Transpose-free Lanczos-type algorithms for nonsymmetric linear systems

C. Brezinski [a] and M. Redivo-Zaglia [b]

[a] *Laboratoire d'Analyse Numérique et d'Optimisation, Université des Sciences et Technologies de Lille, F-59655 Villeneuve d'Ascq Cedex, France*
E-mail: Claude.Brezinski@univ-lille1.fr
[b] *Dipartimento di Elettronica e Informatica, Università degli Studi di Padova, Via Gradenigo 6/a, I-35131 Padova, Italy*
E-mail: michela@dei.unipd.it

Received 8 September 1997; revised 28 January 1998
Communicated by T.F. Chan

The method of Lanczos for solving systems of linear equations is implemented by using recurrence relationships between formal orthogonal polynomials. A drawback is that the computation of the coefficients of these recurrence relationships usually requires the use of the transpose of the matrix of the system. Due to the indirect addressing, this is a costly operation. In this paper, a new procedure for computing these coefficients is proposed. It is based on the recursive computation of the products of polynomials appearing in their expressions and it does not involve the transpose of the matrix. Moreover, our approach allows to implement simultaneously and at a low price a Lanczos-type product method such as the CGS or the BiCGSTAB.

Keywords: Lanczos' method, orthogonal polynomials, CGS, BiCGSTAB

AMS subject classification: 65F10, 33A65

1. Introduction

Lanczos' method [24] for solving a system of linear equations is based on formal orthogonal polynomials. Such polynomials satisfy, under certain assumptions discussed below, some recurrence relationships. However, the computation of the coefficients of these recurrence relationships involves products of the form $A^{\mathrm{T}}v$, where v is an arbitrary vector. When the matrix is large and sparse, such products are difficult to compute due to the indirect addressing needed by the structure of A. When solving a system of nonlinear equations, A is the Jacobian matrix of the application and it is usually unknown. The products Av are approximated by difference operators and it is impossible to approximate the products $A^{\mathrm{T}}v$. So, it was interesting to derive transpose-free algorithms for implementing the method of Lanczos.

In [4], a transpose-free Lanczos/Orthodir algorithm was presented. It was based on a very simple idea known for a long time (see, for example, [2]): the coefficients of the recurrence relationships of the formal orthogonal polynomials can be expressed as ratios of products of Hankel determinants and they can be computed by the qd-algorithm of Rutishauser [26]. It was also mentioned in [4] that the same technique could be used for avoiding the use of A^{T} in the other algorithms for the implementation of Lanczos' method. All these transpose-free algorithms were tested on numerical examples and the quantities $q_k^{(n)}$ and $e_k^{(n)}$ of the qd-algorithm were computed by several algorithms such as the ε-, the η-, the rs- and the qd-algorithms and by the recurrence relationship of the Hankel determinant [8]. However, all these procedures were found to be highly numerically unstable and, so, an open question is still to find a stable way of computing products $A^i v$ and implementing the qd-algorithm.

Let us also recall that, as shown in [2, pp. 184–189], the topological ε-algorithm is a transpose-free method for the implementation of Lanczos' method. However, it requires too much storage and is intractable [19].

In this paper we will propose a different approach to this problem. It is based on the recursive computation of the products of polynomials appearing in the expressions of the coefficients of the recurrence relationships and it does not involve the transpose of the matrix. Thus, we are able to produce a transpose-free version of any algorithm for implementing Lanczos' method. In this framework, we recover the algorithm of Chan et al. [15] as a particular case. Usually, the price to pay for a transpose-free algorithm is an additional matrix–vector product but, as we will see, the products of polynomials to be computed are, in many cases, related to some Lanczos-type product methods (also called CGM methods [3]). Thus, our approach allows us to implement simultaneously, and at almost no extra cost, a Lanczos-type product method such as the CGS or the BiCGSTAB. Such a coupled implementation is, in fact, cheaper than using the two algorithms separately. Moreover, using the hybrid procedure [5] is quite easy. As we will see, it is possible to implement Lanczos' method and a Lanczos-type product method just by adding some instructions (with only one additional matrix–vector product) into the code of the product method.

Although some new algorithms for the implementation of Lanczos-type product methods are obtained in the sequel, our purpose is to derive transpose-free algorithms for Lanczos' method itself. Thus, in passing, known algorithms for product methods will be recovered such as those given in [15,21,25].

Our derivation of the algorithms is based on the interpretation of Lanczos' method in terms of formal orthogonal polynomials (FOP). Such polynomials were already known to Lanczos [23], but, although some considerations and algorithms were given in [2,20], their systematical use for obtaining recursive algorithms for implementing Lanczos' method seems to have been fully exploited in [14] for the first time.

The method of Lanczos will be presented in section 2. Section 3 is devoted to formal orthogonal polynomials. Recursive algorithms for the implementation of Lanczos' method are given in section 4 and their transpose-free versions are discussed in section 5. Numerical experiments end the paper.

2. Lanczos' method

Let us consider the $p \times p$ system of linear equations $Ax = b$. The method of Lanczos [24] consists in constructing the sequence of iterates $x_k \in \mathbb{R}^p$ defined by

$$x_k - x_0 \in K_k(A, r_0) = \operatorname{span}\bigl(r_0, Ar_0, \ldots, A^{k-1}r_0\bigr),$$
$$r_k = b - Ax_k \perp K_k\bigl(A^{\mathrm{T}}, y\bigr) = \operatorname{span}\bigl(y, A^{\mathrm{T}}y, \ldots, \bigl(A^{\mathrm{T}}\bigr)^{k-1}y\bigr),$$

where x_0 is an arbitrary vector, y an almost arbitrary one and $r_0 = b - Ax_0$. These two conditions entirely define the vector x_k. Indeed, the first one yields

$$x_k - x_0 = -a_1 r_0 - a_2 A r_0 - \cdots - a_k A^{k-1} r_0.$$

Multiplying both sides by A, adding and subtracting b in the left hand side gives

$$r_k = r_0 + a_1 A r_0 + \cdots + a_k A^k r_0 = P_k(A) r_0$$

with

$$P_k(\xi) = 1 + a_1\xi + \cdots + a_k\xi^k.$$

The orthogonality conditions for r_k give

$$\bigl(A^{\mathrm{T}^i} y, r_k\bigr) = \bigl(y, A^i r_k\bigr) = \bigl(y, A^i P_k(A) r_0\bigr) = 0, \quad \text{for } i = 0, \ldots, k-1. \tag{1}$$

Thus, the coefficients $a_1, \ldots, a_k$ are the solution of the system

$$a_1\bigl(y, A^{i+1}r_0\bigr) + \cdots + a_k\bigl(y, A^{i+k}r_0\bigr) = -\bigl(y, A^i r_0\bigr), \quad i = 0, \ldots, k-1.$$

Obviously, in practice, solving such systems for increasing values of k is only feasible if their solutions can be obtained recursively, that is, in other words, if the polynomials P_k can be computed recursively. Such a computation is possible since, in fact, the polynomials P_k form a family of formal orthogonal polynomials as will now be explained.

3. Formal orthogonal polynomials

Let us define the linear functional c on the space of polynomials by

$$c\bigl(\xi^i\bigr) = c_i = \bigl(y, A^i r_0\bigr), \quad i = 0, 1, \ldots .$$

So, if p is an arbitrary polynomial, we have $c(p) = (y, p(A)r_0)$. Thus, the orthogonality conditions (1) can be written as

$$c\bigl(\xi^i P_k\bigr) = 0, \quad \text{for } i = 0, \ldots, k-1. \tag{2}$$

A family of polynomials $\{P_k\}$ satisfying these conditions for all k is called the family of formal orthogonal polynomials (FOP) with respect to the linear functional c [2]. They are defined up to a multiplication factor which, in our case, is chosen so that $P_k(0) = 1$ as seen in section 2. From (2), $c(pP_k) = 0$ for any polynomial p of degree strictly less than k. The orthogonality conditions (2) can also be written as

$$c(U_iP_k) = 0, \quad \text{for } i = 0, \ldots, k-1,$$

where U_i is an arbitrary polynomial of exact degree i.

It is well known that, in the usual case (that is, when c can be represented by an integral on the real line with respect to some positive measure), the orthogonal polynomials P_k exist for all k and satisfy a three-term recurrence relationship. Such a recurrence relationship still holds for FOP if, for all k, P_k exists and has degree k exactly. This is true if and only if, $\forall k$, $H_k^{(1)} \neq 0$, where

$$H_k^{(n)} = \begin{vmatrix} c_n & \cdots & c_{n+k-1} \\ \vdots & & \vdots \\ c_{n+k-1} & \cdots & c_{n+2k-2} \end{vmatrix}.$$

The polynomials P_k can also be recursively computed via their adjacent family, that is, the family of polynomials $\{P_k^{(1)}\}$ orthogonal with respect to the linear functional $c^{(1)}$ defined by

$$c^{(1)}\bigl(\xi^i\bigr) = c\bigl(\xi^{i+1}\bigr) = c_{i+1}.$$

For these polynomials, let us choose the multiplication factor so that, $\forall k$, the coefficient of ξ^k in $P_k^{(1)}$ is equal to 1. In that case, the polynomials are said to be *monic*. Since they are monic, the adjacent polynomials $P_k^{(1)}$ exist under the same condition $H_k^{(1)} \neq 0$ (see, for example, [10,14]).

If, for some values of k, $H_k^{(1)} = 0$, then a division by zero (*breakdown*) occurs in the computation of the coefficients of the recurrence relationships. Breakdowns are also possible even if the preceding condition is satisfied. In the case of a breakdown, the computations can be continued by jumping over the polynomials for which a division by zero arises (see, for example, [9,10]). In this paper, we will assume that no breakdown occurs, that is, $\forall k$, P_k and $P_k^{(1)}$ exist, have the exact degree k, and can be computed by the recurrence relationship under consideration.

There are many possible ways of computing recursively the two families of FOP defined above, see [2]. Each of these recurrence relationships (or each pair of them) leads to a different algorithm for the implementation of Lanczos' method.

In the sequel, the coefficients of the recurrence relationships will be computed by using the orthogonality conditions with respect to two arbitrary families of polynomials, $\{U_k\}$ and $\{V_k\}$, such that, $\forall k$, U_k and V_k have the exact degree k.

We will now present the three main recurrences for computing these FOP in their most general form, that is, without specifying, for the moment, the families $\{U_k\}$ and $\{V_k\}$. Other recurrences can be found in [14].

Since the polynomials P_k form a family of FOP, they satisfy a three-term recurrence relationship

$$P_{k+1}(\xi) = (\alpha_{k+1}\xi + \beta_{k+1})P_k(\xi) - \mu_{k+1}P_{k-1}(\xi) \tag{3}$$

for $k = 0, 1, \dots,$ with $P_0(\xi) = 1$ and $P_{-1}(\xi) = 0$. The coefficients are given as the solution of the 3×3 system

$$\left.\begin{aligned}\alpha_{k+1}c(\xi U_{k-1}P_k) - \mu_{k+1}c(U_{k-1}P_{k-1}) &= 0\\ \alpha_{k+1}c(\xi U_kP_k) + \beta_{k+1}c(U_kP_k) - \mu_{k+1}c(U_kP_{k-1}) &= 0\\ \beta_{k+1} - \mu_{k+1} &= 1\end{aligned}\right\}. \tag{4}$$

The recurrence relationship (3) can also be written as

$$P_{k+1}(\xi) = \alpha_{k+1}\left[\left(\xi + \frac{\beta_{k+1}}{\alpha_{k+1}}\right)P_k(\xi) - \frac{\mu_{k+1}}{\alpha_{k+1}}P_{k-1}(\xi)\right],$$

that is, setting $\delta_{k+1} = \mu_{k+1}/\alpha_{k+1}$, $\gamma_{k+1} = -\beta_{k+1}/\alpha_{k+1}$ and $\eta_{k+1} = -\alpha_{k+1}$,

$$P_{k+1}(\xi) = -\eta_{k+1}\big[(\xi - \gamma_{k+1})P_k(\xi) - \delta_{k+1}P_{k-1}(\xi)\big] \tag{5}$$

and the system (4) becomes

$$\begin{aligned}\delta_{k+1} &= \frac{c(\xi U_{k-1}P_k)}{c(U_{k-1}P_{k-1})},\\ \gamma_{k+1} &= \big[c(\xi U_kP_k) - \delta_{k+1}c(U_kP_{k-1})\big]/c(U_kP_k),\\ \eta_{k+1} &= \frac{1}{\gamma_{k+1} + \delta_{k+1}}.\end{aligned} \tag{6}$$

Implementing the method of Lanczos via these relations is known as the *Lanczos/Orthores* algorithm.

Let us now set

$$Q_k(\xi) = (-1)^k\frac{H_k^{(0)}}{H_k^{(1)}}P_k^{(1)}(\xi).$$

As explained, for example, in [14], the coefficients of ξ^k in Q_k and P_k are equal and Q_k is proportional to $P_k^{(1)}$. We have the following relations between these two families of FOP:

$$\left.\begin{aligned}P_{k+1}(\xi) &= P_k(\xi) - \lambda'_{k+1}\xi Q_k(\xi)\\ Q_{k+1}(\xi) &= P_{k+1}(\xi) + \alpha_{k+1}Q_k(\xi)\end{aligned}\right\} \tag{7}$$

with $P_0(\xi) = Q_0(\xi) = 1$ and the coefficients are given by

$$\begin{aligned}\lambda'_{k+1} &= c(U_kP_k)/c(\xi U_kQ_k),\\ \alpha_{k+1} &= -c(\xi V_kP_{k+1})/c(\xi V_kQ_k).\end{aligned} \tag{8}$$

Implementing the method of Lanczos via these relations is known as the *Lanczos/Orthomin* algorithm.

Instead of using the polynomials Q_k, we can use the polynomials $P_k^{(1)}$ and compute them by their three-term recurrence relationship, that is,

$$\left.\begin{aligned} P_{k+1}(\xi) &= P_k(\xi) - \lambda_{k+1}\xi P_k^{(1)}(\xi) \\ P_{k+1}^{(1)}(\xi) &= (\xi + \beta_{k+1})P_k^{(1)}(\xi) - \gamma_{k+1}P_{k-1}^{(1)}(\xi) \end{aligned}\right\} \tag{9}$$

with $P_0(\xi) = P_0^{(1)}(\xi) = 1$ and $P_{-1}^{(1)}(\xi) = 0$. The coefficients are given by

$$\begin{aligned} \lambda_{k+1} &= c(U_k P_k)/c\big(\xi U_k P_k^{(1)}\big), \\ \gamma_{k+1} &= c\big(\xi^2 V_{k-1} P_k^{(1)}\big)/c\big(\xi V_{k-1} P_{k-1}^{(1)}\big), \\ \beta_{k+1} &= \big[\gamma_{k+1} c\big(\xi V_k P_{k-1}^{(1)}\big) - c\big(\xi^2 V_k P_k^{(1)}\big)\big]/c\big(\xi V_k P_k^{(1)}\big). \end{aligned} \tag{10}$$

Implementing the method of Lanczos via these relations is known as the *Lanczos/Orthodir* algorithm.

4. Implementation of Lanczos' method

Let us now use the recurrence relationships given in the previous section for implementing the method of Lanczos. Other algorithms, based on other recurrences for formal orthogonal polynomials, are given in [14] and studied in more detail in [1].

We have $r_k = P_k(A)r_0$ and we set $z_k = P_k^{(1)}(A)r_0$. Replacing the variable ξ by the matrix A in these recurrence relationships and multiplying by r_0 leads to various recursive algorithms for computing the sequences (x_k) and (r_k).

4.1. Lanczos/Orthores

Using the relation (5), we obtain the following algorithm:

$$\begin{aligned} r_{k+1} &= -\eta_{k+1}(Ar_k - \gamma_{k+1}r_k - \delta_{k+1}r_{k-1}), \\ x_{k+1} &= \eta_{k+1}(r_k + \gamma_{k+1}x_k + \delta_{k+1}x_{k-1}) \end{aligned}$$

with

$$\begin{aligned} \delta_{k+1} &= \big(y, AU_{k-1}(A)r_k\big)/\big(y, U_{k-1}(A)r_{k-1}\big), \\ \gamma_{k+1} &= \big[\big(y, AU_k(A)r_k\big) - \delta_{k+1}\big(y, U_k(A)r_{k-1}\big)\big]/\big(y, U_k(A)r_k\big), \\ \eta_{k+1} &= 1/(\gamma_{k+1} + \delta_{k+1}). \end{aligned}$$

For the choice $U_k \equiv P_k$, this algorithm is called *Lanczos/Orthores* [30] or BIORES [20].

4.2. *Lanczos/Orthomin*

Setting $p_k = Q_k(A)r_0$, the relations (7) lead to

$$\begin{aligned} r_{k+1} &= r_k - \lambda'_{k+1} A p_k, \\ x_{k+1} &= x_k + \lambda'_{k+1} p_k, \\ p_{k+1} &= r_{k+1} + \alpha_{k+1} p_k \end{aligned}$$

with $p_0 = r_0 = b - Ax_0$ and

$$\begin{aligned} \lambda'_{k+1} &= \bigl(y, U_k(A)r_k\bigr)/\bigl(y, AU_k(A)p_k\bigr), \\ \alpha_{k+1} &= -\bigl(y, AV_k(A)r_{k+1}\bigr)/\bigl(y, AV_k(A)p_k\bigr). \end{aligned}$$

This algorithm is due to Vinsome [29]. For the choice $U_k \equiv V_k \equiv P_k$, it is called *Lanczos/Orthomin* [30] and is equivalent to the bi-conjugate gradient (BCG) of Lanczos [23,24] which was written under an algorithmic form by Fletcher [16]. This algorithm is also known under the name of BIOMIN [20].

4.3. *Lanczos/Orthodir*

We immediately obtain from the relations (9)

$$\begin{aligned} r_{k+1} &= r_k - \lambda_{k+1} A z_k, \\ x_{k+1} &= x_k + \lambda_{k+1} z_k, \\ z_{k+1} &= (A + \beta_{k+1} I) z_k - \gamma_{k+1} z_{k-1} \end{aligned}$$

with $z_0 = r_0 = b - Ax_0$ and $z_{-1} = 0$. Using the definition of the linear functional c, we have

$$\begin{aligned} \lambda_{k+1} &= \bigl(y, U_k(A)r_k\bigr)/\bigl(y, AU_k(A)z_k\bigr), \\ \gamma_{k+1} &= \bigl(y, A^2 V_{k-1}(A)z_k\bigr)/\bigl(y, AV_{k-1}(A)z_{k-1}\bigr), \\ \beta_{k+1} &= \bigl[\gamma_{k+1}\bigl(y, AV_k(A)z_{k-1}\bigr) - \bigl(y, A^2 V_k(A)z_k\bigr)\bigr]/\bigl(y, AV_k(A)z_k\bigr). \end{aligned}$$

For the choice $U_k \equiv V_k \equiv P_k^{(1)}$ this algorithm for implementing the method of Lanczos is known under the name of *Lanczos/Orthodir* [30] and of BIODIR [20].

Let us now explain why, in order to obtain a cheap algorithm, it is necessary to use the transpose of A. For example, for computing λ_{k+1} in Lanczos/Orthodir with the choice $U_k(\xi) = \xi^k$, we need to compute $(y, A^k r_k)$ and $(y, A^{k+1} z_k)$. Since the vectors r_k and z_k depend on k, the computation of $A^k r_k$ and $A^{k+1} z_k$ requires many matrix–vector products and the algorithm becomes intractable in practice. Thus, we will replace the preceding scalar products by $(A^{\mathrm{T}^k} y, r_k)$ and $(A^{\mathrm{T}^{k+1}} y, z_k)$, respectively, that is, by (y_k, r_k) and (y_{k+1}, z_k), where $\forall i$, $y_i = A^{\mathrm{T}} y_{i-1}$ with $y_0 = y$. Thus, the price to pay for having a cheap algorithm is using A^{T}, an operation often quite difficult because of the indirect addressing required by the structure of A when the system is large and sparse. The same drawback arises in the computation of the coefficients of the other recursive algorithms for implementing the method of Lanczos and again A^{T} has to be used.

5. Transpose-free algorithms

We will now discuss how to avoid the use of A^{T} (or A^* in the case of a complex system) in the computation of the coefficients of the recurrence relationships of the orthogonal polynomials given in the preceding section. The idea consists in computing recursively the products of polynomials appearing in these coefficients. Such relations are obtained by multiplying together the recurrence relationships of the corresponding polynomials. Obviously, there are several ways for doing these computations, each of them leading to a different algorithm with a different numerical behavior. It is not our purpose herein to give all these algorithms but only some of them selected for their properties.

Our approach is the most general one since Lanczos' method can be implemented by any recurrence relationship(s) (see [14,1]) and the polynomials U_k and V_k can also be computed by arbitrary recurrences. Obviously, in this case, that is, when the polynomials U_k and V_k are not specified, the algorithms obtained require many matrix–vector products but most of them will disappear in the particular cases treated. In what follows, we will restrict ourselves to the three main algorithms for implementing the Lanczos' method, namely Lanczos/Orthores, Lanczos/Orthomin and Lanczos/Orthodir, and choose, for U_k and/or V_k, the powers of ξ, P_k (which corresponds to the CGS) and the polynomials corresponding to the BiCGSTAB. As we will see, some choices for these polynomials lead to a coupled implementation of a Lanczos-type product method at almost no extra cost. Such an idea was already implicitly contained in [14], where determinantal formulae for the residuals and the iterates of all Lanczos-type product methods could be found (see also [13]).

A Lanczos-type product method is a method where the residual has the form

$$\tilde{r}_k = W_k(A)P_k(A)r_0, \tag{11}$$

where W_k is an arbitrary polynomial satisfying $W_k(0) = 1$. This condition is mandatory for being able to recover the vector $\tilde{x}_k$ defined by $\tilde{r}_k = b - A\tilde{x}_k$ without using A^{-1}. Thus, for the choice $W_k(\xi) = \xi^k$, $\tilde{r}_k$ is not the residual of a Lanczos-type product method. Among the most well-known Lanczos-type product methods are the CGS of Sonneveld [27], the BiCGSTAB of van der Vorst [28] and its variants [21]. Other methods of this type are given in [7] (see [3] for a preliminary version).

Even if the polynomials U_k and V_k can be arbitrarily chosen, some choices are more interesting than others. In any case, let us remark that all the algorithms simplify if $\forall k$, $U_k \equiv V_k$.

The simplest choice for U_k and/or V_k is, of course, ξ^k. In that case, the recurrence relationship becomes

$$U_{k+1}(\xi) = \xi U_k(\xi)$$

with $U_0(\xi) = 1$. However, this choice does not lead to a Lanczos-type product method.

Another choice is to take $P_k^{(1)}$ and/or P_k to obtain the residual of the CGS.

When we choose $U_k \equiv V_k$, satisfying the recurrence relationship

$$U_{k+1}(\xi) = (1 - \theta_{k+1}\xi)U_k(\xi)$$

with $U_0(\xi) = 1$, and θ_{k+1} chosen to minimize $\|\tilde{r}_{k+1}\|$, we obtain the residual of the BiCGSTAB.

It is also possible to assume that U_k and/or V_k satisfy other recurrence relationships as proposed in [15,18].

We shall assume that the polynomials U_k and V_k satisfy three-term recurrence relationships of the form

$$\begin{aligned} U_{k+1}(\xi) &= (a_{k+1}\xi + b_{k+1})U_k(\xi) - d_{k+1}U_{k-1}(\xi), \\ V_{k+1}(\xi) &= \left(a'_{k+1}\xi + b'_{k+1}\right)V_k(\xi) - d'_{k+1}V_{k-1}(\xi) \end{aligned} \tag{12}$$

for $k = 0, 1, \ldots$, with $U_{-1}(\xi) = V_{-1}(\xi) = 0$, U_0 and V_0 arbitrary nonzero constants, and $\forall k$, $a_k \neq 0$ and $a'_k \neq 0$. This choice includes, as particular cases, $U_{k+1}(\xi) = \xi U_k(\xi)$ and $U_{k+1}(\xi) = (1 - \theta_{k+1})U_k(\xi)$.

If we assume that $\forall k$, $d_k \neq 0$ and/or $d'_k \neq 0$ then, by Shohat–Favard's theorem, the polynomials U_k and/or V_k satisfying (12) are formal orthogonal polynomials with respect to some linear functional. One can take, for example, the Chebyshev polynomials. Due to their minimisation property, this choice could lead to more stable algorithms. One could also, perhaps, take the Faber polynomials corresponding to the spectrum of A. All these questions remain to be studied.

Although the polynomials W_k arising in the definition of the Lanczos-type product methods could also be chosen arbitrarily, taking $W_k \equiv U_k$ or V_k directly leads to product methods.

Taking for W_k a polynomial different from U_k or V_k may lead to more complicated algorithms. A different choice will be shown in section 5.3. On the other hand, for the choice $W_k \equiv U_k$ (respectively $W_k \equiv V_k$), the condition $W_k(0) = 1$ imposes that $b_{k+1} - d_{k+1} = 1$ (respectively $b'_{k+1} - d'_{k+1} = 1$) in (12).

We shall now investigate three different recursive algorithms for the implementation of Lanczos' method and show how they lead to transpose-free processes. They are Lancos/Orthores, Lanczos/Orthomin and Lanczos/Orthodir. Each of them will be used with three different choices of the polynomials U_k, namely the power basis, the CGS basis and the BiCGSTAB basis. The power basis will not lead to a coupled implementation of a product-type method. The two other bases allow us to implement simultaneously the CGS and the BiCGSTAB, respectively.

5.1. Transpose-free Lanczos/Orthores

Let us compute the products involved in (6). In the following tables, the left hand columns contain the products computed and the right hand ones the corresponding recurrence relationships.

U_kP_k	$U_{k+1}P_{k+1} = -\eta_{k+1}[(\xi-\gamma_{k+1})U_{k+1}P_k - \delta_{k+1}U_{k+1}P_{k-1}]$
$U_{k+1}P_k$	$U_{k+1}P_k = (a_{k+1}\xi + b_{k+1})U_kP_k - d_{k+1}U_{k-1}P_k$
$U_{k-1}P_k$	$U_kP_{k+1} = -\eta_{k+1}[(\xi-\gamma_{k+1})U_kP_k - \delta_{k+1}U_kP_{k-1}]$
$U_{k+1}P_{k-1}$	$U_{k+1}P_{k-1} = (a_{k+1}\xi + b_{k+1})U_kP_{k-1} - d_{k+1}U_{k-1}P_{k-1}$

Let us set

$$\tilde r_k = U_k(A)P_k(A)r_0, \qquad \tilde u_k = U_k(A)P_{k-1}(A)r_0,$$
$$\tilde s_k = U_{k-1}(A)P_k(A)r_0, \qquad \tilde q_k = U_{k+1}(A)P_{k-1}(A)r_0.$$

The preceding recurrence relationships give

$$\begin{aligned} \tilde r_{k+1} &= -\eta_{k+1}\bigl(A\tilde u_{k+1} - \gamma_{k+1}\tilde u_{k+1} - \delta_{k+1}\tilde q_k\bigr), \\ \tilde u_{k+1} &= (a_{k+1}A + b_{k+1}I)\tilde r_k - d_{k+1}\tilde s_k, \\ \tilde s_{k+1} &= -\eta_{k+1}\bigl[(A-\gamma_{k+1}I)\tilde r_k - \delta_{k+1}\tilde u_k\bigr], \\ \tilde q_k &= (a_{k+1}A + b_{k+1}I)\tilde u_k - d_{k+1}\tilde r_{k-1}. \end{aligned} \tag{13}$$

The products involved in (6) can be computed in a different way, thus leading to a different algorithm. We have:

U_kP_k	$U_{k+1}P_{k+1} = (a_{k+1}\xi + b_{k+1})U_kP_{k+1} - d_{k+1}U_{k-1}P_{k+1}$
U_kP_{k+1}	$U_kP_{k+1} = -\eta_{k+1}[(\xi-\gamma_{k+1})U_kP_k - \delta_{k+1}U_kP_{k-1}]$
U_kP_{k-1}	$U_{k+1}P_k = (a_{k+1}\xi + b_{k+1})U_kP_k - d_{k+1}U_{k-1}P_k$
$U_{k-1}P_{k+1}$	$U_{k-1}P_{k+1} = -\eta_{k+1}[(\xi-\gamma_{k+1})U_{k-1}P_k - \delta_{k+1}U_{k-1}P_{k-1}]$

Let us set

$$\tilde r_k = U_k(A)P_k(A)r_0, \qquad \tilde u_k = U_k(A)P_{k-1}(A)r_0,$$
$$\tilde s_k = U_{k-1}(A)P_k(A)r_0, \qquad \tilde q'_k = U_{k-1}(A)P_{k+1}(A)r_0.$$

Thus, we obtain

$$\begin{aligned} \tilde r_{k+1} &= (a_{k+1}A + b_{k+1}I)\tilde s_{k+1} - d_{k+1}\tilde q'_k, \\ \tilde u_{k+1} &= (a_{k+1}A + b_{k+1}I)\tilde r_k - d_{k+1}\tilde s_k, \\ \tilde s_{k+1} &= -\eta_{k+1}\bigl[(A-\gamma_{k+1}I)\tilde r_k - \delta_{k+1}\tilde u_k\bigr], \\ \tilde q'_k &= -\eta_{k+1}\bigl[(A-\gamma_{k+1}I)\tilde s_k - \delta_{k+1}\tilde r_{k-1}\bigr] \end{aligned} \tag{14}$$

with, from (6),

$$\delta_{k+1} = \frac{(y, A\tilde{s}_k)}{(y, \tilde{r}_{k-1})}, \qquad \gamma_{k+1} = \frac{(y, A\tilde{r}_k) - \delta_{k+1}(y, \tilde{u}_k)}{(y, \tilde{r}_k)},$$
$$\eta_{k+1} = 1/(\gamma_{k+1} + \delta_{k+1}). \tag{15}$$

This algorithm was also given by Ressel and Gutknecht [25] but without any reference to its possible use as a transpose-free algorithm for the implementation of Lanczos' method.

The iterates and the corresponding residuals of Lanczos' method are given by

$$r_{k+1} = -\eta_{k+1}(Ar_k - \gamma_{k+1}r_k - \delta_{k+1}r_{k-1}),$$
$$x_{k+1} = \eta_{k+1}(r_k + \gamma_{k+1}x_k + \delta_{k+1}x_{k-1}).$$

Let us now examine in more detail some particular choices of the auxiliary polynomials U_k. If $U_k(0) = 1$, then, as explained above, $\tilde{r}_k$ is the residual of a Lanczos-type product method which can be implemented simultaneously at almost no extra cost.

TF Lanczos/Orthores and power basis

For the choice $U_{k+1} = \xi U_k$, it holds $\forall k$, $a_{k+1} = 1$, $b_{k+1} = d_{k+1} = 0$. The relations (13) and (15) simplify since we have $\tilde{q}_{k+1} = A\tilde{u}_{k+1}$, $\tilde{r}_k = A\tilde{s}_k$ and $\tilde{u}_{k+1} = A\tilde{r}_k$.

Thus, we obtain the following TF Lanczos/Orthores algorithm.

Algorithm TFres1(A, b, x_0, y)

```
    Initializations
        $r_0 \leftarrow b - Ax_0$
        $\tilde{r}_0 = r_0$
        $r_{-1} = x_{-1} = \tilde{r}_{-1} = \tilde{u}_0 = \tilde{q}_0 = 0$
    for $k = 0, 1, 2, \ldots$ until convergence do
        $\tilde{u}_{k+1} \leftarrow A\tilde{r}_k$
        if $k > 0$ then
            $\delta_{k+1} \leftarrow (y, \tilde{r}_k)/(y, \tilde{r}_{k-1})$
        else
            $\delta_1 \leftarrow 0$
        end if
        $\gamma_{k+1} \leftarrow [(y, \tilde{u}_{k+1}) - \delta_{k+1}(y, \tilde{u}_k)]/(y, \tilde{r}_k)$
        $\eta_{k+1} \leftarrow 1/(\gamma_{k+1} + \delta_{k+1})$
TF      $r_{k+1} \leftarrow -\eta_{k+1}(Ar_k - \gamma_{k+1}r_k - \delta_{k+1}r_{k-1})$
TF      $x_{k+1} \leftarrow \eta_{k+1}(r_k + \gamma_{k+1}x_k + \delta_{k+1}x_{k-1})$
        if $\|r_{k+1}\| \leqslant \varepsilon$ then stop
        $\tilde{q}_{k+1} \leftarrow A\tilde{u}_{k+1}$
    $\tilde r_{k+1} \leftarrow -\eta_{k+1}(\tilde q_{k+1} - \gamma_{k+1}\tilde u_{k+1} - \delta_{k+1}\tilde q_k)$
**end for**
```

This algorithm needs 3 matrix–vector products and 2 inner products.

When using the relations (14), the vector $\tilde q'_k$ is not needed and we obtain another TF Lanczos/Orthores algorithm named TFres2. It consists in replacing, in the previous pseudo-code, the last two lines by

$$\begin{aligned}\tilde s_{k+1} &\leftarrow -\eta_{k+1}(\tilde u_{k+1} - \gamma_{k+1}\tilde r_k - \delta_{k+1}\tilde u_k),\\ \tilde r_{k+1} &\leftarrow A\tilde s_{k+1}.\end{aligned}$$

Again this algorithm needs 3 matrix–vector products and 2 inner products.

TF Lanczos/Orthores and BiCGSTAB basis

For the choice $U_{k+1} = (1 - \theta_{k+1}\xi)U_k$, we have $a_{k+1} = -\theta_{k+1}$, $b_{k+1} = 1$ and $d_{k+1} = 0$. The parameter θ_{k+1} is chosen to minimize the Euclidean norm of $\tilde r_{k+1}$ and, thus, $\tilde r_k$ is the residual of the BiCGSTAB of van der Vorst [28] implemented via the recurrence relationship (5). In (14), $\tilde q'_k$ is no longer necessary, and (15) simplify since $c(\xi U_{k-1}P_k) = -c(\xi U_k P_k)/\theta_k$ (that is, $(y, A\tilde s_k) = -(y, \tilde r_k)/\theta_k$), and $\tilde u_k = \tilde r_{k-1} - \theta_k A\tilde r_{k-1}$.

We have

$$(\tilde r_{k+1}, \tilde r_{k+1}) = (\tilde s_{k+1}, \tilde s_{k+1}) - 2\theta_{k+1}(\tilde s_{k+1}, A\tilde s_{k+1}) + \theta_{k+1}^2(A\tilde s_{k+1}, A\tilde s_{k+1}).$$

Thus, the value of θ_{k+1} minimizing $\|\tilde r_{k+1}\|_2$ is given by

$$\theta_{k+1} = (\tilde s_{k+1}, A\tilde s_{k+1})/(A\tilde s_{k+1}, A\tilde s_{k+1}).$$

Since

$$\tilde r_{k+1} = b - A\tilde x_{k+1} \quad \text{and} \quad \eta_{k+1}(\gamma_{k+1} + \delta_{k+1}) = 1,$$

we have

$$\begin{aligned} b - A\tilde x_{k+1} &= -\eta_{k+1}(A\tilde r_k - \gamma_{k+1}\tilde r_k - \delta_{k+1}\tilde u_k) - \theta_{k+1}A\tilde s_{k+1}\\ &= -\eta_{k+1}(A\tilde r_k - \gamma_{k+1}\tilde r_k - \delta_{k+1}\tilde r_{k-1} + \delta_{k+1}\theta_k A\tilde r_{k-1}) - \theta_{k+1}A\tilde s_{k+1}\\ &= -\eta_{k+1}\left[A\tilde r_k - \gamma_{k+1}(b - A\tilde x_k) - \delta_{k+1}(b - A\tilde x_{k-1}) + \delta_{k+1}\theta_k A\tilde r_{k-1}\right]\\ &\quad - \theta_{k+1}A\tilde s_{k+1}\\ &= \eta_{k+1}(\gamma_{k+1} + \delta_{k+1})b - \eta_{k+1}\left[A\tilde r_k - \gamma_{k+1}A\tilde x_k - \delta_{k+1}A\tilde x_{k-1}\right.\\ &\quad \left. + \delta_{k+1}\theta_k A\tilde r_{k-1}\right] - \theta_{k+1}A\tilde s_{k+1}, \end{aligned}$$

which allows us to compute recursively $\tilde x_{k+1}$.

Finally, the TF Lanczos/Orthores algorithm coupled with the BiCGSTAB algorithm is the following:

```
Algorithm TFresBiCGSTAB($A, b, x_0, y$)
    Initializations
        $r_0 \leftarrow b - A\,x_0$
        $\tilde{r}_0 = r_0$
        $\tilde{x}_0 = x_0$
        $r_{-1} = x_{-1} = \tilde{r}_{-1} = \tilde{x}_{-1} = \tilde{u}_0 = 0$
        $\theta_0 = 0$
    for $k = 0, 1, 2, \ldots$ until convergence do
        if $k > 0$ then
            $\delta_{k+1} \leftarrow -(y, \tilde{r}_k)/[\theta_k(y, \tilde{r}_{k-1})]$
        else
            $\delta_1 \leftarrow 0$
        end if
        $\gamma_{k+1} \leftarrow \dfrac{(y, A\tilde{r}_k) - \delta_{k+1}[(y, \tilde{r}_{k-1}) - \theta_k(y, A\tilde{r}_{k-1})]}{(y, \tilde{r}_k)}$
        $\eta_{k+1} \leftarrow 1/(\gamma_{k+1} + \delta_{k+1})$
TF      $r_{k+1} \leftarrow -\eta_{k+1}(Ar_k - \gamma_{k+1}r_k - \delta_{k+1}r_{k-1})$
TF      $x_{k+1} \leftarrow \eta_{k+1}(r_k + \gamma_{k+1}x_k + \delta_{k+1}x_{k-1})$
        $\tilde{s}_{k+1} \leftarrow -\eta_{k+1}(A\tilde{r}_k - \gamma_{k+1}\tilde{r}_k - \delta_{k+1}\tilde{u}_k)$
        $\theta_{k+1} \leftarrow (\tilde{s}_{k+1}, A\tilde{s}_{k+1})/(A\tilde{s}_{k+1}, A\tilde{s}_{k+1})$
        $\tilde{r}_{k+1} \leftarrow \tilde{s}_{k+1} - \theta_{k+1}A\tilde{s}_{k+1}$
        $\tilde{x}_{k+1} \leftarrow \eta_{k+1}[\tilde{r}_k + \gamma_{k+1}\tilde{x}_k + \delta_{k+1}(\tilde{x}_{k-1} + \theta_k\tilde{r}_{k-1})] + \theta_{k+1}\tilde{s}_{k+1}$
        if $\|r_{k+1}\| \leqslant \varepsilon$ or $\|\tilde{r}_{k+1}\| \leqslant \varepsilon$ then stop
        $\tilde{u}_{k+1} \leftarrow \tilde{r}_k - \theta_{k+1}A\tilde{r}_k$
    end for
```

This algorithm needs 3 matrix–vector products and 4 inner products. Let us mention that using (13) instead of (14) leads to a more complicated expression for $\|\tilde{r}_{k+1}\|_2$ and, thus, for θ_{k+1}.

TF Lanczos/Orthores and CGS basis

For the choice $U_{k+1} \equiv P_{k+1}$, we have $a_{k+1} = -\eta_{k+1}$, $b_{k+1} = \eta_{k+1}\gamma_{k+1}$ and $d_{k+1} = -\eta_{k+1}\delta_{k+1}$. $\tilde{r}_k$ is the residual of the CGS of Sonneveld [27], implemented via the recurrence relationship (5).

With this choice, $\tilde{s}_k = \tilde{u}_k$ and the relations (13) and (14) coincide. The expressions for δ_{k+1} and γ_{k+1} can be simplified and, as shown in [14], we have

$$\begin{aligned}
\delta_{k+1} &= -\frac{1}{\eta_k}\frac{c(P_k^2)}{c(P_{k-1}^2)} = -\frac{1}{\eta_k}\frac{(y, \tilde{r}_k)}{(y, \tilde{r}_{k-1})}, \\
\gamma_{k+1} &= \frac{c(\xi P_k^2)}{c(P_k^2)} = \frac{(y, A\tilde{r}_k)}{(y, \tilde{r}_k)}.
\end{aligned} \tag{16}$$

The recurrence relationship for $\tilde{x}_{k+1}$ cannot be obtained directly. If we set $\tilde{u}_{k+1} = b - A\tilde{y}_{k+1}$, we have

$$\tilde{y}_{k+1} = \eta_{k+1}\big(\tilde{r}_k + \gamma_{k+1}\tilde{x}_k + \delta_{k+1}\tilde{y}_k\big)$$

and, since

$$\eta_{k+1}(\gamma_{k+1} + \delta_{k+1}) = 1 \quad \text{and} \quad \tilde{q}'_k = b - \eta_{k+1}A\big(\tilde{u}_k + \gamma_{k+1}\tilde{y}_k + \delta_{k+1}\tilde{x}_{k-1}\big),$$

we obtain, from $\tilde{r}_{k+1} = -\eta_{k+1}(A\tilde{u}_{k+1} - \gamma_{k+1}\tilde{u}_{k+1} - \delta_{k+1}\tilde{q}'_k)$,

$$\tilde{x}_{k+1} = \eta_{k+1}\big[\tilde{u}_{k+1} + \gamma_{k+1}\tilde{y}_{k+1} + \delta_{k+1}\eta_{k+1}\big(\tilde{u}_k + \gamma_{k+1}\tilde{y}_k + \delta_{k+1}\tilde{x}_{k-1}\big)\big].$$

The TF Lanczos/Orthores algorithm coupled with the CGS algorithm is the following:

Algorithm TFresCGS(A, b, x_0, y)

```
     Initializations
          $r_0 \leftarrow b - A\,x_0$
          $\tilde{r}_0 = r_0$
          $\tilde{x}_0 = x_0$
          $r_{-1} = x_{-1} = \tilde{r}_{-1} = \tilde{x}_{-1} = \tilde{u}_0 = \tilde{y}_0 = \tilde{q}'_0 = 0$
     for $k = 0, 1, 2, \ldots$ until convergence do
          if $k > 0$ then
               $\delta_{k+1} \leftarrow -(y, \tilde{r}_k)/[\eta_k(y, \tilde{r}_{k-1})]$
          else
               $\delta_1 \leftarrow 0$
          end if
          $\gamma_{k+1} \leftarrow (y, A\tilde{r}_k)/(y, \tilde{r}_k)$
          $\eta_{k+1} \leftarrow 1/(\gamma_{k+1} + \delta_{k+1})$
TF        $r_{k+1} \leftarrow -\eta_{k+1}(Ar_k - \gamma_{k+1}r_k - \delta_{k+1}r_{k-1})$
TF        $x_{k+1} \leftarrow \eta_{k+1}(r_k + \gamma_{k+1}x_k + \delta_{k+1}x_{k-1})$
          $\tilde{u}_{k+1} \leftarrow -\eta_{k+1}(A\tilde{r}_k - \gamma_{k+1}\tilde{r}_k - \delta_{k+1}\tilde{u}_k)$
          $\tilde{q}'_k \leftarrow -\eta_{k+1}(A\tilde{u}_k - \gamma_{k+1}\tilde{u}_k - \delta_{k+1}\tilde{r}_{k-1})$
          $\tilde{r}_{k+1} \leftarrow -\eta_{k+1}(A\tilde{u}_{k+1} - \gamma_{k+1}\tilde{u}_{k+1} - \delta_{k+1}\tilde{q}'_k)$
          $\tilde{y}_{k+1} \leftarrow \eta_{k+1}(\tilde{r}_k + \gamma_{k+1}\tilde{x}_k + \delta_{k+1}\tilde{y}_k)$
          $\tilde{x}_{k+1} \leftarrow \eta_{k+1}[\tilde{u}_{k+1} + \gamma_{k+1}\tilde{y}_{k+1} + \delta_{k+1}\eta_{k+1}(\tilde{u}_k + \gamma_{k+1}\tilde{y}_k + \delta_{k+1}\tilde{x}_{k-1})]$
          if $\|r_{k+1}\| \leqslant \varepsilon$ or $\|\tilde{r}_{k+1}\| \leqslant \varepsilon$ then stop
     end for
```

This algorithm needs 3 matrix–vector products and 2 inner products.

5.2. *Transpose-free Lanczos/Orthomin*

Let us now compute the products involved in (8). In the following table we put all the needed products.

U_kP_k	$U_{k+1}P_{k+1} = (a_{k+1}\xi + b_{k+1})U_kP_{k+1} - d_{k+1}U_{k-1}P_{k+1}$ $= (a_{k+1}\xi + b_{k+1})U_kP_{k+1} - d_{k+1}(U_{k-1}P_k - \lambda'_{k+1}\xi U_{k-1}Q_k)$
U_kP_{k+1}	$U_kP_{k+1} = U_kP_k - \lambda'_{k+1}\xi U_kQ_k$
U_kQ_k	$U_{k+1}Q_{k+1} = U_{k+1}P_{k+1} + \alpha_{k+1}U_{k+1}Q_k$
$U_{k+1}Q_k$	$U_{k+1}Q_k = (a_{k+1}\xi + b_{k+1})U_kQ_k - d_{k+1}U_{k-1}Q_k$
$U_{k-1}Q_k$	$U_kQ_{k+1} = U_kP_{k+1} + \alpha_{k+1}U_kQ_k$
V_kP_k	$V_{k+1}P_{k+1} = (a'_{k+1}\xi + b'_{k+1})V_kP_{k+1} - d'_{k+1}V_{k-1}P_{k+1}$ $= (a'_{k+1}\xi + b'_{k+1})V_kP_{k+1} - d'_{k+1}(V_{k-1}P_k - \lambda'_{k+1}\xi V_{k-1}Q_k)$
V_kP_{k+1}	$V_kP_{k+1} = V_kP_k - \lambda'_{k+1}\xi V_kQ_k$
V_kQ_k	$V_{k+1}Q_{k+1} = V_{k+1}P_{k+1} + \alpha_{k+1}V_{k+1}Q_k$
$V_{k+1}Q_k$	$V_{k+1}Q_k = (a'_{k+1}\xi + b'_{k+1})V_kQ_k - d'_{k+1}V_{k-1}Q_k$
$V_{k-1}Q_k$	$V_kQ_{k+1} = V_kP_{k+1} + \alpha_{k+1}V_kQ_k$

Let us set

$$\begin{aligned}
\tilde r_k &= U_k(A)P_k(A)r_0, & \hat r_k &= V_k(A)P_k(A)r_0,\\
\tilde s_k &= U_{k-1}(A)P_k(A)r_0, & \hat s_k &= V_{k-1}(A)P_k(A)r_0,\\
\tilde z_k &= U_k(A)Q_k(A)r_0, & \hat z_k &= V_k(A)Q_k(A)r_0,\\
\tilde v_k &= U_k(A)Q_{k-1}(A)r_0, & \hat v_k &= V_k(A)Q_{k-1}(A)r_0,\\
\tilde p_k &= U_{k-1}(A)Q_k(A)r_0, & \hat p_k &= V_{k-1}(A)Q_k(A)r_0.
\end{aligned}$$

The preceding recurrence relationships give

$$\begin{aligned}
\tilde r_{k+1} &= (a_{k+1}A + b_{k+1}I)\tilde s_{k+1} - d_{k+1}\big(\tilde s_k - \lambda'_{k+1}A\tilde p_k\big),\\
\tilde s_{k+1} &= \tilde r_k - \lambda'_{k+1}A\tilde z_k,\\
\tilde z_{k+1} &= \tilde r_{k+1} + \alpha_{k+1}\tilde v_{k+1},\\
\tilde v_{k+1} &= (a_{k+1}A + b_{k+1}I)\tilde z_k - d_{k+1}\tilde p_k,\\
\tilde p_{k+1} &= \tilde s_{k+1} + \alpha_{k+1}\tilde z_k,\\
\hat r_{k+1} &= \big(a'_{k+1}A + b'_{k+1}I\big)\hat s_{k+1} - d'_{k+1}\big(\hat s_k - \lambda'_{k+1}A\hat p_k\big),\\
\hat s_{k+1} &= \hat r_k - \lambda'_{k+1}A\hat z_k,\\
\hat z_{k+1} &= \hat r_{k+1} + \alpha_{k+1}\hat v_{k+1},
\end{aligned} \tag{17}$$

$$\hat v_{k+1} == \big(a'_{k+1}A + b'_{k+1}I\big)\hat z_k - d'_{k+1}\hat p_k,$$
$$\hat p_{k+1} = \hat s_{k+1} + \alpha_{k+1}\hat z_k.$$

We have

$$r_{k+1} = r_k - \lambda'_{k+1}Ap_k,$$
$$x_{k+1} = x_k + \lambda'_{k+1}p_k,$$
$$p_{k+1} = r_{k+1} + \alpha_{k+1}p_k$$

with $p_0 = r_0 = b - Ax_0$ and, from (8),

$$\lambda'_{k+1} = \frac{(y,\tilde r_k)}{(y, A\tilde z_k)}, \qquad \alpha_{k+1} = -\frac{(y, A\hat s_{k+1})}{(y, A\hat z_k)}. \tag{18}$$

As in the transpose-free Lanczos/Orthores, we can compute some products in a different way, thus leading to a different algorithm.

U_kP_k	$U_{k+1}P_{k+1} = U_{k+1}P_k - \lambda'_{k+1}\xi U_{k+1}Q_k$
	$= (a_{k+1}\xi + b_{k+1})U_kP_k - d_{k+1}U_{k-1}P_k - \lambda'_{k+1}\xi U_{k+1}Q_k$
V_kP_k	$V_{k+1}P_{k+1} = V_{k+1}P_k - \lambda'_{k+1}\xi V_{k+1}Q_k$
	$= (a'_{k+1}\xi + b'_{k+1})V_kP_k - d'_{k+1}V_{k-1}P_k - \lambda'_{k+1}\xi V_{k+1}Q_k$

All the others products needed have already been computed. There are only two recurrence relationships which have to be changed in (17). They are the following:

$$\begin{aligned} \tilde r_{k+1} &= (a_{k+1}A + b_{k+1}I)\tilde r_k - d_{k+1}\tilde s_k - \lambda'_{k+1}A\tilde v_{k+1},\\ \hat r_{k+1} &= \big(a'_{k+1}A + b'_{k+1}I\big)\hat r_k - d'_{k+1}\hat s_k - \lambda'_{k+1}A\hat v_{k+1}. \end{aligned} \tag{19}$$

The same idea, but with polynomials U_k and/or V_k satisfying recurrence relationships different from (12), is studied in [18] (see also [17]).

Let us now look at some particular choices of U_k and V_k in more detail. We will always take $U_k \equiv V_k$ and, so, the tilde and the hat vectors coincide.

TF Lanczos/Orthomin and power basis

For the choice $U_{k+1} \equiv V_{k+1}$, $U_{k+1} = \xi U_k$, we have $a_{k+1} = 1$, $b_{k+1} = d_{k+1} = 0$. From (17) and (18) we see that the vector $\tilde p_k$ is not used. Moreover, $\tilde v_{k+1} = A\tilde z_k$ and $\tilde r_{k+1} = A\tilde s_{k+1}$.

Then, the TF Lanczos/Orthomin algorithm is the following:

Algorithm TFmin(A, b, x_0, y)

```
  Initializations
    $r_0 \leftarrow b - A\,x_0$
    $p_0 = \tilde r_0 = \tilde z_0 = r_0$
    for $k = 0, 1, 2, \ldots$ until convergence do
        $\lambda'_{k+1} \leftarrow (y, \tilde{r}_k)/(y, A\tilde{z}_k)$
TF      $r_{k+1} \leftarrow r_k - \lambda'_{k+1} A p_k$
TF      $x_{k+1} \leftarrow x_k + \lambda'_{k+1} p_k$
        if $\|r_{k+1}\| \leqslant \varepsilon$ then stop
        $\tilde{s}_{k+1} \leftarrow \tilde{r}_k - \lambda'_{k+1} A\tilde{z}_k$
        $\tilde{r}_{k+1} \leftarrow A\tilde{s}_{k+1}$
        $\alpha_{k+1} \leftarrow -(y, \tilde{r}_{k+1})/(y, A\tilde{z}_k)$
        $\tilde{z}_{k+1} \leftarrow \tilde{r}_{k+1} + \alpha_{k+1} A\tilde{z}_k$
TF      $p_{k+1} \leftarrow r_{k+1} + \alpha_{k+1} p_k$
    end for
```

This algorithm needs 3 matrix–vector products and 2 inner products.

When, instead, we consider the relations (19), we obtain another algorithm, where $\tilde{r}_{k+1} = A\tilde{r}_k - \lambda'_{k+1} A\tilde{v}_{k+1}$, but it needs an additional matrix–vector product.

TF Lanczos/Orthomin and BiCGSTAB basis

For the choice $U_{k+1} \equiv V_{k+1}$, $U_{k+1} = (1 - \theta_{k+1}\xi)U_k$, we have $a_{k+1} = -\theta_{k+1}$, $b_{k+1} = 1$ and $d_{k+1} = 0$. In this case, and if θ_{k+1} is chosen as above, $\tilde{r}_k$ is the residual of the BiCGSTAB as proposed by van der Vorst in [28] which makes use of the relationship (7). The relations (17) simplify and, by replacing $\tilde{v}_{k+1}$ by its expression in $\tilde{z}_{k+1}$, become

$$\begin{aligned} \tilde{s}_{k+1} &= \tilde{r}_k - \lambda'_{k+1} A\tilde{z}_k, \\ \tilde{r}_{k+1} &= \tilde{s}_{k+1} - \theta_{k+1} A\tilde{s}_{k+1}, \\ \tilde{z}_{k+1} &= \tilde{r}_{k+1} + \alpha_{k+1}(\tilde{z}_k - \theta_{k+1} A\tilde{z}_k). \end{aligned} \tag{20}$$

Since $c(\xi U_k P_{k+1}) = -c(U_{k+1}P_{k+1})/\theta_{k+1}$ (that is, $(y, A\tilde{s}_{k+1}) = -(y, \tilde{r}_{k+1})/\theta_{k+1}$), the expression for α_{k+1} in (18) is changed. Moreover, the relation for $\tilde{r}_{k+1}$ is the same as in the algorithm TFresBiCGSTAB, and thus the expression for θ_{k+1} does not change.

We have, from (20),

$$\begin{aligned} \tilde{r}_{k+1} &= b - A\tilde{x}_{k+1} = \tilde{r}_k - \lambda'_{k+1} A\tilde{z}_k - \theta_{k+1} A\tilde{s}_{k+1} \\ &= b - A\tilde{x}_k - \lambda'_{k+1} A\tilde{z}_k - \theta_{k+1} A\tilde{s}_{k+1} \end{aligned}$$

which allows us to compute $\tilde{x}_{k+1}$.

The TF Lanczos/Orthomin algorithm coupled with the BiCGSTAB algorithm is the following:

Algorithm TFminBiCGSTAB(A, b, x_0, y)

```
    Initializations
        $r_0 \leftarrow b - A x_0$
```

```
          $p_0 = \tilde r_0 = \tilde z_0 = r_0$
          $\tilde x_0 = x_0$
      **for** $k = 0, 1, 2, \ldots$ **until** convergence **do**
          $\lambda'_{k+1} \leftarrow (y, \tilde r_k)/(y, A\tilde z_k)$
TF        $r_{k+1} \leftarrow r_k - \lambda'_{k+1} A p_k$
TF        $x_{k+1} \leftarrow x_k + \lambda'_{k+1} p_k$
          $\tilde s_{k+1} \leftarrow \tilde r_k - \lambda'_{k+1} A\tilde z_k$
          $\theta_{k+1} \leftarrow (\tilde s_{k+1}, A\tilde s_{k+1})/(A\tilde s_{k+1}, A\tilde s_{k+1})$
          $\tilde r_{k+1} \leftarrow \tilde s_{k+1} - \theta_{k+1} A\tilde s_{k+1}$
          $\tilde x_{k+1} \leftarrow \tilde x_k + \lambda'_{k+1}\tilde z_k + \theta_{k+1}\tilde s_{k+1}$
          **if** $\|r_{k+1}\| \leqslant \varepsilon$ **or** $\|\tilde r_{k+1}\| \leqslant \varepsilon$ **then** stop
          $\alpha_{k+1} \leftarrow (y, \tilde r_{k+1})/[\theta_{k+1}(y, A\tilde z_k)]$
          $\tilde z_{k+1} \leftarrow \tilde r_{k+1} + \alpha_{k+1}(\tilde z_k - \theta_{k+1} A\tilde z_k)$
TF        $p_{k+1} \leftarrow r_{k+1} + \alpha_{k+1} p_k$
      **end for**
```

This algorithm needs 3 matrix–vector products and 4 inner products.

Taking $\theta_{k+1} = -1$ in this algorithm, we recover the TEA2/Orthomin given in [14]. Conversely, the TEA2/Orthomin leads to a transpose-free algorithm for implementing Lanczos' method. Similarly, a TEA2/Orthores and a TEA2/Orthodir algorithms could be derived.

TF Lanczos/Orthomin (BCG) and CGS basis

For the choice $U_{k+1} \equiv V_{k+1}$ and $U_{k+1} = P_{k+1}$, r_k is the residual of the BCG algorithm and $\tilde r_k$ is exactly the residual of the CGS proposed by Sonneveld in [27]. Again, with this choice, the expressions for λ'_{k+1} and α_{k+1} can be simplified and, as shown in [14], we have

$$\lambda'_{k+1} = c\big(P_k^2\big)/c\big(\xi Q_k^2\big), \qquad \alpha_{k+1} = c\big(P_{k+1}^2\big)/c\big(P_k^2\big). \tag{21}$$

However, in this case, we cannot directly use the relations given in section 5.2 because P_{k+1} does not satisfy a recurrence relationship of the form (12). Thus we have to compute the products involved in (21) as follows:

P_k^2	$P_{k+1}^2 = P_{k+1}P_k - \lambda'_{k+1}\xi P_{k+1}Q_k$
	$\quad = P_k^2 - \lambda'_{k+1}\xi P_k Q_k - \lambda'_{k+1}\xi P_{k+1}Q_k$
P_kQ_k	$P_{k+1}Q_{k+1} = P_{k+1}^2 + \alpha_{k+1}P_{k+1}Q_k$
$P_{k+1}Q_k$	$P_{k+1}Q_k = P_kQ_k - \lambda'_{k+1}\xi Q_k^2$
Q_k^2	$Q_{k+1}^2 = P_{k+1}Q_{k+1} + \alpha_{k+1}Q_kQ_{k+1}$
	$\quad = P_{k+1}Q_{k+1} + \alpha_{k+1}(P_{k+1}Q_k + \alpha_{k+1}Q_k^2)$

Let us set

$$\tilde r_k = P_k^2(A)r_0, \qquad \tilde z_k = P_k(A)Q_k(A)r_0,$$
$$\tilde v_k = P_k(A)Q_{k-1}(A)r_0, \qquad \tilde t_k = Q_k^2(A)r_0.$$

The preceding recurrence relationships give

$$\begin{aligned}\tilde r_{k+1} &= \tilde r_k - \lambda'_{k+1}A\big(\tilde z_k + \tilde v_{k+1}\big),\\ \tilde z_{k+1} &= \tilde r_{k+1} + \alpha_{k+1}\tilde v_{k+1},\\ \tilde v_{k+1} &= \tilde z_k - \lambda'_{k+1}A\tilde t_k,\\ \tilde t_{k+1} &= \tilde z_{k+1} + \alpha_{k+1}\big(\tilde v_{k+1} + \alpha_{k+1}\tilde t_k\big).\end{aligned}$$

From the expression of $\tilde r_{k+1}$, we immediately obtain $\tilde x_{k+1}$.

The TF Lanczos/Orthomin algorithm coupled with the CGS algorithm is the following:

```
**Algorithm** TFminCGS($A, b, x_0, y$)
      **Initializations**
            $r_0 \leftarrow b - A\,x_0$
            $p_0 = \tilde r_0 = \tilde z_0 = \tilde t_0 = r_0$
            $\tilde x_0 = x_0$
      **for** $k = 0, 1, 2, \ldots$ **until** convergence **do**
            $\lambda'_{k+1} \leftarrow (y, \tilde r_k)/(y, A\tilde t_k)$
TF          $r_{k+1} \leftarrow r_k - \lambda'_{k+1}Ap_k$
TF          $x_{k+1} \leftarrow x_k + \lambda'_{k+1}p_k$
            $\tilde v_{k+1} \leftarrow \tilde z_k - \lambda'_{k+1}A\tilde t_k$
            $\tilde r_{k+1} \leftarrow \tilde r_k - \lambda'_{k+1}A(\tilde z_k + \tilde v_{k+1})$
            $\tilde x_{k+1} \leftarrow \tilde x_k + \lambda'_{k+1}(\tilde z_k + \tilde v_{k+1})$
            **if** $\|r_{k+1}\| \leqslant \varepsilon$ **or** $\|\tilde r_{k+1}\| \leqslant \varepsilon$ **then** stop
            $\alpha_{k+1} \leftarrow (y, \tilde r_{k+1})/(y, \tilde r_k)$
            $\tilde z_{k+1} \leftarrow \tilde r_{k+1} + \alpha_{k+1}\tilde v_{k+1}$
            $\tilde t_{k+1} \leftarrow \tilde z_{k+1} + \alpha_{k+1}(\tilde v_{k+1} + \alpha_{k+1}\tilde t_k)$
TF          $p_{k+1} \leftarrow r_{k+1} + \alpha_{k+1}p_k$
      **end for**
```

This algorithm needs 3 matrix–vector products and 2 inner products and it is identical to the TFiBiCG algorithm obtained by Chan et al. in [15].

5.3. *Transpose-free Lanczos/Orthodir*

Let us compute the products involved in (10). There are several possibilities, which are as follows:

U_kP_k	$U_{k+1}P_{k+1} = (a_{k+1}\xi + b_{k+1})U_kP_{k+1} - d_{k+1}U_{k-1}P_{k+1}$ or
	$U_{k+1}P_{k+1} = U_{k+1}P_k - \lambda_{k+1}\xi U_{k+1}P_k^{(1)}$
U_kP_{k+1}	$U_kP_{k+1} = U_kP_k - \lambda_{k+1}\xi U_kP_k^{(1)}$
$U_{k+1}P_k$	$U_{k+1}P_k = (a_{k+1}\xi + b_{k+1})U_kP_k - d_{k+1}U_{k-1}P_k$
$U_{k-1}P_{k+1}$	$U_{k-1}P_{k+1} = U_{k-1}P_k - \lambda_{k+1}\xi U_{k-1}P_k^{(1)}$
$U_{k-1}P_k^{(1)}$	$U_kP_{k+1}^{(1)} = (\xi + \beta_{k+1})U_kP_k^{(1)} - \gamma_{k+1}U_kP_{k-1}^{(1)}$
$U_{k+1}P_k^{(1)}$	$U_{k+1}P_k^{(1)} = (a_{k+1}\xi + b_{k+1})U_kP_k^{(1)} - d_{k+1}U_{k-1}P_k^{(1)}$
$U_kP_k^{(1)}$	$U_{k+1}P_{k+1}^{(1)} = (a_{k+1}\xi + b_{k+1})U_kP_{k+1}^{(1)} - d_{k+1}U_{k-1}P_{k+1}^{(1)}$ or
	$U_{k+1}P_{k+1}^{(1)} = (\xi + \beta_{k+1})U_{k+1}P_k^{(1)} - \gamma_{k+1}U_{k+1}P_{k-1}^{(1)}$
$U_{k-1}P_{k+1}^{(1)}$	$U_{k-1}P_{k+1}^{(1)} = (\xi + \beta_{k+1})U_{k-1}P_k^{(1)} - \gamma_{k+1}U_{k-1}P_{k-1}^{(1)}$
$U_{k+1}P_{k-1}^{(1)}$	$U_{k+1}P_{k-1}^{(1)} = (a_{k+1}\xi + b_{k+1})U_kP_{k-1}^{(1)} - d_{k+1}U_{k-1}P_{k-1}^{(1)}$
$V_{k-1}P_k^{(1)}$	$V_kP_{k+1}^{(1)} = (\xi + \beta_{k+1})V_kP_k^{(1)} - \gamma_{k+1}V_kP_{k-1}^{(1)}$
$V_kP_{k-1}^{(1)}$	$V_{k+1}P_k^{(1)} = (a'_{k+1}\xi + b'_{k+1})V_kP_k^{(1)} - d'_{k+1}V_{k-1}P_k^{(1)}$
$V_kP_k^{(1)}$	$V_{k+1}P_{k+1}^{(1)} = (a'_{k+1}\xi + b'_{k+1})V_kP_{k+1}^{(1)} - d'_{k+1}V_{k-1}P_{k+1}^{(1)}$ or
	$V_{k+1}P_{k+1}^{(1)} = (\xi + \beta_{k+1})V_{k+1}P_k^{(1)} - \gamma_{k+1}V_{k+1}P_{k-1}^{(1)}$
$V_{k-1}P_{k+1}^{(1)}$	$V_{k-1}P_{k+1}^{(1)} = (\xi + \beta_{k+1})V_{k-1}P_k^{(1)} - \gamma_{k+1}V_{k-1}P_{k-1}^{(1)}$
$V_{k+1}P_{k-1}^{(1)}$	$V_{k+1}P_{k-1}^{(1)} = (a'_{k+1}\xi + b'_{k+1})V_kP_{k-1}^{(1)} - d'_{k+1}V_{k-1}P_{k-1}^{(1)}$

Let us set

$$\begin{aligned}
\tilde r_k &= U_k(A)P_k(A)r_0, & \tilde s_k &= U_{k-1}(A)P_k(A)r_0,\\
\tilde u_k &= U_k(A)P_{k-1}(A)r_0, & \tilde q'_k &= U_{k-1}(A)P_{k+1}(A)r_0,\\
\tilde p_k &= U_{k-1}(A)P_k^{(1)}(A)r_0, & \hat p_k &= V_{k-1}(A)P_k^{(1)}(A)r_0,\\
\tilde v_k &= U_k(A)P_{k-1}^{(1)}(A)r_0, & \hat v_k &= V_k(A)P_{k-1}^{(1)}(A)r_0,\\
\tilde z_k &= U_k(A)P_k^{(1)}(A)r_0, & \hat z_k &= V_k(A)P_k^{(1)}(A)r_0,\\
\tilde p'_k &= U_{k-1}(A)P_{k+1}^{(1)}(A)r_0, & \hat p'_k &= V_{k-1}(A)P_{k+1}^{(1)}(A)r_0,\\
\tilde v'_k &= U_{k+1}(A)P_{k-1}^{(1)}(A)r_0, & \hat v'_k &= V_{k+1}(A)P_{k-1}^{(1)}(A)r_0.
\end{aligned}$$

By combining all the possibilities, we obtain eight different algorithms that can be schematized as follows:

$$\begin{array}{l|l} \tilde r_{k+1} = (a_{k+1}A + b_{k+1}I)\tilde s_{k+1} - d_{k+1}\tilde q'_k, & \tilde r_{k+1} = \tilde u_{k+1} - \lambda_{k+1}A\tilde v_{k+1}, \\ \tilde q'_k = \tilde s_k - \lambda_{k+1}A\tilde p_k, & \tilde u_{k+1} = (a_{k+1}A + b_{k+1}I)\tilde r_k - d_{k+1}\tilde s_k, \end{array}$$

$$\begin{aligned} \tilde s_{k+1} &= \tilde r_k - \lambda_{k+1}A\tilde z_k \\ \tilde p_{k+1} &= (A + \beta_{k+1}I)\tilde z_k - \gamma_{k+1}\tilde v_k \\ \tilde v_{k+1} &= (a_{k+1}A + b_{k+1}I)\tilde z_k - d_{k+1}\tilde p_k \end{aligned}$$

$$\begin{array}{l|l} \tilde z_{k+1} = (a_{k+1}A + b_{k+1}I)\tilde p_{k+1} - d_{k+1}\tilde p'_k, & \tilde z_{k+1} = (A + \beta_{k+1}I)\tilde v_{k+1} - \gamma_{k+1}\tilde v'_k, \\ \tilde p'_k = (A + \beta_{k+1}I)\tilde p_k - \gamma_{k+1}\tilde z_{k-1}, & \tilde v'_k = (a_{k+1}A + b_{k+1}I)\tilde v_k - d_{k+1}\tilde z_{k-1}, \end{array}$$

$$\begin{aligned} \hat p_{k+1} &= (A + \beta_{k+1}I)\hat z_k - \gamma_{k+1}\hat v_k \\ \hat v_{k+1} &= (a'_{k+1}A + b'_{k+1}I)\hat z_k - d'_{k+1}\hat p_k \end{aligned}$$

$$\begin{array}{l|l} \hat z_{k+1} = (a'_{k+1}A + b'_{k+1}I)\hat p_{k+1} - d'_{k+1}\hat p'_k, & \hat z_{k+1} = (A + \beta_{k+1}I)\hat v_{k+1} - \gamma_{k+1}\hat v'_k, \\ \hat p'_k = (A + \beta_{k+1}I)\hat p_k - \gamma_{k+1}\hat z_{k-1}, & \hat v'_k = (a'_{k+1}A + b'_{k+1}I)\hat v_k - d'_{k+1}\hat z_{k-1}. \end{array}$$

We have

$$\begin{aligned} r_{k+1} &= r_k - \lambda_{k+1}Az_k, \\ x_{k+1} &= x_k + \lambda_{k+1}z_k, \\ z_{k+1} &= (A + \beta_{k+1}I)z_k - \gamma_{k+1}z_{k-1} \end{aligned}$$

with $z_0 = r_0 = b - Ax_0$ and $z_{-1} = 0$, and from (10)

$$\begin{aligned} \lambda_{k+1} &= \frac{(y, \tilde r_k)}{(y, A\tilde z_k)}, \\ \gamma_{k+1} &= \frac{(y, A^2\hat p_k)}{(y, A\hat z_{k-1})}, \\ \beta_{k+1} &= \frac{\gamma_{k+1}(y, A\hat v_k) - (y, A^2\hat z_k)}{(y, A\hat z_k)}. \end{aligned} \tag{22}$$

Again, let us look at some particular choices of U_k and V_k in more detail. When $\forall k,\ U_k \equiv V_k$, the hat vectors coincide with the tilde vectors.

TF Lanczos/Orthodir and power basis

For the choice $V_{k+1} \equiv U_{k+1}$, and $U_{k+1} = \xi U_k$, we have $a_{k+1} = 1$, $b_{k+1} = d_{k+1} = 0$.

Choosing, among all the possible preceding algorithms, those with the minimal number of matrix–vector products, we obtain two different algorithms.

The first one is based on the following relations:

$$\begin{aligned}\tilde{v}_{k+1} &= A\tilde{z}_k,\\ \tilde{r}_{k+1} &= A\tilde{r}_k - \lambda_{k+1}A\tilde{v}_{k+1},\\ \tilde{z}_{k+1} &= (A+\beta_{k+1}I)\tilde{v}_{k+1} - \gamma_{k+1}A\tilde{v}_k.\end{aligned}$$

Since $(y, A^2\tilde{p}_k) = (y, A\tilde{z}_k) = (y, \tilde{v}_{k+1})$ and $(y, A^2\tilde{z}_k) = (y, A\tilde{v}_{k+1})$, the expressions of the scalars from (22) follow.

The first TF Lanczos/Orthodir algorithm is the following:

```
Algorithm TFdir1($A, b, x_0, y$)
    Initializations
        $r_0 \leftarrow b - A\,x_0$
        $z_0 = \tilde{r}_0 = \tilde{z}_0 = r_0$
        $z_{-1} = \tilde{v}_0 = 0$
    for $k = 0, 1, 2, \ldots$ until convergence do
        $\tilde{v}_{k+1} \leftarrow A\tilde{z}_k$
        $\lambda_{k+1} \leftarrow (y, \tilde{r}_k)/(y, \tilde{v}_{k+1})$
TF      $r_{k+1} \leftarrow r_k - \lambda_{k+1}Az_k$
TF      $x_{k+1} \leftarrow x_k + \lambda_{k+1}z_k$
        if $\|r_{k+1}\| \leqslant \varepsilon$ then stop
        $\tilde{r}_{k+1} \leftarrow A\tilde{r}_k - \lambda_{k+1}A\tilde{v}_{k+1}$
        if $k > 0$ then
            $\gamma_{k+1} \leftarrow (y, \tilde{v}_{k+1})/(y, \tilde{v}_k)$
        else
            $\gamma_1 \leftarrow 0$
        end if
        $\beta_{k+1} \leftarrow [\gamma_{k+1}(y, A\tilde{v}_k) - (y, A\tilde{v}_{k+1})]/(y, \tilde{v}_{k+1})$
        $\tilde{z}_{k+1} \leftarrow A\tilde{v}_{k+1} + \beta_{k+1}\tilde{v}_{k+1} - \gamma_{k+1}A\tilde{v}_k$
TF      $z_{k+1} \leftarrow Az_k + \beta_{k+1}z_k - \gamma_{k+1}z_{k-1}$
    end for
```

This algorithm needs 4 matrix–vector products and 3 inner products.

Setting $\tilde{s}_{k+1} = \tilde{r}_k - \lambda_{k+1}\tilde{v}_{k+1}$, $\tilde{r}_{k+1}$ can also be computed by $\tilde{r}_{k+1} = A\tilde{s}_{k+1}$. This algorithm, named TFdir2, differs from the previous one only for the computation of $\tilde{r}_{k+1}$. It needs the same number of matrix–vector products and of inner products and seems to be slightly more stable than the previous one.

TF Lanczos/Orthodir and BiCGSTAB basis

For the choice $U_{k+1} \equiv V_{k+1}$, $U_{k+1} = (1 - \theta_{k+1}\xi)U_k$, we have $a_{k+1} = -\theta_{k+1}$, $b_{k+1} = 1$ and $d_{k+1} = 0$.

Choosing the algorithm needing the minimal computational cost, and substituting $\tilde{v}'_k$ in $\tilde{z}_{k+1}$, we use the following relations:

$$\begin{aligned}\tilde{s}_{k+1} &= \tilde{r}_k - \lambda_{k+1} A\tilde{z}_k,\\ \tilde{r}_{k+1} &= \tilde{s}_{k+1} - \theta_{k+1} A\tilde{s}_{k+1},\\ \tilde{v}_{k+1} &= \tilde{z}_k - \theta_{k+1} A\tilde{z}_k,\\ \tilde{z}_{k+1} &= A\tilde{v}_{k+1} + \beta_{k+1}\tilde{v}_{k+1} - \gamma_{k+1}\bigl(\tilde{v}_k - \theta_{k+1}A\tilde{v}_k\bigr).\end{aligned} \tag{23}$$

Since

$$c\bigl(\xi^2 U_{k-1} P_k^{(1)}\bigr) = -c\bigl(\xi U_k P_k^{(1)}\bigr)/\theta_k$$

(that is, $(y, A^2\tilde{p}_k) = -(y, A\tilde{z}_k)/\theta_k$), and

$$c\bigl(\xi^2 U_k P_k^{(1)}\bigr) = \bigl[c\bigl(\xi U_k P_k^{(1)}\bigr) - c\bigl(\xi U_{k+1} P_k^{(1)}\bigr)\bigr]/\theta_{k+1}$$

(that is, $(y, A^2\tilde{z}_k) = [(y, A\tilde{z}_k) - (y, A\tilde{v}_{k+1})]/\theta_{k+1}$), we obtain, from (22), the expressions for γ_{k+1}, β_{k+1} and λ_{k+1}.

The expressions for $\tilde{s}_{k+1}$ and $\tilde{r}_{k+1}$ are the same as in TFminBiCGSTAB. Thus, using λ_{k+1} instead λ'_{k+1}, also the expressions for θ_{k+1} and $\tilde{x}_{k+1}$ are the same.

A first TF Lanczos/Orthodir algorithm coupled with the BiCGSTAB is:

```
Algorithm TFdirBiCGSTAB1($A, b, x_0, y$)

    Initializations
        $r_0 \leftarrow b - A\,x_0$
        $z_0 = \tilde{r}_0 = \tilde{z}_0 = r_0$
        $\tilde{x}_0 = x_0$
        $z_{-1} = \tilde{z}_{-1} = \tilde{v}_0 = 0$
        $\theta_0 = 0$
    for $k = 0, 1, 2, \ldots$ until convergence do
        $\lambda_{k+1} \leftarrow (y, \tilde{r}_k)/(y, A\tilde{z}_k)$
TF      $r_{k+1} \leftarrow r_k - \lambda_{k+1} A z_k$
TF      $x_{k+1} \leftarrow x_k + \lambda_{k+1} z_k$
        $\tilde{s}_{k+1} \leftarrow \tilde{r}_k - \lambda_{k+1} A\tilde{z}_k$
        $\theta_{k+1} \leftarrow (\tilde{s}_{k+1}, A\tilde{s}_{k+1})/(A\tilde{s}_{k+1}, A\tilde{s}_{k+1})$
        $\tilde{r}_{k+1} \leftarrow \tilde{s}_{k+1} - \theta_{k+1} A\tilde{s}_{k+1}$
        $\tilde{x}_{k+1} \leftarrow \tilde{x}_k + \lambda_{k+1}\tilde{z}_k + \theta_{k+1}\tilde{s}_{k+1}$
        if $\|r_{k+1}\| \leqslant \varepsilon$ or $\|\tilde{r}_{k+1}\| \leqslant \varepsilon$ then stop
        if $k > 0$ then
            $\gamma_{k+1} \leftarrow -(y, A\tilde{z}_k)/[\theta_k (y, A\tilde{z}_{k-1})]$
        else
```

```
            $\gamma_1 \leftarrow 0$
        end if
        $\tilde{v}_{k+1} \leftarrow \tilde{z}_k - \theta_{k+1} A\tilde{z}_k$
        $\beta_{k+1} \leftarrow [\gamma_{k+1}\theta_{k+1}(y, A\tilde{v}_k) - (y, A\tilde{z}_k) + (y, A\tilde{v}_{k+1})]/[\theta_{k+1}(y, A\tilde{z}_k)]$
        $\tilde{z}_{k+1} \leftarrow A\tilde{v}_{k+1} + \beta_{k+1}\tilde{v}_{k+1} - \gamma_{k+1}(\tilde{v}_k - \theta_{k+1}A\tilde{v}_k)$
TF      $z_{k+1} \leftarrow Az_k + \beta_{k+1}z_k - \gamma_{k+1}z_{k-1}$
    end for
```

This algorithm needs 4 matrix–vector products and 5 inner products.

The products $A\tilde{v}_k$, which are needed in this algorithm, can be computed recursively. Indeed, setting $\widetilde{w}_k = A\tilde{v}_k$, we have a new algorithm, called TFdirBiCGSTAB2 obtained by replacing, in the previous pseudo-code, the expressions for β_{k+1} and $\tilde{z}_{k+1}$ by

$$\begin{aligned}
\beta_{k+1} &\leftarrow \bigl[\gamma_{k+1}\bigl(y, \widetilde{w}_k\bigr) - \bigl(y, A^2\tilde{z}_k\bigr)\bigr]/\bigl(y, A\tilde{z}_k\bigr),\\
\widetilde{w}_{k+1} &\leftarrow A\tilde{z}_k - \theta_{k+1}A^2\tilde{z}_k,\\
\tilde{z}_{k+1} &\leftarrow \widetilde{w}_{k+1} + \beta_{k+1}\tilde{v}_{k+1} - \gamma_{k+1}\bigl(\tilde{v}_k - \theta_{k+1}\widetilde{w}_k\bigr).
\end{aligned}$$

In the inizializations, $\widetilde{w}_0$ must be set to zero. This variant again needs 4 matrix–vector products but now 6 inner products.

TF Lanczos/Orthodir and CGS basis

For the choice $V_{k+1} \equiv P^{(1)}_{k+1}$, we have $a'_{k+1} = 1$, $b'_{k+1} = \beta_{k+1}$ and $d'_{k+1} = \gamma_{k+1}$. Moreover, $\hat{v}_k = \hat{p}_k$, the two expressions for $\hat{z}_k$ coincide and $\hat{v}'_k = \hat{p}'_k$. The expressions for γ_{k+1} and β_{k+1} can be simplified and, as shown in [14], we have

$$\begin{aligned}
\gamma_{k+1} &= c\bigl(\xi P^{(1)^2}_k\bigr)/c\bigl(\xi P^{(1)^2}_{k-1}\bigr) = (y, A\hat{z}_k)/(y, A\hat{z}_{k-1}),\\
\beta_{k+1} &= -c\bigl(\xi^2 P^{(1)^2}_k\bigr)/c\bigl(\xi P^{(1)^2}_k\bigr) = -\bigl(y, A^2\hat{z}_k\bigr)/(y, A\hat{z}_k)
\end{aligned} \tag{24}$$

with $\hat{z}_k = P^{(1)^2}_k(A)r_0$.

Now, for the choice $U_{k+1} \equiv P_{k+1}$, $\tilde{r}_k$ is the residual of the CGS implemented via the recurrence relationships (10) and $\tilde{s}_k = \tilde{u}_k$. However, as in the TFminCGS, we cannot use directly the algorithms given in section 5.3 since P_{k+1} is not computed by a relation of the same form as (12).

Since $\lambda_{k+1} = c(P^2_k)/c(\xi P_k P^{(1)}_k) = (y, \tilde{r}_k)/(y, A\tilde{z}_k)$, we need to compute the following products of polynomials:

P_k^2	$\begin{aligned} P_{k+1}^2 &= P_{k+1}P_k - \lambda_{k+1}\xi P_{k+1}P_k^{(1)} \\ &= P_k^2 - \lambda_{k+1}\xi P_k P_k^{(1)} - \lambda_{k+1}\xi P_{k+1}P_k^{(1)} \end{aligned}$
$P_{k+1}P_k^{(1)}$	$P_{k+1}P_k^{(1)} = P_kP_k^{(1)} - \lambda_{k+1}\xi P_k^{(1)^2}$
$P_kP_k^{(1)}$	$\begin{aligned} P_{k+1}P_{k+1}^{(1)} &= (\xi + \beta_{k+1})P_{k+1}P_k^{(1)} - \gamma_{k+1}P_{k+1}P_{k-1}^{(1)} \\ &= (\xi + \beta_{k+1})P_{k+1}P_k^{(1)} - \gamma_{k+1}(P_kP_{k-1}^{(1)} - \lambda_{k+1}\xi P_k^{(1)}P_{k-1}^{(1)}) \text{ or} \\ P_{k+1}P_{k+1}^{(1)} &= P_kP_{k+1}^{(1)} - \lambda_{k+1}\xi P_k^{(1)}P_{k+1}^{(1)} \\ &= (\xi + \beta_{k+1})P_kP_k^{(1)} - \gamma_{k+1}P_kP_{k-1}^{(1)} - \lambda_{k+1}\xi P_k^{(1)}P_{k+1}^{(1)} \end{aligned}$

In addition, we need the products $P_k^{(1)^2}$, $P_k^{(1)}P_{k-1}^{(1)}$ (and consequentely for computing these products, also $P_{k-1}^{(1)}P_{k+1}^{(1)}$), but they correspond, respectively, to $\hat{z}_k$, $\hat{v}_k$ and $\hat{v}'_k$. Replacing $\hat{v}'_k$ by its expression in $\hat{z}_{k+1}$ allows us to suppress it from the relation and, thus, finally, we have the two following algorithms:

$$\begin{aligned}
\tilde{r}_{k+1} &= \tilde{r}_k - \lambda_{k+1}A\big(\tilde{z}_k + \tilde{v}_{k+1}\big), \\
\tilde{v}_{k+1} &= \tilde{z}_k - \lambda_{k+1}A\hat{z}_k, \\
\hat{v}_{k+1} &= A\hat{z}_k + \beta_{k+1}\hat{z}_k - \gamma_{k+1}\hat{v}_k, \\
\hat{z}_{k+1} &= (A + \beta_{k+1}I)\big(\hat{v}_{k+1} - \gamma_{k+1}\hat{v}_k\big) + \gamma_{k+1}^2\hat{z}_{k-1}, \\
&\left.\begin{aligned}
\tilde{z}_{k+1} &= (A + \beta_{k+1}I)\tilde{v}_{k+1} - \gamma_{k+1}\big(\tilde{v}_k - \lambda_{k+1}A\hat{v}_k\big), \\
&\quad \text{or equivalently,} \\
\tilde{z}_{k+1} &= (A + \beta_{k+1}I)\tilde{z}_k - \gamma_{k+1}\tilde{v}_k - \lambda_{k+1}A\hat{v}_{k+1}.
\end{aligned}\right\rbrace
\end{aligned} \tag{25}$$

Summing up the two relations (25), we have

$$\tilde{z}_{k+1} = 1/2(A + \beta_{k+1}I)\big(\tilde{z}_k + \tilde{v}_{k+1}\big) - \gamma_{k+1}\tilde{v}_k - \lambda_{k+1}/2A\big(\hat{v}_{k+1} - \gamma_{k+1}\hat{v}_k\big).$$

Since $c(\xi P_{k-1}P_k^{(1)}) = 0$ and $c(\xi^2 P_k^{(1)}P_{k-1}^{(1)}) = c(\xi P_k^{(1)^2})$, we have $(y, A\tilde{p}_k) = 0$ and $(y, A^2\hat{v}_k) = (y, A\hat{z}_k)$. Thus, from $\tilde{z}_k = \tilde{p}_k - \lambda_k A\hat{v}_k$, it follows

$$\lambda_{k+1} = -\big(y, \tilde{r}_k\big)/\big[\lambda_k(y, A\hat{z}_k)\big].$$

Since $\tilde{v}_{k+1} = \tilde{z}_k - \lambda_{k+1}A\hat{z}_k$ and $(y, \tilde{v}_{k+1}) = 0$, it is also possible to compute the coefficient λ_{k+1} directly from this relation and we have

$$\lambda_{k+1} = \big(y, \tilde{z}_k\big)/\big(y, A\hat{z}_k\big).$$

It seems that, using this relation, the algorithm is more stable.

From the expression for $\tilde{r}_{k+1}$, it is immediate to obtain $\tilde{x}_{k+1}$.

In this algorithm, the products $A\hat{v}_k$ which are needed can be computed recursively. Indeed, setting $\widehat{w}_k = A\hat{v}_k$, we have

$$\widehat{w}_{k+1} = A^2\hat{z}_k + \beta_{k+1}A\hat{z}_k - \gamma_{k+1}\widehat{w}_k.$$

We also set $\tilde{z} = \tilde{z}_k + \tilde{v}_{k+1}$, $\hat{v} = \hat{v}_{k+1} - \gamma_{k+1}\hat{v}_k$ and $\widehat{w} = \widehat{w}_{k+1} - \gamma_{k+1}\widehat{w}_k$.

The first TF Lanczos/Orthodir algorithm coupled with the CGS algorithm is the following:

```
Algorithm TFdirCGS1($A, b, x_0, y$)
      Initializations
            $r_0 \leftarrow b - A\,x_0$
            $z_0 = \tilde{r}_0 = \tilde{z}_0 = \hat{z}_0 = r_0$
            $\tilde{x}_0 = x_0$
            $z_{-1} = \hat{z}_{-1} = \tilde{v}_0 = \hat{v}_0 = \widehat{w}_0 = 0$
            $\lambda_0 = -1$
      for $k = 0, 1, 2, \ldots$ until convergence do
            $\lambda_{k+1} \leftarrow -(y, \tilde{r}_k)/[\lambda_k(y, A\hat{z}_k)]$ or $\lambda_{k+1} \leftarrow (y, \tilde{z}_k)/(y, A\hat{z}_k)$
TF          $r_{k+1} \leftarrow r_k - \lambda_{k+1} A z_k$
TF          $x_{k+1} \leftarrow x_k + \lambda_{k+1} z_k$
            $\tilde{v}_{k+1} \leftarrow \tilde{z}_k - \lambda_{k+1} A\hat{z}_k$
            $\tilde{z} \leftarrow \tilde{z}_k + \tilde{v}_{k+1}$
            $\tilde{r}_{k+1} \leftarrow \tilde{r}_k - \lambda_{k+1} A\tilde{z}$
            $\tilde{x}_{k+1} \leftarrow \tilde{x}_k + \lambda_{k+1}\tilde{z}$
            if $\|r_{k+1}\| \leqslant \varepsilon$ or $\|\tilde{r}_{k+1}\| \leqslant \varepsilon$ then stop
            $\beta_{k+1} \leftarrow -(y, A^2\hat{z}_k)/(y, A\hat{z}_k)$
            if $k > 0$ then
                  $\gamma_{k+1} \leftarrow (y, A\hat{z}_k)/(y, A\hat{z}_{k-1})$
            else
                  $\gamma_1 \leftarrow 0$
            end if
            $\hat{v}_{k+1} \leftarrow A\hat{z}_k + \beta_{k+1}\hat{z}_k - \gamma_{k+1}\hat{v}_k$
            $\hat{v} \leftarrow \hat{v}_{k+1} - \gamma_{k+1}\hat{v}_k$
            $\widehat{w}_{k+1} \leftarrow A^2\hat{z}_k + \beta_{k+1}A\hat{z}_k - \gamma_{k+1}\widehat{w}_k$
            $\widehat{w} \leftarrow \widehat{w}_{k+1} - \gamma_{k+1}\widehat{w}_k$
            $\hat{z}_{k+1} \leftarrow \widehat{w} + \beta_{k+1}\hat{v} + \gamma_{k+1}^2\hat{z}_{k-1}$
            $\tilde{z}_{k+1} \leftarrow \frac{1}{2}(A + \beta_{k+1}I)\,\tilde{z} - \gamma_{k+1}\tilde{v}_k - \frac{1}{2}\lambda_{k+1}\,\widehat{w}$
TF          $z_{k+1} \leftarrow (A + \beta_{k+1}I)\,z_k - \gamma_{k+1}z_{k-1}$
      end for
```

This algorithm will be denoted by 1.a with the first expression for λ_{k+1}, and by 1.b with the second one.

Now if we choose $U_{k+1} \equiv V_{k+1} \equiv P_{k+1}^{(1)}$, we have $a_{k+1} = 1$, $b_{k+1} = \beta_{k+1}$ and $d_{k+1} = \gamma_{k+1}$, the hat vectors coincide with the tilde vectors, $\tilde{v}_k = \tilde{p}_k$ and the two expressions for $\tilde{z}_k$ coincide. Replacing q'_k or $\tilde{u}_{k+1}$ in the two expressions for $\tilde{r}_{k+1}$, and $\tilde{v}'_k$ in $\tilde{z}_{k+1}$, the algorithms become

$$\left.\begin{aligned} \tilde{r}_{k+1} &= A\tilde{s}_{k+1} + \beta_{k+1}\tilde{s}_{k+1} - \gamma_{k+1}\bigl(\tilde{s}_k - \lambda_{k+1}A\tilde{v}_k\bigr),\\ &\quad \text{or equivalently,}\\ \tilde{r}_{k+1} &= A\tilde{r}_k + \beta_{k+1}\tilde{r}_k - \gamma_{k+1}\tilde{s}_k - \lambda_{k+1}A\tilde{v}_{k+1}, \end{aligned}\right\} \qquad (26)$$

$$\begin{aligned} \tilde{s}_{k+1} &= \tilde{r}_k - \lambda_{k+1}A\tilde{z}_k,\\ \tilde{v}_{k+1} &= A\tilde{z}_k + \beta_{k+1}\tilde{z}_k - \gamma_{k+1}\tilde{v}_k,\\ \tilde{z}_{k+1} &= A\tilde{v}_{k+1} + \beta_{k+1}\tilde{v}_{k+1} - \gamma_{k+1}\bigl(A\tilde{v}_k + \beta_{k+1}\tilde{v}_k - \gamma_{k+1}\tilde{z}_{k-1}\bigr). \end{aligned}$$

As above, from (24), the expressions for γ_{k+1} and β_{k+1} simplify.

The vectors $\tilde{r}_k$ obtained by this algorithm are not the residuals of a product method since $P_k^{(1)}(0) \neq 1$. For obtaining the residual of the CGS we must compute the product P_k^2 (which corresponds to $W_k \equiv P_k$ in (11)). We have

$$P_{k+1}^2 = P_k^2 - 2\lambda_{k+1}\xi P_k P_k^{(1)} + \lambda_{k+1}^2\xi^2 P_k^{(1)^2},$$

where the products $P_k P_k^{(1)}$ and $P_k^{(1)^2}$ correspond to the vectors $\tilde{r}_k$ and $\tilde{z}_k$.

Setting

$$\bar{r}_k = P_k^2(A)r_0,$$

we have the additional relations

$$\begin{aligned} \bar{r}_{k+1} &= \bar{r}_k - 2\lambda_{k+1}A\tilde{r}_k + \lambda_{k+1}^2A^2\tilde{z}_k\\ &= \bar{r}_k - \lambda_{k+1}A\tilde{r}_k - \lambda_{k+1}A\bigl(\tilde{r}_k - \lambda_{k+1}A\tilde{z}_k\bigr)\\ &= \bar{r}_k - \lambda_{k+1}A\tilde{r}_k - \lambda_{k+1}A\tilde{s}_{k+1}. \end{aligned} \qquad (27)$$

If the first relation for $\tilde{r}_{k+1}$, given in (26), is used, we need to compute recursively the products $A\tilde{s}_k$ and $A\tilde{v}_k$. Indeed, setting $\tilde{s}'_k = A\tilde{s}_k$, we have

$$\tilde{s}'_{k+1} = A\tilde{r}_k - \lambda_{k+1}A^2\tilde{z}_k$$

and setting $\widetilde{w}_k = A\tilde{v}_k$, we have

$$\widetilde{w}_{k+1} = A^2\tilde{z}_k + \beta_{k+1}A\tilde{z}_k - \gamma_{k+1}\widetilde{w}_k.$$

From (27) we have

$$\bar{x}_{k+1} = \bar{x}_k + \lambda_{k+1}\tilde{r}_k + \lambda_{k+1}\tilde{s}_{k+1}.$$

Thus we obtain another transpose-free Lanczos/Orthodir, coupled with the CGS.

Algorithm TFdirCGS2(A, b, x_0, y)

```
  Initializations
      $r_0 \leftarrow b - A\,x_0$
      $z_0 = \bar{r}_0 = \tilde{r}_0 = \tilde{z}_0 = r_0$
```

```
        $\bar{x}_0 = x_0$
        $z_{-1} = \tilde{z}_{-1} = \tilde{v}_0 = \tilde{s}_0 = \widetilde{w}_0 = 0$
    **for** $k = 0, 1, 2, \ldots$ **until** convergence **do**
        $\lambda_{k+1} \leftarrow (y, \tilde{r}_k)/(y, A\tilde{z}_k)$
TF      $r_{k+1} \leftarrow r_k - \lambda_{k+1} A z_k$
TF      $x_{k+1} \leftarrow x_k + \lambda_{k+1} z_k$
        $\tilde{s}_{k+1} \leftarrow \tilde{r}_k - \lambda_{k+1} A\tilde{z}_k$
        $\tilde{s}'_{k+1} \leftarrow A\tilde{r}_k - \lambda_{k+1} A^2 \tilde{z}_k$
        $\bar{r}_{k+1} \leftarrow \bar{r}_k - \lambda_{k+1} A\tilde{r}_k - \lambda_{k+1} \tilde{s}'_{k+1}$
        $\bar{x}_{k+1} \leftarrow \bar{x}_k + \lambda_{k+1} \tilde{r}_k + \lambda_{k+1} \tilde{s}_{k+1}$
        **if** $\|r_{k+1}\| \leqslant \varepsilon$ **or** $\|\bar{r}_{k+1}\| \leqslant \varepsilon$ **then** stop
        $\beta_{k+1} \leftarrow -(y, A^2 \tilde{z}_k)/(y, A\tilde{z}_k)$
        **if** $k > 0$ **then**
            $\gamma_{k+1} \leftarrow (y, A\tilde{z}_k)/(y, A\tilde{z}_{k-1})$
        **else**
            $\gamma_1 \leftarrow 0$
        **end if**
        $\tilde{r}_{k+1} \leftarrow \tilde{s}'_{k+1} + \beta_{k+1} \tilde{s}_{k+1} - \gamma_{k+1}(\tilde{s}_k - \lambda_{k+1} \widetilde{w}_k)$
        $\tilde{v}_{k+1} \leftarrow A\tilde{z}_k + \beta_{k+1} \tilde{z}_k - \gamma_{k+1} \tilde{v}_k$
        $\widetilde{w}_{k+1} \leftarrow A^2 \tilde{z}_k + \beta_{k+1} A\tilde{z}_k - \gamma_{k+1} \widetilde{w}_k$
        $\tilde{z}_{k+1} \leftarrow \widetilde{w}_{k+1} + \beta_{k+1} \tilde{v}_{k+1} - \gamma_{k+1}(\widetilde{w}_k + \beta_{k+1} \tilde{v}_k - \gamma_{k+1} \tilde{z}_{k-1})$
TF      $z_{k+1} \leftarrow (A + \beta_{k+1} I)\, z_k - \gamma_{k+1} z_{k-1}$
    **end for**
```

Now, we consider the second relation for $\tilde{r}_{k+1}$, given in (26). We only need to compute recursively $\widetilde{w}_k = A\tilde{v}_k$. From (27), we directly have $\bar{x}_{k+1}$.

Thus we obtain a third transpose-free Lanczos/Orthodir, coupled with the GCS.

Algorithm TFdirCGS3(A, b, x_0, y)

```
    **Initializations**
        $r_0 \leftarrow b - A x_0$
        $z_0 = \bar{r}_0 = \tilde{r}_0 = \tilde{z}_0 = r_0$
        $\bar{x}_0 = x_0$
        $z_{-1} = \tilde{z}_{-1} = \tilde{v}_0 = \tilde{s}_0 = \widetilde{w}_0 = 0$
    **for** $k = 0, 1, 2, \ldots$ **until** convergence **do**
        $\lambda_{k+1} \leftarrow (y, \tilde{r}_k)/(y, A\tilde{z}_k)$
TF      $r_{k+1} \leftarrow r_k - \lambda_{k+1} A z_k$
TF      $x_{k+1} \leftarrow x_k + \lambda_{k+1} z_k$
        $\bar{r}_{k+1} \leftarrow \bar{r}_k - 2\lambda_{k+1} A\tilde{r}_k + \lambda_{k+1}^2 A^2 \tilde{z}_k$
        $\bar{x}_{k+1} \leftarrow \bar{x}_k + 2\lambda_{k+1} \tilde{r}_k - \lambda_{k+1}^2 A\tilde{z}_k$
        **if** $\|r_{k+1}\| \leqslant \varepsilon$ **or** $\|\bar{r}_{k+1}\| \leqslant \varepsilon$ **then** stop
        $\beta_{k+1} \leftarrow -(y, A^2 \tilde{z}_k)/(y, A\tilde{z}_k)$
        if $k > 0$ then
            $\gamma_{k+1} \leftarrow (y, A\tilde{z}_k)/(y, A\tilde{z}_{k-1})$
        else
            $\gamma_1 \leftarrow 0$
        end if
        $\widetilde{w}_{k+1} \leftarrow A^2\tilde{z}_k + \beta_{k+1}A\tilde{z}_k - \gamma_{k+1}\widetilde{w}_k$
        $\tilde{r}_{k+1} \leftarrow A\tilde{r}_k + \beta_{k+1}\tilde{r}_k - \gamma_{k+1}\tilde{s}_k - \lambda_{k+1}\,\widetilde{w}_{k+1}$
        $\tilde{v}_{k+1} \leftarrow A\tilde{z}_k + \beta_{k+1}\tilde{z}_k - \gamma_{k+1}\tilde{v}_k$
        $\tilde{z}_{k+1} \leftarrow \widetilde{w}_{k+1} + \beta_{k+1}\tilde{v}_{k+1} - \gamma_{k+1}(\widetilde{w}_k + \beta_{k+1}\tilde{v}_k - \gamma_{k+1}\tilde{z}_{k-1})$
        $\tilde{s}_{k+1} \leftarrow \tilde{r}_k - \lambda_{k+1}A\tilde{z}_k$
TF      $z_{k+1} \leftarrow (A + \beta_{k+1}I)\,z_k - \gamma_{k+1}z_{k-1}$
    end for
```

All the algorithms of this section need 4 matrix–vector products and 3 inner products.

6. Numerical results

We will now present the results obtained, using the Matlab versions of the algorithms, with various systems, all of dimension 200. The solution was always chosen randomly and the right hand side b was computed accordingly. All the algorithms were started from $x_0 = 0$, with $y_0 = r_0$. The curves represent, in a logarithmic scale, the norm of the residuals.

6.1. Example 1

Let us consider the matrix

$$A = \begin{pmatrix} 2 & 1 & & & & \\ 0 & 2 & 1 & & & \\ 1 & 0 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots & 1 \\ & & & 1 & 0 & 2 \end{pmatrix}.$$

Its condition number is 2.91.

The conclusions of the numerical results are the following:

1. For Lanczos' method, the results with U_k and $V_k = \xi^k$ are not as good as with the other choices (see figure 1).
2. In the algorithm TFdirCGS1, the computation of λ_{k+1} by the second formula (TFdirCGS1.b) leads to better results either for the TF Lanczos method (see fig-

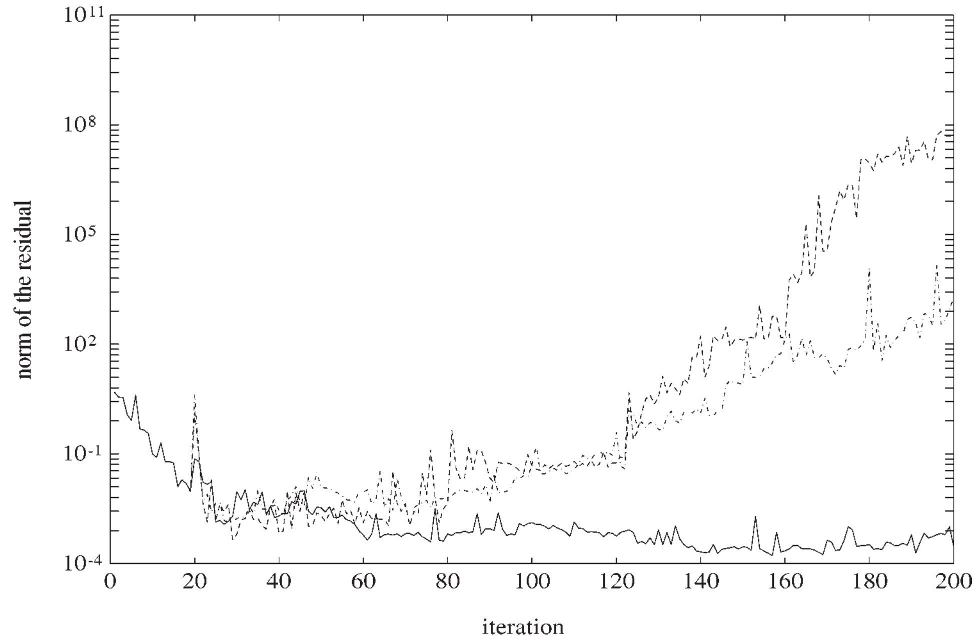

Figure 1. Example 1: TF Lanczos/Orthores TFres1 (solid line), TF Lanczos/Orthomin (dashed line), TF Lanczos/Orthodir TFdir2 (dot-dash line), all with powers.

ure 2) or for the CGS itself. The results of TFdirCGS1.b are quite similar to the results obtained by the other variants (TFdirCGS2 and TFdirCGS3).

3. The best algorithms for implementing the TF Lanczos method with the choice of U_k and V_k corresponding to the CGS seem to be Orthores and Orthomin (see figure 3 where the curves coincide). Orthodir seems to be always less precise.

4. Comparing the algorithms using A^{T} with the various implementations of the method of Lanczos via CGS shows that the results for Orthores (see figure 4) and also for Orthomin coincide while, for Orthodir, the transpose-free algorithm is not as good as the algorithm using the transpose (see figure 5).

6.2. Example 2

Let us now consider the matrix *redheff* of the Matlab Test Matrix Toolbox [22]. Its condition number is 458.8. For this example, the conclusions are the following:

1. The three different implementations of the BiCGSTAB give quite similar results but the recurrence relationships of Orthomin are slightly better at the end of the iterations (see figure 6).

2. The transpose-free Lanczos method coupled with CGS based on Orthores and Orthomin give almost the same results but Orthodir (TFdirCGS1.b) is unstable

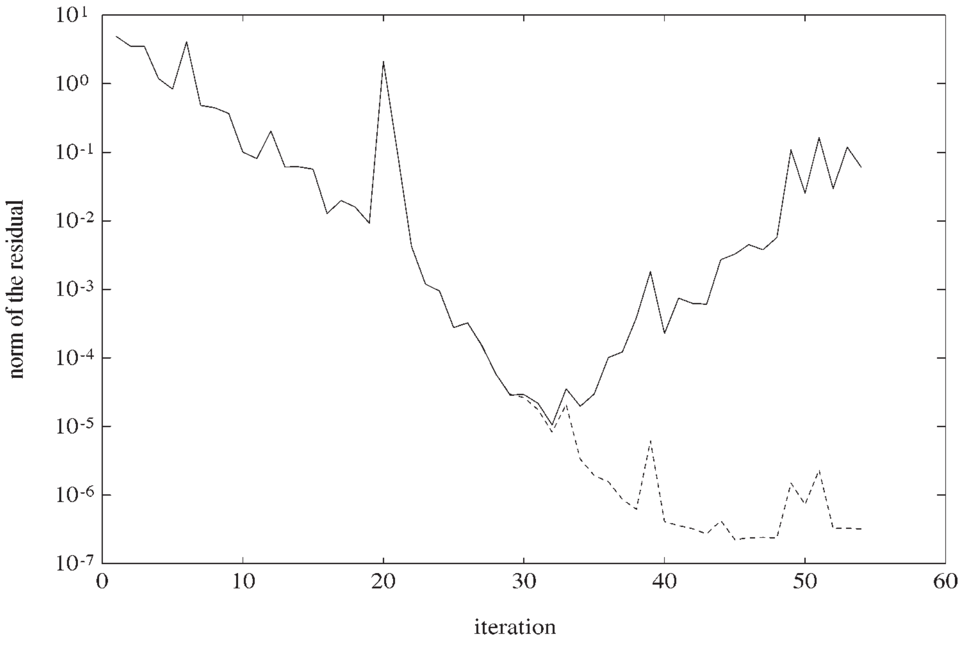

Figure 2. Example 1: TF Lanczos/Orthodir coupled with the CGS, algorithms TFdirCGS1.a (solid line) and TFdirCGS1.b (dashed line).

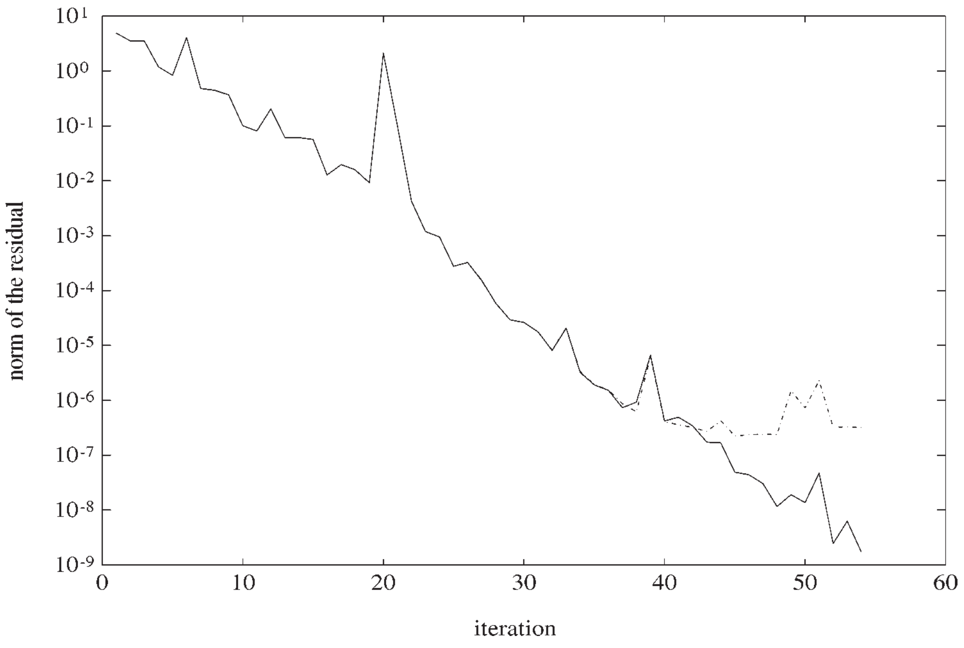

Figure 3. Example 1: TF Lanczos/Orthores (solid line), TF Lanczos/Orthomin (solid line), TF Lanczos/Orthodir TFdirCGS1.b (dot-dash line), all coupled with the CGS.

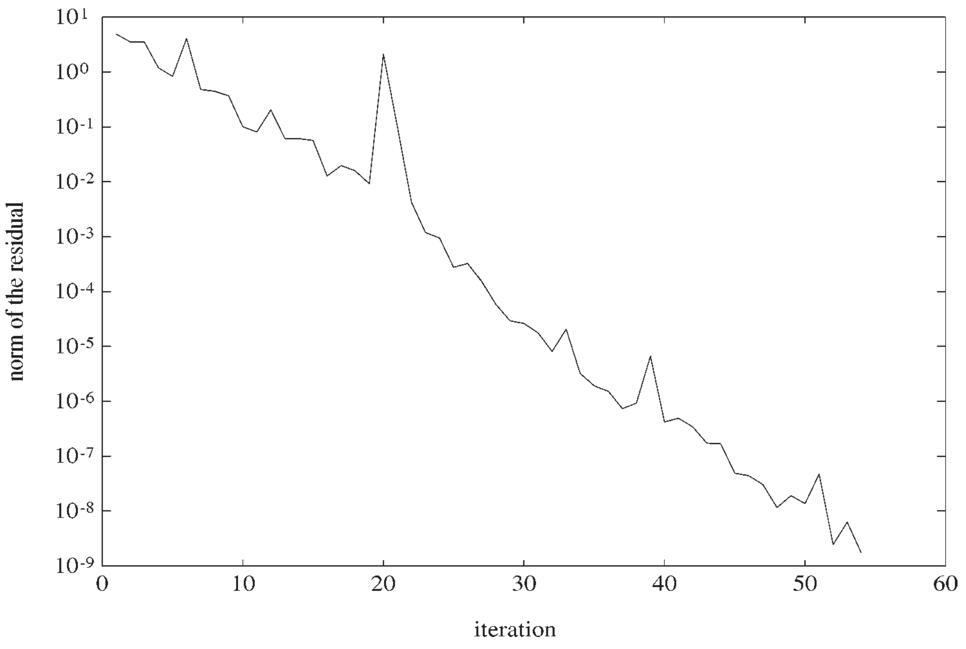

Figure 4. Example 1: Lanczos/Orthores and TF Lanczos/Orthores coupled with the CGS.

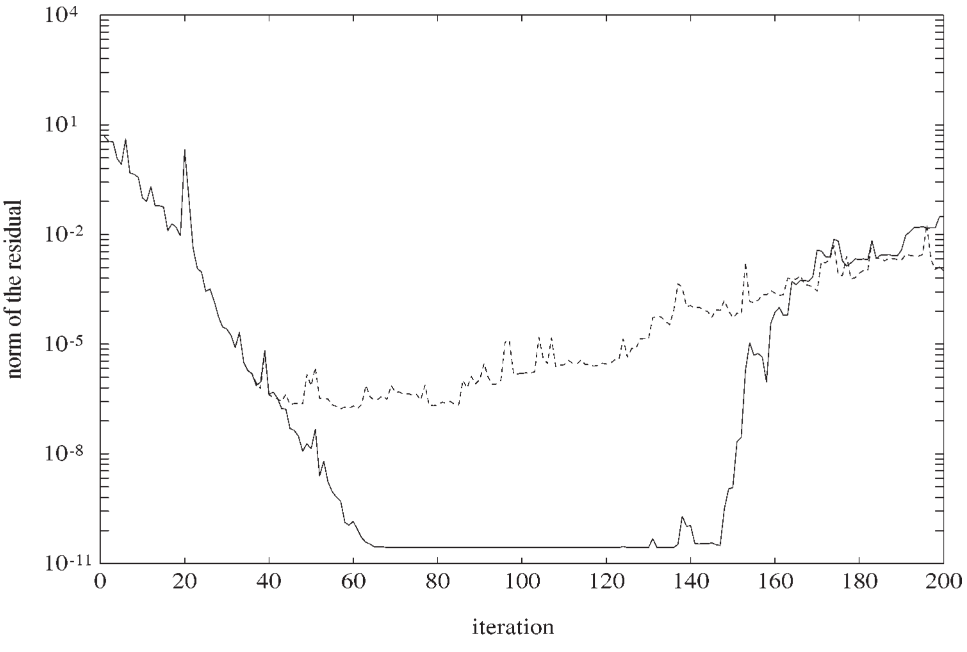

Figure 5. Example 1: Lanczos/Orthodir (solid line) and TF Lanczos/Orthodir TFdirCGS1.b coupled with the CGS (dashed line).

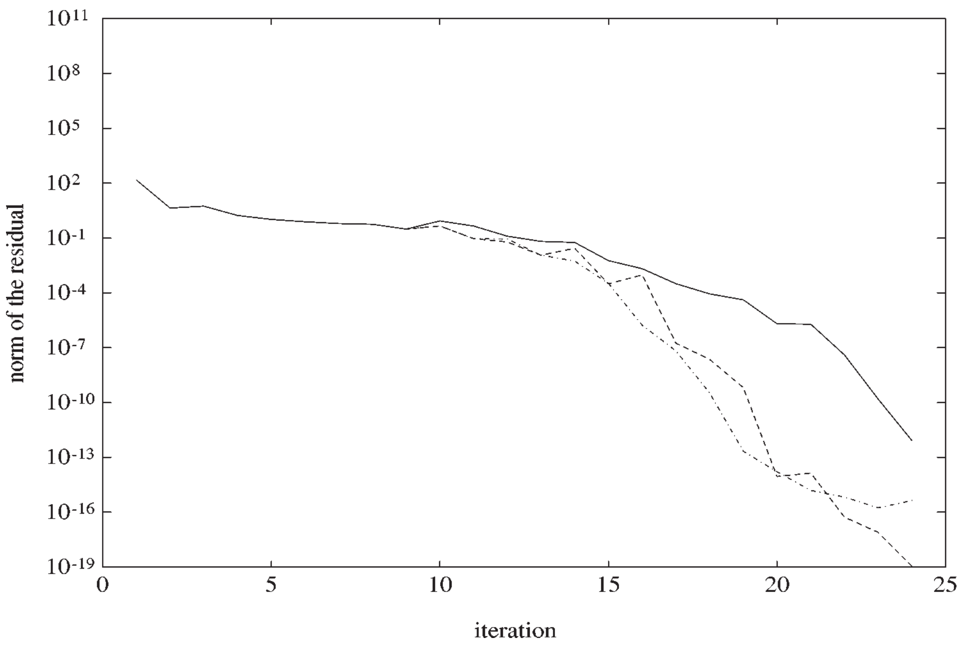

Figure 6. Example 2: BiCGSTAB obtained with TF Lanczos/Orthores (solid line), TF Lanczos/Orthomin (dashed line) and TF Lanczos/Orthodir (dot-dash line).

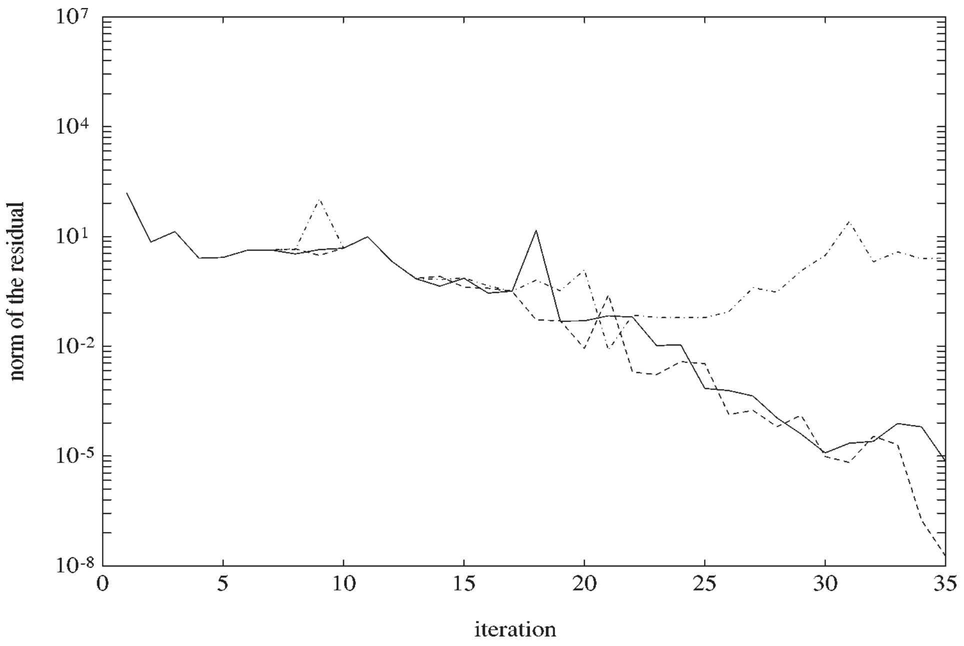

Figure 7. Example 2: TF Lanczos/Orthores (solid line), TF Lanczos/Orthomin (dashed line), TF Lanczos/Orthodir TFdirCGS1.b (dot-dash line), all coupled with the CGS.

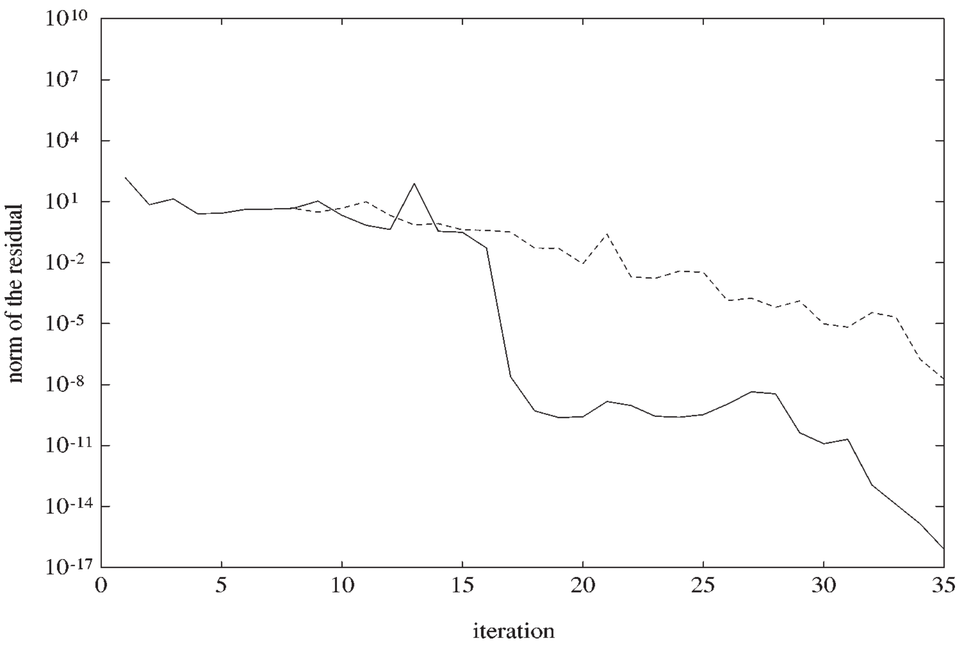

Figure 8. Example 2: Lanczos/Orthomin (solid line) and TF Lanczos/Orthomin coupled with the CGS (dashed line).

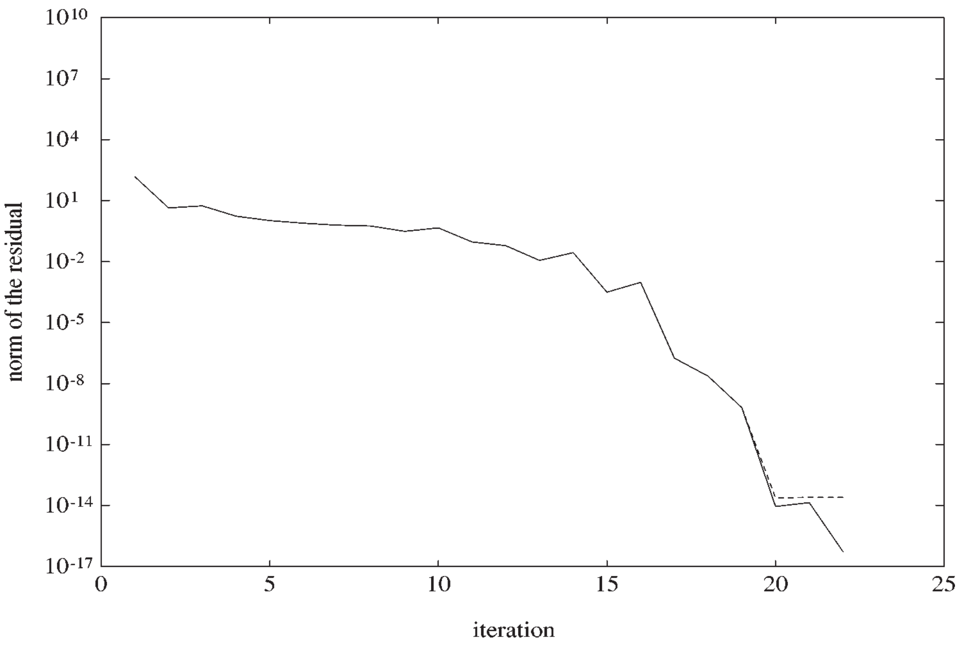

Figure 9. Example 2: Iterative (solid line) and actual (dashed line) residuals for the BiCGSTAB with TFminBiCGSTAB.

(see figure 7). The same conclusions hold for the implementations of the CGS itself.

3. The implementations of Lanczos' method making use of A^{T} give quite different results (see figure 8, for Orthomin).
4. The iterative residuals of the BiCGSTAB (that is, those computed recursively) obtained by our algorithm TFresBiCGSTAB and those obtained by the BIOStab of [25] are almost the same and they reach $\simeq 10^{-14}$ at iteration 26. However, the actual residuals (that is, those computed by the formula $r_k = b - Ax_k$) stagnate around $\simeq 10^{-2}$ after iteration 15 for both methods. On the other hand, using our algorithm TFminBiCGSTAB leads to iterative and actual residuals for the BiCGSTAB which are quite the same and they achieve the value 10^{-14} at iteration 20 (see figure 9).

7. Conclusions

In this paper, we show how to implement the method of Lanczos for solving nonsymmetric linear systems without using the tranpose of the matrix. Our approach is based on the recursive computation of the products of polynomials appearing in the recurrence relationships of the formal orthogonal polynomials underlying the method of Lanczos. In fact, there are many such transpose-free algorithms. In some cases, these algorithms allow a coupled implementation of a Lanczos-type product method at almost no extra cost which consists, in fact, in adding a few instructions (with one additional matrix–vector product) into the code of the product method. In this paper, we gave some of these algorithms and perform a few numerical experiments. It seems that the more stable algorithms are those based on the recurrence relationships of Lanczos/Orthomin, among which are those proposed in [15]. The less stable algorithms are those programmed via Lanczos/Orthodir and, moreover, they need an extra matrix–vector product. The algorithms making use of the power basis are the less interesting ones for, at least, two reasons: it is well known that such a basis is numerically unstable and, moreover, this basis does not correspond to a product method. Thus, the algorithms need an extra matrix–vector product at no gain and it is better to use another basis.

Transpose-free versions of the other algorithms given in [1] for implementing the method of Lanczos can be obtained similarly. They remain to be studied.

In fact, as explained above, the use of the transpose in the implementation of Lanczos' method only results from the computation of the coefficients in the recurrence relationships used in each algorithm. Since these coefficients can be computed without using the transpose in a product method, the idea developed in this paper was to compute the coefficients by a product method and to use them simultaneously for implementing Lanczos' method. Obviously, the same idea can be used for the implementation of the look-ahead Lanczos-type algorithms where breakdowns and near-breakdowns are avoided. In [12,6,7], look-ahead algorithms for avoiding breakdowns or near-breakdowns in the CGS and the BiCGSTAB were given. Since the coefficients in the recurrences used for treating breakdowns and near-breakdowns in Lanczos' method are the same (see [9,10]) as in these algorithms, they lead to transpose-free look-ahead algorithms for implementing simultaneously Lanczos' method and either the CGS or the BiCGSTAB. Such algorithms will also consist in adding some instructions in the codes of the corresponding product method. This will be the subject of a forthcoming paper. Transpose-free versions of the new look-ahead algorithms described in [11] are also under consideration. These algorithms are more stable and they require much less storage.

More details on the derivation of the algorithms presented in this paper can be found in a report, with the same title, at `http://ano.univ-lille1.fr`.

References

[1] C. Baheux, New implementations of Lanczos method, J. Comput. Appl. Math. 57 (1995) 3–15.

[2] C. Brezinski, *Padé-type Approximation and General Orthogonal Polynomials*, International Series of Numerical Mathematics 50 (Birkhäuser, Basel, 1980).

[3] C. Brezinski, CGM: a whole class of Lanczos-type solvers for linear systems, Note ANO-253, Laboratoire d'Analyse Numérique et d'Optimisation, Université des Sciences et Technologies de Lille (November 1991).

[4] C. Brezinski, A transpose-free Lanczos/Orthodir algorithm for linear systems, C. R. Acad. Sci. Paris Sér. I 324 (1997) 349–354.

[5] C. Brezinski and M. Redivo-Zaglia, Hybrid procedures for solving systems of linear systems, Numer. Math. 67 (1994) 1–19.

[6] C. Brezinski and M. Redivo-Zaglia, Treatment of near-breakdown in the CGS algorithm, Numer. Algorithms 7 (1994) 33–73.

[7] C. Brezinski and M. Redivo-Zaglia, Look-ahead in Bi-CGSTAB and other product methods for linear systems, BIT 35 (1995) 169–201.

[8] C. Brezinski and M. Redivo-Zaglia, Transpose-free implementations of Lanczos's method for nonsymmetric linear systems, Note ANO-372, Laboratoire d'Analyse Numérique et d'Optimisation, Université des Sciences et Technologies de Lille (1997).

[9] C. Brezinski, M. Redivo-Zaglia and H. Sadok, Avoiding breakdown and near-breakdown in Lanczos type algorithms, Numer. Algorithms 1 (1991) 261–284.

[10] C. Brezinski, M. Redivo-Zaglia and H. Sadok, A breakdown-free Lanczos type algorithm for solving linear systems, Numer. Math. 63 (1992) 29–38.

[11] C. Brezinski, M. Redivo-Zaglia and H. Sadok, New look-ahead Lanczos-type algorithms for linear systems, submitted.

[12] C. Brezinski and H. Sadok, Avoiding breakdown in the CGS algorithms, Numer. Algorithms 1 (1991) 207–221.

[13] C. Brezinski and H. Sadok, Some vector sequence transformations with applications to systems of equations, Numer. Algorithms 3 (1992) 75–80.

[14] C. Brezinski and H. Sadok, Lanczos-type algorithms for solving systems of linear equations, Appl. Numer. Math. 11 (1993) 443–473.

[15] T.F. Chan, L. de Pillis and H. van der Vorst, Transpose-free formulations of Lanczos-type methods for nonsymmetric linear systems, Numer. Algorithms 17 (1998), this issue.

[16] R. Fletcher, Conjugate gradient methods for indefinite systems, in: *Numerical Analysis, Dundee 1975*, ed. G.A. Watson, Lecture Notes in Mathematics 506 (Springer, Berlin, 1976) pp. 73–89.

[17] D.R. Fokkema, Subspace methods for linear, nonlinear, and eigen problems, thesis, University of Utrecht (1996).

[18] D.R. Fokkema, G.L.G. Sleijpen and H.A. van der Vorst, Generalized conjugate gradient squared, J. Comput. Appl. Math. 71 (1996) 125–146.

[19] W. Gander, G.H. Golub and D. Gruntz, Solving linear equations by extrapolation, in: *Supercomputing*, ed. J.S. Kovalik (Springer, Berlin, 1989) pp. 279–293.

[20] M.H. Gutknecht, The unsymmetric Lanczos algorithms and their relations to Padé approximation, continued fractions, and the qd-algorithm, in: *Proc. of the Copper Mountain Conf. on Iterative Methods*, Vol. 2, Copper Mountain, CO (April 1–5, 1990) unpublished.

[21] M.H. Gutknecht, Variants of BiCGSTAB for matrices with complex spectrum, SIAM J. Sci. Comput. 14 (1993) 1020–1033.

[22] N.J. Higham, The test matrix toolbox for MATLAB (version 3.0), Numer. Analysis Report No. 276, Department of Mathematics, The University of Manchester (September 1995).

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

[24] C. Lanczos, Solution of systems of linear equations by minimized iterations, J. Res. Natl. Bur. Stand. 49 (1952) 33–53.

[25] K.J. Ressel and M.H. Gutknecht, QMR-smoothing for Lanczos-type product methods based on three-term recurrences, to appear.

[26] H. Rutishauser, Der Quotienten–Differenzen-Algorithmus, Z. Angew. Math. Phys. 5 (1954) 233–251.

[27] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 10 (1989) 35–52.

[28] H.A. van der Vorst, Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Statist. Comput. 13 (1992) 631–644.

[29] P.K.W. Vinsome, Orthomin, an iterative method for solving sparse sets of simultaneous linear equations, in: *Proc. 4th Symp. on Reservoir Simulation*, Society of Petroleum Engineers of AIME (1976) pp. 149–159.

[30] D.M. Young and K.C. Jea, Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods, Linear Algebra Appl. 34 (1980) 159–194.