View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

An International Journal
computers & mathematics with applications

PERGAMON Computers and Mathematics with Applications 42 (2001) 1239–1253
www.elsevier.nl/locate/camwa

Implementation of Different Computational Variations of Biconjugate Residual Methods

M. T. VESPUCCI
Facoltà di Ingegneria, Università degli Studi di Bergamo
Via G. Marconi, 5, 24044 Dalmine (BG), Italy

C. G. BROYDEN
Facoltà di Scienze, MM.FF.NN., University of Bologna
via Sacchi N.3, 47023 Cesena (FO), Italy

Abstract—In this paper, we describe the derivation of the biconjugate residual (BCR) method from the general framework of the Block-CG algorithm; we then introduce different versions of BCR and test their numerical performance. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords—Conjugate gradients, Biconjugate residuals, Krylov methods.

1. INTRODUCTION

We want to solve the system of linear equations

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

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a nonsingular (usually large and sparse) nonsymmetric matrix. We are interested in methods with the following properties:

(P1) the theoretically exact solution is obtained (in exact arithmetic) in a finite number of steps;

(P2) the coefficient matrix $\mathbf{A}$ is used along the solution process only for matrix-vector multiplications $\mathbf{A}\mathbf{v}$ or $\mathbf{A}^\top\mathbf{v}$, with $\mathbf{v}$ some vector in $\mathbb{R}^n$; indeed, we do not want to factorize $\mathbf{A}$ as the factors may not be sparse.

Methods with properties (P1) and (P2) can be obtained by defining a compound linear system related to (1), which we denote by

$$\mathbf{G}\mathbf{X} = \mathbf{H}, \tag{2}$$

where $\mathbf{G} \in \mathbb{R}^{p\times p}$ is symmetric and nonsingular and $\mathbf{X}, \mathbf{H} \in \mathbb{R}^{p\times r}$, and then computing the sequences $\{\mathbf{X}_j\}$ and $\{\mathbf{P}_j\}$, starting from $\mathbf{X}_1$ arbitrary in $\mathbb{R}^{p\times r}$ and from $\mathbf{P}_1 \in \mathbb{R}^{p\times r}$ suitably defined, so that

(C1) the matrix $\overline{\mathbf{P}}_i$ defined by

$$\overline{\mathbf{P}}_i = [\mathbf{P}_1 \quad \mathbf{P}_2 \quad \cdots \quad \mathbf{P}_i] \tag{3}$$

has full rank;

(C2) the residual matrix at $\mathbf{X}_{i+1}$ of the compound system (2), i.e., the matrix defined by

$$\mathbf{F}_{i+1} = \mathbf{G}\mathbf{X}_{i+1} - \mathbf{H}, \tag{4}$$

satisfies the so-called Galerkin condition

$$\overline{\mathbf{P}}_i^{\top} \mathbf{F}_{i+1} = \mathbf{O}. \tag{5}$$

Conditions (C1) and (C2) guarantee that, as i increases, the residual matrix $\mathbf{F}_{i+1}$ is squeezed into a sequence of subspaces of ever-decreasing dimension and is eventually forced to become null.

One algorithm that satisfies the above conditions is the Block-CG algorithm, introduced by O'Leary [1], which has the following structure.

BLOCK-CG (BLCG).

```
1. Select $\mathbf{X}_1 \in \mathbb{R}^{p\times r}$ arbitrary initial approximation of the solution. Select $\mathbf{K} \in \mathbb{R}^{p\times p}$
   symmetric and nonsingular. Compute $\mathbf{F}_1 = \mathbf{G}\mathbf{X}_1 - \mathbf{H}$ and $\mathbf{P}_1 = \mathbf{K}\mathbf{F}_1$. Set $i = 1$.
2. while $\mathbf{D}_i = \mathbf{P}_i^\top \mathbf{G}\mathbf{P}_i$ is nonsingular
   (a) compute $\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{P}_i^\top \mathbf{F}_i$
   (b) compute $\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G}\mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{P}_i^\top \mathbf{F}_i$
   (c) compute $\mathbf{P}_{i+1}$ by either of the following formulae:
         $\mathbf{P}_{i+1} = \left(\mathbf{I} - \mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{P}_i^\top\mathbf{G}\right)\mathbf{K}\mathbf{F}_{i+1},$    (6)
         $\mathbf{P}_{i+1} = \left(\mathbf{I} - \mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{P}_i^\top\mathbf{G} - \mathbf{P}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{P}_{i-1}^\top\mathbf{G}\right)\mathbf{K}\mathbf{G}\mathbf{P}_i$    (7)
   (d) set $i = i + 1$
3. endwhile
4. end BlCG.
```

Formula (6) is referred to as the *Hestenes-Stiefel (HS) choice* as from it may be obtained the two-term recurrence formulæ associated with Hestenes and Stiefel [2]. Formula (7) is referred to as the *Lanczos choice* as it leads, under certain circumstances, to the characteristic three-term recurrence formulæ developed by Lanczos [3].

The solution process does not terminate as long as the matrices $\mathbf{D}_i = \mathbf{P}_i^\top\mathbf{G}\mathbf{P}_i$ are nonsingular:

- if $\mathbf{D}_i$ is nonsingular for $i = 1, 2, \ldots, s$ and $\mathbf{P}_{s+1}$ is null, then $\mathbf{X}_{s+1}$ solves the matrix equation (2);
- if for some i, $\mathbf{D}_i$ becomes singular because $\mathbf{P}_i$ is rank-deficient (but not null), the algorithm is forced to stop, and normally a partial solution of the equation can be found;
- if $\mathbf{P}_i$ has full rank and $\mathbf{D}_i$ is singular because $\mathbf{G}$ is indefinite, then serious breakdown has occurred and rescue procedures, involving the so-called look-ahead versions of the algorithms, have to be initiated (see [4]).

O'Leary [1] showed that if both $\mathbf{G}$ and $\mathbf{K}$ are positive definite, then algorithms of both HS and Lanczos type are *robust*, that is, *breakdown-free*; if $\mathbf{G}$ is positive definite and $\mathbf{K}$ is indefinite, then algorithms of Lanczos type are *robust* (with possible temporary stagnation), while algorithms of HS type are *uncertain*, that is, *susceptible to breakdown* (with possible permanent stagnation).

Finally, note that a recursion formula is used in 2(b) to compute the residual matrix $\mathbf{F}_{i+1}$: in the computational algorithms derived later, the use of this recursion formula will make it possible to avoid one matrix-vector multiplication.

2. DERIVATION OF THE BCR ALGORITHM

The BCR algorithm was originally introduced by Hegedüs [5–7] and is obtained in the framework of the Block-CG algorithm by defining $p = 2n$, $r = 2$,

$$\mathbf{K} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^{-1} \\ \mathbf{A}^{-\top} & \mathbf{O} \end{bmatrix}, \tag{8}$$

$$\mathbf{G} = \begin{bmatrix} \mathbf{A}^{\top}\mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{A}\mathbf{A}^{\top} \end{bmatrix}, \tag{9}$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}^{\top}\mathbf{b} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}\mathbf{c} \end{bmatrix}, \tag{10}$$

and

$$\mathbf{X} = \begin{bmatrix} \mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathbf{z} \end{bmatrix}. \tag{11}$$

The residual matrices $\mathbf{F}_{i+1}$ generated by the BCR algorithm satisfy the relation

$$\mathbf{F}_{i+1} = \mathbf{K}^{-1}\mathbf{R}_{i+1}, \tag{12}$$

where $\mathbf{R}_{i+1}$ is defined by

$$\mathbf{R}_{i+1} = \mathbf{K}\mathbf{G}\mathbf{X}_{i+1} - \mathbf{K}\mathbf{H}, \tag{13}$$

and has the structure

$$\mathbf{R}_{i+1} = \begin{bmatrix} \mathbf{0} & \mathbf{s}_{i+1} \\ \mathbf{r}_{i+1} & \mathbf{0} \end{bmatrix}. \tag{14}$$

From (12)–(14), it follows that

$$\mathbf{F}_{i+1} = \begin{bmatrix} \mathbf{A}^{\top}\mathbf{r}_{i+1} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}\mathbf{s}_{i+1} \end{bmatrix}. \tag{15}$$

The residual vector $\mathbf{r}_{i+1}$ could be computed by $\mathbf{r}_{i+1} = \mathbf{A}\mathbf{x}_{i+1} - \mathbf{b}$, but is normally computed recursively to avoid unnecessary matrix-vector multiplications. The vector of *shadow residuals* $\mathbf{s}_{i+1} = \mathbf{A}^{\top}\mathbf{z}_{i+1} - \mathbf{c}$ has to be computed recursively as the sequence $\{\mathbf{z}_i\}$ is not computed at all. The initial value $\mathbf{s}_1$ of $\mathbf{s}_i$ is chosen arbitrarily.

2.1. The HS Version of BCR

The HS version of BCR is obtained by computing $\mathbf{P}_{i+1}$ by the two-term formula (6), which gives rise to block skew-diagonal matrices for all i; i.e.,

$$\mathbf{P}_{i+1} = \begin{bmatrix} \mathbf{0} & \mathbf{u}_{i+1} \\ \mathbf{v}_{i+1} & \mathbf{0} \end{bmatrix}. \tag{16}$$

The following algorithm is obtained.

Algorithm BCR2O.

Initial values: $\mathbf{x}_1$ and $\mathbf{s}_1$ arbitrary, $\mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$,

$\mathbf{u}_1 = \mathbf{s}_1$, $\mathbf{v}_1 = \mathbf{r}_1$, $\mathbf{w}_1 = \mathbf{A}\mathbf{u}_1$, and $\mathbf{y}_1 = \mathbf{A}^\top \mathbf{v}_1$.

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{w}_i^\top \mathbf{r}_i}{\mathbf{w}_i^\top \mathbf{w}_i} \right),$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \left(\frac{\mathbf{w}_i^\top \mathbf{r}_i}{\mathbf{w}_i^\top \mathbf{w}_i} \right),$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \left(\frac{\mathbf{y}_i^\top \mathbf{s}_i}{\mathbf{y}_i^\top \mathbf{y}_i} \right),$$

$$\mathbf{u}_{i+1} = \mathbf{s}_{i+1} - \mathbf{u}_i \left(\frac{\mathbf{w}_i^\top \mathbf{A}\mathbf{s}_{i+1}}{\mathbf{w}_i^\top \mathbf{w}_i} \right),$$

$$\mathbf{v}_{i+1} = \mathbf{r}_{i+1} - \mathbf{v}_i \left(\frac{\mathbf{y}_i^\top \mathbf{A}^\top \mathbf{r}_{i+1}}{\mathbf{y}_i^\top \mathbf{y}_i} \right),$$

$$\mathbf{w}_{i+1} = \mathbf{A}\mathbf{u}_{i+1},$$

$$\mathbf{y}_{i+1} = \mathbf{A}^\top \mathbf{v}_{i+1}.$$

Algorithm BCR2O requires four matrix-vector multiplications per iteration, namely, $\mathbf{A}\mathbf{s}_{i+1}$, $\mathbf{A}^\top\mathbf{r}_{i+1}$, $\mathbf{A}\mathbf{u}_{i+1}$, and $\mathbf{A}^\top\mathbf{v}_{i+1}$. This computational cost can be reduced by the following strategies:

(a) finding an *alternative formulation* of the HS version which requires less computation;

(b) generating the *auxiliary sequences* $\{\mathbf{w}_i\}$ and $\{\mathbf{y}_i\}$ recursively, so that the matrix-vector products $\mathbf{A}\mathbf{u}_i$ and $\mathbf{A}^\top\mathbf{v}_i$ do not need to be computed.

2.1.1. An alternative form of BCR-HS

The number of matrix-vector multiplications per iteration required by algorithm BCR2O can be reduced from four to three if the matrix $\mathbf{P}_{i+1}$ is computed by the alternative formula

$$\mathbf{P}_{i+1} = \mathbf{R}_{i+1} + \mathbf{P}_i \mathbf{C}_i^{-1} \mathbf{C}_{i+1}, \tag{17}$$

where

$$\mathbf{C}_j = \mathbf{R}_j^\top \mathbf{F}_j = \begin{bmatrix} 0 & \mathbf{r}_j^\top \mathbf{A}\mathbf{s}_j \\ \mathbf{r}_j^\top \mathbf{A}\mathbf{s}_j & 0 \end{bmatrix}, \qquad \text{for } j = i, i+1. \tag{18}$$

Formula (17) has been introduced in [1]. We note that formula (17) is the version of the original conjugate gradient method which is quoted in all textbooks. From (17), it follows that the computation of either $\mathbf{A}\mathbf{s}_{i+1}$ or $\mathbf{A}^\top\mathbf{r}_{i+1}$ can be avoided.

The following algorithm is obtained.

Algorithm BCR2A.

Initial values: $\mathbf{x}_1$ and $\mathbf{s}_1$ arbitrary, $\mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$,

$\mathbf{u}_1 = \mathbf{s}_1$, $\mathbf{v}_1 = \mathbf{r}_1$, $\mathbf{w}_1 = \mathbf{A}\mathbf{u}_1$, and $\mathbf{y}_1 = \mathbf{A}^\top \mathbf{v}_1$.

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{r}_i^\top \mathbf{A}\mathbf{s}_i}{\mathbf{w}_i^\top \mathbf{w}_i} \right),$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \left(\frac{\mathbf{r}_i^\top \mathbf{A}\mathbf{s}_i}{\mathbf{w}_i^\top \mathbf{w}_i} \right),$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \left(\frac{\mathbf{r}_i^\top \mathbf{A}\mathbf{s}_i}{\mathbf{y}_i^\top \mathbf{y}_i} \right),$$

$$\begin{aligned}
\mathbf{u}_{i+1} &= \mathbf{s}_{i+1} + \mathbf{u}_i \left(\frac{\mathbf{r}_{i+1}^{\mathsf{T}} \mathbf{A}\mathbf{s}_{i+1}}{\mathbf{r}_i^{\mathsf{T}} \mathbf{A}\mathbf{s}_i} \right), \\
\mathbf{v}_{i+1} &= \mathbf{r}_{i+1} + \mathbf{v}_i \left(\frac{\mathbf{r}_{i+1}^{\mathsf{T}} \mathbf{A}\mathbf{s}_{i+1}}{\mathbf{r}_i^{\mathsf{T}} \mathbf{A}\mathbf{s}_i} \right), \\
\mathbf{w}_{i+1} &= \mathbf{A}\mathbf{u}_{i+1}, \\
\mathbf{y}_{i+1} &= \mathbf{A}^{\mathsf{T}}\mathbf{v}_{i+1}.
\end{aligned}$$

We note that both $\mathbf{A}\mathbf{u}_{i+1}$ and $\mathbf{A}^{\mathsf{T}}\mathbf{v}_{i+1}$ still need to be computed.

2.1.2. The computation of the auxiliary sequences $\{\mathbf{w}_i\}$ and $\{\mathbf{y}_i\}$ in BCR-HS

We show now how the computation of the matrix-vector products $\mathbf{A}\mathbf{u}_{i+1}$ and $\mathbf{A}^{\mathsf{T}}\mathbf{v}_{i+1}$ can be avoided.

In Algorithm BCR2O, the vector $\mathbf{u}_{i+1}$ is computed by the formula

$$\mathbf{u}_{i+1} = \mathbf{s}_{i+1} - \mathbf{u}_i \left(\frac{\mathbf{w}_i^{\mathsf{T}} \mathbf{A}\mathbf{s}_{i+1}}{\mathbf{w}_i^{\mathsf{T}} \mathbf{w}_i} \right). \tag{19}$$

If we substitute (19) in $\mathbf{w}_{i+1} = \mathbf{A}\mathbf{u}_{i+1}$, we obtain the recursion formula for $\mathbf{w}_{i+1}$

$$\mathbf{w}_{i+1} = \mathbf{A}\mathbf{s}_{i+1} - \mathbf{w}_i \left(\frac{\mathbf{w}_i^{\mathsf{T}} \mathbf{A}\mathbf{s}_{i+1}}{\mathbf{w}_i^{\mathsf{T}} \mathbf{w}_i} \right), \tag{20}$$

with $\mathbf{w}_1 = \mathbf{A}\mathbf{u}_1$, which only depends on $\mathbf{w}_i$, the previous vector in the sequence, and on the vector $\mathbf{A}\mathbf{s}_{i+1}$. The matrix-vector product $\mathbf{A}\mathbf{u}_{i+1}$ may therefore be avoided.

Analogously, the vector $\mathbf{v}_{i+1}$ is computed in BCR2O by the formula

$$\mathbf{v}_{i+1} = \mathbf{r}_{i+1} - \mathbf{v}_i \left(\frac{\mathbf{y}_i^{\mathsf{T}} \mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}}{\mathbf{y}_i^{\mathsf{T}} \mathbf{y}_i} \right). \tag{21}$$

The substitution of (21) in $\mathbf{y}_{i+1} = \mathbf{A}^{\mathsf{T}}\mathbf{v}_{i+1}$ yields the recursion formula for $\mathbf{y}_{i+1}$

$$\mathbf{y}_{i+1} = \mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1} - \mathbf{y}_i \left(\frac{\mathbf{y}_i^{\mathsf{T}} \mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}}{\mathbf{y}_i^{\mathsf{T}} \mathbf{y}_i} \right), \tag{22}$$

with $\mathbf{y}_1 = \mathbf{A}^{\mathsf{T}}\mathbf{v}_1$, which only depends on $\mathbf{y}_i$, the previous vector in the sequence, and on the vector $\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$. The matrix-vector product $\mathbf{A}\mathbf{v}_{i+1}$ may therefore be avoided.

In order to test how the convergence behaviour of **BCR2O** is affected by the use of the recursion formulæ for $\mathbf{w}_{i+1}$ and $\mathbf{y}_{i+1}$, we have considered the following four versions.

- **BCR2O-a**: it computes both $\mathbf{w}_{i+1}$ and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires four matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$, $\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$, $\mathbf{A}\mathbf{u}_{i+1}$, and $\mathbf{A}^{\mathsf{T}}\mathbf{v}_{i+1}$).
- **BCR2O-b**: it computes $\mathbf{w}_{i+1}$ by matrix-vector multiplication and $\mathbf{y}_{i+1}$ by formula (22). It requires three matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$, $\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$, and $\mathbf{A}\mathbf{u}_{i+1}$).
- **BCR2O-c**: it computes $\mathbf{w}_{i+1}$ by formula (20) and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires three matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$, $\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$, and $\mathbf{A}^{\mathsf{T}}\mathbf{v}_{i+1}$).
- **BCR2O-d**: it computes $\mathbf{w}_{i+1}$ by (20) and $\mathbf{y}_{i+1}$ by (22). It requires two matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$ and $\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$).

With an analogous procedure applied to Algorithm BCR2A, we obtain the following four versions of the alternative form of Algorithm BCR-HS.

- **BCR2A-a**: it computes both $\mathbf{w}_{i+1}$ and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires three matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{u}_{i+1}$, $\mathbf{A}^\top\mathbf{v}_{i+1}$, and either $\mathbf{A}\mathbf{s}_{i+1}$ or $\mathbf{A}^\top\mathbf{r}_{i+1}$).
- **BCR2A-b**: it computes $\mathbf{w}_{i+1}$ by matrix-vector multiplication and $\mathbf{y}_{i+1}$ by the formula·

$$\mathbf{y}_{i+1} = \mathbf{A}^\top\mathbf{r}_{i+1} + \mathbf{y}_i\left(\frac{\mathbf{r}_{i+1}^\top\mathbf{A}\mathbf{s}_{i+1}}{\mathbf{r}_i^\top\mathbf{A}\mathbf{s}_i}\right). \tag{23}$$

  It requires two matrix-vector multiplications per iteration ($\mathbf{A}^\top\mathbf{r}_{i+1}$ and $\mathbf{A}\mathbf{u}_{i+1}$).
- **BCR2A-c**: it computes $\mathbf{w}_{i+1}$ by the formula

$$\mathbf{w}_{i+1} = \mathbf{A}\mathbf{s}_{i+1} + \mathbf{w}_i\left(\frac{\mathbf{r}_{i+1}^\top\mathbf{A}\mathbf{s}_{i+1}}{\mathbf{r}_i^\top\mathbf{A}\mathbf{s}_i}\right) \tag{24}$$

  and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires two matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$ and $\mathbf{A}^\top\mathbf{v}_{i+1}$).
- **BCR2A-d**: it computes $\mathbf{w}_{i+1}$ by (24) and $\mathbf{y}_{i+1}$ by (23). It requires two matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$ and $\mathbf{A}^\top\mathbf{r}_{i+1}$).

2.2. The Lanczos Version of BCR

The Lanczos version of BCR is obtained by computing $\mathbf{P}_{i+1}$ by the three-term formula (7), which gives rise to matrices whose form alternates from one iteration to the next, that is,

$$\mathbf{P}_{i+1} = \begin{bmatrix} \mathbf{0} & \mathbf{u}_{i+1} \\ \mathbf{v}_{i+1} & \mathbf{0} \end{bmatrix}, \quad \text{for } i \text{ even} \qquad \text{and} \qquad \mathbf{P}_{i+1} = \begin{bmatrix} \mathbf{u}_{i+1} & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_{i+1} \end{bmatrix}, \quad \text{for } i \text{ odd}.$$

It has been observed that when the sequence $\{\mathbf{P}_i\}$ is generated by a three-term recursion, the columns of the successive matrices $\mathbf{P}_j$ often tend rapidly towards linear dependence, unless column scaling is introduced. There is also the possibility of overflow and underflow. We have, therefore, defined the Lanczos version of BCR in terms of the matrices

$$\widetilde{\mathbf{P}}_{i+1} = \mathbf{P}_{i+1}\mathbf{B}_{i+1}^{-1}, \tag{25}$$

where $\mathbf{B}_{i+1}^2 = \mathbf{P}_{i+1}^\top\mathbf{P}_{i+1}$; that is,

$$\mathbf{B}_{i+1} = \begin{bmatrix} \|\mathbf{v}_{i+1}\|_2 & 0 \\ 0 & \|\mathbf{u}_{i+1}\|_2 \end{bmatrix}, \qquad \text{for } i \text{ even}, \tag{26}$$

and

$$\mathbf{B}_{i+1} = \begin{bmatrix} \|\mathbf{u}_{i+1}\|_2 & 0 \\ 0 & \|\mathbf{v}_{i+1}\|_2 \end{bmatrix}, \qquad \text{for } i \text{ odd}, \tag{27}$$

which imply that $\widetilde{\mathbf{P}}_{i+1}^\top\widetilde{\mathbf{P}}_{i+1} = \mathbf{I}$.

The following algorithm is obtained.

ALGORITHM BCR3O.

```
Initial values: $\tilde{\mathbf{u}}_0 = \tilde{\mathbf{v}}_0 = \mathbf{w}_0 = \mathbf{y}_0 = \mathbf{0}, \quad \sigma_0 = \tau_0 = 1,$
                $\mathbf{x}_1$ and $\mathbf{s}_1$ arbitrary, $\quad \mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b},$
                $\mathbf{u}_1 = \mathbf{s}_1, \quad \mathbf{v}_1 = \mathbf{r}_1,$
                $\tilde{\mathbf{u}}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \quad \tilde{\mathbf{v}}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|},$
                $\mathbf{w}_1 = \mathbf{A}\tilde{\mathbf{u}}_1, \quad \mathbf{y}_1 = \mathbf{A}^T\tilde{\mathbf{v}}_1,$
                $\sigma_1 = \mathbf{w}_1^T\mathbf{w}_1, \quad \tau_1 = \mathbf{y}_1^T\mathbf{y}_1.$
Recursions:     $\mathbf{x}_{i+1} = \mathbf{x}_i - \tilde{\mathbf{u}}_i\left(\frac{\mathbf{w}_i^T\mathbf{r}_i}{\sigma_i}\right),$
                $\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i\left(\frac{\mathbf{w}_i^T\mathbf{r}_i}{\sigma_i}\right),$
                $\mathbf{u}_{i+1} = \mathbf{y}_i - \tilde{\mathbf{u}}_i\left(\frac{\mathbf{w}_i^T\mathbf{A}\mathbf{y}_i}{\sigma_i}\right) - \tilde{\mathbf{u}}_{i-1}\left(\frac{\mathbf{w}_{i-1}^T\mathbf{A}\mathbf{y}_i}{\sigma_{i-1}}\right),$
                $\mathbf{v}_{i+1} = \mathbf{w}_i - \tilde{\mathbf{v}}_i\left(\frac{\mathbf{w}_i^T\mathbf{A}\mathbf{y}_i}{\tau_i}\right) - \tilde{\mathbf{v}}_{i-1}\left(\frac{\mathbf{y}_{i-1}^T\mathbf{A}^T\mathbf{w}_i}{\tau_{i-1}}\right),$
                $\tilde{\mathbf{u}}_{i+1} = \frac{\mathbf{u}_{i+1}}{\|\mathbf{u}_{i+1}\|},$
                $\tilde{\mathbf{v}}_{i+1} = \frac{\mathbf{v}_{i+1}}{\|\mathbf{v}_{i+1}\|},$
                $\mathbf{w}_{i+1} = \mathbf{A}\tilde{\mathbf{u}}_{i+1},$
                $\mathbf{y}_{i+1} = \mathbf{A}^T\tilde{\mathbf{v}}_{i+1},$
                $\sigma_{i+1} = \mathbf{w}_{i+1}^T\mathbf{w}_{i+1},$
                $\tau_{i+1} = \mathbf{y}_{i+1}^T\mathbf{y}_{i+1}.$
```

Algorithm BCR3O requires three matrix-vector multiplications per iteration ($\mathbf{A}\tilde{\mathbf{u}}_{i+1}$, $\mathbf{A}^T\tilde{\mathbf{v}}_{i+1}$, and either $\mathbf{A}^T\mathbf{w}_i$ or $\mathbf{A}\mathbf{y}_i$). This computational cost can be reduced by computing the sequences $\{\mathbf{w}_i\}$ and $\{\mathbf{y}_i\}$ recursively: we therefore consider the following versions of BCR3O.

- **BCR3O-a**: it computes both $\mathbf{w}_{i+1}$ and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires three matrix-vector multiplications per iteration ($\mathbf{A}\tilde{\mathbf{u}}_{i+1}$, $\mathbf{A}^T\tilde{\mathbf{v}}_{i+1}$, and either $\mathbf{A}^T\mathbf{w}_i$ or $\mathbf{A}\mathbf{y}_i$).
- **BCR3O-b**: it computes $\mathbf{w}_{i+1}$ by matrix-vector multiplication and $\mathbf{y}_{i+1}$ by the formula

$$\mathbf{y}_{i+1} = \left[\mathbf{A}^T\mathbf{w}_i - \mathbf{y}_i\left(\frac{\mathbf{w}_i^T\mathbf{A}\mathbf{y}_i}{\tau_i}\right) - \mathbf{y}_{i-1}\left(\frac{\mathbf{y}_{i-1}^T\mathbf{A}^T\mathbf{w}_i}{\tau_{i-1}}\right)\right]\frac{1}{\|\mathbf{v}_{i+1}\|}. \tag{28}$$

  It requires two matrix-vector multiplications per iteration ($\mathbf{A}^T\mathbf{w}_i$ and $\mathbf{A}\tilde{\mathbf{u}}_{i+1}$).
- **BCR3O-c**: it computes $\mathbf{w}_{i+1}$ by the formula

$$\mathbf{w}_{i+1} = \left[\mathbf{A}\mathbf{y}_i - \mathbf{w}_i\left(\frac{\mathbf{w}_i^T\mathbf{A}\mathbf{y}_i}{\sigma_i}\right) - \mathbf{w}_{i-1}\left(\frac{\mathbf{w}_{i-1}^T\mathbf{A}\mathbf{y}_i}{\sigma_{i-1}}\right)\right]\frac{1}{\|\mathbf{u}_{i+1}\|} \tag{29}$$

  and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires two matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{y}_i$ and $\mathbf{A}^T\tilde{\mathbf{v}}_{i+1}$).
- **BCR3O-d**: it computes $\mathbf{w}_{i+1}$ by (29) and $\mathbf{y}_{i+1}$ by (28). It requires two matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{y}_i$ and $\mathbf{A}^T\mathbf{w}_i$).

It has been shown [8] that the following relations hold:

$$\mathbf{w}_{i-1}^{\mathsf{T}} \mathbf{A} \mathbf{y}_i = \|\mathbf{v}_i\| \, \tau_i \tag{30}$$

and

$$\mathbf{y}_{i-1}^{\mathsf{T}} \mathbf{A}^{\mathsf{T}} \mathbf{w}_i = \|\mathbf{u}_i\| \, \sigma_i. \tag{31}$$

Relations (30) and (31) imply that the vectors $\mathbf{u}_{i+1}$ and $\mathbf{v}_{i+1}$ can be computed alternatively by

$$\mathbf{u}_{i+1} = \mathbf{y}_i - \tilde{\mathbf{u}}_i \left(\frac{\mathbf{w}_i^{\mathsf{T}} \mathbf{A} \mathbf{y}_i}{\sigma_i} \right) - \tilde{\mathbf{u}}_{i-1} \left(\frac{\|\mathbf{v}_i\| \, \tau_i}{\sigma_{i-1}} \right) \tag{32}$$

and

$$\mathbf{v}_{i+1} = \mathbf{w}_i - \tilde{\mathbf{v}}_i \left(\frac{\mathbf{w}_i^{\mathsf{T}} \mathbf{A} \mathbf{y}_i}{\tau_i} \right) - \tilde{\mathbf{v}}_{i-1} \left(\frac{\|\mathbf{u}_i\| \, \sigma_i}{\tau_{i-1}} \right); \tag{33}$$

the recursion formulæ for $\mathbf{w}_{i+1}$ and $\mathbf{y}_{i+1}$ then become

$$\mathbf{w}_{i+1} = \left[\mathbf{A} \mathbf{y}_i - \mathbf{w}_i \left(\frac{\mathbf{w}_i^{\mathsf{T}} \mathbf{A} \mathbf{y}_i}{\sigma_i} \right) - \mathbf{w}_{i-1} \left(\frac{\|\mathbf{v}_i\| \, \tau_i}{\sigma_{i-1}} \right) \right] \frac{1}{\|\mathbf{u}_{i+1}\|} \tag{34}$$

and

$$\mathbf{y}_{i+1} = \left[\mathbf{A}^{\mathsf{T}} \mathbf{w}_i - \mathbf{y}_i \left(\frac{\mathbf{w}_i^{\mathsf{T}} \mathbf{A} \mathbf{y}_i}{\tau_i} \right) - \mathbf{y}_{i-1} \left(\frac{\|\mathbf{u}_i\| \, \sigma_i}{\tau_{i-1}} \right) \right] \frac{1}{\|\mathbf{v}_{i+1}\|}. \tag{35}$$

Four alternative versions of the BCR-Lanczos algorithm have therefore been considered.

- **BCR3A-a**: it computes both $\mathbf{w}_{i+1}$ and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires three matrix-vector multiplications per iteration (either $\mathbf{A}\mathbf{s}_{i+1}$ or $\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$, $\mathbf{A}\tilde{\mathbf{u}}_{i+1}$, and $\mathbf{A}^{\mathsf{T}}\tilde{\mathbf{v}}_{i+1}$).
- **BCR3A-b**: it computes $\mathbf{w}_{i+1}$ by matrix-vector multiplication and $\mathbf{y}_{i+1}$ by formula (35). It requires two matrix-vector multiplications per iteration ($\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$ and $\mathbf{A}\tilde{\mathbf{u}}_{i+1}$).
- **BCR3A-c**: it computes $\mathbf{w}_{i+1}$ by formula (34) and $\mathbf{y}_{i+1}$ by matrix-vector multiplication. It requires two matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$ and $\mathbf{A}^{\mathsf{T}}\tilde{\mathbf{v}}_{i+1}$).
- **BCR3A-d**: it computes $\mathbf{w}_{i+1}$ by (34) and $\mathbf{y}_{i+1}$ by (35). It requires two matrix-vector multiplications per iteration ($\mathbf{A}\mathbf{s}_{i+1}$and $\mathbf{A}^{\mathsf{T}}\mathbf{r}_{i+1}$).

In all these versions, $\mathbf{u}_{i+1}$ is computed by (32) and $\mathbf{v}_{i+1}$ is computed by (33).

3. NUMERICAL COMPARISONS

3.1. Test Problems

The 16 versions of BCR introduced in the previous sections have been tested on problems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{0}. \tag{36}$$

Indeed, if $\mathbf{A}$ is nonsingular, the unique solution $\mathbf{x}^*$ of (36) is the null vector, which implies that the error vector $\mathbf{e}_i$ at the approximate solution $\mathbf{x}_i$, defined by

$$\mathbf{e}_i = \mathbf{x}_i - \mathbf{x}^*,$$

equals $\mathbf{x}_i$. Note that the error vector $\mathbf{e}_i$ cannot be monitored if the test problem is constructed by the following procedure:

1. give the coefficient matrix $\mathbf{A}$;
2. assign a nonzero solution vector $\mathbf{x}^* \neq \mathbf{0}$;
3. *compute in finite arithmetic* the right-hand side $\widehat{\mathbf{b}} = \mathbf{A}\mathbf{x}^*$;
4. define the test problem: *find* $\mathbf{x}$ *such that*

$$\mathbf{A}\mathbf{x} = \widehat{\mathbf{b}}. \tag{37}$$

Indeed, in general the right-hand side $\widehat{\mathbf{b}}$ computed in finite arithmetic differs from the vector $\mathbf{b}$ which would be obtained if the product $\mathbf{Ax}^*$ were performed in exact arithmetic; therefore, the solution of the test problem (37) is actually unknown, and so is the error $\mathbf{e}_i$ of the approximate solution.

The coefficient matrices we considered, all from the Harwell-Boeing collection, are listed in Table 1, where dimensions and condition number estimates (obtained by the MATLAB routine *cond*) are also given. For all problems, the starting vector was $\mathbf{x}_1^{\mathrm{T}} = (1, 1, \dots, 1)$.

Table 1.

Test Matrix	n	Condition Number
JPWH991	991	$1 \cdot 10^2$
PDE9511	961	$1 \cdot 10^2$
FS5411	541	$4 \cdot 10^3$
IMPCOLE	225	$7 \cdot 10^6$
SHL400	663	$8 \cdot 10^8$
ARC130	130	$6 \cdot 10^{10}$

3.2. Estimation of the Condition Number of A

Along the solution process, any time a matrix-vector multiplication $\mathbf{Aq}$, or $\mathbf{A}^{\mathrm{T}}\mathbf{q}$, with $\mathbf{q}$ some vector, is performed, the ratio $\rho = \|\mathbf{Aq}\|_2/\|\mathbf{q}\|_2$, or $\rho = \|\mathbf{A}^{\mathrm{T}}\mathbf{q}\|_2/\|\mathbf{q}\|_2$, is computed and the current maximum and minimum ratios are then stored in the variables $\rho_{\max}$ and $\rho_{\min}$, respectively: the ratio $\rho_{\max}/\rho_{\min}$ is an estimate (a lower bound in fact) of the condition number of $\mathbf{A}$.

3.3. Indicators of Performance

At each iteration of the solution process, the following indicators of performance have been computed:

- the ratio err_i between the Euclidean norm of the error in the approximate solution $\mathbf{x}_i$ and the Euclidean norm of the initial approximation

$$err_i = \frac{\|\mathbf{x}_i\|_2}{\|\mathbf{x}_1\|_2}; \tag{38}$$

- the ratio re_i between the Euclidean norm of the *exactly* computed residual at $\mathbf{x}_i$ and the Euclidean norm of the initial residual

$$re_i = \frac{\|\mathbf{Ax}_i\|_2}{\|\mathbf{Ax}_1\|_2}; \tag{39}$$

- the ratio ri_i between the Euclidean norm of the *iteratively* computed residual at $\mathbf{x}_i$ and the Euclidean norm of the initial residual

$$ri_i = \frac{\|\mathbf{r}_i\|_2}{\|\mathbf{Ax}_1\|_2}; \tag{40}$$

- the estimate $c\mathbf{A}_i$ of the condition number of $\mathbf{A}$

$$c\mathbf{A}_i = \frac{(\rho_{\max})_i}{(\rho_{\min})_i}, \tag{41}$$

where $(\rho_{\max})_i$ and $(\rho_{\min})_i$ are, respectively, the maximum and the minimum among all ratios $\|\mathbf{A}\mathbf{q}\|_2/\|\mathbf{q}\|_2$ and $\|\mathbf{A}^T\mathbf{q}\|_2/\|\mathbf{q}\|_2$, with $\mathbf{q}$ some vector, computed in the first i iterations of the solution process;

- the product

$$re\_c\mathbf{A}_i = c\mathbf{A}_i \cdot re_i \tag{42}$$

to be compared with err_i.

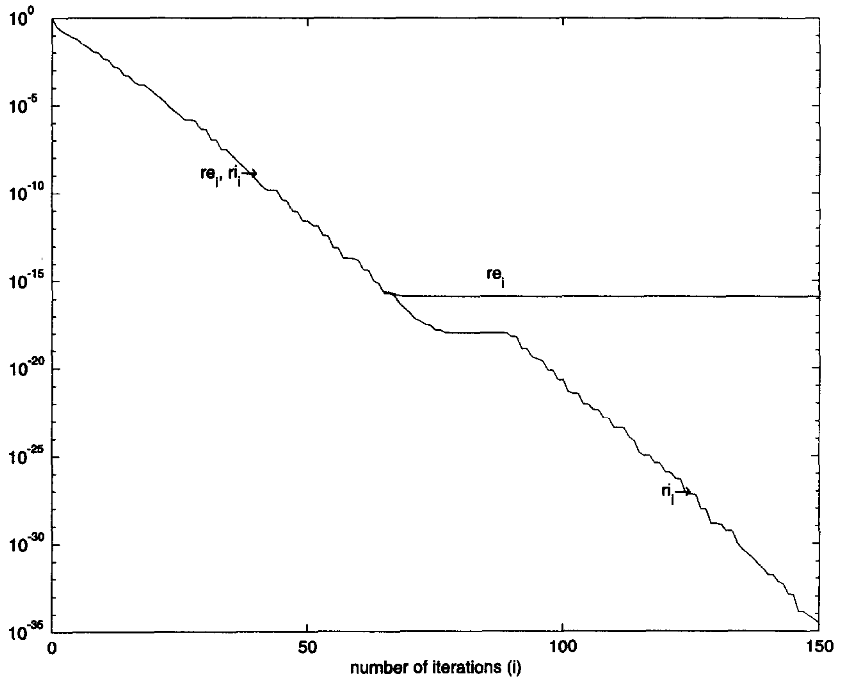

Figure 1. Matrix FS5411 ($n = 541$), Algorithm BCR2A-b, behaviour of re_i and ri_i.

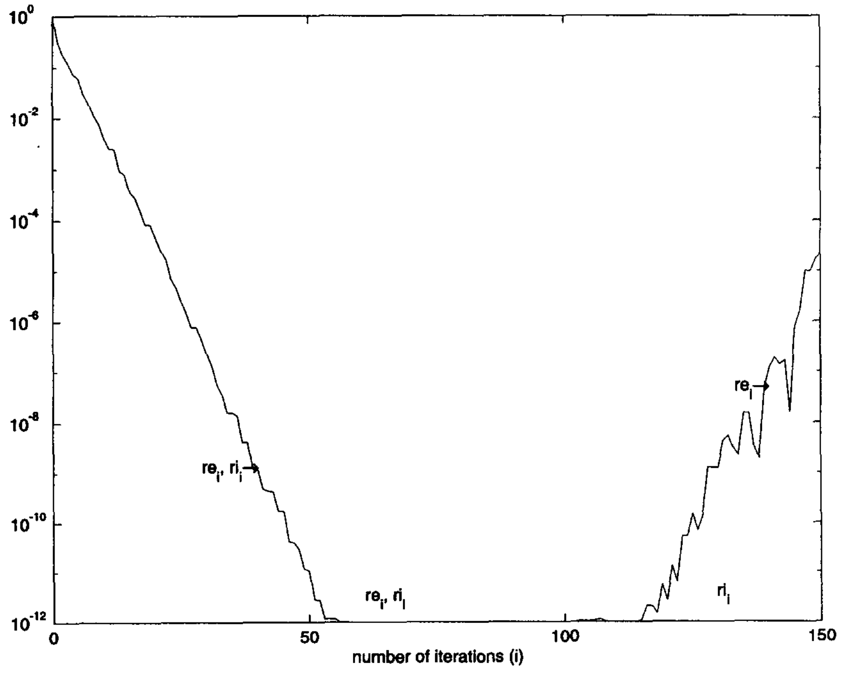

Figure 2. Matrix FS5411 ($n = 541$), Algorithm BCR3O-c, behaviour of re_i and ri_i.

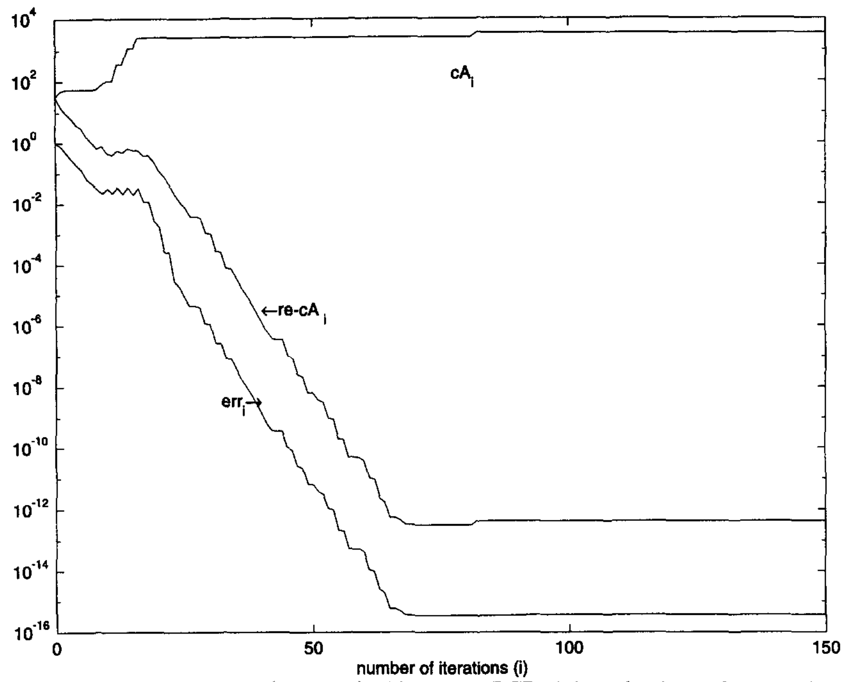

Figure 3. Matrix FS5411 (n = 541), Algorithm BCR2A-b, behaviour of err_i, cA_i, and re_cA_i.

3.4. Numerical Results

We first describe the results obtained by solving the test problem (36) where $\mathbf{A}$ is the matrix FS5411 ($n = 541$):

- all HS versions required about 70 iterations to get the ratio re_i (39) down to 10^{-16}; in the following iterations, the ratio re_i remained constant while the ratio ri_i (whose value equals the value of re_i in the first 70 iterations) continued decreasing; in Figure 1, the graphs of the ratios re_i and ri_i related to the version BCR2A-b are presented;
- all Lanczos versions required about 50 iterations to get the ratio re_i down to 10^{-12}; in the first 50 iterations, the value of ri_i was equal to that of re_i; in the following iterations, two different behaviours were observed:
  - in versions a and b of both BCR3O and BCR3A, the ratios re_i and ri_i remained constant at the value 10^{-12};
  - in versions c and d of both BCR3O and BCR3A, the ratio ri_i remained constant at the value 10^{-12}, while the ratio re_i started increasing again at iteration 105 (see Figure 2, which relates to BCR3O-c);
- all versions of BCR required 20 iterations for the condition number estimate $c\mathbf{A}_i$ to be equal to the estimate computed by the MATLAB routine *cond*; the product $re\_c\mathbf{A}_i$ was in all cases an upper bound to the ratio err_i (see Figure 3, which relates to BCR2A-b).

With problems JPWH991 (n = 991) and PDE9511 (n = 961) (both low condition number problems), the convergence behaviour in terms of the ratio re_i was similar to that of FS5411, and therefore, we do not present the results in detail: we just mention that for both problems all versions required about 750 iterations to reduce the ratio re_i down to 10^{-14}.

On low condition number problems, comparable convergence behaviours were obtained by all versions of BCR, with the exception of BCR3O-c/d and BCR3A-c/d: therefore, we can conclude that for these problems, the most efficient versions are BCR2O-d, BCR2A-b/c/d, BCR3O-b, and BCR3A-b, since they only require two matrix-vector multiplications per iteration.

Let us now consider the results obtained with the matrices SHL400 (n = 663) and ARC130 (n = 130). Among the versions requiring three matrix-vector multiplications per iteration, BCR2A-a showed the fastest convergence, in terms of re_i, on both problems. Among the versions requiring two matrix-vector multiplications per iteration, BCR2A-b showed the fastest convergence, in terms of re_i, on both problems. In Figure 4, the curves re_i related to BCR2O-a (four matrix-vector multiplications per iteration), BCR2A-a, and BCR2A-b are reported for matrix SHL400, and in Figure 5, the curves re_i related to BCR2O-a, BCR2A-a, and BCR2A-b are reported for matrix ARC130. These comparisons again suggest that a good convergence behaviour can be obtained at the cost of only two matrix-vector multiplications per iteration.

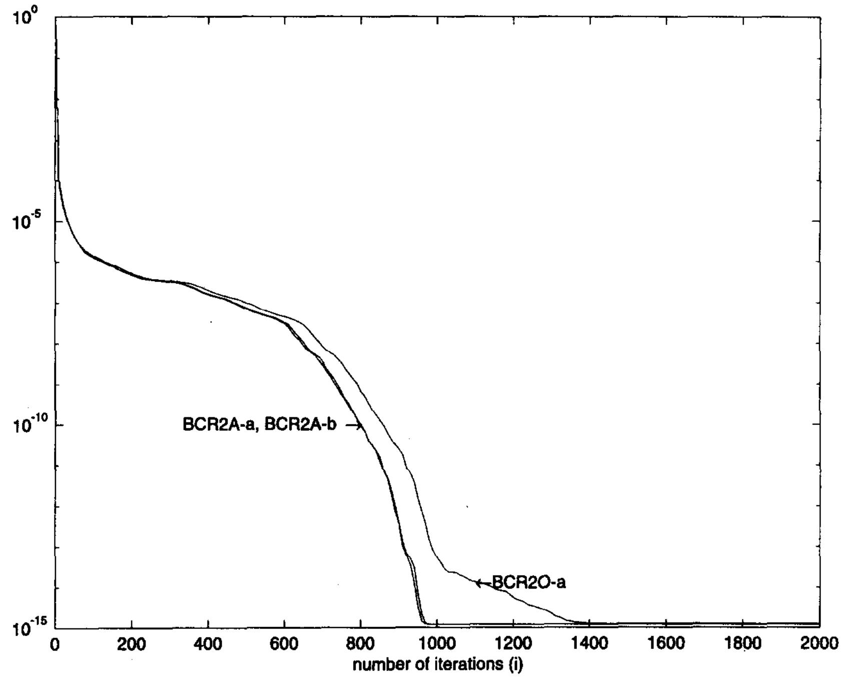

Figure 4. Matrix SHL400 ($n = 663$), behaviour of re_i in BCR2O-a, BCR2A-a, and BCR2A-b.

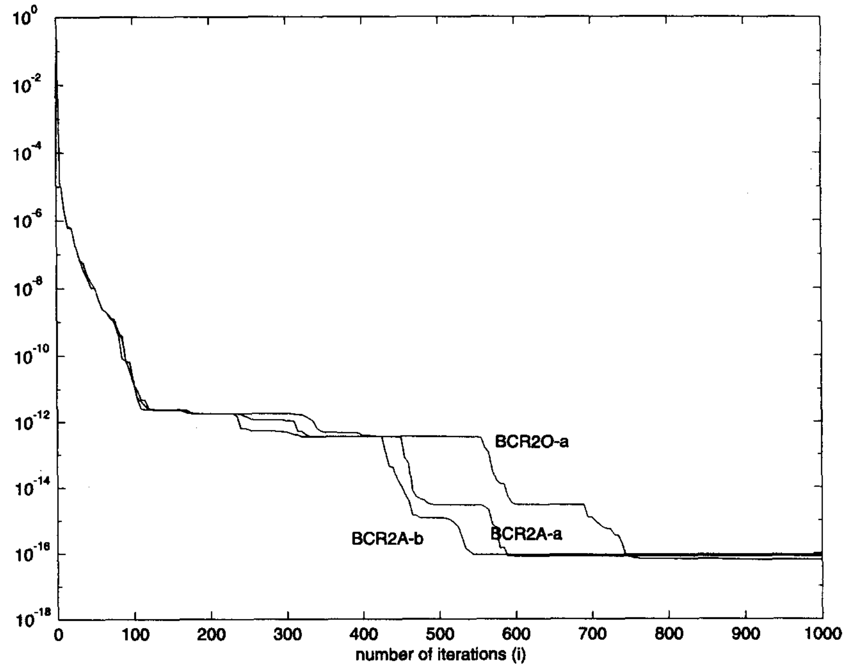

Figure 5. Matrix ARC130 ($n = 130$), behaviour of re_i in BCR2O-a, BCR2A-a, and BCR2A-b.

In Figure 6, the curves err_i, re_i, ri_i, $c\mathbf{A}_i$, and $re\_c\mathbf{A}_i$ are plotted for the version BCR2A-b on problem SHL400. We can observe that 800 iterations were needed for the condition number estimate $c\mathbf{A}_i$ to equal the MATLAB estimate; the product $re\_c\mathbf{A}_i$ was an upper bound to the ratio err_i. In Figure 7, the curves err_i, re_i, ri_i, $c\mathbf{A}_i$, and $re\_c\mathbf{A}_i$ are plotted for the version BCR2A-b on problem ARC130. About 160 iterations were needed for the condition number estimate $c\mathbf{A}_i$ to equal the MATLAB estimate; again, the product $re\_c\mathbf{A}_i$ was an upper bound to the ratio err_i.

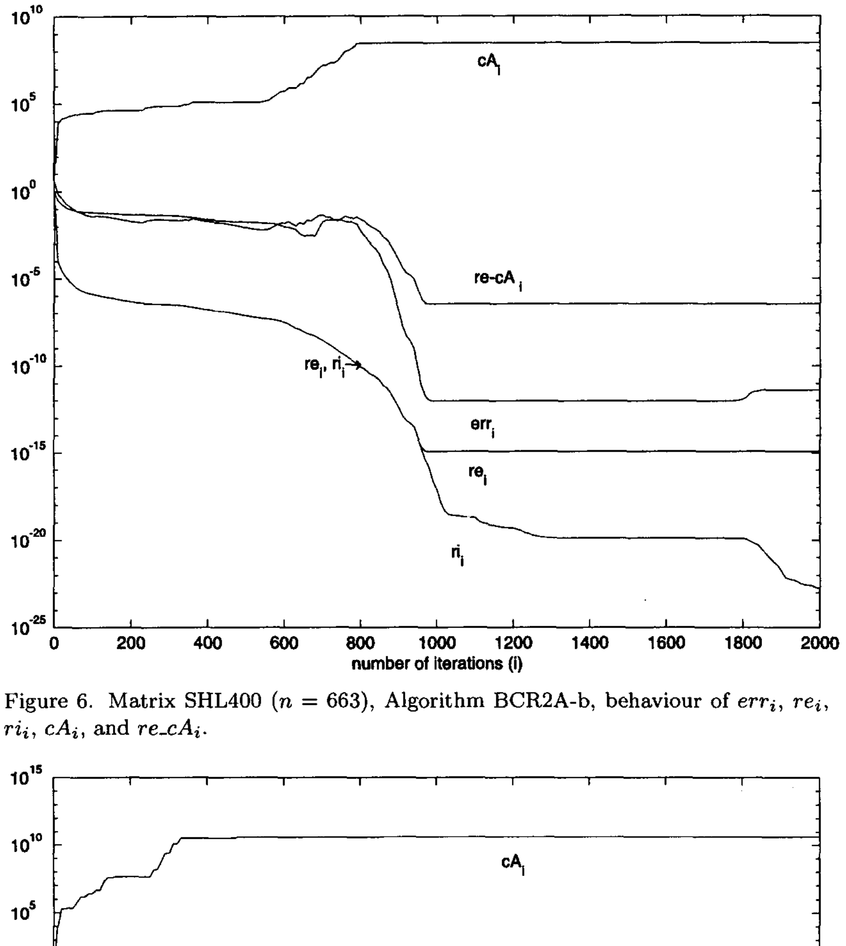

Figure 6. Matrix SHL400 (n = 663), Algorithm BCR2A-b, behaviour of err_i, re_i, ri_i, cA_i, and re_cA_i.

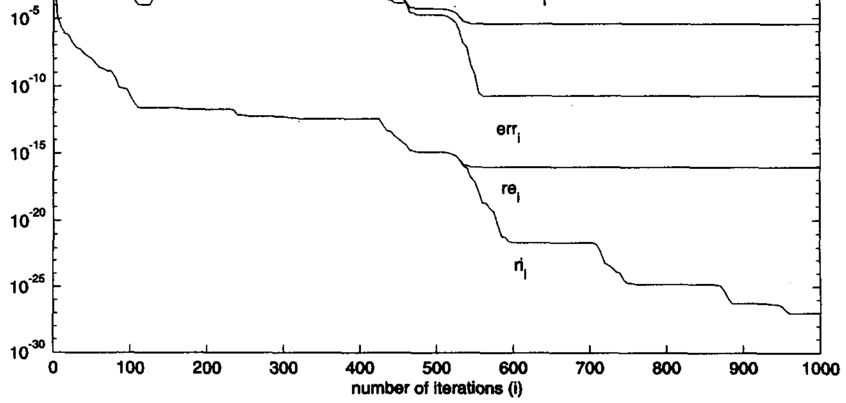

Figure 7. Matrix ARC130 (n = 130), Algorithm BCR2A-b, behaviour of err_i, re_i, ri_i, cA_i, and re_cA_i.

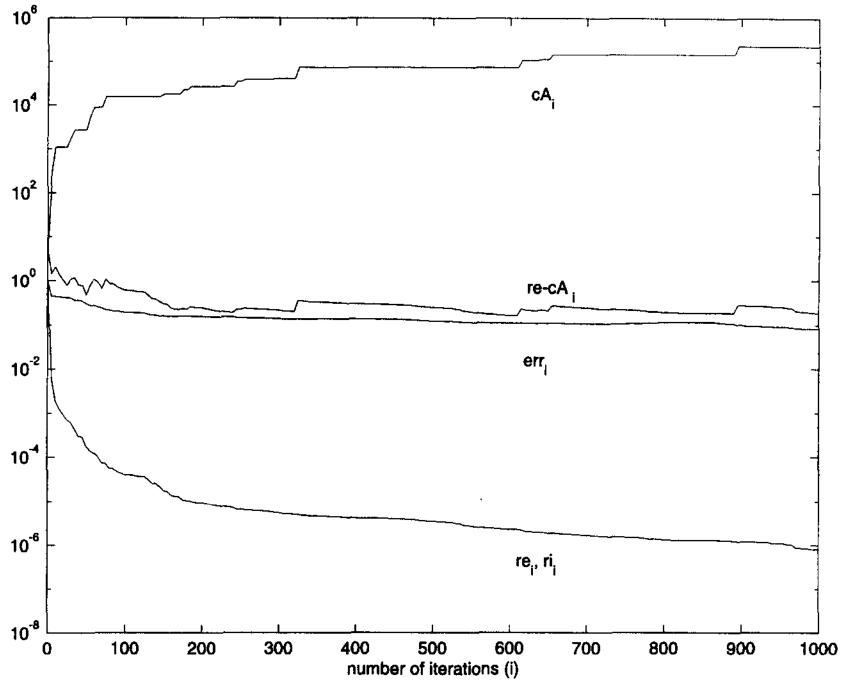

Figure 8. Matrix IMPCOLE ($n = 225$), Algorithm BCR2A-b.

Finally, we mention that for problem IMPCOLE ($n = 225$), all versions reduced the ratio re_i only to 10^{-6} after 1000 iterations. In Figure 8, the curves err_i, re_i, ri_i, $c\mathbf{A}_i$, and $re\_c\mathbf{A}_i$ are plotted for the version BCR2A-b.

3.5. Conclusions

The above-described numerical experiments have shown that efficient versions of Algorithm BCR can be obtained which only require two matrix-vector multiplications per iteration. In particular, the HS versions always performed at least as well as and often better than the Lanczos versions. For the HS versions, it is as good to use the alternative formula for $\mathbf{P}_{i+1}$ and the recursive generation of the auxiliary sequences $\{\mathbf{w}_i\}$ and $\{\mathbf{y}_i\}$. Among all the HS versions, Algorithm BCR2A-b can be considered the most efficient.

All these algorithms should include a way of estimating the condition number of the coefficient matrix. Indeed, the numerical experiments have shown that the value of $re\_c\mathbf{A}_i$ is an upper bound to the ratio err_i, provided the condition number estimate is accurate. Future work will be devoted to the analysis of alternative estimates of the condition number.

Incidentally, we note that the final accuracy should always be assessed by checking the *exactly* computed residual, instead of the *iteratively* computed residual.

Finally, more work will have to be done in order to avoid stagnation, which was shown in problem IMPCOLE.

REFERENCES

1. D.P. O'Leary, The block conjugate gradient algorithm and related methods, *Linear Algebra and Applications* **29**, 293–322, (1980).
2. M.R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bureau of Standards* **49**, 409–436, (1952).
3. C. Lanczos, Solution of systems of linear equations by minimized iterations, *J. Res. Nat. Bureau of Standards* **49**, 33–53, (1952).
4. C.G. Broyden, Look-ahead block-CG algorithms, In *Algorithms for Large Scale Linear Algebraic Systems*, (Edited by G. Winter Althaus and E. Spedicato), pp. 197–215, Kluwer Academic, (1998).
5. Cs.J. Hegedüs, Generating conjugate directions for arbitrary matrices by matrix equations—Part I, *Computers Math. Applic.* **21** (1), 71–85, (1991).

6. Cs.J. Hegedüs, Generating conjugate directions for arbitrary matrices by matrix equations—Part II, *Computers Math. Applic.* **21** (1), 87–94, (1991).
7. Cs.J. Hegedüs, Generation of conjugate directions for arbitrary matrices and solution of linear systems, In *Contributed Papers of the NATO Advanced Study Institute "Computer Algorithms for Solving Linear Algebraic Equations: The State of the Art"*, (Edited by E. Spedicato and M.T. Vespucci), University of Bergamo Research Report, (1991).
8. C.G. Broyden and M.T. Vespucci, Krylov methods for linear systems, (in preparation).