Preprint BUW-IMACM 13/08

Sebastian Birk, Andreas Frommer

A Deflated Conjugate Gradient Method for Multiple Right Hand Sides and Multiple Shifts

June 2013

A DEFLATED CONJUGATE GRADIENT METHOD FOR MULTIPLE RIGHT HAND SIDES AND MULTIPLE SHIFTS

SEBASTIAN BIRK AND ANDREAS FROMMER[*]

Abstract. We consider the task of computing solutions of linear systems that only differ by a shift with the identity matrix as well as linear systems with several different right-hand sides. In the past, Krylov subspace methods have been developed which exploit either the need for solutions to multiple right-hand sides (e.g. deflation type methods and block methods) or multiple shifts (e.g. shifted CG) with some success. In this paper we present a block Krylov subspace method which, based on a block Lanczos process, exploits both features—shifts and multiple right-hand sides—at once. Such situations arise, for example, in lattice quantum chromodynamics (QCD) simulations within the Rational Hybrid Monte Carlo (RHMC) algorithm. We present numerical evidence that our method is superior compared to applying other iterative methods to each of the systems individually as well as, in typical situations, to shifted or block Krylov subspace methods.

Key words. CG method, block Krylov subspaces, multiple right-hand sides, multishift methods, deflation, QCD

AMS subject classifications. 65F10

1. Introduction.

In this paper we consider solving systems of linear equations of the form

$$(\sigma_j I + A)x_{i,j} = b_i \tag{1.1}$$

where $\sigma_j I + A \in \mathbb{C}^{n\times n}$ is a Hermitian positive definite (hpd) matrix for every shift $\sigma_j \in \mathbb{C}$, $x_{i,j}, b_i \in \mathbb{C}^n$, $i = 1,\ldots,m$ and $j = 1,\ldots,s$.

Systems like these arise naturally in lattice QCD, where the b_i represent multiple source terms and $A = M^H M$, with M being a discrete version of the Dirac operator, e.g., the Wilson fermion matrix. For example the Rational Hybrid Monte Carlo algorithm [16] needs to solve $A^\nu x_i = b_i$ with $\nu \in (-1,1)$ for multiple right-hand sides (rhs) b_i. The matrix power A^ν therein is computed using a rational approximation which can be represented as a partial fraction expansion (pfe) thus giving rise to multiple shifted systems. Another application is the computation of symplectic integrators that require the evaluation of $A^\nu B A^\nu x_i = b_i$ for multiple right-hand sides. In that case the inner multiplication with A^ν is approximated via a pfe. The solutions to all the individual terms of the pfe can then be regarded as multiple right-hand sides for the outer multiplication with A^ν even if there was only one right-hand side $B = b_1$ in the beginning.

On first attempt, the systems (1.1) can be solved separately, e.g., by applying the conjugate gradients method anew for every shift and every right-hand side. As one might expect this turns out to be an inferior approach, since there exist methods making use of the nature of the problem, resulting in a considerable gain in computation time.

By arranging the m right-hand sides and the corresponding solutions in the matrices

$$B = [b_1|\cdots|b_m] \text{ and } X_j = \left[x_{1,j}|\cdots|x_{m,j}\right]$$

we can rewrite the systems (1.1) as

$$(\sigma_j I + A)X_j = B. \tag{1.2}$$

[*]Fachbereich C, Mathematik und Naturwissenschaften, Bergische Universität Wuppertal, D-42097 Wuppertal, Germany, {birk,frommer}@math.uni-wuppertal.de, This work was supported by Deutsche Forschungsgemeinschaft through the Collaborative Research Centre SFB-TR 55 "Hadron physics from Lattice QCD"

One might now try to solve these block systems for each shift σ_j and hopefully benefit from doing so. Methods capable of solving all right-hand sides at once in systems like (1.2) have been around for more than thirty years and are called block methods. Essentially the idea of block methods is to build up a common subspace, typically a block Krylov subspace, for all right-hand sides, producing a richer space from which to find iterates. An early description of a block conjugate gradients method dates back to [20]. As was observed in [20], naïve implementations can suffer from linear dependencies which result in a breakdown. Thus, deflation, i.e. removing vectors that became linearly dependent, was proposed as a cure. However, the algorithmic details were not worked out in [20]. In [18] details on the reduction of the block size for addressing the problems caused by linear dependencies were worked out. The resulting variable block CG method from [18] and [19] was then used to accelerate the solution of single right-hand side systems. A detailed description of deflation and explanation of why it is needed can be found in [14]. More recent approaches in developing a CG method for block systems like in [8] try to overcome the problem of singular or nearly singular matrices by computing (QR-)factorisations of the involved matrices and therefore successfully disguising the singularities from the method. Other block methods not especially tailored to hpd include, e.g., block-GMRES [17], Block BiCGStab [26] and block MINRES [24].

Another approach to solve block systems is to treat them as a sequence of linear systems to the same matrix. So called seed methods [1, 22] solve these systems sequentially and use Krylov subspace information gathered in solving one seed system to adjust the starting vectors of the remaining unsolved systems. This update of the starting vectors has to be done in every iteration of the seed system. Seed methods usually work best when the right-hand sides are closely related. Deflation methods [25, 9] pursue a different approach. While solving one system they compute an approximation to the smallest eigenpairs. The obtained eigenvector approximations can either be used to base a preconditioner upon or to adjust the starting vectors for solving the remaining systems.

Instead of grouping the right-hand sides into blocks, another approach to solve (1.1) is to apply shifted methods like shifted CG, see [12]. Shifted CG makes use of the fact that Krylov subspaces are shift-invariant, i.e. for solving the system $(\sigma_j I + A)x = b$ with the CG method one builds up the same Krylov subspace, independently of the shift σ_j. Since usually the multiplications with the matrix A are the dominant cost factor, huge reductions in computation time are attainable by building the Krylov subspace just for one seed system and computing iterates for all shifted systems. As shown in [12] the shifted systems can be dealt with at almost no additional cost using shifted CG.

Previous work on methods combining multiple shifts and multiple right-hand sides includes GMRES [4] and QMR [10] based methods. Both types focus on non-Hermitian matrices where the former—being a GMRES method—uses long recurrences with deflated restarts. While the restarts improve on the memory requirement they deteriorate the optimal convergence properties of full GMRES. The QMR method in [10] needs a multiplication with A and A^H in every step. However, the multiplication with A^H does not contribute to the solution, since it is only needed to span an additional space for orthogonalisation.

We present a new, deflated shifted block CG method (DSBlockCG) that merges multiple aspects: the block idea, the idea of computing solutions for the shifted systems alongside one seed system and the focus on hpd matrices. It is based on a block Lanczos process which is the restriction to the Hermitian case of the non-symmetric block Lanczos process from [2]. This process is capable of handling multiple starting vectors and includes deflation.

After a short overview over block Krylov subspaces in section 2 we describe the deflated block Lanczos process in detail in section 3. In section 4 we work out the algorithmic details of our deflated shifted block CG method before we demonstrate its efficiency with a series of numerical experiments in section 6. We end this introduction explaining some of our notation. Variables having two lower indices like $x_{i,j}$ depend on the right-hand side b_i as well as on the shift σ_j. Single lower indices are to be understood by context. Upper indices in brackets $x^{(k)}$ or square brackets $X^{[k]}$ refer to the iteration index k. The notation $[A_1|A_2|\cdots|A_k]$ indicates a matrix which is composed of the vectors or matrices A_i as its columns.

2. Block Krylov subspace methods.

For ease of presentation we first disregard the shifts, i.e. we deal with systems

$$Ax_i = b_i, \quad i = 1,\dots,m.$$

Without loss of generality we assume that for each right-hand side b_i our initial guess is the zero vector. If we have a better initial guess $x_i^{(0)}$ for the i-th right-hand side, we just have to replace b_i by the residual $b_i - Ax_i^{(0)}$.

We define the *k-th block Krylov subspace* with respect to A and $B = [b_1|\cdots|b_m]$ as

$$\begin{aligned} K_k(A,B) &:= \text{colspan}\left(\left[\, B \mid AB \mid A^2B \mid \cdots \mid A^{k-1}B \,\right]\right) \\ &= \text{span}\left\{b_1,\dots,b_m,Ab_1,\dots,Ab_m,A^2b_1,\dots,A^{k-1}b_m\right\}. \end{aligned} \tag{2.1}$$

The Krylov subspaces $K_k(A,b_i) := \text{span}\left\{b_i,Ab_i,\dots,A^{k-1}b_i\right\}$ are contained in $K_k(A,B)$. While the dimension of each space $K_k(A,b_i)$ is k (unless we have reached an invariant subspace), the dimension of the block subspace can be smaller than mk. This is due to the fact that some of the subspaces $K_k(A,b_i)$ can have non-trivial intersections even when all the b_i are linearly independent.

The block conjugate gradient methods in [8] and [20] create block iterates

$$X^{(k)} = [x_1^{(k)}|\dots|x_m^{(k)}]$$

with $x_i^{(k)} \in K_k(A,B)$ and advance from $K_k(A,B)$ to $K_{k+1}(A,B)$ in each step. The iterates $X^{(k)}$ are obtained such that they satisfy the Galerkin condition

$$X^{(k)} \in K_k(A,B) \text{ and } R^{(k)} = B - AX^{(k)} \perp K_k(A,B),$$

which in the non-block case reduces to the classical variational characterisation of the CG iterates. Let $\mathbf{V}^{[k]}$ denote a matrix whose columns form a basis of $K_k(A,B)$. Then the Galerkin condition is equivalent to

$$X^{(k)} = \mathbf{V}^{[k]} \cdot \left((\mathbf{V}^{[k]})^H A \mathbf{V}^{[k]}\right)^{-1} \cdot (\mathbf{V}^{[k]})^H B. \tag{2.2}$$

Following [20], Algorithm 1 displays a first version of a block CG method. It will break down due to a singular matrix $R^{(k-2)H}R^{(k-2)}$ in line 4 if the block subspace $K_{k-1}(A,B)$ has dimension less than $m(k-1)$. To avoid these breakdowns one can monitor the rank of the matrices $P^{(k)}$ and $R^{(k)}$ as suggested in [20]. If one of these matrices loses full rank, the system belonging to the linearly dependent column has to be removed and the iteration can continue on the remaining systems. The drawback is that the bookkeeping in the resulting algorithm is quite complicated, and checking for rank deficiency is costly and has to be done in each step.

In [8] various other modifications of Algorithm 1 are proposed to resolve the problem of rank deficiency. The key idea is to use QR factorisations of $P^{(k)}$ and/or $R^{(k)}$ and to enforce full rank for Q even if $P^{(k)}$ or $R^{(k)}$ are rank deficient. In this manner the modified block conjugate

Algorithm 1: BlockCG (without deflation)

```
  Data: $A \in \mathbb{C}^{n\times n}$ hpd, linearly independent $B = [b_1|\dots|b_m] \in \mathbb{C}^{n\times m}$
  Result: solution $X$ of $AX = B$
1 begin
2     $X^{(0)} = 0; R^{(0)} = B; R^{(-1)} = I_{n,m}; P^{(0)} = 0$
3     for $k = 1, 2, \dots$ until convergence do
4         $S^{(k-1)} = (R^{(k-2)H}R^{(k-2)})^{-1}R^{(k-1)H}R^{(k-1)}$
5         $P^{(k)} = R^{(k-1)} + P^{(k-1)}S^{(k-1)}$
6         $T^{(k)} = (P^{(k)H}AP^{(k)})^{-1}R^{(k-1)H}R^{(k-1)}$
7         $X^{(k)} = X^{(k-1)} + P^{(k)}T^{(k)}$
8         $R^{(k)} = R^{(k-1)} - AP^{(k)}T^{(k)}$
```

gradients algorithm inverts only non-singular matrices, thus successfully disguising the rank deficient matrices. This method has the advantage that there is no need for book keeping of those vectors that became linearly dependent. On the downside one always invests the equivalent of m matrix-vector multiplications with A per step, even when this would not be necessary due to rank deficiency.

3. The deflated block Lanczos process. In [2] and [11] a non-symmetric block Lanczos-type process was derived for general non-Hermitian matrices. Several simplifications apply if this process is used for Hermitian matrices, and it is the purpose of this section to describe the resulting deflated block Lanczos process for Hermitian matrices in detail. Based on the block Lanczos process, we will then be able to develop another stable variant of the block CG method which—as opposed to those described in section 2—exploits rank deficiency to reduce the number of multiplications with A.

The idea in [2, 11] is to compute a nested orthonormal basis for the block Krylov subspaces $K_k(A,B)$ one vector at a time. For simplicity, assume first that there is no deflation, i.e. all the block spaces $K_k(A,B)$ have full dimension mk. We denote

$$K_{k,i}(A,B) = K_{k-1}(A,B) + \operatorname{span}\left\{A^k b_1, \dots, A^k b_i\right\}$$

the subspaces lying "between" $K_{k-1}(A,b)$ and $K_k(A,B)$. We proceed by extending an orthonormal basis $\mathscr{V}_{k,i-1} = \{v^{(1)}, \dots, v^{(mk+i-1)}\}$ of $K_{k,i-1}$ to one of $K_{k,i}$ by orthogonalising $Av^{(m(k-1)+i)}$ against $\mathscr{V}_{k,i-1}$. Since A is Hermitian, we actually only have to orthogonalise against the vectors $v^{(m(k-2)+i)}, \dots, v^{(mk+i-1)}$. We thus obtain a Lanczos-type relation of the form

$$A\mathbf{V}^{[k]} = \mathbf{V}^{[k+1]}\mathbf{T}^{[k+1,k]}, \tag{3.1}$$

where the orthonormal columns of $\mathbf{V}^{[k+1]} = [v^{(1)}|\dots|v^{(m(k+1))}] \in \mathbb{C}^{n\times mk}$ represent a basis for $K_{k+1}(A,B)$, and the matrix $\mathbf{T}^{[k+1,k]} \in \mathbb{C}^{m(k+1)\times mk}$ is block tridiagonal with triangular off-diagonal blocks. Figure 3.1 displays the structure of $\mathbf{T}^{[k+1,k]}$. The bold font and the indices in square brackets emphasise the block structure of the respective matrices.

Deflation occurs when the orthogonalisation of $Av^{(m(k-1)+i)}$ against $\mathscr{V}_{k,i-1}$ leaves us with the zero vector (such exact deflation is unlikely to be seen in practice) or a vector with very small norm (which we will call inexact deflation). We then skip the remaining computations for the current step and proceed with orthogonalising the next vector $Av^{(m(k-1)+i+1)}$ (which

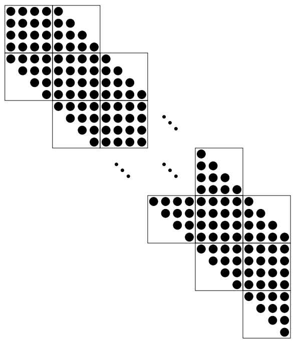

FIG. 3.1. *Example for the sparsity pattern of the matrix* $\mathbf{T}^{[k+1,k]}$ *in case of no deflation for 4 right-hand sides.*

is $Av^{(mk+1)}$ if $i = m$). In the case of inexact deflation we save the respective vector since we still have to orthogonalise against it in the following steps.

Algorithm 2 describes the block Lanczos process including deflation and it is a stripped down version of the non-symmetric block Lanczos-type process from [11]. We modified it to be suited for hpd matrices. The indices of the (inexactly) deflated vectors are kept in the set I. Since deflation may occur several times, we have to be careful with the indexing to use. In Algorithm 2, i is used as the index for the vectors of the orthogonal basis. The index μ keeps track of the vector from which we get the next candidate basis vector by orthogonalising $Av^{(\mu)}$, and the 'history index' h_μ denotes the smallest index of the basis vectors we have to orthogonalise against. In the non-deflated case $\mu = i - m$ and $h_\mu = i - 2m$. Once we have deflated d vectors, we have $\mu = i - m + d > i - m$.

In the table in Figure 3.2 we demonstrate in an example how the iteration counters i and μ and the history indices h_i and h_μ are related. The numbers m_i and m_μ will be explained later. For this example let $m = 4$ and assume that deflation occurs in steps $\mu = 4$ and $\mu = 9$. The table in Figure 3.2 shows that the iteration counter i can not be used to uniquely describe the progress of the iteration because i only counts the successfully created basis vectors. But since μ counts every candidate basis vector, we can use μ to unambiguously describe the progress of Algorithm 2. Therefore, from now on we use μ as the iteration index and i_μ is the corresponding index i in iteration step μ.

As soon as deflation occurs, the simple block Lanczos relation (3.1) does not hold any more. In the transition from $\mathbf{V}^{[k]}$ to $\mathbf{V}^{[k+1]}$ the dimension of the Krylov subspace is now increased by less than m. We emphasise this in our notation and instead of (3.1) we have the relation

$$AV^{(\mu)} = V^{(i_\mu)}\widetilde{T}^{(i_\mu,\mu)} + \widetilde{V}_{\text{defl}}^{(\mu)}. \tag{3.2}$$

Here, the matrix $V^{(\mu)} = [v^{(1)}|\dots|v^{(\mu)}]$ contains the orthonormal basis for what we call the *μ-th deflated block Krylov subspace* $K_\mu^{\text{defl}}(A,B)$ created by the algorithm. The number $m_\mu = \mu - h_\mu \leq m$ is the reduced block size due to deflation. Hence, $m - m_\mu$ is the number of deflated vectors up to step μ. In the table in Figure 3.2 we show how m_μ decreases when deflation occurs. The matrix $\widetilde{T}^{(i_\mu,\mu)} \in \mathbb{C}^{i_\mu \times \mu}$ consists of the values $t_{j,\mu}$ from line 10, i.e.

$$\widetilde{T}^{(i_\mu,\mu)} = (t_{j,\mu})_{j \leq (i_\mu), \mu \leq \mu}.$$

Lastly, the matrix $\widetilde{V}_{\text{defl}}^{(\mu)}$ contains the inexactly deflated, non-normalised—and thus with norm smaller than the deflation tolerance tol—vectors in its respective columns. All other columns

Algorithm 2: deflated block Lanczos-type process

```
Data: $A \in \mathbb{C}^{n\times n}$ hpd, linearly independent $B = [b_1|\dots|b_m] \in \mathbb{C}^{n\times m}$, deflation
      tolerance tol
Result: basis $V^{(i_\mu)} = [v^{(1)}|\dots|v^{(\mu)}|\dots|v^{(i_\mu)}]$ for $K^{\text{defl}}_{i_\mu}(A,B)$ and $\widetilde{T}^{(i_\mu,\mu)} \in \mathbb{C}^{i_\mu\times\mu}$, such
        that (3.2) holds
1  begin
2      $\mu = 0, I = \emptyset, (h_1,\dots,h_m) = (1,\dots,1)$
3      orthonormalise $B$, giving $v^{(1)},\dots,v^{(m)}$
4      for $i = m+1, m+2, \dots, k$ and $i \neq \mu$ do
5          repeat
6              $\mu = \mu + 1$
7              $v = Av^{(\mu)}$
8              $i_v = h_\mu$
9              $J = \{i_v,\dots,k-1\} \cup I$
10             compute $t_{j,\mu} = (v^{(j)})^H v$ for all $j \in J$
11             $v = v - \sum_{j\in J} t_{j,\mu} v^{(j)}$
12             if $\|v\| = 0$ then
13                 exact deflation: discard vector $v$
14             else if $\|v\| < tol$ then
15                 inexact deflation: keep $v$ for later orthogonalisation and set
                   $I = I \cup \{\mu\}$
16             else
17                 $t_{i,\mu} = \|v\|$
18                 $v^{(i)} = \frac{v}{t_{i,\mu}}$
19                 $h_i = \mu$
20         until non-deflated new vector $v^{(i)}$ computed in line 18
```

are zero. A possible sparsity pattern of $\widetilde{T}^{(i_\mu,\mu)}$ is displayed in Figure 3.2 which uses the same example setting as the table. Note that the reduced block size m_ν can be found in $\widetilde{T}^{(i_\mu,\mu)}$ as the number of non-zeros above and left of entry (ν,ν) whilst ignoring entries of small magnitude (boxes). Moreover, h_ν is the row-number of the first non-zero entry in column ν.

To derive our block CG method we need to symmetrise the square part of matrix $\widetilde{T}^{(i_\mu,\mu)}$ in (3.2). To this purpose we remove the entries outside the band (displayed by boxes in Figure 3.2) which leaves us with a matrix $T^{(i_\mu,\mu)}$ the square part of which is Hermitian. Using

$$V^{(i_\mu)}\widetilde{T}^{(i_\mu,\mu)} = V^{(i_\mu)}T^{(i_\mu,\mu)} + V^{(i_\mu)}\left(\widetilde{T}^{(i_\mu,\mu)} - T^{(i_\mu,\mu)}\right), \tag{3.3}$$

we obtain from (3.2) the *deflated Lanczos relation*

$$AV^{(\mu)} = V^{(i_\mu)}T^{(i_\mu,\mu)} + V^{(\mu)}_{\text{defl}}, \tag{3.4}$$

where the change from $\widetilde{V}^{(\mu)}_{\text{defl}}$ to $V^{(\mu)}_{\text{defl}}$ accounts for the second term in (3.3).

For ease of notation we will denote the square part of $T^{(i_\mu,\mu)}$ as $T^{(\mu)}$ from now on. Note

i	μ	h_i	h_μ	m_i	m_μ
5	1	1	1	4	0
6	2	2	1	4	1
7	3	3	1	4	2
8	4	-	1	-	3
8	5	5	1	3	4
9	6	6	2	3	4
10	7	7	3	3	4
11	8	8	5	3	3
12	9	-	6	-	3
12	10	10	7	2	3

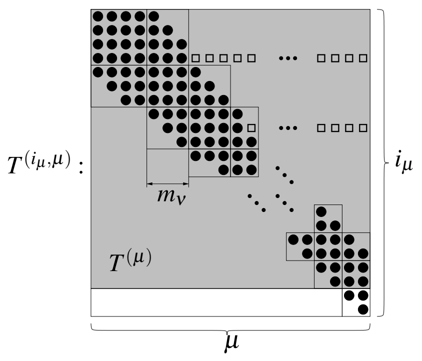

FIG. 3.2. ***Left***: *Example showing the relation of the iteration counters i and μ, the history indices h_i and h_μ and the reduced block sizes m_i and m_μ in Algorithm 2 for $m = 4$ and deflation in steps $\mu = 4$ and $\mu = 9$.* ***Right***: *Example for the sparsity pattern of the square part of matrix $\widetilde{T}^{(i_\mu,\mu)}$ (with the boxes $\square$) and the square part $T^{(\mu)}$ of $T^{(i_\mu,\mu)}$ (without the boxes) in case of deflation in steps $\mu = 4$ and $\mu = 8$ for 4 right-hand sides. The term 'square part' refers to the top $\mu \times \mu$ submatrix that is represented by the shaded background. The boxes $\square$ denote entries of small magnitude arising due to orthogonalisation against inexactly deflated vectors.*

that

$$T^{(\mu)} = (V^{(\mu)})^H A V^{(\mu)}. \tag{3.5}$$

A few remarks on implementation details for Algorithm 2 are in order:

- The initial orthonormalisation of the vectors b_i can be incorporated in the algorithm by starting with a negative index μ and some minor changes to lines 2, 3, 7 and 8.
- An actual implementation should use modified Gram-Schmidt instead of Gram-Schmidt in the orthogonalisation process of lines 10 and 11.
- In line 10 the scalar $t_{j,\mu}$ has to be computed only for $j \geq \mu$. The entries above the diagonal are already known due to symmetry.
- The check for $\|v\| = 0$ in line 12 should be implemented by comparing to some small threshold.

4. The deflated shifted block CG method.

Based on the block Lanczos process presented in the previous section we can now derive the DSBlockCG method. We do so in a similar manner as non-block CG is derived from the Lanczos process [23, Chapter 6.7], for example. We start considering the unshifted block system $AX = B$. Handling additional shifts will be addressed in section 4.4.

A deflated block Krylov subspace method generates the block iterates

$$X^{(\mu)} = [x_1^{(\mu)} | \dots | x_m^{(\mu)}],$$

where $x_j^{(\mu)} \in K_\mu^{\text{defl}}(A,B)$. Building upon (2.2), but using the orthogonal basis of the deflated block Krylov subspace and (3.5), we obtain the iterates $X^{(\mu)}$ as

$$X^{(\mu)} := V^{(\mu)} (T^{(\mu)})^{-1} (V^{(\mu)})^H B. \tag{4.1}$$

Note that we still assume $x_j^{(0)} = 0$ for all j.

4.1. Cholesky decomposition. The Hermitian and positive definite matrix

$$T^{(\mu)} = (V^{(\mu)})^H A V^{(\mu)}$$

that is generated during the block Lanczos process has band structure, and if deflation occurs, the width of the band shrinks as we move down the diagonal. Figure (3.2) illustrates this. The entries at the positions indicated by the boxes are present in $\tilde{T}^{(i_\mu,\mu)}$ when deflation occurs, they are set to 0 in $T^{(\mu)}$. Their values are smaller than the deflation tolerance, so that we assume that they have no substantial influence on the approximation $X^{(\mu)}$ from (4.1). Now let

$$T^{(\mu)} =: L^{(\mu)} D^{(\mu)} (L^{(\mu)})^H$$

be the root-free Cholesky decomposition of $T^{(\mu)}$ where $L^{(\mu)}$ is a lower triangular matrix of unit diagonal with the same sparsity pattern as $T^{(\mu)}$ and $D^{(\mu)}$ is a diagonal matrix.

The matrix $T^{(\mu)}$ gets updated every time a new candidate vector v for extending the Krylov subspace is generated in line 11 of Algorithm 2—even though v occasionally gets removed in case of deflation. The only difference between $T^{(\mu)}$ and $T^{(\mu-1)}$ is the newly appended last column and row. So by writing

$$T^{(\mu)} = \begin{bmatrix} T^{(\mu-1)} & (t^{(\mu)})^H \\ t^{(\mu)} & t^{(\mu,\mu)} \end{bmatrix}, \qquad L^{(\mu)} = \begin{bmatrix} L^{(\mu-1)} & 0 \\ l^{(\mu)} & 1 \end{bmatrix}$$

and

$$D^{(\mu)} = \operatorname{diag}(d^{(1)}, d^{(2)}, \dots, d^{(\mu)})$$

we get updates for $L^{(\mu-1)}$ to $L^{(\mu)}$ and $D^{(\mu-1)}$ to $D^{(\mu)}$ by computing

$$l^{(\mu)} = t^{(\mu)} (L^{(\mu-1)})^{-H} (D^{(\mu-1)})^{-1} \quad \text{and} \quad d^{(\mu)} = t^{(\mu,\mu)} - l^{(\mu)} D^{(\mu-1)} (l^{(\mu)})^H. \tag{4.2}$$

Note that in our notation $l^{(\mu)}$ and $t^{(\mu)}$ are row vectors and due to the sparsity pattern of $T^{(\mu)}$ and $L^{(\mu)}$ only the last at most m entries in each of the vectors are non-zero.

4.2. Updating iterates. Having both the updates in (4.2) for $L^{(\mu)}$ and $D^{(\mu)}$ we can formulate how the iterates $X^{(\mu-1)}$ can be updated. Using (4.1) we get

$$\begin{aligned} X^{(\mu)} &= V^{(\mu)} (L^{(\mu)} D^{(\mu)} (L^{(\mu)})^H)^{-1} (V^{(\mu)})^H B \\ &= \begin{bmatrix} V^{(\mu-1)} & v^{(\mu)} \end{bmatrix} (L^{(\mu)})^{-H} (D^{(\mu)})^{-1} (L^{(\mu)})^{-1} \begin{bmatrix} V^{(\mu-1)} & v^{(\mu)} \end{bmatrix}^H B \\ &= \begin{bmatrix} V^{(\mu-1)} & v^{(\mu)} \end{bmatrix} \begin{bmatrix} L^{(\mu-1)} & 0 \\ l^{(\mu)} & 1 \end{bmatrix}^{-H} (D^{(\mu)})^{-1} \begin{bmatrix} L^{(\mu-1)} & 0 \\ l^{(\mu)} & 1 \end{bmatrix}^{-1} \begin{bmatrix} (V^{(\mu-1)})^H \\ (v^{(\mu)})^H \end{bmatrix} B \\ &= \begin{bmatrix} P^{(\mu-1)} & p^{(\mu)} \end{bmatrix} (D^{(\mu)})^{-1} \begin{bmatrix} U^{(\mu-1)} \\ u^{(\mu)} \end{bmatrix} \\ &= X^{(\mu-1)} + \frac{1}{d^{(\mu)}} p^{(\mu)} u^{(\mu)}. \end{aligned} \tag{4.3}$$

Here we used

$$P^{(\mu)} := \begin{bmatrix} P^{(\mu-1)} & p^{(\mu)} \end{bmatrix} = \begin{bmatrix} V^{(\mu-1)} & v^{(\mu)} \end{bmatrix} \begin{bmatrix} L^{(\mu-1)} & 0 \\ l^{(\mu)} & 1 \end{bmatrix}^{-H}$$

and

$$U^{(\mu)} := \begin{bmatrix} U^{(\mu-1)} \\ u^{(\mu)} \end{bmatrix} = \begin{bmatrix} L^{(\mu-1)} & 0 \\ l^{(\mu)} & 1 \end{bmatrix}^{-1} \begin{bmatrix} (V^{(\mu-1)})^H \\ (v^{(\mu)})^H \end{bmatrix} B.$$

The advantage of introducing the new matrices $P^{(\mu)}$ and $U^{(\mu)}$ is that for the cost of having to store them we have the benefit of being able to do the inversion of $L^{(\mu)}$ implicitly and have simple updates for

$$p^{(\mu)} = v^{(\mu)} - P^{(\mu-1)}(l^{(\mu)})^H \text{ and } u^{(\mu)} = (v^{(\mu)})^H B - l^{(\mu)} U^{(\mu-1)}. \tag{4.4}$$

One must expect that the vectors $v^{(\mu)}$ tend to lose orthogonality rather quickly in numerical computation. However, $(v^{(\mu)})^H B$ in (4.4) should always be taken as zero for $\mu > m$ even though it is not in actual computation since it stabilises the computation of the new iterates. As described before, $L^{(\mu)}$ inherits the banded structure of $T^{(\mu)}$. This implies that the updates in (4.4) need at most the last m columns of $P^{(\mu-1)}$ and, accordingly, the last m rows of $U^{(\mu-1)}$.

4.3. Computing norms of the residuals. In order to be able to stop the iteration we should be able to compute the norms of the residuals $r_j^{(\mu)}, 1 \le j \le m$. Fortunately, although the residuals are not directly available, their norms can be computed at very low additional cost. The iterates $X^{(\mu)}$ are generated in such a way that the residuals $R^{(\mu)}$ are orthogonal to the Lanczos vectors $v^{(j)}$ for $j \le \mu$. Therefore, they may be written as

$$R^{(\mu)} = W^{(\mu)} C^{(\mu)}$$

where $C^{(\mu)} \in \mathbb{C}^{m_\mu \times m}$ and $W^{(\mu)} \in \mathbb{C}^{n \times m_\mu}$ consists of the last m_μ columns of $V^{(i_\mu)}$, i.e. $W^{(\mu)} = [v^{(\mu+1)}|\dots|v^{(i_\mu)}]$. For $k > m$ this can be seen by using relation (3.4) which yields

$$\begin{aligned}
(v^{(k)})^H R^{(\mu)} &= (v^{(k)})^H (B - AX^{(\mu)}) \\
&= \underbrace{(v^{(k)})^H (B}_{=0} - AV^{(\mu)} (T^{(\mu)})^{-1} (V^{(\mu)})^H B) \\
&= (v^{(k)})^H V^{(i_\mu)} T^{(i_\mu,\mu)} (T^{(\mu)})^{-1} (V^{(\mu)})^H B.
\end{aligned}$$

Thus, if we are just interested in the norm of each of the residuals we only have to compute the norm of the columns of $C^{(\mu)} \in \mathbb{C}^{m_\mu \times m}$, $V^{(\mu)}$ having orthonormal columns. Computing $C^{(\mu)}$ is actually quite simple since

$$\begin{aligned}
C^{(\mu)} &= (W^{(\mu)})^H R^{(\mu)} \\
&= (W^{(\mu)})^H (B - AX^{(\mu)}) \\
&= (W^{(\mu)})^H (B - AP^{(\mu)} (D^{(\mu)})^{-1} U^{(\mu)}) \\
&= (W^{(\mu)})^H (B - AV^{(\mu)} (L^{(\mu)})^{-H} (D^{(\mu)})^{-1} U^{(\mu)}) \\
&= \underbrace{(W^{(\mu)})^H (B}_{=0 \text{ for } \mu > m} - V^{(i_\mu)} T^{(i_\mu,\mu)} \underbrace{(L^{(\mu)})^{-H} (D^{(\mu)})^{-1} U^{(\mu)}}_{=: \widetilde{C}^{(\mu)}}) \\
&= -\begin{bmatrix} 0 & I_{m_\mu} \end{bmatrix} T^{(i_\mu,\mu)} \widetilde{C}^{(\mu)}.
\end{aligned} \tag{4.5}$$

The computation of $C^{(\mu)}$ thus essentially only involves one matrix-matrix product of small matrices of dimension m_μ and one inversion of the small triangular matrix $(L^{(\mu)})^H$. So obtaining $C^{(\mu)}$ is computationally cheap.

4.4. Shifts. The block method developed so far can be extended to handle shifted systems and multiple right-hand sides at the same time. The only restriction is on the choice of the starting vectors, because we need to have collinear initial residuals for all shifts of a given right-hand side. This constraint can obviously be fulfilled by choosing 0 as the starting vector for all shifts. For ease of notation we focus on a situation where we have just one additional shift σ and use the notation $A_\sigma := \sigma I + A$. Matrices and vectors belonging to the shifted system will also be noted by the index σ.

Krylov subspaces are shift invariant, i.e. $K_k(A,b) = K_k(A_\sigma, b)$ for all k and σ. Moreover, starting with the initial vector b, the Lanczos process produces exactly the same vectors, whether we take A or A_σ [21]. This property immediately carries over to the block Lanczos process and to the deflated block Lanczos process if we use the same criteria to detect exact or inexact deflation. Moreover, the deflated block Lanczos relation (3.4) turns into

$$A_\sigma V^{(\mu)} = V^{(i_\mu)} T_\sigma^{(i_\mu,\mu)} + V_{\text{defl}}^{(\mu)}$$

with

$$T_\sigma^{(i_\mu,\mu)} = T^{(i_\mu,\mu)} + \sigma \begin{bmatrix} & I_\mu & \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix},$$

where I_μ is the identity matrix of dimension μ. In particular, we have

$$(V^{(\mu)})^H (\sigma_j I_n + A) V^{(\mu)} = \sigma_j I_\mu + T^{(\mu)} =: T_\sigma^{(\mu)},$$

so that we can obtain the block CG iterates for the shifted system in exactly the same manner as described in section 4.2, with $T^{(\mu)}$ replaced by $T_\sigma^{(\mu)}$. This saves us the cost of creating $V^{(\mu)}$ and $T^{(\mu)}$ for each shifted system individually which is expected to be the most expensive part since it involves multiplications with the matrix A_σ. However, for every shift, the Cholesky decomposition $L_\sigma^{(\mu)}$ and $D_\sigma^{(\mu)}$ and the matrices $P_\sigma^{(\mu)}$ and $U_\sigma^{(\mu)}$ needed for generating the iterates, have to be computed and stored. We will give a detailed analysis of the memory requirements in the next section.

5. Algorithm. Summarising the previous sections and using Algorithm 2 we now formulate the DSBlockCG algorithm as Algorithm 3. The loop in line 7 means that if for a fixed shift σ_j all the iterates $x_{i,j}$ with $1 \le i \le m$ are converged to the requested tolerance, the matrices $L_\sigma^{(\mu)}, D_\sigma^{(\mu)}, P_\sigma^{(\mu)}, U_\sigma^{(\mu)}$ and $C_\sigma^{(\mu)}$ do not have to be updated anymore. Additionally then, only those columns of $C_\sigma^{(\mu)}$ have to be computed that belong to non-converged systems for shift σ.

The memory footprint of the DSBlockCG algorithm in a memory efficient implementation consists of $2m+1$ vectors of length n for storing the matrix V, and a temporary vector v. For every shift the search direction matrix $P_\sigma^{(\mu)}$ has to be stored which amounts to additional $s \cdot m$ vectors of length n. The matrix $T^{(\mu)}$ can be stored using $(2m+1) \times (m+1)$ entries. Since for every shifted system the shift only has to be applied to the diagonal of $T^{(\mu)}$, there is no need for storing more than one copy of this matrix. However, for every shift we need to store the matrices $U_\sigma^{(\mu)}$, $L_\sigma^{(\mu)}$, $C_\sigma^{(\mu)}$ and a temporary matrix, which all have a size of at most $m \times m$.

Overall we end up with a memory footprint of $(sm + 2m + 1)n + 6m^2 + \mathcal{O}(m)$ floating point numbers in addition to the storage for the matrix A, the right-hand sides B and the result vectors X.

Algorithm 3: DSBlockCG - deflated shifted BlockCG

```
**Data**: $A \in \mathbb{C}^{n\times n}$ hpd, $B = [b_1|\dots|b_m] \in \mathbb{C}^{n\times m}$, shifts $\sigma_1,\dots,\sigma_s$, deflation tolerance tol
        for Algorithm 2, target relative residual $\delta$
**Result**: solutions $x_{i,j}$ to the systems (1.1) with $\left\|b_i - (\sigma_j I + A)x_{i,j}\right\| < \delta \left\|b_i\right\|$
1  begin
2      set $\mu = -m, I = \emptyset$ for Algorithm 2
3      repeat
4          repeat
5              perform the lines 6 to 19 of Algorithm 2
6              if $\mu > 0$ then
7                  forall the unconverged systems $\sigma$ do
8                      compute $l_\sigma^{(\mu)}$ and $d_\sigma^{(\mu)}$ in the Cholesky decomposition, see (4.2)
9                      update the matrices $P_\sigma^{(\mu)}$ and $U_\sigma^{(\mu)}$ using (4.4)
10                     update the non-converged columns of the iterate $X_\sigma^{(\mu)}$ as in (4.3)
11                     compute $C_\sigma^{(\mu)}$ given in (4.5)
12                     check convergence by calculating the norm of the columns of $C_\sigma^{(\mu)}$
13         until non-deflated new vector $v_i$ created
14     until all systems converged to given target relative residual $\delta$
```

For the computational complexity we assume that $n \gg m$ and $n \gg s$ so that $\mathcal{O}(m^3)$ and $\mathcal{O}(ms)$ are negligible compared to $\mathcal{O}(n)$. For simplicity, we only analyse the non-deflation case. The computational cost for one step in the DSBlockCG algorithm consists of

- a multiplication with the matrix A,
- the orthogonalisation against $2m$ previous vectors,
- the normalisation of the new vector that extends the Krylov subspace,
- the update of the search direction matrices $P_\sigma^{(\mu)}$ for every shift σ and
- the update of the iterates $X_\sigma^{(\mu)}$ for every shift σ.

The cost of the multiplication with A is $c_A\mathcal{O}(n)$ where c_A represents a constant that depends on the number of non-zeros of A. The orthogonalisation needs $2m$ vector additions but just m inner products by exploiting the symmetry of $T^{(\mu)}$. Normalising the resulting vector after orthogonalisation accounts for one operation with cost $\mathcal{O}(n)$. Finally, the updates of the search directions and the iterates account for sm vector operations each. Overall we end up with a computational complexity of $(c_A + 3m + 1 + 2sm)\mathcal{O}(n)$ per step.

Concluding this section we discuss how DSBlockCG performs depending on the number of right-hand sides. Solving for only one right-hand side DSBlockCG has a computational complexity of $(c_A + 4 + 2s)\mathcal{O}(n)$ per step. Now let k be the number of steps that DSBlockCG needs to solve (1.1) and let l be the combined number of steps that DSBlockCG needs if we apply the algorithm to each side separately. Then, solving for m right-hand sides at once is faster if

$$\frac{k}{l} < \frac{(c_A + 4 + 2s)}{(c_A + 3m + 1 + 2sm)}.$$

This means that using DSBlockCG for m right-hand sides is advisable if the matrix-vector multiplications dominate the computation (i.e. c_A is large compared to m and s) or the number of iterations k is significantly lower than l (i.e. the block Krylov subspace contains valuable additional information for each right-hand side compared to the non-block subspaces).

6. Numerical results. In this section we present and discuss results for Algorithm 3 applied to different test problems. In order to achieve fair time measurements all the algorithms we are comparing were implemented in C++ using the Eigen library [13] for sparse and dense BLAS operations. We stored the matrices in the compressed column storage (CCS) sparse format. Our programs were not parallelised and ran on an Intel Xeon X5680 (3.33 GHz, 96 GiB RAM).

We compare our algorithm to three other algorithms—the first is non-preconditioned conjugate gradients which we will simply call CG from now on. In CG we solve each of the systems (1.1)

$$(\sigma_j I + A)x_{i,j} = b_i$$

separately. The second algorithm is the shifted conjugate gradients algorithm from [12] referred to as shifted CG. It has to be applied m times with one single right-hand side b_i at a time: For every single right-hand side all s shifted systems are solved at the same time. The third and last competing algorithm is BCGrQ from [8]. This block CG algorithm solves all the systems belonging to a single shift but several right hand sides in one go.

6.1. Lattice quantum chromodynamics (QCD). In the Rational Hybrid Monte Carlo algorithm (RHMC) used in the context of lattice QCD the j-th root $(M^H M)^{1/j}$ of a matrix $M^H M$ has to be applied to a couple of random vectors [16]. The matrix M represents a nearest neighbour coupling on a four dimensional lattice having 12 variables at each lattice point. Thus, the dimension of M is $n = 12n_1n_2n_3n_4$ and typical values for n_i start from 16 and range up to 128 resulting in n being of magnitude $10^6 - 10^9$. Because of the size of the matrix M, direct and iterative methods for explicitly computing the full matrix $(M^H M)^{1/j}$ are not feasible [15]. Approximating the j-th root via a partial fraction expansion of a rational approximation gives rise to shifted methods. The situation is favourable for Algorithm 3, since $M^H M$ is positive definite and all shifts of the partial fraction expansion are positive, too.

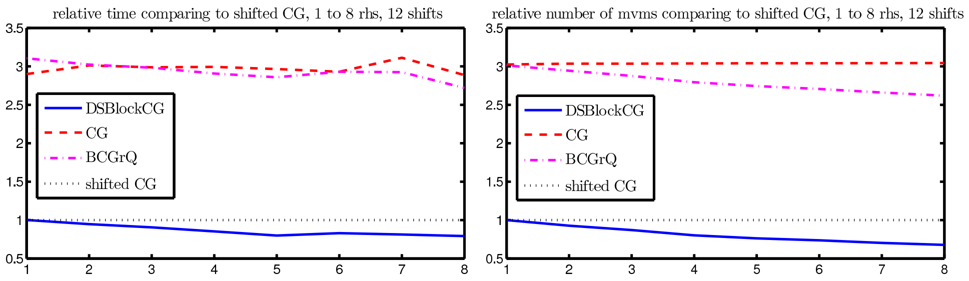

FIG. 6.1. *The left plot shows the relative time (vertical axis) of all compared algorithms on a* 16^4 *lattice for various numbers of right-hand sides (horizontal axis) where the times are normalised w.r.t. shifted CG. The right plot displays the relative number of mvms for the same run with the same normalisation.*

Figure 6.1 shows a comparison of all four algorithms for a 16^4 lattice. The matrix used corresponds to a typical configuration in lattice QCD with parameters taken from run A_3 in [6, 7]. We used several numbers of random right-hand sides and a fixed number of 12 shifts. The shifts were taken for the partial fraction expansion of a Padé approximation to $f(x) = x^{1/4}$. They range between 0.0053 and 964.0944. This Padé approximation differs from $x^{1/4}$ on the interval $[\lambda_{\min}(M^H M), \lambda_{\max}(M^H M)]$ by less than $3 \cdot 10^{-5}$. The target relative residual was chosen as 10^{-12}, s.t. the error of the solution is about in the order of the accuracy of the approximation. For each run with k right-hand sides we used the same k random right-hand sides for all four methods. In our numerical experiments shown in Figure 6.1 we computed the matrix $M^H M$ explicitly instead of multiplying by M and M^H separately for computing $M^H Mx$. The rationale for this is our discussion of the computational complexity in Section 5. Computing $M^H M$ explicitly increased the number of non-zeros per row from 49 in M and M^H to about 310 in $M^H M$. In Section 6.2 we give more examples for tests with matrices with varying numbers of non-zeros per row.

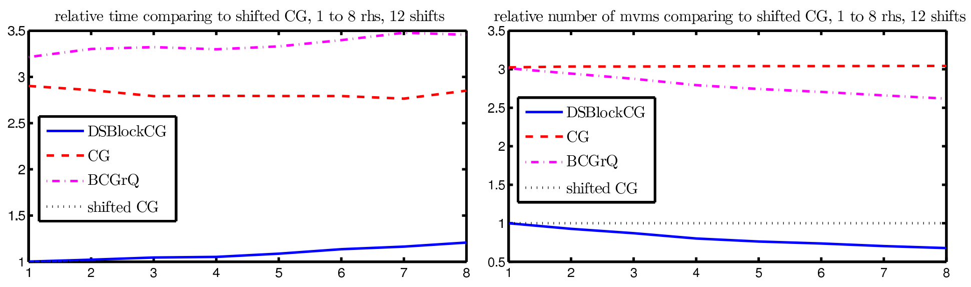

FIG. 6.2. *The left plot shows the relative time (vertical axis) of all compared algorithms on a* 16^4 *lattice for various numbers of right-hand sides (horizontal axis) where the times are normalised w.r.t. shifted CG. The right plot displays the relative number of mvms for the same run with the same normalisation.*

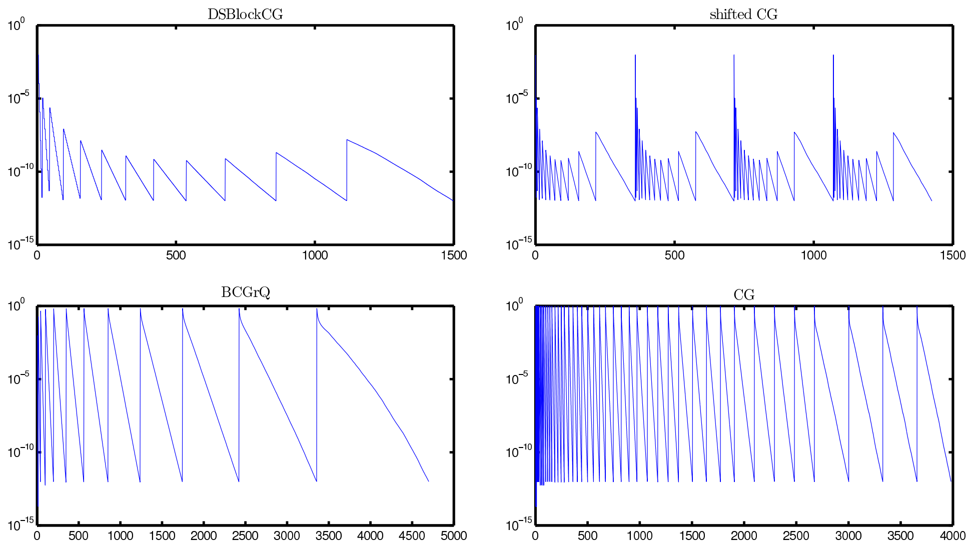

FIG. 6.3. *Convergence plots for all the compared algorithms on a* 16^4 *lattice with 4 right-hand sides and 12 shifts. Horizontal axis: time in seconds. Vertical axis: relative norm of residual. In case of the methods that can not solve shifted systems at the same time, the saw tooth shape of the plots is caused by their subsequent runs. DSBlockCG and shifted CG show a saw tooth shape too, since we always plot the residual for the system belonging to the shift that converges next, i.e. the worst conditioned of the remaining systems.*

Figure 6.2 shows the same test setting as in Figure 6.1 but without computing $M^H M$ explicitly. In this case the matrix-vector multiplications do not dominate the computations anymore which results in shifted CG outperforming our algorithm. Figures 6.1 and 6.2 clearly show that in any case CG and BCGrQ are not competitive. For shifted CG vs. DSBlockCG we will see later that the situation changes in favour of DSBlockCG not only when the sparsity of the matrix M decreases but also when the matrix M is less well conditioned.

In Figure 6.3 we display the convergence behaviour for the run with four right-hand sides in more detail. As seen in the top left plot for DSBlockCG, better conditioned systems converge much faster. Thus, it is important to stop updating them as soon as they reach the target residual to speed up the computation of the remaining systems as we did in our implementation of Algorithm 3. The top right shifted CG plot shows that the different random right-hand side vectors show almost the same convergence behaviour. Table 6.1 gives the plain numbers for the same test run. It shows that CG needs the highest number of matrix-vector multiplications. However, comparing the times reveals that BCGrQ is not faster than CG and that both, shifted CG and DSBlockCG are much faster than CG. DSBlockCG is a bit slower than shifted CG despite the fact hat it needs a significantly lower number of matrix-vector multiplications. This reflects the fact that additional work to be invested to handle multiple right-hand sides is not negligible. As a consequence of the results we now constrain the remaining tests with shifted systems to comparisons of DSBlockCG and shifted CG.

TABLE 6.1
Number of mvms, time, relative number of mvms and relative time as compared to DSBlockCG of the test runs of Figure 6.3.

	CG	BCGrQ	shifted CG	DSBlockCG
mvms	13246	12180	4360	3496
time in seconds	3981.85	4700.07	1424.52	1498.50
relative mvms	3.79	3.48	1.25	1
relative time	2.66	3.14	0.95	1

In Figure 6.4 we used the same matrix as in the previous example. This time, however, we enforced one right-hand side to be linearly dependent from the remaining randomly chosen right-hand sides. This results in immediate deflation and, as a consequence, in a huge speed up in solving all systems. Admittedly, this behaviour is rarely seen in real computations, but it shows that deflation is working as expected. Furthermore, if deflation occurs early in the iteration, DSBlockCG can be significantly faster than using shifted CG.

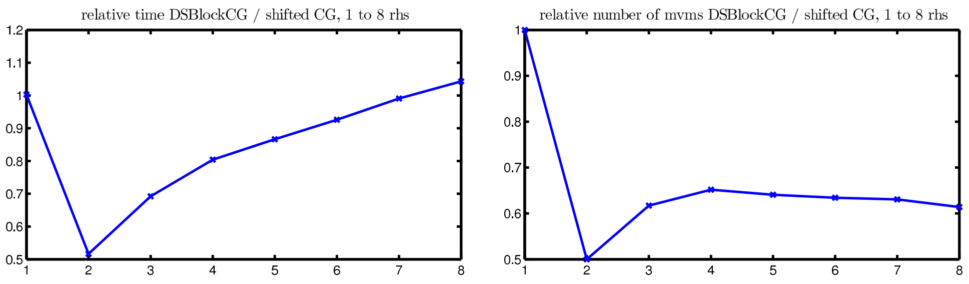

FIG. 6.4. *This figure shows the effect if one right-hand side can be deflated early in the iteration. In the left picture the relative time (vertical axis) of algorithm DSBlockCG over shifted CG on a 16^4 lattice for a different number of right-hand sides (horizontal axis) can be seen. The right plot displays the relative number of mvms for the same run.*

In lattice QCD, bigger lattices allow to adjust parameters close to physically relevant values, typically resulting in less well conditioned systems. Since using larger lattices would require a parallelization of our code which is out of scope for this paper, we simulated less well conditioned systems on the 16^4 lattice by increasing the "hopping parameter" κ in $M =$

$I - \kappa D$ where D represents the coupling to the nearest neighbours. This is consistent to what is done on large lattices, where κ is also increased. Note that for relevant values of κ all eigenvalues of M lie in the right half plane. Whereas the smallest real part of any eigenvalue in M was $7.7 \cdot 10^{-3}$ in our previous experiments, we now set κ such that it becomes 10^{-6}. Again, we used a Padé approximation to $f(x) = x^{1/4}$ with 12 shifts. They range between $5 \cdot 10^{-4}$ and 10^{3}. The results for these settings are shown in Figure 6.5. The number of

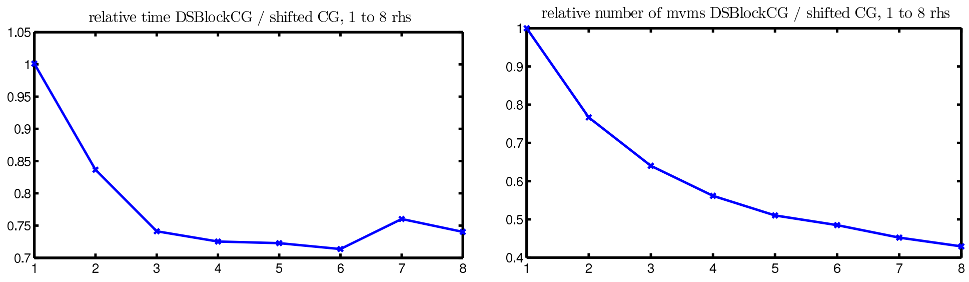

FIG. 6.5. *In this plot we shifted the systems of the previous plots to increase its condition number. The left plot shows the relative time (vertical axis) of algorithm DSBlockCG over shifted CG on a 16^4 lattice for a different number of right-hand sides (horizontal axis). The right plot displays the relative number of mvms for the same run.*

mvms decreases considerably as the number of right-hand sides is increased and drops to less than half the number of iterations needed by shifted CG. Additionally, there is a gain in computational time. For 3 and 6 right-hand sides, solving the system with DSBlockCG needs less than 75% of the time required for shifted CG. For increasingly more right-hand sides the additional costs of handling them in DSBlockCG begins to dominate the computation time.

6.2. Unshifted systems with multiple right-hand sides. DSBlockCG can also be used as a non-shifted multiple right-hand sides method. Our goal is to show that DSBlockCG can compete with block methods like BCGrQ and outperforms CG when applied to non-shifted block systems. For this we used four matrices from the Florida sparse matrix collection [5]. Table 6.2 displays information on the four matrices used in our tests. We chose these ma-

TABLE 6.2
Test matrices from the Florida sparse matrix collection [5]. Including the dimension of the operator, the number of non-zeros and the mean non-zeros per column.

name	dimension	non-zeros	non-zeros per column
`msc04515`	4515	97707	21.6
`Pres_Poisson`	14822	715804	48.3
`smt`	25710	3749582	145.8
`nd12k`	36000	14220946	395.0

trices as examples, because they are hpd and have a relatively high number of non-zeros per column. For more detailed information on the matrices see [5]. For all matrices we computed solutions to random right-hand sides and compared our algorithm to CG and BCGrQ. In our tests we experienced that if inexact deflation happens early in the iteration the stability of the method suffers especially if a high precision as the target residual norm is requested. This behaviour was already described and discussed in [3]. Thus, we have to balance the deflation tolerance in Algorithm 3 and the final required accuracy. We chose a deflation tolerance of 10^{-10} and stopped the iteration as soon as a relative residual norm of 10^{-6} was reached.

Figure 6.6 displays the results for these runs depending on the number of right-hand sides. While the number of matrix-vector multiplications is almost the same for BCGrQ and DS-BlockCG in all examples, there is a noticeable difference in the total execution time for the more sparse matrices like `msc04515` and `Pres_Poisson`: For small numbers of right-hand sides DSBlockCG is significantly faster than BCGrQ. In the tests with the matrix `msc04515` there is a turning point at 10 right-hand sides after which BCGrQ starts to solve the systems faster than DSBlockCG. When solving systems with the matrix `Pres_Poisson` DSBlockCG is faster than BCGrQ only for up to 6 right-hand sides. We attribute this to caching effects that have a higher impact on the computational time of DSBlockCG. For the other two matrices DSBlockCG and BCGrQ show about the same speed and are from 20 up to 80 per cent faster than CG.

Our results show that the achievable gain in computational time depends on the sparsity of the matrix. The reason for this is that our block method saves matrix-vector multiplications while spending more time on vector operations. So for less sparse matrices the savings in matrix-vector multiplications outweigh the additional operations even more than for sparser matrices.

For the QCD matrices we have seen that our method even gains more if deflation occurs. We artificially chose one right-hand side to be linearly dependent from the others after a few iteration steps. In Figure 6.7 as before, we chose 10^{-6} as the target relative residual norm. As expected our algorithm can benefit from deflating the linearly dependent vector. The reduction in the number of matrix-vector multiplications in all tests directly translates into improved timings. Especially for a low number of right-hand sides DSBlockCG outperforms BCGrQ. For matrix `msc04515` the turning point now moves from 10 to 15 right-hand sides from which on BCGrQ is faster.

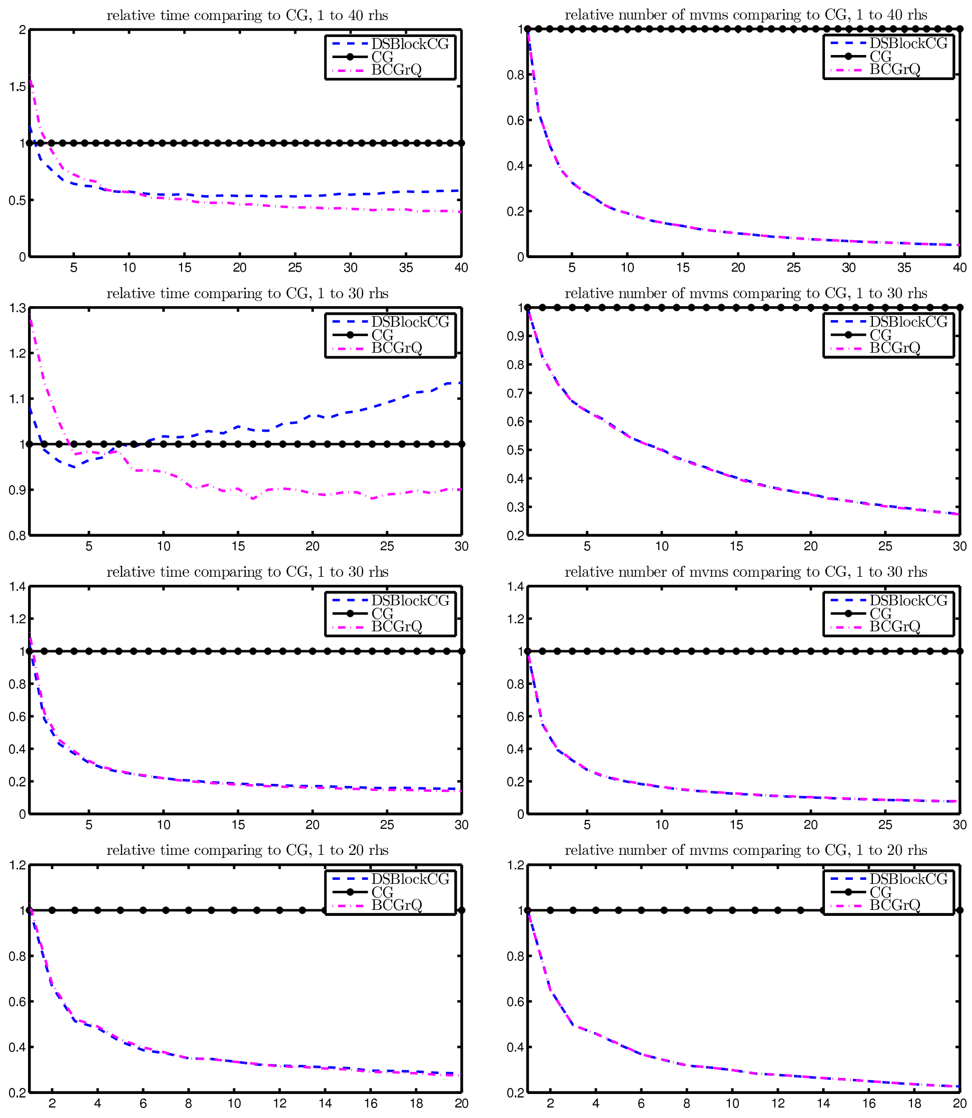

FIG. 6.6. *Matrices from top to bottom:* `msc0415`, `Pres_Poisson`, `smt`, `nd12k`, *left: relative time compared to CG, right: relative number of mvms compared to CG*

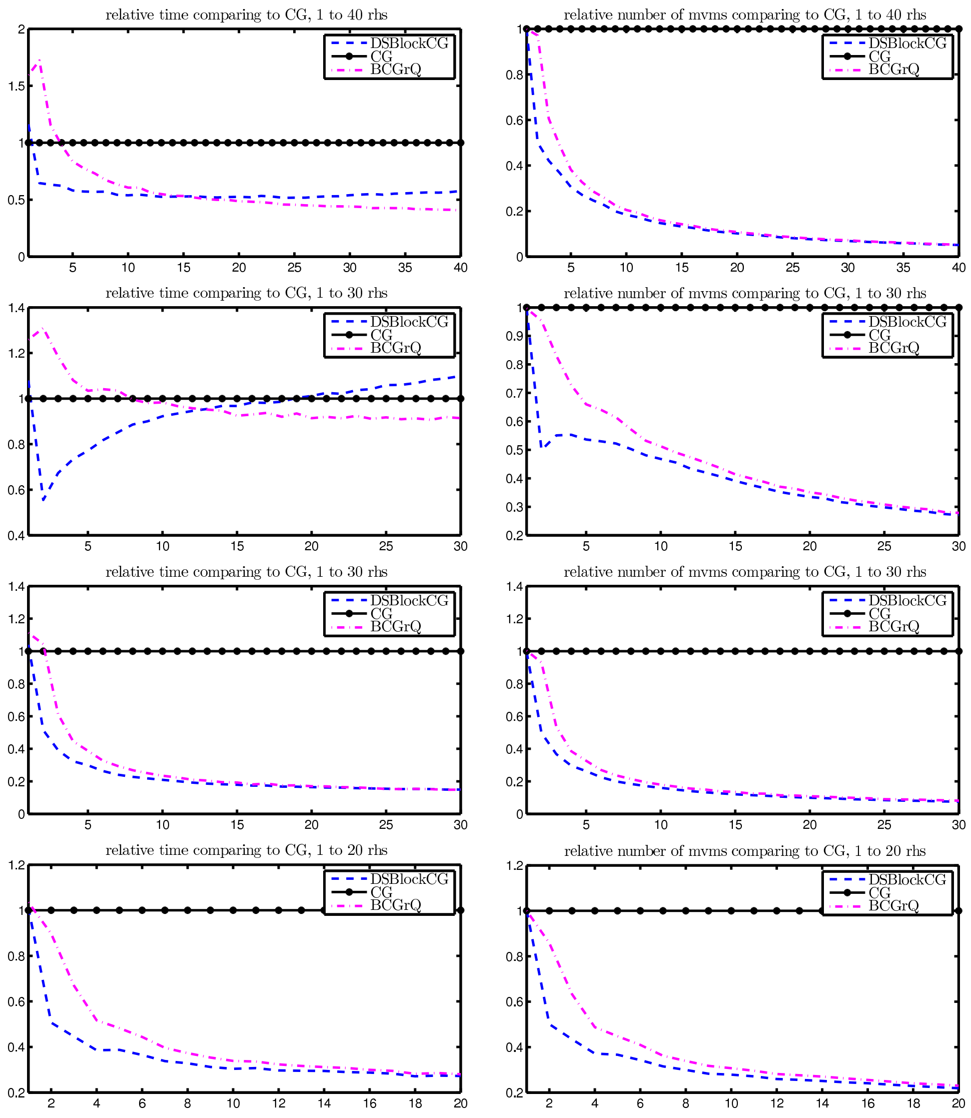

FIG. 6.7. *Matrices from top to bottom:* `msc0415`, `Pres_Poisson`, `smt`, `nd12k`*, left: relative time compared to CG, right: relative number of mvms compared to CG*

7. Conclusions. We proposed a new iterative method for the solution of systems of linear equations of the form $(\sigma_j I + A)x_{i,j} = b_i$ based on a block Lanczos-type process. The method is stable since it accounts for deflation in the block Lanczos process, and it is efficient since it avoids matrix-vector multiplications on deflated vectors. Numerical experiments show that this method converges faster than just applying CG to every system or applying block CG methods to systems belonging to a single shift. For reasonable numbers of right-hand sides and shifts our new method proved to be faster than shifted CG applied to systems belonging to a single right-hand side.

We wish to thank Kirk Soodhalter for his helpful comments on a draft of this paper.

REFERENCES

[1] A. Abdel-Rehim, R. B. Morgan, and W. Wilcox. Seed methods for linear equations in lattice QCD problems with multiple right-hand sides. *PoS*, LAT2008:038, 2008.

[2] J. I. Aliaga, D. L. Boley, R. Freund, and V. Hernández. A Lanczos-type method for multiple starting vectors. *Math. Comp.*, 69(232):1577–1601, 2000.

[3] G. Barbella, F. Perotti, and V. Simoncini. Block Krylov subspace methods for the computation of structural response to turbulent wind. *Computer Methods in Applied Mechanics and Engineering*, 200(2324):2067 – 2082, 2011.

[4] D. Darnell, R. B. Morgan, and W. Wilcox. Deflated GMRES for systems with multiple shifts and multiple right-hand sides. *Linear Algebra and its Applications*, 429(10):2415 – 2434, 2008. Special Issue in honor of Richard S. Varga.

[5] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, Dec. 2011.

[6] L. D. Debbio, L. Giusti, M. Lüscher, R. Petronzio, and N. Tantalo. QCD with light Wilson quarks on fine lattices (i): First experiences and physics results. *JHEP*, 0702:056,2007, 2007.

[7] L. D. Debbio, L. Giusti, M. Lüscher, R. Petronzio, and N. Tantalo. QCD with light Wilson quarks on fine lattices (ii): DD-HMC simulations and data analysis. *JHEP*, 0702:082,2007, 2007.

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

[9] J. Erhel and F. Guyomarc'H. An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1279–1299, 2000.

[10] P. Fiebach, A. Frommer, and R. Freund. Variants of the Block-QMR method and applications in quantum chromodynamics. In *15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*, volume 3, pages 491–496, 1997.

[11] R. Freund and M. Malhotra. A block QMR algorithm for non-hermitian linear systems with multiple right-hand sides. *Linear Algebra Appl.*, 254:119–157, 1997.

[12] A. Frommer and P. Maass. Fast CG-based methods for Tikhonov-Phillips regularization. *SIAM J. Sci. Comput.*, 20(5):1831–1850 (electronic), 1999.

[13] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[14] M. H. Gutknecht. Block Krylov space methods for linear systems with multiple right-hand sides: an introduction. In A. Siddiqi, I. Duff, and O. Christensen, editors, *Modern Mathematical Models, Methods and Algorithms for Real World Systems*, pages 420–447. Anamaya Publishers, New Delhi, India, 2007.

[15] N. J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[16] A. D. Kennedy. Algorithms for dynamical fermions. Technical report, School of Physics, University of Edinburgh, 2006. arXiv:hep-lat/0607038v1.

[17] R. B. Morgan. Restarted block-GMRES with deflation of eigenvalues. *Applied Numerical Mathematics*, 54(2):222 – 236, 2005. 6th IMACS International Symposium on Iterative Methods in Scientific Computing.

[18] A. A. Nikishin and A. Y. Yeremin. Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, i: General iterative scheme. *SIAM Journal on Matrix Analysis and Applications*, 16(4):1135–1153, 1995.

[19] A. A. Nikishin and A. Y. Yeremin. An automatic procedure for updating the block size in the block conjugate gradient method for solving linear systems. *Journal of Mathematical Sciences*, 114:1844–1853, 2003. 10.1023/A:1022462721147.

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

[21] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numer. Linear Algebra Appl.*, 2(2):115–133, 1995.

[22] Y. Saad. On the Lanczos method for solving symmetric linear systems with several right-hand sides. *Mathematics of Computation*, 48(178):pp. 651–662, 1987.

[23] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[24] K. Soodhalter. An alternative block MINRES algorithm: mathematics and implementation. *Preparation*, 20xx.

[25] A. Stathopoulos and K. Orginos. Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics. *SIAM J. Sci. Comput.*, 32(1):439–462, Feb. 2010.

[26] H. Tadano, T. Sakurai, and Y. Kuramashi. Block BiCGGR: A new block Krylov subspace method for computing high accuracy solutions. *JSIAM Lett*, 1:44, 2009.