Les Cahiers du GERAD ISSN: 0711–2440

GPMR: An iterative method for unsymmetric partitioned linear systems

A. Montoison, D. Orban

G–2021–62

November 2021

Citation suggérée : A. Montoison, D. Orban (Novembre 2021). GPMR: An iterative method for unsymmetric partitioned linear systems, Rapport technique, Les Cahiers du GERAD G– 2021–62, GERAD, HEC Montréal, Canada.

Avant de citer ce rapport technique, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2021-62) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

Suggested citation: A. Montoison, D. Orban (November 2021). GPMR: An iterative method for unsymmetric partitioned linear systems, Technical report, Les Cahiers du GERAD G–2021–62, GERAD, HEC Montréal, Canada.

Before citing this technical report, please visit our website (https://www.gerad.ca/en/papers/G-2021-62) to update your reference data, if it has been published in a scientific journal.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2021
– Bibliothèque et Archives Canada, 2021

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2021
– Library and Archives Canada, 2021

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

GPMR: An iterative method for unsymmetric partitioned linear systems

Alexis Montoison
Dominic Orban

GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3C 3A7

alexis.montoison@polymtl.ca
dominique.orban@gerad.ca

November 2021
Les Cahiers du GERAD
G–2021–62

Abstract : We introduce an iterative method named Gpmr for solving 2×2 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.

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

---

Acknowledgements: Research of the first author is supported by a FRQNT grant and an excellence scholarship of the IVADO institute. Research of the second author is partially supported by an NSERC Discovery Grant.

1 Introduction

Consider the partitioned linear system

$$\begin{bmatrix} M & A_\star \\ B_\star & N \end{bmatrix} \begin{bmatrix} x_\star \\ y_\star \end{bmatrix} = \begin{bmatrix} b_\star \\ c_\star \end{bmatrix}, \tag{1}$$

where $M \in \mathbb{R}^{m\times m}$, $N \in \mathbb{R}^{n\times n}$, $A_\star \in \mathbb{R}^{m\times n}$ and $B_\star \in \mathbb{R}^{n\times m}$. We assume that $A_\star$ and $B_\star$ are nonzero, and that $b_\star \in \mathbb{R}^m$ and $c_\star \in \mathbb{R}^n$ are both nonzero. System (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 $\mathcal{I}$ be the set of all indices of the discretization points that belong to the interior of the subdomains and Γ the set of those corresponding to the interfaces between the subdomains. Grouping the unknowns corresponding to $\mathcal{I}$ by subdomain in $u_\mathcal{I}$ and those corresponding to Γ in u_Γ, we obtain the arrowhead partitioning of the stiffness system

$$\begin{bmatrix} A_{\mathcal{I}\mathcal{I}} & A_{\mathcal{I}\Gamma} \\ A_{\Gamma\mathcal{I}} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_\mathcal{I} \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_\mathcal{I} \\ f_\Gamma \end{bmatrix} \quad \iff \quad \begin{bmatrix} A_{11} & & & A_{1\Gamma} \\ & \ddots & & \vdots \\ & & A_{kk} & A_{k\Gamma} \\ A_{\Gamma 1} & \dots & A_{\Gamma k} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_k \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_k \\ f_\Gamma \end{bmatrix}, \tag{2}$$

where $u = (u_\mathcal{I}, u_\Gamma)$ is the vector of nodal displacements and f the vector of nodal forces. For a tour of applications leading to (1), we refer the reader to [2]. We assume that there exist nonsingular P_ℓ and P_r with inexpensive inverses such that

$$K := P_\ell^{-1} \begin{bmatrix} M & A_\star \\ B_\star & N \end{bmatrix} P_r^{-1} = \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix}, \quad \lambda, \mu \in \mathbb{R}, \tag{3}$$

so that the equivalent preconditioned system

$$\begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}, \quad \begin{bmatrix} x_\star \\ y_\star \end{bmatrix} = P_r^{-1} \begin{bmatrix} x \\ y \end{bmatrix}, \quad \begin{bmatrix} b \\ c \end{bmatrix} = P_\ell^{-1} \begin{bmatrix} b_\star \\ c_\star \end{bmatrix} \tag{4}$$

can be solved instead of (1). Note that λ and/or μ may vanish. For example, the ideal preconditioners of Murphy et al. [19] and Ipsen [14] lead to (3). Although ideal preconditioners are typically impractical because they require the solution of systems with the Schur complement $S = N - B_\star M^{-1} A_\star$, viable preconditioners such that $P_\ell P_r = \mathrm{blkdiag}(M, N)$ can be employed when M and N are both nonsingular.

Given an unstructured matrix C, a practical approach to recovering the matrix of (1) 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 (3).

When $\lambda \neq 0$, (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 (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) is symmetric and positive definite. In this paper, we focus on applying an iterative method to (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 (4), and (iii) an efficient software implementation to solve (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).

Specialized iterative methods have been developed for special cases of (1). Estrin and Greif [9] developed Spmr; a family of methods for (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 $M \succeq 0$ 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 w_k below. For a vector v, $\|v\|$ denotes the Euclidean norm of v, and for a matrix M, $\|M\|_F$ denotes the Frobenius norm of M. The shorthand $y \mapsto M\backslash y$ represents an operator that returns the solution of $Mx = y$. 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

$$K_0 := \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}, \quad \text{blkdiag}(\lambda I, \mu I) = \begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix}, \quad d := \begin{bmatrix} b \\ c \end{bmatrix}, \quad D := \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix}. \tag{5}$$

For a matrix C and a vector t, $\mathcal{K}_k(C,t)$ is the Krylov subspace $\text{Span}\left\{t, Ct, \dots, C^{k-1}t\right\}$. For a matrix T with as many rows as C has columns, $\mathcal{K}_k(C,T)$ is the block-Krylov subspace $\text{Span}\left\{T, CT, \dots, C^{k-1}T\right\}$. We abusively write (b,c) and $l = (l_1, \dots, l_n)$ to represent the column vectors $\begin{bmatrix} b^T & c^T \end{bmatrix}^T$ and $l = \begin{bmatrix} l_1 & \cdots & l_n \end{bmatrix}^T$, 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 regularization.

Theorem 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 othonormal columns, and upper Hessenberg $H \in \mathbb{R}^{p\times p}$ and $F \in \mathbb{R}^{p\times p}$ with nonnegative subdiagonal coefficients such that

$$V^T AU = H, \tag{6a}$$

$$U^T BV = F. \tag{6b}$$

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

$$\beta_{k+1}v_{k+1} = Au_k - \textstyle\sum_{i=1}^k (v_i^T Au_k)v_i, \tag{7a}$$

$$\gamma_{k+1}u_{k+1} = Bv_k - \textstyle\sum_{i=1}^k (u_i^T Bv_k)u_i, \tag{7b}$$

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

$$v_j^T v_{k+1} = 0 \quad \text{and} \quad u_j^T u_{k+1} = 0 \quad (j = 1, \dots, k). \tag{8}$$

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 (7) applies. Because v_1 and u_1 are unit vectors,

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

so that the base case $\mathcal{P}(1)$ holds. Let $\mathcal{P}(1), \dots, \mathcal{P}(k-1)$ hold. For $j = 1, \dots, k$, (7) implies

$$\begin{aligned} \beta_{k+1} v_j^T v_{k+1} &= v_j^T A u_k - \textstyle\sum_{i=1}^k (v_i^T A u_k) v_j^T v_i = v_j^T A u_k - (v_j^T A u_k) v_j^T v_j = 0, \\ \gamma_{k+1} u_j^T u_{k+1} &= u_j^T B v_k - \textstyle\sum_{i=1}^k (u_i^T B v_k) u_j^T u_i = u_j^T B v_k - (u_j^T B v_k) u_j^T u_j = 0, \end{aligned}$$

so that $\mathcal{P}(k)$ also holds. For $j = 1, \dots, k-1$, we have from (7) and $\mathcal{P}(k)$ that

$$\begin{aligned} v_{k+1}^T A u_j &= v_{k+1}^T \Big(\beta_{j+1} v_{j+1} + \textstyle\sum_{i=1}^j (v_i^T A u_j) v_i \Big) = 0, \\ u_{k+1}^T B v_j &= u_{k+1}^T \Big(\gamma_{j+1} u_{j+1} + \textstyle\sum_{i=1}^j (u_i^T B v_j) u_i \Big) = 0, \end{aligned}$$

because $k + 1 > j + 1$. Thus, $V := \begin{bmatrix} v_1 & \dots & v_p \end{bmatrix}$, $U := \begin{bmatrix} u_1 & \dots & u_p \end{bmatrix}$,

$$H = \begin{bmatrix} v_1^T A u_1 & v_1^T A u_2 & \dots & v_1^T A u_p \\ \beta_2 & \ddots & \ddots & \vdots \\ & \ddots & \ddots & v_{p-1}^T A u_p \\ & & \beta_p & v_p^T A u_p \end{bmatrix} \text{ and } F = \begin{bmatrix} u_1^T B v_1 & u_1^T B v_2 & \dots & u_1^T B v_p \\ \gamma_2 & \ddots & \ddots & \vdots \\ & \ddots & \ddots & u_{p-1}^T B v_p \\ & & \gamma_p & u_p^T B v_p \end{bmatrix}$$

satisfy (6a)–(6b) and have the properties announced. □

Algorithm 1 formalizes a Hessenberg reduction process derived from Theorem 1.

Algorithm 1 Orthogonal Hessenberg reduction

Require: A, B, b, c, all nonzero

```
1: βv_1 = b, γu_1 = c                              (β, γ) > 0 so that ‖v_1‖ = ‖u_1‖ = 1
2: for k = 1, 2, ... do
3:   for i = 1, ..., k do
4:     h_{i,k} = v_i^T A u_k
5:     f_{i,k} = u_i^T B v_k
6:   end for
7:   h_{k+1,k} v_{k+1} = A u_k − Σ_{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 − Σ_{i=1}^k f_{i,k} u_i      f_{k+1,k} > 0 so that ‖u_{k+1}‖ = 1
9: end for
```

Define $V_k := \begin{bmatrix} v_1 & \dots & v_k \end{bmatrix}$ and $U_k := \begin{bmatrix} u_1 & \dots & u_k \end{bmatrix}$. After k iterations of Algorithm 1, the situation may be summarized as

$$A U_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T = V_{k+1} H_{k+1,k} \tag{9a}$$

$$B V_k = U_k F_k + f_{k+1,k} u_{k+1} e_k^T = U_{k+1} F_{k+1,k} \tag{9b}$$

$$V_k^T V_k = U_k^T U_k = I_k, \tag{9c}$$

where

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

and

$$H_{k+1,k} = \begin{bmatrix} H_k \\ h_{k+1,k} e_k^T \end{bmatrix}, \qquad F_{k+1,k} = \begin{bmatrix} F_k \\ f_{k+1,k} e_k^T \end{bmatrix}.$$

If $B = A^T$, Algorithm 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 1 uses the Gram-Schmidt method for computing ℓ_2-orthonormal bases V_k and U_k for simplicity. In a practical implementation, the modified Gram-Schmidt algorithm would be used instead. While (9a)–(9b) hold to within machine precision despite loss of orthogonality, (9c) holds only in exact arithmetic. In exact arithmetic, (9) yields

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

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 $k \ge 1$,

$$v_{2k} \in \text{Span}\{b, \dots, (AB)^{k-1} b, Ac, \dots, (AB)^{k-1} Ac\}, \tag{10a}$$

$$v_{2k+1} \in \text{Span}\{b, \dots, (AB)^{k} b \quad, Ac, \dots, (AB)^{k-1} Ac\}, \tag{10b}$$

$$u_{2k} \in \text{Span}\{c, \dots, (BA)^{k-1} c, Bb, \dots, (BA)^{k-1} Bb\}, \tag{10c}$$

$$u_{2k+1} \in \text{Span}\{c, \dots, (BA)^{k} c \quad, Bb, \dots, (BA)^{k-1} Bb\}. \tag{10d}$$

The subspaces generated by Algorithm 1 can be viewed as the union of two block-Krylov subspaces generated by AB and BA with respective starting blocks $\begin{bmatrix} b & Ac \end{bmatrix}$ and $\begin{bmatrix} c & Bb \end{bmatrix}$. Note the similarity between (14) and a Krylov process in which basis vectors have been permuted. Let

$$P_k := \begin{bmatrix} e_1 & e_{k+1} & \cdots & e_i & e_{k+i} & \cdots & e_k & e_{2k} \end{bmatrix} = \begin{bmatrix} E_1 & \cdots & E_k \end{bmatrix}, \qquad E_k := \begin{bmatrix} e_k & \\ & e_k \end{bmatrix}$$

denote the permutation introduced by Paige [21] that restores the order in which Algorithm 1 generates basis vectors, i.e.,

$$W_k := \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} P_k = \begin{bmatrix} w_1 & \cdots & w_k \end{bmatrix}, \qquad w_k = \begin{bmatrix} v_k & 0 \\ 0 & u_k \end{bmatrix} := \begin{bmatrix} v_k^\circ & u_k^\circ \end{bmatrix}, \tag{11}$$

where we defined $v_k^\circ := (v_k, 0)$ and $u_k^\circ := (0, u_k)$, and we abusively write $\begin{bmatrix} w_1 \cdots w_k \end{bmatrix}$ instead of $\begin{bmatrix} v_1^\circ \; u_1^\circ \cdots v_k^\circ \; u_k^\circ \end{bmatrix}$. The projection of K_0 into the block-Krylov subspace $\text{Span}\{w_1, \dots, w_k\} := \text{Span}\{v_1^\circ, u_1^\circ, \dots, v_k^\circ, u_k^\circ\}$ is also shuffled to block-Hessenberg form with blocks of size 2. Indeed, if we multiply (14) on the right with P_k and use (11), we obtain

$$K_0 W_k = \begin{bmatrix} V_{k+1} & 0 \\ 0 & U_{k+1} \end{bmatrix} P_{k+1} P_{k+1}^T \begin{bmatrix} 0 & H_{k+1,k} \\ F_{k+1,k} & 0 \end{bmatrix} P_k = W_{k+1} G_{k+1,k}, \tag{12}$$

where

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

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

$$\begin{bmatrix} v_1 & 0 \\ 0 & u_1 \end{bmatrix} \begin{bmatrix} \beta & 0 \\ 0 & \gamma \end{bmatrix} = \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} \quad \iff \quad w_1 \Gamma = D. \tag{13}$$

Identities (12) and (13) characterize the block-Arnoldi process applied to K_0 with initial block D. We summarize the process as Algorithm 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 Block-Arnoldi Process

Require: K_0, D
1: $w_1 \Gamma = D$
2: **for** $k = 1, 2, \dots$ **do**
3: **for** $i = 1, \dots, k$ **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 (9a)–(9b) gives

$$\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} = \begin{bmatrix} V_{k+1} & 0 \\ 0 & U_{k+1} \end{bmatrix} \begin{bmatrix} 0 & H_{k+1,k} \\ F_{k+1,k} & 0 \end{bmatrix}, \tag{14}$$

which is reminiscent of the relation one would obtain from applying an orthogonalization process to K_0. Because $K = K_0 + \operatorname{blkdiag}(\lambda I, \mu I)$, (14) yields

$$\begin{aligned} \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} &= \left(\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} + \begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix} \right) \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} \\ &= \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} \begin{bmatrix} \lambda I & H_k \\ F_k & \mu I \end{bmatrix} + \begin{bmatrix} v_{k+1} & 0 \\ 0 & u_{k+1} \end{bmatrix} \begin{bmatrix} 0 & h_{k+1,k} e_k^T \\ f_{k+1,k} e_k^T & 0 \end{bmatrix} \end{aligned} \tag{15}$$

The same reasoning applied to (12) yields the following result, which parallels Montoison and Orban [17, Theorem 2.1].

Theorem 2. Given the matrix K defined in (3) and the block right-hand side D defined in (5), the Krylov basis $W_k = \begin{bmatrix} w_1 & \cdots & w_k \end{bmatrix}$ generated by Algorithm 2 with regularization has the form (11) where the vectors u_k and v_k are the same as those generated by Algorithm 1 with initial vectors b and c. In addition,

$$K W_k = W_{k+1} S_{k+1,k}, \qquad S_{k+1,k} := \begin{bmatrix} \Theta_{1,1} & \Psi_{1,2} & \dots & \Psi_{1,k} \\ \Psi_{2,1} & \Theta_{2,2} & \ddots & \vdots \\ & \ddots & \ddots & \Psi_{k-1,k} \\ & & \ddots & \Theta_{k,k} \\ & & & \Psi_{k+1,k} \end{bmatrix}, \tag{16}$$

where

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

The scalars $h_{i,j}$, $f_{i,j}$ are those generated by Algorithm 1 applied to A and B with initial vectors b and c.

Proof. Algorithm 2 applied to K_0 generates sparse pairs w_k as in (11) because of the equivalence with Algorithm 1. The term $\mathrm{blkdiag}(\lambda I, \mu I)$ can be seen as a regularization term:

$$\begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix} w_k = w_k \Lambda \quad \text{with} \quad \Lambda := \begin{bmatrix} \lambda & 0 \\ 0 & \mu \end{bmatrix}. \tag{17}$$

The identities (12) and (17) allow us to write

$$KW_k = W_{k+1} \begin{bmatrix} \Psi_{1,1} + \Lambda & \Psi_{1,2} & \dots & \Psi_{1,k} \\ \Psi_{2,1} & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \Psi_{k-1,k} \\ & & \Psi_{k,k-1} & \Psi_{k,k} + \Lambda \\ & & & \Psi_{k+1,k} \end{bmatrix}, \tag{18}$$

which amounts to (16) because $\Theta_{k,k} = \Psi_{k,k} + \Lambda$. □

Note that (16) is identical to (15) where the order of the w_k has been permuted according to P_k.

Because of Theorem 2, the Krylov basis W_k generated by Algorithm 2 must have the sparsity structure (11), so that only u_k and v_k need be generated, and they may be generated directly from Algorithm 1. The key point is that generating orthonormal bases of $\mathcal{K}_k(K, d)$ and $\mathcal{K}_k(K, D)$ by the Arnoldi process and Algorithm 1, respectively, require exactly the same amount of storage and $\mathcal{K}_k(K, d) \subset \mathcal{K}_k(K, D)$. 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 $\mathcal{K}_k(K, D)$. 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 1 with regularization to solve (4) in which the k-th iterate has the form

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = W_k z_k, \tag{19}$$

where $z_k \in \mathbb{R}^{2k}$. Thanks to (13) and (16), the residual can be written

$$\begin{aligned} r_k &= \begin{bmatrix} b \\ c \end{bmatrix} - \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \\ &= w_1 \begin{bmatrix} \beta \\ \gamma \end{bmatrix} - W_{k+1} S_{k+1,k} z_k \\ &= W_{k+1}(\beta e_1 + \gamma e_2 - S_{k+1,k} z_k). \end{aligned} \tag{20}$$

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

$$\underset{z_k \in \mathbb{R}^{2k}}{\text{minimize}} \, \|S_{k+1,k} z_k - (\beta e_1 + \gamma e_2)\|. \tag{21}$$

3.1 Relation between Gpmr and Block-Gmres

The k-th BLOCK-GMRES iterate is defined by the matrix linear least-squares problem

$$\text{minimize} \left\| \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} - \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x_k^b & x_k^c \\ y_k^b & y_k^c \end{bmatrix} \right\|_F \tag{22}$$

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

$$\underset{z_k^b, z_k^c \in \mathbb{R}^{2k}}{\text{minimize}} \left\| S_{k+1,k} \begin{bmatrix} z_k^b & z_k^c \end{bmatrix} - \begin{bmatrix} \beta e_1 & \gamma e_2 \end{bmatrix} \right\|_F, \tag{23}$$

so that z_k^b and z_k^c solve the subproblem associated with right-hand sides βe_1 and γe_2. In exact arithmetic, the solutions of (21) and (23) are connected via $z_k = z_k^b + z_k^c$, and the Gpmr and Block-Gmres approximations are connected via $x_k = x_k^b + x_k^c$ and $y_k = y_k^b + y_k^c$. We now outline the main stages for solving (21).

3.2 A QR factorization

The solution of (21) can be determined via the QR factorization

$$S_{k+1,k} = Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \tag{24}$$

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 (24). We denote their product $Q_{2k-1,2k+2}$ so that $Q_k^T = Q_{2k-1,2k+2} \dots Q_{1,4}$. For $i = 1, \dots, k$, the structure of $Q_{2i-1,2i+2}$ is

$$
\begin{array}{cc}
 & \begin{array}{cccccccccc} 1 & \dots & 2i{-}2 & 2i{-}1 & 2i & 2i{+}1 & 2i{+}2 & 2i{+}3 & \dots & 2k{+}2 \end{array} \\
\begin{array}{c} 1 \\ \vdots \\ 2i{-}2 \\ 2i{-}1 \\ 2i \\ 2i{+}1 \\ 2i{+}2 \\ 2i{+}3 \\ \vdots \\ 2k{+}2 \end{array} &
\begin{bmatrix}
1 & & & & & & & & & \\
 & \ddots & & & & & & & & \\
 & & 1 & & & & & & & \\
 & & & \times & \times & \times & \times & & & \\
 & & & \times & \times & \times & \times & & & \\
 & & & \times & \times & \times & \times & & & \\
 & & & \times & \times & \times & \times & & & \\
 & & & & & & & 1 & & \\
 & & & & & & & & \ddots & \\
 & & & & & & & & & 1
\end{bmatrix}
\end{array}
$$

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

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

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 4×4 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.

Algorithm 3 Procedure ref

Require: i, a_1^{in}, a_2^{in}, a_3^{in}, a_4^{in}

$t = c_{1,i} a_1^{\text{in}} + s_{1,i} a_4^{\text{in}}$, $a_4^{\text{out}} = s_{1,i} a_1^{\text{in}} - c_{1,i} a_4^{\text{in}}$, $a_1^{\text{out}} = t$ *first reflection*

$t = c_{2,i} a_1^{\text{out}} + s_{2,i} a_2^{\text{in}}$, $a_2^{\text{out}} = s_{2,i} a_1^{\text{out}} - c_{2,i} a_2^{\text{in}}$, $a_1^{\text{out}} = t$ *second reflection*

$t = c_{3,i} a_2^{\text{out}} + s_{3,i} a_4^{\text{out}}$, $a_4^{\text{out}} = s_{3,i} a_2^{\text{out}} - c_{3,i} a_4^{\text{out}}$, $a_2^{\text{out}} = t$ *third reflection*

$t = c_{4,i} a_2^{\text{out}} + s_{4,i} a_3^{\text{in}}$, $a_3^{\text{out}} = s_{4,i} a_2^{\text{out}} - c_{4,i} a_3^{\text{in}}$, $a_2^{\text{out}} = t$ *fourth reflection*

At iteration k, Algorithm 1 generates two new columns, and to update the QR decomposition we need first to apply all previous reflections as follows

$$Q_{k-1}^T \begin{bmatrix} \Psi_{1,k} \\ \vdots \\ \Psi_{k-1,k} \\ \Theta_{k,k} \\ \Psi_{k+1,k} \end{bmatrix} = Q_{2k-5,2k-2} \dots Q_{3,6} \begin{bmatrix} r_{1,2k-1} & r_{1,2k} \\ r_{2,2k-1} & r_{2,2k} \\ \bar{r}_{3,2k-1} & \bar{r}_{3,2k} \\ \bar{r}_{4,2k-1} & \bar{r}_{4,2k} \\ \multicolumn{2}{c}{\Psi_{3,k}} \\ \multicolumn{2}{c}{\vdots} \\ \multicolumn{2}{c}{\Psi_{k+1,k}} \end{bmatrix} = \begin{bmatrix} r_{1,2k-1} & r_{1,2k} \\ \vdots & \vdots \\ r_{2k-2,2k-1} & r_{2k-2,2k} \\ \bar{r}_{2k-1,2k-1} & \bar{r}_{2k-1,2k} \\ \bar{r}_{2k,2k-1} & \bar{r}_{2k,2k} \\ & h_{k+1,k} \\ f_{k+1,k} & \end{bmatrix},$$

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

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

A procedure to compute the Givens sines and cosines, and finalize the QR factorization of $S_{k+1,k}$ is described as Algorithm 4. Note that the first parameter of Algorithm 3 and Algorithm 4 is used to define which Givens sines and cosines are read from or written to memory.

Algorithm 4 Procedure givens

Require: k, $\bar{r}_{2k-1,2k-1}$, $\bar{r}_{2k-1,2k}$, $\bar{r}_{2k,2k-1}$, $\bar{r}_{2k,2k}$, $h_{k+1,k}$, $f_{k+1,k}$

$\bar{\bar{r}}_{2k-1,2k-1} = (\bar{r}^2_{2k-1,2k-1} + f^2_{k+1,k})^{\frac{1}{2}}$ *annihilate* $f_{k+1,k}$
$c_{1,k} = \bar{r}_{2k-1,2k-1}/\bar{\bar{r}}_{2k-1,2k-1}$, $s_{1,k} = f_{k+1,k}/\bar{\bar{r}}_{2k-1,2k-1}$
$\bar{\bar{r}}_{2k-1,2k} = c_{1,k}\bar{r}_{2k-1,2k}$
$\bar{r}_{2k+2,2k} = s_{1,k}\bar{r}_{2k-1,2k}$
$r_{2k-1,2k-1} = (\bar{\bar{r}}^2_{2k-1,2k-1} + \bar{r}^2_{2k,2k-1})^{\frac{1}{2}}$ *annihilate* $\bar{r}_{2k,2k-1}$
$c_{2,k} = \bar{\bar{r}}_{2k-1,2k-1}/r_{2k-1,2k-1}$, $s_{2,k} = \bar{r}_{2k,2k-1}/r_{2k-1,2k-1}$
$r_{2k-1,2k} = c_{2,k}\bar{\bar{r}}_{2k-1,2k} + s_{2,k}\bar{r}_{2k,2k}$
$\bar{\bar{r}}_{2k,2k} = s_{2,k}\bar{\bar{r}}_{2k-1,2k} - c_{2,k}\bar{r}_{2k,2k}$
$\mathring{r}_{2k,2k} = (\bar{\bar{r}}^2_{2k,2k} + \bar{r}^2_{2k+2,2k})^{\frac{1}{2}}$ *annihilate* $\bar{r}_{2k+2,2k}$
$c_{3,k} = \bar{\bar{r}}_{2k,2k}/\mathring{r}_{2k,2k}$, $s_{3,k} = \bar{r}_{2k+2,2k}/\mathring{r}_{2k,2k}$
$r_{2k,2k} = (\mathring{r}^2_{2k,2k} + h^2_{k+1,k})^{\frac{1}{2}}$ *annihilate* $h_{k+1,k}$
$c_{4,k} = \mathring{r}_{2k,2k}/r_{2k,2k}$, $s_{4,k} = h_{k+1,k}/r_{2k,2k}$

3.3 Gpmr iterate and residual norm computation

We have from (20) and (24):

$$\|r_k\| = \left\| Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix} z_k - (\beta e_1 + \gamma e_2) \right\| = \left\| \begin{bmatrix} R_k \\ 0 \end{bmatrix} z_k - \bar{t}_k \right\|, \tag{25}$$

where $\bar{t}_k := Q_k^T(\beta e_1 + \gamma e_2) = (t_k, \bar{\tau}_{2k+1}, \bar{\tau}_{2k+2})$, $t_k := (\tau_1, \dots, \tau_{2k})$ represents the first $2k$ 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 (21) is thus $z_k := (\zeta_1, \dots, \zeta_{2k})$ found by solving $R_k z_k = t_k$ with backward substitution.

The definitions of $\bar{t}_k$ and z_k together with (25) yield

$$\|r_k\| = \sqrt{\bar{\tau}^2_{2k+1} + \bar{\tau}^2_{2k+2}}. \tag{26}$$

As in Gmres, we only compute z_k when $\|r_k\|$ is smaller than a user-provided threshold. Thanks to (19), the solution may be computed efficiently as

$$x_k = \textstyle\sum_{i=1}^{k} \zeta_{2i-1} v_i, \tag{27a}$$

$$y_k = \textstyle\sum_{i=1}^{k} \zeta_{2i} u_i. \tag{27b}$$

We summarize the complete procedure as Algorithm 5.

Algorithm 5 Gpmr

Require: A, B, b, c, λ, μ, $\epsilon > 0$, $k_{\max} > 0$

$\beta v_1 = b$, $\gamma u_1 = c$ *$(\beta, \gamma) > 0$ so that $\|v_1\| = \|u_1\| = 1$*
$\bar{\tau}_1 = \beta$, $\bar{\tau}_2 = \gamma$ *Initialize $\bar{t}_0$*
$\|r_0\| = (\bar{\tau}_1^2 + \bar{\tau}_2^2)^{\frac{1}{2}}$ *compute $\|r_0\|$*
$k = 0$
while $\|r_k\| > \epsilon$ **and** $k < k_{\max}$ **do**
  $k \leftarrow k + 1$
  $q = Au_k$ *Orthogonal Hessenberg reduction*
  $p = Bv_k$
  for $i = 1, \ldots, k$ **do**
    $h_{i,k} = v_i^T q$
    $f_{i,k} = u_i^T p$
    $q = q - h_{i,k} v_i$
    $p = p - f_{i,k} u_i$
  end for
  $h_{k+1,k} v_{k+1} = q$ *$h_{k+1,k} > 0$ so that $\|v_{k+1}\| = 1$*
  $f_{k+1,k} u_{k+1} = p$ *$f_{k+1,k} > 0$ so that $\|u_{k+1}\| = 1$*
  $\bar{r}_{1,2k} = h_{1,k}$, $\bar{r}_{2,2k-1} = f_{1,k}$
  if $k \neq 1$ **then** $(\bar{r}_{1,2k-1}, \bar{r}_{2,2k}) = (0, 0)$ **else** $(\bar{r}_{1,2k-1}, \bar{r}_{2,2k}) = (\lambda, \mu)$
  for $i = 1, \ldots, k-1$ **do** *Apply $Q_{2k-5,2k-2}, \ldots, Q_{1,4}$*
    if $i \neq k-1$ **then** $(\rho, \delta) = (0, 0)$ **else** $(\rho, \delta) = (\lambda, \mu)$
    $r_{2i-1,2k-1}, r_{2i,2k-1}, \bar{r}_{2i+1,2k-1}, \bar{r}_{2i+2,2k-1} = \mathrm{ref}(i, \bar{r}_{2i-1,2k-1}, \bar{r}_{2i,2k-1}, \rho, f_{i+1,k})$
    $r_{2i-1,2k}, r_{2i,2k}, \bar{r}_{2i+1,2k}, \bar{r}_{2i+2,2k} = \mathrm{ref}(i, \bar{r}_{2i-1,2k}, \bar{r}_{2i,2k}, h_{i+1,k}, \delta)$
  end for
  $r_{2k-1,2k-1}, r_{2k-1,2k}, r_{2k,2k} =$ *Compute and apply $Q_{2k-1,2k+2}$*
    $\mathrm{givens}(k, \bar{r}_{2k-1,2k-1}, \bar{r}_{2k-1,2k}, \bar{r}_{2k,2k-1}, \bar{r}_{2k,2k}, h_{k+1,k}, f_{k+1,k})$
  $\tau_{2k-1}, \tau_{2k}, \bar{\tau}_{2k+1}, \bar{\tau}_{2k+2} = \mathrm{ref}(k, \bar{\tau}_{2k-1}, \bar{\tau}_{2k}, 0, 0)$ *update $\bar{t}_k$*
  $\|r_k\| = (\bar{\tau}_{2k+1}^2 + \bar{\tau}_{2k+2}^2)^{\frac{1}{2}}$ *compute $\|r_k\|$*
end while
$\zeta_{2k} = \tau_{2k} / r_{2k,2k}$ *compute z_k*
for $i = 2k-1, \ldots, 1$ **do**
  $\zeta_i = (\tau_i - \sum_{j=i+1}^{2k} r_{i,j} \zeta_j) / r_{i,i}$
end for
$x_k = \sum_{i=1}^{k} \zeta_{2i-1} v_i$ *compute x_k*
$y_k = \sum_{i=1}^{k} \zeta_{2i} u_i$ *compute y_k*

3.4 Memory requirements

Table 1 summarizes the storage costs of k iterations of Gpmr, Gmres and Block-Gmres.

Table 1: Memory requirements for k iterations of Gpmr, Gmres and Block-Gmres.

	(x_k, y_k)	(q, p)	(V_k, U_k)	t_k	z_k	Q_k	R_k
Gpmr	$m+n$	$m+n$	$k(m+n)$	$2k$	$2k$	$8k$	$k(2k+1)$
Gmres	$m+n$	$m+n$	$k(m+n)$	k	k	$2k$	$k(k+1)/2$
Block-Gmres	$2(m+n)$	$2(m+n)$	$2k(m+n)$	$4k$	$4k$	$8k$	$k(2k+1)$

Some Gpmr variables are paired in Table 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 5 in Julia [3], version 1.6, as part of our `Krylov.jl` collection of Krylov methods [18]. Our implementation of GPMR is applicable in any floating-point system supported by Julia, and runs on CPU and GPU. The GPU support can be particularly relevant for (2) because, as a Krylov method, GPMR only requires linear operators that model $A_{\mathcal{I}\Gamma}u$, $B_{\Gamma\mathcal{I}}v$, $u \mapsto M_{\mathcal{I}\mathcal{I}}\backslash u$ and $v \mapsto N_{\Gamma\Gamma}\backslash v$. For instance, $v \mapsto N_{\Gamma\Gamma}\backslash v$ can be the forward and backward substitutions with the factors of an LU decomposition of $N_{\Gamma\Gamma}$. The use of abstract linear operators allows us to store $A_{\mathcal{I}\Gamma}$ and $B_{\Gamma\mathcal{I}}$ as well as decompositions of the diagonal blocks of (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 (2).

We evaluate the performance of GPMR on systems generated from unsymmetric matrices in the SuiteSparse Matrix Collection [5]. We use METIS to form a 2×2 block matrix and use the two diagonal blocks to build a right block-Jacobi preconditioner P_r with $\lambda = \mu = 1$. We set $P_\ell = I$ so the residual norm of (1) is identical to that of (4). The right-hand side $(b_\star, c_\star)$ is generated so the exact solution of (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 $\|r_k\| \le \varepsilon_a + \|(b,c)\|\varepsilon_r$ with absolute tolerance $\varepsilon_a = 10^{-12}$ and relative tolerance $\varepsilon_r = 10^{-10}$. Table 2 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 1 reports residual histories of GPMR, GMRES and BLOCK-GMRES where the two approximate solutions are summed on problems *scircuit*, *sme3Dc*, *PR02R* and *sherman5*.

Table 2: Number of iterations of Gpmr and Gmres on systems from the SuiteSparse Matrix Collection.

name	size	nnz	GMRES	GPMR	gain
sherman5	3312	20793	25	20	20%
powersim	15838	67562	141	101	28%
Ill_Stokes	20896	191368	59	54	9%
sme3Dc	42930	3148656	127	78	39%
rma10	46835	2374001	48	41	15%
ecl32	51993	380415	58	42	28%
venkat50	62424	1717792	48	35	27%
poisson3Db	85623	2374949	56	50	11%
ifiss_mat	96307	3599932	42	33	21%
hcircuit	105676	513072	47	37	21%
PR02R	161070	8185136	97	68	30%
scircuit	170998	958936	48	24	50%
transient	178866	961790	567	470	17%
ohne2	181343	11063545	50	39	22%
thermomech_dK	204316	2846228	128	84	34%
marine1	400320	6226538	84	60	29%
Freescale1	3428755	18920347	456	344	25%

The GPMR and BLOCK-GMRES residuals are nearly superposed except for *scircuit*, on which BLOCK-GMRES stagnates. The same phenomenon occurs on a generalized saddle point build using matrices *well1033* as A and *illc1033* as B, $M = I$, $N = 0$, $\lambda = 1$ and $\mu = 0$. Figure 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 double and quadruple precision.

When K, defined in (3), is symmetric, Algorithm 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 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.

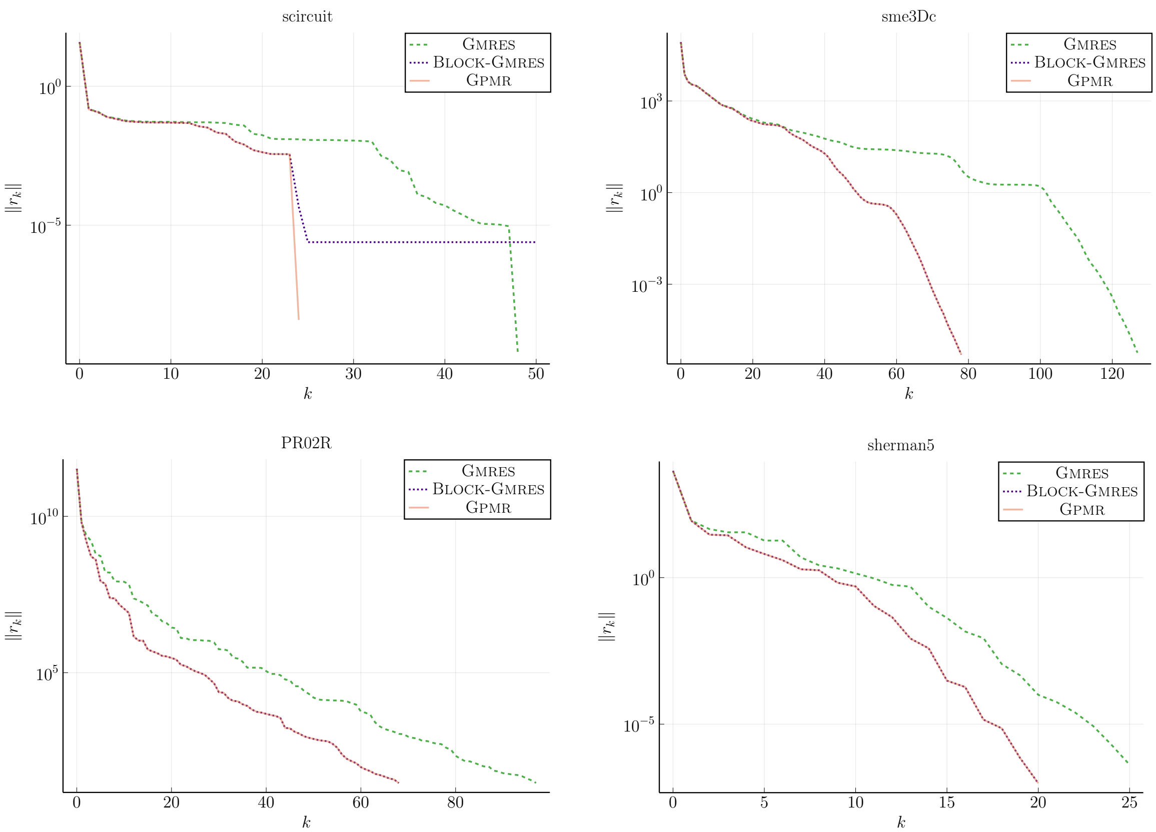

Figure 1: Residual history of Gpmr, Gmres and Block-Gmres.

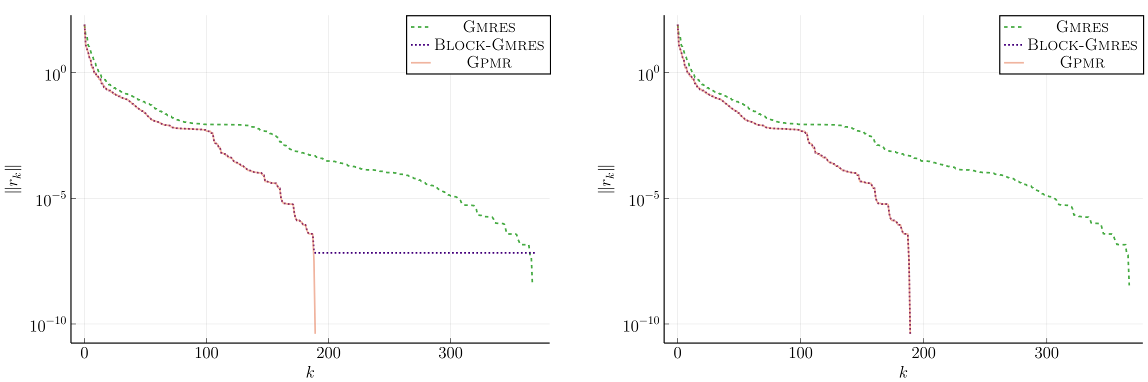

Figure 2: Residual history of Gpmr, Gmres and Block-Gmres on the generalized saddle point system in double (left) and quadruple precision (right).

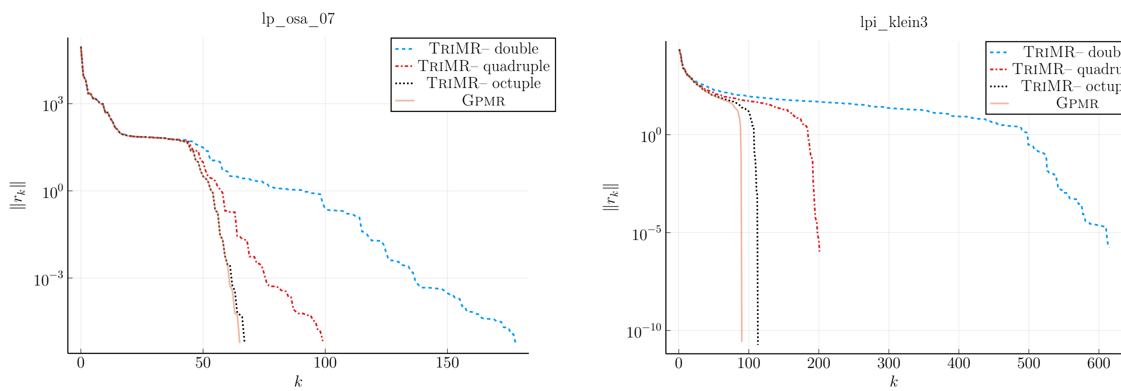

Figure 3: Residual history of Gpmr and TriMR.

5 Discussion and extensions

Based upon Algorithm 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 (19) as the solution of the square system

$$S_k z_k = \beta e_1 + \gamma e_2, \tag{28}$$

where S_k denotes the leading $(2k)\times(2k)$ submatrix of $S_{k+1,k}$ in (16). However, Gpcg may break down if S_k is singular, and in that respect shares the disadvantages of 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 (28) 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 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. DOI: 10.1090/qam/42792.

[2] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. Acta Numerica, 14: 1–137, 2005. DOI: 10.1017/S0962492904000212.

[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. DOI: 10.1137/141000671.

[4] A. Buttari, D. Orban, D. Ruiz, and D. Titley-Peloquin. USYMLQR: A tridiagonalization method for symmetric saddle-point systems. SIAM J. Sci. Comput., 41(5):409–432, 2019. DOI: 10.1137/18M1194900.

[5] T. Davis and Y. Hu. The University of Florida sparse matrix collection. ACM Trans. Math. Software, 38(1):1–25, 2011. DOI: 10.1145/2049662.2049663.

[6] V. Dolean, P. Jolivet, and F. Nataf. An introduction to domain decomposition methods: algorithms, theory, and parallel implementation. SIAM, 2015. DOI: 10.1137/1.9781611974065.

[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. DOI: 10.1016/j.parco.2004.12.008.

[8] H. C. Elman. Preconditioners for saddle point problems arising in computational fluid dynamics. Applied Numerical Mathematics, 43(1–2):75–89, 2002. DOI: 10.1016/S0168-9274(02)00118-6.

[9] R. Estrin and C. Greif. SPMR: A family of saddle-point minimum residual solvers. SIAM J. Sci. Comput., 40(3):1884–1914, 2018. DOI: 10.1137/16M1102410.

[10] M. C. Ferris and J. D. Horn. Partitioning mathematical programs for parallel solution. Mathematical Programming, 80(1):35–61, 1998. DOI: 10.1007/BF01582130.

[11] R. Fletcher. Conjugate gradient methods for indefinite systems. In Numerical Analysis, pages 73–89. Springer, 1976. DOI: 10.1007/BFb0080116.

[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. DOI: 10.1007/BF01385726.

[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. DOI: https://doi.org/10.1016/j.cam.2019.02.016.

[14] I. C. Ipsen. A note on preconditioning nonsymmetric matrices. SIAM J. Sci. Comput., 23(3):1050–1051, 2001. DOI: 10.1137/S1064827500377435.

[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. DOI: 10.1137/19M1290991.

[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. DOI: 10.1137/20M1363030.

[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. DOI: 10.1137/S1064827599355153.

[20] D. Orban and M. Arioli. Iterative Solution of Symmetric Quasi-Definite Linear Systems, volume 3 of Spotlights. SIAM, 2017. DOI: 10.1137/1.9781611974737.

[21] C. C. Paige. Bidiagonalization of matrices and solution of linear equations. SIAM J. Numer. Anal., 11(1): 197–209, 1974. DOI: 10.1137/0711019.

[22] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. Mathematics of computation, 37(155):105–126, 1981. DOI: 10.1090/S0025-5718-1981-0616364-6.

[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. DOI: 10.1137/0907058.

[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. DOI: 10.1002/(sici)1099-1506(199607/08)3:4¡329::aid-nla86¿3.0.co;2-8.

[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. DOI: 10.1137/0725052.

[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. DOI: 10.1007/978-3-642-02677-5˙7.