Gpmr: An iterative method for unsymmetric partitioned linear systems

Alexis Montoison* and Dominique Orban†

Abstract. We introduce an iterative method named Gpmr for solving $2 \times 2$ block unsymmetric linear systems. Gpmr is based on a new process that reduces simultaneously two rectangular matrices to upper Hessenberg form and that is closely related to the block-Arnoldi process. Gpmr is tantamount to Block-Gmres with two right-hand sides in which the two approximate solutions are summed at each iteration, but requires less storage and work per iteration. We compare the performance of Gpmr with Gmres and Block-Gmres on linear systems from the SuiteSparse Matrix Collection. In our experiments, Gpmr terminates significantly earlier than Gmres on a residual-based stopping condition with an improvement ranging from around 10% up to 50% in terms of number of iterations. We also illustrate by experiment that Gpmr appears more resilient to loss of orthogonality than Block-Gmres.

Key words. Sparse linear systems, iterative methods, orthogonal Hessenberg reduction, block-Arnoldi process, Krylov subspaces, generalized saddle-point systems, unsymmetric partitioned matrices, regularization, preconditioners

AMS subject classifications. 15A06, 65F10, 65F08, 65F22, 65F25, 65F35, 65F50

1. Introduction. Consider the partitioned linear system

$$
\begin{bmatrix}
M & A_* \\
B_* & N
\end{bmatrix}
\begin{bmatrix}
x_* \\
y_*
\end{bmatrix}
= \begin{bmatrix}
b_* \\
c_*
\end{bmatrix},
$$

where $M \in \mathbb{R}^{m \times m}$, $N \in \mathbb{R}^{n \times n}$, $A_* \in \mathbb{R}^{m \times n}$ and $B_* \in \mathbb{R}^{n \times m}$. We assume that $A_*$ and $B_*$ are nonzero, and that $b_* \in \mathbb{R}^m$ and $c_* \in \mathbb{R}^n$ are both nonzero. System (1.1) occurs, among others, in the discretization of systems of partial-differential equations, including the Navier-Stokes equations by way of the finite elements method [8]. A prime example is domain decomposition with no overlap, also known as iterative substructuring [6], that consists in splitting a domain into $k$ non-overlapping subregions, and that leads to structured matrices with arrowhead form [10]. Let $I$ be the set of all indices of the discretization points that belong to the interior of the subdomains and $\Gamma$ the set of those corresponding to the interfaces between the subdomains. Grouping the unknowns corresponding to $I$ by subdomain in $u_I$ and those corresponding to $\Gamma$ in $u_\Gamma$, we obtain the arrowhead partitioning of the stiffness system

$$
\begin{bmatrix}
A_{II} & A_{I\Gamma} \\
A_{\Gamma I} & A_{\Gamma\Gamma}
\end{bmatrix}
\begin{bmatrix}
u_I \\
u_\Gamma
\end{bmatrix}
= \begin{bmatrix}
f_I \\
f_\Gamma
\end{bmatrix}
\iff
\begin{bmatrix}
A_{11} & \cdots & A_{1\Gamma} \\
\vdots & \ddots & \vdots \\
A_{kK} & A_{k\Gamma} & A_{\Gamma\Gamma}
\end{bmatrix}
\begin{bmatrix}
u_1 \\
\vdots \\
u_k \\
u_\Gamma
\end{bmatrix}
= \begin{bmatrix}
f_1 \\
\vdots \\
f_k \\
f_\Gamma
\end{bmatrix},
$$

where $u = (u_I, u_\Gamma)$ is the vector of nodal displacements and $f$ the vector of nodal forces. For a tour of applications leading to (1.1), we refer the reader to [2]. We

*GERAD and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, QC, Canada. E-mail: alexis.montoison@polymtl.ca. Research supported by a FRQNT grant and an excellence scholarship of the IVADO institute.

†GERAD and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, QC, Canada. E-mail: dominique.orban@gerad.ca. Research partially supported by an NSERC Discovery Grant.
assume that there exist nonsingular \( P_\ell \) and \( P_r \) with inexpensive inverses such that

\[
K := P_\ell^{-1} \begin{bmatrix} M & A_* \\ B_* & N \end{bmatrix} P_r^{-1} = \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix}, \quad \lambda, \mu \in \mathbb{R},
\]

so that the equivalent preconditioned system

\[
\begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}, \quad \begin{bmatrix} x_* \\ y_* \end{bmatrix} = P_r^{-1} \begin{bmatrix} x \\ y \end{bmatrix}, \quad \begin{bmatrix} b_* \\ c_* \end{bmatrix} = P_\ell^{-1} \begin{bmatrix} b \\ c \end{bmatrix}
\]

can be solved instead of (1.1). Note that \( \lambda \) and/or \( \mu \) may vanish. For example, the ideal preconditioners of Murphy et al. [19] and Ipsen [14] lead to (1.3). Although ideal preconditioners are typically impractical because they require the solution of systems with the Schur complement \( S = N - B_* M^{-1} A_* \), viable preconditioners such that \( P_\ell P_r = \text{blkdiag}(M, N) \) can be employed when \( M \) and \( N \) are both nonsingular.

Given an unstructured matrix \( C \), a practical approach to recovering the matrix of (1.1) is to permute its rows and columns with orderings determined by graph partitioning tools such as METIS [15]. This reordering also provides a uniform partitioning to compute a parallel block-Jacobi preconditioner for (1.3).

When \( \lambda \neq 0 \), (1.4) can be reduced to the Schur complement system

\[
(\mu I - \lambda^{-1} BA)y = c - \lambda^{-1} Bb, \quad x = \lambda^{-1} (b - Ay).
\]

Such eliminated system is attractive because of its smaller size, but may have worse conditioning than (1.4), e.g., when \( B = A^T \), \( M = M^T \succ 0 \) and \( N = N^T \preceq 0 \), though not always, e.g., when (1.1) is symmetric and positive definite. In this paper, we focus on applying an iterative method to (1.4) directly while exploiting its block structure.

Contributions. Our main contributions are (i) a new orthogonal Hessenberg reduction process, (ii) an iterative method based on said process named Gpmr (General Partitioned Minimal Residual) specialized for (1.4), and (iii) an efficient software implementation to solve (1.4) in arbitrary floating-point arithmetic on CPU and GPU.

Related research. Numerous Krylov methods have been developed for solving general unsymmetric linear systems, including BiLQ [16], GMRES [23], or QMR [12]. Few are tailored specifically to the block structure of (1.1).

Specialized iterative methods have been developed for special cases of (1.1). Estrin and Greif [9] developed SPMR; a family of methods for (1.1) that exploit its block structure when \( N = 0 \) and \( b \) or \( c \) is zero. Buttari et al. [4] developed USYMLQR, an interlacing of the methods USYMLQ and USYMQR of Saunders et al. [25], applicable when \( A = B^T \), \( M = M^T \succ 0 \) and \( N = 0 \). Greif and Wathen [13] formulate conditions under which CG may be used in the case where \( M \geq 0 \) is maximally rank deficient and \( N = N^T \preceq 0 \). When \( N = N^T \prec 0 \) also holds, Orban and Arioli [20] propose a family of methods inspired from regularized least norm and least squares that apply after a translation so that either \( b \) or \( c \) is zero, and Montoison and Orban [17] develop TrICG and TrIMR, two methods related to BLOCK-CG and BLOCK-Minres. When \( A = B^T \), and \( M \) and \( N \) are either zero or symmetric definite matrices, our orthogonal Hessenberg reduction process coincides with that of Saunders et al. [25] and Gpmr coincides with TrIMR in exact arithmetic.

Notation. All vectors are columns vectors. Vectors and matrices are denoted by lowercase Latin and capital Latin letters, respectively. The only exceptions are 2×2 blocks, which are represented by capital Greek letters, and the matrices denoted
For a matrix $M$, $\|M\|_F$ denotes the Frobenius norm of $M$. The shorthand $y \mapsto M y$ represents an operator that returns the solution of $M x = y$. $e_i$ is the $i$-th column of an identity matrix of size dictated by the context. $I_k$ represents the $k \times k$ identity operator. We omit the subscript $k$ when it is clear from the context. We let

\[
K_0 := \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}, \quad \text{blkdiag}(\lambda I, \mu I) = \begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix}, \quad d := \begin{bmatrix} b \\ c \end{bmatrix}, \quad D := \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix}.
\]

For a matrix $C$ and a vector $t$, $K_k(C, t)$ is the Krylov subspace $\text{Span} \{ t, Ct, \ldots, C^{k-1}t \}$. For a matrix $T$ with as many rows as $C$ has columns, $K_k(C, T)$ is the block-Krylov subspace $\text{Span} \{ T, CT, \ldots, C^{k-1}T \}$. We abusively write $(b, c)$ and $l = (l_1, \ldots, l_n)$ to represent the column vectors $[b^T \ c^T]^T$ and $l = [l_1 \ \cdots \ l_n]^T$, respectively.

### 2. A Hessenberg reduction process

In this section, we state a new Hessenberg reduction process for general $A$ and $B$, its relationship with the block-Arnoldi process, and the modifications necessary for regularizion.

**Theorem 2.1.** Let $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times m}$, and $p := \min\{m, n\}$. There exist $V \in \mathbb{R}^{m \times p}$ and $U \in \mathbb{R}^{n \times p}$ with orthonormal columns, and upper Hessenberg $H \in \mathbb{R}^{p \times p}$ and $F \in \mathbb{R}^{n \times p}$ with nonnegative subdiagonal coefficients such that

\[
\begin{align*}
V^T A U &= H, \\
U^T B V &= F.
\end{align*}
\]

**Proof.** Choose arbitrary unit $u_1 \in \mathbb{R}^n$ and $v_1 \in \mathbb{R}^m$. For $k = 1, \ldots, p - 1$, define

\[
\begin{align*}
\beta_{k+1} v_{k+1} &= A u_k - \sum_{i=1}^{k} (v_i^T A u_k) v_i, \\
\gamma_{k+1} u_{k+1} &= B v_k - \sum_{i=1}^{k} (u_i^T B v_k) u_i,
\end{align*}
\]

with positive $\beta_{k+1}$ and $\gamma_{k+1}$ such that $v_{k+1}$ and $u_{k+1}$ are unit vectors. In case of breakdown, which happens if $A u_k \in \text{Span}\{v_1, \ldots, v_k\}$ or $B v_k \in \text{Span}\{u_1, \ldots, u_k\}$, we choose an arbitrary unit $v_{k+1} \perp \text{Span}\{v_1, \ldots, v_k\}$ or $u_{k+1} \perp \text{Span}\{u_1, \ldots, u_k\}$ and set $\beta_{k+1} = 0$ or $\gamma_{k+1} = 0$, respectively. We prove by induction that the following statement, denoted $P(k)$, is verified:

\[
\begin{align*}
v_j^T v_{k+1} &= 0 \quad \text{and} \quad u_j^T u_{k+1} = 0 \quad (j = 1, \ldots, k).
\end{align*}
\]

In view of the above, $v_j^T v_{k+1} = 0$ clearly holds if $\beta_{k+1} = 0$, while $u_j^T u_{k+1} = 0$ holds if $\gamma_{k+1} = 0$. Thus we focus on the case where (2.2) applies. Because $v_1$ and $u_1$ are unit vectors,

\[
\begin{align*}
\beta_2 v_1 v_2 &= v_1^T A u_1 - (v_1^T A u_1) v_1 = (1 - \|v_1\|^2)(v_1^T A u_1) = 0, \\
\gamma_2 u_1 u_2 &= u_1^T B v_1 - (u_1^T B v_1) u_1 = (1 - \|u_1\|^2)(u_1^T B v_1) = 0,
\end{align*}
\]

so that the base case $P(1)$ holds. Let $P(1), \ldots, P(k - 1)$ hold. For $j = 1, \ldots, k$, (2.2) implies

\[
\begin{align*}
\beta_{k+1} v_j v_{k+1} &= v_j^T A u_k - \sum_{i=1}^{k} (v_i^T A u_k) v_j v_i = v_j^T A u_k - (v_j^T A u_k) v_j v_j = 0, \\
\gamma_{k+1} u_j u_{k+1} &= u_j^T B v_k - \sum_{i=1}^{k} (u_i^T B v_k) u_j u_i = u_j^T B v_k - (u_j^T B v_k) u_j u_j = 0,
\end{align*}
\]
so that \( P(k) \) also holds. For \( j = 1, \ldots, k - 1 \), we have from (2.2) and \( P(k) \) that

\[
\begin{align*}
v_{k+1}^T A v_j &= v_{k+1}^T \left( \beta_{j+1} v_{j+1} + \sum_{i=1}^j (v_i^T A v_j) v_i \right) = 0, \\
u_{k+1}^T B v_j &= u_{k+1}^T \left( \gamma_{j+1} u_{j+1} + \sum_{i=1}^j (u_i^T B v_j) u_i \right) = 0,
\end{align*}
\]

because \( k + 1 > j + 1 \). Thus, \( V := [v_1 \cdots v_p], U := [u_1 \cdots u_p] \),

\[
H = \begin{bmatrix}
v_1^T A u_1 & v_1^T A u_2 & \cdots & v_1^T A u_p \\
\vdots & \ddots & \ddots & \vdots \\
v_{p-1}^T A u_p & v_p^T A u_1 & \cdots & v_p^T A u_{p-1} \\
\beta_2 & \cdots & \cdots & \beta_p
\end{bmatrix}
\quad \text{and} \quad
F = \begin{bmatrix}
\gamma_1 & \gamma_2 & \cdots & \gamma_{p-1} \\
\vdots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & \gamma_p
\end{bmatrix}
\]

satisfy (2.1a)–(2.1b) and have the properties announced.

Algorithm 2.1 formalizes a Hessenberg reduction process derived from Theorem 2.1.

**Algorithm 2.1 Orthogonal Hessenberg reduction**

**Require:** \( A, B, b, c \), all nonzero

1. \( \beta v_1 = b, \gamma u_1 = c \) \((\beta, \gamma) > 0 \) so that \(|v_1| = |u_1| = 1 \)

2. for \( k = 1, 2, \ldots \) do

3. for \( i = 1, \ldots, k \) do

4. \( h_{i,k} = v_i^T A u_k \)

5. \( f_{k+1,k} = u_k^T B v_k \)

6. end for

7. \( h_{k+1,k} v_{k+1} = A v_k - \sum_{i=1}^k h_{i,k} v_i \) \( h_{k+1,k} > 0 \) so that \(|v_{k+1}| = 1 \)

8. \( f_{k+1,k} u_{k+1} = B v_k - \sum_{i=1}^k f_{i,k} u_i \) \( f_{k+1,k} > 0 \) so that \(|u_{k+1}| = 1 \)

9. end for

Define \( V_k := [v_1 \cdots v_k] \) and \( U_k := [u_1 \cdots u_k] \). After \( k \) iterations of Algorithm 2.1, the situation may be summarized as

\[
\begin{align*}
AU_k &= V_k H_k + h_{k+1,k} v_{k+1} e_k^T = V_{k+1} H_{k+1,k} \\
BV_k &= U_k F_k + f_{k+1,k} u_{k+1} e_k^T = U_{k+1} F_{k+1,k} \\
V_k^T V_k &= U_k^T U_k = I_k,
\end{align*}
\]

where

\[
H_k = \begin{bmatrix}
h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\
h_{2,1} & \ddots & \ddots & \vdots \\
\ddots & \ddots & \ddots & h_{k-1,k} \\
h_{k,k-1} & h_{k,k}
\end{bmatrix}, \quad F_k = \begin{bmatrix}
f_{1,1} & f_{1,2} & \cdots & f_{1,k} \\
f_{2,1} & \ddots & \ddots & \vdots \\
\ddots & \ddots & \ddots & f_{k-1,k} \\
f_{k,k-1} & f_{k,k}
\end{bmatrix},
\]

and

\[
H_{k+1,k} = H_k h_{k+1,k} e_k^T, \quad F_{k+1,k} = F_k f_{k+1,k} e_k^T.
\]

If \( B = A^T \), Algorithm 2.1 reduces to the orthogonal tridiagonalization process of Saunders et al. [25]. \( H_k \) and \( F_k \) are tridiagonal and \( H_k = F_k^T \). Algorithm 2.1 uses
Algorithm 2.1 generates basis vectors, i.e.,

\[ V_k^T A U_k = H_k \quad \text{and} \quad U_k^T B V_k = F_k, \]

which imply that the singular values of \( H_k \) and \( F_k \) are estimates of those of \( A \) and \( B \), respectively. That is in contrast with the process of Arnoldi [1], which can be used to approximate eigenvalues.

### 2.1. Relation with the block-Arnoldi process.

For \( k \geq 1 \),

\[
\begin{align*}
(2.5a) & \quad v_{2k} \in \text{Span}\{ b , \ldots , (AB)^{k-1} b , Ac , \ldots , (AB)^{k-1} Ac \}, \\
(2.5b) & \quad v_{2k+1} \in \text{Span}\{ b , \ldots , (AB)^k b , Ac , \ldots , (AB)^k Ac \}, \\
(2.5c) & \quad u_{2k} \in \text{Span}\{ c , \ldots , (BA)^{k-1} c , Bb , \ldots , (BA)^{k-1} Bb \}, \\
(2.5d) & \quad u_{2k+1} \in \text{Span}\{ c , \ldots , (BA)^k c , Bb , \ldots , (BA)^k Bb \}.
\end{align*}
\]

The subspaces generated by Algorithm 2.1 can be viewed as the union of two block-Krylov subspaces generated by \( AB \) and \( BA \) with respective starting blocks \([ b \quad Ac] \) and \([ c \quad Bb] \). Note the similarity between (2.9) and a Krylov process in which basis vectors have been permuted. Let

\[ P_k := [e_1 \quad e_{k+1} \quad \cdots \quad e_i \quad e_{k+i} \quad \cdots \quad e_k \quad e_{2k}] = [E_1 \quad \cdots \quad E_k], \quad E_k := [e_k \quad e_k] \]

denote the permutation introduced by Paige [21] that restores the order in which Algorithm 2.1 generates basis vectors, i.e.,

\[ W_k := \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} P_k = [w_1 \quad \cdots \quad w_k], \quad w_k = [v_k \quad 0] := [\tilde{v}_k \quad \tilde{u}_k], \]

where we defined \( v_k := (v_k, 0) \) and \( u_k := (0, u_k) \), and we abusively write \([w_1 \quad \cdots \quad w_k] \) instead of \([v_1 \quad \tilde{u}_1 \quad \cdots \quad v_k \quad \tilde{u}_k]\). The projection of \( K_0 \) into the block-Krylov subspace \( \text{Span}\{w_1 , \ldots , w_k\} := \text{Span}\{\tilde{v}_1 \quad \tilde{u}_1 \quad \cdots \quad \tilde{v}_k \quad \tilde{u}_k\} \) is also shuffled to block-Hessenberg form with blocks of size 2. Indeed, if we multiply (2.9) on the right with \( P_k \) and use (2.6), we obtain

\[ K_0 W_k = \begin{bmatrix} V_{k+1} & 0 \\ 0 & U_{k+1} \end{bmatrix} P_{k+1} P_{k+1}^T \begin{bmatrix} 0 & H_{k+1,k} \\ F_{k+1,k} & 0 \end{bmatrix} P_k = W_{k+1} G_{k+1,k}, \]

where

\[ G_{k+1,k} = \begin{bmatrix} \Psi_{1,1} & \Psi_{1,2} & \cdots & \Psi_{1,k} \\ \Psi_{2,1} & \Psi_{2,2} & \cdots & \vdots \\ \vdots & \ddots & \ddots & \Psi_{k-1,k} \\ \vdots & \cdots & \Psi_{k,k} & \Psi_{k+1,k} \end{bmatrix}, \quad \Psi_{i,j} = \begin{bmatrix} 0 & h_{i,j} \\ f_{i,j} & 0 \end{bmatrix}. \]

The two relations at line 1 of Algorithm 2.1 can be rearranged as

\[ \begin{bmatrix} v_1 \\ 0 \quad u_1 \end{bmatrix} \begin{bmatrix} \beta & 0 \\ 0 & \gamma \end{bmatrix} = \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} \iff v_1 \Gamma = D. \]
Identities (2.7) and (2.8) characterize the block-Arnoldi process applied to $K_0$ with initial block $D$. We summarize the process as Algorithm 2.2 where all $w_k \in \mathbb{R}^{(n+m)\times 2}$ and $\Psi_{i,k} \in \mathbb{R}^{2\times 2}$ are determined such that both $w_k^T w_k = I_2$ and the equations on lines 1, 4 and 6 are verified.

**Algorithm 2.2 Block-Arnoldi Process**

**Require:** $K_0$, $D$

1: $w_1 \Gamma = D$

2: for $k = 1, 2, \ldots$ do

3: for $i = 1, \ldots, k$ do

4: $\Psi_{i,k} = w_i^T K_0 w_k$

5: end for

6: $w_{k+1} \Psi_{k+1,k} = K_0 w_k - \sum_{i=1}^{k} w_i \Psi_{i,k}$

7: end for

2.2. Regularization of the block-Arnoldi process. Merging (2.4a)–(2.4b) gives

$$
\begin{bmatrix}
0 & A \\
B & 0
\end{bmatrix}
\begin{bmatrix}
V_k & 0 \\
0 & U_k
\end{bmatrix}
=
\begin{bmatrix}
V_{k+1} & 0 \\
0 & U_{k+1}
\end{bmatrix}
\begin{bmatrix}
0 & H_{k+1,k} \\
F_{k+1,k} & 0
\end{bmatrix},
$$

which is reminiscent of the relation one would obtain from applying an orthogonalization process to $K_0$. Because $K = K_0 + \text{blkdiag}(\lambda I, \mu I)$, (2.9) yields

$$
\begin{bmatrix}
\lambda I & A \\
0 & \mu I
\end{bmatrix}
\begin{bmatrix}
V_k & 0 \\
0 & U_k
\end{bmatrix}
=
\begin{bmatrix}
0 & A \\
B & 0
\end{bmatrix}
\begin{bmatrix}
\lambda I & 0 \\
0 & \mu I
\end{bmatrix}
\begin{bmatrix}
0 & V_k \\
0 & U_k
\end{bmatrix}
+
\begin{bmatrix}
v_{k+1} & 0 \\
0 & u_{k+1}
\end{bmatrix}
\begin{bmatrix}
0 & h_{k+1,k} e_T^k \\
j_{k+1,k} e_T^k & 0
\end{bmatrix}
$$

(2.10)

The same reasoning applied to (2.7) yields the following result, which parallels Montoison and Orban [17, Theorem 2.1].

**THEOREM 2.2.** Given the matrix $K$ defined in (1.3) and the block right-hand side $D$ defined in (1.5), the Krylov basis $W_k = [w_1 \cdots w_k]$ generated by Algorithm 2.2 with regularization has the form (2.6) where the vectors $u_k$ and $v_k$ are the same as those generated by Algorithm 2.1 with initial vectors $b$ and $c$. In addition,

$$
K W_k = W_{k+1} S_{k+1,k}, \quad S_{k+1,k} := \\
\begin{bmatrix}
\Theta_{1,1} & \Psi_{1,2} & \cdots & \Psi_{1,k} \\
\Psi_{2,1} & \Theta_{2,2} & \cdots & \Psi_{k-1,k} \\
& \ddots & \ddots & \Psi_{k-1,k} \\
& & \ddots & \Theta_{k,k} \\
& & & \Psi_{k+1,k}
\end{bmatrix},
$$

where

$$
\Theta_{j,j} = \begin{bmatrix}
\lambda & h_{j,j} \\
j_{j,j} & \mu
\end{bmatrix} \quad \text{and} \quad \Psi_{i,j} = \begin{bmatrix}
0 & h_{i,j} \\
f_{i,j} & 0
\end{bmatrix}. \quad j = 1, \ldots, k, \quad i = 1, \ldots, j+1, \quad i \neq j.
$$

The scalars $h_{i,j}$, $f_{i,j}$ are those generated by Algorithm 2.1 applied to $A$ and $B$ with initial vectors $b$ and $c$. 
Proof. Algorithm 2.2 applied to $K_0$ generates sparse pairs $w_k$ as in (2.6) because of the equivalence with Algorithm 2.1. The term $\text{blkdiag}(\lambda I, \mu I)$ can be seen as a regularization term:

\[(2.12) \quad \begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix} w_k = w_k \Lambda \quad \text{with} \quad \Lambda := \begin{bmatrix} \lambda & 0 \\ 0 & \mu \end{bmatrix}.\]

The identities (2.7) and (2.12) allow us to write

\[(2.13) \quad K W_k = W_{k+1} \begin{bmatrix} \Psi_{1,1} + \Lambda & \Psi_{1,2} & \cdots & \Psi_{1,k} \\ \Psi_{2,1} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \Psi_{k-1,1} & \cdots & \Psi_{k-1,k} & \Psi_{k,k} + \Lambda \\ \Psi_{k,k} & \cdots & \Psi_{k,k} - 1 & \Psi_{k+1,k} \end{bmatrix},\]

which amounts to (2.11) because $\Theta_{k,k} = \Psi_{k,k} + \Lambda$.

Note that (2.11) is identical to (2.10) where the order of the $w_k$ has been permuted according to $P_k$.

Because of Theorem 2.2, the Krylov basis $W_k$ generated by Algorithm 2.2 must have the sparsity structure (2.6), so that only $u_k$ and $v_k$ need be generated, and they may be generated directly from Algorithm 2.1. The key point is that generating orthonormal bases of $K_k(K, d)$ and $K_k(K, D)$ by the Arnoldi process and Algorithm 2.1, respectively, require exactly the same amount of storage and $K_k(K, d) \subset K_k(K, D)$. Thus, residual norms produced by Gmres are certain to be at least as large as those generated by a minimum-residual method that seeks an approximate solution $x_k$ in $K_k(K, D)$. Such a method is the subject of the next section.

3. Derivation of Gpmr. In this section, we develop the method Gpmr based upon Algorithm 2.1 with regularization to solve (1.4) in which the $k$-th iterate has the form

\[(3.1) \quad \begin{bmatrix} x_k \\ y_k \end{bmatrix} = W_k z_k,\]

where $z_k \in \mathbb{R}^{2k}$. Thanks to (2.8) and (2.11), the residual can be written

\[(3.2) \quad r_k = \begin{bmatrix} b \\ c \end{bmatrix} - \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} = w_1 \begin{bmatrix} \beta \\ \gamma \end{bmatrix} - W_{k+1} S_{k+1,k} z_k = W_{k+1} (\beta e_1 + \gamma e_2 - S_{k+1,k} z_k).\]

Because $W_{k+1}$ has orthonormal columns, $\|r_k\|$ can be minimized by defining $z_k$ as the solution of the linear least-squares problem

\[(3.3) \quad \min_{z_k \in \mathbb{R}^{2k}} \|S_{k+1,k} z_k - (\beta e_1 + \gamma e_2)\|.|}

3.1. Relation between Gpmr and Block-Gmres. The $k$-th Block-Gmres iterate is defined by the matrix linear least-squares problem

\[(3.4) \quad \min_{x_k \in \mathbb{R}^{2k}} \left\| \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} - \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x_k^b \\ x_k^c \\ y_k^b \\ y_k^c \end{bmatrix} \right\|.|}
where \((x_k^b, y_k^b) = W_k z_k^b\) and \((x_k^c, y_k^c) = W_k z_k^c\). Accordingly, the \(k\)-th Block-Gmres subproblem is

\[
\text{minimize} \quad \left\| S_{k+1} \begin{bmatrix} z_k^b \\ z_k^c \end{bmatrix} - \begin{bmatrix} \beta e_1 \\ \gamma e_2 \end{bmatrix} \right\|_F, 
\]

so that \(z_k^b\) and \(z_k^c\) solve the subproblem associated with right-hand sides \(\beta e_1\) and \(\gamma e_2\). In exact arithmetic, the solutions of (3.3) and (3.5) are connected via

\[ z_k = z_k^b + z_k^c, \]

and the Gpmr and Block-Gmres approximations are connected via

\[ x_k = x_k^b + x_k^c \quad \text{and} \quad y_k = y_k^b + y_k^c. \]

We now outline the main stages for solving (3.3).

### 3.2. A QR factorization.

The solution of (3.3) can be determined via the QR factorization

\[
S_{k+1,k} = Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix},
\]

which can be updated at each iteration, where \(Q_k \in \mathbb{R}^{(2k+2) \times (2k+2)}\) is a product of Givens reflections, and \(R_k \in \mathbb{R}^{(2k) \times (2k)}\) is upper triangular. At each iteration, four new reflections are necessary to update (3.6). We denote their product \(Q_{1,2k+2} \ldots Q_{1,4}\). For \(i = 1, \ldots, k\), the structure of \(Q_{1,2i+2}\)

\[
\begin{bmatrix}
1 & \cdots & 2i-2 & 2i-1 & 2i & 2i+1 & 2i+2 & 2i+3 & \cdots & 2k+2 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \vdots & & & & & & & & \\
2i-2 & & & & & & & & & \\
2i-1 & \times & \times & \times & \times & & & & & \\
2i & \times & \times & \times & \times & & & & & \\
2i+1 & \times & \times & \times & \times & & & & & \\
2i+2 & \times & \times & \times & \times & & & & & \\
2i+3 & \vdots & & & & & & & & & \\
\vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
2k+2 & & & & & & & & & 1
\end{bmatrix}
\]

where the diagonal block extracted from rows and columns \(2i-1, \ldots, 2i+2\) is the product of the following four Givens reflections

\[
\begin{bmatrix}
1 & c_{4,i} & s_{4,i} \\
s_{4,i} & -c_{4,i} & 1
\end{bmatrix}
\begin{bmatrix}
1 & c_{3,i} & s_{3,i} \\
s_{3,i} & -c_{3,i} & 1
\end{bmatrix}
\begin{bmatrix}
c_{2,i} & s_{2,i} \\
s_{2,i} & -c_{2,i}
\end{bmatrix}
\begin{bmatrix}
c_{1,i} & s_{1,i} \\
s_{1,i} & -c_{1,i}
\end{bmatrix}.
\]

The result \((a_1^\text{out}, a_2^\text{out}, a_3^\text{out}, a_4^\text{out})\) of a matrix-vector product between the above \(4 \times 4\) block and a vector \((a_1^\text{in}, a_2^\text{in}, a_3^\text{in}, a_4^\text{in})\) can be obtained via Algorithm 3.1.

At iteration \(k\), Algorithm 2.1 generates two new columns, and to update the QR
Algorithm 3.1 Procedure ref

Require: $i, a_1^{in}, a_2^{in}, a_4^{in}$

1: $t = c_{1,k}a_1^{in} + s_{1,k}a_4^{in} ightarrow a_4^{out} = s_{1,k}a_1^{in} - c_{1,k}a_4^{in}$, $a_1^{out} = t$ \hspace{1cm} \text{first reflection}
2: $t = c_{2,k}a_1^{in} + s_{2,k}a_2^{in} ightarrow a_2^{out} = s_{2,k}a_1^{in} - c_{2,k}a_2^{in}$, $a_1^{out} = t$ \hspace{1cm} \text{second reflection}
3: $t = c_{3,k}a_2^{out} + s_{3,k}a_4^{out} ightarrow a_4^{out} = s_{3,k}a_2^{out} - c_{3,k}a_4^{out}$, $a_2^{out} = t$ \hspace{1cm} \text{third reflection}
4: $t = c_{4,k}a_2^{out} + s_{4,k}a_3^{in} ightarrow a_3^{out} = s_{4,k}a_2^{out} - c_{4,k}a_3^{in}$, $a_2^{out} = t$ \hspace{1cm} \text{fourth reflection}

decomposition we need first to apply all previous reflections as follows

$$Q_{k-1}^T = Q_{2k-5,2k-2} \cdots Q_{3,6} \begin{bmatrix} r_{1,2k-1} & r_{1,2k} \\ r_{2,2k-1} & r_{2,2k} \\ r_{3,2k-1} & r_{3,2k} \\ r_{4,2k-1} & r_{4,2k} \\ \vdots & \vdots \\ \Psi_{k+1,k} & \end{bmatrix} = \begin{bmatrix} r_{1,2k-1} & r_{1,2k} \\ \vdots & \vdots \\ r_{2k-2,2k-1} & r_{2k-2,2k} \\ r_{2k-1,2k-1} & r_{2k-1,2k} \\ h_{k+1,k} \\ f_{k+1,k} & \end{bmatrix},$$

and then compute and apply the four reflections that constitute $Q_{2k-1,2k+2}$ such that coefficients under the diagonal are zeroed out

$$Q_{2k-1,2k+2} = \begin{bmatrix} r_{1,2k-1} & r_{1,2k} \\ \vdots & \vdots \\ r_{2k-2,2k-1} & r_{2k-2,2k} \\ r_{2k-1,2k-1} & r_{2k-1,2k} \\ r_{2k,2k} & h_{k+1,k} \\ f_{k+1,k} & \end{bmatrix} = \begin{bmatrix} r_{1,2k-1} & r_{1,2k} \\ \vdots & \vdots \\ r_{2k-2,2k-1} & r_{2k-2,2k} \\ r_{2k-1,2k-1} & r_{2k-1,2k} \\ 0 & 0 \\ 0 & 0 & \end{bmatrix}.$$

A procedure to compute the Givens sines and cosines, and finalize the QR factorization of $S_{k+1,k}$ is described as Algorithm 3.2. Note that the first parameter of Algorithm 3.1

Algorithm 3.2 Procedure givens

Require: $k, r_{2k-1,2k-1}, r_{2k-2,2k-1}, r_{2k,2k}, h_{k+1,k}, f_{k+1,k}$

1: $\tilde{r}_{2k-1,2k-1} = (\tilde{r}_{2k-1,2k-1} + f_{k+1,k})^\frac{1}{2}$ \hspace{1cm} \text{annihilate } f_{k+1,k}
2: $c_{1,k} = \tilde{r}_{2k-1,2k-1}/\tilde{r}_{2k,2k-1}$, $s_{1,k} = f_{k+1,k}/\tilde{r}_{2k-1,2k-1}$
3: $r_{2k,2k} = s_{1,k}r_{2k,2k-1}$
4: $s_{2,k} = \tilde{r}_{2k,2k-1}r_{2k,2k-1} + s_{2,k}r_{2k,2k}$
5: $r_{2k-1,2k-1} = (\tilde{r}_{2k-1,2k-1} + r_{2k,2k}^2)^\frac{1}{2}$ \hspace{1cm} \text{annihilate } \tilde{r}_{2k,2k-1}
6: $c_{2,k} = \tilde{r}_{2k-2,2k-1}/r_{2k-1,2k-1}$, $s_{2,k} = r_{2k,2k-1}/r_{2k,2k}$
7: $r_{2k-1,2k} = c_{2,k}r_{2k,2k-1} + s_{2,k}r_{2k,2k}$
8: $\tilde{r}_{2k,2k} = s_{2,k}r_{2k,2k} - c_{2,k}r_{2k,2k}$
9: $\tilde{r}_{2k,2k} = (\tilde{r}_{2k,2k}^2 + r_{2k+2,2k}^2)^\frac{1}{2}$ \hspace{1cm} \text{annihilate } \tilde{r}_{2k+2,2k}
10: $c_{3,k} = \tilde{r}_{2k,2k}/\tilde{r}_{2k,2k}$, $s_{3,k} = \tilde{r}_{2k+2,2k}/\tilde{r}_{2k,2k}$
11: $r_{2k+1,2k} = (\tilde{r}_{2k,2k} + h_{k+1,k})^\frac{1}{2}$ \hspace{1cm} \text{annihilate } h_{k+1,k}
12: $s_{4,k} = \tilde{r}_{2k,2k}/r_{2k,2k}$, $h_{k+1,k} = h_{k+1,k}/\tilde{r}_{2k,2k}$

and Algorithm 3.2 is used to define which Givens sines and cosines are read from or written to memory.
3.3. GMPR iterate and residual norm computation. We have from (3.2) and (3.6):

\[ \|r_k\| = \|Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix} z_k - (\beta e_1 + \gamma e_2)\| = \|\begin{bmatrix} R_k \\ 0 \end{bmatrix} z_k - \bar{t}_k\|, \]

where \( \bar{t}_k := Q_k^T(\beta e_1 + \gamma e_2) = (t_k, \bar{\tau}_2k+1, \bar{\tau}_2k+2) \), \( t_k := (\tau_1, \ldots, \tau_{2k}) \) represents the first 2\( k \) components of \( \bar{t}_k \), and the recurrence starts with \( \bar{t}_0 := (\bar{\tau}_1, \bar{\tau}_2) = (\beta, \gamma) \). \( \bar{t}_k \) can be easily determined from \( \bar{t}_{k-1} \) because \( \bar{t}_k = Q_{2k-1,2k+2}(\bar{t}_{k-1}, 0, 0) \). The solution of (3.3) is thus \( z_k := (\bar{\zeta}_1, \ldots, \bar{\zeta}_{2k}) \) found by solving \( R_k z_k = t_k \) with backward substitution.

The definitions of \( \bar{t}_k \) and \( z_k \) together with (3.7) yield

\[ \|r_k\| = \sqrt{\bar{\tau}_{2k+1}^2 + \bar{\tau}_{2k+2}^2}. \]

As in GMRES, we only compute \( z_k \) when \( \|r_k\| \) is smaller than a user-provided threshold. Thanks to (3.1), the solution may be computed efficiently as

\begin{align}
(3.9a) & \quad x_k = \sum_{i=1}^{k} \bar{\zeta}_{2i-1} v_i, \\
(3.9b) & \quad y_k = \sum_{i=1}^{k} \bar{\zeta}_{2i} u_i.
\end{align}

We summarize the complete procedure as Algorithm 3.3.

3.4. Memory requirements. Table 3.1 summarizes the storage costs of \( k \) iterations of GMPR, GMRES and Block-GMRES.

| Variables | GMPR | GMRES | Block-GMRES |
|-----------|------|-------|--------------|
| \( (x_k, y_k) \) | \( m+n \) | \( m+n \) | \( 2(m+n) \) |
| \( (q, p) \) | \( m+n \) | \( m+n \) | \( 2(m+n) \) |
| \( (V_k, U_k) \) | \( k(m+n) \) | \( k(m+n) \) | \( 2k(m+n) \) |
| \( t_k \) | \( 2k \) | \( k \) | \( 4k \) |
| \( z_k \) | \( 2k \) | \( k \) | \( 4k \) |
| \( Q_k \) | \( 8k \) | \( k \) | \( 8k \) |
| \( R_k \) | \( k(2k+1) \) | \( k(2k+1)/2 \) | \( k(2k+1) \) |

Some GMPR variables are paired in Table 3.1 to easily identify their GMRES and Block-GMRES counterparts. Note that \( t_k \) and \( z_k \) can share the same storage because \( R_k t_k = z_k \) can be solved in-place.

4. Implementation and numerical experiments. We implemented Algorithm 3.3 in Julia [3], version 1.6, as part of our Krylov.jl collection of Krylov methods [18]. Our implementation of GMPR is applicable in any floating-point system supported by Julia, and runs on CPU and GPU. The GPU support can be particularly relevant for (1.2) because, as a Krylov method, GMPR only requires linear operators that model \( A_{TT} u \), \( B_{TT} v \), \( u \mapsto M_{TT}^{-1} u \) and \( v \mapsto N_{TT}^{-1} v \). For instance, \( v \mapsto N_{TT}^{-1} v \) can be the forward and backward substitutions with the factors of an LU decomposition of \( N_{TT} \). The use of abstract linear operators allows us to store \( A_{TT} \) and \( B_{TT} \) as well as decompositions of the diagonal blocks of (1.2) on distinct compute nodes and leverage parallel architectures, such as GPUs. When the matrices are unstructured, Duff and Scott [7] propose a robust arrowhead reordering such that each diagonal block is nonsingular and recovers a system of the form (1.2).

We evaluate the performance of GMPR on systems generated from unsymmetric matrices in the SuiteSparse Matrix Collection [5]. We use METIS to form a \( 2 \times 2 \) block matrix and use the two diagonal blocks to build a right block-Jacobi preconditioner \( P_r \)
Algorithm 3.3 GPMR

Require: $A, B, b, c, \lambda, \mu, \epsilon > 0, k_{\text{max}} > 0$

1. $\beta v_1 = b, \gamma u_1 = c \quad (\beta, \gamma) > 0$ so that $\|v_1\| = \|u_1\| = 1$
2. $\tau_1 = \beta, \tau_2 = \gamma$
3. $\|r_0\| = (\tau_1^2 + \tau_2^2)^{1/2}$
4. $k = 0$
5. while $\|r_k\| > \epsilon$ and $k < k_{\text{max}}$ do
   6. $k \leftarrow k + 1$
   7. $q = A u_k$
   8. $p = B v_k$
   9. for $i = 1, \ldots, k$ do
      10. $h_{i,k} = \bar{v}_i q$
      11. $f_{i,k} = u_i p$
      12. $q = q - h_{i,k} v_i$
      13. $p = p - f_{i,k} u_i$
   14. end for
   15. $h_{k+1,k} v_{k+1} = q$
   16. $f_{k+1,k} u_{k+1} = p$
   17. $\bar{r}_{1,2k} = h_{1,k}, \bar{r}_{2,2k-1} = f_{1,k}$
   18. if $k \neq 1$ then $(\bar{r}_{1,2k-1}, \bar{r}_{2,2k}) = (0, 0)$ else $(\bar{r}_{1,2k-1}, \bar{r}_{2,2k}) = (\lambda, \mu)$
   19. for $i = 1, \ldots, k-1$ do
      20. if $i \neq k - 1$ then $(\rho, \delta) = (0, 0)$ else $(\rho, \delta) = (\lambda, \mu)$
      21. $r_{2i-1,2k-1}, r_{2i+1,2k-1}, r_{2i+2,2k-1} = \text{ref}(i, \bar{r}_{2i-1,2k-1}, \bar{r}_{2i+1,2k-1}, \rho, f_{i+1,k})$
      22. $r_{2i-1,2k}, r_{2i,2k}, r_{2i+1,2k}, r_{2i+2,2k} = \text{ref}(i, \bar{r}_{2i-1,2k}, \bar{r}_{2i,2k}, h_{i+1,k}, \delta)$
   23. end for
   24. $r_{2k-1,2k-1}, r_{2k-2,2k}, r_{2k-2,2k}$ = Compute and apply $Q_{2k-1,2k+2}$
      given $(k, \bar{r}_{2k-1,2k-1}, \bar{r}_{2k-2,2k}, \bar{r}_{2k,2k}, h_{k+1,k}, f_{k+1,k})$
   25. $\tau_{2k-1}, \tau_{2k}, \tau_{2k+1}, \tau_{2k+2} = \text{ref}(k, \bar{r}_{2k-1,2k}, \bar{r}_{2k,2k}, 0, 0)$
   26. $\|r_k\| = (\tau_{2k+1}^2 + \tau_{2k+2}^2)^{1/2}$
   27. end while
   28. $\zeta_{2k} = \tau_{2k}/r_{2k,2k}$
   29. for $i = 2k-1, \ldots, 1$ do
      30. $\zeta_i = (\tau_i - \sum_{j=i+1}^{2k} r_{i,j} \zeta_j)/r_{i,i}$
   31. end for
   32. $x_k = \sum_{i=1}^{k} \zeta_{2i-1} v_i$
   33. $y_k = \sum_{i=1}^{k} \zeta_{2i} u_i$

with $\lambda = \mu = 1$. We set $P_k = I$ so the residual norm of (1.1) is identical to that of (1.4). The right-hand side $(b, c)$ is generated so the exact solution of (1.1) is the vector of ones. We compare GPMR to our implementation of GMRES without restart in terms of number of iterations. Each algorithm stops as soon as $\|r_k\| \leq \varepsilon_a + \|(b, c)\| \varepsilon_w$ with absolute tolerance $\varepsilon_a = 10^{-12}$ and relative tolerance $\varepsilon_r = 10^{-10}$. Table 4.1 summarizes our results, which show an improvement in terms of number of iterations ranging from about 10% up to 50% in favor of GPMR. Figure 4.1 reports residual histories of GPMR, GMRES and BLOCK-GMRES where the two approximate solutions are summed on problems scircuit, sme5Dc, PR02R and sherman5.

The GPMR and BLOCK-GMRES residuals are nearly superposed except for scircuit, on which BLOCK-GMRES stagnates. The same phenomenon occurs on a generalized
Table 4.1
Number of iterations of Gpmr and Gmres on systems from the SuiteSparse Matrix Collection.

| name       | size   | nnz  | Gmres | Gpmr | gain  |
|------------|--------|------|-------|------|-------|
| sherman5   | 3312   | 20736| 25    | 20   | 20%   |
| powersim   | 15838  | 67562| 141   | 101  | 28%   |
| Ill_Stokes | 20886  | 191368| 59    | 54   | 9%    |
| sme3Dc     | 42930  | 314865| 127   | 78   | 39%   |
| rma10      | 46835  | 237401| 48    | 41   | 15%   |
| ecl32      | 51993  | 380415| 58    | 42   | 28%   |
| venkat50   | 62424  | 1717792| 48   | 35   | 27%   |
| poisson3Dc | 85623  | 2374949| 56   | 50   | 11%   |
| ifiss_mat  | 96307  | 359932| 42    | 33   | 21%   |
| hcircuits  | 105676 | 513072| 47    | 37   | 21%   |
| PR02R      | 161070 | 8185136| 97   | 68   | 30%   |
| scircuit   | 170998 | 958936| 48    | 24   | 50%   |
| transient  | 178866 | 961790| 567   | 470  | 17%   |
| nohe2      | 181343 | 11063545| 50  | 39   | 22%   |
| thermomech_dK | 204316 | 2846228| 128  | 84   | 34%   |
| marine1    | 400320 | 18920347| 84   | 60   | 29%   |
| Freescale1 | 3428755| 18920347| 456  | 344  | 25%   |

Fig. 4.1. Residual history of Gpmr, Gmres and Block-Gmres.

saddle point build using matrices well1033 as A and illc1033 as B, M = I, N = 0, λ = 1 and μ = 0. Figure 4.2 reports residual histories of Gpmr, Gmres and Block-Gmres on the generalized saddle point system in double and quadruple precision. Although theoretically equivalent, Gpmr appears to be less sensitive to arithmetic errors due to loss of orthogonality than its counterpart implementation based on Block-Gmres. Indeed, the number of Gpmr and Gmres iterations is the same in
When $K$, defined in (1.3), is symmetric, Algorithm 2.1 coincides with the orthogonal tridiagonalization process of Saunders et al. [25] because $A^T = B$ and GpMr is theoretically equivalent to TriMR. We verify numerically the equivalence between the two methods on symmetric quasi-definite systems, with matrices $A$ from the SuiteSparse Matrix Collection, $M = N = I$, $\lambda = 1$ and $\mu = -1$. Each algorithm stops with the same tolerance as above. Because GpMr can be viewed as TriMR with full reorthogonalization, we use different floating-point systems to observe any loss of orthogonality in the Krylov basis. Figure 4.3 reports residual histories of GpMr in double precision and TriMR in double, quadruple and octuple precision. The plots suggest that reorthogonalization is a more powerful device than extended precision.

5. Discussion and extensions. Based upon Algorithm 2.1, it is possible to develop another method, GpCG, in the spirit of Fom [22]. The $k$-th GpCG iterate is defined by the Galerkin condition $W_k^T r_k = 0$. Its associated subproblem selects $z_k$ in (3.1) as the solution of the square system

$$S_k z_k = \beta e_1 + \gamma e_2,$$

where $S_k$ denotes the leading $(2k) \times (2k)$ submatrix of $S_{k+1,k}$ in (2.11). However, GpCG may break down if $S_k$ is singular, and in that respect shares the disadvantages of double and quadruple precision.

**Fig. 4.2.** Residual history of GpMr, Gmres and Block-Gmres on the generalized saddle point system in double (left) and quadruple precision (right).

**Fig. 4.3.** Residual history of GpMr and TriMR.
FOM, whereas the GPMR iterates are always well defined. GPCG could still be relevant for unsymmetric structured and positive-definite linear systems, such as those arising from the finite-element discretization of advection-diffusion equations [26], where $S_k$ is guaranteed to be nonsingular. Indeed, if $K$ is positive definite, its projection $S_k = W_k^T K W_k$ into the $k$-th Krylov subspace is also positive definite, which ensures that (5.1) has a unique solution. The same observation holds for FOM and BiCG [11], which should be restricted to certain classes of linear systems to avoid breakdowns.

Although the focus of GPMR is on unsymmetric linear systems, Figure 4.3 shows that it is also relevant for ill-conditioned symmetric linear systems. Moreover, GPMR allows to solve symmetric partitioned systems with symmetric indefinite blocks $M$ and $N$, whereas TriMR requires them to be zero or definite matrices.

A variant with restart in the spirit of GMRES($k$) is easily implemented on top of GPMR. A limited-memory variant of GPMR can be also developed and compared to DQGMRES [24]. We leave the investigation of such extension to future work.

REFERENCES

[1] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Q. Appl. Math.*, 9:17–29, 1951.

[2] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

[3] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Rev.*, 59(1):65–98, 2017.

[4] A. Buttari, D. Orban, D. Ruiz, and D. Titeley-Peloquin. USYMLQR: A tridiagonalization method for symmetric saddle-point systems. *SIAM J. Sci. Comput.*, 41(5):409–432, 2019.

[5] T. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Software*, 38(1):1–25, 2011.

[6] V. Dolean, P. Jolivet, and F. Nataf. An introduction to domain decomposition methods: algorithms, theory, and parallel implementation. *SIAM*, 2015.

[7] I. S. Duff and J. A. Scott. Stabilized bordered block diagonal forms for parallel sparse solvers. *Parallel Computing*, 31(3-4):275–289, 2005.

[8] H. C. Elman. Preconditioners for saddle point problems arising in computational fluid dynamics. *Applied Numerical Mathematics*, 43(1-2):75–89, 2002.

[9] R. Estrin and C. Greif. SPMR: A family of saddle-point minimum residual solvers. *SIAM J. Sci. Comput.*, 40(3):1884–1914, 2018.

[10] M. C. Ferris and J. D. Horn. Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80(1):35–61, 1998.

[11] R. Fletcher. Conjugate gradient methods for indefinite systems. In *Numerical Analysis*, pages 73–89. Springer, 1976.

[12] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60(1):315–339, 1991.

[13] C. Greif and M. Wathen. Conjugate gradient for nonsingular saddle-point systems with a maximally rank-deficient leading block. *J. Comput. Appl. Math.*, 358:1–11, 2019.

[14] I. C. Ipsen. A note on preconditioning nonsymmetric matrices. *SIAM J. Sci. Comput.*, 23(3):1050–1051, 2001.

[15] G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, 1997.

[16] A. Montoison and D. Orban. BiLQ: An iterative method for nonsymmetric linear systems with a quasi-minimum error property. *SIAM J. Matrix Anal. Appl.*, 41(3):1145–1166, 2020.

[17] A. Montoison and D. Orban. TriCG and TriMR: Two iterative methods for symmetric quasi-definite systems. *SIAM J. Sci. Comput.*, 43(4):2502–2525, 2021.

[18] A. Montoison, D. Orban, and contributors. Krylov.jl: A Julia basket of hand-picked Krylov methods. https://github.com/JuliaSmoothOptimizers/Krylov.jl, June 2020.

[19] M. F. Murphy, G. H. Golub, and A. J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM J. Sci. Comput.*, 21(6):1969–1972, 2000.

[20] D. Orban and M. Arioli. *Iterative Solution of Symmetric Quasi-Definite Linear Systems*, volume 3 of *Spotlights*. SIAM, 2017.

[21] C. C. Paige. Bidiagonalization of matrices and solution of linear equations. *SIAM J. Numer.*
[22] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of computation*, 37(155):105–126, 1981.

[23] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. and Statist. Comput.*, 7(3):856–869, 1986.

[24] Y. Saad and K. Wu. DQGMRES: A direct quasi-minimal residual algorithm based on incomplete orthogonalization. *Numerical linear algebra with applications*, 3(4):329–343, 1996.

[25] M. A. Saunders, H. D. Simon, and E. L. Yip. Two conjugate-gradient-type methods for unsymmetric linear equations. *SIAM J. Numer. Anal.*, 25(4):927–940, 1988.

[26] T. Xuemin and J. Li. BDDC for nonsymmetric positive definite and symmetric indefinite problems. In *Domain Decomposition Methods in Science and Engineering XVIII*, pages 75–86. Springer, 2009.