Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080509

A Switching Approach to Avoid Breakdown in Lanczos-Type Algorithms

Muhammad Farooq[1,*] *and Abdellah 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, United Kingdom

Received: 25 Jun. 2012, Revised: 19 Dec. 2012, Accepted: 19 Dec. 2012
Published online: 1 Sep. 2014

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 [1,2,3]. 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 useable 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, [5]. Other procedures for avoiding breakdown are also proposed in [1,4,6,7,8,9,10,11,12,13] and the references given there. 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,

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

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

Let $\mathbf{x}_0$ and $\mathbf{y}$ be two arbitrary vectors in R^n such that $\mathbf{y} \neq 0$. The Lanczos method, [14] consists in constructing a sequence of vectors $\mathbf{x}_k \in R^n$ defined as follows, [2,15]

$$\mathbf{x}_k - \mathbf{x}_0 \in K_k(A,\mathbf{r}_0) = span(\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0), \tag{2}$$

* Corresponding author e-mail: mfarooq@upesh.edu.pk

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

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

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

Multiplying both sides by A and adding and subtracting $\mathbf{b}$ on the left hand side gives

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

From (3), the orthogonality condition gives

$(A^{T^i}\mathbf{y}, \mathbf{r}_k) = 0$, for $i = 0, \ldots, k-1$,

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

$$\begin{cases} \alpha_1(\mathbf{y}, A\mathbf{r}_0) + \cdots + \alpha_k(\mathbf{y}, A^k\mathbf{r}_0) = -(\mathbf{y}, \mathbf{r}_0), \\ \vdots \\ \alpha_1(A^{T^{k-1}}\mathbf{y}, A\mathbf{r}_0) + \cdots + \alpha_k(A^{T^{k-1}}\mathbf{y}, A^k\mathbf{r}_0) = -(A^{T^{k-1}}\mathbf{y}, \mathbf{r}_0). \end{cases} \tag{6}$$

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

$$P_k(x) = 1 + \alpha_1 x + \cdots + \alpha_k x^k, \tag{7}$$

then we can write from (5)

$$\mathbf{r}_k = P_k(A)\mathbf{r}_0. \tag{8}$$

The polynomials P_k are commonly known as the residual polynomials, [4]. Another interpretation of the P_k can be found in [16]. Moreover if we set $c_i = (A^{T^i}\mathbf{y}, \mathbf{r}_0) = (\mathbf{y}, A^i\mathbf{r}_0)$, $i = 0, 1, \ldots$, and if we define the linear functional c on the space of polynomials by

$$c(x^i) = c_i, \ i = 0, 1, \ldots, \tag{9}$$

c is completely determined by the sequence $\{c_k\}$ and c_k is said to be the moment of order k, [16]. Now, the system (6) can be written as

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

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, [16,17].

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 $R_{k-1} = \alpha_1 + \alpha_2 x + \ldots + \alpha_k x^{k-1}$. Replacing x by A in the expression of P_k and multiplying both sides by $\mathbf{r}_0$ and using (8), we get

$$\mathbf{r}_k = \mathbf{r}_0 + AR_{k-1}(A)\mathbf{r}_0,$$

which can be written as

$$b - A\mathbf{x}_k = b - A\mathbf{x}_0 + AR_{k-1}(A)\mathbf{r}_0,$$

$$-A\mathbf{x}_k = -A\mathbf{x}_0 + AR_{k-1}(A)\mathbf{r}_0,$$

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

$$\mathbf{x}_k = \mathbf{x}_0 - R_{k-1}(A)\mathbf{r}_0,$$

which shows that $\mathbf{x}_k$ can be computed from $\mathbf{r}_k$ without inverting A.

3. Formal orthogonal polynomials

The orthogonal polynomials P_k defined in the previous section are given by the determinantal formula, [4,18,19]

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

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 $c^{(1)}$, [2,4], 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, [2,4], because their highest degree coefficients are always 1, and are given by the following formula

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

$P_k^{(1)}(x)$ also exists if and only if the Hankel determinant $H_k^{(1)} \neq 0$, [2,4], 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, 4,6,18,19]. Some of these relations have been reviewed in [20] and studied in details in [1,19]. More of these relations have been studied in [3], leading to new Lanczos-type algorithms.

A Lanczos-type algorithm consists in computing P_k recursively, then $\mathbf{r}_k$ and finally $\mathbf{x}_k$ such that $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, without inverting A. In exact arithmetic, this should give the solution to the system $A\mathbf{x} = \mathbf{b}$ in at most n steps [6, 14], where n is the dimension of the system. For more details, see [4,12].

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}, [3]. For details on the derivation of the polynomial A_{12}, its coefficients and the algorithm itself, please refer to [3]. The pseudo-code of Algorithm A_{12} can be described as follows.

Algorithm 1 Algorithm A_{12}

Choose x_0 and y such that $y \neq 0$,
Choose ε small and positive, as a tolerance,
Set $r_0 = b - Ax_0$, $y_0 = y$, $p = Ar_0$, $p_1 = Ap$, $c_0 = (y, r_0)$,
$c_1 = (y, p)$, $c_2 = (y, p_1)$, $c_3 = (y, Ap_1)$, $\delta = c_1c_3 - c_2^2$,
$\alpha = \frac{c_0c_3 - c_1c_2}{\delta}$, $\beta = \frac{c_0c_2 - c_1^2}{\delta}$, $r_1 = r_0 - \frac{c_0}{c_1}p$, $x_1 = x_0 + \frac{c_0}{c_1}r_0$,
$r_2 = r_0 - \alpha p + \beta p_1$, $x_2 = x_0 + \alpha r_0 - \beta p$,
$y_1 = A^T y_0$, $y_2 = A^T y_1$, $y_3 = A^T y_2$.
for k = 3, 4,..., n **do**
  $y_{k+1} = A^T y_k$, $q_1 = Ar_{k-1}$, $q_2 = Aq_1$, $q_3 = Ar_{k-2}$,
  $a_{11} = (y_{k-2}, r_{k-2})$, $a_{13} = (y_{k-3}, r_{k-3})$, $a_{21} = (y_{k-1}, r_{k-2})$, $a_{22} = a_{11}$,
  $a_{23} = (y_{k-2}, r_{k-3})$, $a_{31} = (y_k, r_{k-2})$, $a_{32} = a_{21}$, $a_{33} = (y_{k-1}, r_{k-3})$,
  $s = (y_{k+1}, r_{k-2})$, $t = (y_k, r_{k-3})$, $F_k = -\frac{a_{11}}{a_{13}}$,
  $b_1 = -a_{21} - a_{23}F_k$, $b_2 = -a_{31} - a_{33}F_k$, $b_3 = -s - tF_k$,
  $\Delta_k = a_{11}(a_{22}a_{33} - a_{32}a_{23}) + a_{13}(a_{21}a_{32} - a_{31}a_{22})$,
  $B_k = \frac{b_1(a_{22}a_{33} - a_{32}a_{23}) + a_{13}(b_2a_{32} - b_3a_{22})}{\Delta_k}$,
  $G_k = \frac{b_1 - a_{11}B_k}{a_{13}}$, $C_k = \frac{b_2 - a_{21}B_k - a_{23}G_k}{a_{22}}$, $A_k = \frac{1}{C_k + G_k}$,
  $r_k = A_k\{q_2 + B_kq_1 + C_kr_{k-2} + F_kq_3 + G_kr_{k-3}\}$,
  $x_k = A_k\{C_kx_{k-2} + G_kx_{k-3} - (q_1 + B_kr_{k-2} + F_kr_{k-3})\}$,
  if $||r_k|| \leq \varepsilon$, **then**
    $x = x_k$, Stop.
  end if
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 [2,3].

Algorithm 2 Algorithm A_4

Choose x_0 and y such that $y \neq 0$,
Choose ε small and positive as a tolerance,
Set $r_0 = b - Ax_0$, $y_0 = y$,
for k = 0, 1,..., n **do**
  $E_{k+1} = -\frac{(y_k, r_k)}{(y_{k-1}, r_{k-1})}$, for $k \geq 1$, and $E_1 = 0$,
  $B_{k+1} = -\frac{(y_k, Ar_k) - E_{k+1}(y_k, r_{k-1})}{(y_k, r_k)}$,
  $A_{k+1} = \frac{1}{B_{k+1} + E_{k+1}}$,
  $x_{k+1} = A_{k+1}\{B_{k+1}x_k + E_{k+1}x_{k-1} - r_k\}$,
  $r_{k+1} = A_{k+1}\{Ar_k + B_{k+1}r_k + E_{k+1}r_{k-1}\}$.
  if $||r_{k+1}|| \leq \varepsilon$, **then**
    $y_{k+1} = A^T y_k$,
  end if
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 [2,3]. Its pseudo-code is as follows.

Algorithm 3 Algorithm A_5/B_{10}

Choose x_0, y and tolerance $\varepsilon \geq 0$;
Set $r_0 = b - Ax_0$, $p_0 = r_0$, $y_0 = y$,
$A_1 = -\frac{(y_0,r_0)}{(y_0,Ar_0)}$, $C_0^1 = 1$,
$r_1 = r_0 + A_1 A r_0$, $x_1 = x_0 - A_1 r_0$.
for k = 1,2,3,...,n **do**
  $y_k = A^T y_{k-1}$,
  $D_{k+1} = -\frac{(y_k,r_k)}{C_{k-1}^1 (y_k,p_{k-1})}$,
  $p_k = r_k + D_{k-1} C_{k-1}^1 p_{k-1}$
  $A_{k+1} = -\frac{(y_k,r_k)}{(y_k,Ap_k)}$,
  $r_{k+1} = r_k + A_{k+1} A p_k$,
  $x_{k+1} = x_k - A_{k+1} p_k$.
  if $r_{k+1} \neq \varepsilon$, then if $a_k \neq \varepsilon$, **then**
    $C_k^1 = \frac{C_{k-1}^1}{A_k}$.
  end if
end for

4.4. Algorithm A_8/B_{10}

The pseudo-code of A_8/B_{10}, [2,3], is as follows.

Algorithm 4 Algorithm A_8/B_{10}

Choose x_0 and y such that $y \neq 0$.
Set $r_0 = b - Ax_0$,
$z_0 = r_0$,
$y_0 = y$,
for $k = 0,1,2,\ldots,n$ **do**
  $A_{k+1} = -\frac{(\mathbf{y}_k,\mathbf{r}_k)}{(\mathbf{y}_k,A\mathbf{z}_k)}$,
  $r_{k+1} = r_k + A_{k+1} A z_k$,
  $x_{k+1} = x_k - A_{k+1} z_k$.
  if $||r_{k+1}|| \neq \varepsilon$, **then**
    $y_{k+1} = A^T y_k$,
    $C_{k+1}^1 = \frac{1}{A_{k+1}}$,
    $B_{k+1}^1 = -\frac{C_{k+1}^1 (y_{k+1},r_{k+1})}{(y_k,Az_k)}$,
    $z_{k+1} = B_{k+1}^1 z_k + C_{k+1}^1 r_{k+1}$.
  end if
end for

5. Switching between algorithms as a way to remedy the breakdown problem

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 the 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.

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

```
Start the most stable algorithm, if known.
Choose a switching strategy from ST1, ST2 or ST3.
if ST1 then
    Continue with current algorithm until it halts;
    if solution is obtained then
        Stop.
    else
        switch to another algorithm;
        initialize it with current iterate;
        Go to 4.
    end if
else if ST2 then
    Continue with current algorithm for a fixed
    number of iterations until it stops;
    if solution is obtained then
        Stop.
    else
        switch to another algorithm,
        initialize it with the current iterate,
        Go to 13.
    end if
else
    Continue with current algorithm and monitor
    certain parameters for breakdown, until it halts
    if solution is obtained then
        Stop.
    else
        switch to another algorithm,
        initialize it with current iterate,
        Go to 22.
    end if
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 $\varepsilon = 1.0E^{-013}$ 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}

```
Choose x_0 and y such that y ≠ 0,
set r_0 = b − Ax_0, y_0 = y,
start either algorithm,
run current algorithm for a fixed number of iterations
(a cycle) or until it halts;
if solution is obtained then
    stop;
else
    switch to the algorithm not yet run;
    initialize it with the current iterate;
    go to 4;
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, [21].

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}

```
Choose x_0 and y such that y ≠ 0;
set r_0 = b − Ax_0, y_0 = y, p_0 = r_0;
start with either A_4 or A_5/B_10;
run it for a fixed number of iterations (cycles) or until
it halts
if solution is obtained then
    stop;
else
    switch to either A_4 or A_5/B_10; initialize it with the
    last iterate of the algorithm running in the last cycle;
    go to 4;
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 $r_0 = b - Ax_0$, $y_0 = y$, $p_0 = r_0$;
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 a 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, [1,2,3] and Algorithms 6, 7, 8 and 9, [3] have 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 [1,2]. Comparative results on instances of the problem $A\mathbf{x} = \mathbf{b}$ with A and $\mathbf{b}$ as follows and with dimensions of A and $\mathbf{b}$ ranging from $n = 20$ to $n = 4000$.

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

with

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

and $\alpha = -1 + \delta$, $\beta = -1 - \delta$. The parameter δ takes the values 0.0, 0.2, 5 and 8 respectively. The right hand side $\mathbf{b}$ is taken to be $\mathbf{b} = A\mathbf{X}$, where $\mathbf{X} = (1, 1, \ldots, 1)^T$, 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, [22]. 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.

5.2.6. Results of Algorithms 1, 2, 3, 4, and Algorithms 6, 7, 8 and 9 for different dimensions of Baheux-type problems for different values of δ

The results obtained with algorithms 1, 2, 3, 4, run individually and those obtained with the switching algorithms, Algorithms 6, 7, 8 and 9, for different values of δ on Baheux-type problems, [1,2], are recorded in the tables 1, 2, 3 and 4 below. The results show that the switching algorithms are far superior to any one of the algorithms considered individually.

5.2.7. Comments on numerical evidence

The numerical evidence is strongly in favour of switching. Individual algorithms have consistently performed worse except on the very low dimensional instances with $n \leq 40$. 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 very few of the considered problems if at all and with a very poor accuracy. The switching algorithms, however, solved them all with a higher precision.

Table 1 For $\delta = 0$

Dim of Prob	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		Algorithm 6		Algorithm 7		Algorithm 8		Algorithm 9	
n	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)
20	$3.8545E^{-014}$	0.0038	$1.7536E^{-013}$	0.0086	$2.5256E^{-014}$	0.0022	$1.7489E^{-014}$	0.0052	$5.5067E^{-014}$	0.0012	$3.8545E^{-014}$	0.0010	$7.4781E^{-014}$	0.0040	$8.0533E^{-014}$	0.0018
40	$5.3602E^{-011}$	0.0042	NaN		NaN		NaN		$7.5417E^{-014}$	0.0041	$4.5076E^{-014}$	0.0053	$8.9208E^{-014}$	0.0057	$7.2481E^{-014}$	0.0040
60	NaN		NaN		NaN		NaN		$9.6638E^{-014}$	0.0057	$2.5330E^{-014}$	0.0057	$7.5107E^{-014}$	0.0085	$5.8162E^{-014}$	0.0067
80	NaN		NaN		NaN		NaN		$9.9082E^{-014}$	0.0075	$6.0185E^{-014}$	0.0071	$7.0866E^{-014}$	0.0088	$5.7266E^{-014}$	0.0101
100	NaN		NaN		NaN		NaN		$2.1487E^{-014}$	0.0095	$2.5839E^{-014}$	0.0078	$8.0262E^{-014}$	0.0098	$8.1373E^{-014}$	0.0100
200	NaN		NaN		NaN		NaN		$7.4236E^{-014}$	0.0723	$9.9667E^{-014}$	0.0159	$7.9045E^{-014}$	0.0337	$9.1830E^{-014}$	0.0352
400	NaN		NaN		NaN		NaN		$7.7419E^{-014}$	0.0661	$8.5151E^{-014}$	0.2156	$9.7418E^{-014}$	0.2243	$9.4697E^{-014}$	0.2315
600	NaN		NaN		NaN		NaN		$9.0290E^{-014}$	0.0794	$7.9373E^{-014}$	0.4735	$9.9269E^{-014}$	1.9625	$9.4307E^{-014}$	0.7457
800	NaN		NaN		NaN		NaN		$9.2116E^{-014}$	0.5660	$9.5227E^{-014}$	0.9395	$7.7294E^{-014}$	3.0326	$7.7356E^{-014}$	1.4319
1000	NaN		NaN		NaN		NaN		$8.8463E^{-014}$	0.8509	$8.6238E^{-014}$	2.0539	$9.9181E^{-14}$	4.3479	$9.4512E^{-014}$	2.8984
2000	NaN		NaN		NaN		NaN		$9.7242E^{-014}$	4.8079	$9.1973E^{-014}$	8.6364	$8.1319E^{-014}$	12.8696	$9.1193E^{-014}$	12.8696
3000	NaN		NaN		NaN		NaN		$9.6993E^{-014}$	9.8130	$7.7507E^{-014}$	16.5795	$9.7827E^{-014}$	22.3386	$8.2725E^{-014}$	22.3386
4000	NaN		NaN		NaN		NaN		$9.2641E^{-014}$	17.4673	$8.9681E^{-014}$	23.7658	$9.8438E^{-014}$	44.4430	$9.7911E^{-014}$	44.4567

Table 2 For $\delta = 0.2$

Dim of Prob	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		Algorithm 6		Algorithm 7		Algorithm 8		Algorithm 9	
n	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)
20	$9.5014E^{-014}$	0.0044	NaN		$8.9070E^{-014}$	0.0030	NaN		$6.0041E^{-014}$	0.0044	$1.5104E^{-014}$	0.0029	$1.8618E^{-014}$	0.0056	$8.0533E^{-014}$	0.0044
40	NaN		NaN		NaN		NaN		$1.9868E^{-014}$	0.0064	$4.6814E^{-014}$	0.0068	$3.4094E^{-014}$	0.0089	$7.3581E^{-014}$	0.0100
60	NaN		NaN		NaN		NaN		$6.0788E^{-014}$	0.0134	$6.3548E^{-014}$	0.0104	$2.9827E^{-014}$	0.0113	$9.6583E^{-014}$	0.0262
80	NaN		NaN		NaN		NaN		$8.8550E^{-014}$	0.0159	$9.5483E^{-014}$	0.0108	$8.4187E^{-014}$	0.0120	$7.0744E^{-014}$	0.0269
100	NaN		NaN		NaN		NaN		$5.8020E^{-014}$	0.0144	$6.6962E^{-014}$	0.0151	$7.3889E^{-014}$	0.0126	$7.5236E^{-014}$	0.0273
200	NaN		NaN		NaN		NaN		$9.0970E^{-014}$	0.0213	$9.8054E^{-014}$	0.0353	$8.6331E^{-014}$	0.0313	$8.1282E^{-014}$	0.0352
400	NaN		NaN		NaN		NaN		$6.5593E^{-014}$	0.0748	$9.3591E^{-014}$	0.1054	$6.6660E^{-014}$	0.1875	$8.4316E^{-014}$	0.2315
600	NaN		NaN		NaN		NaN		$9.6153E^{-014}$	0.1802	$8.8169E^{-014}$	0.6066	$6.8135E^{-014}$	0.6751	$5.9937E^{-014}$	0.7457
800	NaN		NaN		NaN		NaN		$9.8605E^{-014}$	0.5922	$8.8399E^{-014}$	0.9088	$8.2550E^{-014}$	1.1436	$7.0295E^{-014}$	1.4319
1000	NaN		NaN		NaN		NaN		$9.7823E^{-014}$	0.8222	$7.7898E^{-014}$	1.3020	$7.4540E^{-014}$	2.1302	$7.7204E^{-014}$	2.8984
2000	NaN		NaN		NaN		NaN		$7.9753E^{-014}$	4.3416	$9.4241E^{-014}$	4.7668	$9.5282E^{-014}$	10.5976	$9.1570E^{-014}$	8.5787
3000	NaN		NaN		NaN		NaN		$8.7448E^{-014}$	10.0287	$9.6831E^{-014}$	12.1561	$9.9608E^{-014}$	25.1173	$9.2806E^{-014}$	21.7380
4000	NaN		NaN		NaN		NaN		$5.8412E^{-014}$	12.9390	$9.7580E^{-014}$	23.3993	$9.9270E^{-014}$	38.9051	$9.7911E^{-014}$	39.0164

Table 3 For $\delta = 5$

Dim of Prob	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		Algorithm 6		Algorithm 7		Algorithm 8		Algorithm 9	
n	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)
20	$3.6838E^{-013}$	0.0042	$1.5589E^{-013}$	0.0090	NaN		NaN		$2.5092E^{-014}$	0.0079	$9.1438E^{-014}$	0.0060	$3.5839E^{-014}$	0.0067	$8.0533E^{-014}$	0.0038
40	NaN		NaN		NaN		NaN		$7.4721E^{-014}$	0.0202	$1.9575E^{-014}$	0.0171	$8.9026E^{-014}$	0.0079	$2.9089E^{-014}$	0.0078
60	NaN		NaN		NaN		NaN		$9.1477E^{-014}$	0.0232	$5.4122E^{-014}$	0.0218	$3.2486E^{-014}$	0.0207	$5.6811E^{-014}$	0.0103
80	NaN		NaN		NaN		NaN		$2.5165E^{-014}$	0.0275	$7.9597E^{-014}$	0.0277	$7.7553E^{-014}$	0.0255	$5.7266E^{-014}$	0.0101
100	NaN		NaN		NaN		NaN		$8.9244E^{-014}$	0.0315	$8.4269E^{-014}$	0.0295	$3.2898E^{-014}$	0.0308	$8.1373E^{-014}$	0.0100
200	NaN		NaN		NaN		NaN		$8.5274E^{-014}$	0.0410	$8.3743E^{-014}$	0.0402	$4.5501E^{-014}$	0.0499	$9.1830E^{-014}$	0.0352
400	NaN		NaN		NaN		NaN		$9.5005E^{-014}$	0.0965	$3.3013E^{-014}$	0.2662	$4.3032E^{-014}$	0.1856	$9.4697E^{-014}$	0.2315
600	NaN		NaN		NaN		NaN		$9.3474E^{-014}$	0.2318	$2.7456E^{-014}$	0.7717	$8.1621E^{-014}$	0.6303	$9.4307E^{-014}$	0.7457
800	NaN		NaN		NaN		NaN		$7.2197E^{-014}$	0.6875	$9.3718E^{-014}$	0.8720	$9.2023E^{-014}$	1.0426	$7.7356E^{-014}$	1.4319
1000	NaN		NaN		NaN		NaN		$9.4690E^{-014}$	1.7006	$8.2225E^{-014}$	2.4118	$6.7618E^{-014}$	2.6779	$9.4512E^{-014}$	4.2678
2000	NaN		NaN		NaN		NaN		$7.0752E^{-014}$	9.2566	$8.8127E^{-014}$	6.9938	$4.6266E^{-014}$	11.2416	$9.1193E^{-014}$	11.0604
3000	NaN		NaN		NaN		NaN		$8.0276E^{-014}$	15.5897	$8.8194E^{-014}$	18.8125	$4.3762E^{-014}$	25.3675	$8.2725E^{-014}$	24.6007
4000	NaN		NaN		NaN		NaN		$9.7667E^{-014}$	29.7400	$8.9260E^{-014}$	30.4619	$8.5908E^{-014}$	42.5710	$9.7911E^{-014}$	41.5276

Table 4 For $\delta = 8$

Dim of Prob	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		Algorithm 6		Algorithm 7		Algorithm 8		Algorithm 9	
n	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)	$\|\|r_k\|\|$	T(s)
20	NaN		NaN		$8.0420E^{-014}$	0.0027	$9.2836E^{-014}$	0.0054	$8.1056E^{-014}$	0.0069	$8.7622E^{-017}$	0.0049	$3.1380E^{-014}$	0.0075	$8.0533E^{-014}$	0.0048
40	NaN		NaN		NaN		NaN		$9.1880E^{-014}$	0.0243	$8.1258E^{-014}$	0.0061	$8.2974E^{-014}$	0.0216	$2.9089E^{-014}$	0.0078
60	NaN		NaN		NaN		NaN		$7.0558E^{-014}$	0.0299	$8.3090E^{-014}$	0.0220	$9.9891E^{-014}$	0.0284	$5.6811E^{-014}$	0.0103
80	NaN		NaN		NaN		NaN		$9.8855E^{-014}$	0.0346	$8.5600E^{-014}$	0.0287	$9.7330E^{-014}$	0.0282	$5.7266E^{-014}$	0.0120
100	NaN		NaN		NaN		NaN		$8.7391E^{-014}$	0.0382	$8.7752E^{-014}$	0.0303	$8.5960E^{-014}$	0.0363	$8.1373E^{-014}$	0.0137
200	NaN		NaN		NaN		NaN		$9.5793E^{-014}$	0.0700	$4.3407E^{-014}$	0.0457	$9.8381E^{-014}$	0.0708	$9.1830E^{-014}$	0.0352
400	NaN		NaN		NaN		NaN		$6.0799E^{-014}$	0.1292	$9.6421E^{-014}$	0.2399	$9.6706E^{-014}$	0.3125	$9.4697E^{-014}$	0.2315
600	NaN		NaN		NaN		NaN		$9.6186E^{-014}$	0.3432	$8.5386E^{-014}$	0.5535	$8.9805E^{-014}$	0.9279	$9.4307E^{-014}$	0.7457
800	NaN		NaN		NaN		NaN		$8.8932E^{-014}$	0.6942	$3.1458E^{-014}$	1.3329	$7.5301E^{-014}$	1.0612	$7.7356E^{-014}$	1.4319
1000	NaN		NaN		NaN		NaN		$9.7821E^{-014}$	1.7060	$4.9703E^{-014}$	2.7150	$9.6384E^{-014}$	2.1978	$9.4512E^{-014}$	2.8984
2000	NaN		NaN		NaN		NaN		$7.3843E^{-014}$	11.2436	$8.1578E^{-014}$	13.0654	$8.2557E^{-014}$	10.7977	$9.1193E^{-014}$	13.2915
3000	NaN		NaN		NaN		NaN		$9.5905E^{-014}$	20.0131	$7.1928E^{-014}$	20.9822	$5.3725E^{-014}$	25.3714	$8.2725E^{-014}$	28.4232
4000	NaN		NaN		NaN		NaN		$2.1552E^{-014}$	31.2356	$9.4300E^{-014}$	40.2119	$9.3869E^{-014}$	40.1782	$9.7911E^{-014}$	45.1523

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.

6. Conclusion

The switching strategies seem to be more successful than individual algorithms in that they did not experience any breakdowns and they solved all problems, most of them in much shorter CPU time.

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, the cost should only be that of a test of the form:
if |denominator value| $\leq$ 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, [1,6,7,12]. 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'implementation 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] M. Farooq. *New Lanczos-type Algorithms and their Implementation*, PhD thesis, University of Essex, UK, (2011). http://serlib0.essex.ac.uk/record=b1754556.

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

[5] 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).

[6] C. Brezinski and Zaglia, M. R. Hybird procedures for solving linear systems, *Numerische Mathematik*, **67**, 1-19 (1994).

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

[8] 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).

[9] 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).

[10] 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).

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

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

[13] B. N. Parlett and D.S. Scott. The Lanczos Algorithm With Selective Orthogonaliztion, *Mathematics of Computation*, **33**, 217-238 (1979).

[14] C. Lanczos. Solution of systems of linear equations by minimized iteration, *Journal of the National Bureau of Standards*, **49**, 33-53 (1952).

[15] 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).

[16] G. Cybenko. An explicit formula for Lanczos polynomials", *Linear Algebra Appl.*, **88**, 99-115 (1987).

[17] C. Brezinski. *Padé-Type Approximation and General Orthogonal Polynomials*, *Internat. Ser. Nuner. Math.* 50. Birkhäuser, Basel, (1980).

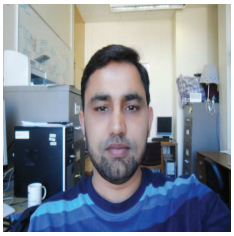
[18] C. Brezinski, M. R. Zaglia and H. Sadok. A Breakdown-free Lanczos type algorithm for solving linear systems, Numerische Mathematik, **63**, 29-38 (1992).

[19] C. Brezinski and M. R. Zaglia. Breakdowns in the implementation of Lanczos method for solving linear systems, *Comput. Math. Appl.*, **33**, 31-44 (1997).

[20] C. Brezinski and M. R. Zaglia. Treatment of near-breakdown in the CGS algorithm, *Numerical Algorithms*, **7**, 33-73 (1994).

[21] M. Farooq and A. Salhi. A Preemptive Restarting Approach to Beating the Inherent Instability of Lanczos-type Algorithms, *Iranian Journal of Science & Technology, Transactions A-Science*, **37(3.1)**, 349-358 (2013). http://ijsts.shirazu.ac.ir/?_action=articleInfo&article=1634&vol=142.

[22] G. Meurant. *The Lanczos and conjugate gradient algorithms, From Theory to Finite Precision Computations.* SIAM, Philadelphia, (2006).



Muhammad Farooq is currently Assistant Professor of Mathematics at the University of Peshawar, Pakistan. He was educated to degree level at the University of Peshawar, in Pakistan. He obtained his PhD recently on New Lanczos-type Algorithms and their Implementation under the supervision of Dr. Abdellah Salhi, from the University of Essex in Wivenhoe Park, Colchester, UK.

Abdellah Salhi is an expert in Operational Research with particular interests in the numerical aspects of optimisation algorithms. He is currently Senior Lecturer and Head of the Department of Mathematical Sciences at the University of Essex, UK. He was educated to degree level at the University of Constantine, in Algeria. He obtained his PhD on Karmarkars algorithm for Linear Programming from the University of Aston in Birmingham, UK. He has published over 60 research articles. He has recently introduced the Plant Propagation Algorithm for global optimisation, a heuristic-type algorithm inspired by the way strawberry plants propagate using runners.