A stabilized superfast solver for indefinite Hankel systems

Marc Van Barel *, Peter Kravanja

K.U. Leuven, Department of Computerwetenschappen, Celestijnenlaan 200 A, B-3001 Leuven (Heverlee), Belgium
Received 15 September 1997; accepted 20 April 1998
Submitted by A.H. Sayed

Abstract

We present a stabilized superfast solver for indefinite Hankel systems whose size is a power of 2. The Hankel system is transformed into a Loewner system, which is solved by using an inversion formula for Loewner matrices. This explicit formula for the inverse of a Loewner matrix contains certain parameters that are computed by solving two linearized rational interpolation problems on the unit circle. The heart of our Hankel solver is a superfast algorithm to solve these interpolation problems. This algorithm is stabilized via pivoting, iterative improvement, and by giving the so-called “difficult” interpolation points an adequate treatment. We have implemented our algorithm in Fortran 90. Numerical examples illustrate the effectiveness of our approach. © 1998 Elsevier Science Inc. All rights reserved.

AMS classification: Primary 65F05; Secondary 47B35, 15A09

1. Introduction

Suppose $n$ is a power of 2. Let $H = H_n := [h_{k+l}]_{k,l=0}^{n-1}$ be a regular $n \times n$ complex Hankel matrix, and let $b \in \mathbb{C}^n$. We consider the problem of computing $x := H^{-1}b$.

The $n \times n$ exchange matrix $E := [\delta_{k,n-1-l}]_{k,l=0}^{n-1}$ transforms the Hankel system $Hx = b$ into the Toeplitz system $Tx = b$ where $T := HE$ and $x := Ex$. Algorithms for solving a Hankel or a Toeplitz system exploit the structure and require less

* Corresponding author. Fax: +32 16 327996; E-mail: marc.vanbarel@cs.kuleuven.ac.be.
arithmetic operations, compared to $O(n^3)$ for general linear systems. The so-called “fast” algorithms require $O(n^2)$ arithmetic operations, while the so-called “superfast” ones need only $O(n \log^2 n)$ operations. The latter use a divide and conquer strategy. The flow of classical (fast or superfast) methods is determined by the exact singularity of the leading principal submatrices of $H$ or $T$. The fast methods compute the solutions corresponding to successive nonsingular leading principal submatrices (sections). However, in finite-precision arithmetic not only singular but also ill-conditioned sections should be avoided. The algorithms that have been developed for this purpose are called “look-ahead” algorithms. They look ahead from one well-conditioned section to the next one and jump over the ill-conditioned sections that lie in between. For (scalar) Hankel systems several look-ahead algorithms have been designed [10,17,7]. For (scalar) Toeplitz systems we refer to [12,11,16,15,26,27,35,19,25,24].

For the block Toeplitz and the block Hankel case, the reader may consult [48]. In [18] a look-ahead Schur algorithm was designed for Hermitian block Toeplitz matrices. Several high performance algorithms for Toeplitz and block Toeplitz matrices are described in [20], including two look-ahead Schur algorithms for symmetric indefinite block Toeplitz matrices. In [43] a look-ahead block Schur algorithm for Toeplitz-like matrices was presented.

It is difficult to design a good look-ahead strategy. Only recently, a completely different approach was considered, in which the (block) Hankel or Toeplitz matrix is transformed into a generalized Cauchy matrix. Such a matrix can be factorized via pivoting without destroying the structure. Hence, the Cauchy system can be solved without using look-ahead and the solution can be transformed back to the solution of the original system. We refer the interested reader to [22,36,31,33,32,20] and the references cited therein. In [37] we presented such a transformation approach based on an inversion formula for Loewner matrices. In [38] we generalized this to the block case. This resulted in a stabilized fast solver for indefinite Hankel and block Hankel systems. Based upon the fast algorithm of [37], we will use a divide and conquer approach together with stabilizing techniques to derive a stabilized superfast solver.

The first superfast algorithms were designed by Sugiyama et al. [45], Bitmead and Anderson [6], Brent, Gustavson and Yun [8] and Morf [41]. More recent algorithms can be found in [9,13,23,42,39,44,2-4]. For the block case we refer to [40,21]. The main disadvantage of these algorithms is that they cannot handle nearly singular leading principal submatrices. To overcome this problem, Gutknecht [26] and Gutknecht and Hochbruck [28,27] developed an algorithm that combines the look-ahead idea with divide and conquer techniques. Because in most practical problems the look-ahead step will be small compared to the order of the system that is to be solved, the algorithm is generically superfast.

This paper is organized as follows. In Section 2 we will summarize the results presented in [37]. We will discuss how the Hankel system $Hx = b$ can be transformed into a Loewner system $Lx' = b'$. An explicit formula for $L^{-1}$
enables us to calculate $x'$ as $L^{-1}b'$. This inversion formula for Loewner matrices involves certain parameters that can be computed by solving two rational interpolation problems. In [37] we presented a fast algorithm to solve these interpolation problems. This led to a fast Hankel solver. In Section 3 we will use a divide and conquer approach to obtain a superfast algorithm for rational interpolation. The subproblems (at the lowest level) are solved via our fast algorithm. The superfast algorithm can be stabilized in two ways:

- Each subproblem (at the lowest level) consists of a set of interpolation conditions that are to be satisfied. Our fast algorithm constructs a solution iteratively by adding interpolation points one by one. Pivoting is used to enhance the numerical stability. Some interpolation points may have residuals with respect to the subproblem solution that are very small: the subproblem may be close to degenerate. In this case these interpolation points, which will be called "bad" or "difficult" points, will not be added at the subproblem level. They will be put aside and added at the very end, after the divide and conquer procedure has finished, by using the fast interpolation algorithm.

- The solutions of the interpolation problems at a specified subdivision level are refined iteratively by using an inversion formula for coupled Vandermonde matrices [30]. The computed solution of the Hankel system can be refined iteratively by using the inversion formula for the Loewner matrix (cf. [37]).

In this way we obtain a stabilized generically superfast solver for indefinite Hankel systems, i.e. superfast in case the number of difficult points is small and fast otherwise. Numerical examples will illustrate the effectiveness of our approach.

2. Transformation into a Loewner system

The fast Hankel solver that we presented in [38] consisted of the following components:

1. The Hankel system $Hx = b$ is transformed into a Loewner system $Lx' = b'$. The solution $x'$ is transformed back into $x$.
2. An explicit formula for $L^{-1}$ is available. The solution $x'$ is found as the product $L^{-1}b'$.
3. This inversion formula for $L$ contains certain parameters. These are calculated by solving two rational interpolation problems.

By considering Loewner matrices based on roots of unity we can use fast Fourier transform (FFTs) to perform parts 1 and 2 with arithmetic complexity $O(n \log n)$ flops. In [38] we presented an $O(n^2)$ algorithm to solve the interpolation problems of part 3. Hence, our Hankel solver had an overall complexity of $O(n^2)$ flops. In this paper we will show how our fast interpolation algorithm
can be replaced by a superfast algorithm. This immediately leads to a superfast Hankel solver, although only for systems whose size is a power of 2. In this section we will summarize parts 1 and 2. We refer the reader to [38] for further background, proofs, and implementation details.

Let $y_1, \ldots, y_n, z_1, \ldots, z_n$ be $2n$ mutually distinct complex numbers, and define $y := (y_1, \ldots, y_n)$ and $z := (z_1, \ldots, z_n)$. Let $\mathcal{L}(y, z)$ be the class of matrices

$$
\mathcal{L}(y, z) := \left\{ \begin{bmatrix}
\frac{c_i - d_i}{y_i - z_i} \\
\end{bmatrix}^{n}_{k,l=1} \mid c_1, \ldots, c_n, d_1, \ldots, d_n \in \mathbb{C} \right\}.
$$

The elements of $\mathcal{L}(y, z)$ are called Loewner matrices.

The set $\mathcal{L}(y, z)$ is a linear space over $\mathbb{C}$, and a subspace of the linear space of all the $n \times n$ complex matrices. Since addition of a constant to all the $2n$ parameters $c_k, d_l$ leads to the same Loewner matrix, its dimension is $2n - 1$. The set of all the $n \times n$ complex Hankel matrices also forms a linear subspace of dimension $2n - 1$. Hankel and Loewner matrices are even more closely related. According to Fiedler [14] every Hankel matrix can be transformed into a Loewner matrix, and vice versa. Before we can formulate Fiedler’s theorem we have to deal with some preliminaries concerning Vandermonde matrices.

Let $t_1, \ldots, t_n$ be $n$ complex numbers and define $t := (t_1, \ldots, t_n)$. The Vandermonde matrix with nodes $t_1, \ldots, t_n$ is given by

$$
V(t) = V(t_1, \ldots, t_n) := \begin{bmatrix}
1 & t_1 & \cdots & t_1^{n-1} \\
1 & t_2 & \cdots & t_2^{n-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & t_n & \cdots & t_n^{n-1}
\end{bmatrix}.
$$

Let $f_t(z)$ be the monic polynomial of degree $n$ that has zeros $t_1, \ldots, t_n$,

$$
f_t(z) := (z - t_1) \cdots (z - t_n),
$$

and define

$$
f_{t,j}(z) := \prod_{k \neq j} (z - t_k), \quad j = 1, \ldots, n.
$$

Note that $f_{t,j}(z)$ is a monic polynomial of degree $n - 1$ for $j = 1, \ldots, n$. Define the $n \times n$ matrix $W(t)$ by the equation

$$
\begin{bmatrix}
f_{t,1}(z) \\
f_{t,2}(z) \\
\vdots \\
f_{t,n}(z)
\end{bmatrix} = W(t) \begin{bmatrix}
1 \\
z \\
\vdots \\
z^{n-1}
\end{bmatrix}.
$$

This means that the $j$th row of $W(t)$ contains the coefficients of $f_{t,j}(z)$ when written in terms of the standard monomial basis $(1, z, \ldots, z^{n-1})$. Then
\[ W(t)\begin{bmatrix} V(t) \end{bmatrix}^T = \text{diag}(f_{t,1}(t), \ldots, f_{t,n}(t)) =: D(t). \] (2)

The Vandermonde matrix \( V(t) \) is regular if and only if its nodes \( t_1, \ldots, t_n \) are mutually distinct. In that case Eq. (2) implies that \( W(t) \) is regular.

Let \( V(y, z) \) be the \( 2n \times 2n \) Vandermonde matrix with nodes \( y_1, \ldots, y_n \) and \( z_1, \ldots, z_n \), and similarly for \( W(y, z) \).

**Theorem 1** (Fiedler). The matrix \( L := W(y)H[W(z)]^T \) is a Loewner matrix in \( \mathcal{L}(y, z) \) whose parameters \( c_1, \ldots, c_n, d_1, \ldots, d_n \) are given by (up to an arbitrary additive constant \( \zeta \in \mathbb{C} \))

\[
\begin{bmatrix} c_1 \\ \vdots \\ c_n \\ d_1 \\ \vdots \\ d_n \end{bmatrix} = W(y, z) \begin{bmatrix} h_0 \\ \vdots \\ h_1 \\ \vdots \\ h_{2n-2} \\ \zeta \end{bmatrix}.
\]

**Proof.** See [14], Theorem 12. \( \Box \)

Note that \( L \) is regular.

A judicious choice of the points \( y \) and \( z \) enables us to transform the Hankel system \( Hx = b \) into a Loewner system \( Lx' = b' \) in \( O(n \log n) \) flops. Let \( \omega := \exp(2\pi i/n) \) and suppose from now on that \( y_k = \omega^{k-1} \) for \( k = 1, \ldots, n \). That is, let \( y = (1, \omega, \ldots, \omega^{n-1}) \). Let \( \zeta := \exp(n\pi i/n) \) and suppose from now on that \( z_k = \zeta y_k \) for \( k = 1, \ldots, n \). That is, let \( z = (\zeta, \zeta^2, \ldots, \zeta^{n-1}) \).

Let \( \Omega_n \) be the \( n \times n \) Fourier matrix,

\[ \Omega_n := \frac{1}{\sqrt{n}} V(1, \omega, \ldots, \omega^{n-1}). \] (3)

Matrix vector products involving \( \Omega_n \) (\( \Omega_n^H \)) amount to \( a(n) \) (inverse) discrete Fourier transform (DFT) and can be evaluated via the celebrated (inverse) FFT in \( O(n \log n) \) flops. Finally, let \( D_{n, \omega} \) and \( D_{n, \zeta} \) be the \( n \times n \) diagonal matrices

\[ D_{n, \omega} := \text{diag}(1, \omega, \ldots, \omega^{n-1}) \quad \text{and} \quad D_{n, \zeta} := \text{diag}(1, \zeta, \ldots, \zeta^{n-1}). \]

**Theorem 2** ([38]). The solution to the Hankel system \( Hx = b \) is given by

\[ x = \sqrt{n\zeta^{n-1}} \text{adj}_{\zeta} \Omega_n^H D_{n, \omega} \chi' \],
where \( x' := L^{-1} b' \) with \( b' := \sqrt{nD_{n,0}} \Omega_{n} b \).

In other words, the transformations \( b \rightarrow b' \) and \( x' \rightarrow x \) can be done via FFTs and diagonal scalings. The same holds for the calculation of the parameters \( c_k, d_l \) of \( L \). We omit the details.

One can easily show that \( p_{2}(L) = p_{2}(H) \): the spectral condition number of \( H \) is left unchanged.

We now consider the inversion formula for \( L \) and the connection with rational interpolation.

**Theorem 3.** Let \( p_k, u_k, \tilde{p}_k \) and \( \check{u}_k \) for \( k = 1, \ldots, n \) be defined by the equations

\[
\begin{align*}
[p_1 \ldots p_n] L &= [1 \ldots 1], \\
[u_1 \ldots u_n] L &= [d_1 \ldots d_n],
\end{align*}
\]

Then the inverse of \( L \) is given by

\[
L^{-1} = \left( \frac{\check{u}_k p_l - \tilde{p}_k u_l}{f(z)} \right)^n_{k,l=1}.
\]

**Proof.** See [49], Theorem 2.2. \( \square \)

A Loewner matrix is a special kind of Cauchy-like matrix. The inversion formula of the previous theorem can therefore be seen as a special case of the formula given in the book of Heinig and Rost [34], p. 161.

The parameters that appear in the inversion formula (4) can be computed by solving the following two (linearized) rational interpolation problems.

**Theorem 4.** Define

\[
\begin{align*}
\mu(z) := f(z) - \sum_{k=1}^{n} u_k f_{y,k}(z), \\
\nu(z) := -\sum_{k=1}^{n} c_k u_k f_{y,k}(z),
\end{align*}
\]
\[ \hat{u}(z) := f(z) - \sum_{k=1}^{n} \hat{u}_k f_{z,k}(z), \]

\[ \hat{v}(z) := -\sum_{k=1}^{n} d_k \hat{v}_k f_{z,k}(z). \]

Then the polynomial pair \((u(z), u(z))\) is the only pair such that
1. \(u(z) \in \mathbb{C}[z]\) and \(\deg u(z) = n\), \(v(z) \in \mathbb{C}[z]\) and \(\deg v(z) < n\), \(u(z)\) is monic,
2. \(v(y_k) = c_k u(y_k)\) and \(v(z_k) = d_k u(z_k)\) for \(k = 1, \ldots, n\).

The polynomial pair \((\hat{v}(z), \hat{u}(z))\) satisfies exactly the same properties and thus \(v(z) \equiv \hat{v}(z)\) and \(u(z) \equiv \hat{u}(z)\).

**Proof.** See [49], Theorem 3.1. Compare with [50], Theorem 2.1. \(\square\)

**Theorem 5.** Define

\[ p(z) := \sum_{k=1}^{n} p_k f_{y,k}(z), \]

\[ q(z) := f(y(z) + \sum_{k=1}^{n} c_k p_k f_{y,k}(z), \]

\[ \hat{p}(z) := \sum_{k=1}^{n} \hat{p}_k f_{z,k}(z), \]

\[ \hat{q}(z) := f(z) + \sum_{k=1}^{n} d_k \hat{p}_k f_{z,k}(z). \]

Then the polynomial pair \((q(z), p(z))\) is the only pair such that
1. \(q(z) \in \mathbb{C}[z]\) and \(\deg q(z) = n\), \(p(z) \in \mathbb{C}[z]\) and \(\deg p(z) < n\), \(q(z)\) is monic,
2. \(q(y_k) = c_k p(y_k)\) and \(q(z_k) = d_k p(z_k)\) for \(k = 1, \ldots, n\).

The polynomial pair \((\hat{q}(z), \hat{p}(z))\) satisfies exactly the same properties and thus \(q(z) \equiv \hat{q}(z)\) and \(p(z) \equiv \hat{p}(z)\).

**Proof.** See [49], Theorem 3.2. \(\square\)

Note that if \(p(y_k)\) and \(p(z_k)\), \(k = 1, \ldots, n\), are different from zero, then the rational function \(q(z)/p(z)\) satisfies the proper rational interpolation conditions \(q(y_k)/p(y_k) = c_k\) and \(q(z_k)/p(z_k) = d_k\), \(k = 1, \ldots, n\). Similarly, if \(u(y_k)\) and \(u(z_k)\), \(k = 1, \ldots, n\), are different from zero, then the rational function \(v(z)/u(z)\) satisfies the proper rational interpolation conditions \(v(y_k)/u(y_k) = c_k\) and \(v(z_k)/u(z_k) = d_k\), \(k = 1, \ldots, n\). The rational functions \(q(z)/p(z)\) and \(v(z)/u(z)\) are different because their degree structure is different.

An easy calculation reveals the following connections with the parameters that appear in the inversion formula (4):
As the points \( y_1, \ldots, y_n, z_1, \ldots, z_n \) are (up to a permutation) equal to the \( 2n \)th roots of unity \( 1, \zeta, \ldots, \zeta^{2n-1} \), we can use the FFT to evaluate the polynomials \( u(z) \) and \( p(z) \) at these Points.

Once these inversion parameters have been computed, the matrix–vector product \( x' = L^{-1}b' \) can be calculated in \( O(n \log n) \) flops. Indeed, \( L^{-1} \) can be written as

\[
L^{-1} = \operatorname{diag}(\tilde{p}_1, \ldots, \tilde{p}_n) C \operatorname{diag}(u_1, \ldots, u_n) - \operatorname{diag}(\tilde{u}_1, \ldots, \tilde{u}_n) C \operatorname{diag}(p_1, \ldots, p_n)
\]

if \( C \) is given by the Cauchy matrix

\[
C := \left[ \frac{1}{z_{k} - y_{j}} \right]_{k,j=1}^{n}.
\]

As we have shown in [38], the product of \( C \) with a vector in \( \mathbb{C}^n \) can be evaluated in \( O(n \log n) \) flops, and thus, because of Eq. (5), the same holds for the product \( L^{-1}b' \).

### 3. A superfast algorithm for rational interpolation

Let \( s_k := y_k \) and \( s_{n+k} := z_k \) for \( k = 1, \ldots, n \). Define the column vectors \( f_1, \ldots, f_{2n} \in \mathbb{C}^{2 \times 1} \) as

\[
f_k := \begin{bmatrix} 1 \\ -c_k \end{bmatrix}, \quad f_{n+k} := \begin{bmatrix} 1 \\ -d_k \end{bmatrix}, \quad k = 1, \ldots, n.
\]

Consider the set \( \mathcal{S} \) of all the vector polynomials \( w(z) \in \mathbb{C}[z]^{2 \times 1} \) that satisfy the interpolation conditions

\[
f_k^T w(s_k) = 0, \quad k = 1, \ldots, 2n.
\]

If \( w(z) \in \mathbb{C}[z]^{2 \times 1} \) is an arbitrary vector polynomial, then the left hand side of Eq. (6) is called the residual with respect to \( w \) at the interpolation point \( s_k \). Every element of \( \mathcal{S} \) is thus such that the residuals at all the interpolation points are equal to zero.
The set \( \mathcal{P} \) forms a submodule of the \( \mathbb{C}[z] \)-module \( \mathbb{C}[z]^{2 \times 1} \). A basis for \( \mathcal{P} \) always consists of two elements, i.e., the dimension of \( \mathcal{P} \) is equal to two [46], Theorem 3.1. Let \( \{B_1(z), B_2(z)\} \) be a basis for \( \mathcal{P} \). Then every element \( w(z) \in \mathcal{P} \) can be written in a unique way as

\[
 w(z) = \alpha_1(z)B_1(z) + \alpha_2(z)B_2(z)
\]

with \( \alpha_1(z), \alpha_2(z) \in \mathbb{C}[z] \). The matrix polynomial \( B(z) := [B_1(z) \ B_2(z)] \in \mathbb{C}[z]^{2 \times 2} \) is called a basis matrix. Basis matrices can be characterized as follows.

**Theorem 6.** A matrix polynomial \( C(z) = [C_1(z) \ C_2(z)] \in \mathbb{C}[z]^{2 \times 2} \) is a basis matrix if and only if \( C_1(z), C_2(z) \in \mathcal{P} \) and \( \deg \det C(z) = 2n \).

**Proof.** This follows from [46], Theorem 4.1. \( \square \)

Theorems 4 and 5 immediately imply that

\[
 B^*(z) := \begin{bmatrix} q(z) & v(z) \\ p(z) & u(z) \end{bmatrix}
\]

is a basis matrix. This matrix polynomial is the unique solution to the following problem: Given the interpolation data

\[
 \{(s_k, f_k): k = 1, \ldots, 2n\}, \]

find a monic \( 2 \times 2 \) matrix polynomial \( B(z) \) of degree \( n \) that satisfies the interpolation conditions

\[
 f_k^T B(s_k) = [0 \ 0], \quad k = 1, \ldots, 2n. \tag{7}
\]

The \( \mathcal{O}(n^2) \) algorithm that we presented in [38] calculates a \( 2 \times 2 \) matrix polynomial \( B(z) \) of degree \( n \) that satisfies Eq. (7). This matrix polynomial is not necessarily monic. However, the determinant of its highest degree coefficient is equal to one, and we have shown that this ensures that the correct value of \( L^{-1} \) is obtained if the inversion parameters are computed from the polynomials that appear in the second row of \( B(z) \). The algorithm takes as input two column vectors of residuals (corresponding to the initialization of \( B(z) \)) and proceeds to zero these vectors. The residuals are updated after each step of the algorithm. Pivoting is used to enhance the numerical stability: the interpolation point that has the largest residual is processed first.

Within the submodule \( \mathcal{P} \) we want to be able to consider solutions \( w(z) \) that satisfy additional conditions concerning their degree-structure. To describe the degree-structure of a vector polynomial, we use the concept of \( \tau \)-degree [46]. Let \( \tau \in \mathbb{Z} \). The \( \tau \)-degree of a vector polynomial \( w(z) = [w_1(z) \ w_2(z)]^T \in \mathbb{C}[z]^{2 \times 1} \) is defined as a generalization of the classical degree:

\[
 \tau \text{-deg } w(z) := \max \{ \deg w_1(z), \deg w_2(z) - \tau \},
\]
with $\tau$-deg $0 := -\infty$. The $\tau$-highest degree coefficient of a vector polynomial $[w_1(z) \, w_2(z)]^T$ with $\tau$-degree $\delta$ is defined as the vector $[\omega_1 \, \omega_2]^T$ with $\omega_1$ the coefficient of $z^\delta$ in $w_1(z)$ and $\omega_2$ the coefficient of $z^{\delta+1}$ in $w_2(z)$. A set of vector polynomials in $\mathbb{C}[z]^{2\times 1}$ is called $\tau$-reduced if the $\tau$-highest degree coefficients are linearly independent. Every basis of $\mathcal{F}$ can be transformed into a $\tau$-reduced one. For details, we refer to [46]. Once we have a basis in $\tau$-reduced form, it is easy to parametrize all elements of $\mathcal{F}$ with an upper bound on the $\tau$-degree.

**Theorem 7.** Let $\{B_1(z), B_2(z)\}$ be a $\tau$-reduced basis for $\mathcal{F}$. Define $\delta_1 := \tau$-deg $B_1(z)$ and $\delta_2 := \tau$-deg $B_2(z)$. Then every element $w(z) \in \mathcal{F}$ having $\tau$-degree $\leq \delta$ can be written in a unique way as

$$w(z) = \alpha_1(z)B_1(z) + \alpha_2(z)B_2(z)$$

with $\alpha_1(z), \alpha_2(z) \in \mathbb{C}[z]$, $\deg \alpha_1(z) \leq \delta - \delta_1$ and $\deg \alpha_2(z) \leq \delta - \delta_2$.

**Proof.** See [46], Theorem 3.2.

**Theorem 8.** Suppose $K$ is a positive integer. Let $\sigma_1, \ldots, \sigma_K \in \mathbb{C}$ be mutually distinct and let $\phi_1, \ldots, \phi_K \in \mathbb{C}^{2\times 1}$. Suppose that $\phi_k \neq [0 \, 0]^T$ for $k = 1, \ldots, K$. Let $1 \leq \kappa \leq K$. Let $\tau_k \in \mathbb{Z}$. Suppose $B_k(z) \in \mathbb{C}[z]^{2\times 2}$ is a $\tau_k$-reduced basis matrix with basis vectors having $\tau_k$-degree $\delta_1$ and $\delta_2$ respectively, corresponding to the interpolation data

$$\{(\sigma_k, \phi_k): k = 1, \ldots, K\}.$$

Let $\tau_{\kappa-K} := \delta_1 - \delta_2$. Let $B_{\kappa-K}(z) \in \mathbb{C}[z]^{2\times 2}$ be a $\tau_{\kappa-K}$-reduced basis matrix corresponding to the interpolation data

$$\{(\sigma_k, B_k^T(\sigma_k)\phi_k): k = \kappa + 1, \ldots, K\}.$$

Then $B_K(z) := B_n(z)B_{\kappa-K}(z)$ is a $\tau_k$-reduced basis matrix corresponding to the interpolation data

$$\{(\sigma_k, \phi_k): k = 1, \ldots, K\}.$$

**Proof.** See [47], Theorem 3.

The following algorithm implements this theorem. We start with the $(2n)$th roots of unity as interpolation points. They are split into the $n$th roots of unity $y$ and the rotated $n$th roots of unity $z$. The fact that we are dealing with (rotated) roots of unity enables us to do all polynomial evaluations and multiplica-
tions via FFTs (and diagonal scalings). As $n$ is a power of 2, this process can be repeated. At the lowest level the interpolation problems are solved by our fast solver [38].

**Recursive function**

\[ B(z) \leftarrow \text{RecRatInt}(s, L_s, R_s, N, \tau) \]

- $\tau \in \mathbb{Z}$
- $N = 2^{p+1}$ for some $p \in \mathbb{N}$: the number of interpolation conditions
- $s \in \mathbb{C}^{N \times 1}$: the (mutually distinct) interpolation points
- $L_s, R_s \in \mathbb{C}^{N \times 1}$: the initial left and right residual vectors
- $B(z) \in \mathbb{C}[z]^{2 \times 2}$: a $\tau$-reduced basis matrix corresponding to the given interpolation data

if $N > 2^{\text{limit}}$ then

\[ [s_1, L_{s_1}, R_{s_1}, s_2, L_{s_2}, R_{s_2}] \leftarrow \text{SPLIT}(s, L_s, R_s) \]

\[ [B_1(z)] \leftarrow \text{RecRatInt}(s_1, L_{s_1}, R_{s_1}, N/2, \tau) \]

for $k = 1(1)N/2$

\[ [\tilde{L}_{s_2}(k), \tilde{R}_{s_2}(k)] \leftarrow [L_{s_2}(k), R_{s_2}(k)] \cdot B_1(s_2(k)) \]

end for

\[ \tilde{\tau} \leftarrow \text{the difference between the left and right $\tau$-degrees of $B_1(z)$} \]

\[ [\tilde{B}_2(z)] \leftarrow \text{RecRatInt}(s_2, \tilde{L}_{s_2}, \tilde{R}_{s_2}, N/2, \tilde{\tau}) \]

\[ B(z) \leftarrow B_1(z) \cdot \tilde{B}_2(z) \]

else

\[ [B(z)] \leftarrow \text{RatInt}(s, L_s, R_s, N, \tau) \]

end if

return

As stated above, the recursive function RecRatInt leads to a 0-reduced basis matrix with equal left and right 0-degrees. Hence, all values of $\tilde{\tau}$ will be equal to zero. This will not necessarily be the case anymore if difficult points are found and handled at the very end, to stabilize the algorithm. This is the reason why we have to consider $\tau$-degrees.

Superfast Hankel (Toeplitz) solvers are notoriously unstable when applied to indefinite systems. We will stabilize our algorithm in two ways.

**Difficult points:** During the execution of RatInt all the residuals at interpolation points that may be chosen as pivot elements can be smaller (in modulus) than a certain threshold. By processing these interpolation points the accuracy would decrease. These points are therefore marked as "difficult". They are handled only at the very end, after RecRatInt has finished, via the fast-only algorithm RatInt. If at this stage the corresponding transformed residuals are still small, this indicates that the problem is ill-conditioned. The overall complexity of our algorithm will be $\mathcal{O}(n \log^2 n)$ as long as the number of difficult points is not too large.

**Iterative improvement:** RecRatInt computes a basis matrix $B(z)$ corresponding to the $N$ interpolation points $s$ having initial left and right residual
vectors $L_s$ and $R_s$. If there are no difficult points, $B(z)$ has the following degree-structure:

\[
\begin{bmatrix}
= N/2 & < N/2 \\
< N/2 & = N/2
\end{bmatrix}
\]

In general, however, **RECRATINT** will discover difficult points. If the number of difficult points is equal to $n_{\text{bad}}$, the degree-structure of $B(z)$ is given by

\[
\begin{bmatrix}
= z & < z \\
< \beta & = \beta
\end{bmatrix}
\]

with $z + \beta + n_{\text{bad}} = N$. Therefore $B(z)$ is not only $\tau$-(column)reduced but also row reduced. Its row highest degree coefficient is equal to the identity matrix.

Let us introduce the following notations. The component polynomials of $B(z)$ are denoted as $a(z), b(z), c(z)$ and $d(z),$

\[
B(z) = \begin{bmatrix}
a(z) & c(z) \\
b(z) & d(z)
\end{bmatrix}
\]

Let $s^+$ be a complex column vector containing all the "easy" points, i.e. all the interpolation points except the difficult ones. Let $L_{s^+}$ and $R_{s^+}$ be the corresponding parts of $L_s$ and $R_s$. Note that these vectors have size $z + \beta$. We denote their components by $s^+_i, L_{s^+},$ and $R_{s^+}$ respectively ($i = 1, \ldots, z + \beta$).

Let $p$ be a complex column vector of size $\#p$ with components $p_i, i = 1, \ldots, \#p$. Let $L_p$ and $R_p$ be complex column vectors of size $\#p$. Let $k, l \in \mathbb{N}$. Then $V_k(L_p, p)$ is defined as the scaled Vandermonde matrix

\[V_k(L_p, p) := \text{diag}(L_p \cdot p_i^{j=0}) \quad i=1, \ldots, \#p\]

Note that $V_k(L_p, p)$ is a matrix of size $\#p \times (k + 1)$. The last column of $V_k(L_p, p)$ is denoted as $v_k(L_p, p)$. The **coupled Vandermonde matrix** $V_{k,l}(L_p, R_p, p)$ is defined as

\[V_{k,l}(L_p, R_p, p) := \begin{bmatrix}
V_k(L_p, p) & V_l(R_p, p)
\end{bmatrix}\]

It is a matrix of size $\#p \times (k + l + 2)$.

Let $p(z) \in \mathbb{C}[z]$ be a polynomial and let $\delta \in \mathbb{N}$ be an arbitrary upper bound for its degree. Then the **stacking vector** $\hat{p}_\delta \in \mathbb{C}^{\delta+1}$ of $p(z)$ with respect to $\delta$ is the defined by the equation

\[\begin{bmatrix}
1 & z & \ldots & z^\delta
\end{bmatrix} \hat{p}_\delta = p(z)\]

The vector $\hat{p}_\delta \in \mathbb{C}^{\delta}$ is obtained by deleting the last component of $\hat{p}_\delta$. Let $p_i$ be the coefficient of $z^i$ in $p(z)$ for $i = 0, 1, \ldots$ Of course, $p_i = 0$ if $i > \text{deg } p(z)$.

The stacking vectors $\hat{a}_\delta, \hat{b}_\delta, \hat{c}_\delta$ and $\hat{d}_\delta$ satisfy the following linear system of homogeneous equations:
\[ V_{x,\beta}(L_{s^+}, R_{s^+}, s^+) \begin{bmatrix} \hat{a}_x & \hat{c}_x \\ \hat{b}_\beta & \hat{d}_\beta \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix} \]

or

\[ V_{x-1,\beta-1}(L_{s^+}, R_{s^+}, s^+) \begin{bmatrix} \hat{a}'_x & \hat{c}'_x \\ \hat{b}'_\beta & \hat{d}'_\beta \end{bmatrix} = \begin{bmatrix} -v_x(L_{s^+}, s^+) & -v_\beta(R_{s^+}, s^+) \end{bmatrix}. \quad (8) \]

Note that \( V_{x-1,\beta-1}(L_{s^+}, R_{s^+}, s^+) \) is a square nonsingular matrix. To enhance the accuracy of the computed approximations to \( \hat{a}'_x, \hat{b}'_\beta, \hat{c}'_x \) and \( \hat{d}'_\beta \) we will use iterative refinement based on an inversion formula for this matrix.

The following theorem provides us with the parameters for an inversion formula for coupled Vandermonde matrices.

**Theorem 9.** Let \( V_{x-1,\beta-1}(L_{s^+}, R_{s^+}, s^+) \) be nonsingular and

\[ B(z) = \begin{bmatrix} a(z) & c(z) \\ b(z) & d(z) \end{bmatrix} \]

as defined by Eq. (8). Then the \( 3 \times 2 \) system of linear equations

\[ B(s_i^+)h_i = \begin{bmatrix} a(s_i^+) & c(s_i^+) \\ b(s_i^+) & d(s_i^+) \end{bmatrix} \begin{bmatrix} h_{i,1} \\ h_{i,2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \]

\[ [L_{s_i^+}, R_{s_i^+}]B'(s_i^+)h_i = 1 \]

has a unique solution

\[ h_i = \begin{bmatrix} h_{i,1} \\ h_{i,2} \end{bmatrix} \in \mathbb{C}^{2 \times 1} \quad \text{for every } i \in \{1, \ldots, \alpha + \beta\}. \]

**Proof.** See [30], Proposition 4.2. \( \square \)

**Theorem 10.** Let \( V := V_{x-1,\beta-1}(L_{s^+}, R_{s^+}, s^+) \) be nonsingular and

\[ B(z) = \begin{bmatrix} a(z) & c(z) \\ b(z) & d(z) \end{bmatrix} \]

as defined by Eq. (8). Then the inverse of \( V \) is given by

\[ V^{-1} = WV_{x-1,\beta-1}(H_1, H_2, s^+)^T \]

with

\[ W := \begin{bmatrix} A & C \\ B & D \end{bmatrix} \in \mathbb{C}^{(\alpha+\beta) \times (\alpha+\beta)} \]
where
\[
A := \begin{bmatrix}
a_1 & \ldots & a_x \\
\vdots & \ddots & \vdots \\
a_a & & \ddots \\
\end{bmatrix} \in \mathbb{C}^{\alpha \times \alpha}, \quad B := \begin{bmatrix}
b_1 & \ldots & b_x \\
\vdots & \ddots & \vdots \\
b_\beta & & \ddots \\
\end{bmatrix} \in \mathbb{C}^{\beta \times \beta},
\]
\[
C := \begin{bmatrix}
c_1 & \ldots & c_\beta \\
\vdots & \ddots & \vdots \\
c_a & & \ddots \\
\end{bmatrix} \in \mathbb{C}^{\alpha \times \beta}, \quad D := \begin{bmatrix}
d_1 & \ldots & d_\beta \\
\vdots & \ddots & \vdots \\
d_\beta & & \ddots \\
\end{bmatrix} \in \mathbb{C}^{\beta \times \beta}
\]
are upper triangular Hankel matrices.

**Proof.** See [30], Theorem 4.1. \qed

The interpolation points \(s^+\) together with the difficult points form a set of \(N\) points that are equally spaced on the unit circle. The computation of the inversion parameters \(H_1\) and \(H_2\) can therefore be done in \(O(N \log N)\) flops. Each iterative refinement step based on the inversion formula of Theorem 2 also involves \(O(N \log N)\) flops. Iterative refinement can be applied at one or more intermediate levels.

We are now ready to formulate the stabilized version of our algorithm. Iterative refinement is applied in case the number of interpolation conditions is equal to \(2\text{refimit}\).

**recursive function** \([B(z), s_{\text{bad}}] \leftarrow \text{RECRATINT}(s, L_s, R_s, N, \tau)\)
- \(\tau \in \mathbb{Z}\)
- \(N = 2^{p+1}\) for some \(p \in \mathbb{N}\): the number of interpolation conditions
- \(s \in \mathbb{C}^{N \times 1}\): the (mutually distinct) interpolation points
- \(L_s, R_s \in \mathbb{C}^{N \times 1}\): the initial left and right residual vectors
- \(B(z) \in \mathbb{C}[z]^{2 \times 2}\): a \(\tau\)-reduced basis matrix corresponding to the given interpolation data
- \(s_{\text{bad}}\): a complex column vector containing the difficult interpolation points

**if** \(N > 2\text{limt}\) **then**
- \([s_1, L_{s_1}, R_{s_1}, s_2, L_{s_2}, R_{s_2}] \leftarrow \text{SPLIT}(s, L_s, R_s)\)
- \([B_1(z), s_{\text{bad}, 1}] \leftarrow \text{RECRATINT}(s_1, L_{s_1}, R_{s_1}, N/2, \tau)\)
  **for** \(k = 1(1)N/2\)
  - \([\tilde{L}_{s_2}(k), \tilde{R}_{s_2}(k)] \leftarrow [L_{s_2}(k), R_{s_2}(k)] \cdot B_1(s_2(k))\)
**end for**
- \(\tilde{\tau}\): the difference between the left and right \(\tau\)-degrees of \(B_1(z)\)
- \([B_2(z), s_{\text{bad}, 2}] \leftarrow \text{RECRATINT}(s_{\hat{2}}, L_{s_2}, R_{s_2}, N/2, \tilde{\tau})\)
- \(B(z) \leftarrow B_1(z) \cdot \tilde{B}_2(z)\)
- \(s_{\text{bad}} \leftarrow s_{\text{bad}, 1} \oplus s_{\text{bad}, 2}\)
else
    \([B(z), s_{\text{bad}}] \leftarrow \text{RATINT}(s, L_s, R_s, N, \tau)\)
end if

if \(N = 2^{\text{reflimit}}\) then
    \(s^+ \leftarrow s \ominus s_{\text{bad}}\)
    \([B(z)] \leftarrow \text{ITREF}(B(z), s^+, L_{s^+}, R_{s^+}, N, N_{\text{ref}})\)
end if

return

function \([B(z)] \leftarrow \text{RATINTALL}(s, L_z, R_z, N, \tau)\)
- \(\tau \in \mathbb{Z}\)
- \(N = 2^{p+1}\) for some \(p \in \mathbb{N}\): the number of interpolation conditions
- \(s \in \mathbb{C}^{N \times 1}\): the (mutually distinct) interpolation points
- \(L_z, R_z \in \mathbb{C}^{N \times 1}\): the initial left and right residual vectors
- \(B(z) \in \mathbb{C}[z]^{2 \times 2}\): a \(\tau\)-reduced basis matrix corresponding to the given interpolation data

\([B^+(z), s_{\text{bad}}] \leftarrow \text{RECRATINT}(s, L_z, R_z, N, \tau)\)
\(N_{\text{bad}} \leftarrow \text{SIZE}(s_{\text{bad}})\)
if \(N_{\text{bad}} > 0\) then
    calculate \(L_{\text{bad}}\) and \(R_{\text{bad}}\)
    \(\tau^- \leftarrow\) the difference between the left and right \(\tau\)-degrees of \(B^+(z)\)
    \([B^-(z)] \leftarrow \text{RAINI}(s_{\text{bad}}, L_{\text{bad}}, R_{\text{bad}}, N_{\text{bad}}, \tau^-)\)
end if

return

4. Numerical examples

We have implemented our Hankel solver in Fortran 90 and in Matlab.\(^1\) In the Fortran version the FFTs are calculated via FFTPACK.

We considered double precision \(n \times n\) real Hankel matrices \(H_n\), whose entries were random uniformly distributed in \([0, 1]\) with \(n = 2^k\) for \(k = 1, \ldots, 17\). Note that \(2^{17} = 131072\). The right hand sides \(b_n \in \mathbb{R}^n\) were calculated such that \(x_n := H_n^{-1}b_n = [1 \ldots 1]^T\). The calculations have been done by an IBM SP2 with machine precision \(\approx 0.2210^{-15}\) in double precision.

Fig. 1 shows the results obtained by our algorithm in case no iterative refinement was applied, and in case up to 15 steps of iterative refinement were applied to enhance the accuracy of the computed solution to the Hankel

\(^1\) We plan to make the Fortran 90 package as well as the Matlab m-files available at http://www.cs.kuleuven.ac.be/~marc/hankel/.
system. As a matrix vector product involving a Hankel matrix amounts to a convolution of two vectors, or, equivalently, the product of two polynomials, the residual \( r_n := b_n - H_n \hat{x}_n \) can be calculated via FFT in \( \mathcal{O}(n \log n) \) flops. This implies that improving an approximation for \( x_n \) iteratively does not add substantially to the \( \mathcal{O}(n \log^2 n) \) complexity of our algorithm. For each value of \( k \), five Hankel matrices were considered. Let \( \hat{x}_n^{(l)} \) be the approximation to the solution \( x_n \) after \( l \) steps of iterative improvement. We made the algorithm stop as soon as the impact of iterative improvement stagnated,

\[
\frac{\| b_n - H_n \hat{x}_n^{(l)} \|_1}{\| b_n \|_1} \geq \frac{1}{2} \frac{\| b_n - H_n \hat{x}_n^{(l-1)} \|_1}{\| b_n \|_1}.
\]

Interpolation problems of size \( \leq 2^6 \) were solved by our fast-only algorithm. We experimented to find an 'optimal' subproblem size. Our algorithm performed equally well with subproblem sizes of \( 2^4, 2^5, 2^7 \) or even \( 2^8 \). Its performance seems to be rather insensitive to the exact subproblem size, as long as this is 'near' \( 2^6 \).

Our next figures represent timings. As on our computer system measurements of very small execution times seem to be rather inaccurate, we limit the \( k \)-axis to that part where the results are meaningful. This is why in the following figures \( k \) does not start at 1 but at a larger value.

Fig. 2 shows the execution time for Gaussian elimination with partial pivoting (GEPP) (these results were calculated via the LAPACK routines CGETRF and CGETRS), our fast algorithm [38] and our superfast algorithm.
Fig. 2. Execution time in seconds.

Fig. 3 presents the results shown in Fig. 2 in a different way. It gives the magnification of the execution time. For each $k$, it tells us by which factor the execution time is to be multiplied if we go from $k - 1$ to $k$.

Fig. 3. Magnification of the execution time.
Fig. 4 shows that our algorithm spends only a small part of its execution time on the transformation from Hankel to Loewner and back. The solution of the rational interpolation problems is (obviously) the most time-consuming part of the algorithm. For each value of $k$, five Hankel matrices were considered.

Fig. 5. Time spent on iterative refinement as percentage of the execution time in case no iterative refinement was applied.
The computed solution to the Hankel system can be refined iteratively. Fig. 5 shows how much execution time was spent on iterative refinement as percentage of the execution time in case no iterative refinement was applied. We considered one, two, three or four steps of iterative refinement. For each value of \( k \) and each number of iterative refinement steps, five Hankel matrices were considered.

5. Conclusion

In this paper, we have designed a generically superfast algorithm for solving Hankel systems. This design is based on several tools that can be used to stabilize the method. Transforming the Hankel system into an interpolation problem allows the use of pivoting at the lowest “fast” level of the algorithm. If the pivots are too small, the corresponding interpolation conditions are handled only after the recursive “superfast” part of the algorithm. The stability can also be enhanced via iterative refinement. This can be applied at (several or even each) intermediate level(s), based on an inversion formula for coupled Vandermonde matrices, or at the very end, based on an inversion formula for Loewner matrices. The implementation and combination of these tools requires the determination of several parameters that have a mutual effect on each other. How to make a good choice for these parameters is still an open problem. In our numerical experiments we have tuned these parameters manually. This led to very satisfactory results.

Acknowledgements

The authors thank G. Heinig for providing them with Ref. [30]. This research was partially supported by the Fund for Scientific Research-Flanders (FWO-V), project “Orthogonal system and their applications”, grant #G.0278.97.

References

