \varphi_n \rangle + \alpha_j^T \langle \Phi^{l(j)}, \varphi_n \rangle + \beta_j^T \langle \Phi^{l(j)-1}, \varphi_n \rangle, \end{aligned} \tag{2.23}$$

where, by (2.18), all terms in (2.23) vanish. Similarly, if φ_j is an inner polynomial, then, with (2.15), (2.14) (with n replaced by j), and (2.18), it follows that

$$\begin{aligned} \langle \varphi_j, \lambda\varphi_n \rangle &= \langle \lambda\varphi_j, \varphi_n \rangle \\ &= \langle \varphi_{j+1}, \varphi_n \rangle + \sum_{m=n_{l(j)}}^{j} \xi_m \langle \varphi_m, \varphi_n \rangle + \beta_j^T \langle \Phi^{l(j)-1}, \varphi_n \rangle \\ &= 0. \end{aligned}$$

Now we turn to the proof of (2.19). First, consider the case that φ_{n+1} is constructed as an inner polynomial. Note that, by (2.21), $l' = l$, and let $k \in \{0, 1, \dots, l-1\}$. Using (2.14), (2.18), and (2.22), one easily verifies that

$$\begin{aligned} \langle \Phi^{(k)}, \varphi_{n+1} \rangle &= \langle \Phi^{(k)}, \lambda\varphi_n \rangle - \sum_{m=n_l}^{n} \xi_m \langle \Phi^{(k)}, \varphi_m \rangle - \langle \Phi^{(k)}, \Phi^{(l-1)} \rangle \beta_n \\ &= \langle \Phi^{(k)}, \lambda\varphi_n \rangle - \langle \Phi^{(k)}, \Phi^{(l-1)} \rangle \beta_n \\ &= 0. \end{aligned}$$

Here, for the case $k = l - 1$, the last equality follows from (2.13). Next, consider the case that φ_{n+1} is constructed as a regular FOP. Note that, by (2.21), $l' = l + 1$, and let $k \in \{0, 1, \ldots, l\}$. Then, with (2.12), (2.18), and (2.22), we obtain

$$\begin{aligned}
\langle \Phi^{(k)}, \varphi_{n+1} \rangle &= \langle \Phi^{(k)}, \lambda\varphi_n \rangle - \langle \Phi^{(k)}, \Phi^{(l)} \rangle \alpha_n - \langle \Phi^{(k)}, \Phi^{(l-1)} \rangle \beta_n \\
&= \begin{cases} 0 & \text{for } k = 0, 1, \ldots, l-2, \\ \langle \Phi^{(l-1)}, \lambda\varphi_n \rangle - \langle \Phi^{(l-1)}, \Phi^{(l-1)} \rangle \beta_n = 0 & \text{for } k = l-1, \\ \langle \Phi^{(l)}, \lambda\varphi_n \rangle - \langle \Phi^{(l)}, \Phi^{(l)} \rangle \alpha_n = 0 & \text{for } k = l. \end{cases}
\end{aligned}$$

Here, the last two equalities follow from the definition of β_n and α_n in (2.11).

Finally, we turn to the proof of (2.20). If φ_{n+1} is an innerpolynomial, then $l' = l$, and (2.20) is identical with the assumption (2.17). Therefore, we only need to consider the case that φ_{n+1} is a regular FOP. Suppose that (2.20) is false, i.e., the matrix $\delta^{(l'-1)} = \langle \Phi^{(l)}, \Phi^{(l)} \rangle$ is singular. This implies that there exists a vector c with

$$\langle \Phi^{(l)}, \Phi^{(l)} c \rangle = \langle \Phi^{(l)}, \Phi^{(l)} \rangle c = 0 \quad \text{and} \quad \Phi^{(l)} c \neq 0, \tag{2.24}$$

and we set

$$\tilde{\varphi} := \varphi_{n+1} + \Phi^{(l)} c.$$

Note that $\tilde{\varphi}$ is a monic polynomial of degree $n + 1$, and, by (2.24), $\tilde{\varphi} \neq \varphi_{n+1}$. On the other hand, by (2.19) and (2.24), it follows that $\tilde{\varphi}$ satisfies

$$\langle \psi, \tilde{\varphi} \rangle = 0 \text{ for all } \psi \in \mathcal{P}_n.$$

Consequently, $\tilde{\varphi}$ and φ_{n+1} are two different monic regular FOPs of degree $n + 1$, which contradicts the uniqueness of regular FOPs. This concludes the proof. □

Remark 2.1 The result of Theorem 2.3 is also contained in Gutknecht's work [17, 18] on the Lanczos algorithm. However, the approach taken in [17, 18] is based on the intimate connection between the Lanczos process and Padé approximation. In particular, for the proofs in [17, 18], nontrivial results from the theory of Padé approximation are used. In contrast, our proof of Theorem 2.3 is self-contained and completely elementary.

Remark 2.2 The Lanczos method is a powerful tool for iterative matrix computations, see, e.g., the survey paper [9] and the references given therein. More precisely, let A be a given $N \times N$ matrix, and let $v_0, w_0 \in \mathbb{C}^N$ be two nonzero starting vectors. Then, the Lanczos process can be used to generate two sequences of vectors $v_0, v_1, \ldots$, and $w_0, w_1, \ldots$, of the form

$$v_n = \varphi_n(A) v_0 \quad \text{and} \quad w_n = \varphi_n(A^T) w_0, \quad n = 0, 1, \ldots. \tag{2.25}$$

If all polynomials φ_n in (2.25) are regular FOPs, then the resulting algorithm is the classical nonsymmetric Lanczos method and the Lanczos vectors are biorthogonal:

$$w_j^T v_k = \langle \varphi_j, \varphi_k \rangle = 0 \text{ for all } j \neq k.$$

Freund, Gutknecht, and Nachtigal [10] have developed a look-ahead Lanczos algorithm that overcomes possible breakdowns and near-breakdowns in the classical Lanczos procedure. The scheme proposed in [10] can also be viewed as a special case of Algorithm 2.2; it generates vectors (2.25) that correspond to regular FOPs or inner polynomials. For details of the resulting algorithm and the look-ahead strategy, we refer the reader to [10].

From now on, we assume that the recurrence coefficients in (2.14) are chosen as

$$\xi_m = 0 \quad \text{for all } m = 0, 1, \dots . \tag{2.26}$$

In Section 3.2, we will need the following result.

Theorem 2.4 ***The blocks $\delta^{(k)}$ in (2.9) are Hankel matrices.***

Proof. By (2.9) and (2.8), the matrix $\delta^{(k)}$ is given by

$$\delta^{(k)} = \left[\langle \varphi_m, \varphi_n \rangle \right]_{m,n=n_k}^{n_{k+1}-1} .$$

Hence, we have to show that

$$\langle \varphi_m, \varphi_{n+1} \rangle = \langle \varphi_{m+1}, \varphi_n \rangle \tag{2.27}$$

for all

$$m, n \in \{n_k, n_k + 1, \dots, n_{k+1} - 2\}. \tag{2.28}$$

Let m and n be any pair satisfying (2.28). Note that φ_{m+1} and φ_{n+1} are both inner polynomials. With (2.14), (2.26), (2.16), and (2.15), it follows that

$$\begin{aligned} \langle \varphi_m, \varphi_{n+1} \rangle &= \langle \varphi_m, \lambda\varphi_n \rangle - \langle \varphi_m, \Phi^{(k-1)} \rangle \beta_n \\ &= \langle \varphi_m, \lambda\varphi_n \rangle \\ &= \langle \lambda\varphi_m, \varphi_n \rangle. \end{aligned} \tag{2.29}$$

Similarly, we have

$$\begin{aligned} \langle \varphi_{m+1}, \varphi_n \rangle &= \langle \lambda\varphi_m, \varphi_n \rangle - \beta_m^T \langle \Phi^{(k-1)}, \varphi_n \rangle \\ &= \langle \lambda\varphi_m, \varphi_n \rangle, \end{aligned}$$

and, together with (2.29), we obtain the equation (2.27). □

3 A Look-Ahead Hankel System Solver

We now turn to the solution of Hankel systems $H_n x_n = b_n$, where

$$b_n = [\mu_0 \quad \mu_1 \quad \cdots \quad \mu_n]^T \in \mathbb{C}^{n+1} \tag{3.1}$$

is a general right-hand side. In this section, we present a look-ahead Hankel solver based on Algorithm 2.2 and discuss some implementation issues. We will also show how to choose the look-ahead step sizes.

3.1 The Algorithm

Let φ_n, $n = 0, 1, \ldots,$ be the sequence of polynomials generated by Algorithm 2.2. Each of these polynomials is monic and can be represented in the form

$$\varphi_n(\lambda) \equiv u_{0n} + u_{1n}\lambda + \cdots + u_{n-1,n}\lambda^{n-1} + \lambda^n.$$

Now, the key observation is that the vector

$$u_n := [u_{0n} \quad u_{1n} \quad \ldots \quad u_{n-1,n} \quad 1]^T,$$

which consists of the coefficients of φ_n, is just the last column of the upper triangular matrix U_n *(1.4)* in the decomposition (1.3) of H_n. Indeed, by simply rewriting, by means of (2.2) and (2.3), the block orthogonality relations (2.9) in terms of the coefficient vectors of the polynomials $\varphi_0, \varphi_1, \ldots, \varphi_n$, we arrive at the factorization (1.3). Furthermore, note that the last block $\delta^{(l)}$, $l = l(n)$, in (1.5) is given by

$$\delta^{(l)} = (U^{(l)})^T H_n U^{(l)}.$$

Here and in the sequel, we denote by

$$U^{(k)} := [u_{ij}]_{i=0,\cdots,n_{k+1}-1;\, j=n_k,\cdots,n_{k+1}-1}, \quad k = 0, 1, \ldots, l-1,$$
$$U^{(l)} := [u_{ij}]_{i=0,\cdots,n;\, j=n_l,\cdots,n},$$

the submatrices of U_n that contain the coefficients of the blocks of polynomials (2.8). Note that the following relations hold:

$$U_{n_{k+1}-1} = \left[\begin{array}{c|c} U_{n_k-1} & U^{(k)} \\ 0 & \end{array}\right], \quad k = 0, 1, \ldots, l-1, \quad U_{-1} := \emptyset,$$
$$U_n = \left[\begin{array}{c|c} U_{n_l-1} & U^{(l)} \\ 0 & \end{array}\right].$$

Finally, the coefficient vector u_n is called ***regular vector*** if the corresponding polynomial φ_n is generated as a regular FOP. If φ_n is an inner polynomial, then u_n is referred to as an ***inner vector.***

With these preparations, we can reformulate Algorithm 2.2 as the following look-ahead procedure for computing Hankel factorizations of the form (1.3).

Algorithm 3.1 (Look-ahead algorithm for Hankel decompositions (1.3))

```
0) Set $u_0 = 1$, $U^{(0)} = u_0$, $\delta^{(0)} = (U^{(0)})^T H_0 U^{(0)}$.
   Set $n_0 = 0$, $l = 0$, $U^{(-1)} = 0$.
For $n = 0, 1, \ldots,$ do :
1) Decide whether to construct $u_{n+1}$ as a regular or as an inner vector
   and go to 2) or 3), respectively.
```

2) (Regular step.) Compute α_n and β_n by solving

$$\delta^{(l)}\alpha_n = \begin{bmatrix} U^{(l)} \\ 0 \end{bmatrix}^T H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix} \quad \text{and} \quad \delta^{(l-1)}\beta_n = \begin{bmatrix} U^{(l-1)} \\ 0 \end{bmatrix}^T H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix}, \tag{3.2}$$

respectively.

Compute

$$u_{n+1} = \begin{bmatrix} 0 \\ u_n \end{bmatrix} - \begin{bmatrix} U^{(l)} \\ 0 \end{bmatrix} \alpha_n - \begin{bmatrix} U^{(l-1)} \\ 0 \end{bmatrix} \beta_n, \tag{3.3}$$

set $n_{l+1} = n + 1$, $l = l + 1$, $U^{(l)} = \emptyset$, and go to 4).

3) (Inner step.) Compute β_n by solving

$$\delta^{(l-1)}\beta_n = \begin{bmatrix} U^{(l-1)} \\ 0 \end{bmatrix}^T H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix} \tag{3.4}$$

and set

$$u_{n+1} = \begin{bmatrix} 0 \\ u_n \end{bmatrix} - \begin{bmatrix} U^{(l-1)} \\ 0 \end{bmatrix} \beta_n. \tag{3.5}$$

4) Set

$$U^{(l)} = \left[\begin{array}{c|c} U^{(l)} & \\ 0 & u_{n+1} \end{array} \right], \quad \delta^{(l)} = (U^{(l)})^T H_{n+1} U^{(l)}.$$

We remark that, by Lemma 2.1, the matrix H_n is nonsingular if u_{n+1} is constructed as a regular vector. In particular, the matrix H_n is guaranteed to be nonsingular if a regular step 2) is performed in Algorithm 3.1. Furthermore, in view of (2.5) (with n replaced by $n + 1$), the vector

$$x_n := \begin{bmatrix} u_{0,n+1} & u_{1,n+1} & \cdots & u_{n,n+1} \end{bmatrix}^T,$$

which is obtained from a regular vector u_{n+1} by deleting the last element $u_{n+1,n+1} = 1$, is the solution of the following Hankel system with special right-hand side:

$$H_n x_n = - \begin{bmatrix} h_{n+1} \\ h_{n+2} \\ \vdots \\ h_{2n+1} \end{bmatrix}. \tag{3.6}$$

Systems of this type are referred to as Yule-Walker equations. Actually, when Hankel systems arise in practice, they are often of the special form (3.6).

The solution of Hankel systems $H_n x_n = b_n$ with general right-hand sides (3.1) can also be obtained easily. Recall that H_n is guaranteed to be nonsingular for $n = n_k - 1$, $k = 1, 2, \ldots, l$, and we only update the solution x_n for these values of n. To this end, we simply need to insert the following procedure at the beginning of each regular step 2) in Algorithm 3.1:

Set $n' = n_l - 1$, and partition $U^{(l)}$, b_n, and H_n as follows:

(3.7)
$$U^{(l)} = \begin{bmatrix} \hat{U} \\ \tilde{U} \end{bmatrix}, \quad b_n = \begin{bmatrix} b_{n'} \\ \mu \end{bmatrix}, \quad H_n = \begin{bmatrix} H_{n'} & S \\ S^T & \tilde{H} \end{bmatrix},$$

where $\tilde{U}$ and μ just contain the last $n - n'$ rows of $U^{(l)}$ and b_n, respectively.
Compute y by solving

(3.8)
$$\delta^{(l)} y = \tilde{U}^T (\mu - S^T x_{n'}),$$

and set

(3.9)
$$x_n = \begin{bmatrix} x_{n'} \\ 0 \end{bmatrix} + U^{(l)} y.$$

Indeed, one easily verifies that x_n given by (3.9) and (3.8) is the solution of $H_n x_n = b_n$ by using (3.7) and the relation

$$H_n U^{(l)} = \begin{bmatrix} 0 \\ \tilde{U}^{-T} \delta^{(l)} \end{bmatrix}.$$

In the following, we denote by

$$s_k := n_{k+1} - n_k, \quad k = 0, 1, \ldots,$$

the dimension of the kth block $\delta^{(k)}$ of the matrix D_n in the factorization (1.3). We will also refer to s_k as the length of the kth look-ahead step.

3.2 Implementation Details

In this section, we discuss some implementation details for Algorithm 3.1 and give an operation count. Among other things, we will show that-despite the look-ahead procedure--each iteration step involves the computation of only ***two*** inner products. This is exactly the same as in the classical Trench algorithm for strongly regular matrices.

For the operation count, we will use the following convention: a computation that requires only arithmetic operations of order $\mathcal{O}(s_k^3)$ or less is considered ***negligible.*** The rationale is that-except for contrived examples—the Hankel matrices we encountered in many applications only have at most a small number of consecutive ill-conditioned leading principal submatrices. Therefore, Algorithm 3.1 will mostly perform regular steps, and if look-ahead steps do occur, their length s_k is small.

At each step (regular or inner) of Algorithm 3.1, we first compute

$$g_l = \tilde{U}^{-T} f_l.$$

Here, as in (3.7)) $\tilde{U}$ just consists of the last s_l rows of $U^{(l)}$, and f_l denotes the last column of $\delta^{(l)}$. Then, we have

(3.10)
$$H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix} = \begin{bmatrix} 0 \\ g_l \\ \tau_1 \\ \tau_2 \end{bmatrix},$$

where τ_1 and τ_2 are obtained by forming two inner products:

$$\tau_1 = [\, h_{n+1} \quad \cdots \quad h_{2n+1} \,]\, u_n,\ \tau_2 = [\, h_{n+2} \quad \cdots \quad h_{2n+2} \,]\, u_n.$$

Now, consider a regular step 2) of Algorithm 3.1. By means of (3.10), the right-hand sides of the equations in (3.2), and hence α_n and β_n, can be generated with negligible work. To form the second term in the right-hand side of (3.3)) we further need to compute s_l SAXPYs[2]. As we will see below, the third term in (3.3) can be obtained by one scalar multiplication of a vector. In summary, at each regular step, the arithmetic operation count is two inner products and $s_l + 1$ SAXPYs.

Now, we consider an inner step 3) of Algorithm 3.1. It can be verified that

$$\begin{bmatrix} U^{(l-1)} \\ 0 \end{bmatrix}^T H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix} = \begin{bmatrix} 0 \\ f_{0l} \end{bmatrix},$$

where f_{0l} is the first element of f_l. We can precompute

$$z_l = U^{(l-1)} (\delta^{(l-1)})^{-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix},$$

which costs s_{l-1} SAXPYs. Then (3.5) can be written as

$$u_{n+1} = \begin{bmatrix} 0 \\ u_n \end{bmatrix} - \begin{bmatrix} z_l \\ 0 \end{bmatrix} f_{0l};$$

hence for each inner step, we only need to compute one additional SAXPY.

Next, it is shown that, for the calculation of $\delta^{(l)}$ in step 4), no extra inner products are needed. If u_{n+1} is generated as a regular vector, then, with (3.10), we have

$$\begin{aligned} \delta^{(l)} &= u_{n+1}^T H_{n+1} u_{n+1} = u_{n+1}^T H_{n+1} \left(\begin{bmatrix} 0 \\ u_n \end{bmatrix} - \begin{bmatrix} U^{(l)} \\ 0 \end{bmatrix} \alpha_n - \begin{bmatrix} U^{(l-1)} \\ 0 \end{bmatrix} \beta_n \right) \\ &= u_{n+1}^T H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix} \\ &= u_{n+1}^T \begin{bmatrix} 0 \\ g_l \\ \tau_1 \\ \tau_2 \end{bmatrix}. \end{aligned}$$

Note that here we also used the orthogonality conditions

$$u_{n+1}^T H_{n+1} \begin{bmatrix} u_j \\ 0 \end{bmatrix} = 0 \text{ for all } j = 0, 1, \ldots, n.$$

[2]A SAXPY operation is $z = \alpha x + y$, where x and y are vectors and α is a scalar, see, e.g., [12].

Now, consider the case of an inner step. Note that $\delta^{(l)}$ is a matrix of dimension bigger than 1. However, the key observation here is that, by Theorem 2.4, $\delta^{(l)}$ is a Hankel matrix, and therefore, at each step we only need to generate the last two elements of the last column of $\delta^{(l)}$. The last element $u_{n+1}^T H_{n+1} u_{n+1}$ is obtained as follows:

$$\begin{aligned} u_{n+1}^T H_{n+1} u_{n+1} &= u_{n+1}^T H_{n+1} \left(\begin{bmatrix} 0 \\ u_n \end{bmatrix} - \begin{bmatrix} U^{(l-1)} \\ 0 \end{bmatrix} \beta_n \right) \\ &= u_{n+1}^T H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix} \\ &= u_{n+1}^T \begin{bmatrix} 0 \\ g_l \\ \tau_1 \\ \tau_2 \end{bmatrix} . \end{aligned}$$

The other element is given by

$$\begin{bmatrix} u_n \\ 0 \end{bmatrix}^T \boldsymbol{H}_{n+1} u_{n+1} = u_n^T \begin{bmatrix} 0 \\ g_l \\ \tau_1 \end{bmatrix} .$$

All these computations are negligible, and hence the work for step 4) is also negligible. In summary, the operation count for each inner step is two inner products and two SAXPYs. We remark that the vector z_l will also be used in the calculation of the next regular vector.

Finally, we turn to the updating procedure of solutions for general right-hand sides. To compute the right-hand side of (3.8), we need to generate $S^T x_n'$, which involves s_l inner products. The computation of the vector x_n in (3.9) requires another s_l SAXPYs. Note that x_n is only updated once within each cycle of s_l steps. Thus, in the average, the update procedure requires one inner product and one SAXPY per step.

In the following table, we summarize the operation count for one step of Algorithm 3.1, and for the updating procedure of solutions for general right-hand sides.

	regular step	inner step	update of x_n
inner products/step	2	2	1
SAXPYs/step	$s_l + 1$	2	1

As we will see in Section 4, the operation count of Trench's algorithm for solving Hankel systems with general right-hand sides is three inner products and three SAXPYs per recursive step. The overhead of the look-ahead Hankel solver is $s_l - 1$ SAXPYs for every look-ahead step with size s_l. Since the algorithm will mostly perform regular steps, and the look-ahead step sizes are usually small, the overhead of our look-ahead algorithm is quite negligible.

3.3 The Look-ahead Step Size

In this section, we discuss the criteria that are used in step 1) of Algorithm 3.1 to decide whether the next vector is constructed as a regular or an inner one. For this purpose, there are two quantities, denoted by $\kappa(\Gamma^{(l)})$ and η_n, that are monitored throughout the process. Both of them are obtained from local information only. In particular, we do not need to estimate the condition number of the current leading principal submatrix H_n. The decision about building a regular or an inner vector is then based on a comparison of $\kappa(\Gamma^{(l)})$ and η_n with two threshold parameters COND (> 0) and GFACTOR (≥ 1), respectively. We stress that COND and GFACTOR need not be specified by the user; they are determined dynamically during the run of the algorithm. The only input that is required from the user is the number $s_{\max}$, which is the maximal allowed block size in the decomposition **(1.3).**

The initialization phase of the algorithm is as follows. As the first block, we set $\delta^{(0)} = H_m$, where H_m is the matrix with the smallest condition number[3] $\kappa(H_m)$ among H_s, $s = 0, 1, \ldots, s_{\max} - 1$. Then we build the next vector u_{m+1} as a regular vector and we set $n_1 = m + 1$. Furthermore, we initialize COND to be this smallest condition number $\kappa(H_m)$ and set GFACTOR to **1.**

Now we consider a general iteration step of Algorithm 3.1. Since in the classical Trench algorithm for strongly regular matrices only regular vectors are built (cf. Section 4), the goal for the look-ahead Hankel solver should be to build as many regular vectors as possible. Therefore, in each iteration step, we first pretend that u_{n+1} can actually be constructed as a regular vector. Then, we compute the condition number of

$$\Gamma^{(l)} := \tilde{U}^{-T} \delta^{(l)} \tilde{U}^{-1},$$

where $\tilde{U}$ is the lower triangular part of $U^{(l)}$ (cf. (3.7)), and check whether

$$\kappa(\Gamma^{(l)}) \leq 2*\text{COND}. \tag{3.11}$$

If (3.11) is not satisfied, then we go to step 3) in Algorithm 3.1 and build u_{n+1} as an inner vector. To justify the criterion (3.11), recall that for a regular step, both $\delta^{(l)}$ and, by (1.3), H_n are required to be nonsingular. Actually, at the end of this section, we will point out that $\kappa(\Gamma^{(l)})$ is also closely related to $\kappa(H_n)$.

If (3.11) is satisfied, we compute α_n in (3.2) and its Ii-norm $\eta_n = \|\alpha_n\|_1$. Then, we check whether

$$\eta_n \leq 2*\text{GFACTOR}. \tag{3.12}$$

If the criterion (3.12) is not satisfied, we proceed with step 3) and construct u_{n+1} as an inner vector. The justification for (3.12) is as follows. Note that (1.3) (with n replaced by $n + 1$) can be rewritten as

$$H_{n+1} = U_{n+1}^{-T} D_{n+1} U_{n+1}^{-1}. \tag{3.13}$$

If one would compute the decomposition (3.13) of H_{n+1} directly by means of Gaussian elimination, then pivoting would be used to ensure that the size of the off-diagonal elements of U_{n+1}^{-1} is bounded by 1. Indeed, this is the key to the numerical stability of Gaussian elimination. However, for an algorithm exploiting the Hankel structure, pivoting is not an option, since it does not preserve the structure. Roughly speaking, the look-ahead Algorithm 3.1

[3] Recall the definition of condition number at the end of Section 1.

avoids pivoting by building a block of size bigger than **1,** whenever a small pivot is encountered in Gaussian elimination. Consequently, the look-ahead strategy should also guarantee that off-diagonal elements of U_{n+1}^{-1} are not too large. Since

$$U_{n+1}^{-1} = I + C + C^2 + \cdots + C^{n+1}, \text{ where } C := I - U_{n+1},$$

a large off-diagonal element of U_{n+1} usually leads to a large off-diagonal element of U_{n+1}^{-1}. Therefore, in each step of the Algorithm **3.1,** we limit the growth in the newly generated off-diagonal elements of U_{n+1}, which are just the components of u_{n+1}. This is done by imposing the check (3.12).

If both criteria (3.11) and (3.12) are satisfied, then we proceed with step 2) in Algorithm 3.1 and construct u_{n+1} as a regular vector.

It cannot be excluded that the algorithm has reached the maximal block size $s_{\max}$, but the two checks for building the next vector as a regular one are still not satisfied. In this case, we increase the values of the threshold parameters COND and GFACTOR, so that, when we would rebuild the block, the two criteria (3.11) and (3.12) are satisfied within the maximal look-ahead size. This can be done by setting COND to be the condition number of $\Gamma^{(l)}$ that is minimal within the maximal look-ahead range. Then, the second parameter GFACTOR is reset to the value of the corresponding η_n. In this way, the algorithm is guaranteed to choose the $\Gamma^{(l)}$ with best possible condition numbers, while at the same time the off-diagonal elements of U_n are not too large. With the described look-ahead strategy, the algorithm will produce accurate solutions of Hankel systems as long as the coefficient matrix has at most $s_{\max} - 1$ consecutive ill-conditioned leading principal submatrices.

Remark 3.1 The philosophy of our look-ahead strategy is similar to the one used by Freund, Gutknecht, and Nachtigal [10] in their implementation of the Lanczos method for nonsymmetric matrices, in that the only input required is the maximal look-ahead size, while any other parameters are determined dynamically during the run of the algorithm. For the nonsymmetric Lanczos process, the crucial point is to keep the Lanczos vectors (2.25) sufficiently linearly independent. In the implementation [10], this is ensured by a look-ahead strategy based only on criteria of the type (3.12)) while the current block $\delta^{(l)}$ is merely checked for nonsingularity. However, for the look-ahead Hankel solver, it is crucial that, at the beginning of each regular step, the block $\delta^{(l)}$ is sufficiently nonsingular, since it is used in updating the solution of the Hankel system (cf. (3.S) and (3.9)). Hence, in contrast to [10], the look-ahead strategy for the Hankel solver also involves the check (3.11) to ensure that the blocks $\delta^{(k)}$ are well-conditioned.

We conclude this section with a. discussion of the connection of the condition numbers of $\Gamma^{(l)}$ and H_n. Set $n' := n_l - 1$ and let H_n be partitioned as follows:

$$H_n = \begin{bmatrix} H_{n'} & S \\ S^T & \tilde{H} \end{bmatrix},$$

where $H_{n'}$ is the last well-conditioned Hankel matrix. With this notation, $\Gamma^{(l)}$ is, in fact, the

Schur complement[4] of $H_{n'}$ in H_n, and we have

$$H_n = \begin{bmatrix} I & 0 \\ S^T H_{n'}^{-1} & I \end{bmatrix} \begin{bmatrix} H_{n'} & 0 \\ 0 & \Gamma^{(l)} \end{bmatrix} \begin{bmatrix} I & H_{n'}^{-1} S \\ 0 & I \end{bmatrix},$$

which implies that

$$\begin{aligned} \Gamma^{(l)} &= [0 \ I] \begin{bmatrix} I & 0 \\ -S^T H_{n'}^{-1} & I \end{bmatrix} H_n \begin{bmatrix} I & -H_{n'}^{-1} S \\ 0 & I \end{bmatrix} \begin{bmatrix} 0 \\ I \end{bmatrix} \\ &= [-S^T H_{n'}^{-1} \ I] H_n \begin{bmatrix} -H_{n'}^{-1} S \\ I \end{bmatrix}. \end{aligned} \qquad (3.14)$$

From (3.14), it follows that

$$\sigma_{\max}(\Gamma^{(l)}) \le \|[-S^T H_{n'}^{-1} \quad I]\| \cdot \left\| \begin{bmatrix} -H_{n'}^{-1} S \\ I \end{bmatrix} \right\| \sigma_{\max}(H_n).$$

It is easy to verify that

$$\|[-S^T H_{n'}^{-1} \quad I]\| \le \sqrt{1 + (\|S\| \kappa(H_{n'})/\|H_{n'}\|)^2}$$

Therefore, letting

$$\varrho = 1 + (\|S\| \kappa(H_{n'})/\|H_{n'}\|)^2,$$

we have

$$\sigma_{\max}(\Gamma^{(l)}) \le \varrho \, \sigma_{\max}(H_n). \qquad (3.15)$$

Similarly, we can show that

$$\sigma_{\min}(H_n) \le \varrho \, \sigma_{\min}(\Gamma^{(l)}). \qquad (3.16)$$

First, consider the case that $\Gamma^{(l)}$ is of size bigger than **1.** Then by combining **(3.15)** and (3.16), we arrive at

$$\kappa(\Gamma^{(l)}) \le \varrho^2 \kappa(H_n).$$

With the same technique, we can show that

$$\kappa(H_n) \le \varrho^2 \kappa(\Gamma^{(l)}),$$

and hence

$$\kappa(H_n)/\varrho^2 \le \kappa(\Gamma^{(l)}) \le \varrho^2 \kappa(H_n). \qquad (3.17)$$

For the case that $\Gamma^{(l)}$ is a scalar., from (3.15) and (3.16), we obtain

$$\frac{1}{\varrho \, \sigma_{\max}(H_n)} \le \kappa(\Gamma^{(l)}) \le \frac{\varrho}{\sigma_{\min}(H_n)}. \qquad (3.18)$$

Roughly speaking, the inequalities (3.17) and (3.18) state that, if $H_{n'}$ is well-conditioned, then H_n is well-conditioned if, and only if, $\Gamma^{(l)}$ is well-conditioned.

[4]We remark that $\tilde{U}$ is an upper triangular Toeplitz matrix. This, combined with the fact that $\delta^{(l)}$ is Hankel, shows that $\Gamma^{(l)}$ is a quasi-Hankel matrix, by which we mean a matrix with Hankel displacement rank 2 [6, Chapter 4], [26].

4 Special Cases

In this section, we point out that the classical Trench algorithm [31] and several of its extensions [30, 27, 16, 7] are special cases of Algorithm 3.1. Specifically, it is shown that, if in Algorithm 3.1 one only skips over exactly singular leading principal submatrices, then the procedure reduces to the one described in [27, 16, 7], and the factorization (1.3) reduces to the decomposition given in Theorem 1 in [16]. In addition, if the Hankel matrix is strongly regular, then we recover the classical Trench algorithm.

We now assume that the sequence $\{n_k\}_{k=0}^{K}$ in (2.7) consists of the indices of all existing regular FOPs (cf. Section 2.2). Then, by Theorem 1 in [16], the blocks $U^{(k)}$ and $\delta^{(k)}$ in the decomposition (1.3) of H_n are of the form

$$(4.1)\qquad U^{(k)} = \left|\begin{matrix} u_{0n_k} & 0 & \cdots & 0 \\ \vdots & u_{0n_k} & \ddots & \vdots \\ u_{n_k-1,n_k} & \vdots & \ddots & 0 \\ 1 & u_{n_k-1,n_k} & \ddots & u_{0n_k} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n_k-1,n_k} \\ 0 & \cdots & 0 & 1 \end{matrix}\right|, \quad \delta^{(k)} = \begin{bmatrix} 0 & \cdots & 0 & \theta_0^{(k)} \\ \vdots & .\cdot\cdot & .\cdot\cdot & \theta_1^{(k)} \\ 0 & \theta_0^{(k)} & .\cdot\cdot & \vdots \\ \theta_0^{(k)} & \theta_1^{(k)} & \cdots & \theta_{s_k-1}^{(k)} \end{bmatrix}.$$

Matrices $\delta^{(k)}$ of the type (4.1) are called ***anti-lower triangular*** Hankel matrices. Next, we show that the construction of the blocks (4.1) is a special case of Algorithm 3.1.

We only need to examine one look-ahead step here. Let H_{n_l-1} be nonsingular and H_{n_l} be singular. Hence, u_{n_l} is a. regular vector, and in Algorithm 3.1, the next vector u_{n_l+1} is constructed as an inner vector. Using (1.3), it is easily verified that $H_{n_l}u_{n_l}$ = 0, and thus the vectors

$$(4.2)\qquad H_{n_l+1}\begin{bmatrix} u_{n_l} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ * \end{bmatrix} \text{ and } H_{n_l+1}\begin{bmatrix} 0 \\ u_{n_l} \end{bmatrix} = \begin{bmatrix} 0 \\ * \\ * \end{bmatrix}$$

have possible nonzero elements (indicated by $*$'s) only in their last position and last two positions, respectively. From (3.4) and the second relation in (4.2), it follows that β_{n_l} = 0. Therefore,

$$u_{n_l+1} = \begin{bmatrix} 0 \\ u_{n_l} \end{bmatrix}$$

is just the vector u_{n_l} shifted down by one position. This procedure of building inner vectors by shifting down the previous vector by one position will continue until we first encounter an n such that the vector

$$H_n u_n = [\gamma_0 \ \cdots \ \gamma_n]^T$$

has a nonzero element $\gamma_0^{(l)} := \gamma_{n_l}$ in position n_l. In matrix form, this can be expressed as follows:

$$(4.3)\qquad H_n \begin{bmatrix} u_{0n_l} & 0 & \cdots & 0 \\ \vdots & u_{0n_l} & \ddots & \vdots \\ u_{n_l-1,n_l} & \vdots & \ddots & 0 \\ 1 & u_{n_l-1,n_l} & \ddots & u_{0n_l} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n_l-1,n_l} \\ 0 & \cdots & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ G^{(l)} \end{bmatrix},$$

where

$$(4.4)\qquad G^{(l)} = \begin{bmatrix} 0 & \cdots & 0 & \gamma_0^{(l)} \\ \vdots & \iddots & \iddots & \gamma_1^{(l)} \\ 0 & \gamma_0^{(l)} & \iddots & \vdots \\ \gamma_0^{(l)} & \gamma_1^{(l)} & \cdots & \gamma_{n-n_l}^{(l)} \end{bmatrix}, \quad \text{with } \gamma_0^{(l)} \neq 0.$$

Clearly, $G^{(l)}$ is nonsingular, and together with the nonsingularity of H_{n_l-1}, it follows that H_n is nonsingular. Hence the next vector u_{n+1} will be constructed as a regular vector, and one sets $n_{l+1} := n + 1$ and $s_l := n_{l+1} - n_l$. Note that $n - n_l = s_l - 1$ in (4.4). Moreover, we have

$$(4.5)\qquad H_{n+1} \begin{bmatrix} 0 \\ u_n \end{bmatrix} = \begin{bmatrix} 0 & \gamma_0^{(l)} & \gamma_1^{(l)} & \cdots & \gamma_{s_l-1}^{(l)} & \gamma_{s_l}^{(l)} & \gamma_{s_l+1}^{(l)} \end{bmatrix}^T.$$

Later, we will also need the relation

$$(4.6)\qquad H_n \begin{bmatrix} u_{n_{l-1}} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & \gamma_0^{(l-1)} & \gamma_1^{(l-1)} & \cdots & \gamma_{s_l-1}^{(l-1)} & \gamma_{s_l}^{(l-1)} \end{bmatrix}^T,$$

which follows from (4.3) (with l replaced by $l-1$) and, in the case $s_l \geq s_{l-1}$, by defining the elements $\gamma_{s_{l-1}}^{(l-1)}, \ldots, \gamma_{s_l}^{(l-1)}$ accordingly.

Clearly, (4.3) shows that Algorithm 3.1 generates blocks $U^{(l)}$ of the form (4.1). Furthermore, from (1.3) and (4.3), we obtain

$$(4.7)\qquad \delta^{(l)} = \tilde{U}^T G^{(l)}.$$

Here, as in (3.7), $\tilde{U}$ consists of the lower triangular part of $U^{(l)}$. To recover the decomposition in Theorem 1 of [16], it remains to note that $\delta^{(l)}$ is an anti-lower triangular Hankel matrix. This can be verified by direct computation using (4.7), (4.3), and (4.4); also note that

$$\theta_0^{(l)} = \gamma_0^{(l)}.$$

Next, we give formulas for the coefficients vectors α_n and β_n in (3.3) in terms of the quantities $\gamma_j^{(l)}$ and $\gamma_j^{(l-1)}$, $j = 0, 1, \ldots, s_l$. First, consider β_n. With (4.5) and (3.2), it follows that $\delta^{(l-1)}\beta_n = \begin{bmatrix} \gamma_0^{(l)} & 0 \end{bmatrix}^T$, and thus

$$(4.8)\qquad \beta_n = \frac{\gamma_0^{(l)}}{\gamma_0^{(l-1)}} \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Now we turn to α_n. By multipling (3.3) from the left by H_{n+1} and using **(4.5)** and (4.8), we obtain

$$(4.9)\qquad \boldsymbol{H}_{n+1}u_{n+1} = \begin{bmatrix} 0 \\ \gamma_0^{(l)} \\ \vdots \\ \gamma_{s_l+1}^{(l)} \end{bmatrix} - H_{n+1}\begin{bmatrix} U^{(l)} \\ 0 \end{bmatrix}\alpha_n - \frac{\gamma_0^{(l)}}{\gamma_0^{(l-1)}} H_{n+1}\begin{bmatrix} u_{n_l-1} \\ 0 \end{bmatrix}.$$

Note that-since u_{n+1} is a regular vector-the left-hand side of (4.9) is zero, except for possibly the last component. Hence, by deleting the last component in (4.9) and by considering only the last s_l rows in the resulting relation, we get the equation

$$\textbf{(4.10)}\qquad 0 = \begin{bmatrix} \gamma_1^{(l)} \\ \vdots \\ \gamma_{s_l}^{(l)} \end{bmatrix} - [0 \quad I]\, H_n U^{(l)}\alpha_n - \frac{\gamma_0^{(l)}}{\gamma_0^{(l-1)}} [0 \quad I]\; H_n \begin{bmatrix} u_{n_l-1} \\ 0 \end{bmatrix}.$$

By (4.3), the second term of the right-hand side of (4.10) is just $G^{(l)}\alpha_n$. Furthermore, from (4.6), we have

$$[0 \; I] H_n \begin{bmatrix} u_{n_l-1} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & \gamma_1^{(l-1)} & \gamma_2^{(l-1)} & \cdot & \cdot & \cdot & \gamma_{s_l}^{(l-1)} \end{bmatrix}^T.$$

Therefore, it follows from (4.10) that

$$\textbf{(4.11)}\qquad \alpha_n = (G^{(l)})^{-1}\left(\begin{bmatrix} \gamma_1^{(l)} \\ \gamma_2^{(l)} \\ \vdots \\ \gamma_{s_l}^{(l)} \end{bmatrix} - \frac{\gamma_0^{(l)}}{\gamma_0^{(l-1)}}\begin{bmatrix} \gamma_1^{(l-1)} \\ \gamma_2^{(l-1)} \\ \vdots \\ \gamma_{s_l}^{(l-1)} \end{bmatrix}\right).$$

We remark that the formulas (4.S) and **(4.11)** are just the ones stated in Corollary 5 ("Magnus's algorithm") in [16].

Finally, we turn to the strongly regular case, and assume that only regular vectors are built, i.e., $l \equiv n$ in Algorithm 3.1. Then, the equation (3.3) reduces to the three-term recurrence

$$u_{n+1} = \begin{bmatrix} 0 \\ u_n \end{bmatrix} - \begin{bmatrix} u_n \\ 0 \end{bmatrix}\alpha_n - \begin{bmatrix} u_{n-1} \\ 0 \end{bmatrix}\beta_n,$$

where, in view of 4.8) and (4.11),

$$\alpha_n = \frac{\gamma_1^{(n)}}{\gamma_0^{(n)}} - \frac{\gamma_1^{(n-1)}}{\gamma_0^{(n-1)}} \quad \text{and} \quad \beta_n = \frac{\gamma_0^{(n)}}{\gamma_0^{(n-1)}}.$$

These are the formulas used in the classical Trench algorithm [31]. Next, we consider the process of updating the solution for $H_n x_n$ = $\boldsymbol{b}$,. The partitioning in (3.7) now reads as follows:

$$u_n = \begin{bmatrix} \hat{u} \\ 1 \end{bmatrix}, \quad b_n = \begin{bmatrix} b_{n-1} \\ \mu_n \end{bmatrix}, \quad H_n = \begin{bmatrix} H_{n-1} & s_{n-1} \\ s_{n-1}^T & h_n \end{bmatrix},$$

where $s_{n-1} = [\, h_n \quad \cdots \quad h_{2n-1} \,]^T$. Note that $H_{n-1}\hat{u} = -s_{n-1}$, and thus

$$\mu_n - s_{n-1}^T x_{n-1} = \mu_n + \hat{u}^T b_{n-1} = u_n^T b_n.$$

Hence, the formulas (3.8) and (3.9) for the general case reduces to the update:

$$x_n = \begin{bmatrix} x_{n-1} \\ 0 \end{bmatrix} + \frac{u_n^T b_n}{\delta^{(n)}} u_n.$$

Finally, we note that the operation count for Trench's algorithm is two inner products and two SAXPYs per recursive step. To update the solutions x_n for Hankel systems with general right-hand sides, we need to compute one extra inner product and one SAXPY per step. This operation count, together with the total number of multiplications and additions after n steps, is summarized in the following table.

	inner products/step	SAXPYs/step	multiplications	additions
factor	2	2	$2n^2 + \mathcal{O}(n)$	$2n^2 + \mathcal{O}(n)$
update of x_n	1	1	$n^2 + \mathcal{O}(n)$	$n^2 + \mathcal{O}(n)$

5 Numerical Experiments

In this section, we report some results of numerical experiments using the look-ahead Algorithm 3.1. First, we give a brief description of how to generate Hankel matrices with prescribed rank or condition number profiles. Then we present some numerical examples, together with some comments.

5.1 Generating Hankel Matrices With Ill-Conditioned Principal Submatrices

In this subsection, we describe two methods we used for generating Hankel matrices with various types of rank profiles and ill-conditioned principal submatrices. The condition numbers of the ill-conditioned principal submatrices can generally be controlled by adding random Hankel matrices to the generated ones. The simplest examples are Hankel matrices with only one ill-conditioned leading principal submatrix. To this end, we first generate a random Hankel matrix H_n, then for some m ($< n$) compute an eigenvalue λ of the Toeplitz matrix $H_m E_m$, where E_m is the exchange matrix, with 1's on the anti-diagonal. Setting the element h_m of H_n to $h_m - \lambda$ gives a. Hankel matrix which in exact arithmetic will have a singular principal submatrix of order $m + 1$. Then we form

$$\tilde{H}_n = H_n + \epsilon \hat{H},$$

where $\hat{H}$ is a Hankel matrix, and ϵ is a small number to control the condition number of the $(m + 1)$th leading principal submatrix of $\tilde{H}_n$.

To obtain Hankel matrices with more general rank or condition number profiles, we make use of the decomposition theorem in [16, Theorem 1]. For some reason which we still do not fully understand, the generated Hankel matrices themselves tend to be very ill-conditioned. Whenever this is the case, as a remedy, we extend the ill-conditioned Hankel matrix until we reach a well-conditioned one. More precisely, we construct

$$H = \begin{bmatrix} H_n & S \\ S^T & H_i \end{bmatrix}$$

such that H is a well-conditioned Hankel matrix, and then continue to use the recursive step in the decomposition theorem to build Hankel matrices of larger dimensions.

5.2 Numerical Examples

All computations reported in this section were carried out using Matlab on a SUN/SPARC workstation or a SiliconGraphics workstation, both with machine precision of order $\mathcal{O}(10^{-16})$. For each set of numerical examples, we generated 100 Hankel matrices H with certain prescribed ill-conditioned leading principal submatrices. The index j, $j = 1, 2, \ldots, 100$, is used as a counter for the 100 matrices. For each H, we solved the linear system

$$Hx = b,$$

where b was chosen such that the vector of all 1's is the exact solution x_{exact}. The relative Euclidean error is defined as

$$\text{relative error} = \frac{\|x_{\text{compt}} - x_{\text{exact}}\|}{\|x_{\text{exact}}\|}.$$

We always plot this relative error versus the index number j.

1. This test set consists of 100 50×50 Hankel matrices with only one ill-conditioned leading principal submatrix. A typical condition number profile of such Hankel matrices is plotted in Figure 1. Figure 2 shows the relative errors generated by the classical Hankel solver without look-ahead (the classical Trench algorithm)[5]. We observe that there is almost no relative accuracy in the computed solution. In contrast, Figure 3 shows the relative errors computed by using the look-ahead Hankel solver with $s_{\max} = 2$.

2. This test set consists of 100 300×300 Hankel matrices with one ill-conditioned leading principal submatrix. The condition number profiles are similar to the one plotted in Figure 1, and hence are not presented here. Figure 4 shows the relative errors computed by using the look-ahead Hankel solver with $s_{\max} = 2$.

3. This test set consists of 100 60×60 Hankel matrices with two or three consecutive ill-conditioned leading principal submatrices. A typical condition number profile is plotted in Figure 5. In Figure 6, we display the relative errors computed by using the look-ahead Hankel solver with $s_{\max} = 4$. We should mention here that, if we choose $s_{\max} = 3$ for this test

[5] Actually, we use the look-ahead algorithm with $s_{\max} = 1$.

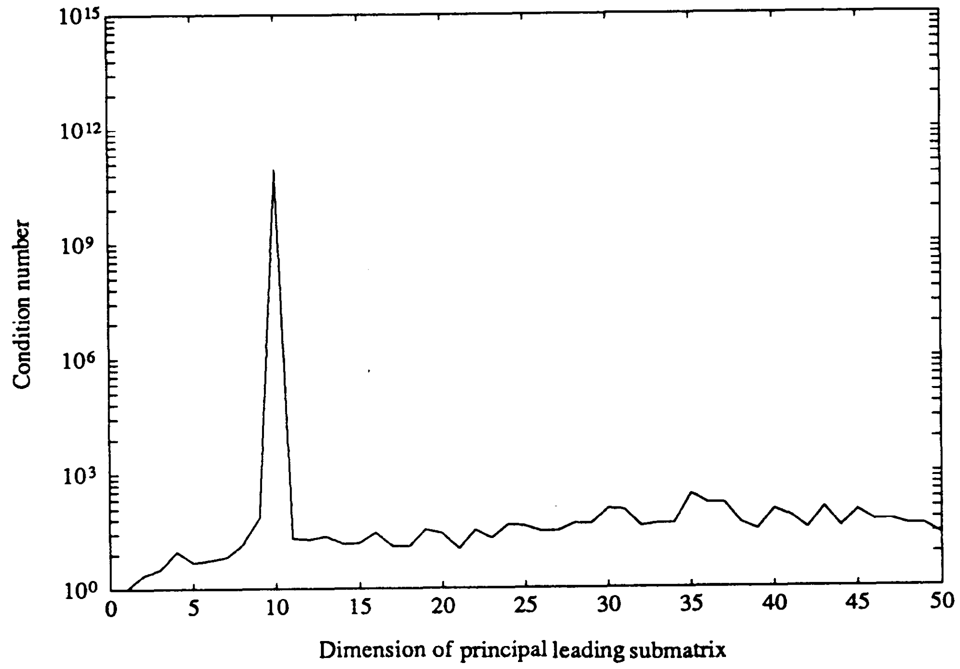

Figure **1:** Condition number profile of a 50 × 50 Hankel matrix

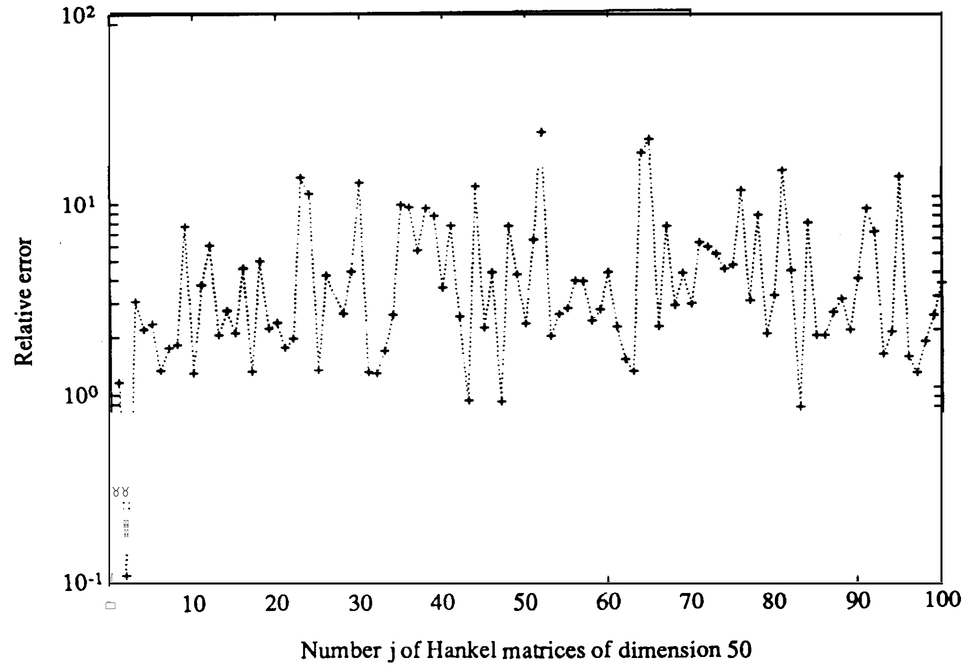

Figure 2: Relative errors for **100 50** x 50 random Hankel matrices with one ill-conditioned submatrix, run without look-ahead

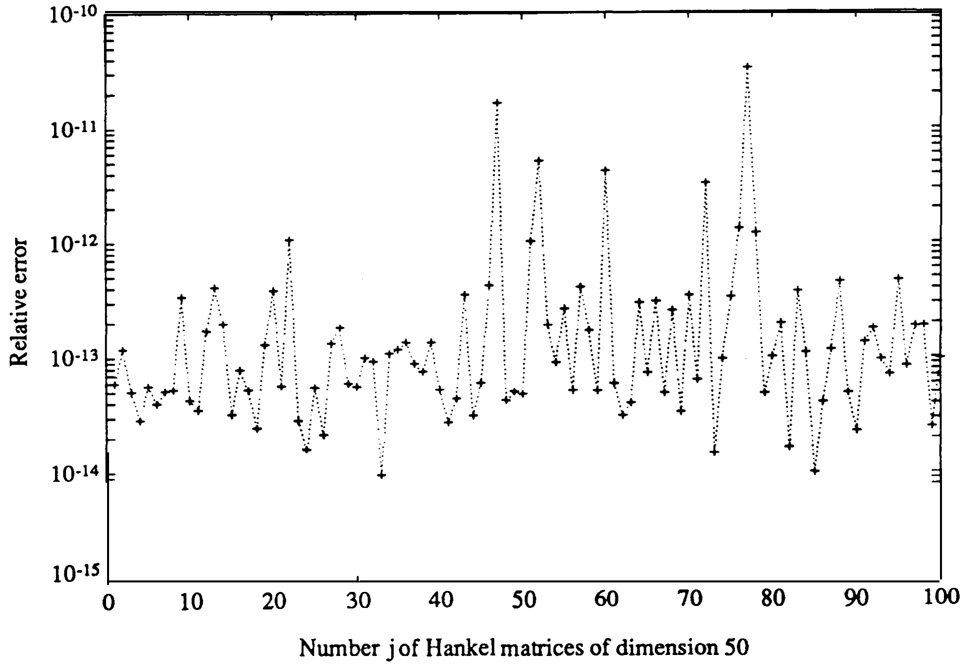

Figure 3: Relative errors for 100 50 x 50 random Hankel matrices with one ill-conditioned submatrix, run with look-ahead

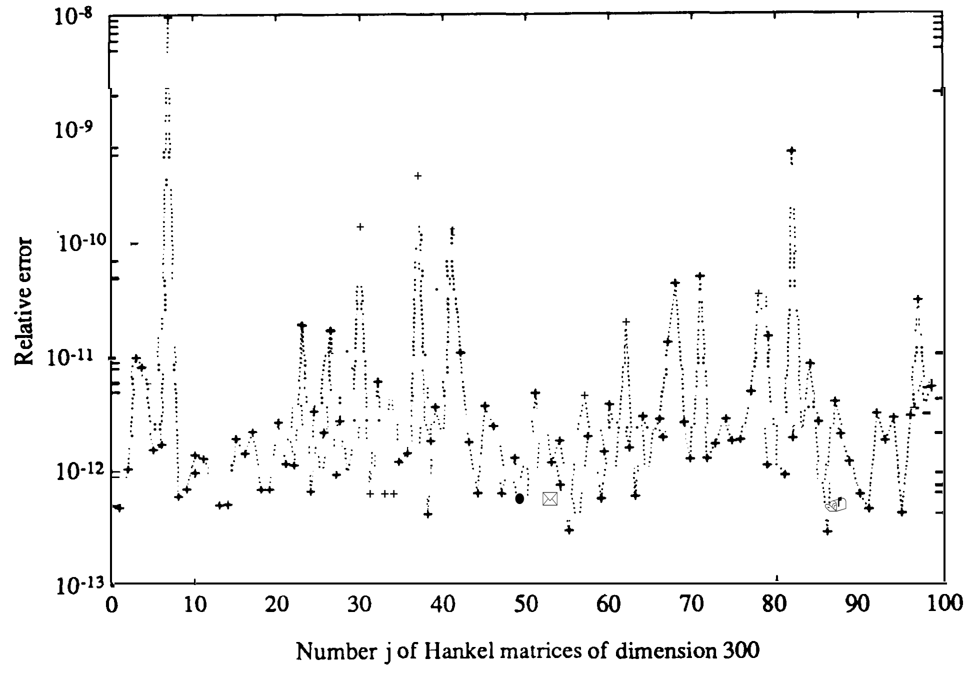

Figure 4: Relative errors for 100 300 x 300 random Hankel matrices with one ill-conditioned submatrix

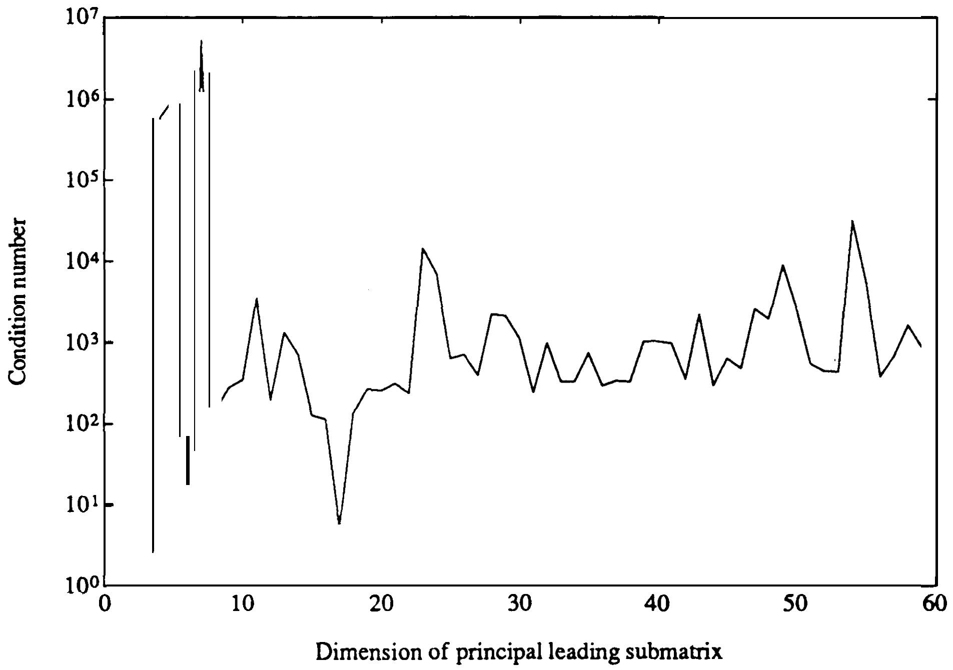

Figure 5: Condition number profile of a 60 × 60 Hankel matrix

set, then we see an obvious deterioration in the accuracy of the computed solutions. This is due to the fact that some of the Hankel matrices have three consecutive ill-conditioned leading principal submatrices. But if we choose $s_{\max} = 5$ in this test set, the accuracy of the solutions will be almost the same as that plotted in Figure 6.

6 Concluding Remarks

We presented a look-ahead Lanczos process for generating formally orthogonal polynomials and gave elementary and self-contained proofs of some of its properties. Based on this polynomial formulation of the look-ahead Lanczos process, we devised a look-ahead algorithm for solving general Hankel systems. The resulting procedure is an extension of the classical Trench algorithm for the strongly regular case. In contrast to other generalizations of Trench's algorithm, which can only skip over exactly singular submatrices, our look-ahead Hankel solver can handle exactly singular as well as ill-conditioned submatrices. We also discussed implementation issues of the look-ahead Hankel algorithm and gave an operation count. It was shown that Trench's algorithm and several of its extensions can be obtained easily as special cases of the look-ahead Hankel solver. Finally, we reported results of numerical experiments, which clearly demonstrate that the look-ahead algorithm generates solutions of Hankel systems with ill-conditioned submatrices nearly to full accuracy. For such systems, the solutions produced by the classical Hankel solver are usually of no accuracy at all.

It remains to give a rigorous stability analysis of the proposed look-ahead Hankel solver.

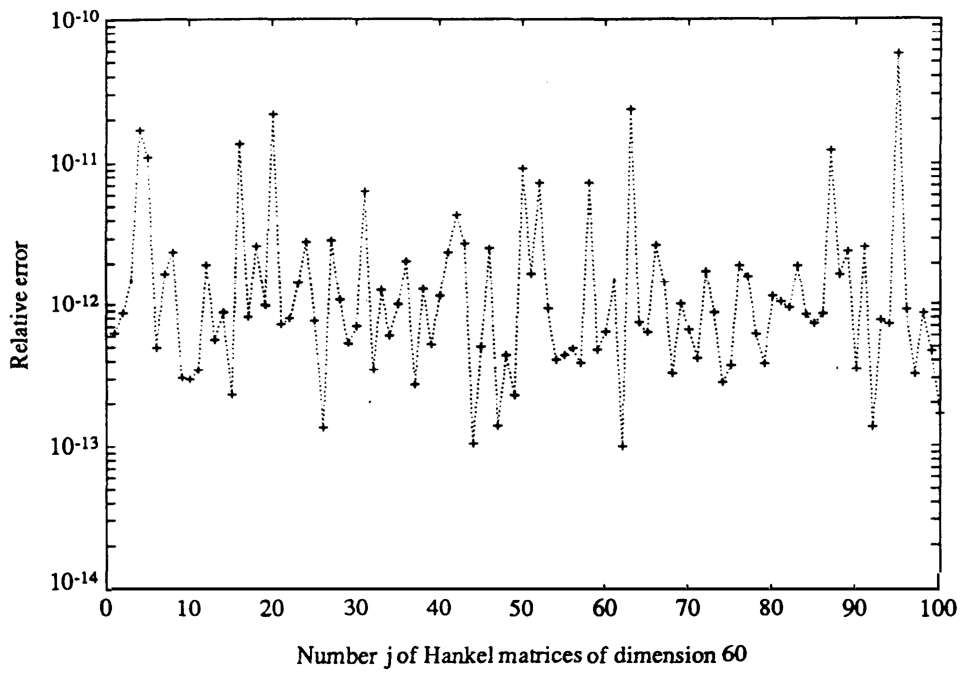

Figure 6: Relative errors for **100 60 $\times$ 60** random Hankel matrices with three ill-conditioned submatrices

This will be the subject of future work.

Also, we remark that similar techniques, based on a Lanczos-type process for formally biorthogonal polynomials, can be used to derive a look-ahead Levinson algorithm for solving general Toeplitz systems. This is described in detail in [11].

Recently, Cabay and Meleshko [4] have proposed an algorithm for stably generating Padé approximants. As a by-product, their procedure can also be used to invert general Hankel matrices. However, the resulting approach is different from the one taken in this paper. In particular, their algorithm for inverting Hankel matrices does not seem to be a direct extension of the classical Hankel solver.

The Hankel algorithms discussed in this paper all involve $\mathcal{O}(n^2)$ work. There are also so-called superfast Hankel solvers (see, e.g., [2, 15]), which require only $\mathcal{O}(n \log^2 n)$ work. However, like Trench's algorithm, these superfast solvers require that the underlying Hankel matrices be strongly regular. It is an open problem whether these algorithms can be extended to numerically stable procedures for solving general Hankel systems.

Another class of fast Hankel solvers, which are essentially different from Trench's algorithm and its extensions, are so-called Schur-like methods (see, e.g., [22, 7]). Unlike Trench's solver and its generalizations, Schur-like Hankel algorithms do not involve inner products of long vectors. On parallel architectures, the computation of such inner products usually represents a bottleneck, and algorithms that are based only on SAXPY operations are preferable. Our polynomial formulation of the look-ahead Lanczos process can also be used to design a Schur-like Hankel solver, which skips over exactly singular as well as ill-conditioned submatrices. The resulting algorithm will be presented in a future report.

Acknowledgment

We would like to thank Susanne Freund, Noël Nachtigal, and Gerhard Starke for their careful reading of parts of this manuscript.

References

[1] Berlekamp, E.R. (1968): Algebraic Coding Theory. McGraw-Hill, New York

[2] Brent, R.P., Gustavson, F.G., Yun, D.Y.Y. (1980): Fast solution of Toeplitz systems of equations and computation of Padé approximants. J. Algorithms 1, 259-295

[3] Bultheel, A. (1987): Laurent Series and their Padé Approximations. Birkhäuser, Basel

[4] Cabay, S., Meleshko, R. (1991): A weakly stable algorithm for Padé approximants and the inversion of Hankel matrices. Preprint

[5] Chihara, T.S. (1978): An Introduction to Orthogonal Polynomials. Gordon and Breach, New York, 1978

[6] Chun, J. (1989): Fast array algorithms for structured matrices. Ph.D. Thesis, Stanford University

[7] Citron, T.K. (1986): Algorithms and architectures for error correcting codes. Ph.D. Thesis, Stanford University

[8] Draux, A. (1983): Polynômes Orthogonaux Formels – Applications. Lecture Notes in Mathematics, Vol. 974. Springer, Berlin Heidelberg New York

[9] Freund, R.W., Golub, G.H., Nachtigal, N.M. (1991): Iterative solution of linear systems. Technical Report 91.21, RIACS, NASA Ames Research Center. Acta Numerica, to appear

[10] Freund, R.W., Gutknecht, M.H., Nachtigal, N.M. (1991): An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. Technical Report 91.09, RIACS, NASA Ames Research Center. SIAM J. Sci. Stat. Comput., to appear

[11] Freund, R.W., Zha, H. (1991): Formally biorthogonal polynomials and a look-ahead Levinson algorithm for general Toeplitz systems. Technical Report 91.27, RIACS, NASA Ames Research Center

[12] Golub, G.H., Van Loan, C.F. (1989): Matrix Computations, Second Edition. The Johns Hopkins University Press, Baltimore

[13] Gragg, W.B. (1972): The Padé table and its relation to certain algorithms of numerical analysis. SIAM Review **14, 1-62**

[14] Gragg, W.B. **(1974):** Matrix interpretations and applications of the continued fraction algorithm. Rocky Mountain J. Math. 4, 213-225

[15] Gragg, W.B., Gust avson, F.G., Warner, D.D., Yun, D.Y.Y. (1982): On fast computation of superdiagonal Padé fractions. Math. Programming Stud. 18, 39-42

[16] Gragg, W.B., Lindquist, A. (1983): On the partial realization problem. Linear Algebra Appl. 50, 277-319

[17] Gutknecht, M.H. (1992): A completed theory of the unsymmetric Lanczos process and related algorithms, Part I. SIAM J. Matrix Anal. Appl. **13**, to appear

[18] Gutknecht, M.H. (1990): A completed theory of the unsymmetric Lanczos process and related algorithms, Part II. IPS Research Report No. 90-16, ETH Zürich

[19] Heinig, G., Rost, K. (1984): Algebraic Methods for Toeplitz-like Matrices and Operators. Birkhäuser, Base1

[20] Jonckheere, E., Ma, C. (1989): A simple Hankel interpretation of the Berlekamp-Massey algorithm. Linear Algebra Appl. 125, 65-76

[21] Kailath, T. (1980): Linear Systems. Prentice-Hall, Englewood Cliffs

[22] Kalman, R.E. (1979): On partial realizations, transfer functions, and canonical forms. Acta Polytech. Scand. Math. Comput. Sci. Ser. 31, 9-32

[23] Kalman, R.E., Falb, P.L., Arbib, M.A. (1969): Topics in Mathematical System Theory. McGraw-Hill, New York

[24] Kung, S.-Y. (1977): Multivariable and multidimensional systems: analysis and design. Ph.D. Dissertation, Stanford University

[25] Lanczos, C. (1950): An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Natl. Bur. Stand. 45, 255-282

[26] Lev-Ari, H., Kailath, T. (1986): Triangular factorization of structured Hermitian matrices. In: I. Schur Methods in Operator Theory and Signal Processing (I. Gohberg, ed.). Operator Theory Adv. Appl. **18,** pp. 301-324, Birkhauser, Base1

[27] Massey, J.L. (1969): Shift-register synthesis and BCH decoding. IEEE Trans. Inform. Theory **IT-15,** 122-127

[28] Parlett, B.N., Taylor, D.R., Liu, Z.A. (1985): A look-ahead Lanczos algorithm for unsymmetric matrices. Math. Comp. 44, 105-124

[29] Phillips, J.L. (1971): The triangular decomposition of Hankel matrices. Math. Comp. 25, 599-602

[30] Rissanen, J. (1973/74): Solution of linear equations with Hankel and Toeplitz matrices. Numer. Math. 22, 361-366

[31] Trench, W. (1965): An algorithm for the inversion of finite Hankel matrices. J. Soc. Indust. Appl. Math. **13**, 1102-1107