Treatment of near-breakdown in the CGS algorithm

C. Brezinski

Laboratoire d'Analyse Numérique et d'Optimisation, UFR IEEA - M3, Université des Sciences et Technologies de Lille, F-59655 Villeneuve d'Ascq Cedex, France
Email: brezinsk@omega.univ-lille1.fr

M. Redivo-Zaglia

Dipartimento di Elettronica e Informatica, Università degli Studi di Padova, via Gradenigo 6/a, I-35131 Padova, Italy
Email: elen@elett1.dei.unipd.it

Received 23 September 1993; revised 18 November 1993
Communicated by T.F. Chan

Lanczos' method for solving the system of linear equations $Ax = b$ consists in constructing a sequence of vectors (x_k) such that $r_k = b - Ax_k = P_k(A)r_0$ where $r_0 = b - Ax_0$. P_k is an orthogonal polynomial which is computed recursively. The conjugate gradient squared algorithm (CGS) consists in taking $r_k = P_k^2(A)r_0$. In the recurrence relation for P_k, the coefficients are given as ratios of scalar products. When a scalar product in a denominator is zero, then a breakdown occurs in the algorithm. When such a scalar product is close to zero, then rounding errors can seriously affect the algorithm, a situation known as near-breakdown. In this paper it is shown how to avoid near-breakdown in the CGS algorithm in order to obtain a more stable method.

Keywords: Conjugate gradient, Lanczos' method, systems of linear equations, orthogonal polynomials.

AMS(MOS) subject classification: 65F05, 65F10, 65F25.

1. Introduction

Let us consider in $\mathbb{C}^n$ the system of linear equations

$$Ax = b,$$

where A is a nonsingular matrix.

Lanczos' method [31,32] for solving this system consists in constructing a sequence of vectors (x_k) as follows

- choose two arbitrary nonzero vectors x_0 and y;
- set $r_0 = b - Ax_0$;

- determine x_k such that

$$x_k - x_0 \in E_k = \operatorname{span}(r_0, Ar_0, \ldots, A^{k-1}r_0)$$

$$r_k = b - Ax_k \perp F_k = \operatorname{span}(y, A^*y, \ldots, A^{*k-1}y),$$

where A^* is the conjugate transpose of A.

These two conditions determine x_k uniquely under the assumption that the determinant $H_k^{(1)}$ defined below is different from zero.

Indeed, $x_k - x_0$ can be written as

$$x_k - x_0 = -b_1^{(k)}r_0 - \cdots - b_k^{(k)}A^{k-1}r_0$$

and the orthogonality conditions give

$$(A^{*i}y, r_k) = 0 \quad \text{for } i = 0, \ldots, k-1,$$

which is a system of k linear equations in the k unknowns $b_1^{(k)}, \ldots, b_k^{(k)}$. This system is nonsingular only if $r_0, Ar_0, \ldots, A^{k-1}r_0$ are linearly independent and $y, A^*y, \ldots, A^{*k-1}y$ also.

If we set

$$P_k(\xi) = 1 + b_1^{(k)}\xi + \cdots + b_k^{(k)}\xi^k$$

then we have

$$r_k = P_k(A)r_0.$$

Moreover, if we define the linear functional c on the space of polynomials by

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

then the preceding orthogonality conditions can be written as

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

These relationships show that P_k is the polynomial of degree at most k belonging to the family of formal orthogonal polynomials with respect to c [3]. This polynomial is defined apart from a multiplying factor which was chosen, in our case, such that $P_k(0) = 1$. Due to this normalization, P_k exists and is unique if and only if the Hankel determinant

$$H_k^{(1)} = \begin{vmatrix} (y, Ar_0) & (y, A^2r_0) & \cdots & (y, A^k r_0) \\ (y, A^2r_0) & (y, A^3r_0) & \cdots & (y, A^{k+1}r_0) \\ \vdots & \vdots & & \vdots \\ (y, A^k r_0) & (y, A^{k+1}r_0) & \cdots & (y, A^{2k-1}r_0) \end{vmatrix}$$

is different from zero.

The polynomials P_k can be recursively computed in different ways leading to the various Lanczos type algorithms known as *Lanczos/Orthores*, *Biores* [28], *Lanc-*

zos/Orthodir, *Biodir* [28,42], *Lanczos/Orthomin* [40,42], *Biomin* [26], *biconjugate gradient* [23],..., and so on. A unified presentation and derivation of all these methods can be based on the theory of formal orthogonal polynomials [14].

An interesting property of Lanczos' method is its finite termination, namely that $\exists k \leqslant n$ such that $r_k = 0$ and $x_k = x = A^{-1}b$.

A variant of Lanczos' method was proposed by Sonneveld [36]. It is the so-called *conjugate gradient squared method* (CGS in short) which consists in taking

$$r_k = P_k^2(A)r_0.$$

This method obviously has the same finite termination property as Lanczos' method.

The coefficients appearing in the various recurrence relations for computing the orthogonal polynomials P_k are given as ratios of scalar products.

When a scalar product in a denominator is zero, then a so-called *breakdown* occurs in the corresponding algorithm which has to be stopped. It can be due to the non-existence of the orthogonal polynomial under consideration (a notion called *true breakdown* [8] or *pivot breakdown* [17]) or to the recurrence relationship used (a notion called *ghost breakdown* [8] or *Lanczos breakdown* [17]). When a breakdown is due to the non-existence of some of the polynomials P_k (that is, in the case of a true breakdown), it can be avoided by jumping over these non-existing polynomials and computing only the existing ones. The corresponding algorithm was presented in [12] with its derivation and theoretical background. It was called the *Method of Recursive Zoom*, in short the MRZ. Our treatment cures true breakdowns (ghost breakdowns cannot occur in the recurrence relationships used) and the MRZ can only suffer from an incurable hard breakdown. The corresponding FORTRAN subroutine can be found in [10,11]. All the other recurrence relationships for the implementation of the Lanczos method can suffer from ghost breakdowns. The treatment of these ghost breakdowns consists in jumping over polynomials which exist but cannot be computed by the recurrence relationship used. In [10,11] we gave two algorithms, called BMRZ and SMRZ, based on different recurrence relationships. They can suffer from true and ghost breakdowns but, in our programs, we only cure the true ones.

When a scalar product appearing as the denominator of a coefficient in a recurrence relationship is close to zero, then roundoff errors can seriously affect the algorithm, a situation known under the name of *near-breakdown*. There are two kinds of near-breakdowns, true and ghost, depending on whether they are related to a true or a ghost breakdown. It was shown in [10,11] how to cure true near-breakdowns. The algorithms, which are modifications of the MRZ and the SMRZ, were called the GMRZ and the BSMRZ respectively. The subroutine corresponding to the BSMRZ can be found in [10,11]. The BMRZ cannot be generalized. See [13] for a review of these questions.

In [15], we showed how to avoid breakdown in the CGS algorithm. In this paper, we shall see how to use the relations given in [10] in order to avoid true and ghost near-breakdowns in the CGS algorithm. Another look-ahead strategy for the CGS

is given in [17]. It is based on the recurrence relationships corresponding to the Lanczos/Orthomin algorithm and it also incorporates a residual smoothing strategy [35,41,43] for curing further numerical instabilities. The hybrid procedures proposed in [7] could also be used for this purpose. Let us mention that a quite similar procedure can be used for treating the near-breakdown in some algorithms generalizing the Bi-CGSTAB of Van der Vorst [38]; see [9].

Let us now begin by recalling some results on orthogonal polynomials that will be useful for our purpose.

2. Orthogonal polynomials

As explained in the introduction we shall consider only the existing orthogonal polynomials (called regular) and we shall not give a name to the other ones. Thus, we shall now denote by P_k (instead of P_{n_k}) the polynomial of degree at most n_k satisfying the orthogonality conditions

$$c(\xi^i P_k(\xi)) = 0 \quad \text{for } i = 0, \ldots, n_k - 1.$$

Such polynomials are defined apart from a multiplying factor. As mentioned above, they are normalized by the condition $P_k(0) = 1$.

Let now $P_k^{(1)}$ (instead of $P_{n_k}^{(1)}$) be the regular monic polynomial of degree n_k belonging to the family of formal orthogonal polynomials with respect to the functional $c^{(1)}$ defined by $c^{(1)}(\xi^i) = c(\xi^{i+1})$. $P_k^{(1)}$ exists and is unique if and only if the Hankel determinant $H_{n_k}^{(1)}$ is different from zero, which is also the condition insuring the existence and uniqueness of the polynomial P_k. Since it has the degree n_k, the polynomial $P_k^{(1)}$ satisfies the orthogonality conditions

$$c^{(1)}(\xi^i P_k^{(1)}(\xi)) = 0 \quad \text{for } i = 0, \ldots, n_k - 1.$$

In some cases, this quantity can also be zero for higher values of i and if $P_k^{(1)}$ satisfies

$$c^{(1)}(\xi^i P_k^{(1)}(\xi)) = 0 \quad \text{for } i = 0, \ldots, n_k + m_k - 2$$

and

$$c^{(1)}(\xi^{n_k+m_k-1} P_k^{(1)}(\xi)) \neq 0$$

then the next regular orthogonal polynomial, $P_{k+1}^{(1)}$, has degree $n_{k+1} = n_k + m_k$ and it was proved by Draux [22] that, in such a case, the following more general recurrence relationships hold

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

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

with $P_{-1}^{(1)} = 0$, $P_0^{(1)}(\xi) = 1$ and $C_1 = 0$. w_k is a polynomial of degree $m_k - 1$ at most and q_k is a monic polynomial of degree m_k. Their coefficients and C_{k+1} are

determined by imposing the orthogonality conditions of P_{k+1} and $P_{k+1}^{(1)}$. The preceding relations hold only if the quantities $c^{(1)}(\xi^i P_k^{(1)}(\xi))$ are exactly zero for $i = 0, \ldots, n_k + m_k - 2$ (which corresponds to a breakdown). Setting $r_k = P_k(A)r_0$ and $z_k = P_k^{(1)}(A)r_0$ in the previous recurrence relations leads to a breakdown-free algorithm for implementing Lanczos' method since the use of non-existing orthogonal polynomials is avoided. This algorithm, which is a look-ahead strategy, was called the *Method of Recursive Zoom* and it is known under its abbreviation MRZ. It can only suffer from an incurable hard breakdown. It has a strong connection with the methods derived by Gutknecht [25]; see also Joubert [29].

Of course, if $|c^{(1)}(\xi^{n_k+m_k-1}P_k^{(1)})|$ is different from zero but small (and possibly badly computed), the coefficients of the two recurrence relations of the MRZ could be large and badly computed and rounding errors could affect the algorithm. The same is true if the quantities $|c^{(1)}(\xi^i P_k^{(1)})|$ are not zero for $i = n_k, \ldots, n_k + m_k - 2$ but small; in that case no breakdown occurs in the method but numerical instability could be present, a situation called *near-breakdown*.

It is possible to avoid such a near-breakdown by jumping over those polynomials which could be badly computed and to compute directly the first regular polynomial following them. The tests for deciding when and how far to jump will be discussed in detail in the next section.

For the moment, let us assume that, at the step k, the regular orthogonal polynomial $P_k^{(1)}$ of degree n_k with respect to $c^{(1)}$ has been obtained and that the length $m_k \geqslant 1$ of the jump has been decided. Let $n_{k+1} = n_k + m_k$. We shall denote by $P_{k+1}^{(1)}$ the regular orthogonal polynomial of degree n_{k+1} with respect to $c^{(1)}$. As explained in the sequel, if such a polynomial does not exist (and we are able to detect such a case) the value of m_k has to be increased until a regular polynomial is obtained. We shall denote by P_{k+1} the corresponding polynomial of degree n_{k+1} at most orthogonal with respect to c and normalized by the condition $P_{k+1}(0) = 1$.

Let us first compute P_{k+1}. This polynomial (of degree $n_{k+1} = n_k + m_k$) is only assumed to satisfy its usual orthogonality conditions $c(\xi^i P_k) = 0$ for $i = 0, \ldots, n_k + m_k - 1$ (which corresponds to a near-breakdown). As proved in [10], it can be computed by

$$P_{k+1}(\xi) = P_k(\xi) - \xi w_k(\xi) P_k^{(1)}(\xi) - \xi v_k(\xi) P_k(\xi), \tag{1}$$

where w_k is a polynomial of degree $m_k - 1$ at most and v_k a polynomial of degree $m_k - 2$ at most. Let us recall that, in the case of a breakdown, v_k is identically zero and the relation (1) reduces to the previous recurrence relationship (due to Draux [22]) for P_{k+1}. The coefficients of the polynomials w_k and v_k are computed by writing down the orthogonality conditions of P_{k+1} with respect to the basis ξ^i for $i = 0, \ldots, n_k + m_k - 1$. This is the procedure followed in [10] which leads to a system of linear equations giving the coefficients of these polynomials. The corresponding procedure for avoiding near-breakdown in Lanczos' method was called the BSMRZ (*Block Symmetric Method of Recursive Zoom*). Of course, the

polynomials P_k computed in this algorithm are the same as those computed in the look-ahead procedure given in [24]. However, the recurrence relationships used for their computation are not the same.

In our case, this procedure could still be used and the first version of our algorithm was coded in that way [6]. However, one of the main interests of the CGS is to avoid the use of A^*, which is achieved, in terms of orthogonal polynomials, by avoiding the use of $\xi^i P_k$ and $\xi^i P_k^{(1)}$ and replacing them by $\xi^{i-n_k} P_k^{(1)} P_k$ and $\xi^{i-n_k} P_k^{(1)^2}$. As shown in [5], this is possible (but, in our case, quite tricky as it will appear below) and it provides a new and easy way for obtaining recurrence relationships between orthogonal polynomials. However, such a change in the polynomial basis can only be made for $i \geqslant n_k$ and is impossible to use for $i = 0, \ldots, n_k - 1$. In that case, since $\xi^i w_k(\xi)$ is a polynomial of degree n_k at least, we can divide it by $P_k^{(1)}$ by the Euclidean algorithm, and thus, for $i \geqslant n_k - m_k$, it can be written as [5]

$$\xi^i w_k(\xi) = U(\xi) + \phi(\xi) P_k^{(1)}(\xi),$$

where U is an arbitrary polynomial of degree $n_k - 1$ at most and ϕ a polynomial of degree $i + m_k - n_k - 1$ (both depending on i), and a similar relation for $\xi^i v_k(\xi)$. As we shall see below, the polynomial U will disappear when imposing the orthogonality conditions (and thus it will not be necessary to compute it) and the coefficients of the polynomial ϕ could be determined as the solution of a linear system. After having obtained the coefficients of ϕ, the coefficients of w_k will be obtained directly from those of ϕ. Thus, the computation of the coefficients of ϕ is only an intermediate (but crucial) step in the procedure. When $i < n_k - m_k$, the orthogonality conditions are automatically satisfied and they are not needed.

Let us now show in detail how to compute the coefficients of the polynomials v_k and w_k. We first set

$$P_k^{(1)}(\xi) = a_0^{(k)} + a_1^{(k)} \xi + \cdots + a_{n_k-1}^{(k)} \xi^{n_k - 1} + \xi^{n_k}$$

and

$$P_k(\xi) = 1 + b_1^{(k)} \xi + \cdots + b_{n_k}^{(k)} \xi^{n_k}.$$

Multiplying both sides of (1) by ξ^i and applying the functional c, we obtain

$$c(\xi^i P_{k+1}) = c(\xi^i P_k) - c^{(1)}(\xi^i w_k P_k^{(1)}) - c(\xi^{i+1} v_k P_k).$$

This quantity must be zero for $i = 0, \ldots, n_k + m_k - 1$. By the orthogonality properties of P_k and $P_k^{(1)}$, it is zero for $i = 0, \ldots, n_k - m_k$. Thus, we shall now write that it must also be zero for $i = n_k - m_k + 1, \ldots, n_k - 1$. We see that we have to compute

$$\begin{array}{ll} c(\xi^{n_k+m_k+2} v_k P_k) & c^{(1)}(\xi^{n_k-m_k+1} w_k P_k^{(1)}) \\ \cdots & \cdots \\ c(\xi^{n_k} v_k P_k) & c^{(1)}(\xi^{n_k-1} w_k P_k^{(1)}). \end{array}$$

We set

$$w_k(\xi) = \gamma_0 + \cdots + \gamma_{m_k-1} \xi^{m_k - 1}.$$

Let us first consider the case where $n_k - m_k + 1 \geqslant 0$. The case $n_k - m_k + 1 < 0$ will be treated below. We set

$$v_k(\xi) = \gamma'_0 + \cdots + \gamma'_{m_k-2}\xi^{m_k-2}.$$

By the Euclidean division algorithm for polynomials, we know that there exists a polynomial U_1 of degree $n_k - 1$ and a constant β_{m_k-1} such that

$$\xi^{n_k-m_k+1}w_k(\xi) = U_1(\xi) + \beta_{m_k-1}P_k^{(1)}(\xi).$$

Identifying the coefficients of ξ^{n_k} is both sides of this relation, we obtain

$$\gamma_{m_k-1} = \beta_{m_k-1}.$$

Similarly, there exists a polynomial U_2 of degree $n_k - 1$ and two constants β_{m_k-1} and β_{m_k-2} such that

$$\xi^{n_k-m_k+2}w_k(\xi) = U_2(\xi) + \beta_{m_k-2}P_k^{(1)}(\xi) + \beta_{m_k-1}\xi P_k^{(1)}(\xi).$$

Identifying the coefficients of ξ^{n_k+1} gives the same relation as above for γ_{m_k-1} which proves that β_{m_k-1} is the same as above. Identifying the coefficients of ξ^{n_k} leads to

$$\gamma_{m_k-2} = \beta_{m_k-2} + \beta_{m_k-1}a_{n_k-1}^{(k)}.$$

And so on, there exists a polynomial U_{m_k-1} of degree $n_k - 1$ and constants $\beta_1, \ldots, \beta_{m_k-1}$ such that

$$\xi^{n_k-1}w_k(\xi) = U_{m_k-1}(\xi) + \beta_1 P_k^{(1)}(\xi) + \cdots + \beta_{m_k-1}\xi^{m_k-2}P_k^{(1)}(\xi).$$

Identifying the coefficients of ξ^i for $i = n_k + 1, \ldots, n_k + m_k - 2$, we obtain the same relations as above for $\gamma_2, \ldots, \gamma_{m_k-1}$. Identifying the coefficients of ξ^{n_k}, we get

$$\gamma_1 = \beta_1 + \beta_2 a_{n_k-1}^{(k)} + \cdots + \beta_{m_k-1}a_{n_k-m_k+2}^{(k)}.$$

Finally, there exists a polynomial U_{m_k} of degree $n_k - 1$ and constants $\beta_0, \ldots \beta_{m_k-1}$ such that

$$\xi^{n_k}w_k(\xi) = U_{m_k}(\xi) + \beta_0 P_k^{(1)}(\xi) + \cdots + \beta_{m_k-1}\xi^{m_k-1}P_k^{(1)}(\xi).$$

Identifying, we obtain one more new relation

$$\gamma_0 = \beta_0 + \beta_1 a_{n_k-1}^{(k)} + \cdots + \beta_{m_k-1}a_{n_k-m_k+1}^{(k)}.$$

It must be noticed that, in fact, we have

$$\xi U_i(\xi) = U_{i+1}(\xi) + \beta_{m_k-i-1}P_k^{(1)}(\xi),$$

which shows that β_{m_k-i-1} is the coefficient of the highest term in U_i.

Now, similarly, we know, by the Euclidean division algorithm for polynomials,

that there exists a polynomial V_1 of degree $n_k - 1$ and a constant β'_{m_k-2} such that

$$\xi^{n_k-m_k+2}v_k(\xi) = V_1(\xi) + \beta'_{m_k-2}P_k^{(1)}(\xi).$$

By identification, we have

$$\gamma'_{m_k-2} = \beta'_{m_k-2}.$$

Similarly, there exists a polynomial V_2 of degree $n_k - 1$ and constants β'_{m_k-2} and β'_{m_k-3} such that

$$\xi^{n_k-m_k+3}v_k(\xi) = V_2(\xi) + \beta'_{m_k-3}P_k^{(1)}(\xi) + \beta'_{m_k-2}\xi P_k^{(1)}(\xi).$$

Identifying, we have

$$\gamma'_{m_k-3} = \beta'_{m_k-3} + \beta'_{m_k-2}a_{n_k-1}^{(k)}.$$

And so on, these exists a polynomial V_{m_k-1} of degree $n_k - 1$ and constants $\beta'_0, \ldots, \beta'_{m_k-2}$ such that

$$\xi^{n_k}v_k(\xi) = V_{m_k-1}(\xi) + \beta'_0P_k^{(1)}(\xi) + \cdots + \beta'_{m_k-2}\xi^{m_k-2}P_k^{(1)}(\xi).$$

By identification, we obtain

$$\gamma'_0 = \beta'_0 + \beta'_1a_{n_k-1}^{(k)} + \cdots + \beta'_{m_k-2}a_{n_k-m_k+2}^{(k)}.$$

The preceding relations allow to compute the γ's and the γ''s from the β's and the β''s. Let us now show how to compute the β's and the β''s.

Writing down the orthogonality condition $c(\xi^{n_k-m_k+1}P_{k+1}) = 0$, we obtain

$$c(\xi^{n_k-m_k+1}P_{k+1}) = c(\xi^{n_k-m_k+1}P_k) - c^{(1)}(\xi^{n_k-m_k+1}w_kP_k^{(1)}) - c(\xi^{n_k-m_k+2}v_kP_k) = 0.$$

Expressing $w_kP_k^{(1)}$ and v_kP_k in terms of the U_i's and the V_i's by means of the relations given above and using the orthogonality of P_k and $P_k^{(1)}$, we get

$$c^{(1)}(U_1P_k^{(1)}) + \beta_{m_k-1}c^{(1)}(P_k^{(1)^2}) + c(V_1P_k) + \beta'_{m_k-2}c(P_k^{(1)}P_k) = 0.$$

But, since U_1 and V_1 have degree $n_k - 1$, then $c^{(1)}(U_1P_k^{(1)}) = c(V_1P_k) = 0$, and it follows

$$\beta_{m_k-1}c^{(1)}(P_k^{(1)^2}) + \beta'_{m_k-2}c(P_k^{(1)}P_k) = 0.$$

Writing down the orthogonality condition $c(\xi^{n_k-m_k+2}P_{k+1}) = 0$ and using $c^{(1)}(U_2P_k^{(1)}) = c(V_2P_k) = 0$, we obtain

$$\beta_{m_k-2}c^{(1)}(P_k^{(1)^2}) + \beta_{m_k-1}c^{(1)}(\xi P_k^{(1)^2}) + \beta'_{m_k-3}c(P_k^{(1)}P_k) + \beta'_{m_k-2}c(\xi P_k^{(1)}P_k) = 0.$$

And so on until $c(\xi^{n_k-1}P_{k+1}) = 0$ which gives

$$\beta_1 c^{(1)}(P_k^{(1)^2}) + \cdots + \beta_{m_k-1}c^{(1)}(\xi^{m_k-2}P_k^{(1)^2}) + \beta'_0 c(P_k^{(1)}P_k) + \cdots + \beta'_{m_k-2}c(\xi^{m_k-2}P_k^{(1)}P_k) = 0.$$

Thus we have obtained $m_k - 1$ relations for determining the $2m_k - 1$ unknown coefficients β_i and β'_i.

We shall now write the remaining orthogonality conditions of P_{k+1} as

$$c(\xi^{i-n_k}P_k^{(1)}P_{k+1}) = 0 \quad \text{for } i = n_k, \ldots, n_k + m_k - 1.$$

Thus we obtain for $i = 0, \ldots, m_k - 1$

$$\gamma_0 c^{(1)}(\xi^i P_k^{(1)^2}) + \cdots + \gamma_{m_k-1}c^{(1)}(\xi^{i+m_k-1}P_k^{(1)^2}) + \gamma'_0 c(\xi^{i+1}P_k^{(1)}P_k) + \cdots + \gamma'_{m_k-2}c(\xi^{i+m_k-1}P_k^{(1)}P_k) = c(\xi^i P_k^{(1)}P_k).$$

Replacing, in these relations, the γ's and the γ''s by their expressions above, we obtain m_k more relations for the β's and β''s. Thus, we have the same number of unknowns and relations.

In the case where $n_k - m_k + 1 < 0$, we shall take for v_k a polynomial of degree at most $n_k - 1$ (for $k = 0$, $n_0 = 0$ and there is no polynomial v_0)

$$v_k(\xi) = \gamma'_0 + \cdots + \gamma'_{n_k-1}\xi^{n_k-1}.$$

Let us write the orthogonality conditions $c(\xi^i P_{k+1}) = 0$ for $i = 0, \ldots, n_k - 1$. We shall have to compute the quantities

$$\begin{array}{ll} c(\xi v_k P_k) & c^{(1)}(\xi w_k P_k^{(1)}) \\ \cdots & \cdots \\ c(\xi^{n_k} v_k P_k) & c^{(1)}(\xi^{n_k-1} w_k P_k^{(1)}). \end{array}$$

Since $n_k < m_k - 1$ and w_k is a polynomial of degree at most $m_k - 1$, then there exists a polynomial U_1 of degree $n_k - 1$ and constants $\beta_{n_k}, \ldots, \beta_{m_k-1}$ such that

$$w_k(\xi) = U_1(\xi) + \beta_{n_k}P_k^{(1)}(\xi) + \cdots + \beta_{m_k-1}\xi^{m_k-n_k-1}P_k^{(1)}(\xi).$$

Identifying the coefficients of the successive terms in ξ^i, we obtain (with the convention that $a_i^{(k)} = 0$ for $i < 0$)

$$\gamma_{n_k} = \beta_{n_k} + \beta_{n_k+1}a_{n_k-1}^{(k)} + \cdots + \beta_{m_k-1}a_{2n_k-m_k+1}^{(k)},$$

$$\gamma_{n_k+1} = \beta_{n_k+1} + \beta_{n_k+2}a_{n_k-1}^{(k)} + \cdots + \beta_{m_k-1}a_{2n_k-m_k+2}^{(k)},$$

$$\cdots$$

$$\gamma_{m_k-1} = \beta_{m_k-1}.$$

There exists a polynomial U_2 of degree $n_k - 1$ and constants $\beta_{n_k-1}, \beta_{n_k}, \ldots, \beta_{m_k-1}$ such that

$$\xi w_k(\xi) = U_2(\xi) + \beta_{n_k-1} P_k^{(1)}(\xi) + \cdots + \beta_{m_k-1} \xi^{m_k-n_k} P_k^{(1)}(\xi).$$

By identification we obtain

$$\gamma_{n_k-1} = \beta_{n_k-1} + \beta_{n_k} a_{n_k-1}^{(k)} + \cdots + \beta_{m_k-1} a_{2n_k-m_k}^{(k)}$$

and the same relations as above for $\gamma_{n_k}, \ldots, \gamma_{m_k-1}$.

And so on, there exists a polynomial U_{n_k+1} of degree $n_k - 1$ and constants $\beta_0, \ldots, \beta_{m_k-1}$ such that

$$\xi^{n_k} w_k(\xi) = U_{n_k+1}(\xi) + \beta_0 P_k^{(1)}(\xi) + \cdots + \beta_{m_k-1} \xi^{m_k-1} P_k^{(1)}(\xi).$$

Identifying, we obtain

$$\gamma_0 = \beta_0 + \beta_1 a_{n_k-1}^{(k)} + \cdots + \beta_{m_k-1} a_{n_k-m_k+1}^{(k)}.$$

Similarly, there exists a polynomial V_1 of degree $n_k - 1$ and a constant β'_{n_k-1} such that

$$\xi v_k(\xi) = V_1(\xi) + \beta'_{n_k-1} P_k^{(1)}(\xi).$$

Identifying, we get

$$\gamma'_{n_k-1} = \beta'_{n_k-1}.$$

And so on, there exists a polynomial V_{n_k} of degree $n_k - 1$ and constants $\beta'_0, \ldots, \beta'_{n_k-1}$ such that

$$\xi^{n_k} v_k(\xi) = V_{n_k}(\xi) + \beta'_0 P_k^{(1)}(\xi) + \cdots + \beta'_{n_k-1} \xi^{n_k-1} P_k^{(1)}(\xi),$$

which gives, by identification

$$\gamma'_0 = \beta'_0 + \beta'_1 a_{n_k-1}^{(k)} + \cdots + \beta'_{n_k-1} a_1^{(k)}.$$

Writing down the orthogonality conditions $c(\xi^i P_{k+1}) = 0$ for $i = 0, \ldots, n_k - 1$ and using $c^{(1)}(U_1 P_k^{(1)}) = c(V_1 P_k) = 0$, we obtain

$$\beta_{n_k} c^{(1)}(P_k^{(1)^2}) + \cdots + \beta_{m_k-1} c^{(1)}(\xi^{m_k-n_k-1} P_k^{(1)^2}) + \beta'_{n_k-1} c(P_k^{(1)} P_k) = 0,$$

$$\beta_{n_k-1} c^{(1)}(P_k^{(1)^2}) + \cdots + \beta_{m_k-1} c^{(1)}(\xi^{m_k-n_k} P_k^{(1)^2}) + \beta'_{n_k-2} c(P_k^{(1)} P_k) + \beta'_{n_k-1} c(\xi P_k^{(1)} P_k) = 0,$$

$$\cdots$$

$$\beta_1 c^{(1)}(P_k^{(1)^2}) + \cdots + \beta_{m_k-1} c^{(1)}(\xi^{m_k-2} P_k^{(1)^2}) + \beta'_0 c(P_k^{(1)} P_k) + \cdots + \beta'_{n_k-1} c(\xi^{n_k-1} P_k^{(1)} P_k) = 0.$$

Thus, we have already obtained n_k relations.

Now, writing down the conditions

$$c(\xi^i P_k^{(1)} P_{k+1}) = 0$$

for $i = 0, \ldots, m_k - 1$, we obtain

$$\gamma_0 c^{(1)}(\xi^i P_k^{(1)^2}) + \cdots + \gamma_{m_k-1} c^{(1)}(\xi^{i+m_k-1} P_k^{(1)^2}) + \gamma_0' c(\xi^{i+1} P_k^{(1)} P_k) + \cdots + \gamma_{n_k-1}' c(\xi^{i+n_k} P_k^{(1)} P_k) = c(\xi^i P_k^{(1)} P_k).$$

Replacing, as above, the γ_i and the γ_i' by their expressions, we obtain m_k more relations for the β_i and the β_i'. Thus, finally, we have $n_k + m_k$ relations with the same number of unknowns.

Let us now try to compute recursively $P_{k+1}^{(1)}$. We shall look for $P_{k+1}^{(1)}$ under the form

$$P_{k+1}^{(1)}(\xi) = q_k(\xi) P_k^{(1)}(\xi) + t_k(\xi) P_k(\xi), \tag{2}$$

where q_k is a monic polynomial of degree m_k and t_k polynomial of degree $m_k - 1$ at most.

We set

$$q_k(\xi) = \eta_0 + \cdots + \eta_{m_k-1}\xi^{m_k-1} + \xi^{m_k}.$$

Since the treatment is quite similar to what we did above for (1), we shall only give the results without entering into the details.

Let us first assume that $n_k - m_k \geqslant 0$. The case $n_k - m_k < 0$ will be treated below.

We set

$$t_k(\xi) = \eta_0' + \cdots + \eta_{m_k-1}'\xi^{m_k-1}.$$

There exists $\alpha_0, \ldots, \alpha_{m_k-1}$ such that the η_i are given by

$$\eta_{m_k-1} = \alpha_{m_k-1} + a_{n_k-1}^{(k)},$$

$$\eta_{m_k-2} = \alpha_{m_k-2} + \alpha_{m_k-1} a_{n_k-1}^{(k)} + a_{n_k-2}^{(k)},$$

$$\cdots$$

$$\eta_0 = \alpha_0 + \alpha_1 a_{n_k-1}^{(k)} + \cdots + \alpha_{m_k-1} a_{n_k-m_k+1}^{(k)} + a_{n_k-m_k}^{(k)}.$$

Similarly, there exists $\alpha_0', \ldots, \alpha_{m_k-1}'$ such that the η_i' are given by

$$\eta_{m_k-1}' = \alpha_{m_k-1}',$$

$$\eta_{m_k-2}' = \alpha_{m_k-2}' + \alpha_{m_k-1}' a_{n_k-1}^{(k)},$$

$$\cdots$$

$$\eta_0' = \alpha_0' + \alpha_1' a_{n_k-1}^{(k)} + \cdots + \alpha_{m_k-1}' a_{n_k-m_k+1}^{(k)}.$$

The α_i and the α'_i are obtained by solving the system of linear equations

$$\alpha'_{m_k-1}c(P_k^{(1)}P_k) = -c^{(1)}(P_k^{(1)^2}),$$

$$\alpha_{m_k-1}c^{(1)}(P_k^{(1)^2}) + \alpha'_{m_k-2}c(P_k^{(1)}P_k) + \alpha'_{m_k-1}c(\xi P_k^{(1)}P_k) = -c^{(1)}(\xi P_k^{(1)^2}),$$

$$\dots$$

$$\alpha_1 c^{(1)}(P_k^{(1)^2}) + \cdots + \alpha_{m_k-1}c^{(1)}(\xi^{m_k-2}P_k^{(1)^2}) + \alpha'_0 c(P_k^{(1)}P_k) + \cdots$$
$$+ \alpha'_{m_k-1}c(\xi^{m_k-1}P_k^{(1)}P_k) = -c^{(1)}(\xi^{m_k-1}P_k^{(1)^2}),$$

followed by the equations

$$\eta_0 c^{(1)}(\xi^i P_k^{(1)^2}) + \cdots + \eta_{m_k-1}c^{(1)}(\xi^{i+m_k-1}P_k^{(1)^2}) + \eta'_0 c(\xi^{i+1}P_k^{(1)}P_k) + \cdots$$
$$+ \eta'_{m_k-1}c(\xi^{i+m_k}P_k^{(1)}P_k) = -c^{(1)}(\xi^{i+m_k}P_k^{(1)^2})$$

for $i = 0, \dots, m_k - 1$, where the η_i and the η'_i have been replaced by their corresponding expressions with respect to the α_i and the α'_i.

In the case $n_k - m_k < 0$, we shall take for t_k a polynomial of degree at most $n_k - 1$ (for $k = 0$, $n_0 = 0$ and there is no polynomial t_0)

$$t_k(\xi) = \eta'_0 + \cdots + \eta'_{n_k-1}\xi^{n_k-1}.$$

With again the convention that $a_i^{(k)} = 0$ for $i < 0$, we have

$$\eta_{m_k-1} = \alpha_{m_k-1} + a_{n_k-1}^{(k)},$$

$$\dots$$

$$\eta_0 = \alpha_0 + \alpha_1 a_{n_k-1}^{(k)} + \cdots + \alpha_{m_k-1}a_{n_k-m_k+1}^{(k)} + a_{n_k-m_k}^{(k)},$$

and

$$\eta'_{n_k-1} = \alpha'_{n_k-1},$$

$$\dots$$

$$\eta'_0 = \alpha'_0 + \alpha'_1 a_{n_k-1}^{(k)} + \cdots + \alpha'_{n_k-1}a_1^{(k)}.$$

The α_i and the α'_i are solution of

$$\alpha_{n_k}c^{(1)}(P_k^{(1)^2}) + \cdots + \alpha_{m_k-1}c^{(1)}(\xi^{m_k-n_k-1}P_k^{(1)^2}) + c^{(1)}(\xi^{m_k-n_k}P_k^{(1)^2})$$
$$+ \alpha'_{n_k-1}c(P_k^{(1)}P_k) = 0$$

$$\dots$$

$$\alpha_1 c^{(1)}(P_k^{(1)^2}) + \cdots + \alpha_{m_k-1}c^{(1)}(\xi^{m_k-2}P_k^{(1)^2}) + c^{(1)}(\xi^{m_k-1}P_k^{(1)^2})$$
$$+ \alpha'_0 c(P_k^{(1)}P_k) + \cdots + \alpha'_{n_k-1}c(\xi^{n_k-1}P_k^{(1)}P_k) = 0,$$

followed by

$$\eta_0 c^{(1)}(\xi^i P_k^{(1)^2}) + \cdots + \eta_{m_k-1} c^{(1)}(\xi^{i+m_k-1} P_k^{(1)^2}) + c^{(1)}(\xi^{i+n_k} P_k^{(1)^2})$$
$$+ \eta_0' c(\xi^{i+1} P_k^{(1)} P_k) + \cdots + \eta_{n_k-1}' c(\xi^{i+n_k} P_k^{(1)} P_k) = 0$$

for $i = 0, \ldots, m_k - 1$ where the η_i and the η_i' have been replaced by their expressions with respect to the α_i and the α_i'.

If the systems giving the coefficients are singular, it means that $P_{k+1}^{(1)}$ does not exist and the value of m_k has to be increased until a regular polynomial $P_{k+1}^{(1)}$ has been obtained.

We shall now use (1) and (2) in order to cure near-breakdowns in the CGS algorithm. Let us mention that other techniques for treating the same problem are known as look-ahead strategies first proposed by Parlett et al. [34]. They have been worked out for the biconjugate gradient algorithm by Freund et al. [24]. However, in our case, writing down explicitly the recurrence relationships of orthogonal polynomials, instead of using matricial notations as in the look-ahead algorithm 3.1 of [24], will allow us to use these relations and to square them for obtaining an algorithm for implementing the CGS.

3. Near-breakdown in the CGS algorithm

3.1. *The algorithm*

Since the CGS algorithm consists in taking

$$r_k = P_k^2(A) r_0,$$

we shall take the relation (1) of the BSMRZ and square it. We obtain

$$P_{k+1}^2 = (1 - \xi v_k)^2 P_k^2 - 2(1 - \xi v_k)\xi w_k P_k P_k^{(1)} + \xi^2 w_k^2 P_k^{(1)^2}.$$

For using this relation, it is necessary to compute recursively the polynomials $P_{k+1}^{(1)^2}$ and $P_{k+1} P_{k+1}^{(1)}$. Thus, let us square the relation (2) of the BSMRZ. We obtain

$$P_{k+1}^{(1)^2} = q_k^2 P_k^{(1)^2} + 2 q_k t_k P_k P_k^{(1)} + t_k^2 P_k^2.$$

Finally, multiplying (1) by (2) leads to

$$P_{k+1} P_{k+1}^{(1)} = (q_k - \xi q_k v_k - \xi t_k w_k) P_k P_k^{(1)} - \xi q_k w_k P_k^{(1)^2} + t_k (1 - \xi v_k) P_k^2.$$

Thus, if we set

$$r_k = P_k^2(A) r_0,$$
$$z_k = P_k^{(1)^2}(A) r_0,$$
$$s_k = P_k(A) P_k^{(1)}(A) r_0,$$

then we obtain the following algorithm called, for obvious reasons, the BSMRZS

$$
\begin{aligned}
r_{k+1} &= (I - Av_k(A))^2 r_k - 2(I - Av_k(A))Aw_k(A)s_k + A^2 w_k^2(A)z_k,\\
x_{k+1} &= x_k - (Av_k(A) - 2I)v_k(A)r_k + 2(I - Av_k(A))w_k(A)s_k - Aw_k^2(A)z_k,\\
z_{k+1} &= q_k^2(A)z_k + 2q_k(A)t_k(A)s_k + t_k^2(A)r_k,\\
s_{k+1} &= (q_k(A) - Aq_k(A)v_k(A) - At_k(A)w_k(A))s_k - Aq_k(A)w_k(A)z_k\\
&\quad + t_k(A)(I - Av_k(A))r_k.
\end{aligned}
\tag{3}
$$

In this algorithm, the only possible breakdown is the *incurable hard* one which occurs when the dimension of the Krylov subspace is equal to n and a zero denominator is encountered somewhere in the algorithm. However, due to the arithmetic of the computer, this situation will almost never arise.

The implementation of this algorithm needs the computation of several products of polynomials. Let us set

$$P(\xi) = a_0 + \cdots + a_n\xi^n, \qquad Q(\xi) = b_0 + \cdots + b_k\xi^k,$$

with $n \leqslant k$. Then the product of these two polynomials is given by

$$P(\xi)Q(\xi) = e_0 + \cdots + e_{n+k}\xi^{n+k},$$

with

$$e_i = \sum_{j=\max(0,i-n)}^{\min(k,i)} b_j a_{i-j}.$$

Let us mention that it would be interesting, in order to reduce the number of arithmetical operations (and thus, possibly, the numerical instability), to use a procedure based on the fast Fourier transform for effecting the product of two polynomials; see [30, pp. 500ff.] and [44, p. 226].

The main interest of the CGS, and in fact the reason for its construction, is that the matrix A^* is not needed in the scalar products giving the coefficients of the recurrence relations. As shown in [15], this feature can be preserved in the MRZS which is the breakdown-free version of the CGS.

This feature can also be kept in the BSMRZS and we have to compute, for $i = 0, \ldots, 2m_k - 1$, the quantities

$$c^{(1)}(\xi^i P_k^{(1)^2}) = (y, A^{i+1}P_k^{(1)^2}(A)r_0) = (y, A^{i+1}z_k)$$

and the following quantities for $i = 0, \ldots, 2m_k - 1$ (in the case $m_k \leqslant n_k$) or $i = 0, \ldots, n_k + m_k - 1$ (in the case $m_k > n_k$)

$$c(\xi^i P_k^{(1)} P_k) = (y, A^i P_k^{(1)}(A)P_k(A)r_0) = (y, A^i s_k).$$

The CGS was obtained by squaring the BCG. Our BSMRZS was obtained

by squaring the BSMRZ which reduces to the method called A8/B8, when $\forall k, m_k = 1$ [14], which is a new algorithm for implementing Lanczos' method.

In the case of a true breakdown, due to $c^{(1)}(\xi^i P_k^{(1)^2}) = 0$ for $i = 0, \ldots, m_k - 2$, the BSMRZS reduces, if $c(P_k^{(1)} P_k) \neq 0$, to the algorithm SMRZS described in [15] which is a breakdown-free implementation of the CGS. It must be noticed that the SMRZS can suffer from ghost breakdowns while the BSMRZS cannot.

In [24], a look-ahead version of the biconjugate gradient algorithm was given. Actually, it is not yet known if this version could be squared for obtaining a look-ahead CGS algorithm working without using A^*. However, our approach, based on orthogonal polynomials, of the problems of breakdown and near-breakdown in such algorithms seems simpler than those based on purely linear algebra techniques. In particular, it seems to us that it would have been much more difficult to obtain the coefficients of the polynomials involved in the recurrence relations without the help of orthogonal polynomials.

It must be noticed that Sonneveld [36] implemented his method by squaring the recurrence relationships of the Lanczos/Orthomin algorithm. Since, in the BSMRZS, one of the recurrence relationships we are squaring is different, our algorithm does not reduce to Sonneveld's algorithm when $\forall k, m_k = 1$. The look-ahead procedure, called CSCGS, proposed by Chan and Szeto [17] is also based on the Lanczos/Orthomin algorithm and it cures only the true breakdowns by a 2×2 composite step. Since these authors assume that no ghost breakdown occurs then, as shown by the discussion in [10] (see also [22]), a jump of length 2 is sufficient. The goal of these authors was not to cure all the possible breakdowns but to obtain only a partial cure at a relatively modest modification of Sonneveld's algorithm. Their procedure reduces to Sonneveld's when no jump occurs. The CSCGS was obtained by squaring the algorithm CSBCG proposed in [1,2] for treating true breakdowns in Lanczos/Orthomin.

3.2. *Tests for near-breakdown*

In section 2, we saw that, in order to avoid a breakdown, we have to find, at each step k, the value of m_k such that

$$c^{(1)}(\xi^i P_k^{(1)}(\xi)) = 0 \quad \text{for } i = 0, \ldots, n_k + m_k - 2.$$

and

$$c^{(1)}(\xi^{n_k + m_k - 1} P_k^{(1)}(\xi)) \neq 0.$$

If the quantity $c^{(1)}(\xi^{n_k + m_k - 1} P_k^{(1)}(\xi))$, which appears as the denominator of some of the coefficients of the recurrence relations (1) and (2), is close to zero, then numerical instability could affect the algorithm. This situation is called a near-breakdown. Thus, it seems that a near-breakdown could be avoided by defining

the length m_k of the jump by

$$\left|c^{(1)}(\xi^i P_k^{(1)})\right| \leqslant \epsilon \quad \text{for } i = n_k, \ldots, n_k + m_k - 2$$

and

$$\left|c^{(1)}(\xi^i P_k^{(1)})\right| > \epsilon \quad \text{for } i = n_k + m_k - 1,$$

where ϵ is a given threshold value.

Let us first notice that, since the polynomials $P_k^{(1)}$ are no more involved in the algorithm, these inequalities can be replaced by the equivalent ones

$$\left|c^{(1)}(\xi^i P_k^{(1)^2})\right| \leqslant \epsilon \quad \text{for } i = 0, \ldots, m_k - 2$$

and

$$\left|c^{(1)}(\xi^i P_k^{(1)^2})\right| > \epsilon \quad \text{for } i = m_k - 1.$$

In an earlier version of our program [6], we used this test for deciding when and how far to jump. However, the jumps were very sensitive to the value of ϵ. We tried many examples. For some values of ϵ, the exact solution was obtained for $n_k \leqslant n$ as expected from the theory while, for some other values of ϵ, the solution could not be attained and sometimes an overflow even occurred. However, it was observed numerically that the quantity

$$\sigma_{k+1}^{(1)} = c(\xi P_k^{(1)} P_k) - c(P_k^{(1)} P_k) c^{(1)}(\xi P_k^{(1)^2}) / c^{(1)}(P_k^{(1)^2})$$

was close to zero in two cases: (i) when a jump has to begin and (ii) when the exact solution is about to be obtained. So, we began thinking about a test based on this observation. We first had to understand the meaning of the quantity $\sigma_{k+1}^{(1)}$ and then to find a test for deciding how far to jump.

At each step k, the value of m_k is first set to 1. Then, we shall check whether $\sigma_{k+1}^{(1)}$, as defined above, is zero or not.

Let us discuss this condition in detail. As we shall see now, $\sigma_{k+1}^{(1)}$ can be zero in two different cases.

We have, when $m_k = 1$,

$$P_{k+1}(\xi) = P_k(\xi) - \delta_k \xi P_k^{(1)}(\xi), \tag{4}$$

with $\delta_k = c(P_k^{(1)} P_k)/c(\xi P_k^{(1)^2})$. Multiplying both sides of (4) by $P_k^{(1)}$ and applying the linear function $c^{(1)}$ we obtain

$$c^{(1)}(P_k^{(1)} P_{k+1}) = c(\xi P_k^{(1)} P_k) - c(P_k^{(1)} P_k) c^{(1)}(\xi P_k^{(1)^2}) / c^{(1)}(P_k^{(1)^2}) = \sigma_{k+1}^{(1)}.$$

Since $\sigma_{k+1}^{(1)} = (y, A P_k^{(1)}(A) P_{k+1}(A) r_0)$, this quantity is zero if $P_{k+1}(A) r_0 = 0$, which means that the exact solution will be obtained at the end of the current iteration, since $r_{k+1} = P_{k+1}^2(A) r_0$.

The quantity $\sigma_{k+1}^{(1)}$ can also be zero in another case. We have

$$\sigma_{k+1}^{(1)} = c^{(1)}(P_k^{(1)} P_{k+1}) = c(\xi P_k^{(1)} P_{k+1}).$$

But $\xi P_k^{(1)}(\xi) = \xi^{n_k+1} + Q_k(\xi)$ where Q_k is a polynomial of degree n_k at most. Since (4) holds, we are in the case $m_k = 1$, that is, $n_{k+1} = n_k + 1$ and, by the orthogonality property of P_{k+1}, we finally obtain

$$\sigma_{k+1}^{(1)} = c^{(1)}(P_k^{(1)} P_{k+1}) = c(\xi^{n_k+1} P_{k+1}).$$

Since $P_{k+1}^{(1)}$ has degree n_{k+1} and, by the orthogonality condition of P_{k+1}, we also have $\sigma_{k+1}^{(1)} = c(P_{k+1}^{(1)} P_{k+1})$. Thus, a breakdown cannot occur at the next step if this quantity is different from zero.

If $\sigma_{k+1}^{(1)} = 0$, it means that the orthogonality conditions of P_{k+1} are satisfied farther than normal since P_{k+1} has degree n_{k+1} at most. Thus a ghost breakdown will occur at the beginning of the next iteration due to a division by zero and thus it is necessary to jump. Obviously a near-breakdown occurs if this quantity is close to zero and we shall also jump in this case. It must be noticed that the quantity $\sigma_{k+1}^{(1)}$ is quite easy to compute since, in fact, by the last relation,

$$\sigma_{k+1}^{(1)} = (y, s_{k+1}).$$

Thus we now have to take a value of m_k greater than 1 and decide how far to jump. The value of m_k is set to 2 and the system giving the coefficients β_i, β_i', α_i and α_i' is solved. If this system is singular (pivot $= 0$) or nearly singular ($|\text{pivot}| \leqslant \epsilon_1$) then m_k is changed to $m_k + 1$ and the procedure is repeated until a non-nearly singular system has been found.

After solving the system, if the new residual vector is zero (or small) then the solution of $Ax = b$ has been obtained. If this is not the case, the iterative process has to be continued further and, at the beginning of the next step, a division by zero could occur if we are not careful enough. Thus, before going on, we have to check this point. If the quantity by which we shall have to divide at the beginning of the next iteration is zero (or close to it) it means that the preceding jump was not long enough and we have to return to that iteration and increase again the value of m_k by 1.

Thus finally, for finding the length of the jump, two different tests have to be performed: (i) the singularity (or the near-singularity) of the system giving the coefficients of the polynomials, and (ii) the value of the quantity by which we shall have to divide at the beginning of the next iteration if the solution of $Ax = b$ has not been reached.

Let us now discuss this second point. In section 2, for obtaining the last m_k equations of the system giving the β_i, and the β_i', we wrote down the orthogonality conditions

$$c(\xi^i P_k^{(1)} P_{k+1}) = 0 \quad \text{for } i = 0, \ldots, m_k - 1.$$

Let us set

$$\sigma_{k+1}^{(m_k)} = c(\xi^{m_k} P_k^{(1)} P_{k+1})$$

and remark that, when $m_k = 1$, this is the same $\sigma_{k+1}^{(1)}$ as above. It is easy to see that

$$\sigma_{k+1}^{(m_k)} = c^{(1)}(\xi^{m_k - 1} P_k^{(1)} P_{k+1}) = c(P_{k+1}^{(1)} P_{k+1}) = c(\xi^{n_k+1} P_{k+1}) = (y, s_{k+1}).$$

As before, if this quantity is zero it means that either we shall obtain the exact solution at the end of the current iteration or have a division by zero at the beginning of the next one. In order to avoid such a division by zero (or by a quantity close to it) we must sill increase the value of m_k in the current step, solve the system giving the coefficients of the polynomials and so on until a satisfactory value of $\sigma_{k+1}^{(m_k)}$ has been obtained.

Thus, we first use the quantity $\sigma_{k+1}^{(1)}$ for deciding to start jumping and then the quantity $\sigma_{k+1}^{(m_k)}$, with $m_k > 1$, for checking if the jump is long enough.

It must be noticed that we proved, in passing, that the SMRZ and the BSMRZ described in [10, 11] are more general than we previously thought since the case of a ghost breakdown, which occurs if $\sigma_k^{(m_k - 1)} = c(\xi^{n_k} P_k) = 0$, can now be avoided. The BMRZ cannot be generalized for treating near-breakdowns.

Near-breakdown (or possible numerical instability) at the step k is also detected by testing some ratios that are needed in the computation of β_0, α_0 and α_0' when $m_k = 1$. Moreover, when $m_k = 1$, another possible cause for breakdown is that $(y, Az_k) = 0$. All these cases are detailed in the pseudocode of the algorithm given in the next section.

4. Algorithm design

We shall now analyze how the algorithm has been developed and first give it in a pseudocode form.

In order to minimize, in the code of the program, the storage and the number of operations to be performed in the algorithm, some tricks have been used.

In the case where a breakdown or a near-breakdown has been detected, it is necessary to jump and to solve two auxiliary systems, the first one giving the coefficients β and β' and the other one the coefficients α and α' and then to compute the coefficients γ, γ', η and η' of the polynomials w_k, v_k, q_k and t_k which appear in the BSMRZS. Thus the following quantities must be computed at the iteration k

	$m_k \leqslant n_k$	$m_k > n_k$
$d_i = (y, A^i s_k)$	$i = 0, \ldots, 2m_k - 1,$	$i = 0, \ldots, n_k + m_k - 1,$
$c_i = (y, A^{i+1} z_k)$	$i = 0, \ldots, 2m_k - 1,$	$i = 0, \ldots, 2m_k - 1,$
$h_{i0} = c_i$	$i = 0, \ldots, m_k - 1,$	$i = 0, \ldots, m_k - 1,$
$h_{ij} = \sum_{l=0}^{j-1} c_{i+l} \cdot a_{n_k - j + l}^{(1)} + c_{i+j}$	$j = 1, \ldots, m_k,$	$j = 1, \ldots, m_k,$

$$f_{i0} = d_{i+1} \qquad i = 0, \ldots, m_k - 1, \qquad i = 0, \ldots, m_k - 1,$$

$$f_{ij} = \sum_{l=0}^{j-1} d_{i+l+1} \cdot a^{(k)}_{n_k - j + l} + d_{i+j+1} \qquad j = 1, \ldots, m_k - 1, \qquad j = 1, \ldots, n_k - 1,$$

where $a^{(k)}_{n_k - j + l} = 0$ for $j - 1 > n_k$.

These systems are

System for the β and the β'

Case $m_k \leqslant n_k$

$2m_k - 1$ equations with:

m_k unknowns $\quad \beta_0, \ldots, \beta_{m_k - 1}$

$m_k - 1$ unknowns $\quad \beta'_0, \ldots, \beta'_{m_k - 2}$

$$m_k - 1 \text{ eq.} \begin{cases} \beta_{m_k-1} c_0 + \beta'_{m_k-2} d_0 = 0, \\ \ldots \\ \beta_1 c_0 + \cdots + \beta_{m_k-1} c_{m_k-2} + \beta'_0 d_0 + \cdots + \beta'_{m_k-2} d_{m_k-2} = 0 \end{cases}$$

$$m_k \text{ eq.} \begin{cases} \beta_0 h_{00} + \beta_1 h_{01} + \cdots + \beta_{m_k-1} h_{0,m_k-1} \\ \quad + \beta'_0 f_{00} + \beta'_1 f_{01} + \cdots + \beta'_{m_k-2} f_{0,m_k-2} = d_0, \\ \ldots \\ \beta_0 h_{m_k-1,0} + \beta_1 h_{m_k-1,1} + \cdots + \beta_{m_k-1} h_{m_k-1,m_k-1} \\ \quad + \beta'_0 f_{m_k-1,0} + \beta'_1 f_{m_k-1,1} + \cdots + \beta'_{m_k-2} f_{m_k-1,m_k-2} = d_{m_k-1}. \end{cases}$$

System for the β and the β'

Case $m_k > n_k$

$n_k + m_k$ equations with:

m_k unknowns $\quad \beta_0, \ldots, \beta_{m_k - 1}$

n_k unknowns $\quad \beta'_0, \ldots, \beta'_{n_k - 1}$

$$n_k \text{ eq.} \begin{cases} \beta_{n_k} c_0 + \cdots + \beta_{m_k-1} c_{m_k-n_k-1} + \beta'_{n_k-1} d_0 = 0, \\ \ldots \\ \beta_1 c_0 + \cdots + \beta_{m_k-1} c_{m_k-2} + \beta'_0 d_0 + \cdots + \beta'_{n_k-1} d_{n_k-1} = 0 \end{cases}$$

$$m_k \text{ eq.} \begin{cases} \beta_0 h_{00} + \beta_1 h_{01} + \cdots + \beta_{m_k-1} h_{0,m_k-1} \\ \quad + \beta'_0 f_{00} + \beta'_1 f_{01} + \cdots + \beta'_{n_k-1} f_{0,n_k-1} = d_0, \\ \ldots \\ \beta_0 h_{m_k-1,0} + \beta_1 h_{m_k-1,1} + \cdots + \beta_{m_k-1} h_{m_k-1,m_k-1} \\ \quad + \beta'_0 f_{m_k-1,0} + \beta'_1 f_{m_k-1,1} + \cdots + \beta'_{n_k-1} f_{m_k-1,n_k-1} = d_{m_k-1}. \end{cases}$$

System for the α and the α'

Case $m_k \leqslant n_k$

$2m_k$ equations with:

m_k unknowns $\alpha_0, \ldots, \alpha_{m_k-1}$

m_k unknowns $\alpha'_0, \ldots, \alpha'_{m_k-1}$

$$
m_k \text{ eq.}\begin{cases}
\alpha'_{m_k-1}d_0 = -c_0, \\
\alpha_{m_k-1}c_0 + \alpha'_{m_k-2}d_0 = -\alpha'_{m_k-1}d_1 - c_1, \\
\ldots \\
\alpha_1 c_0 + \cdots + \alpha_{m_k-1}c_{m_k-2} + \alpha'_0 d_0 + \cdots + \alpha'_{m_k-2}d_{m_k-2} \\
\quad = -\alpha'_{m_k-1}d_{m_k-1} - c_{m_k-1}
\end{cases}
$$

$$
m_k \text{ eq.}\begin{cases}
\alpha_0 h_{00} + \alpha_1 h_{01} + \cdots + \alpha_{m_k-1}h_{0,m_k-1} \\
\quad +\alpha'_0 f_{00} + \alpha'_1 f_{01} + \cdots + \alpha'_{m_k-2}f_{0,m_k-2} = -\alpha'_{m_k-1}f_{0,m_k-1} - h_{0,m_k}, \\
\ldots \\
\alpha_0 h_{m_k-1,0} + \alpha_1 h_{m_k-1,1} + \cdots + \alpha_{m_k-1}h_{m_k-1,m_k-1} + \alpha'_0 f_{m_k-1,0} + \alpha'_1 f_{m_k-1,1} \\
\quad + \cdots + \alpha'_{m_k-2}f_{m_k-1,m_k-2} = -\alpha'_{m_k-1}f_{m_k-1,m_k-1} - h_{m_k-1,m_k}.
\end{cases}
$$

System for the α and the α'

Case $m_k > n_k$

$n_k + m_k$ equations with:

m_k unknowns $\alpha_0, \ldots, \alpha_{m_k-1}$

n_k unknowns $\alpha'_0, \ldots, \alpha'_{n_k-1}$

$$
n_k \text{ eq.}\begin{cases}
\alpha_{n_k}c_0 + \cdots + \alpha_{m_k-1}c_{m_k-n_k-1} + \alpha'_{n_k-1}d_0 = -c_{m_k-n_k}, \\
\ldots \\
\alpha_1 c_0 + \cdots + \alpha_{m_k-1}c_{m_k-2} + \alpha'_0 d_0 + \cdots + \alpha'_{n_k-1}d_{n_k-1} = -c_{m_k-1}
\end{cases}
$$

$$
m_k \text{ eq.}\begin{cases}
\alpha_0 h_{00} + \alpha_1 h_{01} + \cdots + \alpha_{m_k-1}h_{0,m_k-1} \\
\quad +\alpha'_0 f_{00} + \alpha'_1 f_{01} + \cdots + \alpha'_{n_k-1}f_{0,n_k-1} = -h_{0,m_k}, \\
\ldots \\
\alpha_0 h_{m_k-1,0} + \alpha_1 h_{m_k-1,1} + \cdots + \alpha_{m_k-1}h_{m_k-1,m_k-1} \\
\quad +\alpha'_0 f_{m_k-1,0} + \alpha'_1 f_{m_k-1,1} + \cdots + \alpha'_{n_k-1}f_{m_k-1,n_k-1} = -h_{m_k-1,m_k}.
\end{cases}
$$

These two systems can be reduced to one single system with the same matrix and two different right hand sides. We have

System for the case $m_k \leqslant n_k$

System of $2m_k - 1$ equations for β and β'

System of $2m_k - 1$ equations for α and α'

$(\alpha'_{m_k-1} = -c_0/d_0)$

β_0	β'_0	β_1	$\cdots$	β'_{m_k-2}	β_{m_k-1}	for	for
α_0	α'_0	α_1	$\cdots$	α'_{m_k-2}	α_{m_k-1}	β, β'	α, α'
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$h_{m_k-1,0}$	$f_{m_k-1,0}$	$h_{m_k-1,1}$	$\cdots$	f_{m_k-1,m_k-2}	h_{m_k-1,m_k-1}	d_{m_k-1}	$-h_{m_k-1,m_k} + f_{m_k-1,m_k-1}c_0/d_0$
$h_{m_k-2,0}$	$f_{m_k-2,0}$	$h_{m_k-2,1}$	$\cdots$	f_{m_k-2,m_k-2}	h_{m_k-2,m_k-1}	d_{m_k-2}	$-h_{m_k-2,m_k} + f_{m_k-2,m_k-1}c_0/d_0$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
h_{00}	f_{00}	h_{01}	$\cdots$	f_{0,m_k-2}	h_{0,m_k-1}	d_0	$-h_{0,m_k} + f_{0,m_k-1}c_0/d_0$
	d_0	c_0	$\cdots$	d_{m_k-2}	c_{m_k-2}	0	$-c_{m_k-1} + d_{m_k-1}c_0/d_0$
				$\vdots$	$\vdots$	$\vdots$	$\vdots$
				$\vdots$	$\vdots$	$\vdots$	$\vdots$
	0			d_1	c_1	$\vdots$	$-c_2 + d_2c_0/d_0$
				d_0	c_0	0	$-c_1 + d_1c_0/d_0$

System for the case $m_k > n_k$

System of $n_k + m_k$ equations for β and β'

System of $n_k + m_k$ equations for α and α'

β_0	β'_0	β_1	$\cdots$	β'_{n_k-1}	β_{n_k}	β_{n_k+1}	$\cdots$	β_{m_k-1}	for	for
α_0	α'_0	α_1	$\cdots$	α'_{n_k-1}	α_{n_k}	α_{n_k+1}	$\cdots$	α_{m_k-1}	β, β'	α, α'
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$	$\downarrow$
$h_{m_k-1,0}$	$f_{m_k-1,0}$	$h_{m_k-1,1}$	$\cdots$	f_{m_k-1,n_k-1}	h_{m_k-1,n_k}	h_{m_k-1,n_k+1}	$\cdots$	h_{m_k-1,m_k-1}	d_{m_k-1}	$-h_{m_k-1,m_k}$
$h_{m_k-2,0}$	$f_{m_k-2,0}$	$h_{m_k-2,1}$	$\cdots$	f_{m_k-2,n_k-1}	h_{m_k-2,n_k}	h_{m_k-2,n_k+1}	$\cdots$	h_{m_k-2,m_k-1}	d_{m_k-2}	$-h_{m_k-2,m_k}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
$h_{0,0}$	$f_{0,0}$	$h_{0,1}$	$\cdots$	f_{0,n_k-1}	h_{0,n_k}	h_{0,n_k+1}	$\cdots$	h_{0,m_k-1}	d_0	$-h_{0,m_k}$
	d_0	c_0	$\cdots$	d_{n_k-1}	c_{n_k}	c_{n_k+1}	$\cdots$	c_{m_k-2}	0	$-c_{m_k-1}$
				$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
	0			$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
				d_1	c_1	c_2	$\cdots$	$c_{m_k-n_k}$	$\vdots$	$-c_{m_k-n_k+1}$
				d_0	c_0	c_1	$\cdots$	$c_{m_k-n_k-1}$	0	$-c_{m_k-n_k}$

These systems are solved by Gaussian elimination with partial pivoting and the matrix is stored is a quasi upper triangular form to speed up the computation and to reduce the amount of operations. Obviously, Gaussian elimination could be replaced by any other method, for instance the block bordering method whose subroutine can be found in [4].

If at iteration k, for an $m_k > 1$, this system is singular or almost singular ($|\text{pivot}| \leqslant \epsilon_1$ in the triangularization stage or $|\text{determinant}| \leqslant \epsilon_1$), then, as said in section 3.2, the value of m_k has to be increased until a non nearly-singular system has been obtained and $|\sigma_{k+1}^{(m_k)}|$ is greater than some threshold ϵ. Thus at the end of the iteration k, the final value of m_k is used for setting $n_{k+1} = n_k + m_k$.

After solving these systems, the γ_i, γ_i', η_i and η_i' are obtained from the β_i, β_i', α_i and α_i' as indicated in section 2. The computation of the γ_i, γ_i', η_i and η_i' can be put under a unified form by introducing a second index j as follows

	$m_k \leqslant n_k$	$m_k > n_k$
$\gamma_{j,i} = \sum_{l=1}^{j-i} \beta_{l+i} a_{n_k-l}^{(k)} + \beta_i$	$i = 0, \ldots, m_k - 1,$	$i = 0, \ldots, m_k - 1,$
$\gamma_{i,i} = \beta_i$	$j = m_k - 1,$	$j = m_k - 1,$
$\gamma'_{j,i} = \sum_{l=1}^{j-i} \beta'_{l+i} a_{n_k-l}^{(k)} + \beta'_i$	$i = 0, \ldots, m_k - 2,$	$i = 0, \ldots, n_k - 1,$
$\gamma'_{i,i} = \beta'_i$	$j = m_k - 2,$	$j = n_k - 1,$
$\eta_{j,i} = \sum_{l=1}^{j-i} \alpha_{l+1} a_{n_k-l}^{(k)} + \alpha_i + a_{n_k-m_k+i}^{(k)}$	$i = 0, \ldots, m_k - 1,$	$i = 0, \ldots, m_k - 1,$
$\eta_{i,i} = \alpha_i + a_{n_k-1}^{(k)}$	$j = m_k - 1,$	$j = m_k - 1,$
$\eta'_{j,i} = \sum_{l=1}^{j-i} \alpha'_{l+i} a_{n_k-l}^{(k)} + \alpha'_i$	$i = 0, \ldots, m_k - 1,$	$i = 0, \ldots, n_k - 1,$
$\eta'_{i,i} = \alpha'_i$	$j = m_k - 1,$	$j = n_k - 1.$

The four vectors r_{k+1}, x_{k+1}, z_{k+1} and s_{k+1} are obtained as linear combinations (with the coefficients of the polynomials appearing in (3)) of vectors of the form

	$m_k \leqslant n_k$	$m_k > n_k$
$p_i = A^i z_k$	$i = 0, \ldots, 2m_k,$	$i = 0, \ldots, 2m_k,$
$g_i = A^i r_k$	$i = 0, \ldots, 2m_k - 2,$	$i = 0, \ldots, 2n_k,$
$u_i = A^i s_k$	$i = 0, \ldots, 2m_k - 1,$	$i = 0, \ldots, n_k + m_k.$

Thus these vectors are computed once and stored as the columns of three different matrices.

At the end of the iteration k we also need to compute and store for the next step, using (1) and (2), the coefficients of the polynomials P_{k+1} and $P_{k+1}^{(1)}$, that is, the quantities

$$a_i^{(k+1)} \qquad i = 0, \ldots, n_k + m_k - 1,$$

$$b_i^{(k+1)} \qquad i = 1, \ldots, n_k + m_k.$$

Some auxiliary quantities are also needed in the program.

The solution is obtained at the iteration k when $\| r_{k+1} \| \leqslant \epsilon_2$. The corresponding value of n_{k+1} must be less than or equal to n. However, due to the precision of the computer, the solution can possibly be reached for an n_{k+1} greater than n and thus a value $n_{\max} \geqslant n$ has been introduced to let the algorithm continue until this value. Let us recall that, in exact arithmetic, there exists an index $k \leqslant n$ such that $r_k = 0$. Obviously, the value of $n_{\max}$ can also be taken smaller than n. In that case, the program will stop when $n_{k+1} \geqslant n_{\max}$. In order to avoid unnecessary computations and rounding errors, the maximum norm was chosen.

The length of the jump (denoted by m_k) affects in a very heavy way the storage requirements of the algorithm because its value appears in the dimensions of all the working arrays and of all the working vectors. In theory, the size of the jump can reach the value $n_{\max}$ because, at the first iteration, a value of m_k equal to $n_{\max}$ can be found. In practice, especially when the dimension of the system is large, it is necessary to set an upper bound to the value of the jump; this value is denoted $m_{k\max}$.

Let us now give the pseudocode of the algorithm.

Algorithm BSMRZS $(A, b, x_0, y, n, n_{\max}, m_{k\max}, \epsilon, \epsilon_1, \epsilon_2)$

Step 1. Initializations

$r_0 \leftarrow b - Ax_0$
$p_0 = z_0 = r_0$
$u_0 = s_0 = r_0$
$g_0 = r_0$
$a_0^{(0)} \leftarrow 1$
$n_0 \leftarrow 0$
$m_{-1} \leftarrow 0$
$\sigma_0^{(0)} \leftarrow (y, r_0)$

If $\| r_0 \| \leqslant \epsilon_2$ **then**
solution obtained.
stop.
end if

Step 2. For $k = 0, 1, 2, \ldots$ **until** convergence **do**

$m_k \leftarrow 1$
$p_1 \leftarrow Ap_0$
$p_2 \leftarrow Ap_1$
$u_1 \leftarrow Au_0$

$d_0 \leftarrow \sigma_k^{(m_{k-1})}$
$c_0 \leftarrow (y, p_1)$
$d_1 \leftarrow (y, u_1)$
$c_1 \leftarrow (y, p_2)$
jump $\leftarrow$ false

Step 3. **If** $c_0 = 0$ **then**
jump $\leftarrow$ true
breakdown at $m_k = 1$ (impossible to compute $\gamma_0 = \beta_0$)
end if
If jump = false **then**
compute β_0, α_0, γ_0 and η_0
If $n_k = 0$ **then**
$s_{k+1} = \eta_0 u_0 + u_1 - \eta_0 \gamma_0 p_1 - \gamma_0 p_2$
else
compute α_0' and η_0'
$s_{k+1} = \eta_0 u_0 + 2u_1 - \eta_0 \gamma_0 p_1 - \gamma_0 p_2 + \eta_0' g_0$
end if
$\sigma_{k+1}^{(1)} \leftarrow (y, s_{k+1})$
compute $r_{k+1} = g_0 - 2\gamma_0 u_1 + \gamma_0^2 p_2$
If $\left|\sigma_{k+1}^{(1)}\right| \leqslant \epsilon$ **or** $|d_0/c_0| \geqslant 1/\epsilon$ **or** ($n_k = 0$ **and** $|c_1/c_0| \geqslant 1/\epsilon$)
or ($n_k \neq 0$ **and** $|d_0/c_0| \geqslant \epsilon$) **then**
jump $\leftarrow$ true
If $\left|\sigma_{k+1}^{(1)}\right| \leqslant \epsilon$ **then**
If $\sigma_{k+1}^{(1)} = 0$ **then**
breakdown at the next step $k+1$
or possible solution obtained
else
near-breakdown at the next step $k+1$
or possible solution obtained
end if
else if $|c_0/d_0| \leqslant \epsilon$ **then**
near-breakdown at $m_k = 1$ (instability in computation of $\gamma_0 = \beta_0$)
else if $n_k = 0$ **and** $|c_1/c_0| \geqslant 1/\epsilon$ **then**
near-breakdown at $m_k = 1$ (instability in computation of $\eta_0 = \alpha_0$)
else
near-breakdown at $m_k = 1$ (instability in computation of $\eta_0' = \alpha_0'$)
end if
end if

If $\|r_{k+1}\| \leqslant \epsilon_2$ **or** jump = false **then**
compute $x_{k+1} = x_k + 2\gamma_0 u_0 - \gamma_0^2 p_1$
If $\|r_{k+1}\| \leqslant \epsilon_2$ **then**
solution obtained.
stop.
end if
end if
end if

Step 4. **While** jump = true **and** nearly singular system (|pivot| $\leqslant \epsilon_1$ or |determinant| $\leqslant \epsilon_1$) **do**
jump $\leftarrow$ false
$m_k \leftarrow m_k + 1$
If $(m_k = n_{\max} - n_k + 1)$ **or** $(m_k > m_{k\max})$ **then**
solution not obtained at $n_{\max}$ or
solution not obtained because the jump is greater than $m_{k\max}$.
stop.
end if
$p_{2m_k-1} \leftarrow Ap_{2m_k-2}$
$p_{2m_k} \leftarrow Ap_{2m_k-1}$
$c_{2m_k-2} \leftarrow (y, p_{2m_k-1})$
$c_{2m_k-1} \leftarrow (y, p_{2m_k})$
If $m_k \leqslant n_k + 1$ **then**
$g_{2m_k-3} \leftarrow Ag_{2m_k-4}$
$g_{2m_k-2} \leftarrow Ag_{2m_k-3}$
$u_{2m_k-2} \leftarrow Au_{2m_k-3}$
$u_{2m_k-1} \leftarrow Au_{2m_k-2}$
else
$u_{n_k+m_k} \leftarrow Au_{n_k+m_k-1}$
end if
If $m_k > n_k$ **then**
$d_{n_k+m_k-1} \leftarrow (y, u_{n_k+m_k-1})$
else
$d_{2m_k-2} \leftarrow (y, u_{2m_k-2})$
$d_{2m_k-1} \leftarrow (y, u_{2m_k-1})$
end if

Step 5. compute $\beta_i, i = 0, \ldots, m_k - 1$
compute $\gamma_i, i = 0, \ldots, m_k - 1$ (coefficients of w_k)
compute $\alpha_i, i = 0, \ldots, m_k - 1$
compute $\eta_i, i = 0, \ldots, m_k - 1$ (coefficients of q_k)
If $m_k \leqslant n_k$ **then**
compute $\alpha_i', i = 0, \ldots, m_k - 1$
compute $\eta_i', i = 0, \ldots, m_k - 1$ (coefficients of t_k)
compute $\beta_i', i = 0, \ldots, m_k - 2$
compute $\gamma_i', i = 0, \ldots, m_k - 2$ (coefficients of v_k)

else if $n_k \neq 0$ **then**
compute $\alpha_i', i = 0, \ldots, n_k - 1$
compute $\eta_i', i = 0, \ldots, n_k - 1$ (coefficients of t_k)
compute $\beta_i', i = 0, \ldots, n_k - 1$
compute $\gamma_i', i = 0, \ldots, n_k - 1$ (coefficients of v_k)
end if

Step 6. **If** non-nearly singular system (|pivots| $> \epsilon_1$ and |determinant| $> \epsilon_1$) **then**

$$\text{compute } s_{k+1} = (I - Av_k(A))q_k(A)s_k - At_k(A)w_k(A)s_k - Aq_k(A)w_k(A)z_k + t_k(A)(I - Av_k(A))r_k$$

$$\sigma_{k+1}^{(m_k)} \leftarrow (y, s_{k+1})$$

$$\text{compute } r_{k+1} = (I - Av_k(A))^2 r_k - 2(I - Av_k(A))Aw_k(A)s_k + A^2 w_k^2(A)z_k$$

If $\left|\sigma_{k+1}^{(m_k)}\right| \leqslant \epsilon$ **then**
jump $\leftarrow$ true
If $\sigma_{k+1}^{(m_k)} = 0$ **then**
breakdown at the next step $k+1$
or possible solution obtained
else
near-breakdown at the next step $k+1$
or possible solution obtained
end if
end if
If $\|r_{k+1}\| \leqslant \epsilon_2$ **or** jump = false **then**

$$\text{compute } x_{k+1} = x_k - (Av_k(A) - 2I)v_k(A)r_k + 2(I - Av_k(A))w_k(A)s_k - Aw_k^2(A)z_k$$

If $\|r_{k+1}\| \leqslant \epsilon_2$ **then**
solution obtained.
stop.
end if
end if
end if
end while

Step 7. $n_{k+1} \leftarrow n_k + m_k$
If $n_{k+1} = n_{\max}$ **then**
solution not obtained at $n_{\max}$.
stop.
end if
compute $z_{k+1} = t_k^2(A)r_k + 2q_k(A)t_k(A)s_k + q_k^2(A)z_k$
compute $a_i^{(k+1)}, i = 0, \ldots, n_{k+1} - 1$ (coefficients of $P_{k+1}^{(1)}$)
compute $b_i^{(k+1)}, i = 1, \ldots, n_{k+1}$ (coefficients of P_{k+1})
$p_0 \leftarrow z_{k+1}$

$g_0 \leftarrow r_{k+1}$
$u_0 \leftarrow s_{k+1}$
end for

Let us now discuss the number of matrix-by-vector products needed at each iteration. An additional matrix-by-vector product is performed at the step $k = 0$ to compute the value $r_0 = b - Ax_0$. Thus we have

- if $m_k = 1$: 3 products;
- if $m_k > 1$ and:
  (1) $m_k \leqslant n_k + 1$: $6m_k - 3$ products,
  (2) $m_k > n_k + 1$: $5m_k + n_k - 2$ products.

5. Description of the code

The BSMRZS algorithm has been coded in the FORTRAN 77 language and tested on a VAX computer and on a PC with the Microsoft FORTRAN Optimizing Compiler. The translation of the code into another language is very easy because we used structured programming and all the variables have been declared in the corresponding specification statements. For all the REAL data (variables, arrays, functions), only the DOUBLE PRECISION type has been considered.

In order to use the BSMRZS algorithm, the subroutine BSMRZS must be called as follows

```
CALL BSMRZS (N, A, AB, Y, X, R, NORM, MAXBCK, MAXN, V, W, U,
             LMAT, RMAT, ETA, ETAP, GAMMA, GAMMAP, D, C,
             P, P1, PWK, IR, NK, EPS, EPS1, EPS2, INIT, IFLAG, IER)
```

and the arguments are defined as follows

N — Input integer value, dimension of the system.

A — Input real matrix of dimension (N,N) containing the matrix A of the system

AB — Input/working real vector of dimension N containing, before the first call, the right hand side of b of the system. The vector is used as a working area in the next calls and then the input values are always destroyed by the subroutine.

Y — Input/output real vector of dimension N containing, before the first call, the auxiliary vector y. If IFLAG = 0 the vector Y is set to $r_0 = b - Ax_0$ by the subroutine during the first call.

X — Input/output real vector of dimension N containing, after the iteration k, the solution x_{k+1}. Before the first call, X must contain x_0.

R — Output real vector of dimension N containing, after the iteration k, the residual vector r_{k+1}. If, at the first call, the norm of the vector R $= r_0$ is less than EPS2 then the vector R contains r_0 in output.

NORM — Output real value containing, after the iteration k, the norm of the residual vector r_{k+1}

MAXBCK — Input integer value. It represents the maximum value allowed for the jump m_k, that is $m_{k\max}$. It must be greater than or equal to 2 and less than or equal to MAXN. It is used to define, in the main program, some dimensions of the working arrays and, in the subroutine, to control their extension.

MAXN — Input integer value. It represents the maximum value allowed for $n_k + m_k$, that is, $n_{\max}$. It is used to define, in the main program, the dimension of the vectors P, P1 and PWK.

V — Working real matrix of dimension (N,0 : 2*MAXBCK).
V(I), I $= 0, \ldots,$ 2*MAXBCK, contains $A^{\mathrm{I}} z_k$.

W — Working real matrix of dimension (N,0 : 2*MAXBCK $-$ 2).
W(I), I $= 0, \ldots,$ 2*MAXBCK $-$ 2, contains $A^{\mathrm{I}} r_k$.

U — Working real matrix of dimension (N,0 : 2*MAXBCK $-$ 1).
U(I), I $= 0, \ldots,$ 2*MAXBCK -1, contains $A^{\mathrm{I}} s_k$.

LMAT — Working real vector of dimension
(8*MAXBCK) if MAXBCK $\leqslant 2$,
(2*MAXBCK $-1)^2$ if MAXBCK > 2.

RMAT — Working real matrix of dimension (2,0 : 2*MAXBCK $-$ 1).

ETA — Working real vector of dimension (0 : MAXBCK) used for storing the η_i.

ETAP — Working real vector of dimension (0 : MAXBCK $-$ 1) used for storing the η'_i.

GAMMA — Working real vector of dimension (0 : MAXBCK) used for storing the γ_i. In output it contains the coefficients of the polynomial $-\xi w(\xi)$

GAMMAP — Working real vector of dimension (0 : MAXBCK $-$ 1) used for storing the γ'_i. In output (if $n_k \neq 0$) it contains the coefficients of the polynomial $1 - \xi v(\xi)$.

D — Working real vector of dimension (0 : 2*MAXBCK $-$ 1) containing the d_i.

C — Working real vector of dimension (0 : 2*MAXBCK $-$ 1) containing the c_i.

P — Input/output real vector of dimension (0 : MAXN) containing, before the iteration k, the coefficients $b_i^{(k)}$ of the polynomial P_k of degree n_k, and after the iteration k, the coefficients $b_i^{(k+1)}$ of the polynomial P_{k+1} of degree $n_{k+1} = n_k + m_k$. The polynomial P_0 is initialized by the subroutine during the first call.

P1 — Input/output real vector of dimension (0 : MAXN) containing, before the iteration k, the coefficients $a_i^{(k)}$ of the polynomial $P_k^{(1)}$ of degree n_k, and after the iteration k, the coefficients $a_i^{(k+1)}$ of the

polynomial $P_{k+1}^{(1)}$ of degree $n_{k+1} = n_k + m_k$. The polynomial $P_0^{(1)}$ is initialized by the subroutine during the first call.

PWK Working real vector of dimension (0 : 2*MAXN + 1).

IR Input rows dimension exactly as specified in the dimension statements in the calling program for the arrays A, V, W and U.

NK Output integer value $n_{k+1} = n_k + m_k$, dimension of the intermediate Krylov subspace.

EPS Input real value used for testing the near breakdown The subroutine does not control whether or not EPS is a negative real number or zero.

EPS1 Input real value used for testing the pivots and the determinant in Gaussian elimination for solving the auxiliary systems. If DABS(X) $\leqslant$ EPS1, then X is considered to be zero. The subroutine does not control whether or not EPS1 is a negative real number or zero.

EPS2 Input real value used for testing the final solution. The final solution is obtained when, at the iteration k, $\|r_{k+1}\| \leqslant$ EPS2. The subroutine does not control whether or not EPS2 is a negative real number or zero.

INIT Input/output integer to be set to zero before the first call of the subroutine. Its value is changed to 1 by the subroutine during the first call. For a new application of the method, INIT must be set again to zero.

IFLAG Input integer. If IFLAG = 0 then the auxiliary vector y is chosen to coincide with the vector $z_0 = r_0$. If IFLAG $\neq$ 0 then the user must define y in the main program before the first call.

IER Output index warning/error. This index can take the following values

IER = 100 Call of the subroutine with a non zero IER value.

IER = 200 The norm of the residual vector is less or equal to EPS2. The exact solution has been obtained.

IER = 300 Solution not obtained after reaching the value NK + MK = MAXN.

IER = 400 Solution not obtained after reaching the dimension n of the system, due to the precision of the computer. The computation continues until NK + MK = MAXN at most.

IER = 500 The value of MAXBCK is greater than MAXN or less than 2.

IER = 600 The value of m_k exceeds the value of MAXBCK.

IER = 700 Jump for breakdown. Case $C(0) = (y, Az_k) = 0$.

IER = 800 Jump for breakdown. Case $\sigma_{k+1}^{(m_k)} = 0$.

IER = 900 Jump for near-breakdown. Case $\left|\sigma_{k+1}^{(m_k)}\right| \leqslant$ EPS.

IER = 1000 Jump for near-breakdown. Case $|D(0)/C(0)| \geqslant 1/\text{EPS}$.

IER = 1100 Jump for near-breakdown. Case $n_k = 0, |C(1)/C(0)| \geqslant 1/EPS$.

IER = 1200 Jump for near-breakdown. Case $n_k \neq 0, |D(0)/C(0)| \leqslant EPS$.

In addition, there are many functions and subroutines required which are called directly or indirectly by the subroutine BSMRZS. Some of them have been explicitly written for this algorithm and some others coincide with the modules used for the BSMRZ algorithm [10,11]. All these modules can be found in the *netlib* library *numeralgo*. They can be obtained, together with all the modules for using MRZ, BMRZ, SMRZ, BSMRZ algorithms [10,11], by sending the command lines

send na1 **from** numeralgo
send na5 **from** numeralgo

at the Internet address

netlib@research.att.com

The functions required are the following:

SUNORM Computes the maximum norm of a vector.

FH Computes the coefficient f_{ij} (or the h_{ij}), needed in the system giving the α_i, α_i', β_i and β_i', as the linear combination of the d_i (or the c_i) with the coefficients of the polynomial $P_k^{(1)}(\xi)$.

GAMETA Computes the value of the coefficient γ_i or γ_i' or η_i or η_i' (of the polynomials $w_k(\xi)$ or $v_k(\xi)$ or $q_k(\xi)$ or $t_k(\xi)$) as a linear combination of the β_i or the β_i' or the α_i or the α_i', with the coefficients of the polynomial $P_k^{(1)}(\xi)$.

PINNER Computes the inner product of two vectors (given in *na1*).

RXSUMM Computes a component of the vector r_{k+1} or of the vector $x_{k+1} - x_k$ as a linear combination of the coefficients of the polynomials $v_k(\xi)(\xi v_k(\xi) - 2)$, $(1 - \xi v_k(\xi))^2$, $(1 - \xi v_k(\xi))w_k(\xi)$ and $w_k^2(\xi)$ with the vectors p_i, g_i and u_i.

SSUMM Computes a component of the vector s_{k+1} as a linear combination of the coefficients of the polynomials $t_k(\xi)(1 - \xi v_k(\xi))$, $(1 - \xi v_k(\xi))q_k(\xi)$, $t_k(\xi)w_k(\xi)$ and $w_k(\xi)q_k(\xi)$ with the vectors p_i, g_i and u_i.

ZSUMM Computes a component of the vector z_{k+1} as a linear combination of the coefficients of the polynomials $t_k^2(\xi)$, $t_k(\xi)q_k(\xi)$ and $q_k^2(\xi)$ with the vectors p_i, g_i and u_i.

The subroutines required are the following:

CHKSIG Controls the conditions that could produce a breakdown, a near-breakdown or if the solution is about to be obtained.

COMPO1 Computes the coefficients of the polynomials $t_k^2(\xi)$, $t_k(\xi)q_k(\xi)$ and $q_k^2(\xi)$.

COMPO2 Computes the coefficients of the polynomials $t_k(\xi)(1-\xi v_k(\xi))$, $(1-\xi v_k(\xi))q_k(\xi)$, $t_k(\xi)w_k(\xi)$ and $w_k(\xi)q_k(\xi)$.
COMPOL Computes the coefficients of the polynomials $v_k(\xi)(\xi v_k(\xi)-2)$, $(1-\xi v_k(\xi))^2$, $(1-\xi v_k(\xi))w_k(\xi)$ and $w_k^2(\xi)$.
COMPP1 Computes the coefficients of the polynomials $P_{k+1}(\xi)$ and $P_{k+1}^{(1)}(\xi)$.
CRESOL Stores, in separate vectors, the solutions β, β', α and α' of the auxiliary system, and computes the γ_i, γ_i', η_i and the η_i'.
GSOLVD Computes, by Gaussian elimination, the solution of a linear system with two right hand sides.
MATVEC Computes the product between a matrix and a vector (given in *na1*).
POLMUL Computes the product between two polynomials.
SOLSHF Computes the solutions β, β', α and α' of the auxiliary system.
STOMHF Stores the matrix of the auxiliary system.
STORHF Stores the right hand sides of the auxiliary system.

Let us mention that some of our subroutines could be replaced by standard routines, such as the BLAS routines, but ours were specially written for our purpose and, thus, they are better adapted.

6. Using the program

For using the subroutine BSMRZS, a main program must be written by the user (five examples of such a main program are provided with the code). The subroutine must be called in an iterative way, that is, the user creates a loop (for instance K = 0, ..., NBC, with NBC greater than or equal to MAXN − 1), containing the call of the subroutine BSMRZS. At each iteration, a control on the output value IER allows the user to stop (because the solution has been obtained or due to a warning/error code) or to continue until the end of the loop.

Before the start of the loop, that is, before the first call of the subroutine, the user must initialize all the following input arguments:

N Dimension n of the system.
A Matrix A of the system.
AB Right hand side b of the system.
Y This argument denotes the auxiliary vector y. The user can choose to define its components in the main program (in that case he must set also the argument IFLAG to a non zero value) or let the subroutine sets its value, during the first call, to r_0 (in that case the user must set the argument IFLAG to zero).
X The initial vector x_0.
MAXBCK The value of $m_{k\max}$ such that $2 \leqslant m_{k\max} \leqslant n_{k\max}$.
MAXN The value of $n_{\max}$.
IR The value of the first dimension for the arrays A, V, W and U, as specified in the dimension statements of the main program.
EPS The value of ϵ for testing the near-breakdown.

EPS1 The value of ϵ_1 for testing the pivots and the determinant.
EPS2 The value of ϵ_2 for testing the norm of the residual vector.
INIT This value must be set to zero.
IFLAG This value must be set to zero or to any other integer value according to the user's choice (see argument Y).

All the other initializations of the algorithm are made by the subroutine during the first call.

In a pseudocode form the scheme of the main program can be synthesized as follows:

Step 1. Specifications for variables and constants
list of integer variables and parameter constants
list of double precision variables and parameter constants
Step 2. Specifications for parameter constants
N, IR, MAXBCK, EPS, ...
Step 3. Specifications for vector and matrices
list and dimensions of integer arrays
list and dimensions of double precision arrays
Step 4. Initializations
INIT = 0.
other initializations required
Step 5. For K = 0 **to** NBC
CALL BSMRZS
If IER $\neq$ 0 **then**
If IER = 200 **then**
solution obtained.
stop.
else if IER = 400 **then**
solution not obtained at *n*.
continue until MAXN at most.
IER = 0
else if IER $\geqslant$ 700 **then**
jump for breakdown or near-breakdown.
IER = 0
else
stop due to error IER.
end if
end if
end for

7. Numerical results

Let us recall that, in our program, we have to choose the values of

- ϵ for testing the quantities involved in the jumps,

- ϵ_1 for testing the pivots and the determinant in Gaussian elimination and jumping,
- ϵ_2 for testing if $\|r_k\| \leqslant \epsilon_2$ and stopping.

There is an important point which must be noticed. Since after testing if $|\sigma_{k+1}^{(m_k)}| \leqslant \epsilon$, we check whether or not the norm of the current residual is smaller than ϵ_2, then both tests are correlated and a value of ϵ_2 too small could produce a jump. This point will be illustrated in example 2.

It seems quite difficult to give advice about the choices of ϵ, ϵ_1 and ϵ_2 which are valid in all circumstances. ϵ_2 is the stopping criterion for the algorithm and thus it is really the user's choice. However, due to the discrepancy which can occur between the residuals computed iteratively and the actual residuals, some unsuspected problems can arise. Examples of this situation will be given below. The quantities ϵ and ϵ_1 are involved in the jumps. If these quantities are too small, then, in general, no jump will occur (except in the case of an exact division by zero) and no improvement will be brought by our algorithm. For a computer working with 16 decimal digits, it seems reasonable to take ϵ and ϵ_1 between 10^{-14} and 10^{-8}. For bigger values of ϵ and ϵ_1, the jumps have a greater length, thus bringing some other numerical problems from the solution of the linear systems giving the coefficients of the recurrence relationships. Moreover, in that case, a storage problem can also occur due to the dimension of these systems.

Example 1

Let us first consider the system

$$\begin{pmatrix} 0 & 1 & & & & & \\ & 0 & 1 & & & & \\ & & \ddots & \ddots & & & \\ & & & \ddots & 1 & & \\ & & & & 0 & 1 & \\ a & & & & & 0 & \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \\ a \end{pmatrix}.$$

For $n = 40$, $a = 0.95$, $x_0 = 0$, $y = r_0$ $\epsilon = \epsilon_1 = 10^{-60}$ and $\epsilon_2 = 10^{-4}$, we have no jump and we obtain $\|r_{40}\| = 0.44 \times 10^{12}$.

When $\epsilon = \epsilon_1 = 10^{-8}$ we have a jump from $n_1 = 1$ to $n_2 = 38$ and we obtain $n_4 = 40$ and $\|r_4\| = 0.39 \times 10^{-10}$. The same results are obtained with ϵ and ϵ_1 varying between 10^{-5} and 10^{-14}. For $\epsilon = 10^{-4}$ and $\epsilon_1 = 10^{-6}$, we have a jump at the first step of the algorithm until $n_1 = 38$ and we obtain $n_3 = 40$ and $\|r_3\| = 0.44 \times 10^{-13}$. When $\epsilon = 10^{-4}$ and $\epsilon_1 = 10^{-8}$, the same results are obtained. For $\epsilon = 10^{-4}$ and $\epsilon_1 = 10^{-5}$, we still have a jump at the first step, but until $n_1 = 39$ and we obtain $n_2 = 40$ and $\|r_2\| = 0.23 \times 10^{-14}$.

All the jumps in this example are due to IER = 900.

Example 2

We consider now the $n \times n$ block diagonal matrix

$$A = \begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_{n/2} \end{pmatrix},$$

with

$$M_j = \begin{pmatrix} 1 & j-1+a \\ 0 & -1 \end{pmatrix} \qquad \text{for } j = 1, \ldots, n/2.$$

The case $a = 0$ corresponds to the example $B_{\pm 1}$ of [33].
The solution of this system is given by

$$x_{2j-1} = b_{2j-1} + (j-1+a)b_{2j},$$
$$x_{2j} = -b_{2j},$$

where the b_i's are the components of the right hand side b.

We shall take $b = (5, -3, 4, -4, 0, \ldots, 0)^T$, $x_0 = 0$, $y = r_0$, $n = 40$. With these values and $a = 0$, a breakdown occurs at the first iteration and the exact solution is obtained for $n_1 = 2$ after jumping, see [15] (in this paper, read -3 for the second component of the vector b).

For $a = 10^{-6}$, $\epsilon = \epsilon_1 = 10^{-60}$, the norm of the residuals oscillates and there is a jump (due to IER = 1200) from $n_{15} = 15$ to $n_{16} = 17$ and we obtain $\|r_{16}\| = 0.57 \times 10^{-14}$. However, the results are false, a phenomenon due to the discrepancy between the residual computed iteratively and the actual residual, that will be discussed in the next example. For $\epsilon = 10^{-6}$ and ϵ_1 arbitrarily chosen, we have a jump of length 2 at the first iteration and we obtain a residual which is exactly zero.

Let us now illustrate the point mentioned at the beginning of this section about the correlation of ϵ and ϵ_2.

For $a = 10^{-3}$, $\epsilon = \epsilon_1 = 10^{-60}$, no jump occurs and the norm of the residuals oscillates and grows to $\|r_{40}\| = 0.15 \times 10^{14}$.

When $\epsilon = 10^{-6}$, $\epsilon_1 = 10^{-12}$ and $\epsilon_2 = 10^{-8}$, we have $n_1 = 1$, $\|r_1\| = 0.23 \times 10^{8}$, $n_2 = 2$, $\|r_2\| = 0.37 \times 10^{-8}$ and the program stops. The same results are obtained for $\epsilon_2 = 10^{-6}$.

When $\epsilon = 10^{-6}$, $\epsilon_1 = 10^{-12}$ and $\epsilon_2 = 10^{-9}$, we have again $n_1 = 1$, $\|r_1\| = 0.23 \times 10^{8}$ but $n_2 = 3$ (due to IER = 900) and $\|r_2\| = 0.74 \times 10^{-8}$. Then the norm of the residuals oscillates and we have jumps (due to IER = 1200) from $n_5 = 6$ to $n_6 = 8$ and from $n_6 = 8$ to $n_7 = 10$ where $\|r_7\| = 0.13 \times 10^{-22}$. However, the solution has only 7 exact decimal digits due to the same discrepancy phenomenon as above. The same results are obtained for $\epsilon_2 = 10^{-12}$.

In this example, the exact solution is obtained if a jump of length 2 occurs at the first step, which is the case if we take $\epsilon \geqslant a$ and ϵ_1 is arbitrarily chosen.

Example 3

We consider the same example as above but with

$$M_j = \begin{pmatrix} 0 & 1+ja \\ -1 & 0 \end{pmatrix} \qquad \text{for } j = 1, \ldots, n/2.$$

The case $a = 0$ corresponds to the example S of [33] where a breakdown occurs at the first iteration of the CGS.

For $n = 40$, $a = 10^{-10}$, $x_0 = 0$, $b = (1, 2, \ldots, n)^T$, $y = r_0$ and $\epsilon = \epsilon_1 = 10^{-60}$ we have a jump from $n_9 = 9$ to $n_{10} = 11$ and another one from $n_{10} = 11$ to $n_{11} = 13$ (both due to IER = 1200) and we obtain $\| r_{11} \| = 0.21 \times 10^{-7}$. However, the solution is completely false and has no exact digit. This is due to the fact, mentioned in the preceding example and in [43] (see also [33]), that the residual estimates r_k computed recursively by the algorithm can differ significantly from the actual residual $b - Ax_k$. In our case, the actual residual gives $\| r_{11} \| = 8231$. Of course, it is possible to use the actual residuals in the algorithm instead of their estimates but their computation needs one more matrix-by-vector multiplication at each iteration.

For $a = 10^{-9}$, $\epsilon = 10^{-8}$ and $\epsilon_2 = 10^{-13}$, we have a jump from the beginning and we obtain $n_1 = 2$ and $\| r_1 \| = 0.71 \times 10^{-14}$. This result is independent of the value of ϵ_1.

Example 4

Let us now consider the following system

$$\begin{pmatrix} a & 1 & & & & & & \\ -1 & a & 1 & & & & & \\ & -1 & a & \ddots & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & -1 & a & 1 & \\ & & & & & -1 & a & \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} a+1 \\ a \\ a \\ \vdots \\ \vdots \\ a \\ a-1 \end{pmatrix}.$$

This system was studied by Brown [16], and, when $a = 0$, a breakdown occurs every odd step in Lanczos' method. For small values of a, the convergence has a sawtooth behavior. For $n = 200$, $a = 10^{-8}$, $x_0 = 0$, $y = r_0$, $\epsilon = 10^{-200}$ and ϵ_1 arbitrary, there is no jump but an overflow arises at $k = 84$. When $\epsilon = 10^{-5}$ and ϵ_1 is between 10^{-15} and 10^{-10}, we have jumps of length 2 from the beginning (due to IER = 1000) and we obtain $n_{100} = 200$ and $\| r_{100} \| = 0.28 \times 10^{-9}$. The intermediate residuals have a norm which is always almost equal to 1. These results are presented in figure 1.

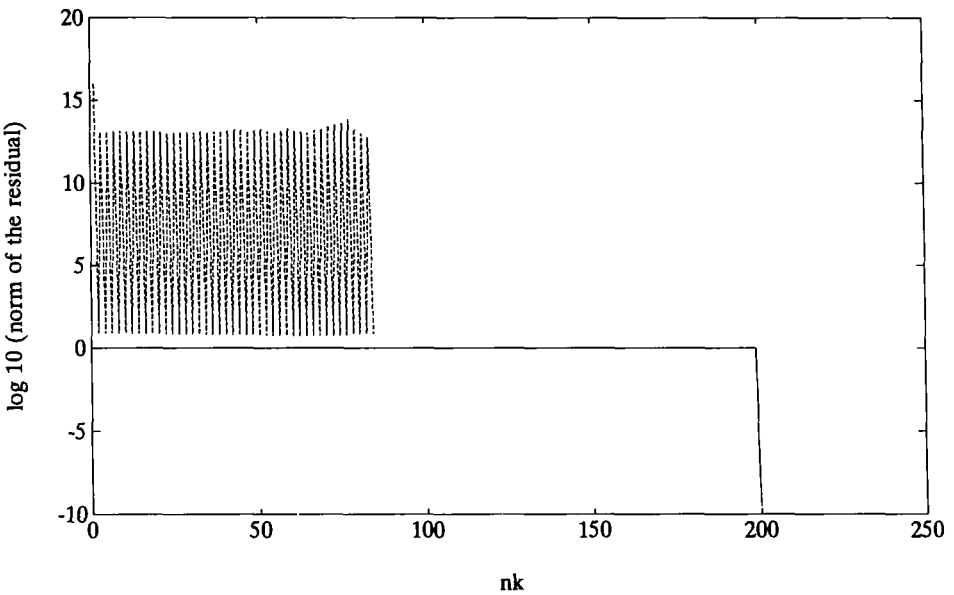

Figure 1.

Example 5

Let us now consider the system

$$\begin{pmatrix} 2 & 1 & & & & & & \\ 0 & 2 & 1 & & & & & \\ 1 & 0 & 2 & \ddots & & & & \\ & 1 & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \\ & & & 1 & 0 & 2 & 1 & \\ & & & & 1 & 0 & 2 & \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 4 \\ \vdots \\ \vdots \\ 4 \\ 3 \end{pmatrix}.$$

This system was given by Gutknecht [27] and, for $x_0 = 0$ and $y = (0, 0, 0, -1, 1, \ldots, -(-1)^{n-1}, 0)^T$, a breakdown occurs at the first iteration when n is even since $(y, r_0) = (y, Ar_0) = 0$. When n is odd, $\sigma_2^{(1)} = 0$ at the first iteration and a breakdown also occurs but for a different reason.

For $n = 40$, $\epsilon = \epsilon_1 = 10^{-60}$, we have only one jump of length 2 at the first iteration (due to IER = 700) and we obtain $n_{32} = 33$ and $\| r_{32} \| = 0.10 \times 10^{-14}$.

When $\epsilon = \epsilon_1 = 10^{-7}$, we have a jump of length 2 at the first iteration (due to IER = 700) and jumps (due to IER = 1200) of length 2 from $n_{18} = 19$ and, when $n_{25} = 33$, we obtain $\| r_{25} \| = 0.10 \times 10^{-14}$.

Quite similar results are obtained for $n = 41$ but the jumps at the first iteration are due to IER = 800.

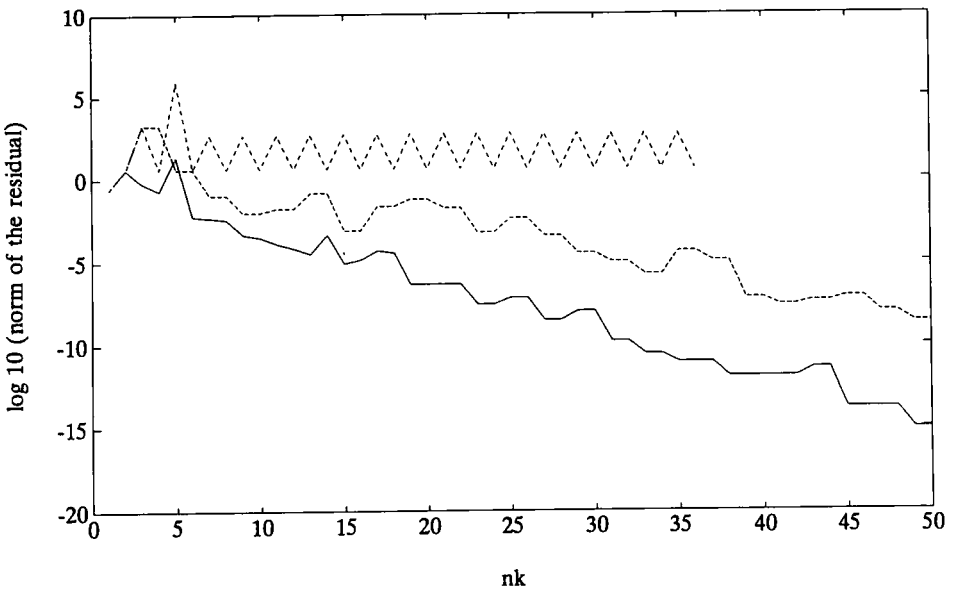

Figure 2.

Let us now replace the last component of the vector y (which was 0) by 10^{-8}.

For $n = 400$, the results can be seen in figure 2. For $\epsilon = \epsilon_1 = 10^{-200}$, there is no jump, the norm of the residual oscillates between 10^1 and 10^3, starting from $n_6 = 6$, and an overflow occurs at $n_{37} = 37$ (highest curve in figure 2).

When $\epsilon = \epsilon_1 = 10^{-10}$ we have jumps (due to IER = 1200) of length 2 from $n_3 = 3$ and, when $n_{26} = 49$, we obtain $\| r_{26} \| = 0.22 \times 10^{-8}$ (middle curve in figure 2).

When $\epsilon = \epsilon_1 = 10^{-7}$, we have a jump (due to IER = 1100) to $n_1 = 2$ at the first iteration. Then we have jumps (due to IER = 1200) of length 2 from $n_{18} = 19$ to $n_{26} = 35$. Then, we obtain various jumps due to IER = 900 and, when $n_{30} = 49$, we obtain $\| r_{30} \| = 0.85 \times 10^{-15}$ (lower curve in figure 2).

For $n = 401$, the results are given in figure 3.

For $\epsilon = \epsilon_1 = 10^{-200}$, there is no jump and an overflow occurs at $n_{108} = 108$ (highest curve in figure 3).

When $\epsilon = \epsilon_1 = 10^{-10}$ we have jumps (due to IER = 1200) of length 2 from $n_7 = 7$ and, when $n_{33} = 59$, we obtain $\| r_{33} \| = 0.10 \times 10^{-9}$ (middle curve in figure 3).

When $\epsilon = \epsilon_1 = 10^{-7}$, we have a jump (due to IER = 900) to $n_1 = 2$ at the first iteration. Then we have jumps (due to IER = 1200) of length 2 from $n_{19} = 20$ to $n_{27} = 36$. Then, we obtain various jumps due to IER = 900 and 1200 and, when $n_{35} = 60$, we obtain $\| r_{35} \| = 0.28 \times 10^{-13}$ (lower curve in figure 3).

For $n = 40$, the preceding results were obtained on a PC with a mathematical coprocessor. They have also been checked on a VAX using the G_FLOAT representation and the normal one. For some examples we observed quite a different behavior, due to the fact that, with G_FLOAT representation, the VAX uses one less decimal digit. The examples with $n > 40$ were only tested on the VAX.

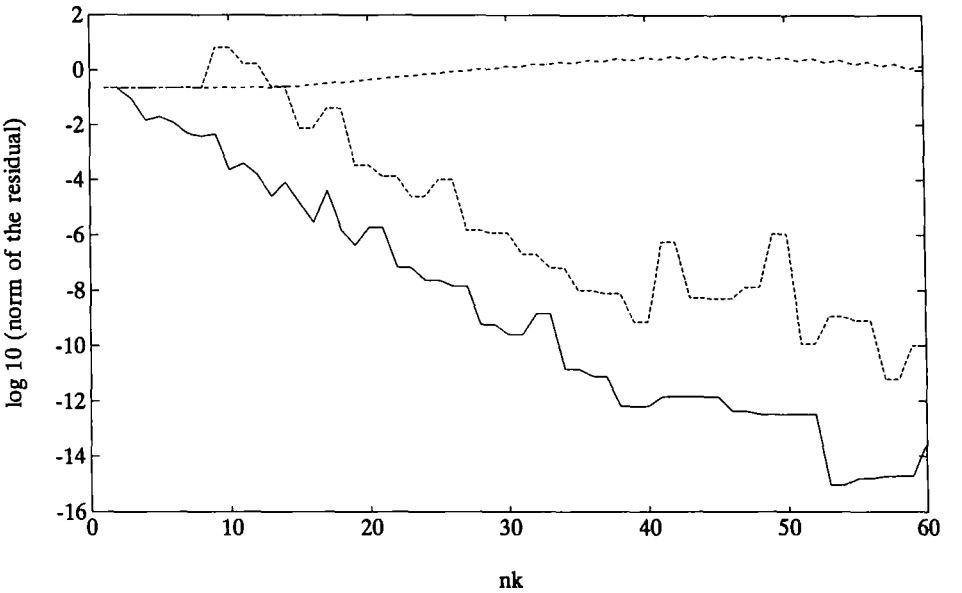

Figure 3.

8. Discussion

Our treatment of near-breakdown in the CGS algorithm is only able to try avoiding some of the instabilities due to the computer's arithmetic. In particular, we tried to avoid those instabilities coming from a denominator close to zero in one of the coefficients of one of the recurrence relations used in the algorithm.

Our program was tested by using the software CADNA [19,21] based on the CESTAC method [18,39] which simulates a stochastic arithmetic as described in [19 – 21] and it was found to be quite stable. Moreover, an important and difficult problem which remains open in our program is the choice of the value of ϵ, ϵ_1 and ϵ_2 as already noticed in [11]. Such tests are eliminated by CADNA and replaced by a control on the number of significant digits of each intermediate computation. Thus the numerical stability of each operation in the algorithm could be checked and validated. The implementation of our software in stochastic arithmetic using CADNA is under consideration.

We do not claim that our software is of lasting value and that it is able to handle successfully all the possible causes for a near-breakdown. It is certainly possible to improve it and to take into account other numerical instabilities. Further research is still necessary to fully understand the numerical behavior of the various algorithms and for healing all the numerical deficiencies of the CGS. For example, as pointed out by Van der Vorst [37], the CGS can be unstable even if the biconjugate gradient is completely stable. However, we decided to put our software into the public domain, even if it is imperfect, for helping other researchers to take up the problem without being obliged to code again the algorithm.

Acknowledgements

We would like to thank Hassane Sadok for several interesting discussions. The understanding and the use of the software CADNA were made possible by Jean-Marie Chesneaux and Jean Vignes. They are thanked for their very effective help. We are also indebted to Tony Chan and Tedd Szeto for their comments which helped us to improve the presentation of the paper.

References

[1] R.E. Bank and T.F. Chan, A composite step bi-conjugate gradient algorithm for solving non-symmetric systems, Numer. Algorithms, this issue.

[2] R.E. Bank and T.F. Chan, An analysis of the composite step bi-conjugate gradient algorithm for solving nonsymmetric systems, Numer. Math. 66 (1993) 295–319.

[3] C. Brezinski, *Padé-Type Approximation and General Orthogonal Polynomials*, ISNM vol. 50 (Birkhäuser, Basel, 1980).

[4] C. Brezinksi and M. Redivo-Zaglia, *Extrapolation Methods. Theory and Practice* (North-Holland, Amsterdam, 1991).

[5] C. Brezinski and M. Redivo-Zaglia, A new presentation of orthogonal polynomials with applications to their computation. Numer. Algorithms 1 (1991) 207–221.

[6] C. Brezinski and M. Redivo-Zaglia, Treatment of near-breakdown in the CGS algorithm, Publication ANO 257, Université des Sciences et Technologies de Lille (November 1991).

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

[8] C. Brezinski and M. Redivo-Zaglia, Breakdowns in the computation of orthogonal polynomials, in: *Nonlinear Numerical Methods and Rational Approximation*, ed. A. Cuyt (Kluwer, Dordrecht, 1994) pp. 49–59.

[9] C. Brezinski and M. Redivo-Zaglia, Look-ahead in Bi-CGSTAB and other methods for linear systems, to appear.

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

[11] C. Brezinski, M. Redivo-Zaglia and H. Sadok, Addendum to "Avoiding breakdown and near-breakdown in Lanczos type algorithms", Numer. Algorithms 2 (1992) 133–136.

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

[13] C. Brezinski, M. Redivo-Zaglia and H. Sadok, Breakdowns in the implementation of the Lánczos method for solving linear systems, Comp. Math. Appl., to appear.

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

[15] C. Brezinski and H. Sadok, Avoiding breakdown in the CGS algorithm, Numer. Algorithms 1 (1991) 199–206.

[16] P.N. Brown, A theoretical comparison of the Arnoldi and GMRES algorithms, SIAM J. Sci. Stat. Comp. 12 (1991) 58–78.

[17] T.F. Chan and T. Szeto, A composite step conjugate gradients squared algorithm for solving nonsymmetric linear systems, Numer. Algorithms, this issue.

[18] J.M. Chesneaux, Study of the computing accuracy by using probabilistic approach, in: *Contribution to Computer Arithmetic and Self-Validating Numerical Methods*, ed. C. Ulrich (Baltzer, Basel, 1990) pp. 19–30.

[19] J.M. Chesneaux, Stochastic arithmetic properties, in: *Computational and Applied Mathematics, I*, eds. C. Brezinski and U. Kulisch (North-Holland, Amsterdam, 1992) pp. 81–91.

[20] J.M. Chesneaux and J. Vignes, Les fondements de l'arithmétique stochastique, C.R. Acad. Sci. Paris, I, 315 (1992) 1435–1440.
[21] J.M. Chesneaux and J. Vignes, L'algorithme de Gauss en arithmétique stochastique, C. R. Acad. Sci. Paris, II, 316 (1993) 171–176.
[22] A. Draux, *Polynômes Orthogonaux Formels. Applications*, LNM 974 (Springer, Berlin, 1983).
[23] R. Fletcher, Conjugate gradient methods for indefinite systems, in: *Numerical Analysis*, ed. G.A. Watson, LNM 506 (Springer, Berlin, 1976) pp. 73–89.
[24] R.W. Freund, M.H. Gutknecht and N.M. Nachtigal, An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, SIAM J. Sci. Comp. 14 (1993) 137–158.
[25] M.H. Gutknecht, A completed theory of the unsymmetric Lanczos process and related algorithms, Part I, SIAM J. Matrix. Anal. Appl. 13 (1992) 594–639.
[26] M.H. Gutknecht, The unsymmetric Lanczos algorithms and their relations to Padé approximation, continued fractions and the qd algorithm, to appear.
[27] M.H. Gutknecht, Variants of BICGSTAB for matrices with complex spectrum, SIAM J. Sci. Comp. 14 (1993) 1020–1033.
[28] K.C. Jea and D.M. Young, On the simplification of generalized conjugate gradient methods for nonsymmetrizable linear systems, Lin. Alg. Appl. 52/53 (1983) 399–417.
[29] W. Joubert, Generalized conjugate gradient and Lanczos methods for the solution of nonsymmetric systems of linear equations, Ph.D. Thesis, University of Texas at Austin, Austin (1990).
[30] T.W. Körner, *Fourier Analysis* (Cambridge University Press, Cambridge, 1988).
[31] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Natl. Bur. Stand. 45 (1950) 255–282.
[32] C. Lanczos, Solution of systems of linear equations by minimized iterations, J. Res. Natl. Bur. Stand. 49 (1952) 33–53.
[33] N.M. Nachtigal, S.C. Reddy and L.N. Trefethen, How fast are nonsymmetric matrix iterations?, SIAM J. Matrix Anal. Appl. 13 (1992) 778–795.
[34] B.N. Parlett, D.R. Taylor and Z.A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, Math. Comp. 44 (1985) 105–124.
[35] W. Schönauer, *Scientific Computing on Vector Computers* (North-Holland, Amsterdam, 1987).
[36] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Stat. Comp. 10 (1989) 36–52.
[37] H.A. Van der Vorst, The convergence behavior of preconditioned CG and CG-S, in: *Preconditioned Conjugate Gradient Methods*, eds. O. Axelsson and L.Yu. Kolotilina, LNM 1457 (Springer, Berlin, 1990) pp. 126–136.
[38] 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. Stat. Comp. 13 (1992) 631–644.
[39] J. Vignes, Review of stochastic approach to round-off error analysis and its applications, Math. Comp. Simul. 30 (1988) 481–491.
[40] 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.
[41] R. Weiss, Convergence behavior of generalized conjugate gradient methods, Thesis, University of Karlsruhe (1990).
[42] D.M. Young and K.C. Jea, Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods, Lin. Alg. Appl. 34 (1980) 159–194.
[43] L. Zhou and H.F. Walker, Residual smoothing techniques for iterative methods, SIAM J. Sci. Stat. Comp., to appear.
[44] R. Zippel, *Effective Polynomial Computation* (Kluwer, Dordrecht, 1993).