A Switching Approach to Avoid Breakdown in Lanczos-type Algorithms

Muhammad Farooq and Abdellah Salhi

Abstract

Lanczos-type algorithms are well known for their inherent instability. They typically breakdown when relevant orthogonal polynomials do not exist. Current approaches to avoiding breakdown rely on jumping over the non-existent polynomials to resume computation. This jumping strategy may have to be used many times during the solution process. We suggest an alternative to jumping which consists in switching between different algorithms that have been generated using different recurrence relations between orthogonal polynomials. This approach can be implemented as three different strategies: ST1, ST2, and ST3. We shall briefly recall how Lanczos-type algorithms are derived. Four of the most prominent such algorithms namely $A_4$, $A_{12}$, $A_5/B_{10}$ and $A_5/B_8$ will be presented and then deployed in the switching framework. In this paper, only strategy ST2 will be investigated. Numerical results will be presented.

2010 Mathematics Subject Classification: 65F10

Keywords: Lanczos algorithm; Systems of Linear Equations (SLE’s); Formal Orthogonal Polynomials (FOP’s); Switching; Restarting; Breakdown.
1 Introduction

Lanczos-type methods for solving SLE’s are based on the theory of FOP’s. 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, 14]. The coefficients of the various recurrence relationships between 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 polynomial or polynomials. So, an important issue is how to continue the solution process in such a situation and arrive at a usable result. Several procedures for that purpose appeared in the literature in the last few decades. It has been shown, for instance, that it is possible to jump over non-existing polynomials, [1, 4]; breakdown-free algorithms were thus obtained. The first attempt in this regard was the look-ahead Lanczos algorithm, [22]. Other procedures for avoiding breakdown are also proposed in [18, 7, 9, 1, 4, 11, 17, 16, 21]. However, they all have their limitations including the possibility of calling the procedure for remedying the breakdown, more than once. In the following, we suggest an alternative to jumping over missing polynomials by switching between different variants of the Lanczos algorithm.

2 The Lanczos approach

We consider a linear system of equations,

$$Ax = b,$$  

(1)

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

Let $x_0$ and $y$ be two arbitrary vectors in $\mathbb{R}^n$ such that $y \neq 0$. The Lanczos method, [19] consists in constructing a sequence of vectors $x_k \in \mathbb{R}^n$ defined as follows, [12, 2]

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

(2)

$$r_k = (b - Ax_k) \perp K_k(A^T, y) = \text{span}(y, A^Ty, \ldots, A^{T^{k-1}}y),$$  

(3)

where $A^T$ denotes the transpose of $A$.

Equation (2) leads to,

$$x_k - x_0 = -\alpha_1 r_0 - \alpha_2 Ar_0 - \cdots - \alpha_k A^{k-1}r_0.$$  

(4)
Multiplying both sides by $A$ and adding and subtracting $b$ on the left hand side gives
\[ \mathbf{r}_k = \mathbf{r}_0 + \alpha_1 A \mathbf{r}_0 + \alpha_2 A^2 \mathbf{r}_0 + \cdots + \alpha_k A^k \mathbf{r}_0. \] (5)

From (3), the orthogonality condition gives
\[ (A^T \mathbf{y}, \mathbf{r}_k) = 0, \text{ for } i = 0, \ldots, k - 1, \]
and, by (5), we obtain the following system of linear equations
\[
\begin{cases}
\alpha_1 (\mathbf{y}, A \mathbf{r}_0) + \cdots + \alpha_k (\mathbf{y}, A^k \mathbf{r}_0) = -(\mathbf{y}, \mathbf{r}_0), \\
\vdots \\
\alpha_1 (A^{T^{k-1}} \mathbf{y}, A \mathbf{r}_0) + \cdots + \alpha_k (A^{T^{k-1}} \mathbf{y}, A^k \mathbf{r}_0) = -(A^{T^{k-1}} \mathbf{y}, \mathbf{r}_0).
\end{cases}
\] (6)

If the determinant of the above system is different from zero then its solution exists and allows to obtain $\mathbf{x}_k$ and $\mathbf{r}_k$. 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
\[ P_k(x) = 1 + \alpha_1 x + \cdots + \alpha_k x^k, \] (7)
then we can write from (5)
\[ \mathbf{r}_k = P_k(A) \mathbf{r}_0. \] (8)

The polynomials $P_k$ are commonly known as the residual polynomials. [4]. Another interpretation of the $P_k$ can be found in [13]. Moreover if we set $c_i = (A^T \mathbf{y}, \mathbf{r}_0) = (\mathbf{y}, A^i \mathbf{r}_0), i = 0, 1, \ldots,$ and if we define the linear functional $c$ on the space of polynomials by
\[ c(x^i) = c_i, \quad i = 0, 1, \ldots, \] (9)
c is completely determined by the sequence $\{c_k\}$ and $c_k$ is said to be the moment of order $k$, [3]. Now, the system (6) can be written as
\[ c(x^i P_k(x)) = 0 \text{ for } i = 0, \ldots, k - 1. \] (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 a family of FOP’s with respect to the linear functional $c$, [10, 3].

Since the constant term of $P_k$ in (7) is 1, it can be written as
\[ P_k(x) = 1 + x R_{k-1}(x) \]
where $R_{k-1} = \alpha_1 + \alpha_2 x + \cdots + \alpha_k x^{k-1}$. Replacing $x$ by $A$ in the expression of $P_k$ and multiplying both sides by $\mathbf{r}_0$ and using (5), we get
\[ \mathbf{r}_k = \mathbf{r}_0 + A R_{k-1}(A) \mathbf{r}_0, \]

3
which can be written as

\[ b - Ax_k = b - Ax_0 + AR_{k-1}(A)r_0, \]
\[ -Ax_k = -Ax_0 + AR_{k-1}(A)r_0, \]
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, \(6, 4\)

\[
P_k(x) = \frac{\begin{vmatrix}
1 & \cdots & x^k \\
c_0 & \cdots & c_k \\
\vdots & & \vdots \\
c_{k-1} & \cdots & c_{2k-1} \\
\end{vmatrix}}{\begin{vmatrix}
c_1 & \cdots & c_k \\
\vdots & & \vdots \\
c_k & \cdots & c_{2k-1} \\
\end{vmatrix}},
\]

(11)

where the denominator of this polynomial is \(H_k^{(1)}\), \(4\). 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 \(H_{k+1}^{(1)} \neq 0\). 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 practice the breakdown can occur even if \(H_k^{(1)} \approx 0\)).

Let us now define a linear functional \(c^{(1)}\), \(4, 2\), 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, \(4, 2\), because their highest degree coefficients are always 1, and are
given by the following formula

\[ P_k^{(1)}(x) = \begin{vmatrix} c_1 & \cdots & c_{k+1} \\ \vdots & \ddots & \vdots \\ c_k & \cdots & c_{2k} \\ 1 & \cdots & x^k \end{vmatrix}. \]  \tag{12} \]

\( P_k^{(1)}(x) \) also exists if and only if the Hankel determinant \( H_k^{(1)} \neq 0 \), \([4,2]\), 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 \( P_k^{(1)} \), \([2,4,6,7]\). Some of these relations have been reviewed in \([5]\) and studied in details in \([1,6]\). More of these relations have been studied in \([14]\), leading to new Lanczos-type algorithms.

A Lanczos-type algorithm consists in computing \( P_k \) recursively, then \( r_k \) and finally \( x_k \) such that \( r_k = b - Ax_k \), without inverting \( A \). In exact arithmetic, this should give the solution to the system \( Ax = b \) in at most \( n \) steps \([19,7]\), where \( n \) is the dimension of the system. For more details, see \([4,11]\).

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 switching framework. The reader should consult the relevant literature for more details.

4.1 Algorithm \( A_{12} \)

Algorithm \( A_{12} \) is based on relation \( A_{12} \), \([14]\). For details on the derivation of the polynomial \( A_{12} \), its coefficients and the algorithm itself, please refer to \([14]\). 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 $\epsilon$ 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: $c_1 = (y, p), c_2 = (y, p_1), c_3 = (y, Ap_1), \delta = c_1 c_3 - c_2^2,$
5: $\alpha = \frac{c_3^2 - c_1 c_2}{\delta}, \beta = \frac{c_3^2 - c_2^2}{\delta}, r_1 = r_0 - \frac{\alpha}{c_1} p, x_1 = x_0 + \frac{\alpha}{c_1} r_0,$
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 $k = 3, 4, \ldots, n$ do 
9: \hspace{1em} $y_{k+1} = A^T y_k, q_1 = Ar_{k-1}, q_2 = Aq_1, q_3 = Ar_{k-2},$
10: \hspace{1em} $a_{11} = (y_{k-2}, r_{k-2}), a_{13} = (y_{k-3}, r_{k-3}), a_{21} = (y_{k-1}, r_{k-2}), a_{22} = a_{11},$
11: \hspace{1em} $a_{23} = (y_{k-2}, r_{k-3}), a_{31} = (y_{k-3}, r_{k-2}), a_{32} = a_{21}, a_{33} = (y_{k-1}, r_{k-3}),$
12: \hspace{1em} $s = (y_{k+1}, r_{k-2}), t = (y_k, r_{k-3}), F_k = \frac{a_{13}}{a_{12}},$
13: \hspace{1em} $b_1 = -a_{21} - a_{23} F_k, b_2 = -a_{31} - a_{33} F_k, b_3 = -s - t F_k,$
14: \hspace{1em} $\Delta_k = a_{11}(a_{22}a_{33} - a_{32}a_{23}) + a_{13}(a_{21}a_{33} - a_{31}a_{22}),$
15: \hspace{1em} $B_k = b_1(a_{22}a_{33} - a_{32}a_{23}) + a_{22}(b_2a_{33} - b_3a_{22}),$
16: \hspace{1em} $G_k = \frac{b_1 - a_{11} B_k}{a_{13}}, C_k = \frac{b_2 - a_{21} B_k - a_{23} G_k}{a_{22}}, A_k = \frac{1}{C_k + G_k},$
17: \hspace{1em} $r_k = A_k \{ q_2 + B_k q_1 + C_k r_{k-2} + F_k q_3 + G_k r_{k-3} \},$
18: \hspace{1em} $x_k = A_k \{ C_k x_{k-2} + G_k x_{k-3} - (q_1 + B_k r_{k-2} + F_k r_{k-3}) \},$
19: \hspace{1em} if $||r_k|| \leq \epsilon$, then 
20: \hspace{1em} \hspace{1em} $x = x_k$, Stop.
21: \hspace{1em} \hspace{1em} \hspace{1em} end if
22: \hspace{1em} \hspace{1em} end for

4.2 Algorithm $A_4$

Algorithm $A_4$ is based on relation $A_4$. Its pseudo-code is as follows. For more details see [1] [2].
Algorithm 2 Algorithm $A_4$

1: Choose $x_0$ and $y$ such that $y \neq 0$,
2: Choose $\epsilon$ small and positive as a tolerance,
3: Set $r_0 = b - Ax_0$, $y_0 = y$,
4: for $k = 0, 1, \ldots, n$ do
5: $E_{k+1} = -\frac{(y_k, r_k)}{(y_k, Ar_k)}$, for $k \geq 1$, and $E_1 = 0$,
6: $B_{k+1} = -\frac{(y_k, Ar_k)}{(y_k, r_k)} - E_{k+1}(y_k, r_k)$,
7: $r_{k+1} = B_{k+1}x_k + E_{k+1}(y_k, r_k)$,
8: $x_{k+1} = A_{k+1}^{-1}[B_{k+1}x_k + E_{k+1}r_{k+1}]$,
9: if $\|r_{k+1}\| \neq \epsilon$ and $A_k \neq \epsilon$, then
10: $C_{k+1} = C_k - A_{k+1}^{-1}$.
11: end if
12: 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 in \[1, 2\].

Its pseudo-code is as follows.

Algorithm 3 Algorithm $A_5/B_{10}$

1: Choose $x_0$, $y$ and tolerance $\epsilon \geq 0$;
2: Set $r_0 = b - Ax_0$, $p_0 = r_0$, $y_0 = y$,
3: $A_1 = -\frac{(y_0, r_0)}{(y_0, Ar_0)}$, $C_0^1 = 1$,
4: $r_1 = r_0 + A_1Ar_0$, $x_1 = x_0 - A_1r_0$,
5: for $k = 1, 2, 3, \ldots, n$ 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: $p_k = r_k + D_{k-1}C_{k-1}^1p_{k-1}$
9: $A_{k+1} = -\frac{(y_k, p_k)}{(y_k, Ap_k)}$,
10: $r_{k+1} = r_k + A_{k+1}Ap_k$,
11: $x_{k+1} = x_k - A_{k+1}p_k$.
12: if $\|r_{k+1}\| \neq \epsilon$ and $A_k \neq \epsilon$, then
13: $C_k^1 = C_{k-1}^1$.
14: end if
15: end for
4.4 Algorithm $A_8/B_{10}$

The pseudo-code of $A_8/B_{10}$, [1, 2], 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: $A_{k+1} = -\frac{(y_k, r_k)}{(y_k, A_k z_k)}$,
7: $r_{k+1} = r_k + A_{k+1} z_k$,
8: $x_{k+1} = x_k - A_{k+1} z_k$.
9: if $||r_{k+1}|| \neq \epsilon$, then
10: $y_{k+1} = A^T y_k$,
11: $C_{k+1}^1 = \frac{1}{A_{k+1}}$,
12: $B_{k+1}^1 = -\frac{C_{k+1}^1 (y_{k+1}, r_{k+1})}{(y_k, A_k z_k)}$,
13: $z_{k+1} = B_{k+1}^1 z_k + C_{k+1}^1 r_{k+1}$.
14: end if
15: end for

5 Switching between algorithms to avoid breakdown

When a Lanczos-type algorithm fails, this 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 problems when used in another Lanczos-type algorithm, based on different recurrence relations. It is therefore obvious that one may consider switching to this other algorithm, when breakdown occurs. This allows the algorithm to work in a Krylov space with a different basis. It is therefore also possible to remedy breakdown by switching. Note that restarting the same algorithm after a pre-set number of iterations works well too, [15].
5.1 Switching strategies

Different strategies can be adopted for switching between two or more algorithms. These are as follows.

1. **ST1: Switching after breakdown:** Start a particular Lanczos algorithm until a breakdown occurs, then switch to another Lanczos algorithm, initializing the latter with the last iterate of the failed algorithm. We call this strategy ST1.

2. **ST2: Pre-emptive switching:** Run a Lanczos-type algorithm for a fixed number of iterations, halt it and then switch to another Lanczos-type algorithm, initializing it with the last iterate of the first algorithm. 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. **ST3: Breakdown monitoring:** Provided monotonicity of reduction in the absolute value of the denominators 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. Switch to another algorithm when the absolute value of any of these denominators drops below a certain level. This is strategy ST3.

5.2 A generic switching algorithm

Suppose we have a set of Lanczos-type algorithms and we want to switch from one algorithm to another using one of the above mentioned strategies ST1, ST2 or ST3.
Algorithm 5 Generic switching algorithm

1: Start the most stable algorithm, if known.
2: Choose a switching 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:     switch to another 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:     switch to another 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:     switch to another algorithm,
27:     initialize it with the current iterate,
28:     Go to 22.
29: end if
30: end if

However, it is important to mention that we have considered only ST2 in this paper. The convergence tolerance in all of the tests performed is \( \epsilon = 1.0e^{-0.13} \) and the number of iterations per cycle is fixed to 20.
5.2.1 Switching between algorithms \( A_4 \) and \( A_{12} \)

In the following, we start with either \( A_4 \) or \( A_{12} \), run it for a fixed number of iterations (cycle) chosen arbitrarily, before switching to the other. The results of this switching algorithm, are compared to those obtained with algorithms \( A_4 \) and \( A_{12} \) run individually. We are not changing any of the parameters involved in both algorithms. Details of \( A_4 \) can be found in [1].

**Algorithm 6 Switching between \( A_4 \) and \( A_{12} \)**

1: Choose \( x_0 \) and \( y \) such that \( y \neq 0 \),
2: set \( r_0 = b - Ax_0, y_0 = y \),
3: start either algorithm,
4: run current algorithm for a fixed number of iterations (a cycle) or until it halts;
5: if solution is obtained then
6: stop;
7: else
8: switch to the algorithm not yet run;
9: initialize it with the current iterate;
10: go to 4;
11: end if

**Remark:** Since restarting can be just as effective as switching, it is easier to implement a random choice between \( A_4 \) and \( A_{12} \) at the end of every cycle. Let heads be \( A_4 \) and tails be \( A_{12} \). At the toss of a coin, if it shows heads and the algorithm running in the last cycle was \( A_4 \), then the switch is a restart. If the coin shows tails then the switch is a “proper” switch, and \( A_{12} \) is called upon. In the numerical results presented below, this is what has been implemented. For more details about restarting see, [15].

5.2.2 Switching between \( A_4 \) and \( A_{5/B_{10}} \) algorithm

Start with \( A_{5/B_{10}} \), (details of \( A_{5/B_{10}} \) can be found in [1, 2]) do a few iterations and then switch to either \( A_4 \) or \( A_{5/B_{10}} \). The procedure is as Algorithm 4 below.
Algorithm 7 Switching between $A_4$ and $A_5/B_{10}$

1: Choose $x_0$ and $y$ such that $y \neq 0$;
2: set $r_0 = b - A_4x_0$, $y_0 = y$, $p_0 = r_0$;
3: start with either $A_4$ or $A_5/B_{10}$;
4: run it for a fixed number of iterations (cycles) or until it halts
5: if solution is obtained then
6: stop;
7: else
8: switch to either $A_4$ or $A_5/B_{10}$; initialize it with the last iterate of the
algorithm running in the last cycle;
9: go to 4;
10: end if

5.2.3 Switching between $A_4$ and $A_8/B_{10}$

Start with either $A_8/B_{10}$ (details of $A_8/B_{10}$ can be found in [1, 2]) or $A_4$; do a few iterations and then switch to either of them chosen randomly. If the chosen algorithm happens to be the same as the one running in the last cycle, then it is a case of restarting. Otherwise, it is switching. The algorithm is as follows.

Algorithm 8 Switching between $A_4$ and $A_8/B_{10}$

1: Choose $x_0$ and $y$ such that $y \neq 0$;
2: set $r_0 = b - A_4x_0$, $y_0 = y$, $p_0 = r_0$;
3: start either $A_4$ or $A_8/B_{10}$;
4: run it for a fixed number of iterations (cycle), or until it halts;
5: if solution is obtained then
6: stop;
7: else
8: switch to either $A_4$ or $A_8/B_{10}$;
9: initialize it with the iterate of the algorithm run in the last cycle;
10: go to 4.
11: end if

5.2.4 Switching between $A_5/B_{10}$ and $A_8/B_{10}$

Here again, switching and restarting are combined in a random way. Start with either $A_8/B_{10}$ or $A_5/B_{10}$. After a pre-set number of iterations (cycle), switch to either $A_5/B_{10}$ or $A_8/B_{10}$, randomly chosen. If the chosen algorithm to switch to is the same as the one running in the last cycle then we have a case of restarting; else it is switching. The algorithm is as follows.
Algorithm 9 Switching between $A_5/B_{10}$ and $A_8/B_{10}$

1: Choose $x_0$ and $y$ such that $y \neq 0$;
2: set $r_0 = b - Ax_0$, $y_0 = y$, $z_0 = r_0$;
3: start either $A_8/B_{10}$ or $A_5/B_{10}$;
4: run it for a fixed number of iterations;
5: if solution is not found then
6: halt current algorithm;
7: switch to either $A_5/B_{10}$ or $A_8/B_{10}$;
8: initialize it with the last iterate of the algorithm running in the last cycle;
9: go to 4;
10: else
11: solution found; stop;
12: end if

5.2.5 Numerical results

Algorithms 1, 2, 3, 4, [1, 2, 14] and Algorithms 6, 7, 8 and 9, [14] have been implemented in Matlab and applied to a number of small to medium size problems. The test problems we have used arise in the 5-point discretisation of the operator $-\partial^2_{xx} - \partial^2_{yy} + \gamma \frac{\partial}{\partial x}$ on a rectangular region [1, 2]. Comparative results are obtained on instances of the problem $A x = b$ with $A$ and $b$ as below, and with dimensions of $A$ and $b$ ranging from $n = 10$ to $n = 100$.

$$A = \begin{pmatrix} B & -I & \cdots & \cdots & 0 \\ -I & B & -I & \cdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & -I & B \end{pmatrix},$$

with

$$B = \begin{pmatrix} 4 & \alpha & \cdots & \cdots & 0 \\ \beta & 4 & \alpha & \cdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \beta & 4 \end{pmatrix},$$

and $\alpha = -1 + \delta$, $\beta = -1 - \delta$. The parameter $\delta$ takes the values 0.0, 0.2, 5 and 8 respectively. The right hand side $b$ is taken to be $b = AX$, where $X = (1,1,\ldots,1)^T$, is the solution of the system. The dimension of $B$ is 10. When $\delta = 0$, the coefficient matrix $A$ is symmetric and the problem is easy to solve because the region is a regular mesh, [20]. For all other values of $\delta$,
the matrix $A$ is non-symmetric and the problem is comparatively harder to solve as the region is not a regular mesh.

5.2.6 Numerical results

Results obtained with Algorithms 1, 2, 3, 4, and Algorithms 6, 7, 8 and 9 on Beahoux-type problems of different dimensions, for different values of $\delta$ are presented in tables 1, 2, 3 and 4, below. Algorithms 1, 2, 3, 4, executed individually, could only solve problems of dimensions 40 or below. In contrast, the switching algorithms, Algorithms 6, 7, 8 and 9, solved all problems of dimensions up to 4000. These results show that the switching algorithms are far superior to any one of the algorithms considered individually. These echo those obtained by restarting the same algorithm after a predefined number of iterations, [15].

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

| Dim of Prob | Algorithm 6 | Algorithm 7 | Algorithm 8 | Algorithm 9 |
|-------------|-------------|-------------|-------------|-------------|
| $n$         | $\|r_n\|$   | $f(s)$      | $\|r_n\|$   | $f(s)$      | $\|r_n\|$   | $f(s)$      | $\|r_n\|$   | $f(s)$      |
| 20          | 5.5067e-014 | 0.0012      | 3.8545e-014 | 0.0010      | 7.4781e-014 | 0.0049      | 8.0536e-014 | 0.0018      |
| 40          | 7.5417e-014 | 0.0041      | 4.5076e-014 | 0.0053      | 8.9288e-014 | 0.0057      | 7.2481e-014 | 0.0049      |
| 60          | 9.6638e-014 | 0.0057      | 2.5330e-014 | 0.0057      | 7.5107e-014 | 0.0085      | 5.8162e-014 | 0.0067      |
| 80          | 9.9082e-014 | 0.0075      | 6.0185e-014 | 0.0071      | 7.8666e-014 | 0.0088      | 5.7206e-014 | 0.0101      |
| 100         | 2.1487e-014 | 0.0095      | 2.5839e-014 | 0.0078      | 8.0262e-014 | 0.0098      | 8.137e-014  | 0.0100      |
| 200         | 7.4236e-014 | 0.0723      | 9.0967e-014 | 0.0159      | 7.8945e-014 | 0.0337      | 9.1836e-014 | 0.0352      |
| 400         | 7.7419e-014 | 0.0661      | 8.5151e-014 | 0.0216      | 7.9148e-014 | 0.2243      | 9.4697e-014 | 0.2415      |
| 600         | 9.0290e-014 | 0.0794      | 7.9747e-014 | 0.4735      | 9.9269e-014 | 1.9625      | 9.4307e-014 | 0.7475      |
| 800         | 9.2116e-014 | 0.5600      | 9.5227e-014 | 0.9395      | 7.7291e-014 | 3.0326      | 7.7356e-014 | 1.4319      |
| 1000        | 8.8493e-014 | 0.8509      | 8.6285e-014 | 2.0339      | 9.9148e-014 | 4.4347      | 9.4512e-014 | 2.3984      |
| 2000        | 9.7243e-014 | 6.8707      | 9.1973e-014 | 8.6384      | 8.3133e-014 | 12.6966     | 9.1395e-014 | 12.8606     |
| 3000        | 9.6903e-014 | 9.8130      | 7.7507e-014 | 16.5795     | 9.7827e-014 | 22.3836     | 8.2725e-014 | 22.3386     |
| 4000        | 9.2641e-014 | 17.4673     | 8.9681e-014 | 23.7695     | 9.8483e-014 | 44.4430     | 9.7911e-014 | 44.4567     |

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

| Dim of Prob | Algorithm 6 | Algorithm 7 | Algorithm 8 | Algorithm 9 |
|-------------|-------------|-------------|-------------|-------------|
| $n$         | $\|r_n\|$   | $f(s)$      | $\|r_n\|$   | $f(s)$      | $\|r_n\|$   | $f(s)$      | $\|r_n\|$   | $f(s)$      |
| 20          | 6.0041e-014 | 0.0044      | 1.5104e-014 | 0.0029      | 1.9614e-014 | 0.0056      | 8.0536e-014 | 0.0044      |
| 40          | 1.9968e-014 | 0.0064      | 4.6814e-014 | 0.0068      | 3.4094e-014 | 0.0089      | 7.3581e-014 | 0.0100      |
| 60          | 6.0758e-014 | 0.0134      | 6.3548e-014 | 0.0104      | 2.5927e-014 | 0.0113      | 9.6936e-014 | 0.0262      |
| 80          | 8.8550e-014 | 0.0159      | 9.5838e-014 | 0.0108      | 8.4187e-014 | 0.0120      | 7.9744e-014 | 0.0209      |
| 100         | 5.8026e-014 | 0.0144      | 6.6962e-014 | 0.0151      | 7.8898e-014 | 0.0126      | 7.5236e-014 | 0.0273      |
| 200         | 9.0970e-014 | 0.0213      | 9.8054e-014 | 0.0353      | 8.6331e-014 | 0.0333      | 8.1289e-014 | 0.0352      |
| 400         | 6.5595e-014 | 0.0748      | 9.3591e-014 | 0.1054      | 6.6606e-014 | 0.1875      | 8.4316e-014 | 0.2315      |
| 600         | 9.6153e-014 | 0.1802      | 8.8169e-014 | 0.0668      | 8.5135e-014 | 0.6751      | 5.9937e-014 | 0.7475      |
| 800         | 9.8605e-014 | 0.5922      | 8.8396e-014 | 0.9088      | 8.2550e-014 | 1.1430      | 7.0295e-014 | 1.4319      |
| 1000        | 9.7823e-014 | 0.8222      | 7.7896e-014 | 1.3020      | 7.4549e-014 | 2.1902      | 7.7294e-014 | 2.3894      |
| 2000        | 9.7953e-014 | 4.3416      | 9.4241e-014 | 4.7668      | 9.5282e-014 | 10.5976     | 9.1570e-014 | 8.5787      |
| 3000        | 8.7448e-014 | 10.0267     | 9.0631e-014 | 12.1561     | 9.9008e-014 | 25.1773     | 9.2866e-014 | 21.7380     |
| 4000        | 5.8412e-014 | 12.9309     | 9.7580e-014 | 23.3993     | 9.9276e-014 | 38.9001     | 9.7911e-014 | 39.0161     |
6 Conclusion

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 5.2.5 with dimensions ranging from 20 to 4000. The results are compared against those obtained by the switching algorithms, Algorithms 6, 7, 8 and 9 on the same problems. These results show that $A_4$, $A_{12}$, $A_5/B_{10}$ and $A_8/B_{10}$ are not as robust as the switching algorithms. In fact, individual algorithms solved only problems of dimension $n \leq 40$ and that with a poor accuracy. The switching algorithms, however, solved them all with a higher precision. The numerical evidence is strongly in favour of switching.

Based on the above results, it is clear that switching is an effective way to deal with the breakdown in Lanczos-type algorithms. It is also clear that
the switching algorithms are more efficient particularly for large dimension problems.

The cost of switching, in terms of CPU time, in ST2 at least, is not substantial, compared to that of the individual algorithms. It is also quite easy to see that it would not be substantial in ST1 since the cost would be similar to that of ST2. Even in the case of monitoring the coefficients that can vanish, i.e. ST3, the cost should only be that of a test of the form:

\[
\text{if } |\text{denominator value}| \leq \text{tolerance then stop.}
\]

We have not measured its impact on the overall computing time, but it should not be excessive. This means that switching strategies are worthwhile considering to enhance the efficiency of Lanczos-type algorithms and not just their robustness.

Having said that, further research and experimentation are necessary, particularly on the very large scale instances of SLE’s, to establish the superiority of switching algorithms against the state-of-the-art Lanczos-type algorithms with in-built precautions to avoid breakdown such as MRZ and BSMRZ, \[1, 7, 8, 11\]. Note that these algorithms are attractive for other reasons too, namely their simplicity and easy implementation. This is the subject of on-going research work.

References

[1] C. Baheux. *Algorithmes d’implémentation de la méthode de Lanczos*. PhD thesis, University of Lille 1, France, 1994.

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

[3] C. Brezinski. *Pade-Type Approximation and General Orthogonal Polynomials*, Internat. Ser. Nuner. Math. 50. Birkhäuser, Basel, 1980.

[4] C. Brezinski and H. Sadok. Lanczos-type algorithms for solving systems of linear equations. *Applied Numerical Mathematics*, 11:443–473, 1993.

[5] C. Brezinski and M. R. Zaglia. Treatment of near-breakdown in the CGS algorithm. *Numer. Algorithms Mathematic*, 7:33–73, 1994.

[6] C. Brezinski and M. R. Zaglia. Breakdowns in the implementation of Lanczos method for solving linear systems. *Comput. Math. Appl.*, 33:31–44, 1997.
[7] C. Brezinski, M. R. Zaglia, and H. Sadok. Avoiding breakdown and near-breakdown in Lanczos type algorithms. *Numerical Algorithms*, 1:261–284, 1991.

[8] C. Brezinski, M. R. Zaglia, and H. Sadok. numeralgo/na1, 1991. http://www.netlib.org/cgi-bin/search.pl.

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

[10] C. Brezinski, M. R. Zaglia, and H. Sadok. A Breakdown-free Lanczos type algorithm for solving linear systems. *Numerische Mathematik*, 63:29–38, 1992.

[11] C. Brezinski, M. R. Zaglia, and H. Sadok. New look-ahead Lanczos-type algorithms for linear systems. *Numerische Mathematik*, 83:53–85, 1999.

[12] C. Brezinski, M. R. Zaglia, and H. Sadok. A review of formal orthogonality in Lanczos-based methods. *Journal of Computational and Applied Mathematics*, 140:81–98, 2002.

[13] G. Cybenko. An explicit formula for Lanczos polynomials. *Linear Algebra Appl.*, 88/89:99–115, 1987.

[14] M. Farooq. *New Lanczos-type Algorithms and their Implementation*. PhD thesis, University of Essex, UK, 2011.

[15] M. Farooq and A. Salhi. A Preemptive Restarting Approach to Beating the Inherent Instability of Lanczos-type Algorithms. 2012. To appear in IJST Transactions A-Science.

[16] 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. Comput.*, 14:137–158, 1993.

[17] P. R. Graves-Morris. A “Look-around Lanczos” algorithms for solving a system of linear equations. *Numer. Algorithms*, 15:247–274, 1997.

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

[19] C. Lanczos. Solution of systems of linear equations by minimized iteration. *Journal of the National Bureau of Standards*, 49:33–53, 1952.
[20] G. Meurant. *The Lanczos and conjugate gradient algorithms, From Theory to Finite Precision Computations.* SIAM, Philadelphia, 2006.

[21] B. N. Parlett and D. S. Scott. The Lanczos Algorithm With Selective Orthogonalization. *Mathematics of Computation*, 33:217–238, 1979.

[22] B. N. Parlett, D. R. Taylor, and Z. A. Liu. A Look-Ahead Lanczos Algorithm for Unsymmetric Matrices. *Mathematics of Computation*, 44:105–124, 1985.

Muhammad Farooq,
Department of Mathematics,
University of Peshawar,
Khyber Pakhtunkhwa, 25120, Pakistan
mfarooq@upesh.edu.pk, Tel: 00 92 91 9221038, Fax: 00 92 91 9216470

Abdellah Salhi,
Department of Mathematical Sciences,
The University of Essex, Wivenhoe Park,
Colchester CO4 3SQ, UK
as@essex.ac.uk, Tel: 00 44 1206 873022, Fax: 00 44 1206 873043