A Lanczos-type method for solving nonsymmetric linear systems with multiple right-hand sides—matrix and polynomial interpretation

C. Musschoot
Laboratoire d’Analyse Numérique et d’Optimisation, UFR IEEA - M3, Université des Sciences et Technologies de Lille, 59655 Villeneuve d’Ascq Cedex, France
Received 9 June 1998

Abstract

In this paper, we propose a method with a finite termination property for solving the linear system $AX = B$ where $A$ is a nonsymmetric complex $n \times n$ matrix and $B$ is an arbitrary $n \times s$ rectangular matrix. $s$ does not have to be small. The method is based on a single Krylov subspace where all the systems are picking informations. A polynomial and a single matrix interpretation is given which seems to be new from a theoretical point of view. Numerical experiments show that the convergence is usually quite good even if $s$ is relatively large. The memory requirements and the computational costs seem to be interesting too. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Nonsymmetric systems; Krylov subspace; Multiple right-hand sides; Lanczos’ method; Bi-orthogonality; Hankel matrix; Orthogonal polynomials; Transpose-free algorithm; BiCGSTAB

1. Introduction

Two main techniques can be found for solving

$$AX = B$$  \hspace{1cm} (1)

where $A$ is a $n \times n$ square nonsymmetric matrix and $B$ is an arbitrary $n \times s$ rectangular matrix.

The first one considers block versions of existing methods such as block GCR (see [12]), block BiCG (see [11]), block GMRES (see [16, 17]) or block QMR (see [14]). All these methods need the inversion of an $s \times s$ matrix per iteration. So $s$ must be relatively small.

The second one avoids this problem. Generally a seed system is considered and the results obtained for solving this single system are used to solve the other ones (see [7]). Other methods have recently been proposed. An iterative method is considered in [15] and requires a general eigenvalue problem to be solved. This method shares the informations obtained for solving each system, in order to get a faster convergence.
In this paper, we use a single seed system to solve (1) based on a modified Lanczos’ method that will lead to transpose-free algorithms and to a modified BiCGSTAB, inspired from van der Vorst [19]. All these methods will allow a single matrix and a polynomial interpretation.

Section 2 briefly recalls the Lanczos’ method for a single right-hand side. Section 3 describes the modifications that have to be made for several right-hand sides and considers the BiCGSTAB with these modifications. Section 4 presents some numerical results and the conclusions stands in Section 5.

2. Lanczos’ method and its implementations

As the new method will be based on the Lanczos’ method, we will first introduce this method [10] and its implementations by means of orthogonal polynomials.

2.1. Lanczos’ method

Let us consider the system

\[ Ax = b, \]

where \( A \) is a \( n \times n \) square matrix and \( b \in \mathbb{C}^n \). Lanczos’ method constructs two sequences \( r_k \) and \( x_k \) such that

\[ x_k - x_0 \in K_k(A, r_0) = \text{span}(r_0, Ar_0, \ldots, A^{k-1}r_0), \]

\[ r_k = b - Ax_k \perp K_k(A^*, y) = \text{span}(y, A^*y, \ldots, A^{*k-1}y), \]

where \( x_0 \) and \( y \) are arbitrarily chosen. Usually, \( y = r_0 \).

From that, we have the

**Property 2.1.** Let us assume that the vectors \( y, A^*y, \ldots, A^{*n-1}y \) are linearly independent. Then

\[ \exists k \leq n, \quad r_k = 0. \]

By the definitions of \( x_k \) and \( r_k \), we obtain

\[ x_k - x_0 = -x_1r_0 - x_2Ar_0 - \cdots - x_kA^{k-1}r_0. \]

So,

\[ r_k = r_0 + x_1Ar_0 + \cdots + x_kA^{k-1}r_0 \]

\[ = P_k(A)r_0, \]

where \( P_k \) is the polynomial of degree \( k \) at most, defined by

\[ P_k(x) = 1 + x_1x + \cdots + x_kx^k. \]

Thus, the orthogonality conditions (3) can be written as

\[ x_i(y, A^{i+1}r_0) + \cdots + x_k(y, A^{i+k}r_0) = -(y, A^i r_0) \quad \text{for} \quad i = 0, \ldots, k - 1. \]
The vector \((x_1, \ldots, x_k)^T\) is the solution of the Hankel system of order \(k\) generated by \(((r_0, A^* y), \ldots, (r_0, A^{*k-1} y))^T\) with the right-hand side \((- (r_0, y), \ldots, -(r_0, A^{*k-1} y))^T\), that is the solution of the system

\[
\begin{pmatrix}
(r_0, A^* y) & (r_0, A^{*2} y) & \cdots & (r_0, A^{*k} y) \\
(r_0, A^{*2} y) & \vdots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
(r_0, A^{*k} y) & \cdots & \cdots & (r_0, A^{*2k-1} y)
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_k
\end{pmatrix}
=
\begin{pmatrix}
(r_0, y) \\
(r_0, A^* y) \\
\vdots \\
(r_0, A^{*k-1} y)
\end{pmatrix}.
\]

2.2. Formal orthogonal polynomials

We now have to introduce the notion of formal orthogonal polynomials, fully studied by Draux in [8]. Let \(c\) be the linear functional defined on the space of polynomials by

\[c(x^i) = c_i = (y, A^i r_0) \quad \text{for all } i \geqslant 0.\]

Then, for all polynomial \(P\), we have \(c(P) = (y, P(A)r_0)\) and the conditions (4) and thus, the orthogonality conditions (3) become

\[c(x^i P_k) = 0 \quad \text{for } i = 0, \ldots, k - 1. \tag{5}\]

A family of polynomials satisfying the conditions (5) is called the family of orthogonal polynomials with respect to the functional \(c\). These polynomials are defined apart a multiplying factor chosen in our case to verify \(P_0(0) = 1\).

Such polynomials satisfy, if they exist for all \(k\), a three-term recurrence relationship. Moreover, we can show that \(P_k\) exists if and only if

\[H_k^{(1)} = \begin{vmatrix}
c_1 & \cdots & c_k \\
\vdots & \ddots & \vdots \\
c_k & \cdots & c_{2k-1}
\end{vmatrix} \neq 0,
\]

and that it is of the exact degree \(k\) if and only if

\[H_k^{(0)} = \begin{vmatrix}
c_0 & \cdots & c_{k-1} \\
\vdots & \ddots & \vdots \\
c_{k-1} & \cdots & c_{2k-2}
\end{vmatrix} \neq 0.
\]

In the sequel, we will assume these two conditions hold for all \(k\).

Then, we can define the family of adjacent polynomials \(P_k^{(1)}\), orthogonal with respect to the functional \(c^{(1)}\) defined by

\[c^{(1)}(x^i) = c(x^{i+1}) = c_{i+1}.
\]

These polynomials are chosen to be monic and, so, they exist if and only if \(H_k^{(1)} \neq 0\). Thus, the condition for the existence of \(P_k^{(1)}\) and \(P_k\) is the same.
2.3. Implementations of Lanczos’ method

All the possible algorithms for implementing Lanczos’ method come out from the recurrence relations between the families \( \{P_k\} \) and \( \{P_k^{(1)}\} \). The three main algorithms are named Lanczos/Orthodir, Lanczos/Orthomin and Lanczos/Orthores and we will now remind them.

2.3.1. Lanczos/Orthodir

The orthogonal polynomials \( P_k^{(1)} \) satisfy a three-term relation of the form

\[
P_{k+1}^{(1)}(x) = (x - \alpha_k)P_k^{(1)}(x) - \beta_k P_{k-1}^{(1)}(x).
\]

In the same way, we can prove that the polynomials \( P_k \) satisfy the relation

\[
P_{k+1}(x) = P_k(x) - \lambda_k x P_{k-1}(x).
\]

Setting \( q_k = P_k^{(1)}(A)r_0 \), the first algorithm follows

**Algorithm 2.2** (Lanczos/Orthodir).

\[
\begin{align*}
q_{k+1} &= (A - \alpha_k)q_k - \beta_k q_{k-1}, \\
r_{k+1} &= r_k - \lambda_k Aq_k, \\
x_{k+1} &= x_k + \lambda_k q_k,
\end{align*}
\]

with

\[
\begin{align*}
\alpha_k &= \frac{(y, A^2 U_k(A)q_k) - \beta_k (y, AU_k(A)q_{k-1})}{(y, AU_k(A)q_k)}, \\
\beta_k &= \frac{(y, A^2 U_{k-1}(A)q_k)}{(y, AU_{k-1}(A)q_{k-1})}, \\
\lambda_k &= \frac{(y, U_k(A)r_k)}{(y, AU_k(A)q_k)},
\end{align*}
\]

where \( U_k \) and \( U_{k-1} \) are two arbitrary polynomials of respective degrees \( k \) and \( k - 1 \). This algorithm is called Lanczos/Orthodir if \( U_k \) and \( U_{k-1} \) are, respectively, \( P_k^{(1)} \) and \( P_{k-1}^{(1)} \).

2.3.2. Lanczos/Orthomin

Setting \( Q_k = \tilde{\beta}_k P_k^{(1)} \) with \( \tilde{\beta}_k \) such that \( P_k \) and \( Q_k \) have the same leading coefficient, we can show that the polynomials \( Q_k \) satisfy the relation

\[
Q_{k+1}(x) = P_{k+1}(x) - \alpha_k Q_k(x).
\]

So, setting \( \tilde{q}_k = Q_k(A)r_0 \), we obtain the
Algorithm 2.3 (Lanczos/Orthomin).

\[ r_{k+1} = r_k - \lambda_k A \bar{q}_k, \]
\[ x_{k+1} = x_k + \lambda_k \bar{q}_k, \]
\[ \bar{q}_{k+1} = r_{k+1} - \alpha_k \bar{q}_k, \]

with

\[ \alpha_k = \frac{(y, U_{k+1}(A)r_{k+1})}{(y, AU_k(A)\bar{q}_k)}, \]
\[ \lambda_k = \frac{(y, U_k(A)r_k)}{(y, AU_k(A)\bar{q}_k)} \]

This algorithm is called Lanczos/Orthomin for the choices \( U_k \equiv P_k \) and \( U_{k+1} \equiv P_{k+1} \).

2.3.3. Lanczos/Orthores

The polynomials \( P_k \) satisfy a three-term relation that can always be written as

\[ P_{k+1}(x) = \alpha_k[(x + \beta_k)P_k(x) - \gamma_k P_{k-1}(x)]. \]

This relation gives us the

Algorithm 2.4 (Lanczos/Orthores).

\[ r_{k+1} = \alpha_k[(A + \beta_k) r_k - \gamma_k r_{k-1}], \]
\[ x_{k+1} = \alpha_k[\beta_k x_k - \gamma_k x_{k-1} - r_k], \]

with

\[ \alpha_k = \frac{1}{\beta_k - \gamma_k}, \]
\[ \beta_k = \frac{\gamma_k(y, U_k(A)r_{k-1}) - (y, AU_k(A)r_k)}{(y, AU_k(A)r_k)}, \]
\[ \gamma_k = \frac{(y, AU_{k-1}(A)r_k)}{(y, U_{k-1}(A)r_{k-1})} \]

This algorithm is called Lanczos/Orthores for the choice \( U_k \equiv P_k \) and \( U_{k-1} \equiv P_{k-1} \).

3. Several right-hand sides

Let us now consider

\[ AX = B, \]

where \( B = [(b^{(1)}, \ldots, b^{(s)})] \) is a matrix of dimension \( n \times s \).
3.1. Description of the method

Let us consider, 2s sequences \( r_k^{(j)} \) and \( x_k^{(j)} \) for \( j = 1, \ldots, s \) defined by

\[
x_k^{(j)} \in K_k(A, z),
\]

\[
r_k^{(j)} = b^{(j)} - Ax_k^{(j)} \perp K_k(A^*, y),
\]

where \( z \) and \( y \) are arbitrarily chosen vectors.

Then, we find from (6)

\[
x_k^{(j)} = z_1^{(j)} + \cdots + z_k^{(j)} A^{k-1} z.
\]

Thus,

\[
r_k^{(j)} = b^{(j)} - Ax_k^{(j)}
= b^{(j)} - z_1^{(j)} A z - \cdots - z_k^{(j)} A^k z
= b^{(j)} - A \Phi_k^{(j)}(A) z,
\]

where \( \Phi_k^{(j)} \) is the polynomial of degree \( k - 1 \) at most, defined by

\[
\Phi_k^{(j)}(x) = z_1^{(j)} + z_2^{(j)} x + \cdots + z_k^{(j)} x^{k-1}.
\]

The orthogonality conditions (7) can be written as

\[
x_1^{(j)} (A^{i+1} z, y) + \cdots + x_k^{(j)} (A^{i+k} z, y) = (A^i b^{(j)}, y) \quad \text{for} \quad \left\{ \begin{array}{l} i = 0, \ldots, k - 1, \\
 j = 1, \ldots, s. \end{array} \right.
\]

The unknown vectors \( (z_1^{(j)}, \ldots, z_k^{(j)})^T \) are the solutions of the Hankel systems generated by

\[
((A z, y), \ldots, (A^{k-1} z, y))^T
\]

with the right-hand sides \( ((b^{(j)}, y), \ldots, (A^{k-1} b^{(j)}, y))^T \) for \( j = 1, \ldots, s \).

This matrix interpretation would not be possible considering the original Lanczos’ method (we only have one Hankel matrix here with several right-hand sides). In the original Lanczos’ method, the Hankel matrix depends on \( r_0^{(j)} \).

3.2. Finite convergence

As in the original Lanczos’ method and under a certain assumption, we can show that the new method gives the exact solution in a finite number of iterations.

The condition is the same as in the original Lanczos’ method and we have the

**Proposition 3.1.** Let us assume that the vectors \( y, A^* y, \ldots, A^{*n-1} y \) are linearly independent. Then,

\[
\exists k \leq n, r_k^{(j)} = 0 \quad \text{for} \quad j = 1, \ldots, s.
\]

**Proof.** By the orthogonality conditions, we have \( r_k^{(j)} \perp (y, A^* y, \ldots, A^{*n-1} y) \) which are \( n \) linearly independent vectors. Then the result is obvious and the exact solutions are obtained in \( n \) iterations at most. \( \square \)
3.3. Implementation of the new method

Let us now explore the possibilities for implementing this method. We will first define the functionals that have to be used. Then we will construct the sequences that lead to the solution of the systems.

3.3.1. Associated functionals—polynomial expression

Let us consider the $n \times s$ linear functionals defined on the space of polynomials by

$$L_{i}^{(j)}(x^l) = (A^{i+l}z, y) \quad \text{if} \quad l > 0,$$

$$L_{i}^{(j)}(1) = -(A^i b^{(j)}, y)$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, s$.

Let us define, as in the original Lanczos’ method, the functional $c^{(1)}$ by

$$c^{(1)}(x^i) = c_{i+1} = (A^{i+1}z, y) \quad \text{for} \quad i \geq 0.$$  \hspace{1cm} (9)

Let the polynomials $\tilde{P}_{k}$ be the orthogonal polynomials with respect to the functional $c^{(1)}$. These polynomials satisfy

$$c^{(1)}(x^k \tilde{P}_k) = 0 \quad \text{for} \quad i = 0, \ldots, k - 1.$$  \hspace{1cm} (10)

We have the

**Proposition 3.2.** The functionals $L_{i}^{(j)}$ and $c^{(1)}$ are related by

$$c^{(1)}(x^i) = L_{i}^{(j)}(x^{i-l+1}) \quad \text{for} \quad i - l + 1 > 0.$$  \hspace{1cm} (11)

**Proof.** By definition, $L_{i}^{(j)}(x^{i-l+1}) = (A^{i+l}z, y)$ since $i - l + 1 \neq 0$. Thus, we have $L_{i}^{(j)}(x^{i-l+1}) = (A^{i+1}z, y) = c^{(1)}(x^i)$. \hspace{1cm} \square

Let the polynomials $\tilde{P}_k$ be monic. Then, from [8], we have the

**Property 3.3.** The monic orthogonal polynomials $\tilde{P}_k$ with respect to the functional $c^{(1)}$ satisfy the relation

$$\tilde{P}_{k+1}(x) = (x - \alpha_k)\tilde{P}_k(x) - \beta_k \tilde{P}_{k-1}(x),$$  \hspace{1cm} (12)

where

$$\beta_0 = 0,$$

$$\beta_k = \frac{c^{(1)}(x^k \tilde{P}_k)}{c^{(1)}(x^{k-1} \tilde{P}_{k-1})} \quad \text{for} \quad k > 0,$$

$$\alpha_0 = \frac{c^{(1)}(x)}{c^{(1)}(1)},$$

$$\alpha_k = \frac{c^{(1)}(x^{k+1} \tilde{P}_k) - \beta_k c^{(1)}(x^k \tilde{P}_{k-1})}{c^{(1)}(x^k \tilde{P}_k)} \quad \text{for} \quad k > 0.$$
The expression of the coefficients is due to the orthogonality of the polynomials $\tilde{P}_k$.
Multiplying both sides in (12) by a polynomial $U_{k-1}$ of exact degree $k - 1$ and applying $c^{(1)}$ gives us
\[
\beta_k = \frac{c^{(1)}(x U_{k-1} \tilde{P}_k)}{c^{(1)}(U_{k-1} \tilde{P}_k)} \quad \text{for } k > 0. \tag{13}
\]
Moreover, multiplying both sides in (12) by a polynomial $U_k$ of exact degree $k$ and applying $c^{(1)}$ leads to
\[
x_k = \frac{c^{(1)}(x U_k \tilde{P}_k) - \beta_k c^{(1)}(U_k \tilde{P}_{k-1})}{c^{(1)}(U_k \tilde{P}_k)} \quad \text{for } k > 0. \tag{14}
\]
Now, setting
\[
P_k^{(j)}(x) = 1 + x_k^{(j)} x + \cdots + x_k^{(j)} x^k
\]
we have, for $j = 0, \ldots, s$,
\[
P_k^{(j)}(0) = 1, \quad L_i^{(j)}(P_k^{(j)}) = 0 \quad \text{for } i = 0, \ldots, k - 1. \tag{15}
\]
Thus, the polynomials $P_k^{(j)}$ are the bi-orthogonal polynomials introduced by Brezinski in [3]. So, they are of degree $k$ at most and satisfy
\[
P_{k+1}^{(j)}(x) = P_k^{(j)}(x) + \lambda_k^{(j)} x \tilde{P}_k(x) \tag{16}
\]
with
\[
\lambda_k^{(j)} = -\frac{L_k^{(j)}(P_k^{(j)})}{L_k^{(j)}(x \tilde{P}_k)}.
\]
The polynomials $P_k^{(j)}$ can be written, by definition
\[
P_k^{(j)}(x) = 1 + x \Phi_k^{(j-1)}(x)
\]
and thus, the polynomials $\Phi_k^{(j)}$ verify
\[
\Phi_{k+1}^{(j)}(x) = \Phi_k^{(j)}(x) + \lambda_k^{(j)} \tilde{P}_k(x).
\]
And, as $x_k^{(j)} = \Phi_{k-1}^{(j)}(A)z$ and $r_k^{(j)} = b^{(j)} - A x_k^{(j)}$, we obtain, setting $q_k = \tilde{P}_k(A)z$
\[
x_{k+1}^{(j)} = x_k^{(j)} + \lambda_k^{(j)} q_k, \quad r_{k+1}^{(j)} = r_k^{(j)} - \lambda_k^{(j)} A q_k. \tag{17}
\]

**Remark 3.4.** Unlike in the original Lanczos’ method, the polynomials $P_k^{(j)}$ are not orthogonal polynomials. Indeed, we do not usually have $L_i^{(j)}(1) = L_i^{(j)}(x)$. Moreover, we do not have a polynomial relation of the form $r_k^{(j)} = P_k^{(j)}(A)r_0$. 
The polynomials $\bar{P}_k$ and $P^{(j)}_k$ can be written in terms of determinants as

$$
\bar{P}_k(x) = \begin{vmatrix}
    c_1 & \cdots & c_{k+1} \\
    \vdots & \ddots & \vdots \\
    c_k & \cdots & c_{2k} \\
    1 & \cdots & x^k \\
    c_1 & \cdots & c_k \\
    \vdots & \ddots & \vdots \\
    c_k & \cdots & c_{2k-1}
\end{vmatrix}.
$$

We can easily check that these polynomials are monic and satisfy the conditions (11).

Similarly, for $P^{(j)}_k$, we have

$$
P^{(j)}_k(x) = \begin{vmatrix}
    1 & x & \cdots & x^k \\
    -(b^{(j)},y) & c_1 & \cdots & c_k \\
    \vdots & \vdots & \ddots & \vdots \\
    -(A^{k-1}b^{(j)},y) & c_k & \cdots & c_{2k-1} \\
    c_1 & \cdots & c_k \\
    \vdots & \ddots & \vdots \\
    c_k & \cdots & c_{2k-1}
\end{vmatrix}.
$$

These polynomials satisfy the conditions (15) and we obviously have $P^{(j)}_k(0) = 1$.

Thus, the polynomials $\bar{P}_k$ and $P^{(j)}_k$ exist if and only if $H^{(1)}_k \neq 0$ and the polynomials $P^{(j)}_k$ are of the exact degree $k$ if and only if

$$
H^{(0)}_{k,j} = \begin{vmatrix}
    (b^{(j)},y) & c_1 & \cdots & c_{k-1} \\
    \vdots & \ddots & \vdots & \vdots \\
    (b^{(j)},A^{k-1}y) & c_k & \cdots & c_{2k-2}
\end{vmatrix} \neq 0.
$$

If the polynomials $\bar{P}_k$ and $P^{(j)}_k$ do not exist, a breakdown occurs. Such a situation can be treated by a look-ahead technique as developed by Brezinski and Redivo Zaglia in [2].

### 3.3.2. Analogy with Lanczos/Orthodir

We can now obtain the relations required for solving the systems.

The Multiple Lanczos/Orthodir algorithm will be similar to Lanczos/Orthodir (for one single system).

**Algorithm 3.5** (M-Lanczos/Orthodir).

**Initializations**

$q_0 = z$
$q_1 = (A - \alpha_0)z$

**for** $j = 1, \ldots, s$ **do**

$r_1^{(j)} = b^{(j)} - \alpha^{(j)}Az$
\[ x_1^{(j)} = z^{(j)}z \]

end for

for \( k = 1, \ldots, n - 1 \) do

for \( j = 1, \ldots, s \) do

\[ r_{k+1}^{(j)} = r_k^{(j)} - \lambda_k^{(j)} Aq_k \]

\[ x_{k+1}^{(j)} = x_k^{(j)} + \lambda_k^{(j)} q_k \]

end for

\[ q_{k+1} = (A - x_k)q_k - \beta_k q_{k-1} \]

end for

where \( z \) is an arbitrary nonzero vector, the scalars \( \lambda_k^{(j)}, x_k \) and \( \beta_k \) are defined above and

\[ x^{(j)} = -\frac{L_0^{(j)}(1)}{L_0^{(j)}(x)} \frac{(b^{(j)}, y)}{(Az, y)} \]

for the conditions (15) with \( k = 0 \).

Let us now give some useful expressions of the coefficients \( x^{(j)}, x_k, \beta_k \) and \( \lambda_k^{(j)} \). By definition of the functionals \( L_i^{(j)} \) and \( c^{(1)} \) and using their linearity properties, we easily prove, setting \( q_k = \overline{P_k(A)z} \), that

\[ c^{(1)}(x\overline{P_k}) = (A^{i+1}\overline{P_k}(A)z, y) = (A^{i+1}q_k, y), \quad (18) \]

\[ L_i^{(j)}(P_k) = -(A^i b^{(j)}, y) + (A^{i+1} P_k^{(j)}(A)z, y) \]

\[ = -(A^i r_k^{(j)}, y). \quad (19) \]

Thus, we can deduce the

**Proposition 3.6.** The scalars \( x^{(j)}, \lambda_k^{(j)}, x_k \) and \( \beta_k \) are given by

\[ x^{(j)} = \frac{(b^{(j)}, y)}{(Az, y)}, \]

\[ \lambda_k^{(j)} = \frac{(A^k r_k^{(j)}, y)}{(A^{k+1} q_k, y)} = \frac{(V_k^{(j)}(A) r_k^{(j)}, y)}{(A V_k^{(j)}(A) q_k, y)}, \]

\[ \beta_k = \frac{(A^{k+1} q_k, y)}{(A^k q_{k-1}, y)} = \frac{(A^2 U_{k-1}(A) q_k, y)}{(A U_{k-1}(A) q_{k-1}, y)}, \quad (20) \]

\[ x_k = \frac{(A^{k+2} q_k, y) - \beta_k (A^{k+1} q_{k-1}, y)}{(A^{k+1} q_k, y)} \]

\[ = \frac{(A^2 U_k(A) q_k, y) - \beta_k (A U_k(A) q_{k-1}, y)}{(A U_k(A) q_k, y)} \]

for \( j = 1, \ldots, s \) and for any polynomial \( V_k^{(j)} \) of exact degree \( k \).
Proof. It is a consequence of (13), (14), (18), (19), Proposition 3.2 and the orthogonality conditions (7).

This algorithm can be directly implemented using $U_k(x) = x^k$ and $V_k^{(j)}(x) = x^k$ but it uses the computation of successive powers of the matrix $A$, which is known to be numerically unstable.

To avoid computing many $V_k^{(j)}(A)q_k$, we choose $V_k^{(j)} = U_k$. We thus only have to compute $U_k(A)q_k$, $U_{k-1}(A)q_k$, $U_k(A)q_{k-1}$ and $U_k(A)P_k^{(j)}$. A natural choice for $U_k$ is $U_k = P_k$ (since then, $U_k(A)q_k = U_k(A)P_k(A)z = P_k(A)P_k^{-1}(A)z = U_k(A)q_k$).

So we must compute the vectors $P_k(A)z$, $P_k(A)P_{k-1}(A)z$ and $P_k(A)P_k^{(j)}(A)z$ using the recurrence relationships the polynomials $P_k$ and $P_k^{(j)}$ satisfy.

Using the technique we can find in [4], we obtain the

**Proposition 3.7.** Setting $\tilde{r}_k^{(j)} = P_k(A)r_k^{(j)}$, $\tilde{q}_k = P_k(A)q_k$, $\tilde{q}_k = P_{k-1}(A)q_k$ and $\tilde{r}_k^{(j)} = P_{k-1}(A)r_k^{(j)}$, we have

\[
\tilde{q}_{k+1} = (A - \alpha_k)^2 \tilde{q}_k - 2\beta_k (A - \alpha_k) \tilde{q}_k + \beta^2 \tilde{q}_{k-1},
\]

\[
\tilde{q}_{k+1} = (A - \alpha_k) \tilde{q}_k - \beta_k \tilde{q}_k,
\]

\[
\tilde{r}_k^{(j)} = \tilde{r}_k^{(j)} - \tilde{r}_k^{(j)} A \tilde{q}_k,
\]

\[
\tilde{r}_k^{(j)} = (A - \alpha_k) \tilde{r}_k^{(j)} - \beta_k \tilde{r}_k^{(j)} + \tilde{r}_k^{(j)} \beta_k A \tilde{q}_k
\]

\[
= (A - \alpha_k) \tilde{r}_k^{(j)} - \beta_k \tilde{r}_k^{(j)} - \tilde{r}_k^{(j)} A \tilde{q}_{k+1}
\]

with

\[
\tilde{r}_k^{(j)} = \frac{(A^2 \tilde{q}_k, y)}{(A \tilde{q}_k, y)}, \quad \beta_k = \frac{(A^2 \tilde{q}_k, y)}{(A \tilde{q}_k, y)}, \quad \alpha_k = \frac{(A^2 \tilde{q}_k, y) - \beta_k (A \tilde{q}_k, y)}{(A \tilde{q}_k, y)}.
\]

Proof. Using the relations (12) and (16) we find the expressions of $\tilde{q}_{k+1}, \tilde{r}_{k+1}, \tilde{q}_{k+1}$ and $\tilde{r}_{k+1}^{(j)}$. Then, replacing $U_k$ and $V_k^{(j)}$ in the Proposition 3.6, the result is obvious for $\alpha_k, \beta_k$ and $\tilde{r}_k^{(j)}$.

We thus obtain a transpose-free algorithm.

**Algorithm 3.8** (TFM-Lanczos/Orthodir).

**Initializations**

\[
\tilde{q}_0 = q_0 = z
\]

\[
\tilde{q}_k = (A - \alpha_0) q_k
\]

\[
\tilde{q}_k = (A - \alpha_0) q_k
\]

\[
\tilde{q}_0 = 0
\]

\[
\text{for } j = 1, \ldots, s \text{ do}
\]

\[
\tilde{r}_1^{(j)} = b^{(j)} - \tilde{r}_1^{(j)} A z
\]

\[
\tilde{r}_1^{(j)} = (A - \alpha_0) \tilde{r}_1^{(j)}
\]
\[ x_1^{(j)} = \chi^{(j)}z \]

**end for**

**for** \( k = 1, \ldots, n - 1 \)  

**for** \( j = 1, \ldots, s \)  

\[
\tilde{r}_{k+1}^{(j)} = r_k^{(j)} - \beta_k^{(j)} A \tilde{q}_k \\
\tilde{r}_k^{(j)} = (A - \alpha_k) \tilde{r}_k^{(j)} - \beta_k \tilde{r}_k^{(j)} + \lambda_k^{(j)} \beta_k A \tilde{q}_k \\
x_{k+1}^{(j)} = x_k^{(j)} + \lambda_k^{(j)} q_k \\
end for
\]

\[
\tilde{q}_{k+1} = (A - \alpha_k) \tilde{q}_k - 2 \beta_k (A - \alpha_k) \tilde{q}_k + \beta_k^2 \tilde{q}_{k-1} \\
\tilde{q}_k = (A - \alpha_k) q_k - \beta_k q_{k-1} \\
q_{k+1} = (A - \alpha_k) q_k - \beta_k q_{k-1} \\
end for
\]

**with** \( \alpha_k, \beta_k, \chi_k^{(j)} \) and \( \lambda_k^{(j)} \) defined as above.

Note that the vectors \( x_k^{(j)} \) are the same as in the M-Lanczos/Orthodir implementation.

### 3.3.3. Analogy with Lanczos/Orthomin

The algorithm Lanczos/Orthomin uses the polynomials \( P_{k+1} \) and \( \tilde{P}_k \) to compute the polynomial \( \tilde{P}_{k+1} \).

To obtain similar relations, we need to introduce the polynomial \( P^{(0)}_k \) which satisfies

\[
\tilde{P}_{k+1}(x) = P^{(0)}_{k+1}(x) + \gamma_k \tilde{P}_k(x). 
\]  

(21)

We can easily prove that the polynomial \( P^{(0)}_k \) is orthogonal with respect to the functional \( c \) defined by

\[
c(x^t) = c^{(1)}(x^{t-1}) = (A^t z, y)
\]

and will be chosen such that \( P^{(0)}_k(0) = 1 \).

As \( \tilde{P}_k \) is orthogonal with respect to \( c^{(1)} \), then \( P^{(0)}_k \) satisfy a relation of the form

\[
P_{k+1}(x) = P^{(0)}_{k+1}(x) + \lambda^{(0)}_{k+1} x \tilde{P}_k.
\]

The polynomial \( \tilde{P}_{k+1} \) is not necessarily monic but has the same leading coefficient as \( P^{(0)}_{k+1} \).

We define \( r^{(0)}_k = b^{(0)} - A_x^{(0)} = b^{(0)} - A \Phi^{(0)}_{k+1}(A) z \) with \( b^{(0)} = -z \) so that we have \( L^{(0)}_i(1) = c_i \).

The polynomial \( P^{(0)}_k \) thus verifies \( r^{(0)}_k = -P^{(0)}_k(A)z \) and we have, setting \( q_k = \tilde{P}_k(A)z \), \( q_{k+1} = \gamma_k A q_k - r^{(0)}_{k+1} \) and \( r^{(0)}_k = -P^{(0)}_{k+1}(A)z \).

The scalar \( \gamma_k \) can be written as

\[
\gamma_k = \frac{c(x^{k+1} P^{(0)}_{k+1})}{c^{(1)}(x^{k+1} P_k)} = \frac{(A^{k+1} r^{(0)}_{k+1}, y)}{(A^{k+1} q_k, y)} \\
= \frac{c(x U_k P^{(0)}_{k+1})}{c^{(1)}(U_k P_k)} = \frac{(A U_k(A) r^{(0)}_{k+1}, y)}{(A U_k(A) q_k, y)}. 
\]

(22)
As in the previous case, if we choose \( U_k(x) = x^k \) and \( V_k^{(j)} = x^k \) then this can be directly implemented. Thus, the Multiple Lanczos/Orthomin algorithm is as follows:

**Algorithm 3.9 (M-Lanczos/Orthomin)**.

**Initializations**

\[
q_0 = z
\]

for \( j = 0, \ldots, s \) do

\[
r_1^{(j)} = b^{(j)} - \alpha^{(j)} Az
\]

\[
x_1^{(j)} = \alpha^{(j)} z
\]

end for

\[
q_1 = \gamma_0 q_0 - r_1^{(0)}
\]

for \( k = 1, \ldots, n - 1 \)

for \( j = 0, \ldots, s \)

\[
r_{k+1}^{(j)} = r_k^{(j)} - \alpha_k^{(j)} A q_k
\]

\[
x_{k+1}^{(j)} = x_k^{(j)} + \alpha_k^{(j)} q_k
\]

end for

\[
q_{k+1} = \gamma_k q_k - r_{k+1}^{(0)}
\]

end for

where the scalars \( \lambda_k^{(j)}, \alpha_k^{(j)} \) and \( \gamma_k \) are defined above.

If we set, again, \( U_k \equiv \overline{P}_k \) and \( V_k^{(j)} \equiv \overline{P}_k \), then we obtain the

**Proposition 3.10.** Setting \( \widetilde{r}_k^{(j)} = \overline{P}_k(A) r_k^{(j)}, \overline{q}_k = \overline{P}_k(A) q_k, \quad \overline{\tau}_k^{(j)} = \overline{P}_{k-1}(A) r_k^{(j)}, \quad \text{and} \quad \overline{r}_k^{(j)} = P_k^{(0)}(A) r_k^{(j)} \), we have

\[
\widetilde{r}_{k+1}^{(j)} = r_k^{(j)} + \lambda_k^{(j)} A \overline{r}_k^{(0)} + \lambda_k^{(j)} A \overline{\tau}_k^{(j)} - \lambda_k^{(j)} A \overline{\tau}_k^{(j)} A \overline{r}_k^{(0)}
\]

\[
\overline{\tau}_{k+1}^{(j)} = \overline{r}_k^{(j)} - \lambda_k^{(j)} A \overline{q}_k,
\]

\[
\overline{r}_{k+1}^{(j)} = r_{k+1}^{(j)} + \gamma_k \overline{\tau}_{k+1}^{(j)}
\]

\[
\overline{\tau}_{k+1}^{(j)} = \overline{\tau}_{k+1}^{(j)} - 2 \gamma_k \overline{\tau}_{k+1}^{(j)}
\]

with

\[
\lambda_k^{(j)} = \frac{(\overline{r}_k^{(j)}, y)}{(A \overline{q}_k, y)}, \quad \gamma_k = \frac{(A \overline{\tau}_{k+1}^{(0)}, y)}{(A \overline{q}_k, y)}.
\]
Proof. Those relations are easily obtained from (21) and (12). Then, replacing $U_k$ and $V_k^{(j)}$ by $\tilde{P}_k$ in (20) and (22), the coefficients $\lambda_k^{(j)}$ and $\gamma_k$ are obtained. □

**Proposition 3.11.** As $P_k^{(0)}(0) = 1$ then we can write $\tilde{r}_k^{(j)} = b^{(j)} - A\tilde{x}_k^{(j)}$ and

$$r_k^{(j)} = 0 \Rightarrow \tilde{r}_k^{(0)} = 0 \Rightarrow A\tilde{x}_k^{(j)} = b^{(j)}.$$  

Then

$$\tilde{x}_{k+1}^{(j)} = \tilde{x}_k^{(j)} - \lambda_k^{(j)} r_k^{(0)} - \lambda_k^{(0)} A\tilde{r}_k^{(j)} + \lambda_k^{(0)} \tilde{r}_k^{(j)} A\tilde{q}_k.$$  

Thus the transpose-free algorithm follows:

**Algorithm 3.12** (TFM-Lanczos/Orthomin).

**Initializations**

$$\bar{q}_0 = z$$

for $j = 0, \ldots, s$ do

$$\bar{r}_1^{(j)} = b^{(j)} - \lambda_k^{(j)} Az$$

$$\tilde{r}_1^{(j)} = (1 + \lambda_k^{(0)} A)\bar{r}_1^{(j)}$$

$$\bar{r}_1^{(j)} = \tilde{r}_1^{(j)} + \gamma_0 \tilde{r}_1^{(j)}$$

$$\bar{x}_1^{(j)} = \lambda_k^{(j)} z - \lambda_k^{(0)} b^{(j)} + \lambda_k^{(0)} \lambda_k^{(j)} Az$$

end for

$$\bar{q}_1 = \gamma_0 \bar{q}_0 - \bar{r}_1^{(0)} - 2\gamma_0 \tilde{r}_1^{(0)}$$

for $k = 1, \ldots, n - 1$ do

for $j = 0, \ldots, s$ do

$$\bar{r}_k^{(j)} = \bar{r}_k^{(j)} + \lambda_k^{(j)} A\tilde{r}_k^{(0)} + \lambda_k^{(0)} A\tilde{r}_k^{(j)} - \lambda_k^{(0)} \lambda_k^{(j)} \tilde{A}\tilde{q}_k$$

$$\tilde{r}_k^{(j)} = \tilde{r}_k^{(j)} - \lambda_k^{(j)} \tilde{A}\tilde{q}_k$$

$$\bar{r}_k^{(j)} = \bar{r}_k^{(j)} + \gamma_k \bar{r}_k^{(j)}$$

$$\bar{x}_k^{(j)} = \lambda_k^{(j)} z - \lambda_k^{(0)} b^{(j)} + \lambda_k^{(0)} \lambda_k^{(j)} Az$$

end for

$$\bar{q}_{k+1} = \gamma_k^2 \bar{q}_k - \bar{r}_k^{(0)} - 2\gamma_k \tilde{r}_k^{(0)}$$

end for

Note that the vectors $\tilde{x}_k^{(j)}$ are not the same as in the M-Lanczos/Orthomin implementation.

An analogy with Lanczos/Orthores cannot be obtained since, usually, the polynomials $P_k^{(j)}$ are not orthogonal.
The following table shows the computational cost per iteration of each algorithm. The number of \( n \)-vectors required are displayed in the column Memory.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( n )-vector DOT</th>
<th>Matrix-vector products</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-Lanczos/Orthodir</td>
<td>( s + 2 )</td>
<td>2</td>
<td>( 2s + 1 )</td>
</tr>
<tr>
<td>M-Lanczos/Orthomin</td>
<td>( s + 2 )</td>
<td>2</td>
<td>( 2s + 2 )</td>
</tr>
<tr>
<td>TFM-Lanczos/Orthodir</td>
<td>( s + 4 )</td>
<td>( s + 4 )</td>
<td>( 3s + 3 )</td>
</tr>
<tr>
<td>TFM-Lanczos/Orthomin</td>
<td>( s + 2 )</td>
<td>( s + 4 )</td>
<td>( 4s + 1 )</td>
</tr>
</tbody>
</table>

This shows that the number of matrix-vector products does not depend on \( s \) for the two first algorithms, while this is not the case for the Transpose-free ones. It must be noticed that the Transpose-free algorithms require much more memory and they can be seen as a generalization of the algorithms found in [4, 6].

We thus obtained a polynomial and a matrix interpretation with one Hankel matrix to linear systems with several right-hand sides that is based on orthogonal polynomials. The method proposed only needs, in the basic implementations (M-Lanczos/Orthodir and M-Lanczos/Orthomin), two matrix-vector products per iteration (this does not depend on \( s \), the number of right-hand sides considered).

Unfortunately, the greater \( n \) is, the worse the numerical results are for the two basic implementations (see below). Unless improved, those two methods only seem to have a theoretical interest. We will now study a modification of the BiCGSTAB. The BiCGSTAB from van der Vorst gives good numerical results in the single case. So we can hope an acceleration (due to a certain minimization) of the convergence with multiple right-hand sides.

### 3.3.4. Modification of BiCGSTAB for several right-hand sides

The algorithm with less computational cost for the Modified BiCGSTAB is the M-Lanczos/Orthonin.

In the BiCGSTAB, the polynomials \( V_k \) defined by

\[
V_{k+1}(x) = (1 + a_k x)V_k(x)
\]

are considered and the sequence \( \tilde{r}_k \) defined by

\[
\tilde{r}_k = V_k(A)r_k
\]

is constructed. The scalar \( a_k \) is chosen such that \( ||\tilde{r}_{k+1}||^2 \) is minimum.

Let us set

\[
\tilde{r}_k^{(j)} = V_k(A)r_k^{(j)}.
\]
From (17), we obtain
\[ \tilde{r}_{k+1}^{(j)} = r_k^{(j)} - \lambda_k^{(j)} A V_k(A) q_k. \]

Thus, to avoid computing several vectors \( V_k(A) q_k \), the polynomial \( V_k \) must not depend on \( j \).

This is why we now choose \( a_k \) which minimizes
\[ \sum_{j=1}^{s} \| \tilde{r}_{k+1}^{(j)} \|^2. \]

Setting \( \tilde{q}_k = V_k(A) q_k \) and \( \tilde{s}_k^{(j)} = \tilde{r}_k^{(j)} - \lambda_k^{(j)} \tilde{q}_k \), we easily find
\[ a_k = \frac{\sum_{j=1}^{s} \tilde{s}_k^{(j)} A \tilde{s}_k^{(j)}}{\sum_{j=1}^{s} \| A \tilde{s}_k^{(j)} \|^2}. \]

Then we obtain the algorithm Multiple BiCGSTAB/Orthomin.

Algorithm 3.13 (M-BiCGSTAB/Orthomin).

Initializations
\[ \tilde{q}_0 = z \]
for \( j = 0, \ldots, s \) do
\[ \tilde{r}_1^{(j)} = (I + a_0 A) (b^{(j)} - \alpha^{(j)} A z) \]
\[ x_1^{(j)} = \alpha^{(j)} z \]
end for
for \( k = 1, \ldots, n - 1 \)
for \( j = 0, \ldots, s \) do
\[ \tilde{s}_k^{(j)} = \tilde{r}_k^{(j)} - \lambda_k^{(j)} A \tilde{q}_k \]
\[ \tilde{r}_{k+1}^{(j)} = (I + a_k A) \tilde{s}_k^{(j)} \]
\[ x_{k+1}^{(j)} = x_k^{(j)} + \lambda_k^{(j)} - a_k A \tilde{s}_k^{(j)} \]
end for
\[ \tilde{q}_{k+1} = \tilde{r}_{k+1}^{(0)} - \gamma_k (I + a_k A) \tilde{q}_k. \]
end for

The scalars \( \lambda_k^{(j)} \) and \( \gamma_k \) can be expressed by
\[ \lambda_k^{(j)} = \frac{(\tilde{r}_k^{(j)}, y)}{(A \tilde{q}_k, y)}, \quad \gamma_k = \frac{1}{a_k} \frac{(\tilde{r}_{k+1}^{(0)}, y)}{(A \tilde{q}_k, y)}. \]
Let us see the computational cost of such an algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>n-vector DOT</th>
<th>Matrix-vector products</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-BiCGSTAB/Orthomin</td>
<td>$3s + 2$</td>
<td>$s + 2$</td>
<td>$3s + 1$</td>
</tr>
</tbody>
</table>

This method is ought to be more numerically accurate but it requires $s$ matrix-vector products more per iteration (to compute the $A s_k^{(j)}$).

**Remark 3.14.** The BiCGSTAB seems to be the most efficient method of the Lanczos Type Products Methods (LTPM). The Conjugate Gradient Square (CGS) in particular [18], cannot be used here since the $V_k$ must be independent from $j$.

4. **Numerical examples**

Before considering the examples, let us remark that the M-Lanczos/Orthomin and the M-Lanczos/Orthodir only seem to be efficient on small matrices (dimension less than 20), even if the computational cost is theoretically lower than for the M-BiCGSTAB or the other Transpose-free algorithms. Secondly, the TFM-Lanczos/Orthomin only seem to have good numerical results on matrices of dimension less than 100. This is why we will only focus the numerical study on the M-BiCGSTAB and on the TFM-Lanczos/Orthomin.

Every routine was written in Matlab 4.2c.l. All the matrices we considered are of order $n = 500$. All the right-hand sides were randomly chosen, using the $RAND$ function in Matlab. The stopping criteria used is $(1/s) \sum_{i=1}^{s} \| r_k^{(i)} \|^2 < 10^{-16}$ (unless the matrix dimension is reached).

We used three symmetric matrices and three nonsymmetric matrices to point out how fast the convergence was in each case. For each matrix, we computed the condition number using the $COND$ function in Matlab. Then, we considered $s = 1, 10, 20, 30, 40$ and $50$ right-hand sides to see the behaviour of the M-BiCGSTAB when increasing the number of right-hand sides (since the coefficient $a_k$ in the M-BiCGSTAB depends on every residue). There is no need to do such a comparison for the TFM-Lanczos/Orthomin since each residue is considered independently from the other ones. All the results are presented in a table for the M-BiCGSTAB.

On each figure, we show the results for $s = 1$ and $s = 50$ for the M-BiCGSTAB as well as the results for $s = 50$ for the TFM-Lanczos/Orthomin. This will allow us to compare the behaviour of the M-BiCGSTAB step by step when increasing the number of right-hand sides. The M-BiCGSTAB and the TFM-Lanczos/Orthomin can be compared too. The graphs represent the norms of the residuals, in logarithmic scale, versus the iterations.

4.1. **Symmetric matrices**

We will first study the implementation of the M-BiCGSTAB and of the TFM-Lanczos/Orthomin on symmetric matrices since such matrices generally give better results than the nonsymmetric ones.
The first matrix considered is the matrix
\[
M_1 = \begin{pmatrix}
20 & -1 \\
-1 & 20 & \ddots \\
& \ddots & \ddots & -1 \\
& & -1 & 20
\end{pmatrix}
\]
whose condition number is 1.22.

We obtained the following results (iteration where the stopping criteria is satisfied) for the M-BiCGSTAB:

<table>
<thead>
<tr>
<th>(s)</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>16</td>
</tr>
</tbody>
</table>

The convergence for \(s = 1\) and \(s = 50\) is as follows:

As we could expect, the convergence is fast whatever the method is. This might be due to two different factors. First, the matrix considered is symmetric. Secondly, the problem is very well conditioned. We note that the number of right-hand sides is not important for the M-BiCGSTAB. And the shape of convergence is the same for \(s = 1\) and \(s = 50\) for the M-BiCGSTAB. The TFM-Lanczos/Orthomin gives a quite good result too.

The next matrix is
\[
M_2 = \begin{pmatrix}
B & -I \\
-I & B & \ddots \\
& \ddots & \ddots & -I \\
& & -I & B
\end{pmatrix}
\]
where the matrix $I$ is the identity matrix of order 20 and

$$B = \begin{pmatrix} 4 & & & \\ & \ddots & & \\ & & 4 & \\ & & & 4 \end{pmatrix}$$

is of order 20. The condition number is 2.97.

We obtained the following results when applying the M-BiCGSTAB

<table>
<thead>
<tr>
<th>$s$</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>40</td>
<td>45</td>
<td>41</td>
<td>41</td>
<td>49</td>
<td>43</td>
</tr>
</tbody>
</table>

and the convergence is as follows:

In this example, we see that the number of right-hand sides does not really interact with the convergence of the M-BiCGSTAB. The smaller number of iterations needed is 40 for $s=1$ while the larger one is 49 for $s=40$. The two curves for the M-BiCGSTAB are close again and the convergence is quite good. The TFM-Lanczos/Orthomin gives here a better result.

The last symmetric matrix considered is the diagonal matrix

$$M_3 = \begin{pmatrix} 1 \\ 2 \\ & \ddots \\ & & \ddots \\ & & & 500 \end{pmatrix}$$
used in [7] with size 200. The condition number is obviously 500. Even if this matrix is particular, let us see what the results are.

We obtained the following results with the M-BiCGSTAB

<table>
<thead>
<tr>
<th>$s$</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>189</td>
<td>192</td>
<td>192</td>
<td>198</td>
<td>198</td>
<td>205</td>
</tr>
</tbody>
</table>

and the graph for $s = 1$ and $s = 50$ is

![Graph showing convergence](image)

The convergence is good, despite a condition number equal to 500 (which is not very big but not so small). The two graphs are again very close for the M-BiCGSTAB. We can see that the smaller number of iterations is 189 for $s = 1$ as the larger one is 205 for $s = 50$. Thus, the number of right-hand sides does not seem to be very important here. The M-BiCGSTAB is a better than the TFM-Lanczos/Orthomin for $s = 50$ but the two methods give the same behaviour.

4.2. **Nonsymmetric matrices**

As the M-BiCGSTAB and the TFM-Lanczos/Orthomin seemed to be efficient on symmetric matrices, we can wonder if it would be the same with nonsymmetric ones. (We already know that theoretically, the methods converge).

The first matrix we considered is

$$M_4 = \begin{pmatrix} B & -I \\ -I & B & \ddots \\ & \ddots & \ddots & -I \\ & & -I & B \end{pmatrix}$$
with

\[
B = \begin{pmatrix}
  4 & 0 & & & \\
  -2 & 4 & & & \\
  & & \ddots & & \\
  & & & \ddots & 0 \\
  & & & & -2 & 4
\end{pmatrix}.
\]

The condition number of this matrix is $1.01 \times 10^3$. We obtained the following results for the M-BiCGSTAB

<table>
<thead>
<tr>
<th>$s$</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>311</td>
<td>304</td>
<td>307</td>
<td>315</td>
<td>311</td>
<td>316</td>
</tr>
</tbody>
</table>

The convergence behaviour is as follows:

Even if the matrix is not well conditioned, we can see that the method gives us a quite good convergence. It required 304 iterations for $s=10$ and 316 for $s=50$. From the graph, we can see that the behaviour of each curve is slightly the same with stagnation until the 100th iteration for the M-BiCGSTAB. For the TFM-Lanczos/Orthomin, the convergence is much slower and at iteration 500, the stopping criteria is not reached.

The next matrix used is the matrix

\[
M_5 = \begin{pmatrix}
  2 & 1 & & & \\
  0 & 2 & 1 & & \\
  1 & 0 & 2 & 1 & \\
  & \ddots & \ddots & \ddots & \ddots \\
  & & & \ddots & 1 \\
  & & & & 1 & 0 & 2
\end{pmatrix}
\]
considered in [4]. The condition number is 2.91 and the results we obtained are, for the M-BiCGSTAB,

<table>
<thead>
<tr>
<th>$s$</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>274</td>
<td>285</td>
<td>284</td>
<td>279</td>
<td>286</td>
<td>294</td>
</tr>
</tbody>
</table>

The convergence behaves as follows:

Even if the condition number is small, we have a linear but not fast convergence of the method for the M-BiCGSTAB. This might partly be due to the fact that the matrix is nonsymmetric. However, for the TFM-Lanczos/Orthomin, the convergence for $s = 50$ is much faster. The number of iterations needed for each number of right-hand sides $s$ for the M-BiCGSTAB is very close (from 274 if $s = 1$ to 294 for $s = 50$), as we can see in the table.

The last nonsymmetric matrix we considered is the matrix $\text{REDHEFF}$ of MATLAB Test Matrix Toolbox of Higham [9], also considered in [4]. If we write this matrix $M_6 = (m_{i,j})$, then the coefficients satisfy

$$m(i,j) = \begin{cases} 
1 & \text{if } j = 1, \\
1 & \text{if } i \text{ divides } j, \\
0 & \text{otherwise.}
\end{cases}$$

Its condition number is $2.41 \times 10^3$.

We obtained the following results for the M-BiCGSTAB

<table>
<thead>
<tr>
<th>$s$</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>50</td>
<td>45</td>
<td>47</td>
<td>46</td>
<td>48</td>
<td>46</td>
</tr>
</tbody>
</table>
Then, for \( s = 1 \) and \( s = 50 \), we obtained the following graph:

![Graph showing convergence for M-BiCGSTAB and TFM-Lanczos/Orthomin]

Again, even with a badly conditioned matrix, the M-BiCGSTAB behaves quite good for all \( s \), with a smaller iteration number of 45 (\( s = 10 \)) and a larger one of 50 (\( s = 1 \)). In this last example, we can see that the two curves are again very close. The TFM-Lanczos/Orthomin reached the stopping criteria in much more iteration for \( s = 50 \).

5. Conclusion

Studying the examples, several remarks can be made.

Firstly, the M-Lanczos/Orthomin and the M-Lanczos/Orthodir do not give a good convergence. The TFM-Lanczos/Orthodir does not give a good convergence either. This might be due to the fact that the sequence \( x^{(j)}_k \) is the same for M-Lanczos/Orthodir and TFM-Lanczos/Orthodir (the difference is the way to compute it).

Secondly, the M-BiCGSTAB and the TFM-Lanczos/Orthomin gave us better convergence with symmetric matrices, even if nonsymmetric matrices have a good behaviour too. In the examples, we can see that neither the M-BiCGSTAB nor the TFM-Lanczos/Orthomin is a better method. From the memory point of view, we can only say that the TFM-Lanczos/Orthomin requires one more vector to be stored as from the computational cost point of view, the M-BiCGSTAB needs much more dot products.

Thirdly, the number of right-hand sides does not seem to be very important in all the examples considered (except, of course, on computational cost and memory requirements) for the M-BiCGSTAB.

Fourthly, the behaviour of the M-BiCGSTAB seems to be the same whatever the number of right-hand sides is. The behaviour of the M-BiCGSTAB only depends on the considered matrix, even if the coefficient \( a_k \) is computed regardless to the different right-hand sides. Thus, the computation of the orthogonal polynomials we consider must be accurate. So, we may see [1] and use quasi-orthogonality (in a way, a numerical orthogonality) instead of orthogonality to improve the stability of the algorithms and if it possible to apply this to the methods.

Finally, even with the minimization property of the M-BiCGSTAB, this does not seem to be a criteria of better convergence (since it should then be better than the TFM-Lanczos/Orthomin, but it is not really the case).
On a theoretical point of view, we saw that we got a matrix and a polynomial interpretation of the methods (only depending on one Hankel matrix), which seems to be new.

The main drawback of the methods, as we can see in the graphs, is that we do not have decreasing residues, as in most of the Lanczos-types algorithms. So we must see if we can modify it to get this property, conserving the finite termination property. Particularly, we now have to study if the M-Lanczos/Orthodir and M-Lanczos/Orthomin can be improved since it requires a very small computational cost. This method has to be compared with Lanczos type ones, and particularly with the Global Lanczos process [13]. This is under consideration.

Acknowledgements

I would like to thank V. Simoncini for her helpful suggestions and advises that helped to improve this paper.

References