Block conjugate gradient methods with error norm estimates for least squares problems

Gérard Meurant[1*], Jan Papež[2†] and Petr Tichý[3†]

[1*]Paris, France.
[2]Institute of Mathematics, Czech Academy of Sciences, Žitná 25, Prague, 115 67, Czech Republic.
[3]Faculty of Mathematics and Physics, Charles University, Sokolovská 83, Prague, 186 65, Czech Republic.

*Corresponding author(s). E-mail(s): gerard.meurant@gmail.com;
Contributing authors: papez@math.cas.cz; petr.tichy@mff.cuni.cz;
†These authors contributed equally to this work.

Abstract

Least squares problems with multiple right-hand sides naturally arise in many practical applications. When the system matrix $\boldsymbol{A}$ is large and sparse, it is often convenient to solve such problems using suitably adapted variants of the block conjugate gradient method (block CGLS) or the block LSQR method. These block methods allow efficient use of modern computational architectures, and the number of iterations needed to achieve the required accuracy is typically much smaller than that required when solving each system separately and successively. However, a known limitation of these block methods is, for some problems, the occurrence of (near) breakdowns caused by (near) rank deficiencies within block vectors.

Assuming that $\boldsymbol{A}$ has a full column rank, we consider both the block CGLS and block LSQR algorithms. We show how ideas presented in 2001 by A. Dubrulle for block CG can be incorporated to prevent numerical instabilities or breakdowns caused by (near) rank deficiencies within block vectors. For the considered (preconditioned) algorithms, we derive estimates of the $\boldsymbol{A^TA}$-norm of the error for each individual system, as well as for the trace of the corresponding bilinear form. These estimates are often well suited for use in stopping criteria. We consider both lower and upper bounds and show how the estimates can be adaptively refined to heuristically achieve a prescribed level of accuracy. Numerical experiments clearly illustrate which block algorithms are the most effective for practical computations and demonstrate that adaptive estimates perform reliably.

Keywords: Least squares, Multiple right-hand sides, Block CGLS, Block LSQR, Error norm estimates

MSC Classification: 15A06 , 65F10

This paper is dedicated to Åke Björck on the occasion of his ninetieth birthday.

1 Introduction

Let A be an $n \times m$ real matrix with $n \geq m$ and full column rank, i.e., $\text{rank}(A) = m$. We are interested in solving in the least squares sense s linear systems

$$Ax^{(i)} = b^{(i)}, \quad i = 1, \ldots, s, \tag{1}$$

where $b^{(i)} \in \mathbb{R}^n$. The minimization of the norms $\|b^{(i)} - Ax^{(i)}\|$ is equivalent to solving square linear systems

$$A^T Ax^{(i)} = A^T b^{(i)}, \quad i = 1, \ldots, s. \tag{2}$$

This can be written in block form as

$$A^T AX = A^T B, \quad X = \left[x^{(1)}, \ldots, x^{(s)}\right], \quad B = \left[b^{(1)}, \ldots, b^{(s)}\right]. \tag{3}$$

Since the matrix A has full rank m, the matrix $A^T A$ is symmetric positive definite and there is a unique solution X.

When solving only one linear system ($s = 1$), applying the conjugate gradient (CG) algorithm to $A^T Ax = A^T b$ was proposed by M.R. Hestenes and E. Stiefel (originally, as a method to solve nonsymmetric linear systems) [1]. For least squares problems, this method is called CGLS, and CGNR for square linear systems in [2]. An alternative is the LSQR algorithm developed by C.C. Paige and M.A. Saunders [3, 4], which is based on the bidiagonalization process introduced by G.H. Golub and W. Kahan [5].

We would like to solve the linear system (3) with several right-hand sides using (preconditioned) block conjugate gradient methods. When solving square nonsingular linear systems, the block conjugate gradient method [6, 7] is known to suffer from rank deficiency issues in certain cases. Recently in [8], several variants have been revisited to address these difficulties. Our aim in this paper is to use some of these variants for the solution of least squares problems like (3). Moreover, we show how to estimate norms of the error during the iterations as it is done in [9] for square linear systems, and in [10] for least squares problems with only one right-hand side.

In Section 2 we recall some block CG algorithms for solving square linear systems with several right-hand sides, along with basic techniques for the error norm estimation. Section 3 describes how to adapt these block CG methods to solve the problem (3) without explicitly computing $A^T A$, and introduces the block LSQR algorithm. Section 4 presents strategies for estimating the quantities of interest in block methods applied to least squares problems with multiple right-hand sides. Preconditioning, an important component of these algorithms, is discussed in Section 5.

Section 6 describes numerical experiments that demonstrate the effectiveness of the proposed techniques across a range of problems.

2 Block CG algorithms

In this section we recall block CG methods for solving a block linear system $AX = B$ where A is a square real symmetric positive definite matrix of order n and B has s columns. The standard block conjugate gradient (BCG) algorithm was introduced by D.P. O'Leary [6] in 1980. It is described as Algorithm 1 and is referred to as HS-BCG to recall its close relationship with the conjugate gradient algorithm of M.R. Hestenes and E. Stiefel [1].

Algorithm 1 HS-BCG

input A, B, X_0
$R_0 = B - AX_0$
$P_0 = R_0$
for $k = 1, \ldots$ until convergence **do**
  $\Upsilon_{k-1} = (P_{k-1}^T A P_{k-1})^{-1}(R_{k-1}^T R_{k-1})$
  $X_k = X_{k-1} + P_{k-1}\Upsilon_{k-1}$
  $R_k = R_{k-1} - AP_{k-1}\Upsilon_{k-1}$
  $\Xi_k = (R_{k-1}^T R_{k-1})^{-1}(R_k^T R_k)$
  $P_k = R_k + P_{k-1}\Xi_k$
end for

In Algorithm 1, it is assumed that the $s \times s$ matrices $P_{k-1}^T A P_{k-1}$ and $R_{k-1}^T R_{k-1}$ are nonsingular. However, in practical computations, the columns of the $n \times s$ block vectors R_{k-1} or P_{k-1} may become linearly dependent or nearly linearly dependent. In that case, the inverses of $P_{k-1}^T A P_{k-1}$ or $R_{k-1}^T R_{k-1}$ cannot be reliably computed, and this may lead to a delayed convergence or even to a breakdown of the algorithm.

To address this issue, D.P. O'Leary [6] proposed the use of scaling matrices to enhance the numerical stability of the algorithm. While this approach can mitigate some problems, it does not fully resolve the issue. A BCG algorithm designed to handle rank deficiencies was proposed by H. Ji and Y. Li [11]. It is referred to as the breakdown-free block CG (BF-BCG) algorithm. In the presence of (near) rank deficiency, this algorithm selectively removes dependent vectors from the direction blocks P_k. This is sometimes called "deflation". However, as discussed in [8], it is not always easy to choose which columns to discard.

Other ideas to overcome the rank deficiency problems were proposed by A. Dubrulle [12]. Some of Dubrulle's algorithms are studied in [8, 9]. The most promising method is Algorithm 2 which was renamed DR-BCG in [8]. It was obtained by changing variables in HS-BCG; see [8, 9]. A key advantage of DR-BCG is that it avoids computing the inverses of $R_k^T R_k$, in contrast to HS-BCG, and eliminates the need for deflation. It uses a QR factorization of the residual block vectors such that the Q-factor of size $n \times s$ has full column rank, while the upper triangular R-factor of size $s \times s$ may eventually be

Algorithm 2 Dubrulle-R BCG (DR-BCG)

input A, B, X_0
$[Q_0, \Sigma_0] = \mathtt{qr}(B - AX_0)$
$S_0 = Q_0$
for $k = 1, \ldots$ until convergence **do**
  $\Pi_{k-1} = (S_{k-1}^T A S_{k-1})^{-1}$
  $X_k = X_{k-1} + S_{k-1}\Pi_{k-1}\Sigma_{k-1}$
  $[Q_k, \Psi_k] = \mathtt{qr}(Q_{k-1} - AS_{k-1}\Pi_{k-1})$
  $S_k = Q_k + S_{k-1}\Psi_k^T$
  $\Sigma_k = \Psi_k \Sigma_{k-1}$
end for

singular. This type of QR factorization can be computed using Householder reflections, for example, in MATLAB using the command `qr` with the parameter '`econ`'. As a result, in Algorithm 2, the vectors within a block Q_k are always linearly independent even if the matrices Ψ_k are singular. We observe that the inverse of the matrix Ψ_k is not needed in the algorithm. Therefore, deflation and the reduction of block size are not necessary.

In [13], it is proved that the matrices $S_{k-1}^T A S_{k-1}$ are always nonsingular. Therefore, DR-BCG cannot breakdown, even if some matrices Ψ_k are singular. Note that the residual blocks R_k are not explicitly available in Algorithm 2; however, they can always be reconstructed using the relation $R_k = Q_k \Sigma_k$.

When solving a symmetric positive definite linear system $Ax = b$ with the conjugate gradient algorithm, it is natural to consider the A-norm of the error $\|x - x_k\|_A$ defined by

$$\|x - x_k\|_A^2 = (x - x_k)^T A (x - x_k),$$

since it is minimized at each CG iteration. Lower and upper bounds of the A-norm of the error can be cheaply computed during the iterations using Gauss and Gauss-Radau quadrature rules; see an overview in [14].

To measure the convergence of block CG algorithms, we can consider the $s \times s$ matrices

$$\mathfrak{E}_k = (X - X_k)^T A (X - X_k) \tag{4}$$

and estimate either their diagonals or their traces. As shown in [9], it is possible to compute bounds for these quantities when solving square linear systems. Let us define

$$\Theta_{k-1} = \mathfrak{E}_{k-1} - \mathfrak{E}_k.$$

The $s \times s$ matrix Θ_{k-1} is readily computable in all block CG variants we consider. This matrix is positive (semi-)definite and its diagonal entries are lower bounds on the squares of the A-norms of the columns of the error block vectors at iteration $k-1$, that is, we have

$$\|x^{(i)} - x_{k-1}^{(i)}\|_A^2 \geq [\Theta_{k-1}]_{i,i}, \quad i = 1, \ldots, s.$$

The lower bounds can be improved by going back to previous iterations. As in [14, 15], we have

$$\|x^{(i)} - x_\ell^{(i)}\|_A^2 \geq \sum_{j=\ell}^{k-1} [\Theta_j]_{i,i}, \quad i = 1, \ldots, s, \tag{5}$$

with $\ell \leq k-1$. Hence, we go back a few iterations. This is called the delay, see [14]. As we will see later, this quantity may depend on both the system index i and the iteration number ℓ.

In HS-BCG, the $s \times s$ matrix Θ_{k-1} is given by

$$\Theta_{k-1} = (R_{k-1}^T R_{k-1}) \Upsilon_{k-1}, \tag{6}$$

while in DR-BCG it takes the form

$$\Theta_{k-1} = \Sigma_{k-1}^T \Pi_{k-1} \Sigma_{k-1},$$

as shown in [9].

Computing upper bounds is a little more tricky. Let μ be a lower bound for the smallest eigenvalue λ_1 of A. In [9] we show how to define block Gauss-Radau quadrature rules. Following an approach analogous to the single-vector case, we modify the block tridiagonal matrix T_{k+1} arising from the underlying block Lanczos algorithm such that $T_{k+1}^{(\mu)}$ has μ as an eigenvalue with multiplicity s. Note that the initial block Lanczos vector V_1 is obtained via the QR factorization of $R_0 = B - AX_0$, $[V_1, \Sigma_0] = \mathtt{qr}(R_0)$. We then consider the $s \times s$ matrix

$$\Theta_k^{(\mu)} \equiv \Sigma_0^T ([(T_{k+1}^{(\mu)})^{-1}]_{1:s,1:s} - [T_k^{-1}]_{1:s,1:s}) \Sigma_0,$$

whose diagonal entries can be used to construct Gauss–Radau upper bounds on the squared A-norms of the errors. As in the single-vector case, this matrix does not need to be computed via the above formula during the block CG iterations. Instead, it can be updated recursively. As shown in [9], it satisfies the recurrence

$$\Theta_k^{(\mu)} = \mathfrak{R}_k \left[\mu(\Theta_{k-1}^{(\mu)} - \Theta_{k-1}) + \mathfrak{R}_k\right]^{-1} (\Theta_{k-1}^{(\mu)} - \Theta_{k-1}), \tag{7}$$

with the initial value given by $\Theta_0^{(\mu)} = \mathfrak{R}_0/\mu$, where $\mathfrak{R}_k$ is defined as

$$\mathfrak{R}_k = (B - AX_k)^T (B - AX_k). \tag{8}$$

It is straightforward to show that

$$\begin{aligned} \mathfrak{R}_k &= R_k^T R_k \quad \text{in HS-BCG, and} \\ \mathfrak{R}_k &= \Sigma_k^T \Sigma_k \quad \text{in DR-BCG.} \end{aligned}$$

Observe that if the matrices $\Theta_j^{(\mu)} - \Theta_j$ are symmetric and positive definite, then the recurrence formula (7) is well defined even if the matrices $\mathfrak{R}_j$ are singular. In [9,

Lemma 7.5] we have shown that if the matrices R_j, for $j = 0, \ldots, k-1$, have full column rank, then the matrices $\Theta_j^{(\mu)} - \Theta_j$ are symmetric and positive definite. A detailed analysis of the behavior of DR-BCG in the rank-deficient case and the impact of finite precision arithmetic will be the subject of our forthcoming work [13].

As shown in [9], the diagonal entries of $\Theta_k^{(\mu)}$ provide upper bounds on the squared error norms

$$\|x^{(i)} - x_k^{(i)}\|_A^2 \leq [\Theta_k^{(\mu)}]_{i,i}, \quad i = 1, \ldots, s.$$

Moreover, improved upper bounds can be obtained for previous iterations by introducing a delay, as proposed in [14]. In particular, to compute Gauss–Radau upper bounds at iteration $\ell < k$, one obtains

$$\|x^{(i)} - x_\ell^{(i)}\|_A^2 \leq \sum_{j=\ell}^{k-1} [\Theta_j + \Theta_k^{(\mu)}]_{i,i}, \quad i = 1, \ldots, s.$$

3 Block algorithms for least squares problems

The block CG algorithms described in the previous section can be modified in an easy way to solve the block linear system (3) without explicitly computing the matrix $A^T A$. This modification follows the same principles as in the derivation of the classical CGLS algorithm for a single right-hand side; see [16, p. 282]. This was done for HS-BCG and BF-BCG by H. Ji and Y. Li [17] who adapted their previous algorithm [11, Alg. 2] to least squares problems. For the sake of simplicity, we do not present the algorithm from [17] here.

Algorithm 3 HS-BCGLS

input A, B, X_0
$R_0 = B - AX_0$
$\widetilde{R}_0 = A^T R_0$
$P_0 = \widetilde{R}_0$
for $k = 1, \ldots$ until convergence **do**
  $\Upsilon_{k-1} = [(AP_{k-1})^T AP_{k-1}]^{-1} (\widetilde{R}_{k-1}^T \widetilde{R}_{k-1})$
  $X_k = X_{k-1} + P_{k-1} \Upsilon_{k-1}$
  $R_k = R_{k-1} - AP_{k-1} \Upsilon_{k-1}$
  $\widetilde{R}_k = A^T R_k$
  $\Xi_k = (\widetilde{R}_{k-1}^T \widetilde{R}_{k-1})^{-1} (\widetilde{R}_k^T \widetilde{R}_k)$
  $P_k = \widetilde{R}_k + P_{k-1} \Xi_k$
end for

The standard algorithm HS-BCGLS is described as Algorithm 3. At each iteration this algorithm requires two matrix-block vector multiplications: one with A and one with A^T. The block vectors $\widetilde{R}_k$ and P_k are $m \times s$ matrices. For this algorithm to work, we have to assume that $(AP_{k-1})^T AP_{k-1}$ and $\widetilde{R}_{k-1}^T \widetilde{R}_{k-1}$ are nonsingular. As in the

Algorithm 4 Dubrulle-R BCGLS (DR-BCGLS)

1: **input** A, B, X_0
2: $R_0 = B - AX_0$
3: $[Q_0, \Sigma_0] = \mathtt{qr}(A^T R_0)$
4: $S_0 = Q_0$
5: **for** $k = 1, \ldots$ until convergence **do**
6: $\quad Y_{k-1} = AS_{k-1}$
7: $\quad \Pi_{k-1} = (Y_{k-1}^T Y_{k-1})^{-1}$
8: $\quad X_k = X_{k-1} + S_{k-1}\Pi_{k-1}\Sigma_{k-1}$
9: $\quad [Q_k, \Psi_k] = \mathtt{qr}(Q_{k-1} - A^T Y_{k-1}\Pi_{k-1})$
10: $\quad S_k = Q_k + S_{k-1}\Psi_k^T$
11: $\quad \Sigma_k = \Psi_k \Sigma_{k-1}$
12: **end for**

block CG case, (near) rank deficiency problems may arise and lead to instability or breakdown.

By applying DR-BCG to the block linear system (3) we derive the DR-BCGLS algorithm (Algorithm 4). To avoid using the matrix $A^T A$, we introduce new block vectors $Y_k = AS_k$, which are $n \times s$ matrices. As HS-BCGLS, the new algorithm DR-BCGLS requires two matrix-block vector multiplications per iteration: one with A and one with A^T. Although the block coefficients Ψ_k may become (nearly) singular, the columns within a block Q_k remain linearly independent. As for DR-BCG, DR-BCGLS cannot break down because $S_{k-1}^T A^T A S_{k-1} = Y_{k-1}^T Y_{k-1}$ is nonsingular. There is only one economy-size QR factorization of an $m \times s$ matrix at each iteration.

Note that the residual blocks R_k are not explicitly available in Algorithm 4. We can only reconstruct the block vectors $\widetilde{R}_k = A^T R_k$, using $\widetilde{R}_k = Q_k \Sigma_k$. It is worth noting that the numerical stability of the algorithm can be slightly improved by performing a QR factorization of the $m \times s$ matrix Y_{k-1}. In this variant, lines 6, 7, and 9 of Algorithm 4 are replaced by

$$\begin{aligned} [\widetilde{Y}_{k-1}, \widetilde{\Pi}_{k-1}] &= \mathtt{qr}(AS_{k-1}), \\ \Pi_{k-1} &= (\widetilde{\Pi}_{k-1}^T \widetilde{\Pi}_{k-1})^{-1}, \\ [Q_k, \Psi_k] &= \mathtt{qr}(Q_{k-1} - A^T \widetilde{Y}_{k-1}(\widetilde{\Pi}_{k-1}\Pi_{k-1})), \end{aligned}$$

respectively. In most cases, this modification has a negligible effect on the overall convergence of the algorithm. For this reason, we adopt Algorithm 4 in our numerical experiments. However, there are cases where this variant, which requires one additional QR factorization, can lead to a faster convergence. We refer to this variant as DR-BCGLS-Stab.

Let us now describe a block algorithm for least squares problems that generalizes the LSQR algorithm [3]. The algorithm is based on the block Golub–Kahan bidiagonalization process [18] and was derived in [19] by S. Karimi and F. Toutounian who introduced two variants of block LSQR. The first variant, BL-LSQR 1, minimizes the norms of the individual columns of the block residual vector, while the second

variant, BL-LSQR 2, minimizes the Frobenius norm of the entire block residual. The first variant very much resembles the standard LSQR algorithm, replacing the scalar coefficients by $s \times s$ blocks.

Algorithm 5 KT-BLSQR

1: **input** A, B, X_0
2: $[U_1, \beta_1] = \texttt{qr}(B - AX_0)$
3: $[V_1, \alpha_1] = \texttt{qr}(A^T U_1)$
4: Set $W_1 = V_1$, $\bar{\Phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1^T$
5: **for** $k = 1, 2, \ldots$ until convergence **do**
6: $\quad [U_{k+1}, \beta_{k+1}] = \texttt{qr}(AV_k - U_k \alpha_k^T)$
7: $\quad [V_{k+1}, \alpha_{k+1}] = \texttt{qr}(A^T U_{k+1} - V_k \beta_{k+1}^T)$
8: $\quad \begin{bmatrix} G_k^T, \begin{bmatrix} \rho_k \\ 0 \end{bmatrix} \end{bmatrix} = \texttt{QR}\left(\begin{bmatrix} \bar{\rho}_k \\ \beta_{k+1} \end{bmatrix} \right)$
9: $\quad \begin{bmatrix} \Phi_k & \Omega_{k+1} \\ \bar{\Phi}_{k+1} & \bar{\rho}_{k+1} \end{bmatrix} = G_k \begin{bmatrix} \bar{\Phi}_k & 0 \\ 0 & \alpha_{k+1}^T \end{bmatrix}$
10: $\quad X_k = X_{k-1} + W_k \left(\rho_k^{-1} \Phi_k \right)$
11: $\quad W_{k+1} = V_{k+1} - W_k \left(\rho_k^{-1} \Omega_{k+1} \right)$
12: **end for**

Algorithm 5 corresponds to the BL-LSQR 1 algorithm from [19]. We present it in a compact form and incorporate three important modifications. First, the original paper [19] contains a typographical error: it defines $\bar{\rho}_1$ as α_1, omitting the transposition. Second, we allow for a nonzero initial approximation X_0. Finally, unlike the original work [19], which assumes that the matrices β_i are nonsingular, we follow Dubrulle's approach from [12]. We assume that the QR factorizations in lines 6 and 7 are computed such that the Q-factor of size $n \times s$ has full column rank, while the upper triangular R-factor of size $s \times s$ may be singular. This modification results in an algorithm without deflation and with a constant block size. However, the occurrence of breakdowns may not be fully resolved and requires a more thorough theoretical analysis which is outside the scope of this paper. In particular, while ρ_k must be nonsingular whenever β_{k+1} is nonsingular, see line 8 in Algorithm 5, it is not clear that ρ_k^{-1} exists in the rank deficient case. Note that in line 8 we compute a full QR factorization of a $(2s) \times s$ matrix. As a result, the Q-factor (the matrix G_k^T) is of size $(2s) \times (2s)$, and the R-factor is of size $(2s) \times s$. To distinguish this from the economy-size QR factorization, we denote the procedure for computing the full QR factorization by `QR`. We refer to the modified version presented in Algorithm 5 as KT-BLSQR. In Algorithm 5, both lowercase and uppercase Greek letters are used to denote $s \times s$ blocks. This slightly deviates from the overall notation used throughout this paper. The reason for this choice is a limited availability of distinct uppercase Greek letters.

Note that KT-BLSQR is slightly more expensive per iteration than DR-BCGLS, both in terms of computational cost and memory requirements. KT-BLSQR performs two QR factorizations of block vectors per iteration, whereas DR-BCGLS requires only

one. Additionally, KT-BLSQR must store one more block vector in memory compared to DR-BCGLS.

Let us briefly recall the idea behind the block LSQR algorithm. As already mentioned, Algorithm 5 is based on the block Golub–Kahan bidiagonalization process [18]. Suppose that there are no issues with rank deficiency; that is, both α_{k+1} and β_{k+1} are nonsingular. Define the matrices

$$\mathcal{V}_k = [V_1, V_2, \ldots, V_k], \quad \mathcal{U}_k = [U_1, U_2, \ldots, U_k],$$

and the block bidiagonal matrix

$$\mathcal{B}_k = \begin{bmatrix} \alpha_1^T & & & \\ \beta_2 & \alpha_2^T & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k^T \\ & & & \beta_{k+1} \end{bmatrix}.$$

Then, the block Golub–Kahan process satisfies

$$A\mathcal{V}_k = \mathcal{U}_{k+1}\mathcal{B}_k, \quad A^T\mathcal{U}_{k+1} = \mathcal{V}_k\mathcal{B}_k^T + V_{k+1}\alpha_{k+1}E_{k+1}^T, \tag{9}$$

where $E_{k+1} = [0, 0, \ldots, 0, I_s]^T$ is $(k+1)s \times s$. Assuming exact arithmetic, the matrices $\mathcal{V}_k$ and $\mathcal{U}_k$ have orthonormal columns, $\mathcal{V}_k^T\mathcal{V}_k = I_{ks} = \mathcal{U}_k^T\mathcal{U}_k$. Moreover, from the recurrence defining W_k (line 11) we can write

$$W_k = V_k + \sum_{i=1}^{k-1} V_i C_i^{(k)}$$

for some $s \times s$ matrices $C_i^{(k)}$. Using the orthogonality condition $V_i^T V_j = \delta_{i,j} I_s$, where $\delta_{i,j}$ is the Kronecker delta, we easily obtain

$$V_k^T W_k = I_s \quad \text{and} \quad V_{k+1}^T W_k = 0. \tag{10}$$

The approximations X_k are sought in the form

$$X_k = X_0 + \mathcal{V}_k Y_k, \quad Y_k \in \mathbb{R}^{(ks)\times s},$$

such that each column-wise residual $r_k^{(i)} = b^{(i)} - Ax_k^{(i)}$, $i = 1, \ldots, s$, has minimal Euclidean norm. Since

$$\begin{aligned} R_k &= B - AX_k = R_0 - A\mathcal{V}_k Y_k \\ &= U_1\beta_1 - \mathcal{U}_{k+1}\mathcal{B}_k Y_k \\ &= \mathcal{U}_{k+1}\left(E_1\beta_1 - \mathcal{B}_k Y_k\right), \end{aligned} \tag{11}$$

where $E_1 = [I_s, 0, \ldots, 0, 0]^T$ is $(k+1)s \times s$, the block vector $Y_k \in \mathbb{R}^{(ks)\times s}$ is the solution of the (block) normal equations

$$\mathcal{B}_k^T \mathcal{B}_k Y_k = \mathcal{B}_k^T E_1 \beta_1. \tag{12}$$

In block LSQR, this system is solved by updating the QR factorization of the extended matrix $[\mathcal{B}_k, E_1\beta_1]$; see [19] and the proof of the forthcoming Theorem 1 for more details.

We now prove that $A^T(B - AX_k)$ is equal to V_{k+1} multiplied by an $s \times s$ matrix. We then identify this matrix explicitly. These relations will be used in Section 4 in the context of error estimation.

Theorem 1 *Suppose that Algorithm 5 is applied to solve the systems (3), and assume that the block coefficients α_j and β_j are nonsingular for $j = 1, \ldots, k+1$. Then, it holds that*

$$A^T(B - AX_k) = -V_{k+1}\Omega_{k+1}^T \Phi_k \tag{13}$$

and

$$\rho_k^T \Phi_k = -\Omega_k^T \Phi_{k-1}. \tag{14}$$

Proof The proof of (13) follows the proof of [10, Lemma 1], using the analogous relationships for blocks; see [19, Section 3]. From (9), (11), and (12), we get

$$\begin{aligned} A^T(B - AX_k) &= A^T \mathcal{U}_{k+1} \left(E_1\beta_1 - \mathcal{B}_k Y_k\right) \\ &= \left(\mathcal{V}_k \mathcal{B}_k^T + V_{k+1}\alpha_{k+1}E_{k+1}^T\right)\left(E_1\beta_1 - \mathcal{B}_k Y_k\right) \\ &= \mathcal{V}_k\left(\mathcal{B}_k^T E_1\beta_1 - \mathcal{B}_k^T \mathcal{B}_k Y_k\right) + V_{k+1}\alpha_{k+1}E_{k+1}^T\left(E_1\beta_1 - \mathcal{B}_k Y_k\right) \\ &= V_{k+1}\alpha_{k+1}E_{k+1}^T\left(E_1\beta_1 - \mathcal{B}_k Y_k\right). \end{aligned}$$

Using the block QR factorization

$$[\mathcal{B}_k, E_1\beta_1] = Q_k^T \begin{bmatrix} \widetilde{R}_k & F_k \\ 0 & \bar{\Phi}_{k+1} \end{bmatrix} \tag{15}$$

and $\widetilde{R}_k Y_k = F_k$, we obtain

$$E_1\beta_1 - \mathcal{B}_k Y_k = Q_k^T \begin{bmatrix} \widetilde{R}_k & F_k \\ 0 & \bar{\Phi}_{k+1} \end{bmatrix} \begin{bmatrix} -Y_k \\ I_s \end{bmatrix} = Q_k^T \begin{bmatrix} 0 \\ \bar{\Phi}_{k+1} \end{bmatrix}.$$

Therefore,

$$A^T(B - AX_k) = V_{k+1}\alpha_{k+1}E_{k+1}^T Q_k^T \begin{bmatrix} 0 \\ \bar{\Phi}_{k+1} \end{bmatrix}. \tag{16}$$

The matrix Q_k in (15) is a $(k+1)s \times (k+1)s$ unitary matrix, which can be updated using the previously computed $ks \times ks$ unitary matrix Q_{k-1} as follows

$$Q_k = \begin{bmatrix} I_{(k-1)s} & 0 \\ 0 & G_k \end{bmatrix} \begin{bmatrix} Q_{k-1} & 0 \\ 0 & I_s \end{bmatrix}, \qquad Q_1 = G_1,$$

where $G_k \in \mathbb{R}^{2s\times 2s}$ is unitary. For ease of presentation, we partition G_k from line 8 of Algorithm 5 into $s \times s$ blocks and rewrite lines 8 and 9 as

$$\begin{bmatrix} \rho_k \\ 0 \end{bmatrix} = \begin{bmatrix} \widetilde{\alpha}_k & \widetilde{\beta}_k \\ \widetilde{\gamma}_k & \widetilde{\delta}_k \end{bmatrix} \begin{bmatrix} \bar{\rho}_k \\ \beta_{k+1} \end{bmatrix}, \tag{17}$$

$$\begin{bmatrix} \Phi_k & \Omega_{k+1} \\ \bar{\Phi}_{k+1} & \bar{\rho}_{k+1} \end{bmatrix} = \begin{bmatrix} \widetilde{\alpha}_k & \widetilde{\beta}_k \\ \widetilde{\gamma}_k & \widetilde{\delta}_k \end{bmatrix} \begin{bmatrix} \bar{\Phi}_k & 0 \\ 0 & \alpha_{k+1}^T \end{bmatrix}. \tag{18}$$

Observe that by considering the identity $G_k^T G_k = I_{2s}$ and examining the $(2,1)$-block, we obtain the relation $\widetilde{\beta}_k^T \widetilde{\alpha}_k + \widetilde{\delta}_k^T \widetilde{\gamma}_k = 0$.

From (16), $A^T(B - AX_k)$ is equal to V_{k+1} multiplied by the $s \times s$ matrix

$$\alpha_{k+1} E_{k+1}^T Q_k^T \begin{bmatrix} 0 \\ \bar{\Phi}_{k+1} \end{bmatrix} = \alpha_{k+1} \widetilde{\delta}_k^T \bar{\Phi}_{k+1}.$$

Using (18) and $\widetilde{\delta}_k^T \widetilde{\gamma}_k = -\widetilde{\beta}_k^T \widetilde{\alpha}_k$ we obtain

$$\alpha_{k+1} \widetilde{\delta}_k^T \bar{\Phi}_{k+1} = \alpha_{k+1} \widetilde{\delta}_k^T \widetilde{\gamma}_k \bar{\Phi}_k = -\alpha_{k+1} \widetilde{\beta}_k^T \widetilde{\alpha}_k \bar{\Phi}_k = -\Omega_{k+1}^T \Phi_k, \tag{19}$$

which proves (13).

To show (14), let us multiply (17) by G_k^T to obtain the relation $\bar{\rho}_k = \widetilde{\alpha}_k^T \rho_k$. From (19) for $k-1$,

$$-\Omega_k^T \Phi_{k-1} = \alpha_k \widetilde{\delta}_{k-1}^T \bar{\Phi}_k = \bar{\rho}_k^T \bar{\Phi}_k = \rho_k^T \widetilde{\alpha}_k \bar{\Phi}_k = \rho_k^T \Phi_k,$$

where we have used (18) and $\bar{\rho}_k = \widetilde{\alpha}_k^T \rho_k$. □

4 Adaptive error norm estimates

Before discussing the block case, let us first recall the adaptive error estimation strategies developed in [20] for CG and applied to least squares problems with a single right-hand side in [10]. For a summary and several minor refinements, see the book [14]. We will then extend these results to the block setting.

4.1 The single-vector case

Assume that $A \in \mathbb{R}^{n \times m}$ has full column rank. For a least squares problem of the form

$$\min_y \|b - Ay\|,$$

it is often natural to measure the quality of an approximate solution using the $A^T A$-norm norm of the error, as it is directly related to the residual norm being minimized. Let $b_{|\mathcal{R}(A)}$ denotes the orthogonal projection of the right-hand side b onto the range of A, and define the residual $r = b - b_{|\mathcal{R}(A)}$. This is the component of b orthogonal to the range of A, and it corresponds to the residual associated with the least squares solution x. Then, for the iterates x_k and the corresponding residual vector $r_k = b - Ax_k$, it holds that

$$\|x - x_k\|_{A^T A}^2 = \|r_k\|^2 - \|r\|^2.$$

In the consistent case, where $b \in \mathcal{R}(A)$, the residual r vanishes, and the norm of r_k directly gives the $A^T A$-norm of the error. However, when $b \notin \mathcal{R}(A)$, the residual r is nonzero, and estimating or bounding $\|x - x_k\|_{A^T A}$ becomes more subtle. Approaches to handle such situations, particularly for problems with a single right-hand side, have been studied in [10].

Note that the estimates of $\|x - x_k\|_{A^TA}$ can be used in stopping criteria, which often depend on the specific context or application in which the least squares problems are solved. In particular, as discussed in [10], we may wish to stop the iterations when

$$\frac{\|r_k\|^2 - \|r\|^2}{\|r\|^2} \leq \varepsilon,$$

where ε is a user-specified tolerance. This condition is equivalent to

$$\|x - x_k\|^2_{A^TA} \leq \frac{\varepsilon}{1+\varepsilon}\|r_k\|^2.$$

Another stopping criterion can be based on the concept of backward error; see [21]. Specifically, we may wish to stop the iterations when x_k satisfies the perturbed system of normal equations

$$(A+E)^T(A+E)x_k = (A+E)^T(b+f), \tag{20}$$

where $E \in \mathbb{R}^{n\times m}$ and $f \in \mathbb{R}^n$ are small in some sense. For instance, the iterations can be terminated when the normwise relative backward error for the system of normal equations

$$\eta_k \equiv \min_{E,f}\{\xi : \ (20) \text{ holds with } \frac{\|E\|}{\|A\|} \leq \xi, \ \frac{\|f\|}{\|b\|} \leq \xi\},$$

satisfies $\eta_k \leq \varepsilon$ for a user-specified tolerance ε. Unfortunately, no explicit analytical expression for η_k is known. However, it has been shown in [21] that

$$\eta_k \leq \frac{\|x - x_k\|_{A^TA}}{\|A\|\|x_k\| + \|b\|},$$

and that this upper bound becomes asymptotically tight as x_k approaches the least squares solution; see [21, Theorem 5.2]. Therefore, instead of the idealized stopping criterion $\eta_k \leq \varepsilon$, one may use the stricter, computable criterion

$$\frac{\|x - x_k\|_{A^TA}}{\|A\|\|x_k\| + \|b\|} \leq \varepsilon,$$

where both $\|x - x_k\|_{A^TA}$ and $\|A\|$ are replaced by their respective estimates. Note that $\|A\|$ can be efficiently estimated during the solution process; see [14, Section 5.3]. More generally, one can consider the criterion

$$\|x - x_k\|_{A^TA} \leq \alpha\|A\|\|x_k\| + \beta\|b\|,$$

where $0 \leq \alpha, \beta \ll 1$ are prescribed tolerances; see [21, 22].

Let us now focus on techniques to estimate the A^TA-norm of the error. Building on earlier results from [14, 20] on adaptive error estimation in the conjugate gradient method, an analogous strategy for the CGLS and LSQR algorithms was proposed

in [10]. In more detail, the adaptive estimation exploits a computable quantity that captures the reduction in the squared A^TA-norm of the error between consecutive iterations. Specifically, we use the identity

$$\|x - x_{k-1}\|^2_{A^TA} - \|x - x_k\|^2_{A^TA} = \theta_{k-1},$$

and θ_{k-1} is easily computable in both CGLS and LSQR, which yields

$$\|x - x_\ell\|^2_{A^TA} = \theta_{\ell:k-1} + \|x - x_k\|^2_{A^TA}, \qquad \theta_{\ell:k-1} \equiv \sum_{j=\ell}^{k-1} \theta_j, \tag{21}$$

for $0 \le \ell < k$. Given a prescribed tolerance $\tau \in (0, 1)$ (typically $\tau = 0.25$), the goal of the adaptive error estimation strategy is to determine the largest index ℓ, with $0 \le \ell < k$, such that

$$\frac{\|x - x_k\|^2_{A^TA}}{\|x - x_\ell\|^2_{A^TA}} = \frac{\|x - x_\ell\|^2_{A^TA} - \theta_{\ell:k-1}}{\|x - x_\ell\|^2_{A^TA}} \le \tau, \tag{22}$$

i.e., such that the relative accuracy of the estimate $\theta_{\ell:k-1}$ does not exceed the prescribed tolerance. From the previous discussion, it is clear that $\theta_{\ell:k-1}$ yields a lower bound for $\|x - x_\ell\|^2_{A^TA}$. We also observe that inequality (22) is equivalent to

$$\|x - x_\ell\|^2_{A^TA} \le \frac{\theta_{\ell:k-1}}{1 - \tau}.$$

This implies that whenever (22) is satisfied, we also obtain an upper bound on the A^TA-norm of the error.

Using (21), the numerator in (22) corresponds to the squared A^TA-norm of the error at iteration k. Thus, our goal is to find the (largest possible) index ℓ such that the squared A^TA-norm of the error has sufficiently decreased from iteration ℓ to iteration k. To approach this goal, we first replace the original criterion (22) with a stricter but more tractable one:

$$\frac{\|x - x_k\|^2_{A^TA}}{\|x - x_\ell\|^2_{A^TA}} < \frac{\|x - x_{k-1}\|^2_{A^TA}}{\|x - x_\ell\|^2_{A^TA}} \le \tau.$$

The idea used in [14, 20] is to replace both the numerator $\|x - x_{k-1}\|^2_{A^TA}$ and denominator $\|x - x_\ell\|^2_{A^TA}$ in the criterion with suitable bounds. For the denominator, we use the lower bound $\theta_{\ell:k-1}$. An upper bound for the numerator is not easily available. Therefore, we introduce a safety factor $\mathcal{S}_{k-1}$ that should satisfy

$$\|x - x_{k-1}\|^2_{A^TA} \le \mathcal{S}_{k-1}\theta_{k-1}.$$

Then, we choose ℓ as the largest index such that

$$\frac{\mathcal{S}_{k-1}\theta_{k-1}}{\theta_{\ell:k-1}} \leq \tau. \tag{23}$$

It remains to explain how to heuristically determine $\mathcal{S}_{k-1}$. We define it as

$$\mathcal{S}_{k-1} = \max_{p \leq j \leq k-1} \frac{\theta_{j:k-1}}{\theta_j}.$$

For a given i, the integer p is the largest integer satisfying

$$\frac{\theta_{j:k-1}}{\theta_{p:k-1}} \leq 10^{-4}.$$

The choice of p is done to only use information from the latest iterations that caused a significant decrease of the square of the error norm. The adaptive strategy aims at keeping the delay as small as possible. For a detailed description and derivation, we refer the reader to [10, 14, 20].

Algorithm 6 `adaptive`

input k, d, ℓ, $\{\theta_j\}_{j=0}^{k-1}$, τ, h
if $k = 0$ **then**
    $d = 0$; $\ell = 0$; $h = [\,]$; return
end if
set p as the largest index j, $0 \leq j < k-1$, such that

$$\frac{\theta_{\ell:k-1}}{\theta_{j:k-1}} \leq 10^{-4}$$

if such an p does not exist, set $p = 0$
determine

$$\mathcal{S} = \max_{p \leq j < k-1} \theta_{j:k-1}/\theta_j$$

while ($d \geq 0$ **and** $\mathcal{S}\theta_{k-1}/\theta_{\ell:k-2} \leq \tau$) **do**
    $h_\ell = \theta_{\ell:k-1}^{1/2}$
    $\ell = \ell + 1$, $d = d - 1$
end while
$d = d + 1$
output d, ℓ, h

In [10, Algorithm 2] we present a function named `adaptive`, which implements the above error estimation strategy and can be used in any iterative algorithm that computes the quantities θ_k. A MATLAB implementation of this function is available in the

GitHub repository[1]. Here, in Algorithm 6, we provide a slightly modified version of the function from [10], adapted to be more consistent with the MATLAB implementation available in the repository.

Before starting the iterative process, we initialize the quantities required for adaptive error estimation by calling $[d, \ell, h] = \texttt{adaptive}(0)$. Subsequently, the function

$$[d, \ell, h] = \texttt{adaptive}(k, d, \ell, \{\theta_j\}_{j=0}^{k-1}, \tau, h)$$

should be called within the main loop of the algorithm, after the scalar θ_{k-1} has been computed. The function `adaptive` takes as an input the current iteration index k, the indices d and ℓ determined in the previous iteration $k-1$, the vector of scalars θ_j, $j = 0, \ldots, k-1$, the prescribed tolerance τ (as introduced in (22)), and the vector h, which stores the adaptive estimates of the A^TA-norm of the error. The function returns updated values of d, ℓ, and the vector h. The indices d and ℓ may either remain unchanged or be updated. If the returned value is $\ell = 0$, it indicates that no adaptive estimate has yet been determined, i.e., the current iteration k is not large enough for the criterion (22) to be heuristically satisfied for any $\ell \geq 0$. If $\ell > 0$, then

$$h_{\ell-1} \approx \|x - x_{\ell-1}\|_{A^TA}$$

provides an adaptive estimate of $\|x - x_{\ell-1}\|_{A^TA}$.

The case of the Gauss–Radau upper bound is easier to handle; see [14, Section 8.2]. In analogy with the approach described above, we define

$$\theta_{\ell:k}^{(\mu)} = \theta_{\ell:k-1} + \theta_k^{(\mu)},$$

where $\theta_k^{(\mu)}$ can be updated by the scalar version of the formula (7). Our goal is to ensure that

$$\frac{\theta_{\ell:k}^{(\mu)} - \|x - x_\ell\|_{A^TA}^2}{\|x - x_\ell\|_{A^TA}^2} \leq \tau.$$

As shown in [14, p. 87], it is sufficient to choose ℓ as the largest integer such that

$$\frac{\theta_k^{(\mu)}}{\theta_{\ell:k-1}} \leq \tau.$$

4.2 The block case

The formulas for the bounds derived in [14] and recalled in Section 2 for HS-BCG and DR-BCG depend only on the residual blocks and the block coefficients, and not directly on the matrix or the right-hand sides. Therefore, they can also be used for HS-BCGLS and DR-BCGLS without any modification.

Let us denote

$$\mathfrak{E}_k = (X - X_k)^T A^T A (X - X_k).$$

[1] https://github.com/JanPapez/CGlike-methods-with-error-estimate

Analogously to the block CG algorithms (Section 2), we define the $s \times s$ matrices

$$\Theta_{k-1} = \mathfrak{E}_{k-1} - \mathfrak{E}_k \quad \text{and} \quad \mathfrak{R}_k = (B - AX_k)^T AA^T (B - AX_k), \tag{24}$$

which will be used for error estimation. We will show that both of these matrices are readily computable within all of the considered block algorithms, as follows:

$$\begin{aligned} \text{(HS-BCGLS)} \qquad & \Theta_{k-1} = (\widetilde{R}_{k-1}^T \widetilde{R}_{k-1}) \Upsilon_{k-1}, \\ \text{(DR-BCGLS)} \qquad & \Theta_{k-1} = \Sigma_{k-1}^T \Pi_{k-1} \Sigma_{k-1}, \\ \text{(KT-BLSQR)} \qquad & \Theta_{k-1} = \Phi_k^T \Phi_k, \end{aligned}$$

and

$$\begin{aligned} \text{(HS-BCGLS)} \qquad & \mathfrak{R}_k = \widetilde{R}_k^T \widetilde{R}_k, \\ \text{(DR-BCGLS)} \qquad & \mathfrak{R}_k = \Sigma_k^T \Sigma_k, \\ \text{(KT-BLSQR)} \qquad & \mathfrak{R}_k = \Phi_k^T \Omega_{k+1} \Omega_{k+1}^T \Phi_k. \end{aligned}$$

The proof for HS-BCGLS and DR-BCGLS is straightforward. For KT-BLSQR, we build on Theorem 1, and, analogously to [10, Lemma 1], establish the following result.

Theorem 2 *Suppose that Algorithm 5 is applied to solve the systems (3), and assume that the block coefficients α_j and β_j are nonsingular for $j = 1, \ldots, k+1$. Then, the matrices Θ_{k-1} and $\mathfrak{R}_k$ defined by (24) can be computed as*

$$\Theta_{k-1} = \Phi_k^T \Phi_k \quad \textit{and} \quad \mathfrak{R}_k = \Phi_k^T \Omega_{k+1} \Omega_{k+1}^T \Phi_k. \tag{25}$$

Proof The expression for $\mathfrak{R}_k$ follows immediately from (13) and $V_{k+1}^T V_{k+1} = I_s$.

We now focus on the matrix $\Theta_{k-1} = \mathfrak{E}_{k-1} - \mathfrak{E}_k$. Using a simple algebraic manipulation, Θ_{k-1} can be written as

$$\Theta_{k-1} = (X_k - X_{k-1})^T A^T A (X - X_{k-1}) + (X_k - X_{k-1})^T A^T A (X - X_k).$$

If β_{k+1} is nonsingular, ρ_k is also nonsingular, and from the update formula for X_k (see line 10 in Algorithm 5), we have

$$X_k - X_{k-1} = W_k \rho_k^{-1} \Phi_k.$$

Furthermore, using $A^T A(X - X_k) = A^T (B - AX_k)$, (13), and (14) we obtain

$$\begin{aligned} A^T A(X - X_k) &= -V_{k+1} \Omega_{k+1}^T \Phi_k, \\ A^T A(X - X_{k-1}) &= V_k \rho_k^T \Phi_k. \end{aligned}$$

Substituting into the expression for Θ_{k-1}, and using (10), we get

$$\Theta_{k-1} = \Phi_k^T \rho_k^{-T} W_k^T \left(V_k \rho_k^T \Phi_k - V_{k+1} \Omega_{k+1}^T \Phi_k \right) = \Phi_k^T \Phi_k,$$

which finishes the proof. □

The matrices Θ_{k-1} and $\mathfrak{R}_k$ play a key role both in the computation of lower bounds and in the recursive evaluation of Gauss–Radau upper bounds in the block setting. To get the Gauss-Radau upper bounds, we need to update the block coefficient denoted now as $\Theta_k^{(\mu)}$ using an analogue of the formula (7),

$$\Theta_k^{(\mu)} = \mathfrak{R}_k \left[\mu(\Theta_{k-1}^{(\mu)} - \Theta_{k-1}) + \mathfrak{R}_k\right]^{-1} (\Theta_{k-1}^{(\mu)} - \Theta_{k-1}),$$

where $\Theta_0^{(\mu)} = \mathfrak{R}_0/\mu$.

Since all the quantities required for error estimation in the block case are available on the diagonals of the block coefficient Θ_{k-1} and $\Theta_k^{(\mu)}$, we can apply the results from the single-vector case to each individual system. As before, let us index the individual systems by i, $i = 1, \ldots, s$.

For the adaptive lower bounds, each system will have its own individual indices $d^{(i)}$, $\ell^{(i)}$, and vector $h^{(i)}$. Denoting $\theta_j^{(i)} \equiv [\Theta_j]_{i,i}$, i.e., the (i,i)-entry on the diagonal of Θ_j, the adaptive lower bound for the ith system can be updated using the call

$$[d^{(i)}, \ell^{(i)}, h^{(i)}] = \texttt{adaptive}(k, d^{(i)}, \ell^{(i)}, \{\theta_j^{(i)}\}_{j=0}^{k-1}, \tau, h^{(i)}),$$

employing the same function `adaptive` as defined in Algorithm 6.

Analogously, we can determine the adaptive Gauss–Radau upper bounds for each individual system. More precisely, let us define

$$\theta_k^{(\mu,i)} = \left[\Theta_k^{(\mu)}\right]_{i,i}, \quad \theta_{\ell:k-1}^{(i)} = \sum_{j=\ell}^{k-1} \theta_j^{(i)}.$$

Then, for the ith system, we choose the index $\ell^{(\mu,i)}$ as the largest integer satisfying

$$\frac{\theta_k^{(\mu,i)}}{\theta_{\ell^{(\mu,i)}:k-1}^{(i)}} \leq \tau.$$

In summary, for each linear system, we can compute two adaptive delays: one associated with the Gauss lower bound and another one with the Gauss–Radau upper bound.

Instead of treating each system individually, one may compute adaptive delays based on the trace of $\mathfrak{E}_k$, thereby providing a global measure of convergence across all systems. Define

$$\theta_k^{(\mu)} = \sum_{i=1}^{s} \theta_k^{(\mu,i)}, \quad \theta_j = \sum_{i=1}^{s} \theta_j^{(i)}, \quad \theta_{\ell:k-1} = \sum_{j=\ell}^{k-1} \theta_j,$$

and update the adaptive trace lower bound using

$$[d, \ell, h] = \texttt{adaptive}(k, d, \ell, \{\theta_j\}_{j=0}^{k-1}, \tau, h).$$

For the adaptive trace upper bounds, we choose the index $\ell^{(\mu)}$ as the largest integer satisfying

$$\frac{\theta_k^{(\mu)}}{\theta_{\ell^{(\mu)}:k-1}} \leq \tau.$$

5 Preconditioning

In many practical problems, the matrices $A^T A$ are badly conditioned, and preconditioning is necessary to ensure an efficient convergence; see, e.g., [23], [16, Sect. 7.5], [24], [25]. We focus on the class of split preconditioners for the normal equations, which formally transform the system (3) into

$$\left(L^{-1} A^T A L^{-T}\right)\left(L^T X\right) = L^{-1} A^T B,$$

where $L \in \mathbb{R}^{m \times m}$ is a given nonsingular matrix. We assume that linear systems with L and L^T can be solved efficiently. Typical examples of split preconditioners include incomplete factorizations of the matrix $A^T A$. Efficient implementations that avoid explicitly forming $A^T A$ are available; see, e.g., the routine `HSL_MI35` from the HSL library[2], which is a collection of Fortran codes for large-scale scientific computation.

The transformed system can be written in the form

$$\widehat{A}^T \widehat{A} \widehat{X} = \widehat{A}^T B, \tag{26}$$

where $\widehat{A} = AL^{-T}$ and $\widehat{X} = L^T X$. All block algorithms considered in Section 3 can then be formally applied to this preconditioned system (26), generating approximate solutions $\widehat{X}_k$. Using the relation between $\widehat{X}$ and X, we define the corresponding approximate solutions to the original problem as $X_k = L^{-T} \widehat{X}_k$. Note that, analogously to the single-vector case, the following identity holds:

$$(\widehat{X} - \widehat{X}_k)^T \widehat{A}^T \widehat{A} (\widehat{X} - \widehat{X}_k) = (X - X_k)^T A^T A (X - X_k).$$

This implies that the error estimation techniques discussed in the previous section remain directly applicable in the preconditioned setting, using the block coefficients computed within the preconditioned algorithms.

In the following, we present the preconditioned versions of the algorithms discussed in Section 3. The preconditioned HS-BCGLS algorithm is given as Algorithm 7, preconditioned DR-BCGLS as Algorithm 8, and preconditioned KT-BLSQR as Algorithm 9. In these three algorithms there are one solve of a linear system with L and one with L^T in every iteration.

In each algorithm, we indicate with arrows the lines where the coefficient matrices $\mathfrak{R}_j$ $(j = 0, \ldots, k)$ and Θ_j $(j = 0, \ldots, k-1)$, are computed. As before, if Gauss–Radau upper bounds are desired, the coefficient matrix $\Theta_j^{(\mu)}$ $(j = 0, \ldots, k)$ can be computed using the recurrence formula (7). The diagonal entries of these matrices can then be used in the adaptive error estimation techniques described in Section 4.

[2] `http://www.hsl.rl.ac.uk/`

Algorithm 7 Preconditioned HS-BCGLS (HS-PBCGLS)

input A, B, X_0, L, μ
$R_0 = B - AX_0$
$\widetilde{R}_0 = A^T R_0$
$Z_0 = L^{-T}L^{-1}\widetilde{R}_0$, $P_0 = Z_0$
$\rightarrow\ \mathfrak{R}_0 = \widetilde{R}_0^T Z_0$
for $k = 1, \ldots$ until convergence **do**
    $\Upsilon_{k-1} = [(AP_{k-1})^T AP_{k-1}]^{-1}(\widetilde{R}_{k-1}^T Z_{k-1})$
    $X_k = X_{k-1} + P_{k-1}\Upsilon_{k-1}$
    $R_k = R_{k-1} - AP_{k-1}\Upsilon_{k-1}$
    $\widetilde{R}_k = A^T R_k$
    $Z_k = L^{-T}L^{-1}\widetilde{R}_k$
    $\Xi_k = (\widetilde{R}_{k-1}^T Z_{k-1})^{-1}(\widetilde{R}_k^T Z_k)$
    $P_k = Z_k + P_{k-1}\Xi_k$
    $\rightarrow \Theta_{k-1} = \mathfrak{R}_{k-1}\Upsilon_{k-1}$, $\mathfrak{R}_k = \widetilde{R}_k^T Z_k$
end for

Algorithm 8 Preconditioned Dubrulle-R BCGLS (DR-PBCGLS)

input A, B, X_0, L, μ
$R_0 = B - AX_0$
$[Q_0, \Sigma_0] = \mathtt{qr}(L^{-1}A^T R_0)$
$S_0 = L^{-T}Q_0$
$\rightarrow \mathfrak{R}_0 = \Sigma_0^T \Sigma_0$
for $k = 1, \ldots$ until convergence **do**
    $Y_{k-1} = AS_{k-1}$
    $\Pi_{k-1} = (Y_{k-1}^T Y_{k-1})^{-1}$
    $X_k = X_{k-1} + S_{k-1}\Pi_{k-1}\Sigma_{k-1}$
    $[Q_k, \Psi_k] = \mathtt{qr}(Q_{k-1} - L^{-1}A^T Y_{k-1}\Pi_{k-1})$
    $S_k = L^{-T}Q_k + S_{k-1}\Psi_k^T$
    $\Sigma_k = \Psi_k \Sigma_{k-1}$
    $\rightarrow \Theta_{k-1} = \Sigma_{k-1}^T \Pi_{k-1}\Sigma_{k-1}$, $\mathfrak{R}_k = \Sigma_k^T \Sigma_k$
end for

6 Numerical experiments

In this section we describe numerical experiments comparing various algorithms for solving block least squares problems and experiments on bounding the associated errors. The experiments are performed in MATLAB R2023b with our implementation of the algorithms. The scripts for the algorithms and the experiments performed below are freely available from the GitHub repository [26].

Algorithm 9 Preconditioned KT-BLSQR (KT-PBLSQR)

input A, B, L, μ
$\widehat{X}_0 = 0$
$[U_1, \beta_1] = \mathtt{qr}(B)$
$[V_1, \alpha_1] = \mathtt{qr}(L^{-1}A^T U_1)$
$W_1 = V_1$, $\bar{\phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1^T$
$\rightarrow \mathfrak{R}_0 = \beta_1^T \alpha_1^T \alpha_1 \beta_1$
for $k = 1, 2, \ldots$ until convergence **do**
  $[U_{k+1}, \beta_{k+1}] = \mathtt{qr}(AL^{-T}V_k - U_k \alpha_k^T)$
  $[V_{k+1}, \alpha_{k+1}] = \mathtt{qr}(L^{-1}A^T U_{k+1} - V_k \beta_{k+1}^T)$
  $\begin{bmatrix} G_k^T, \begin{bmatrix} \rho_k \\ 0 \end{bmatrix} \end{bmatrix} = \mathtt{QR}\left(\begin{bmatrix} \bar{\rho}_k \\ \beta_{k+1} \end{bmatrix} \right)$
  $\begin{bmatrix} \Phi_k & \Omega_{k+1} \\ \bar{\Phi}_{k+1} & \bar{\rho}_{k+1} \end{bmatrix} = G_k \begin{bmatrix} \bar{\Phi}_k & 0 \\ 0 & \alpha_{k+1}^T \end{bmatrix}$
  $\widehat{X}_k = \widehat{X}_{k-1} + W_k \left(\rho_k^{-1} \Phi_k \right)$
  $W_{k+1} = V_{k+1} - W_k \left(\rho_k^{-1} \Omega_{k+1} \right)$
  $\rightarrow \Theta_{k-1} = \Phi_k^T \Phi_k$, $\mathfrak{R}_k = \Phi_k^T \Omega_{k+1} \Omega_{k+1}^T \Phi_k$
end for
$X_k = L^{-T} \widehat{X}_k$

As a measure of convergence, we consider the quantity

$$\tau_k = \sqrt{\mathrm{trace}(\mathfrak{E}_k)} = \sqrt{\mathrm{trace}((X - X_k)^T A^T A (X - X_k))}. \tag{27}$$

All algorithms are initialized with $X_0 = 0$. Unless otherwise specified, the exact solution X is tightly approximated using MATLAB's backslash operator.

6.1 Comparison of the three algorithms

Let us start with an example for which HS-BCGLS does not have difficulties to converge. We use the full-rank sparse matrix `illc1850` from the SuiteSparse Matrix collection[3]. This matrix is 1850×712 with $8,636$ nonzero entries, $\kappa(A) \approx 1.4 \times 10^3$, and a smallest singular value which is 1.511378×10^{-3}. The right-hand side matrix B has two random columns generated using the MATLAB `randn` command.

For this problem, as shown in Figure 1, the three algorithms converge similarly, even though DR-BCGLS and KT-BLSQR are slightly faster than HS-BCGLS.

Let us now consider examples for which HS-BCGLS has convergence problems. Matrices $P(n, m, d, p)$ were defined by Paige and Saunders [3, p. 62] for testing least squares algorithms. We use $P(80, 40, 1, 3)$ which is a 80×40 matrix with $\kappa(A) = 6.4 \times 10^4$. Let $A = USV^T$ be the economy-size SVD of A. The block right-hand side B with four columns ($s = 4$) is computed as

$$\widetilde{B} = US^{-1}K,$$

[3] https://sparse.tamu.edu/HB/illc1850

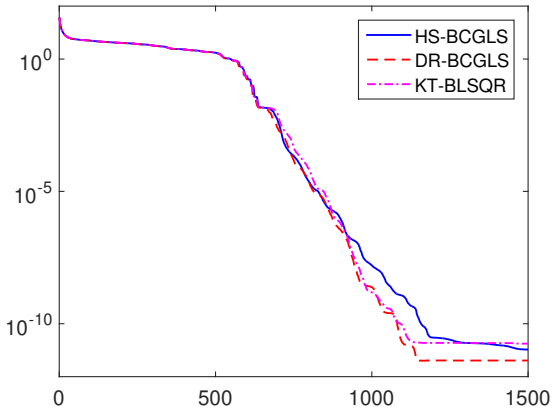

Fig. 1 The quantity τ_k in (27) for the three algorithms and the matrix `illc1850`

where K is a 40×4 random matrix computed with the MATLAB `randn` function after setting `rng('default')`. Then, B is obtained by normalizing the columns of $\widetilde{B}$. The matrix B is of full rank with a smallest singular value which is approximately 5.9782×10^{-2}. In Figure 2, we see that HS-BCGLS starts to converge, then slows down after iteration 20, and finally stagnates after iteration 35. DR-BCGLS and KT-BLSQR both converge quite similarly around iterations 13-14 and give more or less the same final accuracy, which is many orders of magnitude better than that of HS-BCGLS.

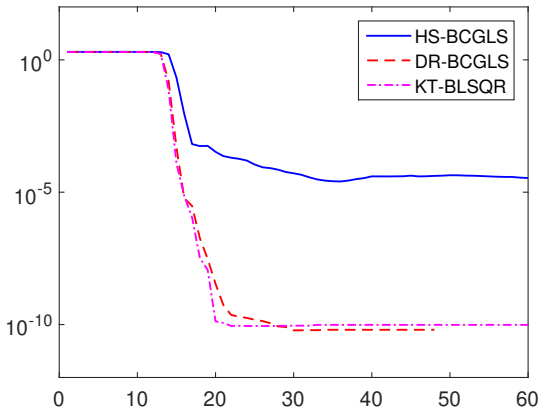

Fig. 2 The quantity τ_k in (27) for the three algorithms and the matrix $P(80, 40, 1, 3)$

6.2 An example with initial deficiency

In the next experiment, we use the same matrix $P(80, 40, 1, 3)$, but the right-hand side is computed as

$$\widetilde{B} = US^{-1}\left(\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \otimes e\right),$$

with $e = [1, \ldots, 1]^T \in \mathbb{R}^{m/2}$. Then, as above, B is obtained by normalizing the columns of $\widetilde{B}$. In this example, we have three columns in the block right-hand side B, but its rank is only 2. Figure 3 shows that HS-BCGLS stagnates from the beginning, when DR-BCGLS and KT-BLSQR converge after an initial stagnation phase. The initial rank deficiency does not prevent their convergence.

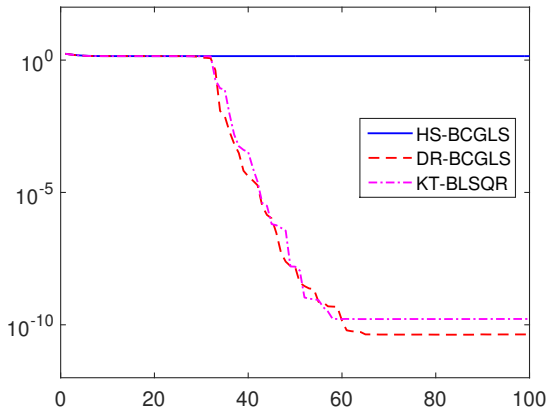

Fig. 3 The quantity τ_k in (27) for the three algorithms with initial rank deficiency and the matrix $P(80, 40, 1, 3)$

In this example, the second system converges fast while the first and third systems converge more slowly. Figure 4 shows the A^TA-norms of the error for the first and second systems solved using DR-BCGLS and KT-BLSQR. Clearly, these two algorithms are not affected by the differences in convergence speed between the various systems.

These two examples show that there are cases for which HS-BCGLS does not converge or converges slowly and could stagnate before delivering accurate approximations to the solutions. From now on, we therefore only concentrate on DR-BCGLS and KT-BLSQR.

6.3 A polynomial fitting problem

We now describe a simple polynomial fitting problem that will later be used to test the algorithms. Suppose we are given distinct points x_i, $i = 1, \ldots, n$, in the interval $[-1, 1]$, and let $m < n$ be a given integer. We observe the values of functions f_j, $j = 1, \ldots, s$, at these points x_i, and our goal is to find polynomial approximations that best fit

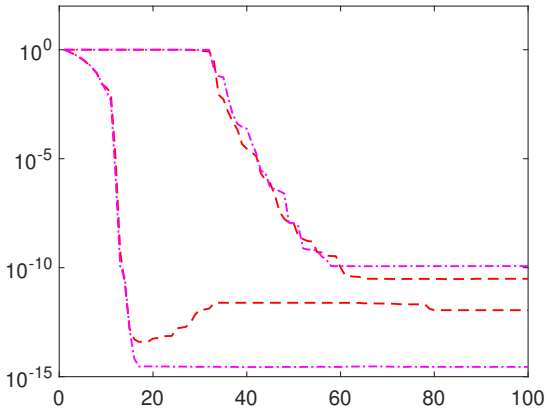

Fig. 4 A^TA-norms of the errors for the first (right curves) and second systems (left curves) for DR-BCGLS (dashed) and KT-BLSQR (dot-dashed) with initial rank deficiency and the matrix $P(80, 40, 1, 3)$

these values in the least squares sense. Using the basis of Chebyshev polynomials, we seek polynomials $p_{m,j}$ of degree m,

$$p_{m,j}(x) = \sum_{i=0}^{m} \alpha_{i,j} C_i(x),$$

where C_i is the i-th Chebyshev polynomial, such that

$$\begin{pmatrix} f_1(x_1) & \dots & f_s(x_1) \\ \vdots & & \\ f_1(x_n) & \dots & f_s(x_n) \end{pmatrix} \approx A \begin{pmatrix} \alpha_{0,1} & \dots & \alpha_{0,1} \\ \vdots & & \\ \alpha_{m-1,1} & \dots & \alpha_{m-1,2} \end{pmatrix},$$

with an $n \times m$ matrix

$$A = \begin{pmatrix} C_0(x_1) & \dots & C_m(x_1) \\ \vdots & & \\ C_0(x_n) & \dots & C_m(x_n) \end{pmatrix}.$$

In our experiments, we choose equispaced points and consider the example functions

$$f_j(x) = \frac{\cos(4jx)}{1 + 0.1\sin(1000x)^2}, \quad j = 1, \dots, s.$$

To approximate these functions uniformly in Chebfun [27], we would require polynomials of degree several thousands. However, to approximate only the basic shapes, given by the terms $\cos(4jx)$, it suffices to use polynomials of relatively low degrees, such as 20, 30, 40, and 50.

6.4 Computation of upper bounds

In the experiments of this subsection we test the upper bound in block algorithms. As discussed in Section 4.2, the error norms in solving individual systems can be estimated as well as the trace quantity τ_k in (27). Due to space limitations, we present here only the results for τ_k. The codes, however, allow to also compute bounds of the error norms for individual systems.

For these experiments, we consider two test problems: the polynomial fitting problem described in Section 6.3, and the matrix `sls` of size $1,748,122 \times 62,729$ with $6,804,304$ nonzero entries from the SuiteSparse Matrix collection[4] preconditioned using an incomplete factorization computed with the `HSL_MI35` code from the HSL library[5]. The right-hand sides for `sls` are generated as follows. The first right-hand side is computed as in [10, Sect. 6], i.e.,

```
x = ones(size(A,2),1);
x(2:2:end) = -2;
x(5:5:end) = 0;
b_tmp = A * x;
b_1 = b_tmp + randn(size(b_tmp)) * norm(b_tmp);
b_1 = b_1/norm(b_1);
```

The second, consistent, right-hand side is given as

```
b_2 = -5 * A(:,5) + 7 * A(:,10); b_2 = b_2/norm(b_2);
```

When we set $s = 4$ in the experiments, the third and fourth right-hand sides are generated with random entries (MATLAB `randn` function) and normalized to 1.

As above, for the polynomial fitting problem, the exact solution X, only needed to evaluate the error $X - X_k$, is computed using MATLAB's backslash operator. For the tests with the matrix `sls`, X is computed using a large number of KT-BLSQR iterations.

To compute the upper bound, we need a lower bound of the smallest eigenvalue of A^TA. For the polynomial fitting problem, we do not consider preconditioning and compute an approximation to the smallest singular value $\sigma_{\min}(A)$ of the matrix A by MATLAB `svds` function. The parameter μ, a lower bound on $\lambda_{\min}(A^TA) = \sigma^2_{\min}(A)$, is given as

$$\mu_1 = \sigma^2_{\min}(A)(1 - 10^{-4}) \qquad \text{and} \quad \mu_2 = \sigma^2_{\min}(A)(1 - 10^{-10}). \tag{28}$$

For the choice of parameters $n = 3000$, $m = 50$, we have $\sigma^2_{\min}(A) = 78.6711$. When setting $n = 3000$, $m = 300$, we get $\sigma^2_{\min}(A) = 7.4086 \times 10^{-7}$.

For the preconditioned `sls` matrix, the parameter $\mu = 6 \times 10^{-4}$ is set as the largest value (with an increment 10^{-5}) such that the upper bound can be computed in all iterations.[6]

[4] https://sparse.tamu.edu/Bates/sls

[5] `http://www.hsl.rl.ac.uk/`

[6] In other words, the upper bound with $\mu = 6.1 \times 10^{-4}$ failed in some iterations. This typically indicates that $6.1 \times 10^{-4} > \lambda_{\min}(\widehat{A}^T\widehat{A})$.

In our experiments, the bounds behave very similarly for both DR-BCGLS and KT-BLSQR. We therefore only show the results for one of the algorithms in each of the experiments.

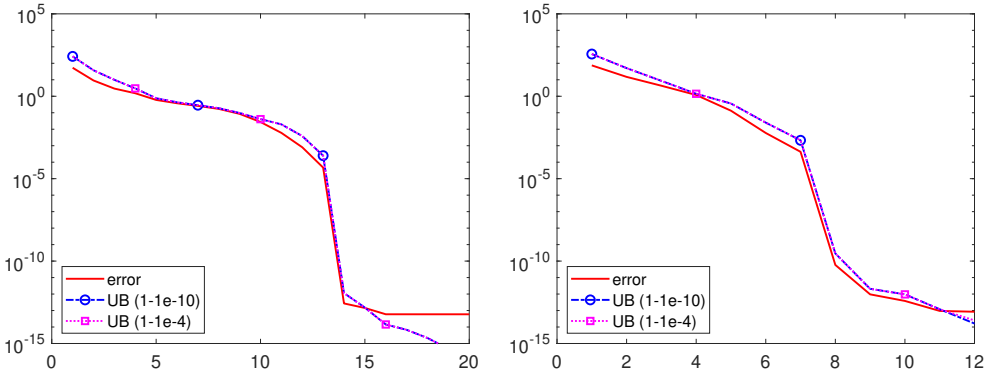

Fig. 5 DR-BCGLS: The quantity τ_k in (27) for the polynomial fitting problem with parameters $n = 3000$, $m = 50$, and $s = 2$ (left) and with $s = 4$ (right) together with its upper bounds for two choices of μ

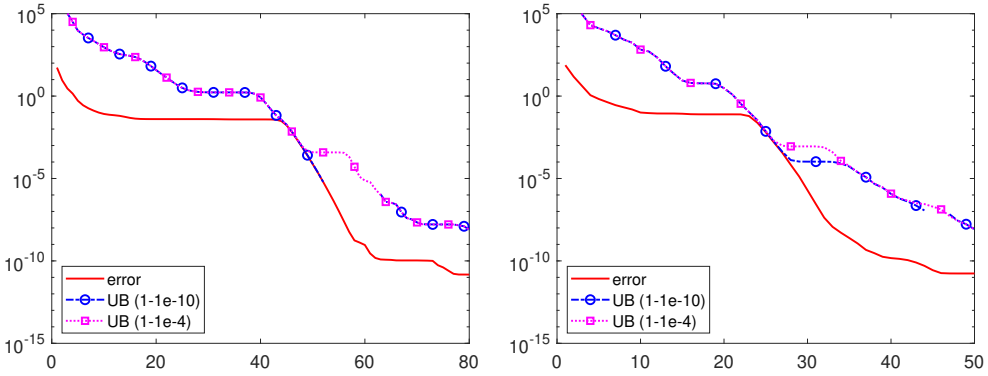

Fig. 6 KT-BLSQR: The quantity τ_k in (27) for the polynomial fitting problem with parameters $n = 3000$, $m = 300$, and $s = 2$ (left) and with $s = 4$ (right) together with its upper bounds for two choices of μ

Figure 5 depicts the results for the polynomial fitting problem with parameters $n = 3000$, $m = 50$, and $s = 2, 4$ and DR-BCGLS. The upper bounds are quite close to the measure of the error τ_k in (27), at least, as long as the maximum attainable accuracy has not been reached. They do not depend much on the chosen values of μ.

Results with a more difficult problem (from the perspective of error estimation) with $n = 3000$, $m = 300$, $s = 2, 4$, and KT-BLSQR are displayed in Figure 6. For this problem, the upper bounds are far from being sharp. Moreover, in some iterations, the upper bound for the parameter μ_2 is not computed due to a numerical singularity

of the matrix $\mu(\Theta_{k-1}^{(\mu)} - \Theta_{k-1}) + \mathfrak{R}_k$. A similar behavior —where the bound initially overestimates the error, becomes tight for a few iterations, and then diverges again— has been observed in both single-system and block CG methods; see, e.g., [28, Sect. 4], [9, Fig. 9.3].

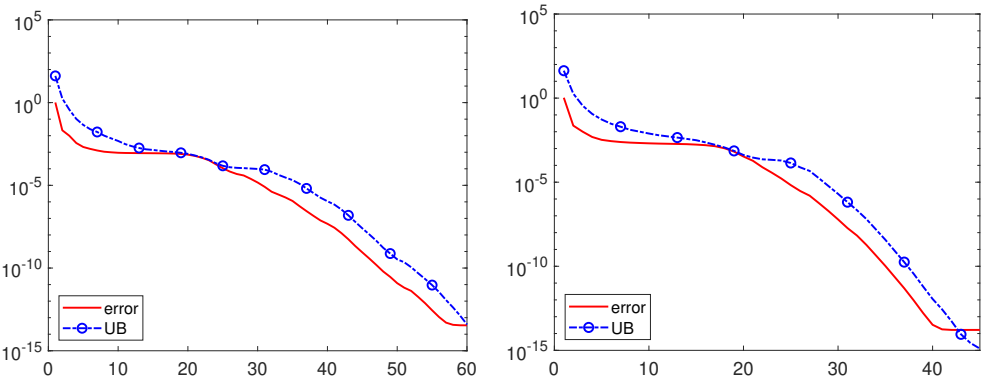

Fig. 7 DR-PBCGLS: The quantity τ_k in (27) for the matrix `sls` with $s = 2$ (left) and with $s = 4$ (right) together with its upper bound with $\mu = 6 \times 10^{-4}$

For the `sls` matrix the results for DR-PBCGLS are displayed in Figure 7. The bound is tight for a few iterations around 22nd, respectively 19th iterations. However, in most of the iterations, the upper bound overestimates the error by around two orders of magnitude. The overestimation of the upper bound can be reduced (or nearly removed) by considering the adaptivity described at the end Section 4.2, which will be tested in the upcoming section for the lower bound. The codes on GitHub repository [26] also include the adaptive upper bound.

6.5 Computation of lower bounds with adaptivity

Finally, we test the computation of a lower bound on τ_k in (27) with an adaptive choice of the delay. We consider the same test problems as in the previous section. In the figures, we plot analogous quantities as in [10, 20], i.e., τ_k with the lower bound in the upper panel, the prescribed relative tolerance $\tau = 0.25$ from the block analogue to (22) together with the relative error (middle panel) and the adaptive delay together with the ideal value, that is, the minimal value that assures (22), in the lower panel.

Figure 8 depicts the results for the polynomial fitting problem with parameters $n = 3000$, $m = 50$, $s = 2, 4$, and KT-BLSQR. The results with $n = 3000$, $m = 300$, and $s = 2, 4$ are given in Figure 9 with DR-BCGLS. Therein, we observe an underestimation of the error from the 10th to 20th iterations where the error τ_k starts to stagnate. However, in the later iterations where one might want to stop the solver, the error τ_k is tightly estimated with a nearly ideal delay.

For the matrix `sls`, the results are given in Figure 10. Here also, we observe some (mild) underestimation of the error τ_k in some iterations at the beginning of the stagnation phase but for most of the iterations, the adaptively chosen delay is close to the ideal value and τ_k is tightly estimated.

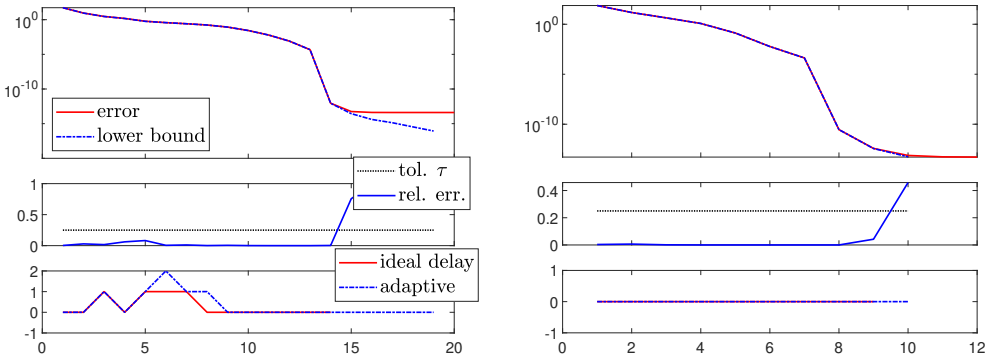

Fig. 8 KT-BLSQR: The quantity τ_k in (27) for the polynomial fitting problem with parameters $n = 3000$, $m = 50$, and $s = 2$ (left) and with $s = 4$ (right) together with the adaptive lower bound

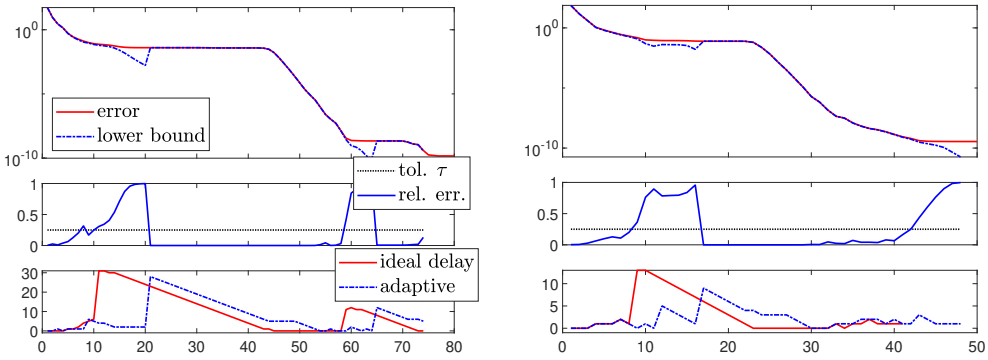

Fig. 9 DR-BCGLS: The quantity τ_k in (27) for polynomial fitting problem with parameters $n = 3000$, $m = 300$, and $s = 2$ (left) and with $s = 4$ (right) together with the adaptive lower bound

Overall, the lower bound with an adaptive choice of the delay provides very satisfactory results, as it enables tight error estimates in most iterations with a nearly optimal delay. Therefore, we could recommend its use for stopping the iterations of the solvers.

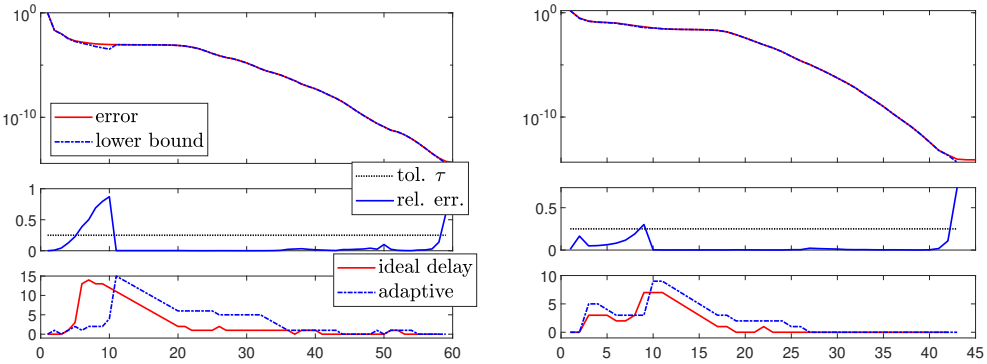

Fig. 10 KT-PBLSQR: The quantity τ_k in (27) for the matrix `sls` with $s = 2$ (left) and with $s = 4$ (right) together with the adaptive lower bound

7 Conclusions

In this paper, we introduced two algorithms, DR-BCGLS and KT-BLSQR, which are suitable for efficiently solving least squares problems with multiple right-hand sides. These algorithms can be viewed as generalizations of the classical CGLS and LSQR algorithms to the block case. A key ingredient of both algorithms is the use of what we refer to as Dubrulle's approach, which maintains a constant block size and prevents breakdowns even in rank-deficient situations. As a result, no complicated deflation techniques are needed, and the algorithms remain simple and easy to implement. Numerical experiments with rank-deficient block right-hand sides confirmed these expectations.

We have shown how to estimate an important quantity: the A^TA-norm of the error. This quantity serves as a useful measure of the solution's progress and plays a key role in stopping criteria based on the normwise backward error for the normal equations. Building on our previous work on error estimation in CG [14, 20], CG-like methods [10], and block CG algorithms [9], we derived both lower and upper bounds for the A^TA-norm of the error individually for each system and for the trace of the associated bilinear form. We demonstrated how these bounds can be adaptively refined to produce estimates that heuristically attain a prescribed level of accuracy. These estimates can then be incorporated into practical stopping criteria. Numerical experiments confirm that the adaptive techniques perform quite well in practice.

To avoid theoretical complications, we assumed that A has full column rank. However, our experiments indicate that the considered algorithms perform reliably even when A is rank-deficient. Further analyses and a deeper understanding of the behavior of the considered algorithms that use Dubrulle's approach – both in exact and finite-precision arithmetic – remain topics for future investigations.

References

[1] Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Natl. Bur. Stand. **49**(6), 409–436 (1952)

[2] Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia, (PA) (2003)

[3] Paige, C.C., Saunders, M.A.: LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Soft. (TOMS) **8**(1), 43–71 (1982)

[4] Paige, C.C., Saunders, M.A.: Algorithm 583; LSQR: Sparse linear equations and least-squares problems. ACM Trans. Math. Soft. (TOMS) **8**(2), 195–209 (1982)

[5] Golub, G.H., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. SIAM J. Numer. Anal. **2**(2), 205–224 (1965)

[6] O'Leary, D.P.: The block conjugate gradient algorithm and related methods. Linear Algebra Appl. **29**, 293–322 (1980)

[7] O'Leary, D.P.: Parallel implementation of the block conjugate gradient algorithm. Parallel Comput. **5**(1-2), 127–140 (1987)

[8] Tichý, P., Meurant, G., Šimonová, D.: Block CG algorithms revisited. Numer. Algorithms, (2025). published online

[9] Meurant, G., Tichý, P.: Error norm estimates for the block conjugate gradient algorithm. Accepted for publication in SIAM J. Matrix Anal. Appl. (2025)

[10] Papež, J., Tichý, P.: Estimating the error in CG-like algorithms for least-squares and least-norm problems. Numer. Algorithms **97**(1), 1–28 (2024)

[11] Ji, H., Li, Y.: A breakdown-free block conjugate gradient method. BIT Numerical Mathematics **57**, 379–403 (2017)

[12] Dubrulle, A.A.: Retooling the method of block conjugate gradients. Electron. Trans. Numer. Anal. **12**, 216–233 (2001)

[13] Meurant, G., Tichý, P.: Dubrulle's variants of the block conjugate gradient algorithm. In preparation (2025)

[14] Meurant, G., Tichý, P.: Error Norm Estimation in the Conjugate Gradient Algorithm. SIAM, Philadelphia, (PA) (2024)

[15] Strakoš, Z., Tichý, P.: On error estimation in the conjugate gradient method and why it works in finite precision computations. Electron. Trans. Numer. Anal. **13**, 56–80 (2002)

[16] Björck, A.: Numerical Methods for Least Squares Problems, Second Edition. SIAM, Philadelphia (PA) (2024)

[17] Ji, H., Li, Y.: Block conjugate gradient algorithms for least squares problems. J. Comput. Appl. Math. **317**, 203–217 (2017)

[18] Golub, G.H., Luk, F.T., Overton, M.L.: A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. ACM Trans. Math. Software **7**(2), 149–169 (1981)

[19] Karimi, S., Toutounian, F.: The block least squares method for solving non-symmetric linear systems with multiple right-hand sides. Appl. Math. Comput. **177**(2), 852–862 (2006)

[20] Meurant, G., Papež, J., Tichý, P.: Accurate error estimation in CG. Numer. Algorithms **88**(3), 1337–1359 (2021)

[21] Chang, X.-W., Paige, C.C., Titley-Peloquin, D.: Stopping criteria for the iterative solution of linear least squares problems. SIAM J. Matrix Anal. Appl. **31**(2), 831–852 (2009)

[22] Jiránek, P., Titley-Peloquin, D.: Estimating the backward error in LSQR. SIAM J. Matrix Anal. Appl. **31**(4), 2055–2074 (2009/10)

[23] Bru, R., Marín, J., Mas, J., Tůma, M.: Preconditioned iterative methods for solving linear least squares problems. SIAM J. Sci. Comput. **36**(4), 2002–2022 (2014)

[24] Scott, J., Tůma, M.: Sparse linear least-squares problems. Acta Numerica **34**, 891–1010 (2025)

[25] Havelková, E., Hnětynková, I.: Preconditioning of LSQR and CGLS: Variants, properties and relations. In: Numerical Mathematics and Advanced Applications ENUMATH 2023, Volume 1, pp. 415–424. Springer, Cham (2025)

[26] Meurant, G., Papež, J., Tichý, P.: Block CG-like methods with error estimate. GitHub repository: `https://github.com/JanPapez/Block-CGlike-methods-with-error-estimate` (2025)

[27] Driscoll, T.A., Hale, N., Trefethen, L.N.: Chebfun Guide. Pafnuty Publications, Oxford (2014). http://www.chebfun.org/docs/guide/

[28] Meurant, G., Tichý, P.: The behavior of the Gauss-Radau upper bound of the error norm in CG. Numer. Algorithms **94**(2), 847–876 (2023)