Look-ahead procedures for Lanczos-type product methods based on three-term Lanczos recurrences

Author(s):
Gutknecht, Martin; Ressel, K.J.

Publication Date:
1999

Permanent Link:
https://doi.org/10.3929/ethz-a-004288003

Rights / License:
In Copyright - Non-Commercial Use Permitted
Look-Ahead Procedures for Lanczos-Type Product Methods Based on Three-Term Lanczos Recurrences

M.H. Gutknecht and K.J. Ressel

Research Report No. 99-20
October 1999
Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

---

*To appear in SIAM J. Matrix Anal. Appl. (accepted June 4, 1999). This paper is a completely revised version of Ref. [26] and differs also in the algorithm proposed.

1German Aerospace Research Establishment (DLR), German Remote Sensing Data Center (DFD), D-82234 Oberpfaffenhofen, Germany
Look-Ahead Procedures for Lanczos-Type Product Methods Based on
Three-Term Lanczos Recurrences*

M.H. Gutknecht and K.J. Ressel†
Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Research Report No. 99-20 October 1999

Abstract.
Lanczos-type product methods for the solution of large sparse non-Hermitian linear systems either
square the Lanczos process or combine it with a local minimization of the residual. They inherit
from the underlying Lanczos process the danger of breakdown. For various Lanczos-type product
methods that are based on the Lanczos three-term recurrence, look-ahead versions are presented,
which avoid such breakdowns or near breakdowns at the cost of a small computational overhead.
Different look-ahead strategies are discussed and their efficiency is demonstrated by several numerical
examples.

Keywords: Lanczos-type product methods, look-ahead, iterative methods, non-
Hermitian matrices, sparse linear systems

AMS Subject Classification: 65F15, 65F10

*To appear in SIAM J. Matrix Anal. Appl. (accepted June 4, 1999). This paper is a completely
revised version of Ref. [26] and differs also in the algorithm proposed.
†German Aerospace Research Establishment (DLR), German Remote Sensing Data Center (DFD), D-82234 Oberpfaffenhofen, Germany
1. Introduction. Lanczos-type product methods (LTPMs) like (Bi)CGS [42], BiCGStab [44], BiCGStab2 [22], and BiCGStab($\ell$) [38], [41] are among the most efficient methods for solving large systems of linear equations $Ax = b$ with nonsymmetric sparse system matrix $A \in \mathbb{C}^{N \times N}$. Compared to the biconjugate gradient (BiCG) method they have the advantage of converging roughly twice as fast and of not requiring a routine for applying the adjoint system matrix $A^H$ to a vector. Nevertheless, they inherit from BiCG the short recurrence formulas for generating the approximations $x_k$ and the corresponding residuals $r_k := b - Ax_k$.

As for BiCG, the convergence can be smoothed by applying the quasi-minimal residual (QMR) method [16] or the local minimum residual process (MR smoothing) [37], [47], product methods can be combined with the same techniques [14], [47] to avoid the likely “erratic” convergence behavior [11], [36]; the benefit of these smoothing techniques is disputed, however.

A well-known problem of all methods that make implicit use of the Lanczos polynomials generated by a non-Hermitian matrix is the danger of breakdown. Although exact breakdowns are very rare in practice, it has been observed that near-breakdowns can slow down or even prevent convergence [15]. Look-ahead techniques for the Lanczos process [15], [21], [23], [34], [35], [43] allow us to avoid this problem when we use variants of the BiCG method with or without the mentioned smoothing techniques. However, the general look-ahead procedures that have so far been proposed for (Bi)CGS or other LTPMs are either limited to exact breakdowns [8], [9] or are based on different look-ahead recursions: in fact, the look-ahead steps proposed by Brezinski and Redivo Zaglia [5] to avoid near-breakdowns in CGS and those in [6], which are applicable to all LTPMs, are based on the so-called BSMRZ algorithm, which is itself based on a generalization of coupled recurrences (different from those of the standard BiOMin and BiODir versions of the BiCG method) for implementing the Lanczos method. For a theoretical comparison of the two approaches we refer to [25, § 9]. Recently, a number of further “look-ahead-like” algorithms for Lanczos-type solvers have been proposed by Ayachour [1], Brezinski et al. [7], Graves-Morris [19], and Ziegler [48]. Their discussion is beyond the scope of this paper, but it seems that [1, 7, 48] are restricted to exact breakdowns.

In this paper, we start from the approach in [15] and [23, § 9] and derive an alternative look-ahead procedure for LTPM algorithms that make use of the Lanczos three-term recurrences. Compared to the standard coupled two-term recurrences they have the advantage of being simpler to handle with regard to look-ahead, since they are only affected by one type of breakdown. In contrast to the first version of this work [26], we also capitalize upon an enhancement for the look-ahead Lanczos algorithm pointed out by Hochbruck recently (see [28], [25]), which is here adapted to LTPMs. Other improvements help to further reduce the overhead and stabilize the algorithm.

Starting with an initial approximation $x_0$ and a corresponding initial residual $y_0 := b - Ax_0$, LTPMs generate in $n$ steps a basis of the $2n$-dimensional Krylov space $K_{2n} := K_{2n}(A, y_0)$ in such a way that the even indexed basis vectors are of the form

$$
\tau_n(A)p_n(A)y_0,
$$

(1.1)

where $p_n$ is the $n$th Lanczos polynomial (see below) and $\tau_n$ is another suitably chosen polynomial of exact degree $n$. In the algorithms we discuss, these vectors will be either the residual of the $n$th approximation $x_n$ or a scalar multiple of it. By allowing them to be multiples of the residuals, that is, by considering so-called unnormalized [20] or inconsistent [25] Krylov space solvers, we avoid the occurrence of pivot (or ghost)
breakdowns. (For the various types of breakdowns and their connection to the block structure of the Padé table see [20, 24, 30, 31]. In the setting of [5, 6], they have been addressed particularly in [4].)

More generally, we define a doubly indexed sequence of product vectors \( w_n^j \) by

\[
    w_n^j := \tau(A)y_n := \tau(A)\rho_n(A)y_0.
\]

The aim is to find in the \((n + 1)\)th step of an LTPM an improved approximation \( x_{n+1} \) by computing a new product vector \( w_{n+1}^{j+1} \) from previously determined ones in a stable way. To visualize the progression of an algorithm and the recurrences it uses we arrange the product vectors \( w_n^j \) in a \( w \)-table\(^1\). Its \( n \)-axis points downwards and its \( l \)-axis to the right. We describe then how an algorithm moves in this table from the upper left corner downwards to the right.

This paper is organized as follows. In Section 2 we review the look-ahead Lanczos process. Versions based on the Lanczos three-term recurrences of various LTPMs are introduced in Section 3. In Section 4 we present for these LTPMs look-ahead procedures and analyze their computational overhead, and in Section 5 several look-ahead strategies are discussed. Our preferred way of applying the look-ahead procedures to obtain a solution of a linear system is presented in Section 6. In Section 7 the efficiency of the proposed algorithms is demonstrated by numerical examples, and in Section 8 we draw some conclusions.

2. The Look-Ahead Lanczos Process. The primary aim of the Lanczos process [32] is the construction of a pair of biorthogonal bases for two nested sequences of Krylov spaces. Given \( A \in \mathbb{C}^{N \times N} \) and a pair of starting vectors, \( (y_0, \bar{y}_0) \), the Lanczos Biorthogonalization (BiO) Algorithm generates a pair of finite sequences, \( \{y_n\}_{n=0}^\nu \), \( \{\bar{y}_n\}_{n=0}^\nu \), of left and right Lanczos vectors, such that

\[
    y_n \in K_{n+1} := \text{span} \{y_0, Ay_0, \ldots, A^n y_0\},
\]

\[
    \bar{y}_n \in \bar{K}_{n+1} := \text{span} \{\bar{y}_0, \bar{A}y_0, \ldots, (\bar{A}^H)^n \bar{y}_0\},
\]

and

\[
    \langle \bar{y}_m, y_n \rangle = \begin{cases} 
        0 & \text{if } m \neq n \text{ or } m = n = \nu, \\
        \delta_n & \text{if } m = n < \nu, 
    \end{cases}
\]

or, equivalently,

\[
    \bar{y}_n \perp K_n, \quad \bar{K}_n \perp y_n.
\]

Here, \( \langle \cdot, \cdot \rangle \) denotes the inner product in \( \mathbb{C}^N \), which we choose to be linear in the second argument, and \( \perp \) indicates the corresponding orthogonality.

The sequence of pairs of Lanczos vectors can be constructed by the three-term recurrences

\[
    y_{n+1} = (Ay_n - y_n\alpha_n - y_{n-1}\beta_n) / \gamma_n
\]

\[
    \bar{y}_{n+1} = (A^H\bar{y}_n - \bar{y}_n\bar{\alpha}_n - \bar{y}_{n-1}\bar{\beta}_n) / \gamma_n,
\]

with coefficients \( \alpha_n \) and \( \beta_n \) that are determined from the orthogonality condition (2.2) and nonvanishing scale factors \( \gamma_n \) that can be chosen arbitrarily. Choosing

---

\(^1\)The \( w \)-table is different from the scheme introduced by Sleijpen and Fokkema [38] for BiCGStab(\( \ell \)).
\( \gamma_n = -\alpha_n - \beta_n \) would allow us to consider the right Lanczos vectors \( y_n \) as residuals and to update the iterates \( x_n \) particularly simply. But this choice also introduces the possibility of a breakdown due to \( \alpha_n + \beta_n = 0 \). Therefore, we suggest in Section 6 a different way of defining iterates.

From (2.4) it follows directly that the Lanczos vectors can be written in the form

\[
\begin{align*}
\ y_n &= \rho_n (A)y_0, \\
\ y_n &= \tau_n (A^H)z_0,
\end{align*}
\]

where \( \rho_n \) denotes the \( n \)-th Lanczos polynomial. Since we aim here at LTPMs, we consider for the Krylov spaces \( K_n \) more general basis vectors of the form

\[
\ z_n = \eta_n (A^H)z_0,
\]

with arbitrary, but suitably chosen polynomials \( \eta_n \) of exact degree \( n \) and \( z_0 := y_0 \). In general, \( K_{n+1} \perp K_{n+1} \) will no longer hold, but \( K_{n+1} \perp y_{n+1} \) can still be attained by enforcing \( z_{n+1} \perp y_{n+1} \) and \( z_n \perp y_{n+1} \), which means choosing the coefficients \( \alpha_n \) and \( \beta_n \) in (2.4) in the following way: if \( n > 0 \) we need

\[
0 = \langle z_{n-1}, y_{n+1} \rangle = (\langle z_{n-1}, Ay_n \rangle - \langle z_{n-1}, y_n \rangle \alpha_n - \langle z_{n-1}, y_{n-1} \rangle \beta_n \rangle / \gamma_n,
\]

but since \( \langle z_{n-1}, y_n \rangle = 0 \), we have, with \( \delta_n := \langle z_n, y_n \rangle \),

\[
\beta_n = \frac{\langle z_{n-1}, Ay_n \rangle}{\langle z_{n-1}, y_{n-1} \rangle} = \frac{\langle z_{n-1}, Ay_n \rangle}{\delta_n}.
\]

When \( n = 0 \), we set \( \beta_0 = 0 \). Similarly, from the orthogonality condition \( \langle z_n, y_{n+1} \rangle = 0 \) we obtain

\[
\alpha_n = \frac{\langle z_n, Ay_n - y_n - \beta_n \rangle}{\langle z_n, y_n \rangle} = \frac{\langle z_n, Ay_n - y_n - \beta_n \rangle}{\delta_n}.
\]

Equations (2.7)–(2.9) and the first recurrence of (2.4) specify what one might call the one-sided Lanczos process, formulated in [36] to derive LTPMs.

Clearly, this recursive process terminates with \( y_k = 0 \) or \( z_k = 0 \), or it breaks down with \( \delta_k = 0 \) and \( y_k \neq 0 \), \( z_k \neq 0 \). The look-ahead Lanczos process [23, §9], [15] overcomes such a breakdown if curable, i.e., if for some \( k \), \( \langle z_k, Ay_k \rangle \neq 0 \). However, its role is not restricted to treating such exact breakdowns with \( \delta_k = 0 \): it allows us to continue the biorthogonalization process whenever for stability reasons we choose to enforce the orthogonality condition (2.2) only partially for a couple of steps. We will come back later to the conditions that make us start such a look-ahead phase.

In the look-ahead Lanczos process the price we have to pay is that the Gramian matrix \( D := \langle z_k, y_k \rangle \) becomes block-lower triangular instead of triangular (as in the generic one-sided Lanczos algorithm) or diagonal (as in the generic two-sided Lanczos algorithm). In other words, we replace the conditions \( K_n \perp y_n \) and \( \delta_n \neq 0 \) by

\[
K_{n_j} \perp y_n, \quad \text{if } n \geq n_j \quad (j = 0, \ldots, J),
\]

and

\[
D_j := \langle z_k, y_j \rangle_{k, l=n_j}^{n_j+1} \quad \text{nonsingular if } \quad j < J,
\]
where $0 = n_0 < n_1 < \cdots < n_J = \nu \leq N$ is the subsequence of indices of (well-conditioned) regular Lanczos vectors $y_{n:j}$; for the other indices the vectors are called inner vectors. Likewise, we refer to $n$ as a regular index if $n = n_j$ for some $j$, while $n$ is called an inner index otherwise. Note that there is considerable freedom in choosing the subsequence $\{n_j\}_{j=0}^J$; for example, we might request that the smallest singular value $\sigma_{\text{min}}(D_j)$ is sufficiently large.

The sequence $\{y_n\}$ is determined by the condition $K_{n_j} \perp y_n$ if $n_j \leq n < n_{j+1}$ and hence it does not depend directly on the sequence $\{\tilde{z}_n\}$ but only on the spaces $K_{n_j}$ that are spanned by the first $n_j$ elements of $\{\tilde{z}_n\}$. On the other hand, the smallest singular value of each $D_j$ depends on the basis chosen, a fact that one should take into account. Forming the blocks

$$Y_j := \begin{bmatrix} y_{n_j} & y_{n_j+1} & \cdots & y_{n_{j+1}-1} \end{bmatrix}, \quad \tilde{Z}_j := \begin{bmatrix} \tilde{z}_{n_j} & \tilde{z}_{n_j+1} & \cdots & \tilde{z}_{n_{j+1}-1} \end{bmatrix},$$

we can express relation (2.10) as

$$\tilde{Z}_j^H Y_j = \begin{cases} D_j & \text{if } l = j, \\ 0 & \text{if } l < j. \end{cases}$$

(2.12)

The look-ahead step size is in the following denoted by $h_j := n_{j+1} - n_j$.

In the look-ahead case the Lanczos vectors $\{y_n\}$ can be generated by the recursion [23, §9], [15]:

$$y_{n+1} = (A y_n - \tilde{Y}_j \alpha_n - Y_{j-1} \beta_n) / \gamma_n \quad (n_j < n + 1 \leq n_{j+1}),$$

(2.13)

where $\tilde{Y}_j := \begin{bmatrix} y_{n_j} & \cdots & y_{n} \end{bmatrix}$ denotes the not yet fully completed $j$-block if $n+1$ is an inner index, while $\tilde{Y}_j = Y_j$ if $n+1$ is regular, that is, if $n+1 = n_{j+1}$. The coefficient vector $\beta_n$ is determined by the condition

$$0 = \tilde{Z}_{j-1}^H y_{n+1} = \tilde{Z}_{j-1}^H A y_n - \tilde{Z}_{j-1}^H Y_{j-1} \beta_n,$$

thus,

$$\beta_n = D_{j-1}^{-1} \tilde{Z}_{j-1}^H A y_n.$$

(2.14)

The coefficient vector $\alpha_n$ can be chosen arbitrarily if $n+1$ is an inner index. Obviously, the choice $\alpha_n = 0$ yields the cheapest recursion, but we may gain numerical stability by choosing $\alpha_n \neq 0$. On the other hand, if $n+1$ is a regular index, $\alpha_n$ results from the condition

$$0 = \tilde{Z}_j^H y_{n+1} = \tilde{Z}_j^H A y_n - \tilde{Z}_j^H Y_j \alpha_n - \tilde{Z}_j^H Y_{j-1} \beta_n,$$

that is

$$\alpha_n = D_j^{-1} \tilde{Z}_j^H (A y_n - Y_{j-1} \beta_n).$$

(2.15)

Recently, it was pointed out by Hochbruck [28] that the recursion (2.13) can be simplified since the contribution of the older block $Y_{j-1}$ can be represented by multiples of a single auxiliary vector $y'_{j-1}$. This is due to the fact (noticed in [23]) that the matrix made up of the coefficient vectors $\beta_{n_j}, \ldots, \beta_{n_{j+1}-1}$ for block $j$ is of rank one; see also [25, §19]. This simplification has also been capitalized upon in
the look-ahead Hankel solver of Freund and Zha [17], which is closely related to the look-ahead Lanczos algorithm.

In particular, due to (2.10) or (2.12) we have

\[(2.16) \quad \langle \tilde{z}_{l+1}, y_l \rangle = 0 \quad \text{and} \quad \langle \tilde{z}_l, Ay_l \rangle = 0 \quad \text{if} \quad l < n - 1 < n.\]

Therefore, (2.14) simplifies to

\[(2.17) \quad \beta_n = D_{j-1}^{-1} \tilde{z}_j^H A y_n = D_{j-1}^{-1} \ell \tilde{z}_{n-j-1} A y_n =: D_{j-1}^{-1} \ell \beta_n',\]

where \( \ell := [0, \ldots, 0, 1]^T \) and

\[(2.18) \quad \beta_n' := \langle \tilde{z}_{n-j-1}, Ay_n \rangle.\]

Introducing the auxiliary vector

\[(2.19) \quad y_{j-1}' := Y_{j-1} (D_{j-1}^{-1} \ell),\]

we have

\[(2.20) \quad Y_{j-1} \beta_n = (Y_{j-1} D_{j-1}^{-1} \ell) \left( \tilde{z}_{n-j-1}^H A y_n \right) = y_{j-1}' \beta_n',\]

so that the recurrence (2.13) becomes

\[(2.21) \quad y_{n+1} = \left( Ay_n - \tilde{Y}_j \alpha_n - y_{j-1}' \beta_n' \right) / \gamma_n \quad (n_j < n + 1 \leq n_{j+1}),\]

and (2.15) changes to

\[(2.22) \quad \alpha_n = D_{j-1}^{-1} \tilde{z}_j^H (A y_n - y_{j-1}' \beta_n').\]

3. **Lanczos-Type Product Methods (LTPMs).** LTPMs are based on two ideas. The first one is to derive for a certain sequence of product vectors \( w_{n+1} \) recursion formulas that involve only previously computed product vectors, so that there is no need of explicitly computing the vectors \( \tilde{z}_l \) and \( y_n \). By multiplying the three-term recursion (2.4) for the right Lanczos vectors \( y_n \) with \( \tau_j(A) \) we obtain

\[(3.1) \quad w_{n+1} = (A w_n - w_n \alpha_n - w_{n-1} \beta_n) / \gamma_n.\]

This recursion can be applied to move forward in the vertical direction in the \( w \)-table. To obtain a formula for proceeding in the horizontal direction, the recursion for the chosen polynomials \( \tau_j(z) \) is capitalized upon in an analogous way.

The second basic idea is to rewrite the inner products that appear in the Lanczos process in terms of product vectors:

\[\langle \tilde{z}_l, y_n \rangle = \langle \tau_j(A^H) \tilde{z}_0, y_n \rangle = \langle \tilde{z}_0, \tau_j(A) y_n \rangle = \langle \tilde{z}_0, w_n \rangle,\]

\[\langle \tilde{z}_l, Ay_n \rangle = \langle \tau_j(A^H) \tilde{z}_0, Ay_n \rangle = \langle \tilde{z}_0, A \tau_j(A) y_n \rangle = \langle \tilde{z}_0, Aw_n \rangle.\]

For the coefficients \( \alpha_n \) and \( \beta_n \) of (2.8) and (2.9) this results in

\[(3.2) \quad \beta_n = \frac{\langle \tilde{z}_0, Aw_n^{-1} \rangle}{\langle \tilde{z}_0, w_{n-1} \rangle}, \quad \alpha_n = \frac{\langle \tilde{z}_0, Aw_n - w_{n-1} \beta_n \rangle}{\langle \tilde{z}_0, w_n \rangle}.\]
LTPMs still have short recurrence formulas if the polynomials \( \tau_l \) have a short one. In addition, they have two advantages over BiCG: first, multiplications with the adjoint system matrix \( A^H \) are avoided; second, for an appropriate choice of the polynomials \( \tau_l(\zeta) \) smaller new residuals \( r_l := u^l_l = \tau_l(A)y_l \) can be expected because of a further reduction of \( y_l \) by the operator \( \tau_l(A) \). Different choices of the polynomials \( \tau_l(\zeta) \) lead to different LTPMs. In the following we briefly review some possible choices for these polynomials. We start with the general class where they satisfy a three-term recursion. Since then both the ‘left’ polynomials \( \tau_l \) and the ‘right’ polynomials \( \rho_l \) fulfill a three-term recursion, we say that this is the class of (3,3)-type LTPMs.

In the following we briefly review some possible choices for these polynomials. For a more detailed discussion and a pseudocode for the resulting algorithms we refer to [36].

3.1. LTPMs based on a three-term recursion for \{\( \tau_l \)\}: BiOxCheb and BiOxMR2. Assume the polynomials \( \tau_l(\zeta) \) are generated by a three-term recursion of the form [22]

\[
\tau_{l+1}(\zeta) = (\xi + \eta \zeta) \tau_l(\zeta) + (1 - \xi_l) \tau_{l-1}(\zeta),
\]

with \( \tau_{-1}(\zeta) \equiv 0, \tau_0(\zeta) \equiv 1, \xi_0 = 1, \) and \( \eta_l \neq 0 \) for all \( l \). Note that, by induction, \( \tau_l \) has exact degree \( l \) and \( \tau_l(0) = 1 \) for all \( l \). Hence, the polynomials qualify as residual polynomials of a Krylov space method.

Multiplying \( y_n \) by \( \tau_{l+1}(A) \) from the left and applying (3.3) yields the horizontal recurrence

\[
w^{l+1}_n = A w^l_n \xi + w^l_n \xi_l + w^{l-1}_n (1 - \xi_l).
\]

By applying (3.1) and (3.4) the following loop produces a new product vector \( w^{n+1}_{n+1} \) from previously calculated ones, namely \( w^l_m, (m, l) := (n, n), (n-1, n), (n, n-1), (n-1, n-1) \), and the product \( A w^{n-1}_n \) also evaluated before. At the end of the loop, besides \( w^{n+1}_{n+1} \) also \( w^{n+1}_n, w^{n+1}_{n-1} \) and \( A w^{n+1}_n \), which will be needed in the next run through the loop, are available.

**Loop 3.1.** (General (3,3)-type LTPM)

1. Compute \( A w^n_n \) and determine \( \beta_n \) and \( \alpha_n \).
2. Use (3.1) to compute \( w^{n-1}_{n+1} \) (if \( n > 0 \)) and \( w^{n+1}_{n+1} \).
3. Compute \( A w^n_{n+1} \) and determine \( \xi_n \) and \( \eta_n \).
4. Use (3.4) to compute \( w^n_{n+1} \) and \( w^{n+1}_{n+1} \).

Next we discuss two ways of choosing the polynomials \( \tau_l \), that is, of specifying the recurrence coefficients \( \xi_l \) and \( \eta_l \).

One possibility is to combine the Lanczos process with the Chebyshev method [13], [45] by choosing \( \tau_l \) as a suitably shifted and scaled Chebyshev polynomial. This combination was suggested in [3, 22, 44]. Let us call the resulting (3,3)-type LTPM BiOxCheb. It is well known that these polynomials satisfy a recurrence of the form (3.3). After acquiring some information about the spectrum of the matrix \( A \), for example, by performing a few iterations with another Krylov space method, one can determine the recursion for the scaled and shifted Chebyshev polynomials that correspond to an ellipse surrounding the estimated spectrum [33]. However, the necessity to provide spectral information is often seen as a drawback of this method.

Another idea is to use the coefficients \( \xi_l \) and \( \eta_l \) of (3.3) for locally minimizing the norm of the new residual \( w^{l+1}_{l+1} \). This idea is borrowed from the BiCGStab and
BiCGStab2 methods [44], [22] reviewed below: \( \xi \) and \( \eta \) are determined by solving the two-dimensional minimization problem

\[
(3.5) \quad \min_{\xi, \eta \in \mathbb{C}} \| w_{i+1}^{l-1} + (w_{i+1}^{l} - w_{i+1}^{l-1})\xi + Au_{i+1} \eta \|.
\]

Introducing the \( N \times 2 \) matrix \( B_{i+1} := \begin{bmatrix} (w_{i+1}^{l} - w_{i+1}^{l-1}) & Au_{i+1} \end{bmatrix} \) we can write (3.5) as the least-squares problem

\[
\min_{\xi, \eta \in \mathbb{C}} \| w_{i+1}^{l-1} + B_{i+1} \begin{bmatrix} \xi \\ \eta \end{bmatrix} \|.
\]

Therefore, \( \xi \) and \( \eta \) can be computed as the solution of the normal equations

\[
(3.6) \quad (B_{i+1}^H B_{i+1}) \begin{bmatrix} \xi \\ \eta \end{bmatrix} = -B_{i+1}^H w_{i+1}^{l-1}.
\]

In view of the two-dimensional local residual norm minimization performed at every step (except the first one) we call this the BiOxMR2 method. A version based on coupled two-term Lanczos recurrences of this method was introduced by the first author in a talk in Oberwolfach (April 1994). Independently it was as well proposed by Cao [10] and by Zhang [46], whose Technical Report is dated April 1993. Zhang also considered two-term formulas for \( \tau \) and presented very favorable numerical results.

3.2. LTPMs based on a two-term recursion for \( \{ \tau \} \): BiOSstab. In Van der Vorst’s BiCGStab [44] the polynomials \( \tau_l(\zeta) \) are built up successively as products of polynomials of degree 1:

\[
(3.7) \quad \tau_0(\zeta) \equiv 1, \quad \tau_{l+1}(\zeta) = (1 - \chi_l(\zeta))\tau_l(\zeta) \quad \text{for} \ l \geq 0.
\]

Inserting here for \( \zeta \) the system matrix \( A \) and multiplying by \( y_n \) from the right yields

\[
(3.8) \quad w_{n+1}^{l+1} = w_n^l - Au_n^l \chi_l.
\]

The coefficient \( \chi_l \) is determined by minimizing the norm of \( w_{l+1}^{l+1} = w_{l+1}^l - Au_{l+1}^l \chi_l \), that is by solving the one-dimensional minimization problem

\[
(3.9) \quad \min_{\chi \in \mathbb{C}} \| w_{l+1}^{l+1} - Au_{l+1}^l \chi \|,
\]

which leads to

\[
(3.10) \quad \chi_l = \frac{\langle Au_{l+1}^l, w_{l+1}^{l+1} \rangle}{\langle Au_{l+1}^l, Au_{l+1}^l \rangle}.
\]

Recurrences (3.1) and (3.8) can be used to compute a new product vector \( w_{n+1}^{l+1} \) from \( w_n^l \) and \( w_{n-1}^l \) as described in Loop 3.1 with \( \xi_n = 1 \) and \( \eta_n = -\chi_n \), where \( \chi_n \) is calculated from (3.10). However, since (3.8) is only a two-term recurrence, there is no need now to compute \( w_{n+1}^{l+1} \) in substep 2 of the loop.

We call this algorithm BiOSstab since it is a version of BiCGStab that is based on the three-term recurrences of the Lanczos biorthogonalization process.

---

1The letter ‘x’ in the name BiOxMR2 reflects the fact that the residual polynomials of this method are the products of the Lanczos polynomials generated by the BiO process with polynomials obtained from a successive two-dimensional minimization of the residual (MR2). Similarly, BiOxCheb means a combination of the BiO process with a Chebyshev process.

2Elkhour [12] also proposed such a variant of BiCGStab; however, his way of computing the Lanczos coefficients is much too complicated.
3.3. BiOStab2. BiOStab2 is the version of BiCGStab2 [22] based on the three-term Lanczos recurrences. The polynomials \( \eta(\zeta) \) satisfy the recursions

\[
\begin{align*}
\eta_0(\zeta) & \equiv 1, \\
\eta_{l+1}(\zeta) & = (1 - \chi_l \zeta) \eta_l(\zeta) & \text{if } l \text{ is even,} \\
\eta_{l+1}(\zeta) & = (\xi_l + \eta_l \zeta) \eta_l(\zeta) + (1 - \xi_l) \eta_{l-1}(\zeta) & \text{if } l \text{ is odd.}
\end{align*}
\]

Here \( \chi_l \) may be obtained by solving the one-dimensional minimization problem (3.9), so \( \chi_l \) is given by (3.10). However, if \( |\chi_l| \) is small, this choice is dangerous since the vector component needed to enlarge the Krylov space becomes negligible [38]. Then some other value of \( \chi_l \) should be chosen. Except for roundoff, the choice has no effect on later steps, because \( \xi_l \) and \( \eta_l \) are determined by solving the two-dimensional minimization problem (3.5).

Multiplying \( y_n \) by \( \eta_{l+1}(A) \) and applying (3.11) leads to

\[
\begin{align*}
\tilde{w}_{n+1}^l = & \begin{cases} \\
\frac{w_n^l - A w_n^l \chi_l}{\eta_n^l + w_n^l \xi_l + w_n^{l-1} (1 - \xi_l)} & \text{if } l \text{ is odd.}
\end{cases}
\end{align*}
\]

Now, Loop 3.1 applies with \( \xi_n = 1, \eta_n = -\chi_n \) if \( n \) is even, while \( \xi_n \) and \( \eta_n \) are chosen as indicated above if \( n \) is odd. If \( n \) is even, there is no need to compute \( w_{n+1}^{n-1} \) in substep 2 of the loop.

3.4. BiO-Squared (BiOS). BiOS is obtained by “squaring” the three-term Lanczos process: among the basis vectors generated are those Krylov space vectors that correspond to the squared Lanczos polynomials. By complementing BiOS with a recursion for Galerkin iterates we will obtain BiOResS, a (3,3)-type version of Sonneveld’s (2,2)-type conjugate gradient squared (CGS) method [42]. The method fits into the framework of LTPMs if we identify

\[
\tau_l(\zeta) = \rho_l(\zeta).
\]

The vectors \( \tilde{z}_n = \tilde{\pi}_n (A^H \tilde{z}_0) = \tilde{\pi}_n (A^H) \tilde{y}_0 = \tilde{y}_n \) are then exactly the left Lanczos vectors so that now \( y_n \perp \tilde{\mathcal{K}}_n \) as well as \( \tilde{z}_n \perp \mathcal{K}_n \) is fulfilled. Thus, the coefficient \( \alpha_n \) in (3.2) simplifies to

\[
\alpha_n = \frac{\langle \tilde{z}_0, A w_n^m \rangle}{\langle \tilde{z}_0, w_n^m \rangle},
\]

and the \( w \)-table becomes symmetric since

\[
w_n^l = \rho_l(A) \rho_n(A) y_0 = \rho_n(A) \rho_l(A) y_0 = w_n^l.
\]

Consequently, we have in analogy to (3.1)

\[
w_{n+1}^{l+1} = \left( A w_n^l - w_n^l \alpha_l - w_n^{l-1} \beta_l \right) / \gamma_l.
\]
These two recursions lead to the following loop:

**Loop 3.2. (BiOS)**
1. Compute $Au^n_n$ and determine $\beta_n$ and $\alpha_n$.
2. Use (3.1) to compute $w_{n+1}^{n+1}$ (if $n > 0$) and $w_{n+1}^n$.
3. Compute $Au_{n+1}^n$.
4. Use (3.16) to compute $w_{n+1}^{n+1}$.

Note that we exploit in substep 2 the symmetry of the $w$-table: the product vector $w_{n-1}^n$, which is needed for the calculation of $w_{n+1}^n$ by (3.1), is equal to $w_{n-1}^n$ and need not be stored. We also point out that (3.16) is not of the form (3.4); in particular, the coefficients of $w_{n-1}^n$ and $w_n^n$ need not sum up to 1.

**4. Look-Ahead Procedures for LTPMs.** Look-ahead steps in an LTPM serve to stabilize the Lanczos process, the vertical movement in the $w$-table. Except for BiOS, the recursion formulas for the horizontal movement remain the same. The vertical movement is now in general based on the recurrence formula (2.21) for the Lanczos vectors, but we need to replace these vectors by product vectors $w_n^n$; see (1.2).

We introduce the blocks of product vectors

$$
\tilde{W}_j^l := \begin{bmatrix} w_{n_j}^l & \ldots & w_n^l \end{bmatrix}, \quad W_{j-1}^l := \begin{bmatrix} w_{n_{j-1}}^l & \ldots & w_{n_j-1}^l \end{bmatrix}
$$

and the auxiliary product vector $u_{j-1}^l$ defined by

$$
u_{j-1}^l := \alpha_n (A) y_{j-1}^l = \alpha_n (A) y_{j-1}^l (D_{j-1}^{-1} \ell) = W_{j-1}^l (D_{j-1}^{-1} \ell).
$$

Again $\tilde{W}_j^l = W_j^l$ if $n + 1 = n_{j+1}$ is a regular index, while $\tilde{W}_j^l$ denotes the not yet completed $j$th block if $n + 1$ is an inner index. Now, multiplying (2.21) by $\alpha_n$ from the left, we obtain

$$
u_{j-1}^l = (Au_n - \tilde{W}_j^l \alpha_n - u_{j-1}^l \beta_n^l) / \gamma_n \quad (n_j < n + 1 \leq n_{j+1}).
$$

As in Section 3, the coefficient vectors $\beta_n$ and, in the regular case, $\alpha_n$ can be expressed in terms of the product vectors by rewriting all inner products in such a way that the part $\alpha_n$ of $\gamma_n$ on the left side is transferred to the right side of the inner product. For the diagonal blocks $D_j$ of the Gramian this yields

$$
D_j = [(\tilde{z}_0, w_n^l)_{i,k=1}^{n_{j+1}-1},
$$

and $\beta_n$ of (2.18) becomes

$$
\beta_n = \tilde{z}_0^H Au_{n_j}^{n_j-1} \quad (n_j < n + 1 \leq n_{j+1}).
$$

Likewise, $\alpha_n$ of (2.22) turns into

$$
\alpha_n = D_j^{-1} \begin{bmatrix} \langle \tilde{z}_0, Au_{n_j}^{n_j-1} - w_{j-1}^{n_j-1} \beta_n^l \rangle \\ \vdots \\ \langle \tilde{z}_0, Au_{n_j}^{n_j-1} - w_{j-1}^{n_j-1} \beta_n^l \rangle \\ \langle \tilde{z}_0, Au_{n_j}^{n_j-1} - w_{j-1}^{n_j-1} \beta_n^l \rangle \end{bmatrix} (n + 1 = n_{j+1}).
$$
For advancing in the horizontal direction of the \(w\)-table we will still use (3.4), (3.8), the combination (3.12), or (3.16). Substituting \( Aw_{n_j}^{n_j-1} \) in (4.4) according to these formulas and letting \( \delta_{n_j}^{n_j} = \langle z_0, w_{n_j}^{n_j} \rangle \) be the \((n_j, n)\)-element of \( D \) (4.4) simplifies to

\[
\beta_{n_j}^{n_j} = \begin{cases} 
\delta_{n_j}^{n_j} / \eta_{n_j-1}, & \text{if (3.4) holds for } l = n_j - 1, \\
-\delta_{n_j}^{n_j} / \chi_{n_j-1}, & \text{if (3.8) holds for } l = n_j - 1, \\
\delta_{n_j}^{n_j} \gamma_{n_j-1}, & \text{if (3.16) holds for } l = n_j - 1.
\end{cases}
\]

We remark that by using (4.6) instead of \( \langle z_0, Aw_{n_j}^{n_j-1} \rangle \) we avoid to compute and store the complete \( W_j^{n_j-1} \) block. Now only few elements of this block are needed, see Figure 4.1 below. In this figure we display those entries \( w_{n}^{n_j} \) and products \( Aw_{n}^{n_j} \) in the \( w\)-table that are needed to compute by (4.2) a new vector \( w_{n+1}^{n_j} \) marked by ‘*’.

\[
\begin{array}{c|ccc|c|ccc|c|}
\text{inner case} & \text{regular case} \\
\hline
n_j-1 & \eta_{n_j-1} & \vdots & \eta_{n_j-1} & \eta_{n_j-1} & \eta_{n_j-1} & \cdots & \eta_{n_j-1} & \eta_{n_j-1} \\
\hline
n_j & \delta_{n_j} & \delta_{n_j} & \delta_{n_j} & \delta_{n_j} & \delta_{n_j} & \cdots & \delta_{n_j} & \delta_{n_j} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
n & \chi_{n_j} & \chi_{n_j} & \chi_{n_j} & \chi_{n_j} & \chi_{n_j} & \cdots & \chi_{n_j} & \chi_{n_j} \\
n + 1 & * & * & * & * & * & \cdots & * & * \\
\hline
\end{array}
\]

**Fig. 4.1.** Entries in the \( w\)-table needed for the construction of a new inner or regular vector marked by ‘*’ by recurrence formula (4.2). Here, ‘d’ indicates entries needed for \( w_{n_j}^{n_j-1} \), ‘V’ those for \( \bar{W}_j \), ‘d’ those for \( D_{n_j} \), ‘D’ those additionally required for \( D_j \). Moreover, ‘\( \alpha’ \) marks auxiliary vectors \( w_{n_j}^{n_j} (n_j \leq k < n_{j+1}) \) in the formula (4.5) for \( \alpha_n \), and ‘A’ stands for a matrix-vector product \( (\hat{M}V) \) with \( A \) in this formula. One such product also appears explicitly in (4.2).

Note that the horizontal recursions are valid for the auxiliary product vectors \( w_{n_j}^{n_j+1} \) too: for example, (3.4) and (4.1) imply

\[
w_{n_j}^{n_j+1} = Aw_{n_j}^{n_j} \eta_l + w_{n_j}^{n_j} \xi_j + w_{n_j}^{n_j-1} (1 - \xi_j),
\]

while (3.8) and (4.1) provide

\[
w_{n_j}^{n_j+1} = w_{n_j}^{n_j} - Aw_{n_j}^{n_j} \chi_l.
\]

Thus, the auxiliary vector can be updated in the horizontal direction at the cost of one matrix-vector product \((MV)\) per step.

Combining the recursions for the vertical movement with those for the horizontal movement leads to the look-ahead version of an LTPM. (Actually, we will also need to compute approximations to the solution \( A^{-1} b \) of the linear system; but we defer this till section 6). Because of the additional vectors involved in the above recurrence formulas, it seems that such a look-ahead LTPM requires much more computational work and storage than its unstable, standard version. However, the number of look-ahead steps as well as the size of their blocks is small in practice, so that the overhead

---

10
is moderate. Moreover, we describe in the following for various choices of polynomials \( \tau_n(\zeta) \) how MVs that are needed can be computed indirectly by applying the recurrence formulas. In the same way also the values of inner products can be obtained indirectly at nearly no cost.

4.1. Look-ahead LTPMs based on a three-term recursion for \( \{ \tau_n \} \): LA-BiOxCheb and LA-BiOxMR2. For methods incorporating a horizontal three-term recurrence, such as BiOxCheb and BiOxMR2, applying the above principles for look-ahead LTPMs leads to the general Loop 4.1. Note that the first four substeps are identical in both cases. We will see in Section 5 that the decision between a regular and an inner loop is made during substep 5.

In Figure 4.2 we display the action of this loop in the \( w \)-table, but now we use a different format than before, which, on the one hand specifies what has been known before the current sweep through the loop and what is being computed in this sweep. In particular, the following symbols and indicators are used:

- ‘V’ indicates that the corresponding product vector is already known;

- ‘A’ as ‘denominator’ indicates that the product of the vector represented by ‘V’ with the matrix \( A \) was needed;

- a solid box around a ‘fraction’ means that this product by \( A \) required (or requires) an MV;

- no box around a ‘fraction’ means that this product can be obtained by applying a recurrence formula;

- a number as entry specifies in which substep of the current sweep this entry is calculated;

- a prime indicates that the vector is an auxiliary one, as defined in (4.1) (these vectors are displayed in the last row of the corresponding block of the \( w \)-table);

- ‘A’ as ‘denominator’ indicates that the product of the vector represented by ‘V’ with the matrix \( A \) was needed;

- double primes will indicate that the vector is an auxiliary one of the type defined in (4.11) used in LA-BiCGS (these vectors are displayed in the lower right corner of the corresponding block of the \( w \)-table);

- ‘S’ will indicate that the vector or the MV is obtained for free due to the symmetry of the \( w \)-table of LA-BiCGS.
**Loop 4.1.** (Look-ahead for (3,3)-type LTPM) Let \( \mu := \min \{ n_j, n - 1 \} \).

<table>
<thead>
<tr>
<th>Inner Loop: ( n_j &lt; n + 1 &lt; n_j + 1 )</th>
<th>Regular Loop: ( n + 1 = n_j + 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Compute ( A_{n}^{n} ).</td>
<td>1. Compute ( A_{n}^{n} ).</td>
</tr>
<tr>
<td>2. If ( n &gt; n_j ), use (4.2) to compute ( A_{n}^{n} ) indirectly ( A_{n_j}^{n} ), ( A_{n_j + 1}^{n} ), ..., ( A_{n - 1}^{n} ).</td>
<td>2. If ( n &gt; n_j ), use (4.2) to compute ( A_{n}^{n} ) indirectly ( A_{n_j}^{n} ), ( A_{n_j + 1}^{n} ), ..., ( A_{n - 1}^{n} ).</td>
</tr>
<tr>
<td>3. If ( n \geq n_j + 2 ), compute ( A_{n}^{n} ).</td>
<td>3. If ( n \geq n_j + 2 ), compute ( A_{n}^{n} ).</td>
</tr>
<tr>
<td>4. If ( n \geq n_j + 3 ), use (3.4) to compute ( A_{n_j}^{n} ), ( A_{n_j + 1}^{n} ), ..., ( A_{n - 2}^{n} ).</td>
<td>4. If ( n \geq n_j + 3 ), use (3.4) to compute ( A_{n}^{n} ), ( A_{n+1}^{n} ), ..., ( A_{n - 2}^{n} ).</td>
</tr>
<tr>
<td>5. Use (4.2) to compute ( w_{n+1}^{n} ), ( w_{n}^{n} ), ..., ( w_{n+1}^{n} ).</td>
<td>5. Use (4.2) to compute ( w_{n+1}^{n} ) and ( w_{n+1}^{n} ).</td>
</tr>
<tr>
<td>6. Compute ( A_{n+j}^{n} ) and ( \xi_n, \eta_n ).</td>
<td>6. Compute ( A_{n}^{n} ) and ( \xi_n, \eta_n ).</td>
</tr>
<tr>
<td>7. Use (3.4) to compute ( w_{n}^{n+1} ), ..., ( w_{n+1}^{n+1} ).</td>
<td>7. Use (3.4) to compute ( w_{n}^{n+1} ), ..., ( w_{n+1}^{n+1} ).</td>
</tr>
<tr>
<td>8. Compute ( A_{n}^{n} ) and use (4.7) to compute ( w_{n+1}^{n+1} ).</td>
<td>8. Compute ( w_{n}^{n} ) and ( w_{n+1}^{n+1} ) according to definition (4.1).</td>
</tr>
</tbody>
</table>

**Fig. 4.2.** Action of Loop 4.1 (Look-ahead for (3,3)-type LTPM) in the \( w \)-table. For blocks of different sizes an inner step (at left) and the following regular step (at right) are shown.

In the first two \( w \)-tables of Figure 4.2 we display what happens in a first inner step (at left) and in a regular step (at right) that follows directly a regular step (so that \( h_j = 1 \)). In the second pair of \( w \)-tables of Figure 4.2 we consider an inner and a
regular step in case of a large block size. Note that the substeps 2-4 of the loop do not appear in the first pair, and that substep 4 would still not be active in the regular loop at the end of a look-ahead step of length $h_j = 3$, while, when $h_j \geq 4$, it becomes active.

For such a look-ahead step of length $h_j$, the cost in terms of MVs is $4h_j - 3$ MVs if $h_j > 1$. In fact, in the first substep of the $h_j - 1$ inner and the one regular loops $h_j$ MVs are consumed; another $h_j - 2$ MVs are needed in substep 3, further $h_j$ MVs in substep 6, and the remaining $h_j - 1$ MVs are used in substep 6 of the inner loops. If no look-ahead is needed, that is if $h_j = 1$, then 2 MVs are required, both in our procedure and in the standard one that does not allow for look-ahead. Hence, in a step of length 2, we have 25% overhead, in a step of length 3 there is 50% overhead, and for even longer steps, which are very rare in practice, the overhead grows gradually towards 100%.

4.2. Look-ahead for BiOStab and BiOStab2. Since the horizontal recurrence (3.8) for BiOStab is only a two-term one, there is no need to compute elements of the second subdiagonal of the $w$-table as long as we are not in a look-ahead step. In case of a look-ahead step, this remains true for those of these elements that lie in a subdiagonal block, but, of course, not for those in a diagonal block. For Loop 4.1 this means that $\mu$ simplifies to $\mu := n_j$ and that in substep 5 of a regular loop there is no need to compute $w_{n_j+1}$. All the other changes refer to equation numbers or the coefficients $\xi_n$ and $\eta_n$. In summary, we obtain Loop 4.2. Again the first four substeps are the same in both cases, and the choice between them will be made in substep 5. In Figure 4.3 we display for this loop the two sections of $w$-tables that correspond to the second pair in Figure 4.2.

Since those product vectors from Loop 4.1 that are no longer needed in Loop 4.2 were found without an extra MV before, the overhead in terms of MVs remains the same here.

Look-ahead for BiOStab2 could be defined along the same lines, by alternating between steps of Loop 4.1 and Loop 4.2. At this point it also becomes clear how to obtain a look-ahead version of BiOStab($\ell$), an algorithm analogous to BiCGStab($\ell$) of Sleijpen and Fokkema [38], but based on the three-term Lanczos process instead of coupled two-term BiCG formulas.

4.3. Look-ahead (bi)conjugate gradient squared: LA-BiOS. For BiOS, which will be the underlying process for our BiOResS version of BiCGS, the horizontal recurrence (3.4) has to be substituted by the Lanczos recurrence given in (3.16), which may need to be replaced by the look-ahead formula that is analogous to (4.2) with $l$ and $n$ exchanged and with suitably defined auxiliary vectors and blocks of vectors. But since the $w$-table is symmetric, we can build it up by vertical recursions only and reflections at the diagonal.

We only formulate the recursion for $w_{j-1}^{l+1}$ as a horizontal one that replaces (4.7): if $n_k < l + 1 \leq n_{k+1}$,

$$u_{j-1}^{l+1} = (Au_{j-1}^l - \hat{W}_{j-1}^l \alpha_l - W_{j-1}^{l+1} \beta_l) / \gamma_l,$$

where now

$$\hat{W}_{j-1}^l := [ u_{j-1}^{l+1} \ldots u_{j-1}^l ], \quad W_{j-1}^{l+1} := [ u_{j-1}^{l+1} \ldots u_{j-1}^{l+1} ].$$

Recurrence (4.9) follows from the horizontal Lanczos look-ahead recurrences for computing $u_{n_j-1}^{l+1}, \ldots, u_{n_j-1}^{l+1}$, derived from (2.13) instead of (2.21), which can be gathered
**Loop 4.2. (Look-ahead for BiOStab.)**

**Inner Loop: \((n_j < n+1 < n_{j+1})\)**

1. Compute \(A^{n_j}_n\).
2. If \(n > n_j\), use (4.2) to compute indirectly \(A^{n_j+1}_n, \ldots, A^{n_{n-1}}_n\).
3. If \(n \geq n_j + 2\), compute \(A^{n_j}_n\).
4. If \(n \geq n_j + 3\), use (3.8) to compute indirectly \(A^{n_j+1}_n, \ldots, A^{n_{n-2}}_n\).
5. Use (4.2) to compute \(w^{n+1}_{n+1}, \ldots, w^{n_j+1}_{n+1}\).
6. Compute \(A^{n+1}_{n+1}\) and \(\chi_n\).
7. Use (3.8) to compute \(w^{n+1}_{n+1}, \ldots, w^{n_j+1}_{n+1}\).
8. Compute \(A^{n+1}_{n+1}\) and use (4.8) to compute \(w^{n+1}_{n+1}\).

**Regular Loop: \((n+1 = n_{j+1})\)**

1. Compute \(A^{n}_{n}\).
2. If \(n > n_j\), use (4.2) to compute indirectly \(A^{n+1}_n, \ldots, A^{n_{n-1}}_n\).
3. If \(n \geq n_j + 2\), compute \(A^{n}_{n}\).
4. If \(n \geq n_j + 3\), use (3.8) to compute indirectly \(A^{n+1}_n, \ldots, A^{n_{n-2}}_n\).
5. Use (4.2) to compute \(w^{n+1}_{n+1}\).
6. Compute \(A^{n+1}_{n+1}\) and \(\chi_n\).
7. Use (3.8) to compute \(w^{n+1}_{n+1}, \ldots, w^{n_j+1}_{n+1}\).
8. Compute \(w^{n+1}_{n+1}\) and \(w^{n+1}_{n+1}\) according to definition (4.1).

---

**Fig. 4.3. Action of Loop 4.2 (Look-ahead for BiOStab) in the \(w\)-table.** An inner step (at left) and the following regular step (at right) are shown.

---

into one recurrence for these \(h_{j-1}\) columns of \(W^{j+1}_{j-1}\) and then post-multiplied by \(D^{-1}_{j-1}\ell\), as in (4.1).

Again, (4.9) can be simplified: if we define for block \(k - 1\) and the needed values of \(j\) the auxiliary vector

\[
W^{k-1}_{j-1} \beta''_l := W^{k-1}_{j-1} D^{-1}_{k-1} \ell
\]

and for each \(l\) with \(n_k \leq l < n_{k+1}\) the same coefficient \(\beta'_l := q^{n_k}_s \gamma_{n_k-1}\) as in (4.6), then in view of (2.17) and (4.6),

\[
W^{k-1}_{j-1} \beta_l = W^{k-1}_{j-1} D^{-1}_{k-1} \ell \beta''_l = W^{k-1}_{j-1} \beta'_l.
\]
**Loop 4.3. (Look-ahead for BIOS)**

**Inner Loop: \( n_j < n + 1 < n_{j+1} \)**

1. Compute \( A_{n_j} \).
2. If \( n \geq n_j + 2 \), use (4.2) to compute indirectly \( A_{n_j}, \ldots, A_{n_{j+1}} \).
3. Use (4.2) to compute \( w_{n_j}^{n+1}, w_{n_j+1}^{n+1}, \ldots, w_{n_{j+1}}^{n+1} \).
4. Compute \( A_{n_j+1} \) and \( A_{n_j+1} \).
5. Use (4.13) to compute \( w_{j+1}^{n+1} \).
6. Use (4.2) to compute \( w_{n_j+1}^{n+1} \).

**Regular Loop: \( n + 1 = n_{j+1} \)**

1. Compute \( A_{n_{j+1}} \).
2. If \( n \geq n_j + 2 \), use (4.2) to compute indirectly \( A_{n_j}, \ldots, A_{n_{j+1}} \).
3. Use (4.2) to compute \( w_{n_j}^{n+1}, w_{n_j+1}^{n+1}, \ldots, w_{n_{j+1}}^{n+1} \).
4. Compute \( A_{n_j+1} \) and \( A_{n_j+1} \).
5. Use (4.13) to compute \( w_{j+1}^{n+1} \).
6. Use definition (4.1) to compute \( w_{j}^{n+1}, \ldots, w_{j}^{n_{j+1}+1} \) and use definition (4.11) to compute \( w_{j}^{n+1} \).
7. Use (4.2) to compute \( w_{n_{j+1}}^{n+1} \).
8. Use definition (4.1) to compute \( w_{j}^{n_{j+1}+1} \).

---

**Fig. 4.4. Action of Loop 4.3 (Look-ahead for BIOS) in the w-table. An inner loop (at the top) and the following regular loop (at the bottom) are shown.**
Consequently, (4.9) simplifies to

\[
(4.13) \quad w_{j+1}^{h+1} = \left( Aw_j^{h} - \tilde{W}_j^{h} \alpha_i - w_{j-1}^{h-1} \beta_i \right) \beta_i.
\]

A step of the resulting LA-BIOS algorithm is summarized as Loop 4.3.

Here, the decision between a regular and an inner loop is made in substep 3 (which corresponds to substep 5 in Loops 4.1 and 4.2). Now even the first five substeps of the two versions of the loop are identical. In Figure 4.4 we display for LA-BiOS the same pair of loops as we did in Figure 4.3 LA-BiOStab.

For a look-ahead step of length \( h_j \), the cost in terms of MVs is now \( 3h_j \) MVs if \( h_j > 1 \) and \( h_{j-1} > 1 \), while it is only \( 3h_j - 1 \) if \( h_{j-1} = 1 \). As the standard, non-look-ahead algorithm requires \( 2 \) MVs per step, this means that the overhead is at most 50%. Here, in the first substep of the \( h_j - 1 \) inner and the one regular loop \( h_j \) MVs are consumed, and another \( 2h_j \) MVs are needed in substep 4.

### 4.4. Overhead for Look-Ahead

In this subsection we further discuss the overhead of the look-ahead process in terms of MVs, inner products (IPs) and the necessary storage of \( N \)-vectors.

First we note that all IPs required in the look-ahead algorithms have the fixed initial vector \( z_0 \) as their first argument. Therefore, if the second argument of an IP was computed by a recurrence formula, then also the IP of this vector with \( z_0 \) can be computed indirectly applying the same recurrence formula. Thus, only IPs of the form \( \langle z_0, Aw_n^h \rangle \), for which \( Aw_n^h \) is computed directly, need to be computed explicitly. This means that in all algorithms the number of required IPs is equal to the number of required MVs. We must admit, however, that such recursively computed inner products, as well as the recursively computed matrix-vector products, may be the source of additional roundoff, which may cause instability.

Actually, for understanding how to compute all the inner products needed, the reader may want to introduce a \( \delta \)-table and a \( \sigma \)-table with entries \( \delta_n^0 := \langle z_0, w_n^h \rangle \) and \( \sigma_n^0 := \langle z_0, Aw_n^h \rangle \), respectively. Our loops and figures about generating the \( w \)-table then hold as well for these two tables. Note that the \( \delta \)-table just contains the transposed of the matrix \( D \).

<table>
<thead>
<tr>
<th>method</th>
<th>total cost (MVs, IPs)</th>
<th>cost overhead (MVs, IPs)</th>
<th>relativ</th>
<th>storage overhead (N-vectors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3, 3)-type LA-LTPM</td>
<td>( 4h_j - 3 )</td>
<td>( 2h_j - 3 )</td>
<td>25%-100%</td>
<td>( 2h_j^2 + 3h_j + 5 )</td>
</tr>
<tr>
<td>LA-BiOS</td>
<td>( 4h_j - 3 )</td>
<td>( 2h_j - 3 )</td>
<td>25%-100%</td>
<td>( 2h_j^2 + 3h_j + 3 )</td>
</tr>
<tr>
<td>LA-BiOS (if ( h_{j-1} &gt; 1 ))</td>
<td>( 3h_j )</td>
<td>( h_j )</td>
<td>50%</td>
<td>( h_j^2 + 4h_j + 4 )</td>
</tr>
<tr>
<td>LA-BiOS (if ( h_{j-1} = 1 ))</td>
<td>( 3h_j - 1 )</td>
<td>( h_j - 1 )</td>
<td>25%-50%</td>
<td>( h_j^2 + 4h_j + 4 )</td>
</tr>
</tbody>
</table>

In terms of MVs, the cost of a look-ahead step of length \( h_j > 1 \) has been specified in the previous subsection. By subtracting the cost for \( h_j \) non-look-ahead steps, that is \( 2h_j \) MVs, we obtain the overhead summarized in Table 4.1. We stress that when
$h_j = 1$ our algorithm has no overhead except for the necessary test of regularity, which, if it fails, would reveal an upcoming instability and initiate a look-ahead step. The table lists additionally the overhead in storage of $N$-vectors in a straightforward implementation that is not optimized with respect to memory usage.

For comparison, we cite from page 60 of [5] or page 180 of [6] that the CGS look-ahead procedure of Brezinski and Redivo Zaglia requires $6h_j - 3$ MVs if $h_j \leq n_j + 1$ (the typical case) and $5h_j + n_j - 2$ MVs if $h_j > n_j + 1$ (which means that a relatively large look-ahead step is needed in one of the first few iterations). Therefore, compared to our numbers, the overhead in terms of MVs is about four times (if $h_j$ is large, but $h_j \leq n_j + 1$) to five times (if $h_j = 2$) larger than in our LA-BIOS (assuming as the basis a standard CGS implementation requiring 2MV per ordinary iteration). However, we also note that, according to the above numbers, for a step without look-ahead ($h_j = 1$), the methods of [5] and [6] need 3MV instead of 2MV.

5. Look-Ahead Strategies. In this section we address the delicate issue of when to perform a look-ahead step, which means in an LTPM to decide whether the new vertical index $n + 1$ is a regular index or an inner index, i.e., whether the required product vectors $w_{n+1}^t$ in the $(n + 1)$th row of the $w$-table should be computed as regular or as inner vectors. Therefore, the look-ahead procedure in LTPMs serves to stabilize the underlying Lanczos method, the vertical movement of the product method in the $w$-table. Consequently, the criterion, when to carry out a look-ahead step in an LTPM can be based on the criterion given in [15] for the Lanczos algorithm. However, since the Lanczos vectors $\tilde{y}_n$ and $y_n$ are not computed explicitly in a product method, we need to rewrite the conditions of this criterion in terms of the product vectors $w_{n}^t$. Let us first motivate these conditions.

In the case of an exact breakdown, where $0 = \delta_n^* = \langle \tilde{z}_0, w_n^t \rangle = \langle \tilde{z}_n, y_n \rangle$ and $\tilde{z}_n \neq 0, y_n \neq 0$, a division by zero would occur in the next Lanczos step. The first task of the look-ahead process is to circumvent these exact breakdowns without the necessity of restarting the Lanczos process and loosing its superlinear convergence.

In finite precision arithmetic, exact breakdowns are very unlikely. However, near breakdowns, where $|\delta_n^*|$ is very small, may occur and cause large relative roundoff errors in the Lanczos coefficients $\alpha_n$ and $\beta_n$ given by (3.2). To be more precise, we recall that the relative roundoff error in the computation of the inner product $\delta_n^* = \langle \tilde{z}_0, w_n^t \rangle$ is bounded by [18, p. 64]

$$
\frac{|\delta_n^* - \delta_n^*|}{|\delta_n^*|} \leq 1.01 N \varepsilon \frac{|\langle \tilde{z}_0, w_n^t \rangle|}{|\delta_n^*|} \leq 1.01 N \varepsilon \frac{||\tilde{z}_0|| ||w_n^t||}{|\delta_n^*|},
$$

where $\varepsilon$ denotes the roundoff unit. Thus, a small value of $|\delta_n^*|$ leads in finite precision arithmetic to a big relative roundoff error in the computation of the inner product $\delta_n^*$, which also causes a perturbation of the Lanczos coefficients $\alpha_n$ and $\beta_n$, since they depend on $\delta_n^*$ and $\delta_{n-1}^*$ respectively. The second task of the look-ahead process is therefore, to avoid a convergence deterioration due to perturbed Lanczos coefficients. Of course, similar roundoff effects may come up in the numerators of the formulas (3.2) for $\alpha_n$ and $\beta_n$, but large relative errors in those will only be harmful if the denominators are small too.

We would like to point out that in an LTPM the inner products $\delta_n^*$ can be enlarged to a certain extent by an appropriate adaptive choice of the polynomials $\tau_n$ [39, 40], as long as $\langle (\Lambda^H)^n \tilde{z}_0, y_n \rangle \neq 0$. For example, considering the BiOSTab case, where
\( u_n^m = u_{n+m}^m - A u_{n+m-1}^m \chi_{n-1} \), we obtain, since \( \langle \tilde{z}_0, u_n^m \rangle = 0 \):

\[
\frac{\|\tilde{z}_0\|}{\|u_n^m\|} = \frac{\|u_{n+1}^m - A u_{n+1}^m \chi_{n-1}\|}{\|\tilde{z}_0\|} = \frac{\|u_{n+1}^m - A u_{n+1}^m \chi_{n-1}\|}{\|\chi_{n-1}\|}.
\]

Thus, minimizing the relative roundoff error in the calculation of \( \delta_n^* \) is equivalent to choosing \( \chi_{n-1} \) such that it minimizes \( \|u_{n+1}^m - A u_{n+1}^m \chi_{n-1}\| /\|\chi_{n-1}\| \). This leads to \( \chi_{n-1} \colon= \|u_{n+1}^m\|^2 /\|u_{n+1}^m, A u_{n+1}^m\| \), which corresponds to the use of orthogonal residual polynomials of degree 1 instead of minimal residual polynomials of degree 1 in the recursive definition of \( \eta(2) \). Therefore, minimizing the relative roundoff error in the inner product \( \delta_n^* \) conflicts often with the objective of avoiding large intermediate residuals in order to prevent the recursive residual to drift apart from the true residual \([40]\). Performing a look-ahead step is then the only possible remedy.

We need now to find a criterion for deciding when a look-ahead step should be performed so that both the above objectives can be attained. In view of the recursion \((4.2)\) for the product vectors in the case of look-ahead, it follows that a block can be closed and a new product vector can be computed as regular vector only if the diagonal blocks \( D_{j-1} \) and \( D_{j} \) of the Gramian are numerically nonsingular. Thus, the first condition that needs to be fulfilled in order to compute a new product vector \( u_{n+1} \) as regular vector \((n + 1 = n_{j+1})\) is given by

\[
\sigma_{\min}(D_j) \geq \varepsilon,
\]

(5.1)

where \( \sigma_{\min}(D_j) \) denotes the minimal singular value of the block \( D_j \). Note that this does not mean that \( D_j \) is well conditioned; it just guarantees numerical nonsingularity. In practice, \((5.1)\) can be replaced by some other condition that implies this nonsingularity, for example by one implemented in a linear solver used for computing \( D_{j-1} \) in \((2.19)\) and \( \alpha_\ell \) in \((2.22)\).

Freund et al. \([15]\) use a second condition to guarantee that the Krylov space is stably extended in the next Lanczos step in the sense that the basis of Lanczos vectors is sufficiently well conditioned. In terms of product vectors, this second condition amounts to computing, for any \( \ell \), the new product vector \( u_{n+1}^\ell \) as regular vector if in addition to \((5.1)\) the following conditions for the coefficients \( a_n \) and \( \beta_n \) are fulfilled:

\[
\|a_n\|_1 \leq n(A), \quad \|\beta_n\|_1 \leq n(A).
\]

(5.2)

Here \( \|\cdot\|_1 \) denotes the \( \ell_1 \)-norm and \( n(A) \) is an estimate for \( \|A\| \) which is updated dynamically to ensure that the blocks \( W_j^n \) do not become larger than a user specified maximal size \([15]\). The motivation for \((5.2)\) was to ensure that in the new regular vector

\[
u_{n+1}^\ell = (A u_n^\ell - W_j^n a_n - W_{j-1}^n \beta_n) / \gamma_n = (A u_n^\ell - W_j^n a_n - w_{j-1}^\ell \beta_n^\ell) / \gamma_n
\]

(5.3)

(obtained from \((4.2)\) with \( \tilde{W}_j^n = W_j^n \) since \( n + 1 = n_{j+1} \) is regular) the component in the new direction \( A u_n^\ell \) is sufficiently large, which will be the case if

\[
\|A u_n^\ell\| \geq \text{tol}_2 \|u_l\|
\]

where \( u_l := W_j^n a_n + W_{j-1}^n \beta_n = W_j^n a_n - w_{j-1}^\ell \beta_n^\ell \) and \( \text{tol}_2 \) is a chosen tolerance.

Compared to \((5.2)\), condition \((5.3)\) costs additional two inner products and the calculation of \( u_l \), which only in the regular case can be reused for the computation of the new product vector \( u_{n+1}^\ell \). Since we have

\[
\|u_l\| \leq C (\|a_n\| + \|\beta_n\|) \quad \text{with} \quad C := \max \{\|w_{j-1}^\ell\| : \|w_k^\ell\| : n_j \leq k < n_{j+1}\},
\]

(5.4)
we could replace (5.3) by the less expensive condition

\[(5.5) \quad \|Aw_n^0\| \geq \text{tol}_2 C \left(\|\alpha_n\|_1 + |\beta_n^0|\right)\].

However, in an LTPM it is not possible to normalize all product vectors \(u_n^l\) (see Section 6); so \(C\) is not equal to 1. Moreover, (5.5) is less strict than (5.3) and (5.2).

Since a look-ahead step is more expensive than regular steps providing the same increase of the Krylov space dimension, a tight look-ahead criterion can save overall computational cost. Therefore, it is reasonable to spend extra effort for it. For this reason we favor criterion (5.3).

A drawback of (5.3) is that it does not take the angle between \(Aw_n^0\) and \(w_l\) into account. If \(C_{\varepsilon} := 1 - |\langle Aw_n^0, w_l\rangle|/\|Aw_n^0\| \|w_l\|\) is small, \(\text{tol}_2\) in (5.3) should be chosen larger than in the case where it is nearly 1. This motivates the choice

\[(5.6) \quad \text{tol}_2 = \frac{C_1}{1 - (1 - C_2) \frac{|\langle Aw_n^0, w_l\rangle|}{\|Aw_n^0\| \|w_l\|}},\]

with suitably chosen constants \(C_1 = C_1(\varepsilon)\) and \(C_2 = C_2(\varepsilon)\) depending on the roundoff unit \(\varepsilon\). This criterion requires an extra inner product and an appropriate choice for \(C_1\) and \(C_2\). For many small problems \(C_1 = 10^{-3}\) and \(C_2 = 10^{-2}\) worked well, but for larger problems we observed that \(C_{\varepsilon}\) decays dramatically with the block size. Therefore, the probability that (5.3) with \(\text{tol}_2\) as defined in (5.6) will be fulfilled decreases with the block length and leads very often (especially in BiOS) to situations where the maximal user specified block size was reached. Further investigations are needed to see if this problem can be solved by a better choice of \(C_1\) and \(C_2\) or by a more appropriate selection for \(\alpha_n\) in the inner case, instead of using, as in [15], 
\(\alpha_n^0 = 1, \alpha_n^{n-1} = 1, \alpha_n^k = 0\) for \(k = 1, \ldots, n - 2\).

6. Obtaining the solution of \(Ax = b\). So far we have only introduced various algorithms for constructing a sequence of product vectors \(u_n^l\) that provide a basis for \(K_m\). However, our goal is to solve the linear system \(Ax = b\). We now describe how to accomplish this with these algorithms. There are several basic approaches to constructing approximate solutions of linear systems from a Krylov space basis. Ours is related to the Galerkin method, but avoids the difficulty that arises when the Galerkin solution does not exist. (This difficulty causes, for example, the so-called pivot breakdown of the biconjugate gradient method.) Our approach is a natural generalization of the one that lead to “unnormalized BiRes” introduced in [20] and renamed “inconsistent BiRes” in [25]. In contrast to the BiOMin, BiODir, and BiRes versions of the BiCG method, the inconsistent BiRes variant is not endangered by pivot breakdowns. An alternative would be to construct approximate solutions based on the quasi-minimal residual (QMR) approach [16]. For a combination of this approach with LTPMs we refer the reader to [36].

Let the doubly indexed sequence of scalars \(\rho_n^l\) be given by

\[(6.1) \quad \rho_n^l := \tau_l(0) \rho_n(0)\].

We define a doubly indexed sequence of product iterates \(x_n^l\) as follows. Starting with an arbitrary \(x_0^0 \in \mathbb{C}^N\), we choose the initial product vector \(u_0^0\) so that

\[(6.2) \quad u_0^0 = b_0 - Ax_0^0\].
Here, \( \rho_0^n = 1 \) since \( \tau_0(\zeta) = \rho_0(\zeta) \equiv 1 \). For \( l, n > 0 \) the product iterates are now implicitly defined by

\[
(6.3) \quad \tilde{b} \tilde{\rho}_n - A \tilde{x}_n^l := \tilde{u}_n^l, \quad \text{so that} \quad b - A \frac{x_n^l}{\tilde{\rho}_n} = \frac{u_n^l}{\tilde{\rho}_n} \quad \text{if} \quad \tilde{\rho}_n \neq 0.
\]

Of course, \( \tilde{x}_n^l \) will be constructed only when \( \tilde{u}_n^l \) is. If \( \tilde{\rho}_n \neq 0 \), it follows from (6.3) that \( \tilde{x}_n^l / \tilde{\rho}_n \) can be considered as an approximate solution of \( Ax = b \), whose corresponding residual is \( \tilde{u}_n^l / \tilde{\rho}_n \). In order to derive recursions for the scalars \( \tilde{\rho}_n \) and the product iterates \( \tilde{x}_n^l \), we introduce the blocks

\[
X_{\tilde{j}+1}^j := \begin{bmatrix} \tilde{x}_{n_{j+1}} & \cdots & \tilde{x}_{n_{j+1}} \\ \end{bmatrix}, \quad \tilde{X}_{\tilde{j}}^j := \begin{bmatrix} \tilde{x}_{n_j} & \cdots & \tilde{x}_n^l \end{bmatrix},
\]

\[
P_{\tilde{j}+1}^j := \begin{bmatrix} \tilde{\rho}_{n_{j+1}} & \cdots & \tilde{\rho}_{n_{j+1}} \\ \end{bmatrix}, \quad \tilde{P}_{\tilde{j}}^j := \begin{bmatrix} \tilde{\rho}_{n_j} & \cdots & \tilde{\rho}_{n} \end{bmatrix},
\]

as well as the auxiliary product iterates \( \tilde{x}_{\tilde{j}+1}^j \) and auxiliary scalars \( \tilde{\rho}_{\tilde{j}+1}^j \) defined by

\[
(6.4) \quad \tilde{x}_{\tilde{j}+1}^j := X_{\tilde{j}+1}^j D_{\tilde{j}+1}^{-1} \ell, \quad \tilde{\rho}_{\tilde{j}+1}^j := P_{\tilde{j}+1}^j D_{\tilde{j}+1}^{-1} \ell.
\]

Again, \( \tilde{X}_j^l = X_j^l \) and \( \tilde{P}_j^l = P_j^l \) if \( n + 1 \) is a regular index.

Then, by (6.3), (4.1), and (6.4),

\[
(6.5) \quad \tilde{W}_{\tilde{j}}^l = b \tilde{P}_{\tilde{j}}^l - A \tilde{X}_{\tilde{j}}^l, \quad w_{\tilde{j}+1}^l = b \tilde{\rho}_{\tilde{j}+1}^l - A x_{\tilde{j}+1}^l.
\]

Next, using (4.2) we conclude that

\[
\begin{aligned}
b \tilde{\rho}_{n+1}^l - A x_{n+1}^l &= w_{n+1}^l = \left( A \omega_{n}^l - \tilde{W}_{\tilde{j}}^l \alpha_n - w_{\tilde{j}+1}^l \beta_n \right) / \gamma_n \\
&= \left( A \omega_{n}^l - \left[ b \tilde{P}_{\tilde{j}}^l - A \tilde{X}_{\tilde{j}}^l \right] \alpha_n - \left[ b \tilde{\rho}_{\tilde{j}+1}^l - A x_{\tilde{j}+1}^l \right] \beta_n \right) / \gamma_n \\
&= b \left[ -\tilde{P}_{\tilde{j}}^l \alpha_n - \tilde{\rho}_{\tilde{j}+1}^l \beta_n \right] / \gamma_n + A \left[ \omega_{n}^l + \tilde{X}_{\tilde{j}}^l \alpha_n + x_{\tilde{j}+1}^l \beta_n \right] / \gamma_n.
\end{aligned}
\]

This shows that

\[
(6.6) \quad x_{n+1}^l = -\left[ w_{n}^l + \tilde{X}_{\tilde{j}}^l \alpha_n + x_{\tilde{j}+1}^l \beta_n \right] / \gamma_n, \quad \tilde{\rho}_{n+1}^l = -\left[ \tilde{P}_{\tilde{j}}^l + \tilde{\rho}_{\tilde{j}+1}^l \beta_n \right] / \gamma_n.
\]

If we arrange the product iterates \( x_n^l \) and the scalars \( \tilde{\rho}_n \) in two tables analogous to the \( w \)-table (with the \( n \)-axis pointing downwards and the \( l \)-axis to the right), these two recursions can be used to proceed in vertical direction.

To obtain recursions for a horizontal movement, we assume first that the polynomials \( \tau_l(\zeta) \) are given by the normalized three-term recurrence (3.3). This covers all algorithms described in this paper except look-ahead BiOS. Using (3.4) and (6.3) we see that the product iterates satisfy

\[
(6.7) \quad x_{n+1}^{l+1} = -w_{n}^l \eta_l + x_{n}^{l+1} \zeta_l + x_{n}^{l-1} (1 - \zeta_l).
\]

For the scalars \( \tilde{\rho}_n \) the recursion \( \tilde{\rho}_n^{l+1} = \tilde{\rho}_n \zeta_l + \tilde{\rho}_n^{l-1} (1 - \zeta_l) \) are valid, but since the polynomials \( \tau_l \) are normalized (that is, \( \tau_l(0) = 1 \) for all \( l \)), the scalars \( \tilde{\rho}_n \) do not change with the index \( l \), and we have simply \( \tilde{\rho}_n = \rho_0^n = \rho_n(0) \).

In look-ahead BiOS only one horizontal movement is explicitly computed per step, namely in substep 5 of Loop 4.3 based on the recurrence (4.13). If we define in analogy to (4.10) and (4.11)

\[
\begin{aligned}
\tilde{X}_{\tilde{j}-1}^{k} &:= \begin{bmatrix} x_{j-1}^m & \cdots & x_{j-1}^{\tilde{j}-1} \end{bmatrix}, & X_{\tilde{j}-1}^{k} &:= \begin{bmatrix} x_{j-1}^{m-l} & \cdots & x_{j-1}^{\tilde{j}-1} \end{bmatrix}, \\
\tilde{P}_{\tilde{j}-1}^{k} &:= \begin{bmatrix} \tilde{\rho}_{j-1}^m & \cdots & \tilde{\rho}_{j-1}^{\tilde{j}-1} \end{bmatrix}, & P_{\tilde{j}-1}^{k} &:= \begin{bmatrix} \rho_{j-1}^{m-l} & \cdots & \rho_{j-1}^{\tilde{j}-1} \end{bmatrix},
\end{aligned}
\]

then...
and

\[ x_{j-1}^{n+1} := X_{j-1}^{n+1} D_{j-1} \ell , \quad \beta_{j-1}^{n+1} := P_{j-1}^{n+1} D_{j-1} \ell , \]

recurrences for the auxiliary iterates and the corresponding scalars are given by

\[ x_{j-1}^{n+1} = - \left[ w_{j-1}^{n+1} + X_{j-1}^{n+1} \alpha_l + x_{j-1}^{n+1} \beta_l \right] / \gamma, \]

\[ \beta_{j-1}^{n+1} = - \left[ \beta_{j-1}^{n+1} + \beta_{j-1}^{n+1} \beta_l \right] / \gamma. \]

Using for the scaling parameter \( \gamma_n \) in (4.2) the special choice

\[ \gamma_n = \begin{cases} -1^{T} \alpha_{n} - 1^{T} \beta_{n} & \text{if } n + 1 = n_{j+1} \\ -1_{n-n+1}^{T} \alpha_{n} - 1_{n-j+1}^{T} \beta_{n} & \text{if } n_{j} < n + 1 = n_{j+1}, \end{cases} \]

with \( 1_{m} := [1 \cdots 1]_{T} \), also the Lanczos polynomials \( \rho_{n} \) could be normalized \( \rho_{n}(0) = 1 \), so that \( \tilde{\rho}_{0} = 1 \) for all \( n, l \geq 0 \). However, as we mentioned before, some \( \gamma_n \) might turn out to be zero, which would lead to a so-called pivot breakdown. Moreover, to avoid overflow or underflow, in the Lanczos process the scaling parameter \( \gamma_n \) is often used to normalize the Lanczos vectors \( y_n \). But since the Lanczos vectors \( y_n \) are not explicitly computed in an LTPM, we cannot base the choice of \( \gamma_n \) here on their norm. However, independent of the size of the blocks generated by the look-ahead process, it is always necessary to compute the product vectors \( w_{n+1}^{n+1} \) in an LTPM. Therefore, we choose here \( \gamma_n \) to normalize \( w_{n+1}^{n} \), that is,

\[ \gamma_n := \| A w_{n}^{n} - \tilde{W}_{n}^{n} \alpha_{n} - W_{j-1}^{n} \beta_{n} \|, \quad \text{if } n_{j} < n + 1 \leq n_{j+1}. \]

7. Numerical Examples. In this section we demonstrate the practical performance of our look-ahead versions of LTPMs in numerical examples. The tests are restricted to BiOStab, BioOxMR2 and BiOS. The look-ahead versions of these LTPMs are denoted by LABIOStab, LABIOxMR2, and LABIOS, respectively. For all tests the initial iterate \( x_0 = 0 \) is used, and the iteration is terminated when the norm of the recursive residual is less than \( \sqrt{\varepsilon} \), the square root of the roundoff unit. The test programs were written in FORTRAN90/95 and run on workstations with 64-bit IEEE arithmetic. We start with small, artificially constructed model problems and move gradually to large real-world problems.

Example 1. The following small test example was proposed by Joubert [29] and also used by Brezinski and Redivo-Zaglia [6]:

\[ A := \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 1 & 3 \end{bmatrix}, \quad b := \begin{bmatrix} 0 \\ 2 \\ 2 \\ 4 \end{bmatrix}, \]

\( x_0 = 0 \), and \( z_0 = [1,1,1,1]^T \). The Lanczos process, and hence, BiOStab, BioOxMR2, and BiOS without look-ahead break down at step 2. On the contrary, all look-ahead versions avoid this breakdown and converge after 4 iterations as shown in our plots of the true residual norms \( \| b - A x_n^{n+1} \| / \| b \| \) in Figure 7.1.
Example 2. Our second example is taken from [6], Example 5.2: the matrix

\[
A := \begin{bmatrix}
2 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 2 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 2 & 1 & 0 & 0 & 1 \\
1 & 0 & 2 & 1 & 1 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

of order 400 and the right-hand side \( b = (3, 3, 4, \ldots, 4, 3, 3)^T \) imply that the solution is \( x = (1, 1, \ldots, 1)^T \). With \( x_0 = 0, z_0 = (0, 0, 0, -1, 1, \ldots, 0)^T \) the Lanczos process, and thus the LTPMs break down in the first iteration. Our look-ahead versions perform two inner steps at the first and second iteration. All following iterations are regular steps. This is a typical behavior: there are only few and short look-ahead steps, and therefore the resulting mean overhead per iteration is nearly negligible.

Fig. 7.2. The true residual norm history (i.e. \( \log(||b - Ax_n - \frac{1}{\mu_n}||) \) vs. \( n \)) for the linear system defined in (7.2) solved by different LTPMs with look-ahead.
### Table 7.1

Indices of regular steps in LTPMs for three problems with a p-cyclic system matrix.

<table>
<thead>
<tr>
<th>Example</th>
<th>LABiOStab</th>
<th>LABiOxMR2</th>
<th>LABiOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1,5,6,10,11,15</td>
<td>1,5,6,10,11,15</td>
<td>1,5,6,10,11,15</td>
</tr>
<tr>
<td>4</td>
<td>1,4,5,8,9,12,13,16</td>
<td>1,4,5,8,9,12,13,16</td>
<td>1,4,5,8,9,12,13,16</td>
</tr>
<tr>
<td>5</td>
<td>1,8,9,16,17</td>
<td>1,8,9,16,17</td>
<td>1,8,9,16,17</td>
</tr>
</tbody>
</table>

In the next set of examples we consider p-cyclic matrices of the form

\[
A := \begin{bmatrix}
I_1 & B_1 \\
B_2 & I_2 \\
& \vdots \\
& \vdots \\
B_p & I_p
\end{bmatrix}.
\]

Hochbruck [27] showed that the computational work for solving a linear system with a p-cyclic system matrix by QMR with look-ahead can be reduced by approximately a factor \(1/p\) (compared to a straight-forward implementation using sparse matrix-vector multiplications with \(A\)), if the initial Lanczos vectors have only one nonzero block conforming to the block structure of \(A\), if the inner vectors are chosen so that the nonzero structure of \((A - I)y_n\) is not destroyed, and if the blocks \(B_k\) are used for generating only possibly nonzero components of the Krylov space basis. Then it can be proven that in each cycle of \(p\) steps there are at least \(p - 2\) consecutive exact breakdowns for \(p > 2\). But when using directly the system matrix \(A\) to generate the Krylov subspace, we have only in the first cycle of \(p\) steps \(p - 2\) consecutive exact breakdowns, while in the following cycles these will, in general, no longer persist, but must be expected to become near-breakdowns. Therefore, such problems provide good test examples for look-ahead algorithms.

**Example 3.** In this example we consider a 5-cyclic matrix with \(B_k = B\) for \(k = 1, \ldots, 5\), where \(B\) is a randomly generated \(10 \times 10\) matrix. As right-hand side and as initial left Lanczos vector we choose

\[
b = \begin{bmatrix} f_1 \\ 0 \end{bmatrix}, \quad \bar{z}_0 = \begin{bmatrix} g_1 \\ 0 \end{bmatrix},
\]

where \(f_1, g_1\) have random entries. The convergence history for the different LTPMs applied to this problem are shown in Figure 7.3, and the indices of the regular steps are listed in Table 7.1. Those are found exactly where predicted.

**Example 4.** We move now to a bigger 4-cyclic system matrix with \(B_k = B\) for \(k = 1, \ldots, 4\), where \(B\) is a \(100 \times 100\) matrix with random entries. The results for this problem are shown in Figure 7.4, and the indices of the regular steps are depicted also in Table 7.1. Again, they occur where predicted.

**Example 5.** Finally, we consider an 8-cyclic system matrix with \(B\) defined as in Example 4. The convergence history plotted in Figure 7.5 shows oscillations in the residual norm history of LABiOS, but overall LABiOS needs one iteration step less than LABiOStab and LABiOxMR2 to fulfill the convergence condition. For all methods the same look-ahead criterion ((5.3) with \(tol_2\) defined as in (5.6) and \(C_1 = 10^{-3}, C_2 = 10^{-2}\)) is used. Especially for LABiOS the correct choice of the look-ahead criterion seems to be crucial. While with the above values of \(C_1\) and \(C_2\) the breakdowns occurred only where expected, we discovered for this larger problem...
that in BiOS the maximal user specified block length of 10 was reached very often for \( C_1 \gg 10^{-3} \) and \( C_2 \ll 10^{-2} \), which indicates, that the constructed inner vectors become more and more linear dependent. Therefore, further investigations are needed to figure out a better choice for the coefficient vector \( \alpha_n \) in the inner case. For example, following the proposal in [21], Hochbruck used in [27] a Chebyshev iteration for the generation of the inner vectors instead of \( \alpha_1^n = 1, \alpha_{n-1}^n = 1 \) and \( \alpha_k^n = 0 \) for \( k = 1, \ldots, n - 1 \), which we adapted from [15].

Example 6. In our last example we take a real world problem from the Harwell-
Boeing Sparse Matrix Collection, namely SHERMAN1, a matrix of order 1000 with 3750 nonzero entries. The right-hand side \( b \) and the initial left Lanczos vector \( z_0 \) were generated as different unit vectors with random entries. Without look-ahead, all LTPMs introduced here break down (BiOStab at step 186, BiOStab2 at step 176, BiOxMR2 at step 145 and BiOS at step 352). On the contrary, the look-ahead versions in combination with the look-ahead criterion (5.3) with tol2 defined as in (5.6) and \( C_1 = 10^{-3} \) and \( C_2 = 10^{-2} \) converge as shown in Figure 7.6. Due to the real spectrum of the system matrix \( A \), there is only a slight difference in the convergence of LABioStab and LABioxMR2. It was reported to us that BiCGStab(\( \ell \)) with \( \ell = 8 \) and no look-ahead can handle this problem in about the same number of MVs as our BiCGStab with look-ahead.

8. Conclusions. We have proposed look-ahead versions for various Lanczos-type product methods that make use of the Lanczos three-term recurrences. Since they are based on the Lanczos look-ahead version of Gutzwecht [23] and Freund et al. [15], they can handle look-ahead steps of any length and avoid steps that are longer than needed. The algorithms proposed in this work should be easy to understand due to the introduction of an array of product vectors, symbolically displayed in the \( w \)-table, and the visualization of the progress in this \( w \)-table. Furthermore, the \( w \)-table proved to be a useful tool to derive optimal variants, for which the computational work in terms of MVs is minimized.

A variety of numerical examples demonstrate the practical performance of the proposed algorithms. However, larger problems indicate that further work should be directed to finding an improved look-ahead criterion that more reliably avoids critical perturbations of the Lanczos coefficients by roundoff errors. Moreover, one should investigate if there is a better way of constructing inner vectors than the choice adapted from [15]: Alternatives would be to orthogonalize them within each block [2] or to construct them by Chebyshev iteration [21]; but it is not clear if the additional cost involved pays off.
The look-ahead process in an LTPM stabilizes primarily the vertical movement in the \( w \)-table, except in the BioS algorithm where the \( w \)-table is symmetric. For the horizontal movement it is also important to generate the Krylov space stably, and both BioStab2 and BioxMR2 (in particular when suitably modified) do that more reliably than BiCGStab, since the two-dimensional steps offer more flexibility. This has also a lasting positive effect on the rounding in the vertical movement. To stabilize the horizontal movement further, a local minimal residual polynomial of degree \( \ell \geq 1 \) with an adaptive choice of \( \ell \), as in BiCGStab(\( \ell \)) could be used. A further possibility is to adapt \( \ell \) to the size \( h_j \) of the current Lanczos block, which would mean to perform in each regular step an \( h_j \)-dimensional local minimization of the residual. An alternative is to trade in the local residual minimization for a more stable Krylov space generation whenever the former causes a problem. Yet another possibility, indicated in Section 4.2, is to combine the Lanczos process with a hybrid Chebyshev iteration.

It is known that in finite-precision arithmetic BiOREs is usually more affected by rounding than the standard BiOMin version of BiCG, at least with regard to the gap between recursively and explicitly computed residuals. Therefore, we are in the process to extend this work to look-ahead procedures for LTPMs that are based on coupled two-term recurrences.

REFERENCES


[28] ———, The Padé table and its relation to certain numerical algorithms, Habilitationsschrift, Universität Tübingen, Germany, 1996.


