IJST (2013) 37A3 (Special issue-Mathematics): 349-358
Iranian Journal of Science & Technology

http://ijsts.shirazu.ac.ir

A preemptive restarting approach to beating the inherent instability of Lanczos-type algorithms

M. Farooq[1]* and A. Salhi[2]

[1]*Department of Mathematics, University of Peshawar, 25120, Khyber Pakhtunkhwa, Pakistan*
[2]*Department of Mathematical Sciences, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK*
E-mails: mfarooq@upesh.edu.pk & as@essex.ac.uk

Abstract

Lanczos-type algorithms are well known for their inherent instability. They typically breakdown occurs when relevant orthogonal polynomials do not exist. Current approaches to curing breakdown rely on jumping over the non-existent polynomials to resume computation. This may have to be used many times during the solution process. We suggest an alternative to jumping, which consists of restarting the algorithms that fail. Three different strategies can be taken: (ST1) Restarting following breakdown of the algorithm in use; (ST2) pre-emptive restarting after a fixed number of iterations; (ST3) restarting when near breakdown is detected through monitoring. We describe a restarting framework with a generic algorithm that invokes one or the other of the three strategies suggested. Four of the most prominent recently developed Lanczos-type algorithms namely, $A_4, A_{12}, A_5/B_{10}$ and A_8/B_{10} will be presented and then deployed in the restarting framework. However, we will only report on results obtained with strategy ST2 as it is the only viable one at the moment.

Keywords: Lanczos algorithm; Systems of Linear Equations; Formal Orthogonal Polynomials, Restarting, Switching, Breakdown

1. Introduction

The Lanczos-type methods for solving Systems of Linear Equations (SLEs) are based on the theory of Formal Orthogonal Polynomials (FOPs). All such methods are implemented via some recurrence relationships between polynomials $P_k(x)$ represented by A_i or between two adjacent families of orthogonal polynomials $P_k(x)$ and $P_k^{(1)}(x)$ represented by A_i and B_j as described in [1, 2]. The coefficients of the various recurrence relationships for orthogonal polynomials are given as ratios of scalar products. When a scalar product in a denominator vanishes, then a breakdown occurs in the algorithm and the process normally has to be stopped. Equivalently, the breakdown is due to the non-existence of some orthogonal polynomials. So quite an important problem is how to be able to continue the solution process in such a situation and arrive at a usable result. Several procedures for that purpose have appeared in the literature in the last few years. It has been proved that it is possible to jump over non-existing polynomials [3], and breakdown-free algorithms were thus obtained. The first attempt in this regard was the look-ahead Lanczos algorithm [4]. Other procedures for avoiding breakdown are also proposed in [5-12]. The procedure for jumping over non-existing orthogonal polynomials can be described as follows:

1. Recognize the occurrence of breakdown;
2. Determine the degree of the next existing (regular) orthogonal polynomial;
3. Find the recurrence relationship to compute it, i.e. jump over the non-existing orthogonal polynomials.

This idea of the degree of the next regular polynomial or the length of the jump is explained in [13-19]. Note that it is not our purpose here to compare the advantages of these various methods.

2. The Lanczos approach

We consider a linear system of equations,

$$Ax = b, \tag{1}$$

where $A \in R^{n\times n}$, $b \in R^{n\times n}$ and $x \in R^{n\times n}$.

Let x_0 and y be two arbitrary vectors in R^n such that $y \neq 0$. The Lanczos method [20] consists of constructing a sequence of vectors $x_k \in R^n$ defined as follows [21],

*Corresponding author

Received: 23 July 2012 / Accepted: 1 December 2012

$$x_k - x_0 \in K_k(A, \mathbf{r}_0) = span(\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0), \qquad (2)$$

$$\mathbf{r}_k = (\mathbf{b} - A\mathbf{x}_k) \perp K_k(A^T, \mathbf{y}) = span(\mathbf{y}, A^T\mathbf{y}, \ldots (A^T)^{k-1}\mathbf{y}), \quad (3)$$

where A^T denotes the transpose of A.
Equation (2) leads to,

$$\mathbf{x}_k - \mathbf{x}_0 = -\alpha_1\mathbf{r}_0 - \alpha_2 A\mathbf{r}_0 - \cdots - \alpha_k A^{k-1}\mathbf{r}_0. \qquad (4)$$

Multiplying both sides by A and adding and subtracting **b** on the left hand side gives

$$\mathbf{r}_k = \mathbf{r}_0 + \alpha_1 A\mathbf{r}_0 + \alpha_2 A^2\mathbf{r}_0 + \cdots + \alpha_k A^k\mathbf{r}_0. \qquad (5)$$

From (3), the orthogonality condition gives

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

and, by (5), we obtain the following system of linear equations

$$\begin{cases} \alpha_1(y, Ar_0) + \ldots + \alpha_k(y, A^k r_0) = -(y, r_0) \\ \alpha_1((A^T)^{k-1}y, Ar_0) + \ldots + \alpha_k((A^T)^{k-1}y, A^k r_0) = ((A^T)^{k-1}y, r_0) \end{cases} \qquad (6)$$

If the determinant of the above system is different from zero then its solution exists and allows us to obtain x_k and r_k. Obviously, in practice, solving the above system directly for the increasing value of k is not feasible. We shall now see how to solve this system for increasing values of k recursively. If we set

$$P_k(x) = 1 + \alpha_1 x + \ldots + \alpha_k x^k, \qquad (7)$$

then we can write from (5)

$$r_k = P_k(A)r_0. \qquad (8)$$

The polynomials P_k are commonly known as the residual polynomials.
Another interpretation of the P_k can be found in [22]. Moreover, if we set

$$c_i = (A^{T^i} y, r_0) = (y, A^i r_0) \qquad i = 0, 1, \ldots$$

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

$$c(x^i) = c_i, \; i = 0, 1, \ldots \qquad (9)$$

c is completely determined by the sequence $\{c_k\}$ and c_k is said to be the moment of order k, [23]. Now, the system (6) can be written as

$$c(x^i P_k(x)) = 0, \text{ for } i = 0, \ldots, k-1. \qquad (10)$$

These conditions show that P_k is the polynomial of degree at most k, normalized by the condition $P_k(0) = 1$, belonging to FOP's with respect to the linear functional c [23], which is briefly reviewed in Section 3. For more details see [24]. Such a family is called family of orthogonal polynomials with respect to the functional c and the preceding relations are called the orthogonality relations.
Since the constant term of P_k in (7) is 1, it can be written as

$$P_k(x) = 1 + xR_{k-1}(x)$$

where $R_{k-1}(x) = \alpha_1 + \alpha_2 x + \ldots + \alpha_k x^{k-1}$. Replacing x by A in the expression of P_k and multiplying both sides by r_0 and using (8), we get

$$r_k = r_0 + AR_{k-1}(A)r_0,$$

which can be written as

$$b - Ax_k = b - Ax_0 + AR_{k-1}(A)r_0,$$

or

$$-Ax_k = -Ax_0 + AR_{k-1}(A)r_0.$$

Multiplying both sides by $-A^{-1}$, we get

$$x_k = x_0 - R_{k-1}(A)r_0,$$

which shows that x_k can be computed from r_k without inverting A.

3. Formal orthogonal polynomials

The orthogonal polynomials P_k defined in the previous section are given by the determinantal formula,

$$P_k(x) = \frac{\begin{vmatrix} 1 & \cdots & x^k \\ c_0 & \cdots & c_k \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_{k-1} & \cdots & c_{2k-1} \end{vmatrix}}{\begin{vmatrix} c_1 & \cdots & c_k \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_k & \cdots & c_{2k-1} \end{vmatrix}}, \qquad (11)$$

where the denominator of this polynomial is $H_k^{(1)}$ [25]. Obviously, P_k exists if and only if the Hankel determinant $H_k^{(1)} \neq 0$, thus, P_{k+1} exists if and only if $H_k^{(1)} \neq 0$. We assume that $\forall k$, $H_k^{(1)} \neq 0$, if for some k, $H_k^{(1)} = 0$, then P_k does not exist and breakdown occurs in the algorithm (in numerical calculations the breakdown can occur even if $H_k^{(1)} \approx 0$).

Let us now define a linear functional $c^{(1)}$ [25] on the space of real polynomials as $c^{(1)}(x^i) = c(x^{i+1}) = c_{i+1}$ and let $P_k^{(1)}$ be a family of orthogonal polynomials with respect to $c^{(1)}$. These polynomials are called monic polynomials [25] (The polynomial in which the highest degree coefficient is always 1) and are given by the following formula:

$$P_k^{(1)}(x) = \frac{\begin{vmatrix} c_1 & \cdots & c_{k+1} \\ . & & . \\ . & & . \\ c_k & \cdots & c_{2k} \\ 1 & \cdots & x^k \end{vmatrix}}{\begin{vmatrix} c_1 & \cdots & c_k \\ . & & . \\ . & & . \\ . & & . \\ c_k & \cdots & c_{2k-1} \end{vmatrix}}. \qquad (12)$$

$P_k^{(1)}(x)$ also exists if and only if the Hankel determinant $H_k^{(1)} \neq 0$ [25], which is also a condition for the existence of $P_k(x)$. There exist many recurrence relations between the two adjacent families of polynomials P_k and $P_k^{(1)}$ [25]. Some of these recurrence relations have been reviewed by Magnus, Gragg, Draux, and Gutknecht describing the recursions for formal orthogonal polynomials (FOPs) and, on the other hand, an investigation by Parlett and his student Taylor. See, e.g., Gutknecht's survey paper [26] for a description of the historic development and for the exact references. See also [27, 28] for more details. Some more recurrence relations between the two families of orthogonal polynomials have been studied in [29], leading to new Lanczos-type algorithms.

A Lanczos-type method consists of computing P_k recursively, then r_k and finally x_k such that $r_k = b - Ax_k$, without inverting A, which gives the solution of the system $Ax = b$ in at most n steps, in exact arithmetic, where n is the dimension of the linear system.

4. Recalling some existing algorithms

In the following we will recall some of the most recent and efficient Lanczos-type Algorithms to be used in the restarting framework. The reader should consult the relevant literature for more details.

4.1. Algorithm A_{12}

Algorithm A_{12} is based on relation A_{12}. For details on the derivation of relation A_{12}, its coefficients and the algorithm itself, please refer to [29]. The pseudo-code of Algorithm A_{12} can be described as follows.

Algorithm 1. Algorithm A_{12}

```
1: Choose $x_0$ and $y$ such that $y \neq 0$,
2: Choose ε small and positive, as a tolerance,
3: Set $r_0 = b - Ax_0, y_0 = y, p = Ar_0, p_1 = Ap$,
   $c_0 = (y, r_0)$,
4: $c_1 = (y, p), c_2 = (y, p_1), c_3 = (y, Ap_1)$,
   $\delta = c_1c_3 - c_2^2$,
5: $\alpha = \frac{c_0c_3 - c_1c_2}{\delta}, \beta = \frac{c_0c_2 - c_1^2}{\delta}$,
   $r_1 = r_0 - \frac{c_0}{c_1}p, x_1 = x_0 + \frac{c_0}{c_1}r_0$.
6: $r_2 = r_0 - \alpha p + \beta p_1, x_2 = x_0 + \alpha r_0 - \beta p$,
7: $y_1 = A^T y_0, y_2 = A^T y_1, y_3 = A^T y_2$.
8: for k = 3, 4,. . . , n do
9: $y_{k+1} = A^T y_k, q_1 = Ar_{k-1}, q_2 = Aq_1, q_3 = Ar_{k-2}$,
10: $a_{11} = (y_{k-2}, r_{k-2}), a_{13} = (y_{k-3}, r_{k-3}), a_{21} = (y_{k-1}, r_{k-2})$,
    $a_{22} = a_{11}$,
11: $a_{23} = (y_{k-2}, r_{k-3}), a_{31} = (y_k, r_{k-2}), a_{32} = a_{21}$,
    $a_{33} = (y_{k-1}, r_{k-3})$,
12: $s = (y_{k+1}, r_{k-2}), t = (y_k, r_{k-3}), F_k = -\frac{a_{11}}{a_{13}}$
13: $b_1 = -a_{21} - a_{23}F_k, b_2 = -a_{31} - a_{33}F_k$,
    $b_3 = -s - tF_k$,
14: $\Delta_k = a_{11}(a_{22}a_{33} - a_{32}a_{23}) + a_{13}(a_{21}a_{32} - a_{31}a_{22})$,
15: $B_k = \frac{b_1(a_{22}a_{33} - a_{32}a_{23}) + a_{13}(b_2a_{32} - b_3a_{22})}{\Delta_k}$
```

```
16: $G_k = \frac{b_1 - a_{11}B_k}{a_{13}}$, $C_k = \frac{b_2 - a_{21}B_k - a_{23}G_k}{a_{22}}$, $A_k = \frac{1}{C_k + G_k}$,
17: $r_k = A_k\{q_2 + B_k q_1 + C_k r_{k-2} + F_k q_3 + G_k r_{k-3}\}$,
18: $x_k = A_k\{C_k x_{k-2} + G_k x_{k-3} - (q_1 + B_k r_{k-2} + F_k r_{k-3})\}$,
19: if $\| r_k \| \le \varepsilon$, then
20: $x = x_k$, stop.
21: end if
22: end for
```

4.2. Algorithm A_4

Algorithm A_4 is based on relation A_4 which is already considered by C. Bahuex in her PhD thesis. Its pseudo-code is as follows.

Algorithm 2. Algorithm A_4

```
1: Choose $x_0$ and $y$ such that $y \neq 0$,
2: Choose ε small and positive as a tolerance,
3: Set $r_0 = b - Ax_0$, $y_0 = y$
4: for k = 0, 1,. . . , n do
5: $E_{k+1} = -\frac{(y_k, r_k)}{(y_{k-1}, r_{k-1})}$, $for\ k \ge 1\ and\ E_1 = 0$
6: $B_{k+1} = -\frac{(y_k, Ar_k) - E_{k+1}(y_k, r_{k-1})}{(y_k, r_k)}$,
7: $A_{k+1} = -\frac{1}{B_{k+1} + E_{k+1}}$,
8: $x_{k+1} = A_{k+1}\{B_{k+1}x_k + E_{k+1}x_{k-1} - r_k\}$,
9: $r_{k+1} = A_{k+1}\{Ar_k + B_{k+1}r_k - E_{k+1}r_{k-1}\}$,
10: if $\| r_{k+1} \| \le \varepsilon$, then
11: $y_{k+1} = A^T y_k$.
12: end if
13: end for
```

4.3. Algorithm A_5 / B_{10}

Algorithm A_5 / B_{10} is based on relations A_5 and B_{10}, first investigated by C. Bahuex in her PhD thesis. Its pseudo-code is as follows.

Algorithm 3. Algorithm A_5/B_{10}

```
1: Choose $x_0$, $y$ and tolerance $\varepsilon \ge 0$;
2: Set $r_0 = b - Ax_0$, $p_0 = r_0$, $y_0 = y$,
3: $A_1 = -\frac{(y_0, r_0)}{(y_0, Ar_0)}$, $C_1^0 = 1$
4: $r_1 = r_0 + A_1 Ar_0$, $x_1 = x_0 - A_1 r_0$.
5: for k = 1, 2, 3, . . . , n do
6: $y_k = A^T y_{k-1}$
7: $D_{k+1} = -\frac{(y_k, r_k)}{C_{k-1}^1 (y_k, p_{k-1})}$,
8: $p_k = r_k + D_{k-1} C_{k-1}^1 p_{k-1}$,
9: $A_{k+1} = -\frac{(y_k, r_k)}{(y_k, Ap_k)}$,
10: $r_{k+1} = r_k + A_{k+1} Ap_k$,
11: $x_{k+1} = x_k - A_{k+1} p_k$,
12: if $\| r_{k+1} \| \le \varepsilon$, then if $A_k \le \varepsilon$, then
13: $C_k^1 = \frac{C_{k-1}^1}{A_k}$.
14: end if
15: end for
```

4.4 Algorithm A_8/B_{10}: The pseudo-code of A_8/B_{10} is as follows.

Algorithm 4 Algorithm A_8/B_{10}

```
1: Choose $x_0$ and $y$ such that $y \neq 0$.
2: Set $r_0 = b - Ax_0$
3: $z_0 = r_0$
```

```
4: $y_0 = y$
5: for $k = 0, 1, 2, \ldots, n$ do
6:   $A_{k+1} = -\frac{(y_k, r_k)}{(y_k, Az_k)}$,
7:   $r_{k+1} = r_k + A_{k+1}Az_k$,
8:   $x_{k+1} = x_k - A_{k+1}z_k$,
9:   if $\| r_{k+1} \| \neq \varepsilon$ then
10:    $y_{k+1} = A^T y_k$,
11:    $C^1_{k+1} = \frac{1}{A_{k+1}}$,
12:    $B^1_{k+1} = -\frac{C^1_{k+1}(y_{k+1}, r_{k+1})}{(y_k, Az_k)}$,
13:    $z_{k+1} = B^1_{k+1}z_k + C^1_{k+1}r_{k+1}$.
14:  end if
15: end for
```

5. Restarting an algorithm as a way to remedy the breakdown problem

When a Lanczos-type algorithm fails, it is due to the non-existence of some coefficients of the recurrence relations on which the algorithm is based. The iterate which causes these coefficients not to exist does not cause and should not necessarily cause any problem when used in another Lanczos-type algorithm, based on different recurrence relations. It is therefore obvious that one may consider switching to another algorithm when breakdown occurs [30]. This means that it is possible to remedy breakdown by switching. It is also the case that the iterate generated by a Lanczos-type algorithm that causes it to fail can be used to initialize the same algorithm successfully. This allows the algorithm to work in a Krylov space with a different basis. It is therefore also possible to remedy breakdown by restarting.

5.1. Restarting strategies

They follow the same pattern as switching [30], except that here, the alternative algorithm to switch to is the same as the one we started with. This means that, by re-initializing the Lanczos process with another iterate, the last one found starts working with a different Krylov space base. This is enough to fix any numerical difficulties that have occurred and avert any that might occur. Different strategies can be adopted for restarting different algorithms. These are as follows.

1. **Restarting after breakdown:** Start a particular Lanczos algorithm until a breakdown occurs, then restart the same Lanczos algorithm, initializing it with the last iterate of the failed algorithm. We call this strategy **ST1**.
2. **Pre-emptive restarting:** Run a Lanczos-type algorithm for a fixed number of iterations, halt it and then restart it, initializing it with the last iterate. Note that there is no way to guarantee that breakdown would not occur before the end of the interval. This strategy is called **ST2**.
3. **Breakdown monitoring:** Provided monotonicity of reduction in the absolute value of denominators involved in the coefficients of the polynomials involved can be established, breakdown can be monitored as follows. Evaluate regularly those coefficients with denominators that are likely to become zero. Restart the algorithm when the absolute value of any of these denominators drops below a certain level. This is strategy **ST3**.

5.2. A generic restarting algorithm

Suppose we have a set of Lanczos-type algorithms and we want to use one of these algorithms in the restarting framework using one of the above mentioned strategies ST1, ST2 or ST3. Then the following algorithm can be used.

Algorithm 5 Generic restarting algorithm

```
1: Start the most stable algorithm, if known.
2: Choose a restarting strategy from {ST1, ST2, ST3}.
3: if ST1 then
4:   Continue with current algorithm until it halts;
5:   if solution is obtained then
6:     Stop.
7:   else
8:     restart the same algorithm;
```

```
9: initialize it with current iterate,
10: Go to 4.
11: **end if**
12: **else if** *ST2* **then**
13: Continue with current algorithm for a fixed number of iterations until it stops;
14: **if** solution is obtained **then**
15: Stop.
16: **else**
17: restart the same algorithm,
18: initialize it with the current iterate,
19: Go to 13.
20: **end if**
21: **else**
22: Continue with current algorithm and monitor certain parameters for breakdown, until it halts
23: **if** solution is obtained **then**
24: Stop.
25: **else**
26: restart the same algorithm,
27: initialize it with current iterate,
28: Go to 22.
29: **end if**
30: **end if**
```

Note that we have only considered strategy *ST2* in this paper. Strategy *ST1* is not going to be efficient because a breakdown causes the programme to halt. This requires restarting it by hand. Since this occurs rather frequently, particularly in high dimensions, it is not practical to carry out experiments in this case. Strategy *ST3* is not implemented since it is clear that it will not be efficient, at least in the sequential environment we are operating in. Looking at algorithms A_4, A_{12}, A_5/B_{10}, and A_8/B_{10} given earlier, one notices easily that they involve identifying the coefficients of FOP's used in the Lanczos process. Many of these coefficients are ratios. It is the denominator quantities of these ratios that have to be monitored in *ST*2. There may be a lot of them and there is no obvious way to see which one is decreasing alarmingly and threatening to cause breakdown. There is a serious amount of work yet to be done to understand how these coefficients behave in order to find which ones should be monitored closely. It is also reasonable to assume that, in a parallel environment many can be monitored simultaneously, not in a sequential environment. For these reasons, we have only investigated strategy *ST*2. In the following, a generic algorithm implementing *ST*2 is given. Note that a convergence tolerance $\varepsilon = 1.0e^{-013}$ and 20 iterations per cycle are used in all experiments.

5.3. Implementing ST2

*ST*2 takes as input a given algorithm from a pre-specified list. Here, these algorithms are the ones already listed above, i.e. A_4, A_{12}, A_5/B_{10}, and A_8/B_{10}. Depending on whether the algorithms are of the A_i-type (i.e. Lanczos-type algorithm based on a single recurrence relation) or A_i/B_j-type (i.e. Lanczos-type algorithm based on two recurrence relations), initialisation has to be done differently; A_i-type requires x_0, $r_0 = b - Ax_0$ and $y_0 = y$, and A_i/B_j-type requires x_0, $r_0 = b - Ax_0$ and $y_0 = y$ as well as $z_0 = r_0$. The general *ST*2 algorithm can be described, therefore, as follows.

5.3.1. Algorithm ST2

Algorithm 6 *ST*2 restarting algorithm

```
1: Choose algorithm ALG from {A_4, A_12, A_5/B_10, A_8/B_10}
2: Choose $x_0$ and $y$ such that $y \neq 0$,
3: set $r_0 = b - Ax_0$, $y_0 = y$,
4: **if** $ALG \neq A_4 \wedge ALG \neq A_{12}$ **then**
5: $z_0 = r_0$;
6: **end if**
7: run ALG for a fixed number of iterations (a cycle) or until it halts;
8: **if** solution is obtained **then**
```

```
9: stop;
10: **else**
11: initialize it with the current iterate;
12: go to 3;
13: **end if**
```

5.3.2. Numerical results

Algorithms 1, 2, 3, 4, and algorithm *ST*2 restarting each one of them [17] has been implemented in Matlab and applied to a number of small to medium size problems for different values of δ. The test problems we have used arise in the 5-point discretisation of the operator $-\frac{\partial^2}{\partial x^2}-\frac{\partial^2}{\partial y^2}+\gamma\frac{\partial}{\partial x}$ on a rectangular region. We refer to them as the Baheux-type problems since we borrowed them from her PhD thesis. These problems are important because they represent the typical SLE's that crop up in the real world. They are also used by Baheux herself to test three of the algorithms used here. Comparative results are carried out on instances of size ranging from n = 20 to n = 4000 of the problem $Ax=b$ with A and b as follows

$$A=\begin{pmatrix} B & -I & \dots\dots & & 0 \\ -I & B & & & : \\ : & .. & & & .. \\ : & & & B & -I \\ 0 & \dots\dots & & -I & B \end{pmatrix},$$

with

$$B=\begin{pmatrix} 4 & \alpha & \dots\dots & & 0 \\ \beta & 4 & \alpha & & : \\ : & .. & & & .. \\ : & & \beta & 4 & \alpha \\ 0 & \dots\dots & & \beta & 4 \end{pmatrix}$$

and $\alpha=-1+\delta$, $\beta=-1-\delta$. The parameter δ takes values 0.0, 0.2, 5 and 8 respectively. The right hand side **b** is taken to be $b=AX$, where $X=(1,1,...,1)^T$ the solution of the system is. The dimension of B is 10. When δ = 0, the coefficient matrix A is symmetric and the problem is easy to solve because the region is a regular mesh [31]. For all other values of δ, the matrix A is non-symmetric and the problem is comparatively hard to solve as the region is not a regular mesh.

The results obtained with algorithms $A_4, A_{12}, A_5/B_{10}$ and A_8/B_{10} executed with the *ST*2 restarting each one of them, for different values of δ on Baheux type problems, are recorded in Tables 1, 2, 3 and 4 below. The results show that the restarting algorithm ST2 is far superior to any one of the algorithms considered individually.

5.3.3. Comments on the numerical evidence

We have implemented $A_4, A_{12}, A_5/B_{10}$ and A_8/B_{10} to solve a number of problems of the type described in Section 5.3.2 with dimensions ranging from 20 to 4000. The results are compared against those obtained by the restarting strategy *ST*2 used on each one of the algorithms $A_4, A_{12}, A_5/B_{10}$ and A_8/B_{10}, on the same problems. They show that algorithms $A_4, A_{12}, A_5/B_{10}$ and A_8/B_{10} are not as robust as when they are restarted in a preemptive fashion (under *ST*2). Outside the restarting framework, the algorithms failed to solve any of the problems for dimensions greater than 40. Within the restarting framework, all problems have been solved with high precision. This is recorded in Table 1 through Table 4. The results, undoubtedly point to restarting as another way of curing breakdown in Lanczos-type algorithms.

Table 1. Numerical results for $\delta = 0$

Dim of Prob	ST2(A_4)		ST2(A_{12})		ST2(A_5/B_{10})		ST2(A_8/B_{10})	
n	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)
20	$3.8545e^{-014}$	0.0010	$8.4317e^{-014}$	0.0075	$1.8157e^{-014}$	0.0050	$8.0535e^{-014}$	0.0076
40	$6.0376e^{-014}$	0.0043	$7.7873e^{-014}$	0.0126	$2.2438e^{-014}$	0.0102	$7.4281e^{-014}$	0.0135
60	$8.8041e^{-014}$	0.0125	$3.8138e^{-014}$	0.0161	$2.3873e^{-014}$	0.0127	$6.6555e^{-014}$	0.0214
80	$9.8201e^{-014}$	0.0140	$6.1219e^{-014}$	0.0205	$7.9308e^{-014}$	0.0163	$7.1708e^{-014}$	0.0220
100	$2.6273e^{-014}$	0.0193	$9.4670e^{-014}$	0.0216	$2.4008e^{-014}$	0.0161	$7.5103e^{-014}$	0.0227
200	$7.5106e^{-014}$	0.0803	$6.1319e^{-014}$	0.0931	$9.9663e^{-014}$	0.0652	$7.6984e^{-014}$	0.0705
400	$8.2317e^{-014}$	0.5170	$9.0230e^{-014}$	0.4105	$8.0372e^{-014}$	0.9814	$9.1328e^{-014}$	0.7058
600	$8.9464e^{-014}$	0.9418	$9.6683e^{-014}$	2.0031	$9.7269e^{-014}$	2.9632	$6.9507e^{-014}$	2.1481
800	$8.5604e^{-014}$	1.5466	$8.8581e^{-014}$	3.2419	$9.9864e^{-014}$	5.8682	$9.7003e^{-014}$	4.7235
1000	$8.7519e^{-014}$	2.4005	$8.4853e^{-014}$	4.7680	$9.6715e^{-014}$	8.7822	$9.2346e^{-014}$	8.5468
2000	$8.3024e^{-014}$	9.3332	$5.5149e^{-014}$	21.6685	$7.8361e^{-014}$	20.4821	$8.2291e^{-014}$	39.4606
3000	$9.9026e^{-014}$	24.8801	$9.4196e^{-014}$	56.1770	$8.7229e^{-014}$	61.0065	$9.3587e^{-014}$	103.5353
4000	$9.2511e^{-014}$	48.0754	$6.0811e^{-014}$	113.4374	$7.8230e^{-014}$	141.9979	$9.1120e^{-014}$	176.2555

Table 2. Numerical results for $\delta = 0.2$

Dim of Prob	ST2(A_4)		ST2(A_{12})		ST2(A_5/B_{10})		ST2(A_8/B_{10})	
n	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)
20	$9.9896e^{-014}$	0.0084	$8.9978e^{-014}$	0.0130	$5.0531e^{-014}$	0.0108	$6.1999e^{-014}$	0.0137
40	$4.9296e^{-014}$	0.0191	$5.1552e^{-014}$	0.0292	$8.7027e^{-014}$	0.0205	$5.2118e^{-014}$	0.0240
60	$6.0957e^{-014}$	0.0210	$5.5319e^{-014}$	0.0328	$5.4364e^{-014}$	0.0280	$7.2870e^{-014}$	0.0314
80	$5.7129e^{-014}$	0.0250	$9.2631e^{-014}$	0.0388	$9.1910e^{-014}$	0.0362	$3.5101e^{-014}$	0.0410
100	$5.3007e^{-014}$	0.0305	$9.4369e^{-014}$	0.0440	$3.2209e^{-014}$	0.0392	$7.0508e^{-014}$	0.0431
200	$2.2236e^{-014}$	0.0980	$9.9659e^{-014}$	0.1485	$5.8291e^{-014}$	0.0851	$6.1683e^{-014}$	0.1164
400	$9.0428e^{-014}$	0.4163	$8.5500e^{-014}$	0.3117	$8.0768e^{-014}$	0.5826	$8.2751e^{-014}$	0.7639
600	$6.6926e^{-014}$	1.0532	$9.2957e^{-014}$	1.1802	$6.9817e^{-014}$	2.4674	$9.9801e^{-014}$	2.0367
800	$8.6281e^{-014}$	1.5170	$4.8311e^{-014}$	1.8420	$8.2814e^{-014}$	3.2158	$7.1025e^{-014}$	4.7437
1000	$9.0327e^{-014}$	2.1245	$8.1326e^{-014}$	3.3795	$9.8567e^{-014}$	7.8925	$7.8650e^{-014}$	8.1761
2000	$7.7193e^{-014}$	8.0167	$9.1597e^{-014}$	12.8866	$9.3404e^{-014}$	33.4418	$8.1368e^{-014}$	30.7758
3000	$8.1792e^{-014}$	19.2173	$9.1855e^{-014}$	45.6061	$6.1042e^{-014}$	123.3084	$8.2046e^{-014}$	89.0659
4000	$8.9238e^{-014}$	36.0361	$9.8884e^{-014}$	74.3756	$9.2065e^{-014}$	280.4058	$8.0349e^{-014}$	180.6880

Table 3. Numerical results for $\delta = 5$

Dim of Prob	ST2(A_4)		ST2(A_{12})		ST2(A_5/B_{10})		ST2(A_8/B_{10})	
N	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)	$\lVert r_k \rVert$	T(s)
20	$1.2543e^{-014}$	0.0110	$2.1773e^{-014}$	0.0168	$1.6184e^{-014}$	0.0133	$5.8176e^{-014}$	0.0159
40	$6.7523e^{-014}$	0.0442	$4.5708e^{-014}$	0.0691	$9.8328e^{-014}$	0.0556	$8.3294e^{-014}$	0.0466
60	$2.9307e^{-014}$	0.0524	$9.5280e^{-014}$	0.1127	$6.4059e^{-014}$	0.0573	$7.5235e^{-014}$	0.0584
80	$6.9125e^{-014}$	0.0634	$4.0822e^{-014}$	0.0909	$3.3983e^{-014}$	0.0766	$9.7386e^{-014}$	0.0759
100	$6.9001e^{-014}$	0.0707	$6.7233e^{-014}$	0.1195	$9.3530e^{-014}$	0.0846	$8.6400e^{-014}$	0.0914
200	$5.4021e^{-014}$	0.1661	$7.2644e^{-014}$	0.1957	$9.1453e^{-014}$	0.1553	$9.2653e^{-014}$	0.1747
400	$6.9921e^{-014}$	0.4889	$9.8208e^{-014}$	1.0148	$9.2802e^{-014}$	0.8543	$5.7742e^{-014}$	0.8976
600	$2.6118e^{-014}$	0.9507	$9.4438e^{-014}$	1.2438	$9.9646e^{-014}$	2.1217	$9.0410e^{-014}$	1.6980
800	$4.5923e^{-014}$	1.9086	$8.6228e^{-014}$	1.9261	$5.7573e^{-014}$	4.0326	$9.9560e^{-014}$	3.2380
1000	$4.5632e^{-014}$	3.0028	$9.1161e^{-014}$	5.6501	$9.9696e^{-014}$	6.1502	$8.7909e^{-014}$	6.6809
2000	$9.8816e^{-014}$	9.4847	$8.0887e^{-014}$	32.3405	$9.1481e^{-014}$	29.4155	$5.2993e^{-014}$	29.2111
3000	$7.1675e^{-014}$	37.2835	$7.1249e^{-014}$	66.3288	$8.5008e^{-014}$	81.7649	$6.3973e^{-014}$	85.8018
4000	$7.9284e^{-014}$	60.0409	$6.0671e^{-014}$	118.7937	$8.1534e^{-014}$	173.6725	$5.1241e^{-014}$	145.5099

Table 4. Numerical results for $\delta = 8$

Dim of Prob	ST2(A_4)		ST2(A_{12})		ST2(A_5/B_{10})		ST2(A_8/B_{10})	
n	$\|\| r_k \|\|$	T(s)	$\|\| r_k \|\|$	T(s)	$\|\| r_k \|\|$	T(s)	$\|\| r_k \|\|$	T(s)
20	$1.2127e^{-014}$	0.0116	$9.9725e^{-014}$	0.0162	$7.6040e^{-014}$	0.0132	$6.4653e^{-014}$	0.0121
40	$9.1661e^{-014}$	0.0403	$8.8142e^{-014}$	0.0727	$2.1073e^{-014}$	0.0601	$6.9719e^{-014}$	0.0547
60	$6.2183e^{-014}$	0.0782	$8.2240e^{-014}$	0.0928	$9.1420e^{-014}$	0.0783	$2.4335e^{-014}$	0.0671
80	$9.5772e^{-014}$	0.0750	$6.8254e^{-014}$	0.0893	$6.1842e^{-014}$	0.0974	$6.4862e^{-014}$	0.0933
100	$8.9719e^{-014}$	0.0729	$9.2600e^{-014}$	0.1520	$9.5086e^{-014}$	0.1059	$5.3579e^{-014}$	0.1075
200	$9.7792e^{-014}$	0.1528	$5.7848e^{-014}$	0.2396	$6.8288e^{-014}$	0.1784	$6.4354e^{-014}$	0.2528
400	$6.3091e^{-014}$	0.3592	$9.0009e^{-014}$	0.5326	$6.1809e^{-014}$	0.9791	$9.7548e^{-014}$	1.1240
600	$9.5208e^{-014}$	1.0349	$9.7036e^{-014}$	1.5674	$4.1802e^{-014}$	2.7026	$4.8723e^{-014}$	2.3533
800	$7.7096e^{-014}$	1.9936	$2.4209e^{-014}$	4.2043	$6.2004e^{-014}$	4.9577	$9.3335e^{-014}$	4.6741
1000	$9.8712e^{-014}$	4.8990	$9.4187e^{-014}$	5.2569	$3.8734e^{-014}$	7.3104	$8.7525e^{-014}$	7.1334
2000	$8.6980e^{-014}$	13.9378	$5.5222e^{-014}$	29.6383	$9.4111e^{-014}$	36.2889	$6.8768e^{-014}$	38.3224
3000	$7.8024e^{-014}$	42.6996	$7.5285e^{-014}$	73.6804	$4.5003e^{-014}$	84.8624	$7.9196e^{-014}$	100.3153
4000	$6.6516e^{-014}$	115.1201	$9.2836e^{-014}$	159.9751	$6.7992e^{-014}$	179.9848	$8.6748e^{-014}$	228.4823

6. Conclusion

The restarting strategy *ST*2 considered seems to be very successful at remedying and avoiding breakdown in Lanczos-type algorithms. The supporting numeral evidence is very strong in this respect. Indeed, restarting solved all problems up to dimension of 4000 while individual algorithms only managed to solve them for lower dimensions (≤ 40). The cost of preemptive restarting is not substantial. In the case of monitoring the coefficients that can vanish, although the cost is only that of a test of the form **if** |denominator value| ≤ tolerance **then** stop, many such tests may be carried out. We have not measured its impact on the overall computing time. Positive experimental results point to restarting as a significant approach to avoiding breakdown in solving SLE's by Lanczos-type algorithms. This idea is not only different from existing strategies for dealing with breakdown, [32-34], but also quite easy to understand and follow. However, it remains for some more extensive testing to be done on both large real and randomly generated problems to fully understand the behavior and cost of the restarting approach compared to state-of-the-art Lanczos-type algorithms with in-built precautions to avoid breakdown such as MRZ and BSMRZ. Further work is being expanded in this direction.

References

[1] Baheux, C. (1994). Algorithmes d'implementation de la méthode de Lanczos, PhD thesis, University of Lille 1, France.

[2] Baheux, C. (1995). New Implementations of Lanczos Method. *Journal of Computational and Applied Mathematics*, *57*, 3-15.

[3] Brezinski, C. & Sadok, H. (1993). Lanczos-type algorithms for solving systems of linear equations. *Applied Numerical Mathematics*, *11*, 443-473.

[4] Parlett, B. N., Taylor, D. R. & Liu, Z. A. (1985). A Look-Ahead Lanczos Algorithm for Unsymmetric Matrices. *Mathematics of Computation*, *44*, 105-124.

[5] Brezinski, C. M., Zaglia, R. & Sadok, H. (1991). Avoiding breakdown and nearbreakdown in Lanczos type algorithms. *Numerical Algorithms*, *1*, 261-284.

[6] Brezinski, C., Zaglia, M. R. & Sadok, H. (1992). Addendum to Avoiding breakdown and near-breakdown in Lanczos type algorithms. *Numerical Algorithms*, *2*, 133-136.

[7] Gutknecht, M. H. (1992). A completed theory of the unsymmetric Lanczos process and related algorithms, Part I. SIAM *J. Matrix Anal. Appl.*, *13*, 594-639.

[8] Freund, R. W., Gutknecht, M. H. & Nachtigal, N. M. (1993). An Implementation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices. *SIAM J. Sci. Comput.*, *14*, 137-158.

[9] Graves-Morris, P. R. (1997). A "Look-around Lanczos" algorithms for solving a system of linear equations. *Numerical Algorithms*, *15*, 247-274.

[10] Brezinski, C., Zaglia, M. R. & Sadok, H. (1999). New look-ahead Lanczos-type algorithms for linear systems. *Numerische Mathematik*, *83*, 53-85.

[11] Liu, Q. (2008). Some preconditioning techniques for linear systems. *WSEAS Transactions on Mathematics*, *7*(9), 579-588.

[12] Parlett, B. N. & Scott, D. S. (1979). The Lanczos algorithm with selective orthogonaliztion. *Mathematics of Computation*, *33*, 217-238.

[13] Brezinski, C. & Zaglia, M. R. (1995). Look-ahead in Bi-CGSTAB and other product methods for linear systems. *BIT Numerical Mathematics*, *35*(2), 169-201.

[14] Hoffnung, L., Li, R. C. & Krylov, Q. Ye. (2006). Krylov type subspace methods for matrix polynomials. *Linear Algebra and its Applications*, *415*(1), 52-81.

[15] Brezinski, C., Zaglia, M. R. & Sadok, H. (2000). The matrix and polynomial approaches to Lanczos-type algorithms. *Journal of Computational and Applied Mathematics*, *123*, 241-260.

[16] Brezinski, C. & Zaglia, M. R. (1994). Breakdowns In The Computation Of Orthogonal Polynomials. *In Nonlinear Numerical Methods and Rational Approximation*, 49-59.

[17] Brezinski, C., Zaglia, M. R. & Sadok, H. (1992). A Breakdown-free Lanczos type algorithm for solving linear systems, *Numerische Mathematik*, *63*, 29-38.
[18] El Guennouni, A. (1999). A unified approach to some strategies for the treatment of breakdown in Lanczos-type algorithms. *Applicationes Mathematicae*, *26*, 477-488.
[19] Brezinski, C. & Zaglia, M. R. (1994). Treatment of near-breakdown in the CGS algorithm. *Numerical Algorithms*, *7*(1), 33-73.
[20] Lanczos, C. (1952). Solution of systems of linear equations by minimized iteration. *Journal of the National Bureau of Standards*, *49*, 33-53.
[21] Brezinski, C., Zaglia, M. R. & Sadok, H. (2002). A review of formal orthogonality in Lanczos-based methods. *Journal of Computational and Applied Mathematics*, *140*, 81-98.
[22] Cybenko, G. (1987). An explicit formula for Lanczos polynomials. *Linear Algebra Appl.*, *88*, 99-115.
[23] Chihara, T. S. (1984). *An Introduction to Orthogonal Polynomials*. New York, London, Paris. Gordon and Breach.
[24] Brezinski, C. (1980). Padé-Type Approximation and General Orthogonal Polynomials, Internat. *Ser. Nuner. Math*, *50*, Birkh¨*a*user, Basel.
[25] Draux, A. (1983). Polyńomes Orthogonaux Formels. Application, LNM 974, Berlin, Springer-Verlag.
[26] Gutknecht, M. H. (1994). A completed theory of the unsymmetric Lanczos process and related algorithms, Part II. SIAM *SIAMJ. Matrix Anal. Appl.*, *15*, 15-58.
[27] Golub, G. H. & O'Leaxy, D. P. (1989) Some history of the conjugate and Lanczos algorithms. *SIAM Rev. 31*, 50-102.
[28] Hestenes, M. R. (1980). Conjugate Direction Methods in Optimization. Berlin, Springer-Verlag.
[29] Farooq, M. (2011). New Lanczos-type Algorithms and their Implementation. PhD thesis, University of Essex, UK. http://serlib0.essex.ac.uk/record=b1754556
[30] Farooq, M. & Salhi, A. (2014). A Switching Approach to Avoid Breakdown in Lanczos-type Algorithms. *Applied Mathematics and Information Sciences*, to appear in Volume 8.
[31] Meurant, G. (2006). The Lanczos and conjugate gradient algorithms, from Theory to Finite Precision Computations. Philadelphia, SIAM.
[32] Sonneveld, P. (1989). A fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput*. *10*, 35-52.
[33] Taylor, D. R. (1982). Analysis of the look-ahead Lanczos algorithm, Ph.D. Thesis, Dept. of Mathematics, University of California, Berkeley.
[34] Brezinski, C. M., Zaglia, R. & Sadok, H. (1991). numeralgo/na1, http://www.netlib.org/cgi-bin/search.pl.