

Titre: Méthodes de Krylov pour l'algèbre linéaire et implémentation
Title: polymorphe

Auteur: Alexis Montoison
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Montoison, A. (2023). Méthodes de Krylov pour l'algèbre linéaire et
Citation: implémentation polymorphe [Thèse de doctorat, Polytechnique Montréal].
PolyPublie. <https://publications.polymtl.ca/57098/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/57098/>
PolyPublie URL:

**Directeurs de
recherche:** Dominique Orban
Advisors:

Programme: Doctorat en mathématiques
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Méthodes de Krylov pour l'algèbre linéaire et implémentation polymorphe

ALEXIS MONTTOISON

Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*

Mathématiques

Décembre 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

Méthodes de Krylov pour l'algèbre linéaire et implémentation polymorphe

présentée par **Alexis MONTISON**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*

a été dûment acceptée par le jury d'examen constitué de :

Serge PRUDHOMME, président

Dominique ORBAN, membre et directeur de recherche

Youssef DIOUANE, membre

David TITLEY-PELOQUIN, membre

Xiao-Wen CHANG, membre

Daniel RUIZ, membre externe

DÉDICACE

*Je dédie cette thèse à ma mère,
décédée trop tôt pour voir ce que son fils est devenu.
Ce que nous sommes est le cadeau que la Vie nous à fait.
Ce que nous devenons est le cadeau que nous faisons à la Vie.*

REMERCIEMENTS

Je tiens tout d’abord à remercier mon directeur de recherche, Dominique Orban, d’avoir pris part à cette aventure avec moi. Nous avons traversé bien des tempêtes ensemble... Mais nous avons toujours su nous relever et trouver la force de continuer. Je suis reconnaissant pour la qualité de nos échanges, tant sur le plan professionnel que personnel. Bien que ce document marque la fin de ce périple à ses côtés, je suis convaincu que notre relation ne s’arrête pas là. J’espère sincèrement que nos chemins continueront de se croiser. Merci, Dominique, pour tout ce que vous avez fait. Votre soutien inébranlable et votre mentorat ont fait de cette expérience une période mémorable de ma vie. Je suis honoré de vous avoir comme directeur de thèse.

Je tiens également à exprimer ma gratitude envers une personne exceptionnelle, Mériem Belkahla. Elle a su m’épauler et m’aider à me retrouver lorsque j’étais perdu dans les multiples projets qui composent cette thèse. Je la remercie pour son amour et son incroyable capacité à me soutenir dans les moments cruciaux.

Je remercie maintenant les professeurs Serge Prudhomme, Youssef Diouane, David Titley-Peloquin, Xiao-Wen Chang et Daniel Ruiz d’avoir accepté d’être membres de mon jury.

Ensuite, je tiens à exprimer ma gratitude envers la fondation Arbour, le fonds de recherche du Québec - Nature et technologies (FRQNT), l’institut de valorisation des données (IVADO), ainsi que le groupe d’études et de recherche en analyse des décisions (GERAD) pour leur soutien financier qui a permis de réaliser ce projet.

Je n’oublie pas mes collègues du GERAD et nos nombreuses discussions autour d’un café. En particulier, je tiens à mentionner Pierre-Yves Bouchet, qui a partagé ce chemin de croix avec moi depuis le début.

Une pensée spéciale pour Michael Saunders, dont les connaissances sont aussi exceptionnelles que sa gentillesse. Mes séjours en Californie et au Japon en sa compagnie resteront inoubliables !

Un immense merci à l’équipe STFC Scientific Computing du Rutherford Appleton Laboratory de m’avoir accueilli en décembre 2022 et à l’été 2023. Iain Duff et Nick Gould, j’admire votre passion pour coder dans ce langage “moderne” qu’est le Fortran. Merci à Andrew Lister d’avoir partagé quelques verres de mon thé préféré, la tequila, à la *St Aldates Tavern* d’Oxford.

Merci également à l’équipe de recherche de Mihai Anitescu du laboratoire national d’Argonne

avec qui j'ai pu passer les derniers mois de mon doctorat à Chicago. Nos discussions autour d'une pinte les mardis à la *Miskatonic Brewing Company* vont me manquer...

Enfn, je souhaite exprimer ma gratitude envers mon père, Olivier Montoisson, pour son amour inconditionnel et son soutien tout au long de ma vie.

RÉSUMÉ

Ce projet de recherche se situe à la frontière entre les mathématiques appliquées et l’informatique théorique. Il dépend du domaine de l’algèbre linéaire numérique et se focalise sur le développement d’algorithmes de résolution de systèmes linéaires.

Dans notre cadre d’étude, les algorithmes de résolution sont itératifs, c’est-à-dire qu’ils génèrent une suite de solutions approximatives. Ces itérés sont construits de telle sorte qu’ils résident dans un *espace de Krylov*. Un espace de Krylov est généré par le produit répété d’une matrice à un vecteur initial. Un exemple d’espace de Krylov est l’ensemble des combinaisons linéaires de b , Ab et A^2b . Les algorithmes ayant cette propriété sont considérés comme des méthodes de Krylov.

Les recherches résumées dans ce document se décomposent en deux composantes. La première est théorique et englobe le développement d’un nouveau processus de Krylov appelé ORTHOGONAL HESSENBERG REDUCTION ainsi que huit nouvelles méthodes de Krylov (BiLQ, BiLQR, TriLQR, TriCG, TrimR, GPMR, MINARES, CAR), mieux adaptées aux caractéristiques de certains systèmes linéaires. La seconde, axée sur l’aspect logiciel, concerne l’implémentation générique et modulaire de ces nouveaux algorithmes. Ces implémentations doivent s’adapter à divers domaines d’applications, incluant différents modes de stockage et de modélisation sur ordinateur. Un système linéaire peut être stocké avec différentes précisions machine, telles que simple, double ou quadruple précision, être résolu sur CPU ou GPU, ou encore être représenté dans différents formats de matrices, qu’elles soient creuses ou denses. Ces aspects soulignent la nécessité d’une implémentation polymorphe pour s’adapter à ces multiples configurations. Une implémentation polymorphe se réfère à la capacité d’un code informatique à traiter des paramètres de manière flexible, en leur permettant d’adopter différentes formes ou types en fonction du contexte.

Ces travaux sont rassemblés dans la bibliothèque numérique Krylov.jl, implémentée dans le langage de programmation Julia. Ce langage de programmation est adéquat au projet en raison de sa fonctionnalité de dispatch multiple ; un mécanisme informatique permettant d’écrire une fonction une seule fois, et voir celle-ci compilée à la volée pour différentes configurations suivant le type des arguments.

Krylov.jl s’est illustré pour résoudre des systèmes linéaires creux sur les deux supercalculateurs les plus puissants du monde, *Frontier* du laboratoire national d’Oak Ridge et *Aurora* du laboratoire national d’Argonne. Il offre la portabilité nécessaire aux nouvelles architectures des cartes graphiques AMD et Intel, respectivement présentes dans *Frontier* et *Aurora*.

ABSTRACT

This research project lies at the intersection of applied mathematics and computer science. It is based on numerical linear algebra and focuses on the development of algorithms for solving linear systems.

In our framework, the algorithms are iterative, meaning that they generate a sequence of approximate solutions. These iterates are built such that they lie in a *Krylov subspace*. A Krylov subspace is generated by the repeated product of a matrix to an initial vector. An example of a Krylov space is the set of linear combinations of b , Ab , and A^2b . The algorithms with this property are Krylov methods.

The research summarized in this document consists of two components. The first is theoretical and encompasses the development of a new Krylov process called **ORTHOGONAL HESSENBERG REDUCTION** as well as eight new Krylov methods (**BiLQ**, **BiLQR**, **TriLQR**, **TriCG**, **TriMR**, **GPMR**, **MINARES**, **CAR**), better suited to the characteristics of certain linear systems. The second, focusing on the software aspect, concerns the generic and modular implementation of these new algorithms. These implementations must adapt to various application domains, including different modeling and computer storage. A linear system may be stored with different machine precisions, such as single, double, or quadruple precision, solved on CPU or GPU, or represented in different matrix formats, whether sparse or dense. These aspects highlight the need for a polymorphic implementation to adapt to these multiple configurations. A polymorphic implementation refers to the ability of a computer code to handle parameters flexibly, allowing them to take on different forms or types depending on the context.

These works are consolidated in the numerical library *Krylov.jl*, implemented in the Julia programming language. This programming language is suitable for the project due to its multiple dispatch feature; a mechanism that allows writing a function only once and having it compiled on-the-fly for different configurations depending on the types of the arguments.

Krylov.jl has showcased its capabilities in solving sparse linear systems on world's two most powerful supercomputers, *Frontier* at the Oak Ridge National Laboratory and *Aurora* at the Argonne National Laboratory. It provides portability to the new architectures of AMD and Intel graphics cards, present respectively in *Frontier* and *Aurora*.

TABLE DES MATIÈRES

DÉDICACE	iii
REMERCIEMENTS	iv
RÉSUMÉ	vi
ABSTRACT	vii
TABLE DES MATIÈRES	viii
LISTE DES TABLEAUX	xii
LISTE DES FIGURES	xiii
LISTE DES SIGLES ET ABRÉVIATIONS	xvi
CHAPITRE 1 INTRODUCTION	1
1.1 Objectifs de recherche	1
CHAPITRE 2 REVUE DE LITTÉRATURE	2
2.1 Méthodes de Krylov	2
2.1.1 Origine des méthodes de Krylov	2
2.1.2 Convergence des méthodes de Krylov	2
2.1.3 Préconditionneurs	3
2.1.4 Processus de Krylov	3
2.1.5 Synthèse des processus et méthodes de Krylov existantes	4
2.2 Programmation polymorphe	5
CHAPITRE 3 ORGANISATION DE LA THÈSE	7
CHAPITRE 4 ARTICLE 1: BiLQ: AN ITERATIVE METHOD FOR NONSYMMETRIC LINEAR SYSTEMS WITH A QUASI-MINIMUM ERROR PROPERTY	8
4.1 Introduction	8
4.2 Derivation of BiLQ	11
4.2.1 The Lanczos biorthogonalization process	11
4.2.2 Definition of BiLQ	12

4.2.3	An LQ factorization	13
4.2.4	Definition and update of the BiLQ and BICG iterates	14
4.2.5	Residual estimates	15
4.2.6	Properties	16
4.2.7	Numerical experiments	20
4.2.8	Discussion	23
4.3	Adjoint systems	23
4.3.1	Description of BiLQR	25
4.3.2	Description of TRILQR	26
4.3.3	Applications	30
4.4	Discussion	33
CHAPITRE 5 ARTICLE 2: TRICG AND TRIMR: TWO ITERATIVE METHODS		
	FOR SYMMETRIC QUASI-DEFINITE SYSTEMS	36
5.1	Introduction	36
5.2	Processes	39
5.2.1	The orthogonal tridiagonalization process in elliptic norms	39
5.2.2	Relation with preconditioned block-Lanczos process	40
5.2.3	Regularization of the preconditioned block-Lanczos process	42
5.3	Methods	43
5.3.1	Derivation of TRICG	44
5.3.2	Derivation of TRIMR	49
5.4	Implementation and numerical experiments	55
5.5	Discussion and extensions	63
CHAPITRE 6 ARTICLE 3: GPMR: AN ITERATIVE METHOD FOR UNSYM-		
	METRIC PARTITIONED LINEAR SYSTEMS	65
6.1	Introduction	65
6.2	A Hessenberg reduction process	68
6.2.1	Relation with the block-Arnoldi process	71
6.2.2	Regularization of the block-Arnoldi process	72
6.3	Derivation of GPMR	74
6.3.1	Relation between GPMR and BLOCK-GMRES	74
6.3.2	A QR factorization	75
6.3.3	GPMR iterate and residual norm computation	77
6.3.4	Memory and computational requirements	79
6.4	Implementation and numerical experiments	79

6.5	Discussion and extensions	84
CHAPITRE 7 ARTICLE 4: MINARES: AN ITERATIVE SOLVER FOR SYM-		
	METRIC LINEAR SYSTEMS	87
7.1	Introduction	87
7.1.1	Notation	89
7.2	Applications	89
7.2.1	Null vector, eigenvector, and singular value problems	89
7.2.2	Singular systems with semi-positive definite matrices	89
7.3	Symmetric systems	90
7.3.1	CG, SYMMLQ, MINRES, MINARES	91
7.4	Derivation of MINARES	91
7.4.1	Subproblems of MINARES	91
7.4.2	QR factorization of T_k	93
7.4.3	Definition of N_k	94
7.4.4	QR factorization of N_k	95
7.4.5	Computation of x_k	97
7.5	Stopping rules	98
7.5.1	Estimating $\ r_k\ $	98
7.5.2	Estimating $\ Ar_k\ $	103
7.6	CAR	103
7.7	Implementation and numerical experiments	108
7.8	Summary	110
CHAPITRE 8 ARTICLE 5: KRYLOV.JL: A JULIA BASKET OF HAND-PICKED		
	KRYLOV METHODS	111
8.1	Summary	111
8.2	Statement of need	112
8.2.1	Largest collection of Krylov processes and methods	112
8.2.2	Support for any floating-point system supported by Julia	112
8.2.3	Support for Nvidia, AMD and Intel GPUs	113
8.2.4	Support for linear operators	113
8.2.5	In-place methods	113
8.2.6	Performance optimizations and storage requirements	113
8.2.7	Examples	114
CHAPITRE 9 DISCUSSION GÉNÉRALE		
		118

9.1 Synthèse des travaux	118
9.2 Limitations de la solution proposée	119
9.3 Améliorations futures	119
CHAPITRE 10 CONCLUSION ET RECOMMANDATIONS	121
RÉFÉRENCES	122

LISTE DES TABLEAUX

Tableau 2.1	Type de préconditionnement avec P_ℓ et P_r des matrices carrées inversibles.	3
Tableau 2.2	Synthèse des méthodes de Krylov existantes	5
Table 4.1	Storage and cost per iteration of methods based on the Lanczos Biorthogonalization Process.	16
Table 4.2	Storage and cost per iteration of methods for solving (4.1) and (4.29) simultaneously.	28
Table 6.1	Memory requirements for k iterations of GPMR, GMRES and BLOCK-GMRES.	79
Table 6.2	Number of iterations of GPMR and GMRES on systems from the SuiteSparse Matrix Collection.	81
Table 6.3	Number of iterations of GPMR(ℓ) and GMRES(ℓ) on systems from the SuiteSparse Matrix Collection.	83
Tableau 9.1	Synthèse des nouvelles méthodes de Krylov.	118

LISTE DES FIGURES

Figure 4.1	Solution $u(r, \theta) = r(1 - r)\cos(\theta)$ of (4.27) with $g(\theta) = 0$, $f(r, \theta) = -3\cos(\theta)$ and $R = 1$	21
Figure 4.2	Convergence curves of BiLQ, BiCG and QMR iterates on (4.27). The figures show the residual (left) and error (right) history for each method.	21
Figure 4.3	Convergence curves of BiLQ, BiCG and QMR iterates for the SHERMAN5 system. The figures show the residual (left) and error (right) history for each method.	22
Figure 4.4	Convergence curves of BiLQ, BiCG and QMR iterates for the RAEFSKY1 system. The figures show the residual (left) and error (right) history for each method.	22
Figure 4.5	Residuals and errors norms of BiLQR, TriLQR and MINRES-QLP iterates on (4.41).	31
Figure 4.6	Residuals and errors norms of BiLQR, TriLQR and MINRES-QLP iterates on (4.43).	32
Figure 4.7	Functional evaluation errors for (4.41)–(4.43) (left) and (4.45)–(4.46) (right).	32
Figure 4.8	Residuals and errors norms of BiLQR, TriLQR and MINRES-QLP iterates for on (4.45).	34
Figure 4.9	Residuals and errors norms of BiLQR, TriLQR and MINRES-QLP iterates on (4.46).	34
Figure 5.1	$\mathcal{E}(x, y)$ history of TRICG on illc1850, where A is 1850×712 with 8636 nonzeros.	49
Figure 5.2	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on lp_czprob, where A is 929×3562 with 10708 nonzeros.	57
Figure 5.3	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on lp_osa_07, where A is 1108×25067 with 144812 nonzeros.	57
Figure 5.4	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on Maragal_6, where A is 21255×10152 with 537694 nonzeros.	57

Figure 5.5	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on landmark, where A is 71952×2704 with 1146848 nonzeros.	58
Figure 5.6	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR iterates on lp_osa_07 in quadruple (top) and octuple precision (bottom).	59
Figure 5.7	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on channel_domain problem with Q_1 - P_0 discretization. The discretized linear system has size 12546×12546 with 147742 nonzeros.	59
Figure 5.8	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on channel_domain problem with Q_1 - Q_1 discretization. The discretized linear system has size 12675×12675 with 242381 nonzeros.	60
Figure 5.9	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on lp_d6cube, where A is 415×6184 with 37704 nonzeros.	60
Figure 5.10	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on well1033, where A is 1033×320 with 4732 nonzeros.	61
Figure 5.11	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR iterates on colliding_flow problem with $Q_1 - P_0$ discretization. The discretized linear system has size 12546×12546 with 146241 nonzeros.	61
Figure 5.12	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR iterates on colliding_flow problem with $Q_1 - Q_1$ discretization. The discretized linear system has size 12675×12675 with 239873 nonzeros.	61
Figure 5.13	Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR iterates on lpi_klein3 in double (top), quadruple (center) and octuple precision (bottom) where A is 994×1082 with 13101 nonzeros.	62
Figure 6.1	Residual history of GPMR and GMRES.	82
Figure 6.2	Residual history of GPMR and GMRES.	82
Figure 6.3	Residual history of GPMR and TRIMR.	83
Figure 6.4	Residual history of GPMR(ℓ) and GMRES(ℓ).	84

Figure 6.5	Residual history of GPMR, GMRES, GMRES-SCHUR(BA) and GMRES-SCHUR(AB) on systems generated from an interior-point method. Top: well-conditioned systems generated at the first interior-point iteration. Bottom: ill-conditioned systems generated at interior-point iteration 5.	85
Figure 6.6	Relations between the relevant standard and block processes, and methods based on them. Minimum-residual methods appear in white on gray, while Galerkin methods appear in white on black. Thick arrows indicate that a method is based on a process. Thin lines between processes indicate equivalence when the operator is symmetric. Dashed lines between processes indicate a block relationship. Double stroke arrows between processes indicate equivalence.	86
Figure 7.1	Residual and A-residual histories for MINARES and MINRES-QLP on consistent systems generated from the SuiteSparse Matrix Collection. Top: System based on the nonsingular matrix rail_5177 ($n = 5177$). Bottom: System based on the singular matrix bcsstm36 ($n = 23052$).	109
Figure 7.2	A-residual history for MINARES, MINRES-QLP and LSMR on singular inconsistent systems generated from the SuiteSparse Matrix Collection. Left: System based on the singular matrix zenios ($n = 2873$). Right: System based on the singular matrix laser ($n = 3002$).	110

LISTE DES SIGLES ET ABRÉVIATIONS

Symboles

A, B, C	Matrices (lettres latines majuscules)
u, v, w	Vecteurs (lettres latines minuscules)
α, β, γ	Scalars (lettres grecques minuscules)
$M \succ 0$	Matrice définie positif
M^{-1}	Matrice inverse
M^T	Matrice transposée
M^H	Matrice transconjuguée
$\ v\ $	Norme euclidienne
$\ M\ _F$	Norme de Frobenius
I_k	Matrice identité de taille k
e_i	Vecteur représentant la i -ème colonne de la matrice identité

Espaces et ensembles

$\text{Vect}\{v_1, \dots, v_n\}$	Ensembles des combinaisons linéaires des vecteurs v_i
$\mathbb{R}$	Ensemble des réels
$\mathbb{C}$	Ensemble des complexes
$\mathcal{K}_k(A, b)$	Espace de Krylov
$\mathcal{K}_k^\square(A, B)$	Espace de Krylov par blocs

CHAPITRE 1 INTRODUCTION

L'algèbre linéaire numérique est un domaine des mathématiques appliquées qui fait usage d'outils informatiques pour résoudre des problèmes impliquant des opérations sur des vecteurs ou des matrices. Ses principaux concepts incluent la résolution de systèmes d'équations linéaires, les factorisations matricielles, ainsi que les décompositions en valeurs propres et en valeurs singulières.

Cet outil est fondamental pour la résolution de sous-problèmes dans des domaines variés tels que l'optimisation, les équations différentielles, l'analyse de données et le traitement d'images. Son utilisation répandue dans une diversité de contextes souligne l'importance de développer des codes informatiques génériques capables de s'adapter aux besoins spécifiques de chaque secteur.

L'implémentation polymorphe joue ce rôle en assurant la flexibilité et la portabilité de ces codes, garantissant ainsi qu'ils peuvent être utilisés de manière efficace dans une multitude de contextes.

1.1 Objectifs de recherche

Ce projet de recherche poursuit deux objectifs. Le premier objectif est le développement théorique de nouvelles méthodes de Krylov adaptées à des caractéristiques spécifiques des systèmes d'équations linéaires. Le deuxième objectif vise à créer une bibliothèque Julia contenant une implémentation polymorphe des nouvelles méthodes de Krylov.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Méthodes de Krylov

2.1.1 Origine des méthodes de Krylov

Les méthodes de Krylov ont pour origine le théorème de Cayley-Hamilton. Pour une matrice carrée A de taille n , si son polynôme caractéristique est donné par

$$\det(A - \lambda I_n) = \lambda^n + p_{n-1}\lambda^{n-1} + \cdots + p_1\lambda + p_0,$$

alors la relation suivante est vérifiée :

$$A^n + p_{n-1}A^{n-1} + \cdots + p_1A + p_0I_n = 0_n.$$

Si A est inversible alors $p_0 \neq 0$ et l'inverse de A peut être exprimé comme

$$A^{-1} = -\frac{1}{p_0}(A^{n-1} + p_{n-1}A^{n-2} + \cdots + p_1I_n).$$

Ainsi, pour un système linéaire $Ax = b$, où b est un vecteur de taille n , la solution $x^* = A^{-1}b$ réside dans l'espace $\text{Vect}\{b, Ab, \dots, A^{n-1}b\}$, appelé espace de Krylov et noté $\mathcal{K}_n(A, b)$.

Le mathématicien russe Aleksey Nikolaevich Krylov [53] a introduit pour la première fois les sous-espaces de Krylov. Saad [83] offre une analyse approfondie sur les origines des méthodes de Krylov, fournissant des détails supplémentaires.

2.1.2 Convergence des méthodes de Krylov

La convergence des méthodes de Krylov, un aspect essentiel de leur efficacité, est intimement liée aux valeurs propres de la matrice A lorsque A est normale ($AA^H = A^HA$). Comme le polynôme minimal d'une matrice A est un polynôme de degré minimal m qui divise l'ensemble des polynômes annulateurs de A , il divise le polynôme caractéristique de A et possède les mêmes racines. On en déduit donc que $x^* = A^{-1}b \in \mathcal{K}_m(A, b)$ et qu'on a une convergence rapide lorsque le nombre de racines distinctes du polynôme caractéristique est faible. Cela indique que la matrice A possède un nombre limité de valeurs propres distinctes. Si les valeurs propres de A sont condensées alors on obtient une bonne approximation de x^* en peu d'itérations. Lorsque A n'est pas normale, la convergence des méthodes de Krylov est

beaucoup plus complexe à caractériser en fonction des propriétés spectrales de A [Greenbaum et al. \[39\]](#).

2.1.3 Préconditionneurs

Pour accélérer la convergence, on a recourt à des preconditionneurs, c'est à dire des transformations qui modifient un système linéaire $Ax = b$ en une forme équivalente avec des propriétés spectrales plus intéressantes. La conception des preconditionneurs dépend fortement de l'origine du problème, et la plupart des preconditionneurs doivent prendre en compte des informations spécifiques au problème. En général, des preconditionneurs spécialisés surpassent des preconditionneurs génériques tels que les factorisations incomplètes. La conception d'un preconditionneur implique des compromis entre son coût d'application et sa capacité à capturer les caractéristiques du système d'origine car il doit être appliqué au moins une fois par itération dans une méthode de Krylov.

Il existe trois variantes de preconditionnement :

TABLEAU 2.1 Type de preconditionnement avec P_ℓ et P_r des matrices carrées inversibles.

Préconditionnement à gauche	$P_\ell^{-1}Ax = P_\ell^{-1}b$
Préconditionnement à droite	$AP_r^{-1}y = b$ avec $x = P_r^{-1}y$
Préconditionnement des deux côtés	$P_\ell^{-1}AP_r^{-1}y = P_\ell^{-1}b$ avec $x = P_r^{-1}y$

Le preconditionnement gauche préserve l'erreur $x_k - x^*$ tandis que le preconditionnement droit maintient invariant le résidu $b - Ax_k$. Le preconditionnement des deux côtés est la seule variante qui permet de préserver la symétrie d'un système linéaire.

Une méthode de Krylov dédiée aux systèmes linéaires non symétriques permet les trois variantes de preconditionnement. Lorsque A est symétrique, nous ne pouvons utiliser que le preconditionnement centré $L^{-1}AL^{-T}y = L^{-1}b$ avec $x = L^{-T}y$. Le preconditionnement centré est un cas particulier du preconditionnement des deux côtés avec $P_\ell = L = P_r^T$ qui maintient la symétrie.

2.1.4 Processus de Krylov

Un processus de Krylov est essentiel pour construire de manière itérative une solution $x_k \in \mathcal{K}_k(A, b)$, au système linéaire $Ax = b$. Ces processus permettent de construire des bases, principalement orthonormales, pour les espaces de Krylov. À partir d'un processus de Krylov, on projette A et b dans l'espace de Krylov généré, simplifiant ainsi le problème en se ramenant

à un sous-problème avec une matrice présentant une structure facilement exploitable. Chaque processus requiert des hypothèses différentes sur A et b , amenant à des projections avec différentes structures. Par exemple, la projection de A est toujours une matrice tridiagonale symétrique avec le processus de Lanczos [55] et une matrice Hessenberg supérieure avec le processus de Arnoldi [2] peu importe la structure de A . En notant V_k comme la base de l'espace de Krylov à l'itération k , les itérations se présentent sous la forme $x_k = V_k y_k$, où y_k est un vecteur de taille k . y_k est déterminé en résolvant un sous-problème qui utilise la projection de A . En fonction du sous-problème résolu, les itérées x_k présentent différentes propriétés, telles que la décroissance monotone de la norme des résidus $\|b - Ax_k\|$ ou de la norme des erreurs $\|x_k - x^*\|$. La variété des méthodes de Krylov résulte de la combinaison des processus de Krylov et des sous-problèmes. Chaque méthode formule ainsi des hypothèses distinctes quant à A et b . En cas de matrice rectangulaire A , fréquemment rencontrée dans des problèmes de moindres carrés ou de moindre norme, on se ramène à des systèmes linéaires carrés en résolvant les équations normales $A^T A x = A^T b$ et $A A^T x = b$. Le processus de Golub et Kahan [34] est approprié pour résoudre ces problèmes car il génère des bases orthonormales aux espaces $\mathcal{K}_k(A^T A, A^T b)$ et $\mathcal{K}_k(A A^T, b)$.

L'article de Ipsen et Meyer [50] offre une introduction alternative aux méthodes de Krylov, complémentaire aux livres de Greenbaum [38] et Saad [82].

2.1.5 Synthèse des processus et méthodes de Krylov existantes

Les méthodes de Krylov peuvent être regroupées en cinq catégories en fonction du processus à partir duquel elles sont dérivées.

Le premier processus est celui de Lanczos [55], adapté aux systèmes linéaires symétriques. Les méthodes basées sur ce processus incluent SYMMLQ [72], CG [48], CR [48], MINRES [72] et MINRES-QLP [13].

Une version non symétrique du processus de Lanczos a également été développée pour résoudre des systèmes linéaires carrés non symétriques. Les méthodes basées sur ce processus incluent BICG [27], QMR [30], TFQMR [31], BICGSTAB [91] et CGS [87].

Le processus d'Arnoldi [2] est une autre référence pour les systèmes linéaires carrés non symétriques sur lequel reposent les méthodes DIOM [78], FOM [81], DQGMRES [84], GMRES [80], et FGMRES [79].

Pour les problèmes rectangulaires, on dépend principalement du processus de Golub-Kahan [34]. Les méthodes de référence pour les problèmes de moindres carrés comprennent LSLQ [24], LSQR [73] et LSMR [28]. Pour les problèmes de moindre norme, on dispose de LNLQ

[23], CRAIG [14] et CRAIGMR [70].

Enfin, un processus moins courant est le processus de Saunders-Simon-Yip [85]. Il peut être utilisé pour résoudre des problèmes de moindres carrés avec USYMQR [85] ou de moindre norme avec USYMLQ [85]. Une combinaison des deux a également été développée pour résoudre des problèmes de point de selle avec USYMLQR [9].

TABLEAU 2.2 Synthèse des méthodes de Krylov existantes

$Ax = b, A = A^H, A \succ 0$	CG, CR
$Ax = b, A = A^H$	SYMMLQ, MINRES, MINRES-QLP
$Ax = b, A \neq A^H$	BICG, QMR, CGS, BICGSTAB, TFQMR, DIOM, FOM, DQGMRES, GMRES, FGMRES
$\min_{x \in \mathbb{C}^n} \ Ax - b\ $	LSLQ, LSQR, LSMR, USYMQR
$\min_{x \in \mathbb{C}^n} \ x\ \text{ s.t. } Ax = b$	LNLQ, CRAIG, CRAIGMR, USYMLQ
$\begin{bmatrix} M & A \\ A^H & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$	USYMLQR

2.2 Programmation polymorphe

La programmation polymorphe permet à un code informatique de prendre en charge différents types de données sans nécessiter une implémentation spécifique pour chaque type. Ses principaux avantages résident dans la réutilisabilité du code, la facilité de maintenance et la robustesse face aux évolutions matérielles et logicielles futures.

Cette approche est présente dans plusieurs langages de programmation tels que C++, Python, Java, et Julia. Notre choix s'est porté sur Julia en raison de sa syntaxe simple, similaire à un langage de haut niveau comme Python, et de ses performances comparables à celles d'un langage de bas niveau compilé tel que C++. Cette particularité découle de sa compilation à la volée pour générer du code à l'exécution, s'il n'a pas été préalablement compilé.

Julia propose un polymorphisme dit paramétrique, souvent considéré comme plus générique que le polymorphisme par héritage présent dans les langages orientés objets (C++, Python, Java), qui nécessite une hiérarchie de classes.

Un autre avantage de Julia est sa faculté à définir de multiples méthodes pour une fonction selon le type de ses arguments, offrant un fort potentiel d'optimisation.

L'interopérabilité de Julia avec les langages C et Fortran a également été un facteur dé-

terminant de notre choix, puisqu'aucune latence additionnelle n'est nécessaire pour appeler des routines de ces langages. L'intégration de bibliothèques d'algèbres linéaires numériques comme BLAS et LAPACK, implémentées dans ces langages, est donc optimale.

Les similitudes de Julia avec le logiciel MATLAB en font également un outil facile d'accès en termes d'apprentissage pour les chercheurs. La migration est d'autant plus facilitée par l'accès libre du langage et sa présence multiplateforme (Windows, Mac, Linux).

CHAPITRE 3 ORGANISATION DE LA THÈSE

Cette thèse contient dix chapitres dont cinq sont dédiés à des articles scientifiques qui synthétisent mes travaux de recherche. Elle est organisée comme suit.

Le [chapitre 1](#) établit le cadre et les objectifs de la recherche.

Le [chapitre 2](#) est un état de l'art sur les méthodes de Krylov et la programmation polymorphe.

Le [chapitre 3](#) décrit l'organisation de la thèse.

Le [chapitre 4](#) présente trois nouvelles méthodes de Krylov ; BiLQ pour résoudre des systèmes linéaires non symétriques carrés ainsi que BiLQR et TriLQR pour résoudre des systèmes adjoints. L'article associé [\[61\]](#) a été publié dans le *SIAM Journal on Matrix Analysis and Applications* (SIMAX).

Le [chapitre 5](#) présente deux nouvelles méthodes de Krylov, TriCG et TrIMR, pour résoudre des systèmes linéaires symétriques structurés par blocs, comme les systèmes de point de selle ou les systèmes quasi-définis. L'article associé [\[62\]](#) a été publié dans le *SIAM Journal on Scientific Computing* (SISC).

Le [chapitre 6](#) introduit la nouvelle méthode de Krylov, GPMR, qui généralise TrIMR pour les systèmes linéaires non-symétriques structurés par blocs, comme les systèmes de point de selle généralisées. L'article associé [\[63\]](#) a été publié dans le *SIAM Journal on Matrix Analysis and Applications* (SIMAX).

Le [chapitre 7](#) présente deux nouvelles méthodes de Krylov, MINARES et CAR, pour résoudre des systèmes linéaires symétriques. L'article associé [\[65\]](#) a été soumis au *SIAM Journal on Matrix Analysis and Applications* (SIMAX).

Le [chapitre 8](#) présente la bibliothèque numérique Julia Krylov.jl qui contient une implémentation polymorphe de plus de trente-cinq méthodes de Krylov. L'article associé [\[64\]](#) a été publié dans le *Journal of Open Source Software* (JOSS).

Le [chapitre 9](#) contient une discussion générale des contributions.

Le [chapitre 10](#) apporte une conclusion à l'ensemble du travail de recherche.

CHAPITRE 4 ARTICLE 1: BiLQ: AN ITERATIVE METHOD FOR NONSYMMETRIC LINEAR SYSTEMS WITH A QUASI-MINIMUM ERROR PROPERTY

ALEXIS MONTISON AND DOMINIQUE ORBAN

Manuscript published on August 4, 2020, in the *SIAM Journal on Matrix Analysis and Applications* (SIMAX)

Abstract

We introduce an iterative method named BiLQ for solving general square linear systems $Ax = b$ based on the Lanczos biorthogonalization process defined by least-norm subproblems, and that is a natural companion to BiCG and QMR. Whereas the iterates of BiCG, CGs and BiCGSTAB may not exist when the tridiagonal projection of A is singular, BiLQ is reliable on compatible systems even if A is ill-conditioned or singular. As in the symmetric case, the BiCG residual is often smaller than the BiLQ residual and, when the BiCG iterate exists, an inexpensive transfer from the BiLQ iterate is possible. Although the Euclidean norm of the BiLQ error is usually not monotonic, it is monotonic in a different norm that depends on the Lanczos vectors. We establish a similar property for the QMR residual. BiLQ combines with QMR to take advantage of two initial vectors and solve a system and an adjoint system simultaneously at a cost similar to that of applying either method. We derive an analogous combination of USYMLQ and USYMQR based on the orthogonal tridiagonalization process. The resulting combinations, named BiLQR and TrILQR, may be used to estimate integral functionals involving the solution of a primal and an adjoint system. We compare BiLQR and TrILQR with MINRES-QLP on a related augmented system, which performs a comparable amount of work and requires comparable storage. In our experiments, BiLQR terminates earlier than TrILQR and MINRES-QLP in terms of residual and error of the primal and adjoint systems.

4.1 Introduction

We consider the square consistent linear system

$$Ax = b, \tag{4.1}$$

where $A \in \mathbb{R}^{n \times n}$ can be nonsymmetric, is either large and sparse, or is only available as a linear operator, i.e., via operator-vector products. Systems such as (4.1) arise in the discretization of partial differential equations (PDEs) in numerous applications, including compressible turbulent fluid flow [12], and circuit simulation [16]. We consider Krylov subspace methods and are interested in generating iterates with guaranteed decrease of the error $x_k - x_\star$ in a certain norm, where $x_\star$ is the solution of (4.1).

The foundation of Krylov methods is a basis-generation process upon which three methods may be developed: one computing the minimum-norm solution of an under-determined system, one solving a square system and imposing a Galerkin condition, and one solving an over-determined system in the least-squares sense. These methods may be implemented with the help of LQ, LU or QR factorizations of a related operator, respectively.

In this paper, we develop an iterative method named BiLQ of the first type based on the Lanczos [55] biorthogonalization process. Together with BiCG [27] and QMR [30], BiLQ completes the family of methods based on the biorthogonalization process. Our motivation for developing BiLQ is that the BiCG iterate is not necessarily well defined at each iteration. By contrast, the BiLQ iterate is always determined from a well-posed subproblem. In addition, whenever the BiCG iterate is well defined, it is possible to transition from the BiLQ iterate to the BiCG iterate if desired. In this sense, it is always safe to use BiLQ in place of BiCG. We begin by stating the defining properties of BiLQ, describing its implementation in detail, and illustrating its behavior on numerical examples side by side with BiCG and QMR.

In a second stage, we exploit the fact that the biorthogonalization process requires two initial vectors to develop a combination of BiLQ and QMR that solves (4.1) together with a dual system

$$A^T t = c \tag{4.2}$$

simultaneously at a cost comparable to that of applying BiLQ or QMR to solve only one of those systems. The resulting combination is named BiLQR and is employed to illustrate the computation of superconvergent estimates of integral functionals arising in certain PDE problems.

We note that a similar approach may be developed for the Saunders *et al.* [85] orthogonal tridiagonalization process, which also requires two initial vectors, by combining USYMLQ and USYMQR. The resulting combination is named TrILQR.

Finally, we compare BiLQR and TrILQR with MINRES-QLP on a related augmented system to solve both (4.1) and (4.2) simultaneously. In our experiments, BiLQR terminates earlier

than TRILQR and MINRES-QLP in terms of residual and error of the primal and adjoint systems. BiLQR and TRILQR are also cheaper to apply than the methods of [Lu and Darmofal \[56\]](#) and [Golub *et al.* \[35\]](#), as discussed in the next section.

Our Julia [5] implementation of BiLQ, QMR, USYMLQ, USYMQR, BiLQR, TRILQR, and MINRES-QLP are available from github.com/JuliaSmoothOptimizers/Krylov.jl. Thanks to multiple dispatch, a language feature allowing automatic compilation of variants of each method corresponding to inputs expressed in various floating-point systems, our implementations run in any floating-point precision supported.

Related research

[Paige and Saunders \[72\]](#) developed one of the best-known minimum error methods, SYMMLQ, based on the symmetric Lanczos process. SYMMLQ inspired [Estrin *et al.* \[23, 24\]](#) to develop LSLQ and LNLQ for rectangular problems based on the [Golub and Kahan \[34\]](#) process. LSLQ and LNLQ are equivalent to SYMMLQ applied to the normal equations and normal equations of the second kind, respectively.

[Saunders *et al.* \[85\]](#) define USYMLQ for square consistent systems based on the orthogonal tridiagonalization process. USYMLQ is based on a subproblem similar to that of SYMMLQ, and coincides with SYMMLQ in the symmetric case. Its companion method, USYMQR, is similar in spirit to MINRES. [Buttari *et al.* \[9\]](#) combine both into a method named USYMLQR designed to solve symmetric saddle-point systems with general right-hand side, and inspire the development of BiLQR and TRILQR in the present paper.

[Weiss \[94\]](#) describes two types of error-minimizing Krylov methods for square A ; one based on a process applied to $A^T A$, and one to A^T . Our approach is to apply the biorthogonalization process directly to A . We defer a numerical stability analysis to future work, but note that [Paige *et al.* \[74\]](#) study the augmented stability of the biorthogonalization process. In this sense, we make the implicit assumption that computations are carried out in exact arithmetic. This assumption prompted us to develop our implementations so that they can be applied in any supported floating-point arithmetic.

The simultaneous solution of a system and an adjoint system has attracted attention in the past. Notably, [Lu and Darmofal \[56\]](#) devise a variant of QMR to solve both systems at once at a cost approximately equal to that of QMR applied to one of the systems but with an increase in storage requirements. [Golub *et al.* \[35\]](#) follow a similar approach and use a variant of USYMQR to solve both (4.1) and (4.2). An advantage of USYMQR is to produce monotonic residuals in the Euclidean norm for both systems. We illustrate in [Table 4.2](#) that our methods

are cheaper and have smaller storage requirements than those of [Lu and Darmofal \[56\]](#) and [Golub *et al.* \[35\]](#) though residuals are not monotonic in the Euclidean norm.

Notation

Matrices and vectors are denoted by capital and lowercase Latin letters, respectively, and scalars by Greek letters. An exception is made for cosines and sines (c, s) that compose Givens reflections. For a vector v , $\|v\|$ denotes the Euclidean norm of v , and for symmetric and positive-definite N , the N -norm of v is $\|v\|_N^2 = v^T N v$. For a matrix M , $\|M\|_F$ denotes the Frobenius norm of M . The vector e_i is the i -th column of an identity matrix of size dictated by the context. Vectors and scalars decorated by a bar will be updated at the next iteration. For $j = 2, \dots, k$, we use the compact representation

$$Q_{j-1,j} = \begin{bmatrix} & j-1 & j \\ c_j & s_j \\ s_j & -c_j \end{bmatrix} := \begin{bmatrix} I_{j-2} & & \\ & c_j & s_j \\ & s_j & -c_j \\ & & & I_{k-j} \end{bmatrix},$$

for orthogonal reflections, where $s_j^2 + c_j^2 = 1$, where border indices indicate row and column numbers, and where I_k represents the $k \times k$ identity operator. We abuse the notation $\bar{z}_k = (z_{k-1}, \bar{\zeta}_k)$ to represent the column vector $\begin{bmatrix} z_{k-1}^T & \bar{\zeta}_k \end{bmatrix}^T$.

4.2 Derivation of BiLQ

4.2.1 The Lanczos biorthogonalization process

The [Lanczos](#) biorthogonalization process generates sequences of vectors $\{v_k\}$ and $\{u_k\}$ such that $v_i^T u_j = \delta_{ij}$ in exact arithmetic for as long as the process does not break down. The process is summarized as [Algorithm 1](#).

We denote $V_k = [v_1 \ \dots \ v_k]$ and $U_k = [u_1 \ \dots \ u_k]$. We choose the scaling factors β_k and γ_k so that $v_k^T u_k = 1$ for all $k \geq 1$, i.e., $V_k^T U_k = I_k$. After k iterations, the situation may be summarized as

$$AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T = V_{k+1} T_{k+1,k} \quad (4.3a)$$

$$A^T U_k = U_k T_k^T + \gamma_{k+1} u_{k+1} e_k^T = U_{k+1} T_{k,k+1}^T, \quad (4.3b)$$

Algorithm 1 Lanczos Biorthogonalization Process

Require: A, b, c

 1: $v_0 = 0, u_0 = 0$

 2: $\beta_1 v_1 = b, \gamma_1 u_1 = c$
 (β_1, γ_1) so that $v_1^T u_1 = 1$

 3: **for** $k = 1, 2, \dots$ **do**

 4: $q = Av_k - \gamma_k v_{k-1}, \alpha_k = u_k^T q$

 5: $p = A^T u_k - \beta_k u_{k-1}$

 6: $\beta_{k+1} v_{k+1} = q - \alpha_k v_k$
 $(\beta_{k+1}, \gamma_{k+1})$ so that $v_{k+1}^T u_{k+1} = 1$

 7: $\gamma_{k+1} u_{k+1} = p - \alpha_k u_k$

 8: **end for**

where

$$T_k = \begin{bmatrix} \alpha_1 & \gamma_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \gamma_k \\ & & \beta_k & \alpha_k \end{bmatrix}, \quad T_{k,k+1} = \begin{bmatrix} T_k & \gamma_{k+1} e_k \end{bmatrix}, \quad T_{k+1,k} = \begin{bmatrix} T_k \\ \beta_{k+1} e_k^T \end{bmatrix}.$$

The columns of V_k and U_k form a basis for $\mathcal{K}_k := \text{Span}\{b, Ab, \dots, A^{k-1}b\}$ and $\mathcal{L}_k := \text{Span}\{c, A^T c, \dots, (A^T)^{k-1}c\}$, respectively. Though V_k cannot be expected to be orthogonal to U_k in inexact arithmetic, and therefore $U_k^T A V_k = T_k$ cannot be expected to hold, (4.3) holds to within machine precision.

4.2.2 Definition of BiLQ

BICG [27] generates an approximation $x_k^C = V_k y_k^C$ to a solution of (4.1) where $y_k^C \in \mathbb{R}^k$ solves

$$T_k y = \beta_1 e_1. \quad (4.4)$$

Unfortunately, T_k could be singular and (4.4) could be inconsistent or have multiple solutions. In such situations, x_k^C is undefined.

To remedy such situations, we define BiLQ as the method that generates an approximation $x_k^L = V_k y_k^L$, where $y_k^L \in \mathbb{R}^k$ solves

$$\underset{y}{\text{minimize}} \quad \|y\| \quad \text{subject to} \quad T_{k-1,k} y = \beta_1 e_1. \quad (4.5)$$

Note that the constraints of (4.5) are consistent as long as Algorithm 1 does not break down so that x_k^L is always well defined.

QMR [30] generates $x_k^Q = V_k y_k^Q$ where $y_k^Q \in \mathbb{R}^k$ solves

$$\underset{y}{\text{minimize}} \quad \|T_{k+1,k}y - \beta_1 e_1\|, \quad (4.6)$$

so that x_k^Q is also always well defined.

When A is symmetric and $b = c$, Algorithm 1 coincides with the symmetric Lanczos process and the three above methods are equivalent to CG [48], SYMMLQ [72], and MINRES [72], respectively.

4.2.3 An LQ factorization

We determine y_k^L , the solution of (4.5), via the LQ factorization of $T_{k-1,k}$, which we obtain from the LQ factorization

$$T_k = \bar{L}_k Q_k, \text{ where} \quad (4.7a)$$

$$\bar{L}_k = \begin{bmatrix} \delta_1 & & & & \\ \lambda_1 & \delta_2 & & & \\ \varepsilon_1 & \lambda_2 & \delta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \varepsilon_{k-3} & \lambda_{k-2} & \delta_{k-1} \\ & & & \varepsilon_{k-2} & \lambda_{k-1} & \bar{\delta}_k \end{bmatrix} = \begin{bmatrix} L_{k-1} & 0 \\ \varepsilon_{k-2} e_{k-2}^T + \lambda_{k-1} e_{k-1}^T & \bar{\delta}_k \end{bmatrix}, \quad (4.7b)$$

and $Q_k^T = Q_{1,2} Q_{2,3} \cdots Q_{k-1,k}$ is orthogonal and defined as a product of Givens reflections. Indeed, the above yields the LQ factorization

$$T_{k-1,k} = \begin{bmatrix} L_{k-1} & 0 \end{bmatrix} Q_k. \quad (4.8)$$

If we initialize $\bar{\delta}_1 := \alpha_1$, $\bar{\lambda}_1 := \beta_2$, $c_1 = -1$, and $s_1 = 0$, individual factorization steps may be represented as an application of $Q_{k-2,k-1}$ to $T_k Q_{k-2}^T$:

$$\begin{array}{c} k-2 \\ k-1 \\ k \end{array} \begin{bmatrix} & k-2 & k-1 & k \\ \bar{\delta}_{k-2} & \gamma_{k-1} & & \\ \bar{\lambda}_{k-2} & \alpha_{k-1} & \gamma_k & \\ & \beta_k & \alpha_k & \end{bmatrix} \begin{bmatrix} & k-2 & k-1 & k \\ c_{k-1} & s_{k-1} & & \\ s_{k-1} & -c_{k-1} & & \\ & & & 1 \end{bmatrix} = \begin{bmatrix} & k-2 & k-1 & k \\ \delta_{k-2} & 0 & & \\ \lambda_{k-2} & \bar{\delta}_{k-1} & \gamma_k & \\ \varepsilon_{k-2} & \bar{\lambda}_{k-1} & \alpha_k & \end{bmatrix},$$

followed by an application of $Q_{k-1,k}$ to the result:

$$\begin{array}{c} k-2 \\ k-1 \\ k \end{array} \begin{bmatrix} & k-2 & k-1 & k \\ \delta_{k-2} & & & \\ \lambda_{k-2} & \bar{\delta}_{k-1} & \gamma_k & \\ \varepsilon_{k-2} & \bar{\lambda}_{k-1} & \alpha_k & \end{bmatrix} \begin{bmatrix} & k-2 & k-1 & k \\ 1 & & & \\ & c_k & s_k & \\ & s_k & -c_k & \end{bmatrix} = \begin{bmatrix} & k-2 & k-1 & k \\ \delta_{k-2} & & & \\ \lambda_{k-2} & \delta_{k-1} & & \\ \varepsilon_{k-2} & \lambda_{k-1} & \bar{\delta}_k & \end{bmatrix}.$$

The reflection $Q_{k-1,k}$ is designed to zero out γ_k on the superdiagonal of T_k and affects three rows and two columns. It is defined by

$$\delta_{k-1} = \sqrt{\bar{\delta}_{k-1}^2 + \gamma_k^2}, \quad c_k = \bar{\delta}_{k-1}/\delta_{k-1}, \quad s_k = \gamma_k/\delta_{k-1}, \quad (4.9)$$

and yields the recursion

$$\varepsilon_{k-2} = s_{k-1}\beta_k, \quad k \geq 3, \quad (4.10a)$$

$$\bar{\lambda}_{k-1} = -c_{k-1}\beta_k, \quad k \geq 3, \quad (4.10b)$$

$$\lambda_{k-1} = c_k\bar{\lambda}_{k-1} + s_k\alpha_k, \quad k \geq 2, \quad (4.10c)$$

$$\bar{\delta}_k = s_k\bar{\lambda}_{k-1} - c_k\alpha_k, \quad k \geq 2. \quad (4.10d)$$

4.2.4 Definition and update of the BiLQ and BiCG iterates

In order to compute the solution y_k^L of (4.5) using (4.8), we solve $[L_{k-1} \ 0] Q_k y_k^L = \beta_1 e_1$. If $z_{k-1} := (\zeta_1, \dots, \zeta_{k-1})$ is defined from $L_{k-1} z_{k-1} = \beta_1 e_1$, the minimum-norm solution of (4.5) is $y_k^L = Q_k^T \begin{bmatrix} z_{k-1} \\ 0 \end{bmatrix}$, and $\|y_k^L\| = \|z_{k-1}\|$.

We may compute y_k^C in (4.4) simultaneously as a cheap update of y_k^L . Indeed, (4.4) and (4.7) yield $\bar{L}_k Q_k y_k^C = \beta_1 e_1$. Let $\bar{z}_k := (z_{k-1}, \bar{\zeta}_k)$ be defined so $\bar{L}_k \bar{z}_k = \beta_1 e_1$. Then, $y_k^C = Q_k^T \bar{z}_k$. If $\bar{\delta}_k = 0$, y_k^C and the BiCG iterate x_k^C are undefined. The components of $\bar{z}_k$ are computed from

$$\eta_k = \begin{cases} \beta_1, & k = 1, \\ -\lambda_1 \zeta_1, & k = 2, \\ -\varepsilon_{k-2} \zeta_{k-2} - \lambda_{k-1} \zeta_{k-1}, & k \geq 3, \end{cases} \quad (4.11a)$$

$$\zeta_{k-1} = \eta_{k-1}/\delta_{k-1}, \quad k \geq 2, \quad (4.11b)$$

$$\bar{\zeta}_k = \eta_k/\bar{\delta}_k, \quad \text{if } \bar{\delta}_k \neq 0. \quad (4.11c)$$

By definition, $x_k^L = V_k y_k^L$ and $x_k^C = V_k y_k^C$. To avoid storing V_k , we let

$$\bar{D}_k := V_k Q_k^T = [d_1, d_2, \dots, d_{k-1}, \bar{d}_k], \quad \bar{d}_1 = v_1, \quad (4.12)$$

defined by the recursion

$$\begin{aligned} d_{k-1} &= c_k \bar{d}_{k-1} + s_k v_k \\ \bar{d}_k &= s_k \bar{d}_{k-1} - c_k v_k. \end{aligned} \quad (4.13)$$

Finally,

$$x_k^L = V_k y_k^L = \bar{D}_k \begin{bmatrix} z_{k-1} \\ 0 \end{bmatrix} = D_{k-1} z_{k-1} = x_{k-1}^L + \zeta_{k-1} d_{k-1} \quad (4.14a)$$

$$x_k^C = V_k y_k^C = \bar{D}_k \bar{z}_k = D_{k-1} z_{k-1} + \bar{\zeta}_k \bar{d}_k = x_k^L + \bar{\zeta}_k \bar{d}_k. \quad (4.14b)$$

We see from (4.14b) that it is possible to transfer from x_k^L to x_k^C cheaply provided $\bar{\zeta}_k \neq 0$. Such transfer was described by [Paige and Saunders \[72\]](#) as an inexpensive update from the SYMMLQ to the CG point in the symmetric case.

4.2.5 Residual estimates

The identity (4.3a) allows us to write the residual associated with $x_k = V_k y_k$ as

$$r_k = b - Ax_k = \beta_1 v_1 - AV_k y_k = \beta_1 v_1 - V_{k+1} T_{k+1,k} y_k.$$

Thus, (4.5) yields the residual at the BiLQ iterate:

$$\begin{aligned} r_k^L &= V_{k-1}(\beta_1 e_1 - T_{k-1,k} y_k^L) - (\beta_k e_{k-1} + \alpha_k e_k)^T y_k^L v_k - \beta_{k+1} e_k^T y_k^L v_{k+1} \\ &= -(\beta_k e_{k-1} + \alpha_k e_k)^T y_k^L v_k - \beta_{k+1} e_k^T y_k^L v_{k+1}, \end{aligned} \quad (4.15)$$

and (4.4) yields the residual at the BiCG iterate:

$$r_k^C = V_k(\beta_1 e_1 - T_k y_k^C) - \beta_{k+1} v_{k+1} e_k^T y_k^C = -\beta_{k+1} e_k^T y_k^C v_{k+1}. \quad (4.16)$$

Because $Q_k^T = Q_{1,2} Q_{2,3} \cdots Q_{k-1,k}$, we have

$$\begin{aligned} e_{k-1}^T Q_k^T &= e_{k-1}^T Q_{k-2,k-1} Q_{k-1,k} = s_{k-1} e_{k-2}^T - c_{k-1} c_k e_{k-1}^T - c_{k-1} s_k e_k^T, \\ e_k^T Q_k^T &= e_k^T Q_{k-1,k} = s_k e_{k-1}^T - c_k e_k^T, \end{aligned}$$

so that

$$\begin{aligned} e_{k-1}^T y_k^L &= e_{k-1}^T Q_k^T \begin{bmatrix} z_{k-1} \\ 0 \end{bmatrix} = s_{k-1} \zeta_{k-2} - c_{k-1} c_k \zeta_{k-1}, \\ e_k^T y_k^L &= e_k^T Q_k^T \begin{bmatrix} z_{k-1} \\ 0 \end{bmatrix} = s_k \zeta_{k-1}, \\ e_k^T y_k^C &= e_k^T Q_k^T \bar{z}_k = s_k \zeta_{k-1} - c_k \bar{\zeta}_k. \end{aligned}$$

Therefore, if we define $\mu_k = \beta_k(s_{k-1}\zeta_{k-2} - c_{k-1}c_k\zeta_{k-1}) + \alpha_k s_k \zeta_{k-1}$, $\omega_k = \beta_{k+1} s_k \zeta_{k-1}$ and $\rho_k = \beta_{k+1}(s_k \zeta_{k-1} - c_k \bar{\zeta}_k)$, we obtain

$$\|r_k^L\| = \sqrt{\mu_k^2 \|v_k\|^2 + \omega_k^2 \|v_{k+1}\|^2 + 2\mu_k \omega_k v_k^T v_{k+1}},$$

and

$$\|r_k^C\| = |\rho_k| \|v_{k+1}\|.$$

We summarize the complete procedure as [Algorithm 2](#). For simplicity, we do not include a lookahead procedure, although a robust implementation should in order to avoid serious breakdowns [75]. The scaling factors used in our implementation are $\beta_k = |v_k^T u_k|^{\frac{1}{2}}$ and $\gamma_k = (v_k^T u_k)/\beta_k$. In [Algorithm 2](#), we only form x_k^C at the final iteration if it is well defined. In practice, a user may choose to form it at an intermediate iteration and exit if $\bar{\delta}_k \neq 0$ and $\|r_k^C\|$ is sufficiently small.

[Table 4.1](#) summarizes the cost per iteration of BiLQ, BiCG and QMR. Each method requires one operator-vector product with A and one with A^T per iteration. We assume that in-place “gemv” updates of the form $y \leftarrow Av + \gamma y$ and $y \leftarrow A^T u + \beta y$ are available. Otherwise, each method requires two additional n -vectors to store Av and $A^T u$. In the table, “dots” refers to dot products of n -vectors, “scal” refers to scaling an n -vector by a scalar, and “axpy” refers to adding a multiple of one n -vector to another one.

Table 4.1 Storage and cost per iteration of methods based on the Lanczos Biorthogonalization Process.

	n -vectors	dots	scal	axpy
BiLQ	6	2	3	7
BiCG	6	2	3	6
QMR	7	2	4	7

Algorithm 2 BiLQ

Require: A, b, c

```

1:  $\beta_1 v_1 = b, \gamma_1 u_1 = c$   $(\beta_1, \gamma_1)$  so that  $v_1^T u_1 = 1$ 
2:  $\alpha_1 = u_1^T A v_1$  begin biorthogonalization
3:  $\beta_2 v_2 = A v_1 - \alpha_1 v_1$ 
4:  $\gamma_2 u_2 = A^T u_1 - \alpha_1 u_1$ 
5:  $c_1 = -1, s_1 = 0, \bar{\delta}_1 = \alpha_1$  begin LQ factorization
6:  $\eta_1 = \beta_1, \bar{d}_1 = v_1, x_1^L = 0$ 
7: for  $k = 2, 3, \dots$  do
8:    $q = A v_k - \gamma_k v_{k-1}, \alpha_k = u_k^T q$  continue biorthogonalization
9:    $p = A^T u_k - \beta_k u_{k-1}$ 
10:   $\beta_{k+1} v_{k+1} = q - \alpha_k v_k$   $(\beta_{k+1}, \gamma_{k+1})$  so that  $v_{k+1}^T u_{k+1} = 1$ 
11:   $\gamma_{k+1} u_{k+1} = p - \alpha_k u_k$ 
12:   $\delta_{k-1} = (\bar{\delta}_{k-1}^2 + \gamma_k^2)^{\frac{1}{2}}$  compute  $Q_{k-1,k}$ 
13:   $c_k = \bar{\delta}_{k-1} / \delta_{k-1}$ 
14:   $s_k = \gamma_k / \delta_{k-1}$ 
15:   $\varepsilon_{k-2} = s_{k-1} \beta_k$  continue LQ factorization
16:   $\lambda_{k-1} = -c_{k-1} c_k \beta_k + s_k \alpha_k$ 
17:   $\bar{\delta}_k = -c_{k-1} s_k \beta_k - c_k \alpha_k$ 
18:   $\zeta_{k-1} = \eta_{k-1} / \delta_{k-1}$  update  $z_{k-1}$ 
19:   $\eta_k = -\varepsilon_{k-2} \zeta_{k-2} - \lambda_{k-1} \zeta_{k-1}$ 
20:   $\mu_k = \beta_k (s_{k-1} \zeta_{k-2} - c_{k-1} c_k \zeta_{k-1}) + \alpha_k s_k \zeta_{k-1}$ 
21:   $\omega_k = \beta_{k+1} s_k \zeta_{k-1}$ 
22:   $\|r_k^L\| = (\mu_k^2 \|v_k\|^2 + \omega_k^2 \|v_{k+1}\|^2 + 2\mu_k \omega_k v_k^T v_{k+1})^{\frac{1}{2}}$  compute  $\|r_k^L\|$ 
23:  if  $\bar{\delta}_k \neq 0$  then
24:     $\bar{\zeta}_k = \eta_k / \bar{\delta}_k$  optional: update  $\bar{z}_k$ 
25:     $\rho_k = \beta_{k+1} (s_k \zeta_{k-1} - c_k \bar{\zeta}_k)$ 
26:     $\|r_k^C\| = |\rho_k| \|v_{k+1}\|$  optional: compute  $\|r_k^C\|$ 
27:  end if
28:   $d_{k-1} = c_k \bar{d}_{k-1} + s_k v_k$  update  $\bar{D}_k$ 
29:   $\bar{d}_k = s_k \bar{d}_{k-1} - c_k v_k$ 
30:   $x_k^L = x_{k-1}^L + \zeta_{k-1} d_{k-1}$  BiLQ point
31: end for
32: if  $\bar{\delta}_k \neq 0$  then
33:    $x_k^C = x_k^L + \bar{\zeta}_k \bar{d}_k$  optional: BiCG point
34: end if

```

4.2.6 Properties

By construction, assuming [Algorithm 1](#) does not break down, there exists an iteration $p \leq n$ such that $x_{p+1}^L = x_p^C = x_\star$, the exact solution of (4.1). In particular, there exists $y_\star$ such that $x_\star = V_p y_\star$.

Because $V_k^T U_k = I_k$ at each iteration $k \leq p$, we have

$$\|x_k^L\|_{U_k U_k^T} := \|U_k^T x_k^L\| = \|y_k^L\|. \quad (4.18)$$

Note that $U_k U_k^T$ is only positive semi-definite, and therefore, it does not define a norm. However, it is positive definite on $\text{Range}(V_k)$ by biorthogonality, and $x_k^L \in \text{Range}(V_k)$. Thus, technically, $U_k U_k^T$ only defines a seminorm on $\text{Range}(V_k)$. Because

$$x_k^L = V_k y_k^L = V_p \begin{bmatrix} y_k^L \\ 0 \end{bmatrix}, \quad (4.19)$$

we also have $\|x_k^L\|_{U_p U_p^T} = \|x_k^L\|_{U_k U_k^T}$.

The following result establishes properties of x_k^L that are analogous to those of the SYMMLQ iterate in the symmetric case.

Proposition 1. *Let $x_\star$ be as above. The k th BiLQ iterate x_k^L solves*

$$\underset{x}{\text{minimize}} \|x\|_{U_k U_k^T} \text{ subject to } x \in \text{Range}(V_k), \quad b - Ax \perp \text{Range}(U_{k-1}), \quad (4.20)$$

and both

$$\underset{x}{\text{minimize}} \|x - x_\star\|_{U_p U_p^T} \text{ subject to } x \in \text{Range}(V_p V_p^T A^T U_{k-1}). \quad (4.21a)$$

$$\underset{x}{\text{minimize}} \|x - x_\star\|_{U_p U_p^T} \text{ subject to } x \in \text{Range}(V_k V_k^T A^T U_{k-1}). \quad (4.21b)$$

Proof. The first set of constraints of (4.20) requires that there exist $y \in \mathbb{R}^k$ such that $x = V_k y$. By biorthogonality, the objective value at such an x can be written $\|V_k y\|_{U_k U_k^T} = \|y\|$. Biorthogonality again and (4.15) show that y_k defined in (4.5) is primal feasible for (4.20). Dual feasibility of (4.20) requires that there exist a vector q such that $y = V_k^T A^T U_{k-1} q$. By (4.3b) and biorthogonality one more time, this amounts to $y = T_{k-1,k}^T q$, which is the same as dual feasibility for (4.5). Thus, $V_k y_k^L$ is optimal for (4.20).

To establish primal feasibility of x_k^L for (4.21a), note first that (4.3b) yields $A^T U_{k-1} = U_k T_{k-1,k}^T$. Let $\bar{V}_{p-k}$ denote the last $p-k$ columns of V_p . Biorthogonality yields

$$V_p^T U_k = \begin{bmatrix} V_k^T \\ \bar{V}_{p-k}^T \end{bmatrix} U_k = \begin{bmatrix} I_k \\ 0 \end{bmatrix}, \quad \text{and} \quad V_p V_p^T U_k = V_k.$$

As in the first part of the proof, $y_k^L = T_{k-1,k}^T q$ for some $q \in \mathbb{R}^{k-1}$, and therefore, $x_k^L =$

$V_p V_p^T A^T U_{k-1} q$. Dual feasibility requires that

$$\begin{aligned}
0 &= U_{k-1}^T A V_p V_p^T U_p U_p^T (x_k^L - x_\star) \\
&= U_{k-1}^T A V_p U_p^T V_p \left(\begin{bmatrix} y_k^L \\ 0 \end{bmatrix} - y_\star \right) \\
&= U_{k-1}^T A (x_k^L - x_\star) \\
&= -U_{k-1}^T r_k^L,
\end{aligned}$$

where we used biorthogonality and (4.19), and is satisfied because of (4.15).

Finally, x_k^L also solves (4.21b) because $V_p V_p^T A^T U_{k-1} = V_k V_k^T A^T U_{k-1}$. $\square$

Note that (4.20) continues to hold if the objective is measured in the $(U_p U_p^T)$ -seminorm. Although this seminorm is no longer iteration dependent, it is unknown until the end of the biorthogonalization process.

Because $\text{Range}(V_p V_p^T A^T U_{k-1}) \subseteq \text{Range}(V_p V_p^T A^T U_k)$, (4.21a) indicates that

$$\|x_{k+1}^L - x_\star\|_{U_p U_p^T} \leq \|x_k^L - x_\star\|_{U_p U_p^T},$$

i.e., the $(U_p U_p^T)$ -seminorm of the error is monotonic. The identity (4.19) and biorthogonality then imply that

$$\|x_k^L - x_\star\|_{U_p U_p^T} = \left\| V_p \left(\begin{bmatrix} y_k^L \\ 0 \end{bmatrix} - y_\star \right) \right\|_{U_p U_p^T} = \left\| \begin{bmatrix} y_k^L \\ 0 \end{bmatrix} - y_\star \right\| \quad (4.22)$$

is also monotonically decreasing.

Note that (4.18) is readily computable because $\|y_k^L\| = \|z_{k-1}\|$, where z_{k-1} is defined at the beginning of section 4.2.4, and can be updated as

$$\|x_{k+1}^L\|_{U_{k+1} U_{k+1}^T}^2 = \|x_k^L\|_{U_k U_k^T}^2 + \zeta_k^2. \quad (4.23)$$

It is apparent from (4.23) that the $(U_k U_k^T)$ - and $(U_p U_p^T)$ -seminorms of the BiLQ iterates are monotonically increasing. The identity (4.19) and biorthogonality again imply that the same goes with $\|y_k^L\|$.

A lower bound on the error (4.22) can be obtained as $\|z_{k-d} - z_{k-1}\|$ for a user-defined *delay* of d iterations. Such a lower bound may be used to define a simple, though not robust, error-based stopping criterion [23].

In the symmetric case, where $V_k = U_k$ is orthogonal and $T_k = T_k^T$, the SYMMLQ iterate solves the problem

$$\underset{x}{\text{minimize}} \|x - x_\star\| \quad \text{subject to } x \in \text{Range}(AV_{k-1}), \quad (4.24)$$

which coincides with (4.21).

We now establish a similar property for x_k^C provided that it is well defined.

Proposition 2. *Let $x_\star$ be as above. If T_k is nonsingular, the k th BICG iterate x_k^C is well defined and solves*

$$\underset{x}{\text{minimize}} \|x - x_\star\|_{U_p U_p^T} \quad \text{subject to } x \in \text{Range}(V_k V_k^T A^T U_k). \quad (4.25)$$

Proof. Primal feasibility for (4.25) follows from the fact that $V_k^T A^T U_k = T_k^T$ and $x_k^C = V_k y_k^C = V_k T_k^T q$. Dual feasibility follows from (4.16) as in the proof of Proposition 1. $\square$

Contrary to Proposition 1, there is no second problem involving V_p in (4.25) because $V_p V_p^T A^T U_k = V_{k+1} T_{k,k+1}^T$ whereas $V_k V_k^T A^T U_k = T_k^T$.

4.2.7 Numerical experiments

Non-homogeneous linear PDEs with variable coefficients of the form

$$\sum_{i=1}^n \sum_{j=1}^p a_{i,j}(x) \frac{\partial^j u(x)}{\partial x_i^j} = b(x) \quad (4.26)$$

are frequent when physical phenomena are modeled in polar, cylindrical or spherical coordinates. The discretization of (4.26) often leads to a nonsymmetric square system. Such is the case with Poisson's equation $\Delta u = f$ used, for instance, to describe the gravitational or electrostatic field caused by a given mass density or charge distribution. The 2D Poisson equation in polar coordinates with Dirichlet boundary conditions is

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u(r, \theta)}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 u(r, \theta)}{\partial \theta^2} = f(r, \theta), \quad (r, \theta) \in (0, R) \times [0, 2\pi) \quad (4.27a)$$

$$u(R, \theta) = g(\theta), \quad \theta \in [0, 2\pi), \quad (4.27b)$$

where $R > 0$, the source term f and the boundary condition g are given. We discretize (4.27) using central differences with 50 discretization points for r and 50 for θ , with $g(\theta) = 0$, $f(r, \theta) = -3 \cos(\theta)$ and $R = 1$ so that (4.27) models the response of an attached circular elastic membrane to a force. The resulting matrix has size 2,500 with 12,400 nonzeros, and

is block tridiagonal with extra diagonal blocks in the northeast and southwest corners. Each block on the main diagonal is tridiagonal but not symmetric. Each off-diagonal block is diagonal. More details on the discretization used are given by Lai [54]. The exact solution is represented in Figure 4.1.

We compare BiLQ with our implementation of QMR without lookahead. We also simulate BiCG by way of the transition from x_k^L to x_k^C in Algorithm 2. Figure 4.2 reports the residual and error history of BiLQ, BiCG and QMR on (4.27). To compute $\|r_k\|$ and $\|e_k\|$, residuals $b - Ax_k$ and errors $x_k - x_\star$ are explicitly calculated at each iteration. We compute a reference solution with Julia's backslash command. We run each method with an absolute tolerance $\varepsilon_a = 10^{-10}$ and a relative tolerance $\varepsilon_r = 10^{-7}$ such that the algorithms stop when $\|r_k\| \leq \varepsilon_a + \|b\|\varepsilon_r$.

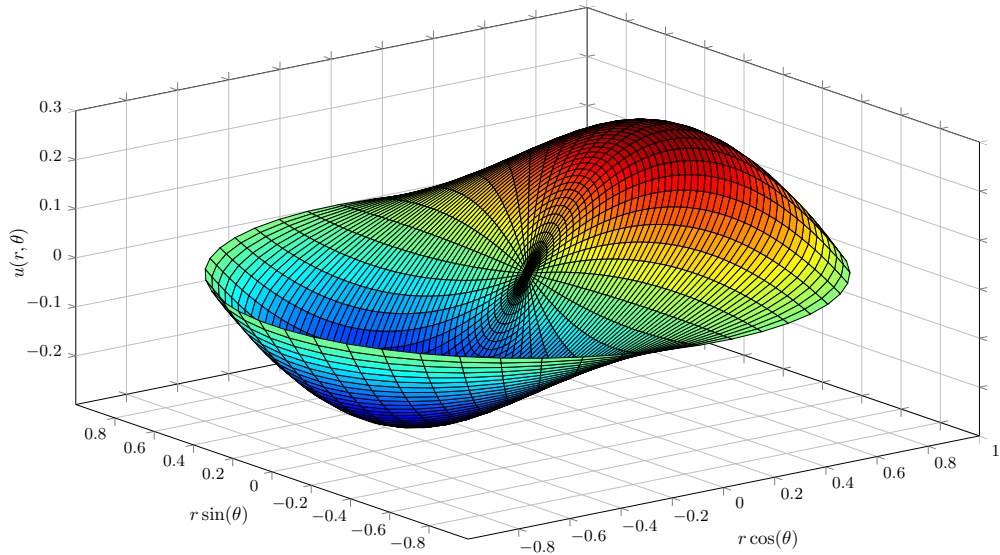


Figure 4.1 Solution $u(r, \theta) = r(1 - r) \cos(\theta)$ of (4.27) with $g(\theta) = 0$, $f(r, \theta) = -3 \cos(\theta)$ and $R = 1$.

We also compare BiLQ with BiCG and QMR on the matrices SHERMAN5 and RAEFSKY1, with their respective right-hand side, from the UFL collection of Davis and Hu [15].¹ System SHERMAN5 has size 3,312 with 20,793 nonzeros and RAEFSKY1 has size 3,242 with 293,409 nonzeros. A diagonal preconditioner is used for both systems.

Figure 4.2, Figure 4.3 and Figure 4.4 all show that in BiLQ, neither the residual nor the error are monotonic in general. They also appear more erratic than those of QMR. As in the symmetric case, both generally lag compared to those of BiCG and QMR, but are not far

1. Now the SuiteSparse Matrix Collection sparse.tamu.edu.

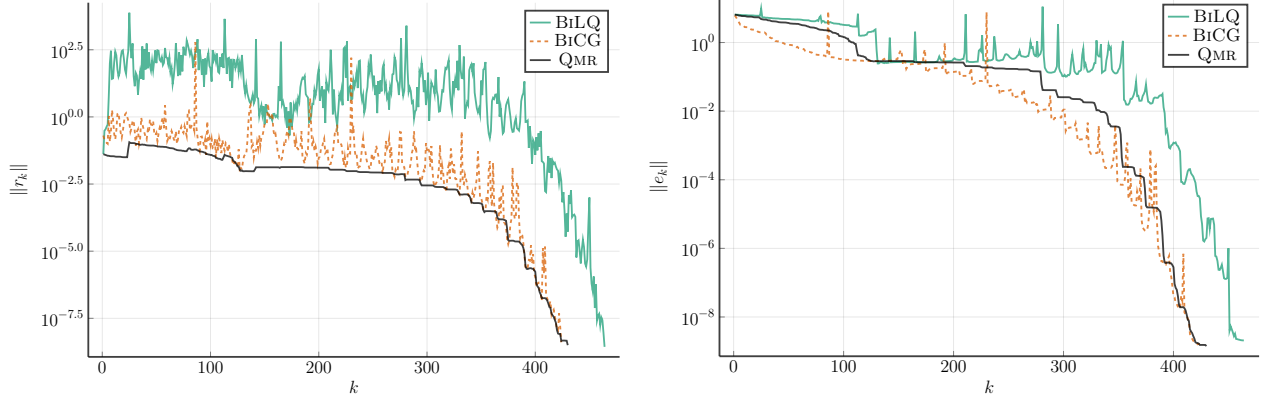


Figure 4.2 Convergence curves of BiLQ, BiCG and QMR iterates on (4.27). The figures show the residual (left) and error (right) history for each method.

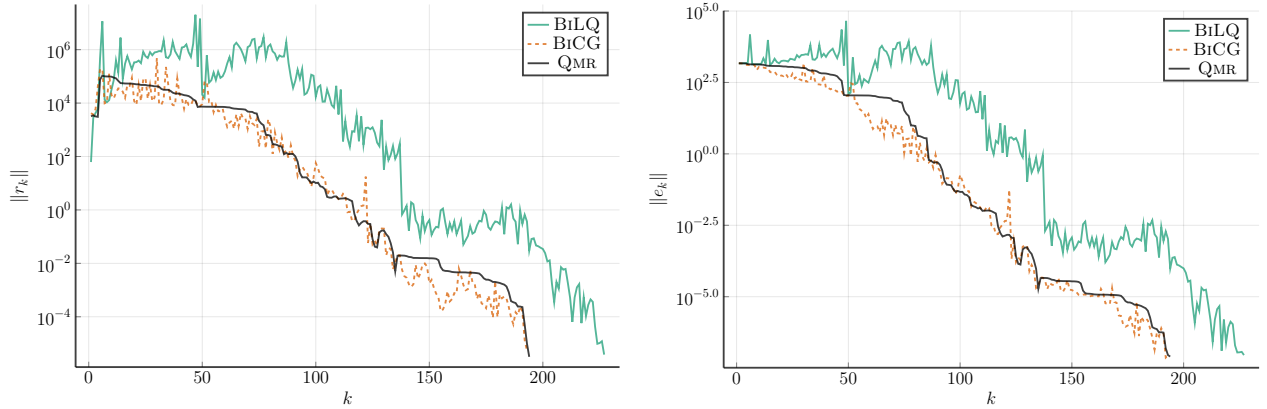


Figure 4.3 Convergence curves of BiLQ, BiCG and QMR iterates for the SHERMAN5 system. The figures show the residual (left) and error (right) history for each method.

behind. We experimented with other systems and observed the same qualitative behavior. As shown in section 4.2.6, although BiLQ is a minimum-error-type method, this error is minimized over a different space than that where x_k^L and x_k^C reside—see Proposition 1. This situation is analogous to that between SYMMLQ and CG in the symmetric case [22]. Thus, the possibility of transferring to the BiCG point, when it exists, is attractive. Because the BiCG residual is easily computable, transferring based on the residual norm is readily implemented. The determination of upper bounds on the error suitable as stopping criteria remains the subject of active research [22–24].

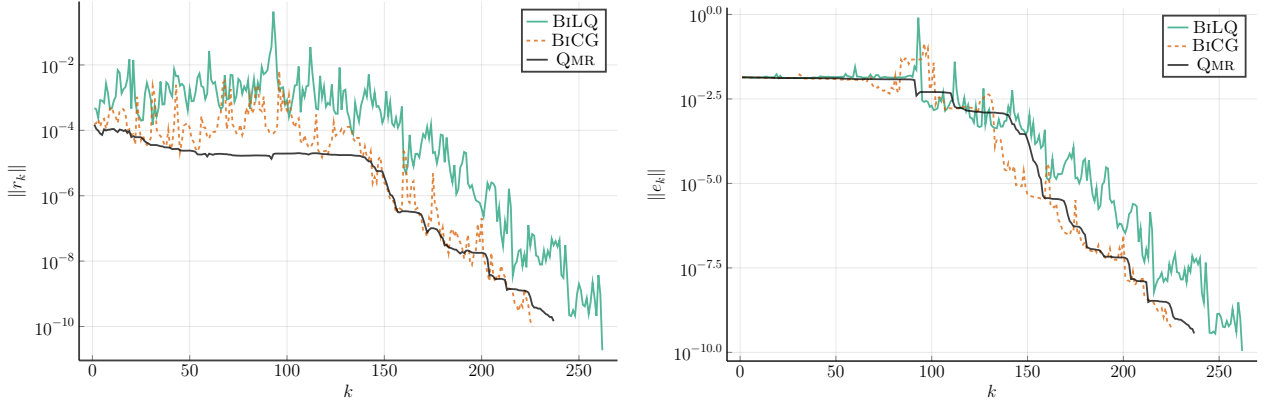


Figure 4.4 Convergence curves of BiLQ, BiCG and QMR iterates for the RAEFSKY1 system. The figures show the residual (left) and error (right) history for each method.

4.2.8 Discussion

Like QMR, the BiLQ iterate is well defined at each step even if T_k is singular, whereas x_k^C is undefined when $\bar{\delta}_k = 0$. A simple example is

$$A = \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}, \quad b = c = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

According to [Algorithm 1](#), $\beta_1 = \gamma_1 = 1$, $v_1 = u_1 = b = c$. Then $\alpha_1 = u_1^T A v_1 = 0$, $T_1 = [\alpha_1]$ is singular, and $T_1 y_1 = \beta_1$ is inconsistent. BiCG and its variants CGS [87] and BiCGSTAB [91] all fail. However, T_2 is not singular and the BiCG point exists, although we cannot compute it without lookahead. In finite-precision arithmetic, such exact breakdowns are rather rare. But near-breakdowns ($\bar{\delta}_k \approx 0$) may happen and lead to numerical instabilities in ensuing iterations. Another drawback of BiCG is that the LU decomposition of T_k might not exist without pivoting even if T_k is nonsingular, whereas (4.8) is always well defined.

4.3 Adjoint systems

Motivated by fluid dynamics applications, [Pierce and Giles \[76\]](#) describe a method for doubling the order of accuracy of estimates of integral functionals involving the solution of a PDE. Consider a well-posed linear PDE $Lu = f$ on a domain Ω subject to homogeneous boundary conditions, where L is a differential operator of the form (4.26) and $f \in L_2(\Omega)$. Suppose we wish to evaluate the functional $J(u) := \langle u, g \rangle$, where $g \in L_2(\Omega)$ and $\langle \cdot, \cdot \rangle$ represents an integral inner product on $L_2(\Omega)$. The problem may be stated equiv-

alently as evaluating the functional $\langle v, f \rangle$ where v solves the adjoint PDE $L^*v = g$ because $\langle v, f \rangle = \langle v, Lu \rangle = \langle L^*v, u \rangle = \langle g, u \rangle$.

Let the discretization of L yield the linear system $Au_D = f_D$ with D a set of points that define a grid on Ω . For certain types of PDEs and certain discretization schemes, A^T is an appropriate discretization of L^* . [Pierce and Giles \[76\]](#) provide examples with linear operators such as Poisson's equation discretized by finite differences in 1D and by finite elements in 2D, but their discretizations are symmetric. Their method also applies to cases where $A \neq A^T$ but in such cases, the discretizations of the primal and dual equations commonly differ. Therefore, there is a need for methods that solve an unsymmetric primal system and its adjoint simultaneously. [Lu and Darmofal \[56\]](#) and [Golub *et al.* \[35\]](#) were also interested in this problem for scattering amplitude evaluation. [Lu and Darmofal \[56\]](#) devise a modification of QMR in which the two initial vectors are b and c and a quasi residual is minimized for both the primal and adjoint systems via an updated QR factorization. [Golub *et al.* \[35\]](#) apply USYMQR [85] to both the primal and the adjoint system² simultaneously by updating two QR factorizations. The advantage of their approach is that it produces monotonic residuals for both systems.

Assume we use a method to compute u_D and to solve $A^T v_D = g_D$ such that $\|u - u_D\| \in \mathcal{O}(h^p)$ and $\|v - v_D\| \in \mathcal{O}(h^p)$, where h describes the grid coarseness. From u_D and v_D we compute approximations $u_h \approx u$ and $v_h \approx v$ over Ω by way of an interpolation of higher order than the discretization. Define $f_h := Lu_h$ and $g_h := L^*v_h$. Instead of $J(u) \approx \langle u_h, g \rangle$, an approximation of order p , we may obtain one of order $2p$ via the identity

$$\langle g, u \rangle = \langle g, u_h \rangle - \langle v_h, f_h - f \rangle + \langle g_h - g, u_h - u \rangle. \quad (4.28)$$

The first two terms constitute our new approximation while the remaining error term can be expressed as $\langle g_h - g, L^{-1}(f_h - f) \rangle = \mathcal{O}(h^{2p})$.

From this point, we consider, in addition to (4.1), the adjoint system

$$A^T t = c. \quad (4.29)$$

Solving simultaneously primal and dual systems can also be formulated as solving the symmetric and indefinite system

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} t \\ x \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}. \quad (4.30)$$

MINRES or MINRES-QLP [13] are prime candidates for (4.30) and will serve as a basis for

2. Although they follow [Reichel and Ye \[77\]](#) and call USYMQR the “generalized LSQR”.

comparison. We use MINRES-QLP in the numerical experiments below as it should be more stable numerically but MINRES should behave similarly because the two methods generate the same iterates in exact arithmetic on consistent systems. For this reason, we refer to MINRES and MINRES-QLP somewhat interchangeably in the sequel.

In the context of [Algorithm 1](#), we can take advantage of the two initial vectors b and c to combine BiLQ and QMR and solve both the primal and adjoint systems simultaneously at no other extra cost than that of updating solution and residual estimates. We call the resulting method BiLQR. Contrary to the approach of [Lu and Darmofal \[56\]](#), no extra factorization updates are necessary. Instead of approximating u_D and v_D by minimizing two quasi residuals, BiLQR minimizes one quasi residual and computes the second approximation via a minimum-norm subproblem.

A similar method based on the orthogonal tridiagonalization process of [Saunders *et al.* \[85\]](#) can be derived by combining USYMLQ and USYMQR, which we call TrILQR, and which is to the approach of [Golub *et al.* \[35\]](#) as BiLQR is to that of [Lu and Darmofal \[56\]](#). TrILQR remains well defined for rectangular A [\[77\]](#).

4.3.1 Description of BiLQR

BiLQR updates an approximate solution $t_{k-1}^Q = U_{k-1}f_{k-1}^Q$ of $A^T t = c$ by solving the QMR least-squares subproblem

$$\underset{f}{\text{minimize}} \quad \|T_{k-1,k}^T f - \gamma_1 e_1\| \quad \Longleftrightarrow \quad \underset{f}{\text{minimize}} \quad \left\| \begin{bmatrix} L_{k-1}^T \\ 0 \end{bmatrix} f - Q_k \gamma_1 e_1 \right\| \quad (4.31)$$

because the QR factorization of $T_{k-1,k}^T$ is readily available. Define $\bar{h}_k = Q_k \gamma_1 e_1 = (h_{k-1}, \bar{\psi}_k) = (\psi_1, \dots, \psi_{k-1}, \bar{\psi}_k)$. The components of $\bar{h}_k$ are updated according to

$$\bar{\psi}_1 = \gamma_1, \quad (4.32a)$$

$$\psi_k = c_{k+1} \bar{\psi}_k, \quad k \geq 1, \quad (4.32b)$$

$$\bar{\psi}_{k+1} = s_{k+1} \bar{\psi}_k, \quad k \geq 1. \quad (4.32c)$$

The solution of (4.31) is $f_{k-1}^Q = L_{k-1}^{-T} h_{k-1}$ and the least-squares residual norm is $|\bar{\psi}_k|$. To avoid storing U_k , we define $W_k = U_k L_k^{-T}$, which can be updated as

$$w_1 = u_1 / \delta_1, \quad (4.33a)$$

$$w_2 = (u_2 - \lambda_1 w_1) / \delta_2, \quad (4.33b)$$

$$w_k = (u_k - \lambda_{k-1} w_{k-1} - \varepsilon_{k-2} w_{k-2}) / \delta_k, \quad k \geq 3. \quad (4.33c)$$

At the next iteration, t_k^Q can be recursively updated according to

$$t_k^Q = U_k f_k^Q = U_k L_k^{-T} h_k = W_k h_k = W_{k-1} h_{k-1} + \psi_k w_k = t_{k-1}^Q + \psi_k w_k.$$

The QMR residual is

$$r_k^Q = c - A^T t_k^Q = U_{k+1} (\gamma_1 e_1 - T_{k,k+1}^T f_k^Q) = \bar{\psi}_{k+1} U_{k+1} Q_{k+1}^T e_{k+1},$$

so that

$$\|r_k^Q\| \leq \|U_{k+1}\|_F \|\bar{\psi}_{k+1} Q_{k+1}^T e_{k+1}\| \leq |\bar{\psi}_{k+1}| \sqrt{\tau_{k+1}},$$

where $\tau_{k+1} = \sum_{i=1}^{k+1} \|u_i\|^2 = \tau_k + \|u_{k+1}\|^2$. If the u_k are normalized, then $\tau_k = k$. [Algorithm 4](#) states the complete procedure.

The following result states a minimization property of the QMR residual in an iteration-dependent norm.

Proposition 3. *The $(k-1)$ th QMR iterate t_{k-1}^Q solves*

$$\underset{t}{\text{minimize}} \|c - A^T t\|_{V_k V_k^T} \text{ subject to } t \in \text{Range}(U_{k-1}). \quad (4.34)$$

In addition, $\|r_k^Q\|_{V_k V_k^T}$ is monotonically decreasing.

Proof. The set of constraints of (4.34) requires that there exist $f \in \mathbb{R}^{k-1}$ such that $t = U_{k-1} f$. By biorthogonality, the objective value at such an t can be written $\|c - A^T U_{k-1} f\|_{V_k V_k^T} = \|c - U_k T_{k-1,k}^T f\|_{V_k V_k^T} = \|\gamma_1 e_1 - T_{k-1,k}^T f\|$. We recover the subproblem (4.31).

For the second part, $\|r_k^Q\|_{V_{k+1} V_{k+1}^T} = |\bar{\psi}_{k+1}| = |s_{k+1}| |\bar{\psi}_k| = |s_{k+1}| \|r_{k-1}^Q\|_{V_k V_k^T}$. $\square$

Note that [Proposition 3](#) continues to hold if r_k^Q is measured in the $V_p V_p^T$ -norm.

4.3.2 Description of TrILQR

The [Saunders *et al.* \[85\]](#) tridiagonalization process generates sequences of vectors $\{v_k\}$ and $\{u_k\}$ such that $v_i^T v_j = \delta_{ij}$ and $u_i^T u_j = \delta_{ij}$ in exact arithmetic. The process is summarized as [Algorithm 3](#).

Algorithm 3 Orthogonal Tridiagonalization Process

Require: A, b, c

- 1: $v_0 = 0, u_0 = 0$
 - 2: $\beta_1 v_1 = b, \gamma_1 u_1 = c$ $(\beta_1, \gamma_1) > 0$ so that $\|v_1\| = \|u_1\| = 1$
 - 3: **for** $k = 1, 2, \dots$ **do**
 - 4: $q = Au_k - \gamma_k v_{k-1}, \alpha_k = v_k^T q$
 - 5: $p = A^T v_k - \beta_k u_{k-1}$
 - 6: $\beta_{k+1} v_{k+1} = q - \alpha_k v_k$ $\beta_{k+1} > 0$ so that $\|v_{k+1}\| = 1$
 - 7: $\gamma_{k+1} u_{k+1} = p - \alpha_k u_k$ $\gamma_{k+1} > 0$ so that $\|u_{k+1}\| = 1$
 - 8: **end for**
-

At the end of the k -th iteration, we have

$$AU_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T = V_{k+1} T_{k+1,k} \quad (4.35a)$$

$$A^T V_k = U_k T_k^T + \gamma_{k+1} u_{k+1} e_k^T = U_{k+1} T_{k,k+1}^T, \quad (4.35b)$$

to be compared with (4.3).

[Saunders *et al.* \[85\]](#) develop two methods based on [Algorithm 3](#). USYMLQ generates an approximation to a solution of (4.1) of the form $x_k^{\text{LQ}} = U_k y_k^{\text{LQ}}$, where $y_k^{\text{LQ}} \in \mathbb{R}^k$ solves

$$\underset{y}{\text{minimize}} \quad \|y\| \quad \text{subject to} \quad T_{k-1,k} y = \beta_1 e_1. \quad (4.36)$$

With (4.35) and (4.36), we have the following analogue of [Proposition 1](#) and (4.24).

Proposition 4. *Let x_* be the exact solution of (4.1). The k th USYMLQ iterate x_k^{LQ} solves*

$$\underset{x}{\text{minimize}} \quad \|x\| \quad \text{subject to} \quad x \in \text{Range}(U_k), \quad b - Ax \perp \text{Range}(U_{k-1}), \quad (4.37)$$

and

$$\underset{x}{\text{minimize}} \quad \|x - x_*\| \quad \text{subject to} \quad x \in \text{Range}(A^T V_{k-1}). \quad (4.38)$$

Proof. The proof is nearly identical to that of [Proposition 1](#) and relies on the fact that $r_k^{\text{LQ}} := b - Ax_k^{\text{LQ}}$ is a combination of u_k and u_{k+1} [Buttari *et al.* \[9, §3.2.2\]](#). $\square$

Similar to BICG, there exists a method based on [Algorithm 3](#) that one might name USYMCG. USYMCG generates an approximation of the form $x_k^{\text{CG}} = U_k y_k^{\text{CG}}$, where $y_k^{\text{CG}} \in \mathbb{R}^k$ solves

$$T_k y = \beta_1 e_1. \quad (4.39)$$

It shares the main disadvantage of BICG; that it fails to define an iterate when T_k is singular. However, when the iterate exists, a transfer from the USYMLQ point is possible. This method is also alluded to by [Buttari et al. \[9\]](#).

The second method, USYMQR, used to solve (4.29) generates an approximation $t_k^{\text{QR}} = V_k f_k^{\text{QR}}$ where $f_k^{\text{QR}} \in \mathbb{R}^k$ solves

$$\underset{f}{\text{minimize}} \quad \|T_{k,k+1}^T f - \gamma_1 e_1\|. \quad (4.40)$$

The following property applies to t_k^{QR} due to our assumption that (4.1) is consistent.

Proposition 5 ([Buttari et al. \[9\]](#), Theorem 1). *Assume $b \in \text{Range}(A)$. Then USYMQR finds the minimum-norm solution of*

$$\underset{t}{\text{minimize}} \quad \|A^T t - c\|.$$

Of course, A nonsingular implies that the solution to (4.29) is unique but [Proposition 5](#) applies more generally to rectangular and/or rank-deficient A .

When $A = A^T$ and $b = c$, [Algorithm 3](#) coincides with the symmetric Lanczos process, and USYMLQ and USYMQR are equivalent to SYMMLQ and MINRES [72], respectively. Besides the orthogonalization process, differences between those methods and BiLQ, BICG and QMR are the definition of $\bar{D}_k$ and W_k , and the fact that u_k and v_k are swapped. If stopping criteria are based on residual norms, expressions derived for methods based on [Algorithm 1](#) apply to methods based on [Algorithm 3](#), but their expressions can be simplified because V_k and U_k are both orthogonal:

$$\|r_k^{\text{LQ}}\| = \sqrt{\mu_k^2 + \omega_k^2}, \quad \|r_k^{\text{CG}}\| = |\rho_k| \quad \text{and} \quad \|r_k^{\text{QR}}\| = |\bar{\psi}_{k+1}|.$$

USYMLQ and USYMQR can be combined into TRI LQR to solve both the primal and adjoint system simultaneously. We summarize the complete procedure as [Algorithm 5](#) and highlight lines with differences between the two algorithms.

BiLQR and TRI LQR both need nine n -vectors: $u_k, u_{k-1}, v_k, v_{k-1}, w_k, w_{k-1}, \bar{d}_k, x_k$ and t_{k-1} whereas MINRES-QLP applied to (4.30) can be implemented with five $(2n)$ -vectors. Two more n -vectors are needed when in-place “gemv” updates are not explicitly available. [Table 4.2](#)

Algorithm 4 BiLQR**Require:** A, b, c

```

1:  $\beta_1 v_1 = b, \gamma_1 u_1 = c$ 
2:  $\alpha_1 = u_1^T A v_1$ 
3:  $\beta_2 v_2 = A v_1 - \alpha_1 v_1$ 
4:  $\gamma_2 u_2 = A^T u_1 - \alpha_1 u_1$ 
5:  $c_1 = -1, s_1 = 0, \bar{\delta}_1 = \alpha_1$ 
6:  $\eta_1 = \beta_1, \bar{d}_1 = v_1, \bar{\psi}_1 = \gamma_1$ 
7:  $x_1^L = 0, t_0^Q = 0$ 
8: for  $k = 2, 3, \dots$  do
9:    $q = A v_k - \gamma_k v_{k-1}, \alpha_k = u_k^T q$ 
10:   $p = A^T u_k - \beta_k u_{k-1}$ 
11:   $\beta_{k+1} v_{k+1} = q - \alpha_k v_k$ 
12:   $\gamma_{k+1} u_{k+1} = p - \alpha_k u_k$ 
13:   $\delta_{k-1} = (\bar{\delta}_{k-1}^2 + \gamma_k^2)^{\frac{1}{2}}$ 
14:   $c_k = \bar{\delta}_{k-1} / \delta_{k-1}$ 
15:   $s_k = \gamma_k / \delta_{k-1}$ 
16:   $\varepsilon_{k-2} = s_{k-1} \beta_k$ 
17:   $\lambda_{k-1} = -c_{k-1} c_k \beta_k + s_k \alpha_k$ 
18:   $\bar{\delta}_k = -c_{k-1} s_k \beta_k - c_k \alpha_k$ 
19:   $\zeta_{k-1} = \eta_{k-1} / \delta_{k-1}$ 
20:   $\eta_k = -\varepsilon_{k-2} \zeta_{k-2} - \lambda_{k-1} \zeta_{k-1}$ 
21:   $d_{k-1} = c_k \bar{d}_{k-1} + s_k v_k$ 
22:   $\bar{d}_k = s_k \bar{d}_{k-1} - c_k v_k$ 
23:   $\psi_{k-1} = c_k \bar{\psi}_{k-1}$ 
24:   $\bar{\psi}_k = s_k \bar{\psi}_{k-1}$ 
25:   $w_{k-1} = \frac{u_{k-1} - \lambda_{k-2} w_{k-2} - \varepsilon_{k-3} w_{k-3}}{\delta_{k-1}}$ 
26:   $x_k^L = x_{k-1}^L + \zeta_{k-1} d_{k-1}$ 
27:   $t_{k-1}^Q = t_{k-2}^Q + \psi_{k-1} w_{k-1}$ 
28: end for
29: if  $\bar{\delta}_k \neq 0$  then
30:    $\bar{\zeta}_k = \eta_k / \bar{\delta}_k$ 
31:    $x_k^C = x_k^L + \bar{\zeta}_k \bar{d}_k$ 
32: end if

```

Algorithm 5 TrILQR**Require:** A, b, c

```

 $\beta_1 v_1 = b, \gamma_1 u_1 = c$ 
 $\alpha_1 = u_1^T A v_1$ 
 $\beta_2 v_2 = A u_1 - \alpha_1 v_1$ 
 $\gamma_2 u_2 = A^T v_1 - \alpha_1 u_1$ 
 $c_1 = -1, s_1 = 0, \bar{\delta}_1 = \alpha_1$ 
 $\eta_1 = \beta_1, \bar{d}_1 = u_1, \bar{\psi}_1 = \gamma_1$ 
 $x_1^{LQ} = 0, t_0^{QR} = 0$ 
for  $k = 2, 3, \dots$  do
   $q = A u_k - \gamma_k v_{k-1}, \alpha_k = v_k^T q$ 
   $p = A^T v_k - \beta_k u_{k-1}$ 
   $\beta_{k+1} v_{k+1} = q - \alpha_k v_k$ 
   $\gamma_{k+1} u_{k+1} = p - \alpha_k u_k$ 
   $\delta_{k-1} = (\bar{\delta}_{k-1}^2 + \gamma_k^2)^{\frac{1}{2}}$ 
   $c_k = \bar{\delta}_{k-1} / \delta_{k-1}$ 
   $s_k = \gamma_k / \delta_{k-1}$ 
   $\varepsilon_{k-2} = s_{k-1} \beta_k$ 
   $\lambda_{k-1} = -c_{k-1} c_k \beta_k + s_k \alpha_k$ 
   $\bar{\delta}_k = -c_{k-1} s_k \beta_k - c_k \alpha_k$ 
   $\zeta_{k-1} = \eta_{k-1} / \delta_{k-1}$ 
   $\eta_k = -\varepsilon_{k-2} \zeta_{k-2} - \lambda_{k-1} \zeta_{k-1}$ 
   $d_{k-1} = c_k \bar{d}_{k-1} + s_k u_k$ 
   $\bar{d}_k = s_k \bar{d}_{k-1} - c_k u_k$ 
   $\psi_{k-1} = c_k \bar{\psi}_{k-1}$ 
   $\bar{\psi}_k = s_k \bar{\psi}_{k-1}$ 
   $w_{k-1} = \frac{v_{k-1} - \lambda_{k-2} w_{k-2} - \varepsilon_{k-3} w_{k-3}}{\delta_{k-1}}$ 
   $x_{k-1}^{LQ} = x_{k-1}^{LQ} + \zeta_{k-1} d_{k-1}$ 
   $t_{k-1}^{QR} = t_{k-2}^{QR} + \psi_{k-1} w_{k-1}$ 
end for
if  $\bar{\delta}_k \neq 0$  then
   $\bar{\zeta}_k = \eta_k / \bar{\delta}_k$ 
   $x_k^{CG} = x_k^{LQ} + \bar{\zeta}_k \bar{d}_k$ 
end if

```

summarizes the cost of BiLQR, TrILQR, MINRES-QLP and variants from Lu and Darmofal [56] and Golub *et al.* [35], developed for adjoint systems. An advantage of MINRES-QLP and TrILQR is that adjoint systems can be solved even if $b^T c = 0$, which is not possible with BiLQR. In addition, breakdowns $q^T p = 0$ with $p \neq 0$ and $q \neq 0$ are not a problem with TrILQR. TrILQR is similar in spirit to the recent method USYMLQR of Buttari *et al.* [9]

for solving symmetric saddle-point systems, but is slightly cheaper.

Table 4.2 Storage and cost per iteration of methods for solving (4.1) and (4.29) simultaneously.

	n -vectors	dots	scal	axpy
BiLQR	9	2	5	10
TriLQR	9	2	5	10
MINRES-QLP	10	4	8	14
Lu and Darmofal [56]	10	2	6	10
Golub <i>et al.</i> [35]	10	2	6	10

4.3.3 Applications

As a simple illustration, we consider a 1D ODE and a 2D PDE. Consider first the linear ODE with constant coefficients

$$\chi_1 u''(x) + \chi_2 u'(x) + \chi_3 u(x) = f(x) \quad x \in \Omega \quad (4.41a)$$

$$u(x) = 0 \quad x \in \partial\Omega, \quad (4.41b)$$

where $\Omega = [0, 1]$, and say we are interested in the value of the linear functional

$$J(u) = \int_{\Omega} u(x)g(x) \, dx, \quad (4.42)$$

where u solves (4.41) and $g \in L_2(\Omega)$. The adjoint equation can be derived from (4.41) using integration by parts:

$$\chi_1 v''(x) - \chi_2 v'(x) + \chi_3 v(x) = g(x) \quad x \in \Omega \quad (4.43a)$$

$$v(x) = 0 \quad x \in \partial\Omega. \quad (4.43b)$$

Note that the only difference between the primal and adjoint equations resides in the sign of odd-degree derivatives. The discussion in Section 4.3 ensures that

$$G(v) := \int_{\Omega} f(x)v(x) \, dx = J(u). \quad (4.44)$$

Consider the uniform discretization $x_i = ih$, $i = 0, \dots, N+1$, where $h = 1/(N+1)$. We use

central differences of order 2, i.e.,

$$u'(x_i) = \frac{u_{i+1} - u_{i-1}}{2h} + \mathcal{O}(h^2), \quad u''(x_i) = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + \mathcal{O}(h^2).$$

We obtain $u(x_i)$ for $x_i \in D := \{x_i \mid i \in 1, \dots, N\}$ from the tridiagonal linear system

$$\begin{bmatrix} -2\chi_1 + \chi_3 h^2 & \chi_1 + \chi_2 h & & & \\ \chi_1 - \chi_2 h & -2\chi_1 + \chi_3 h^2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \chi_1 + \chi_2 h \\ & & & \chi_1 - \chi_2 h & -2\chi_1 + \chi_3 h^2 \end{bmatrix} \begin{bmatrix} u(x_1) \\ \vdots \\ \vdots \\ u(x_N) \end{bmatrix} = h^2 \begin{bmatrix} f(x_1) \\ \vdots \\ \vdots \\ f(x_N) \end{bmatrix}.$$

More compactly, we write $Au_D = f_D$. Similarly, we compute $v(x_i)$ for $x_i \in D$ from $A^T v_D = g_D$. Next, we compute an approximation of u and v over Ω by cubic spline interpolation, and the resulting functions are denoted u_h and v_h . We require that $Lu_h = f$ and $L^*v_h = g$ on $\partial\Omega$. We subsequently obtain $f_h(x) := \chi_1 u_h''(x) + \chi_2 u_h'(x) + \chi_3 u_h(x)$. The end-point conditions of the cubic splines require that f_h coincide with f on $\partial\Omega$. Finally, we compute the improved estimate (4.28) using a three-point Gauss quadrature to approximate each

$$\int_{x_i}^{x_{i+1}} g(x) u_h(x) dx - \int_{x_i}^{x_{i+1}} v_h(x) (f_h(x) - f(x)) dx$$

on each subinterval to ensure that the numerical quadrature errors are smaller than the discretization error.

We choose $n = 50$, $\chi_1 = \chi_2 = \chi_3 = 1$, $g(x) = e^x$ and $f(x)$ such that the exact solution of (4.41) is $u_\star(x) = \sin(\pi x)$. The resulting linear system has dimension 50 with 148 nonzeros. Those parameters ensure that $J_\star = \langle g, u_\star \rangle = (\pi(e + 1))/(\pi^2 + 1)$. Figures 4.5 and 4.6 report the evolution of the residual and error on (4.1) and (4.29) for (4.41) and (4.43), respectively. BiLQR terminates in 51 iterations, TriLQR in 87 iterations and MINRES-QLP in 198 iterations. The left plot of Figure 4.7 illustrates the error in the evaluation of $J(u)$ as a function of h using the naive $J(u) \approx J(u_h)$ and improved (4.28) approximations.

The steady-state convection-diffusion equation with constant coefficients

$$\kappa_1 \Delta u(x) + \kappa_2 \nabla \cdot u(x) = f(x) \quad x \in \Omega \tag{4.45a}$$

$$u(x) = 0 \quad x \in \partial\Omega, \tag{4.45b}$$

where $f \in L_2(\Omega)$, describes the flow of heat, particles, or other physical quantities in situations where there is both diffusion and convection or advection. Assume as before that we

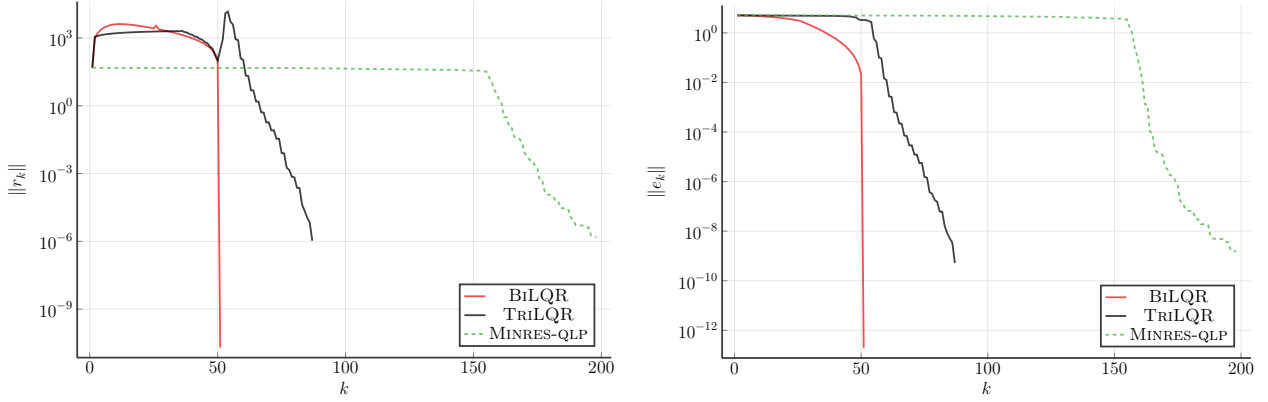


Figure 4.5 Residuals and errors norms of BiLQR, TriLQR and MINRES-QLP iterates on (4.41).

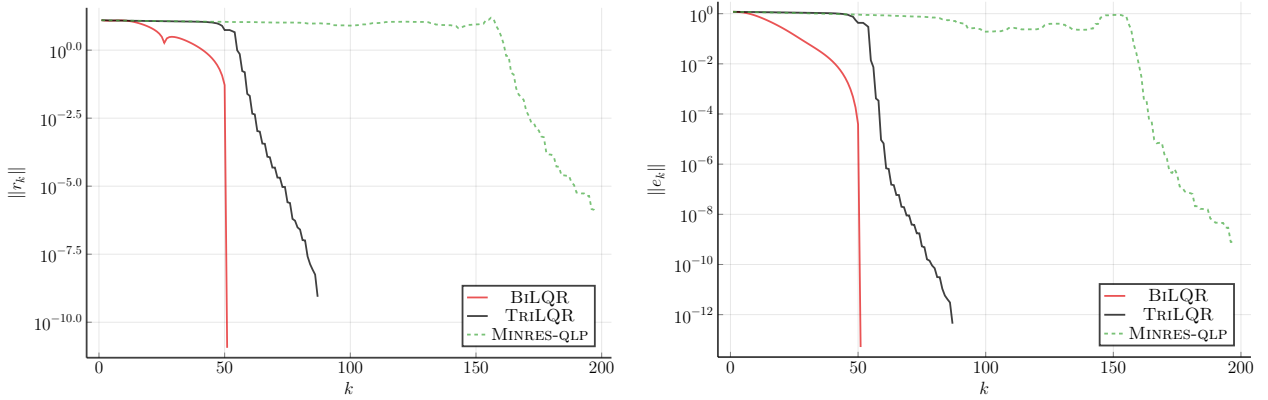


Figure 4.6 Residuals and errors norms of BiLQR, TriLQR and MINRES-QLP iterates on (4.43).

are interested in the linear functional (4.42). The adjoint equation of (4.45), again obtained via integration by parts, reads

$$\kappa_1 \Delta v(x) - \kappa_2 \nabla \cdot v(x) = g(x) \quad x \in \Omega \quad (4.46a)$$

$$v(x) = 0 \quad x \in \partial\Omega, \quad (4.46b)$$

and duality ensures (4.44).

In the case of heat transfer, $u(x)$ represents temperature and $f(x)$ sources or sinks. For example, with $g(x) = 1/\text{vol}(\Omega)$, $J(u)$ represents the average temperature in Ω .

We choose $\Omega = [0, 1] \times [0, 1]$ and discretize (4.45) on a uniform $N \times N$ grid with the finite

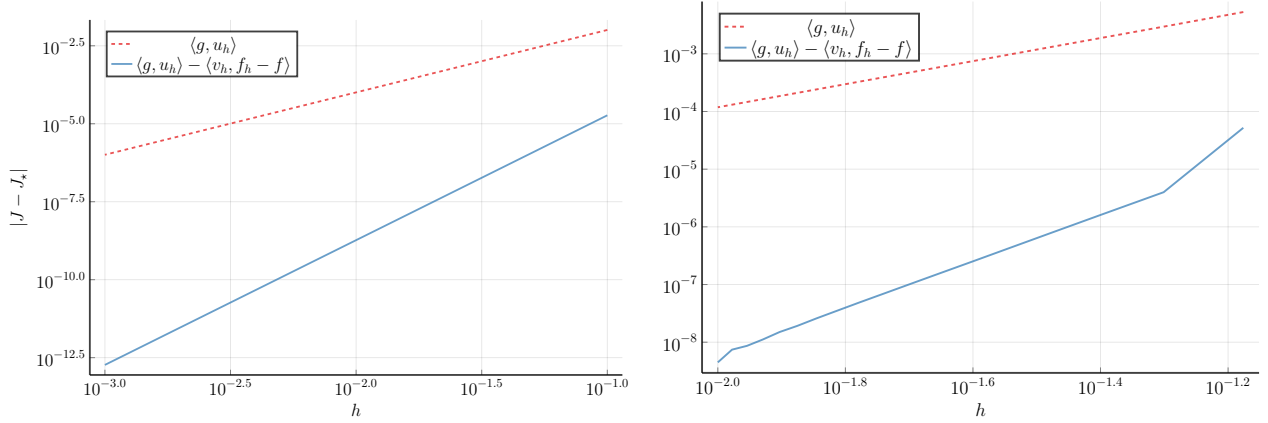


Figure 4.7 Functional evaluation errors for (4.41)–(4.43) (left) and (4.45)–(4.46) (right).

difference method such that the step along both coordinates is $h = 1/(N + 1)$. With second-order central differences for first and second derivatives, the discretized operator has the structure

$$A = \begin{bmatrix} T & D_U & & \\ D_L & T & \ddots & \\ & \ddots & \ddots & D_U \\ & & D_L & T \end{bmatrix}, \quad T = \begin{bmatrix} -4\kappa_1 & \kappa_1 + \frac{1}{2}\kappa_2 h & & \\ \kappa_1 - \frac{1}{2}\kappa_2 h & -4\kappa_1 & \ddots & \\ & \ddots & \ddots & \kappa_1 + \frac{1}{2}\kappa_2 h \\ & & \kappa_1 - \frac{1}{2}\kappa_2 h & -4\kappa_1 \end{bmatrix},$$

$D_U = \text{diag}(\kappa_1 + \frac{1}{2}\kappa_2 h)$, $D_L = \text{diag}(\kappa_1 - \frac{1}{2}\kappa_2 h)$, where the right-hand sides b and c include the h^2 term. Solutions u_D and v_D contain an approximation of u and v at grid points stored column by column. The discretization of (4.46) with the same scheme yields A^T . We compare BiLQR, TrILQR and MINRES-QLP on (4.45) and (4.46) with $\kappa_1 = 5$, $\kappa_2 = 20$, $N = 50$, $g(x, y) = e^{x+y}$ and $f(x, y)$ such that the exact solution of (4.45) is $u_*(x, y) = \sin(\pi x) \sin(\pi y)$. The resulting linear system has dimension 2,500 with 12,300 nonzeros. We use an absolute tolerance $\varepsilon_a = 10^{-10}$ and a relative tolerance $\varepsilon_r = 10^{-7}$, and terminate when both $\|r_k\| \leq \varepsilon_a + \|b\|\varepsilon_r$ for (4.1) and $\|r_k\| \leq \varepsilon_a + \|c\|\varepsilon_r$ for (4.29) hold.

Figures 4.8 and 4.9 report the evolution of the residual and error on (4.1) and (4.29) for (4.45) and (4.46), respectively. In this numerical illustration, residuals and errors are computed explicitly at each iteration as $b - Ax$, $c - A^T t$, $x - x_*$, and $t - t_*$ in order to discount errors in the approximation formulae for those expressions. In this example, BiLQR terminates in about six times fewer iterations than TrILQR and ten times fewer iterations than MINRES-QLP. Only the USYMLQ error and the USYMQR residual are monotonic. Although the

MINRES-QLP residual on (4.30) is monotonic, individual residuals on (4.1) and (4.29) are not.

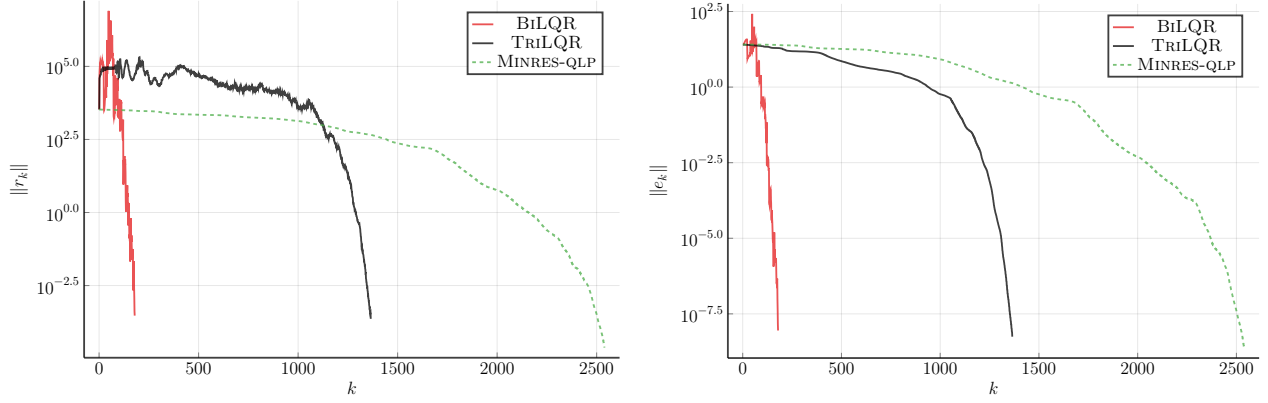


Figure 4.8 Residuals and errors norms of BiLQR, TrILQR and MINRES-QLP iterates for on (4.45).

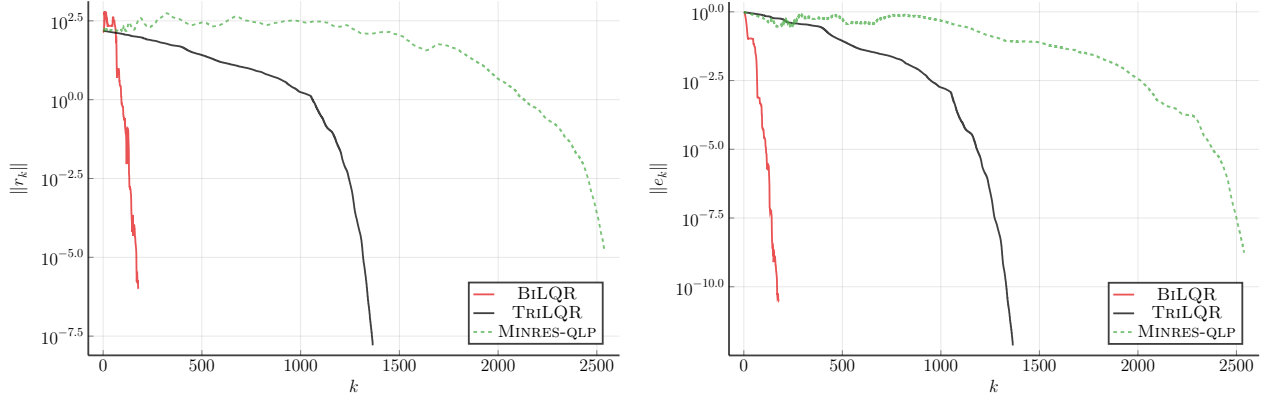


Figure 4.9 Residuals and errors norms of BiLQR, TrILQR and MINRES-QLP iterates on (4.46).

We use bicubic spline interpolation and 3×3 points Gauss quadrature to compute estimates of $J(u)$ with and without correction term. With the $u_\star$ given above, $J_\star := J(u_\star) = (\pi(e + 1))^2 / (\pi^2 + 1)^2$. The right plot of Figure 4.7 illustrates the error in the evaluation of $J(u)$ as a function of h using the naive $J(u) \approx J(u_h)$ and improved (4.28) approximations.

4.4 Discussion

BiLQ completes the family of Krylov methods based on the Lanczos biorthogonalization process, and is a natural companion to BiCG and QMR. In general, neither the error nor the residual in the Euclidean norm is monotonic. However, BiLQ is a quasi-minimum error method, its error is monotonic in the $U_p U_p^T$ -norm, which is not iteration dependent, but is unknown until the end of the biorthogonalization process. The same property holds for the QMR residual.

Contrary to the Arnoldi [2] and the Golub and Kahan [34] processes, the Lanczos biorthogonalization and orthogonal tridiagonalization processes require two initial vectors. This distinguishing feature makes them readily suited to the simultaneous solution of primal and adjoint systems. A prime application is the superconvergent estimation of integral functionals in the context of discretized ODEs and PDEs. In our experiments, we observed that BiLQR outperforms both TrILQR and MINRES-QLP applied to an augmented system in both error and residual norms.

MINRES applied to (4.30) does not produce monotonic residuals in the individual primal and adjoint systems, as is apparent from the left plot of Figure 4.9. Herzog and Soodhalter [47] devised a variant of MINRES that allows block residuals to be monitored and that could be of interest in the context of estimating integral functionals.

Our Julia implementation of BiLQ, QMR, USYMLQ, USYMQR, BiLQR, TrILQR and MINRES-QLP are available at github.com/JuliaSmoothOptimizers/Krylov.jl and can be applied in any floating-point arithmetic supported by the language.

CHAPITRE 5 ARTICLE 2: TRICG AND TRIMR: TWO ITERATIVE METHODS FOR SYMMETRIC QUASI-DEFINITE SYSTEMS

ALEXIS MONTISON AND DOMINIQUE ORBAN

Manuscript published on July 15, 2021, in the *SIAM Journal on Scientific Computing*
(SISC)

Abstract

We introduce iterative methods named TRICG and TRIMR for solving symmetric quasi-definite systems based on the orthogonal tridiagonalization process proposed by Saunders, Simon and Yip in 1988. TRICG and TRIMR are tantamount to preconditioned BLOCK-CG and BLOCK-MINRES with two right-hand sides in which the two approximate solutions are summed at each iteration, but require less storage and work per iteration. We evaluate the performance of TRICG and TRIMR on linear systems generated from the SuiteSparse Matrix Collection and from discretized and stabilized Stokes equations. We compare TRICG and TRIMR with SYMMLQ and MINRES, the recommended Krylov methods for symmetric and indefinite systems. In all our experiments, TRICG and TRIMR terminate earlier than SYMMLQ and MINRES on a residual-based stopping condition with an improvement of up to 50% in terms of number of iterations. They also terminate more reliably than BLOCK-CG and BLOCK-MINRES. Experiments in quadruple and octuple precision suggest that loss of orthogonality in the basis vectors is significantly less pronounced in TRICG and TRIMR than in BLOCK-CG and BLOCK-MINRES.

5.1 Introduction

Consider a linear system of the form

$$\begin{bmatrix} M & A \\ A^T & -N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}, \quad (5.1)$$

where $M \in \mathbb{R}^{m \times m}$ and $N \in \mathbb{R}^{n \times n}$ are symmetric positive definite, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ are not simultaneously zero, and $A \in \mathbb{R}^{m \times n}$ can have any shape.

We use the definition given by [Orban and Arioli \[70\]](#): a matrix K is symmetric quasi-definite (SQD) if $K = K^T$ and there exists a permutation matrix P such that $P^T K P$ has

the form (5.1). In particular, (5.1) arises in interior-point methods for inequality-constrained optimization [1, 32] and in the context of stabilized mixed finite elements methods [20]. SQD matrices are indefinite and nonsingular [92].

In this paper, we develop two iterative methods named TRICG and TRIMR specialized for (5.1), based on the orthogonal tridiagonalization process in elliptic norms and a closely-related formulation as a preconditioned block-Lanczos method. Relations between both processes are described in detail and we show how the preconditioned block-Lanczos process with two specific right-hand sides can generate structured Krylov bases with SQD systems. Our main motivation for developing TRICG and TRIMR comes from an absence of iterative methods for (5.1) that exploit the SQD structure when both b and c are nonzero.

The rest of this paper is organized as follows. We state the defining properties of TRICG of TRIMR and describe their implementations in detail. In a second stage, we compare TRICG and TRIMR with their block counterparts BLOCK-CG and BLOCK-MINRES as well as SYMMLQ and MINRES on two set of problems. The first set uses the SuiteSparse Matrix Collection without preconditioning. The second set is composed of discretized and stablized Stokes equations and requires preconditioning. Finally, we discuss extensions of TRICG and TRIMR and their uses outside the context of SQD systems.

Related research

Orban and Arioli [70] expose the state of the art on iterative methods for SQD systems. They explain that existing Krylov methods for symmetric indefinite systems, such as SYMMLQ and MINRES [72] or MINRES-QLP [13] do not exploit the rich structure of (5.1). CG [48] is well defined on (5.1) provided $b = 0$ or $c = 0$. Otherwise, it may break down. For example, any SQD system such that $b^T M b + 2b^T A c - c^T N c = 0$ causes breakdown on the first CG iteration. Orban and Arioli [70] show that preconditioned and regularized LSQR [73] and LSMR [28] take advantage of block structure to solve

$$\begin{bmatrix} M & A \\ A^T & -N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \quad (5.2)$$

and are equivalent to CG and MINRES applied to the normal equations

$$(A^T M^{-1} A + N)y = A^T M^{-1} b \quad \text{with} \quad x = M^{-1}(b - Ay). \quad (5.3)$$

They also show that preconditioned and regularized CRAIG [14] and CRAIGMR [70] solve

$$\begin{bmatrix} M & A \\ A^T & -N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ c \end{bmatrix}, \quad (5.4)$$

and are equivalent to CG and MINRES applied to the Schur-complement equations

$$(A^T M^{-1} A + N)y = -c \quad \text{with} \quad x = -M^{-1} A y. \quad (5.5)$$

In a similar vein, Estrin *et al.* [23, 24] develop LSLQ and LNLQ to solve (5.2) and (5.4), respectively, and explain that preconditioned and regularized variants of those methods are equivalent to SYMMLQ applied to (5.3) and to (5.5).

When $b \neq 0$ and $c \neq 0$, one possibility is to shift the right-hand side to recover (5.2) or (5.4). For instance, we can solve $-N\Delta y = c$ and add $(0, \Delta y)$ to the solution of (5.2) with right-hand side $(b - A\Delta y, 0)$. Buttari *et al.* [9] developed USYMLQR upon the orthogonal tridiagonalization process [85], henceforth the *SSY process*, for the saddle-point system

$$\begin{bmatrix} M & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}. \quad (5.6)$$

The SSY process requires two initial vectors, which makes it ideal to develop TRICG and TRIMR, two methods specialized for SQD systems in the case where both b and c are nonzero. Its close relation to a block-Lanczos process induces similarities between TRICG and the BLOCK-CG method [68] as well as TRIMR and BLOCK-MINRES methods. The block-Lanczos process was initially developed to compute eigenvalues [36]. Thereafter it was used to find nullspaces [59] and solve linear systems with multiple right-hand sides [43]. In this paper, a novel application of this process is presented for SQD linear systems with a single right-hand side.

Notation

Vectors and scalars are denoted by lowercase Latin and Greek letters, respectively. Matrices are denoted by capital Latin letters, except for 2×2 blocks, which are represented by capital Greek letters. Rare exceptions to those rules are indicated explicitly in the text. For a vector v , $\|v\|$ denotes the Euclidean norm of v , and for a symmetric and positive-definite matrix M , the M -norm of v is $\|v\|_M^2 = v^T M v$. The shorthand $v \mapsto M \setminus v$ represents an operator that returns the solution of $Mu = v$. The vector 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. The shorthand $\text{diag}(\omega_1, \dots, \omega_k)$ represents the $k \times k$ diagonal matrix with $\omega_1, \dots, \omega_k$ on its diagonal. We denote by K the SQD matrix of (5.1) and

$$K_0 := \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}, \quad H := \text{blkdiag}(M, N) = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix}, \quad B := \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix}. \quad (5.7)$$

We abuse notation and write (b, c) to represent the column vector $\begin{bmatrix} b^T & c^T \end{bmatrix}^T$.

5.2 Processes

In this section, we state the SSY process in elliptic norms, which is the foundation for the development of the methods TRICG and TRIMR, its relation with the preconditioned block-Lanczos process, and how they can accommodate regularization.

5.2.1 The orthogonal tridiagonalization process in elliptic norms

The SSY process generalized by Buttari *et al.* [9] in terms of elliptic norms defined by positive definite M and N generates sequences of vectors v_k and u_k such that $v_i^T M v_j = \delta_{ij}$ and $u_i^T N u_j = \delta_{ij}$ in exact arithmetic. The process is stated as Algorithm 6, where we use the shorthand notation $\beta_1 M v_1 = b$ to summarize the normalization operations

- set $\bar{v}_1 = b$;
- solve $M v_1 = \bar{v}_1$;
- compute $\beta_1 = (\bar{v}_1^T v_1)^{\frac{1}{2}}$;
- normalize $\bar{v}_1 \leftarrow \bar{v}_1 / \beta_1$ and $v_1 \leftarrow v_1 / \beta_1$,

and similarly for $\gamma_1 N u_1 = c$ and subsequent normalization steps. When M and N are not the identity, the above normalization operations only require solves with M and N . Occurrences of $M v_k$ and $N u_k$ in a right-hand side in Algorithm 6 simply mean that we substitute $\bar{v}_k$ and $\bar{u}_k$, respectively, so that M and N themselves are not needed. The process terminates if there exists an index k such that $\beta_{k+1} = 0$ or $\gamma_{k+1} = 0$.

We denote $V_k = \begin{bmatrix} v_1 & \dots & v_k \end{bmatrix}$ and $U_k = \begin{bmatrix} u_1 & \dots & u_k \end{bmatrix}$. After k iterations of Algorithm 6, the situation may be summarized as

$$A U_k = M V_k T_k + \beta_{k+1} M v_{k+1} e_k^T = M V_{k+1} T_{k+1,k} \quad (5.8a)$$

$$A^T V_k = N U_k T_k^T + \gamma_{k+1} N u_{k+1} e_k^T = N U_{k+1} T_{k,k+1}^T \quad (5.8b)$$

$$V_k^T M V_k = U_k^T N U_k = I_k, \quad (5.8c)$$

Algorithm 6 Orthogonal Tridiagonalization Process in Elliptic Norms

Require: $A, b, c, v \mapsto M \setminus v, u \mapsto N \setminus u$

- 1: $v_0 = 0, u_0 = 0$
 - 2: $\beta_1 M v_1 = b, \gamma_1 N u_1 = c$ $(\beta_1, \gamma_1) > 0$ so that $\|v_1\|_M = \|u_1\|_N = 1$
 - 3: **for** $k = 1, 2, \dots$ **do**
 - 4: $q = A u_k - \gamma_k M v_{k-1}, \alpha_k = v_k^T q$
 - 5: $p = A^T v_k - \beta_k N u_{k-1}$
 - 6: $\beta_{k+1} M v_{k+1} = q - \alpha_k M v_k$ $\beta_{k+1} > 0$ so that $\|v_{k+1}\|_M = 1$
 - 7: $\gamma_{k+1} N u_{k+1} = p - \alpha_k N u_k$ $\gamma_{k+1} > 0$ so that $\|u_{k+1}\|_N = 1$
 - 8: **end for**
-

where

$$T_k = \begin{bmatrix} \alpha_1 & \gamma_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \gamma_k \\ & & \beta_k & \alpha_k \end{bmatrix}, \quad T_{k,k+1} = \begin{bmatrix} T_k & \gamma_{k+1} e_k \end{bmatrix}, \quad T_{k+1,k} = \begin{bmatrix} T_k \\ \beta_{k+1} e_k^T \end{bmatrix}.$$

Equations (5.8a)–(5.8b) hold to within machine precision despite loss of orthogonality, but (5.8c) and $V_k^T A U_k = T_k$ hold only in exact arithmetic.

5.2.2 Relation with preconditioned block-Lanczos process

Saunders *et al.* [85] note Beresford Parlett’s observation that the subspaces generated by Algorithm 6 in the Euclidean norm can be viewed as the union of those generated by the block-Lanczos process applied to $A^T A$ and $A A^T$ with respective starting blocks $\begin{bmatrix} c & A^T b \end{bmatrix}$ and $\begin{bmatrix} b & A c \end{bmatrix}$. Golub *et al.* [35] pushed the observation further in terms of the block-Lanczos process applied to K_0 in (5.7). This section summarizes the latter observations and incorporates the preconditioner H .

Pasting (5.8) together results in

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix} \begin{bmatrix} V_{k+1} & 0 \\ 0 & U_{k+1} \end{bmatrix} \begin{bmatrix} 0 & T_{k+1,k} \\ T_{k,k+1}^T & 0 \end{bmatrix}, \quad (5.9)$$

which resembles a Krylov process in which basis vectors have been permuted. Let

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

denote the permutation, introduced by Paige [71], that restores the order in which Algorithm 6 generates basis vectors:

$$W_k := \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} P_k = [w_1 \ \cdots \ w_k] \quad \text{where} \quad w_k = \begin{bmatrix} v_k & 0 \\ 0 & u_k \end{bmatrix}. \quad (5.10)$$

Although w_k is a matrix, we use a lowercase letter due to the close link with the vectors v_k and u_k . The projection of K_0 in the Krylov subspace $\text{Span}\{w_1, \dots, w_k\}$ is also shuffled to symmetric block-tridiagonal form with blocks of size 2:

$$\begin{aligned} K_0 W_k &= \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix} \begin{bmatrix} V_{k+1} & 0 \\ 0 & U_{k+1} \end{bmatrix} P_{k+1} P_{k+1}^T \begin{bmatrix} 0 & T_{k+1,k} \\ T_{k,k+1}^T & 0 \end{bmatrix} P_k \\ &= H W_{k+1} F_{k+1,k}, \end{aligned} \quad (5.11)$$

where

$$F_{k+1,k} = \begin{bmatrix} \Omega_1 & \Psi_2 & & & \\ \Psi_2^T & \Omega_2 & \ddots & & \\ & \ddots & \ddots & \Psi_k & \\ & & \ddots & \Omega_k & \\ & & & \Psi_{k+1}^T & \end{bmatrix}, \quad \Omega_k = \begin{bmatrix} 0 & \alpha_k \\ \alpha_k & 0 \end{bmatrix}, \quad \Psi_k = \begin{bmatrix} 0 & \gamma_k \\ \beta_k & 0 \end{bmatrix}.$$

The two relations at line 2 of Algorithm 6 can be rearranged as

$$\begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix} \begin{bmatrix} v_1 & 0 \\ 0 & u_1 \end{bmatrix} \begin{bmatrix} \beta_1 & 0 \\ 0 & \gamma_1 \end{bmatrix} = \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} \iff H w_1 \Psi_1^T = B. \quad (5.12)$$

The identities (5.11) and (5.12) characterize the preconditioned block-Lanczos process applied to K_0 with preconditioner H and initial block B . We summarize the process as Algorithm 7 where all $w_k \in \mathbb{R}^{(n+m) \times 2}$ and $\Psi_k \in \mathbb{R}^{2 \times 2}$ are determined such that both $w_k^T H w_k = I_2$ and the equations on lines 2 and 5 are verified.

Note that Algorithm 6 and Algorithm 7 require operators that return the solution of systems with coefficient M , N and H . A specificity of Algorithm 7 is that w_k and Ψ_k are not unique. They are commonly determined from the Gram-Schmidt process: $w_k \Psi_k^T$ is the QR decomposition of the right-hand side on lines 2 and 5 of Algorithm 7. For instance, $F_{k+1,k}$ is

Algorithm 7 Preconditioned Block-Lanczos Process

Require: $K_0, B, w \mapsto H \setminus w$

- 1: $w_0 = 0$
 - 2: $Hw_1\Psi_1^T = B$
 - 3: **for** $k = 1, 2, \dots$ **do**
 - 4: $\Omega_k = w_k^T K_0 w_k$
 - 5: $Hw_{k+1}\Psi_{k+1}^T = K_0 w_k - Hw_k\Omega_k - Hw_{k-1}\Psi_k$
 - 6: **end for**
-

pentadiagonal when we force $\Psi_k = \text{diag}(\beta_k, \gamma_k)$ for all k , in which case the structure of w_k is

$$w_k = \begin{bmatrix} v_k & 0 \\ 0 & u_k \end{bmatrix} \quad (k \text{ odd}) \quad \text{and} \quad w_k = \begin{bmatrix} 0 & v_k \\ u_k & 0 \end{bmatrix} \quad (k \text{ even}).$$

5.2.3 Regularization of the preconditioned block-Lanczos process

Theorem 1. *Given the SQD matrix K and block right-hand side B , the preconditioned Krylov basis W_k generated by [Algorithm 7](#) has the form (5.10) where the vectors u_k and v_k are the same as those generated by [Algorithm 6](#) with initial vectors b and c . In addition,*

$$KW_k = HW_{k+1}S_{k+1,k}, \quad S_{k+1,k} := \begin{bmatrix} \Theta_1 & \Psi_2 & & & \\ \Psi_2^T & \Theta_2 & \ddots & & \\ & \ddots & \ddots & \Psi_k & \\ & & \ddots & \Theta_k & \\ & & & \Psi_{k+1}^T & \end{bmatrix}, \quad (5.13)$$

where

$$\Theta_k = \begin{bmatrix} 1 & \alpha_k \\ \alpha_k & -1 \end{bmatrix} \quad \text{and} \quad \Psi_k = \begin{bmatrix} 0 & \gamma_k \\ \beta_k & 0 \end{bmatrix}.$$

The scalars α_k, β_k and γ_k are those generated by [Algorithm 6](#) when it is applied to A with initial vectors b and c .

Proof. Observe that $K = K_0 + \text{blkdiag}(M, -N)$. [Algorithm 7](#) applied to K_0 generates sparse pairs w_k as in (5.10) because of the equivalence with [Algorithm 6](#). The term $\text{blkdiag}(M, -N)$ can be seen as a regularization term:

$$\begin{bmatrix} M & 0 \\ 0 & -N \end{bmatrix} w_k = Hw_k\Lambda_k \quad \text{with} \quad \Lambda_k := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \quad (5.14)$$

The identities (5.11) and (5.14) allow us to write

$$KW_k = H \left(W_k \begin{bmatrix} \Omega_1 + \Lambda_1 & \Psi_2 & & \\ & \Psi_2^T & \ddots & \ddots \\ & & \ddots & \ddots & \Psi_k \\ & & & \Psi_k^T & \Omega_k + \Lambda_k \end{bmatrix} + w_{k+1} \Psi_{k+1}^T \right), \quad (5.15)$$

which amounts to (5.13) because $\Theta_k = \Omega_k + \Lambda_k$. The Krylov basis W_k is not modified; only the projection of K in the Krylov subspace is updated. $\square$

Because of Theorem 1, the Krylov basis W_k generated by Algorithm 7 must have the sparsity structure (5.10), so that only u_k and v_k need be generated, and they may be generated directly from Algorithm 6. In addition, products with M and N are not required to generate W_k , so that the computational cost per iteration is reduced and less storage is required compared to Algorithm 7.

5.3 Methods

In this section, we develop two methods based upon Algorithm 6 in which iterates have the form

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = W_k z_k, \quad (5.16)$$

where $z_k \in \mathbb{R}^{2k}$ is defined by certain optimality properties. Thanks to (5.12) and (5.13), the residual of (5.1) at any iterate of the form (5.16) can be written

$$\begin{aligned} r_k &= \begin{bmatrix} b \\ c \end{bmatrix} - \begin{bmatrix} M & A \\ A^T & -N \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \\ &= H \left(w_1 \begin{bmatrix} \beta_1 \\ \gamma_1 \end{bmatrix} - W_{k+1} S_{k+1,k} z_k \right) \\ &= HW_{k+1} (\beta_1 e_1 + \gamma_1 e_2 - S_{k+1,k} z_k). \end{aligned} \quad (5.17)$$

In the next few sections, the particular choice of z_k yields a simplified expression for the residual.

5.3.1 Derivation of TRICG

The k -th TRICG iterate has the form (5.16) with z_k defined by the Galerkin condition

$$W_k^T r_k = 0 \iff W_k^T \left(\begin{bmatrix} b \\ c \end{bmatrix} - \begin{bmatrix} M & A \\ A^T & -N \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \right) = 0, \quad (5.18)$$

which, thanks to (5.17), can be written as

$$W_k^T H W_{k+1} (\beta_1 e_1 + \gamma_1 e_2 - S_{k+1,k} z_k) = 0.$$

By construction of the Krylov basis, $W_k^T H W_k = I_{2k}$ and $w_i^T H w_j = 0$ for $i \neq j$ in exact arithmetic. Let $S_k \in \mathbb{R}^{2k \times 2k}$ denote the leading $(2k) \times (2k)$ submatrix of $S_{k+1,k}$. This gives the TRICG subproblem:

$$S_k z_k = \beta_1 e_1 + \gamma_1 e_2. \quad (5.19)$$

Relation between TRICG and BLOCK-CG

The k -th BLOCK-CG iterate is defined by the block-Galerkin condition

$$W_k^T \left(\begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} - \begin{bmatrix} M & A \\ A^T & -N \end{bmatrix} \begin{bmatrix} x_k^b & x_k^c \\ y_k^b & y_k^c \end{bmatrix} \right) = 0, \quad (5.20)$$

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-CG subproblem is

$$S_k \begin{bmatrix} z_k^b & z_k^c \end{bmatrix} = \begin{bmatrix} \beta_1 e_1 & \gamma_1 e_2 \end{bmatrix}, \quad (5.21)$$

so that z_k^b and z_k^c solve the subproblem associated with right-hand sides $(b, 0)$ and $(0, c)$. The solutions of (5.19) and (5.21) are connected via $z_k = z_k^b + z_k^c$, and the TRICG and BLOCK-CG approximations are connected via $x_k = x_k^b + x_k^c$ and $y_k = y_k^b + y_k^c$.

An LDL^T factorization

The connection between Algorithm 6 and Algorithm 7 induces

$$S_k = P_k \begin{bmatrix} I_k & T_k \\ T_k^T & -I_k \end{bmatrix} P_k^T, \quad (5.22)$$

so that S_k is SQD, and therefore nonsingular, and (5.19) has a unique solution. Contrary to standard CG, the TRICG iterates are always well-defined. Vanderbei [92] proved that SQD

matrices are strongly factorizable, which means that, in particular, the factorization $S_k = L_k D_k L_k^T$ where L_k is unit lower triangular and D_k is diagonal always exists. Subsequently, the solution z_k of (5.19) can be determined via forward and backward sweeps, although the next section shows that computing z_k is not necessary. The factorization of S_k can be updated at each iteration. Let

$$D_k = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_{2k} \end{bmatrix}, \quad L_k = \begin{bmatrix} \Delta_1 & & & \\ \Gamma_2 & \Delta_2 & & \\ & \ddots & \ddots & \\ & & \Gamma_k & \Delta_k \end{bmatrix}, \quad \Delta_k = \begin{bmatrix} 1 & \\ \delta_k & 1 \end{bmatrix}, \quad \Gamma_k = \begin{bmatrix} & \sigma_k \\ \eta_k & \lambda_k \end{bmatrix}.$$

If we initialize $d_{-1} = d_0 = \sigma_1 = \eta_1 = \lambda_1 = 0$, individual factorization steps are obtained from the recursion formulae

$$d_{2k-1} = 1 - \sigma_k^2 d_{2k-2}, \quad k \geq 1, \quad (5.23a)$$

$$d_{2k} = -1 - \eta_k^2 d_{2k-3} - \lambda_k^2 d_{2k-2} - \delta_k^2 d_{2k-1}, \quad k \geq 1, \quad (5.23b)$$

$$\delta_k = (\alpha_k - \lambda_k \sigma_k d_{2k-2}) / d_{2k-1}, \quad k \geq 1, \quad (5.23c)$$

$$\sigma_k = \beta_k / d_{2k-2}, \quad k \geq 2, \quad (5.23d)$$

$$\eta_k = \gamma_k / d_{2k-3}, \quad k \geq 2, \quad (5.23e)$$

$$\lambda_k = -\eta_k \delta_{k-1} d_{2k-3} / d_{2k-2}, \quad k \geq 2. \quad (5.23f)$$

Update of the TRICG iterate

In order to compute the solution z_k of (5.19), we update the solution $p_k := (\pi_1, \dots, \pi_{2k})$ of $L_k D_k p_k = (\beta_1 e_1 + \gamma_1 e_2)$. The components of p_k are computed from

$$\pi_{2k-1} = \begin{cases} \beta_1 / d_1, & k = 1, \\ -\sigma_k d_{2k-2} \pi_{2k-2} / d_{2k-1}, & k \geq 2, \end{cases} \quad (5.24a)$$

$$\pi_{2k} = \begin{cases} (\gamma_1 - \delta_1 \beta_1) / d_2, & k = 1, \\ -(\delta_k d_{2k-1} \pi_{2k-1} + \lambda_k d_{2k-2} \pi_{2k-2} + \eta_k d_{2k-3} \pi_{2k-3}) / d_{2k}, & k \geq 2. \end{cases} \quad (5.24b)$$

If we were to update (x_k, y_k) directly from (5.16), all components of $z_k := (\zeta_1, \dots, \zeta_{2k})$ would have to be recomputed because of the backward substitution required to solve $L_k^T z_k = p_k$, which would require us to store W_k entirely. To avoid such drawbacks, we employ the strategy

of Paige and Saunders [72]. Let

$$G_k := W_k L_k^{-T} \iff L_k G_k^T = W_k^T, \quad G_k = \begin{bmatrix} G_k^x \\ G_k^y \end{bmatrix} = \begin{bmatrix} g_1^x & \cdots & g_{2k}^x \\ g_1^y & \cdots & g_{2k}^y \end{bmatrix}, \quad (5.25)$$

defined by $g_{-1}^x = g_0^x = g_{-1}^y = g_0^y = 0$, and the recursion

$$\begin{aligned} g_{2k-1}^x &= -\sigma_k g_{2k-2}^x + v_k, \\ g_{2k-1}^y &= -\sigma_k g_{2k-2}^y, \\ g_{2k}^x &= -\delta_k g_{2k-1}^x - \lambda_k g_{2k-2}^x - \eta_k g_{2k-3}^x, \\ g_{2k}^y &= -\delta_k g_{2k-1}^y - \lambda_k g_{2k-2}^y - \eta_k g_{2k-3}^y + u_k. \end{aligned} \quad (5.26)$$

This gives $(x_k, y_k) = W_k z_k = G_k L_k^T z_k = G_k p_k$ and the solution may be updated efficiently as

$$x_k = G_k^x p_k = x_{k-1} + \pi_{2k-1} g_{2k-1}^x + \pi_{2k} g_{2k}^x, \quad (5.27a)$$

$$y_k = G_k^y p_k = y_{k-1} + \pi_{2k-1} g_{2k-1}^y + \pi_{2k} g_{2k}^y. \quad (5.27b)$$

Residual computation

The expression (5.17) combines with (5.19) to yield the residual at the TRICG iterate:

$$\begin{aligned} r_k &= -HW_k(S_k z_k - \beta_1 e_1 - \gamma_1 e_2) - Hw_{k+1} \Psi_{k+1}^T \begin{bmatrix} e_{2k-1}^T \\ e_{2k}^T \end{bmatrix} z_k \\ &= -Hw_{k+1} \Psi_{k+1}^T \begin{bmatrix} \zeta_{2k-1} \\ \zeta_{2k} \end{bmatrix} \\ &= -Hw_{k+1} \begin{bmatrix} \beta_{k+1} \zeta_{2k} \\ \gamma_{k+1} \zeta_{2k-1} \end{bmatrix}. \end{aligned} \quad (5.28)$$

Because $L_k^T z_k = p_k$, we have $\zeta_{2k} = \pi_{2k}$ and $\zeta_{2k-1} = \pi_{2k-1} - \delta_k \pi_{2k}$. Therefore, with the relation $w_{k+1}^T H w_{k+1} = I_2$, it is natural to measure the residual in the H^{-1} -norm:

$$\|r_0\|_{H^{-1}} = \sqrt{\beta_1^2 + \gamma_1^2}, \quad (5.29a)$$

$$\|r_k\|_{H^{-1}} = \sqrt{\gamma_{k+1}^2 (\pi_{2k-1} - \delta_k \pi_{2k})^2 + \beta_{k+1}^2 \pi_{2k}^2}, \quad k \geq 1. \quad (5.29b)$$

We summarize the complete procedure as [Algorithm 8](#).

Algorithm 8 TRICG

Require: $A, b, c, v \mapsto M \setminus v, u \mapsto N \setminus u$

```

1:  $x_0 = 0, y_0 = 0$ 
2:  $g_{-1}^x = 0, g_0^x = 0, g_{-1}^y = 0, g_0^y = 0$ 
3:  $u_0 = 0, v_0 = 0$ 
4:  $\beta_1 M v_1 = b, \gamma_1 N u_1 = c$ 
5:  $\|r_0\|_{H^{-1}} = (\beta_1^2 + \gamma_1^2)^{\frac{1}{2}}$ 
6:  $d_{-1} = d_0 = \sigma_1 = \eta_1 = \lambda_1 = 0$ 
7: for  $k = 1, 2, \dots$  do
8:    $q = Au_k - \gamma_k M v_{k-1}, \alpha_k = v_k^T q$ 
9:    $p = A^T v_k - \beta_k N u_{k-1}$ 
10:   $\beta_{k+1} M v_{k+1} = q - \alpha_k M v_k$ 
11:   $\gamma_{k+1} N u_{k+1} = p - \alpha_k N u_k$ 
12:   $d_{2k-1} = 1 - \sigma_k^2 d_{2k-2}$ 
13:   $\delta_k = (\alpha_k - \lambda_k \sigma_k d_{2k-2}) / d_{2k-1}$ 
14:   $d_{2k} = -1 - \eta_k^2 d_{2k-3} - \lambda_k^2 d_{2k-2} - \delta_k^2 d_{2k-1}$ 
15:  if  $k == 1$  then
16:     $\pi_{2k-1} = \beta_k / d_{2k-1}$ 
17:     $\pi_{2k} = (\gamma_k - \delta_k \beta_k) / d_{2k}$ 
18:  else
19:     $\sigma_k = \beta_k / d_{2k-2}$ 
20:     $\eta_k = \gamma_k / d_{2k-3}$ 
21:     $\lambda_k = -(\eta_k \delta_{k-1} d_{2k-3}) / d_{2k-2}$ 
22:     $\pi_{2k-1} = -(\sigma_k \pi_{2k-2} d_{2k-2}) / d_{2k-1}$ 
23:     $\pi_{2k} = -(\delta_k \pi_{2k-1} d_{2k-1} + \lambda_k \pi_{2k-2} d_{2k-2} + \eta_k \pi_{2k-3} d_{2k-3}) / d_{2k}$ 
24:  end if
25:   $g_{2k-1}^x = v_k - \sigma_k g_{2k-2}^x$ 
26:   $g_{2k}^x = -\delta_k g_{2k-1}^x - \lambda_k g_{2k-2}^x - \eta_k g_{2k-3}^x$ 
27:   $g_{2k-1}^y = -\sigma_k g_{2k-2}^y$ 
28:   $g_{2k}^y = u_k - \delta_k g_{2k-1}^y - \lambda_k g_{2k-2}^y - \eta_k g_{2k-3}^y$ 
29:   $x_k = x_{k-1} + \pi_{2k-1} g_{2k-1}^x + \pi_{2k} g_{2k}^x$ 
30:   $y_k = y_{k-1} + \pi_{2k-1} g_{2k-1}^y + \pi_{2k} g_{2k}^y$ 
31:   $\|r_k\|_{H^{-1}} = (\gamma_{k+1}^2 (\pi_{2k-1} - \delta_k \pi_{2k})^2 + \beta_{k+1}^2 \pi_{2k}^2)^{\frac{1}{2}}$ 
32: end for

```

begin orthogonal triorthogonalization
 $(\beta_1, \gamma_1) > 0$ so that $\|v_1\|_M = \|u_1\|_N = 1$
compute $\|r_0\|_{H^{-1}}$
initialize the LDL^T factorization

continue orthogonal triorthogonalization

$\beta_{k+1} > 0$ so that $\|v_{k+1}\|_M = 1$
 $\gamma_{k+1} > 0$ so that $\|u_{k+1}\|_N = 1$
continue the LDL^T factorization
compute Δ_k
update D_k

initial solution of $L_k D_k p_k = \beta_1 e_1 + \gamma_1 e_2$

compute Γ_k

update p_k

update G_k^x

update G_k^y

update x_k

update y_k

compute $\|r_k\|_{H^{-1}}$

Storage

TRICG requires one operator-vector product with A and one with A^T per iteration. With the assumption that in-place *gemv* updates of the form $y \leftarrow Au + \gamma y$ and $y \leftarrow A^T v + \beta y$ are available, TRICG requires five n -vectors ($y_k, u_{k-1}, u_k, g_{2k-1}^y, g_{2k}^y$) and five m -vectors ($x_k, v_{k-1}, v_k, g_{2k-1}^x, g_{2k}^x$). If in-place *gemv* updates are not available, additional m - and n -vectors are required to store Au and $A^T v$. Note that A, A^T, M^{-1} and N^{-1} do not need to be formed explicitly, and can be implemented as abstract operators. For instance, we could compute the

Cholesky factorization of M and N and create abstract operators that perform the forward and backsolves. Extra m - and n -vectors could be necessary to store the results of those operators.

Properties

In this section, we formulate optimality properties of the TRICG iterates.

Proposition 6. *The k -th TRICG iterate (x_k, y_k) solves*

$$\underset{x \in \mathbb{R}^m}{\text{minimize}} \underset{y \in \mathbb{R}^n}{\text{maximize}} \mathcal{L}(x, y) \quad \text{subject to} \quad \begin{bmatrix} x \\ y \end{bmatrix} \in \text{Range}(W_k), \quad (5.30)$$

where $\mathcal{L}(x, y) = \frac{1}{2}\|x\|_M^2 - \frac{1}{2}\|y\|_N^2 + x^T A y - b^T x - c^T y$. Equivalently, (x_k, y_k) solves

$$\underset{x \in \mathbb{R}^m}{\text{minimize}} \underset{y \in \mathbb{R}^n}{\text{maximize}} \mathcal{E}(x, y) \quad \text{subject to} \quad \begin{bmatrix} x \\ y \end{bmatrix} \in \text{Range}(W_k), \quad (5.31)$$

where $\mathcal{E}(x, y)$ is the indefinite error metric

$$\mathcal{E}(x, y) := e_r^T \begin{bmatrix} M & A \\ A^T & -N \end{bmatrix} e_r, \quad e_r := (x^* - x, y^* - y),$$

and (x^*, y^*) is the exact solution of (5.1).

Proof. $\mathcal{L}(x, y)$ is strictly convex in x because $\nabla_{xx}^2 \mathcal{L}(x, y) = M \succ 0$ and strictly concave in y because $\nabla_{yy}^2 \mathcal{L}(x, y) = -N \prec 0$. Therefore, (5.30) admits a unique solution because the feasible set $\text{Range}(W_k) \neq \emptyset$. Its first-order optimality conditions are

$$W_k^T \begin{bmatrix} Mx + Ay - b \\ A^T x - Ny - c \end{bmatrix} = 0,$$

and coincide with (5.18). The rest of the proof follows from the fact that $\mathcal{L}(x, y)$ and $\mathcal{E}(x, y)$ are equal up to a constant. $\square$

Although SQD matrices are indefinite, $\mathcal{E}(x, y)$ can be seen as a metric that generalizes the energy norm. A similar metric is used by Orban and Arioli [70] in the context of their *generalized conjugate gradient method* for SQD systems. Figure 5.1 illustrates the evolution of $\mathcal{E}(x_k, y_k)$ along the TRICG iterations on problem *illc1850*, to be described in section 5.4, where oscillations from positive to negative values and decreasing amplitude are evident.

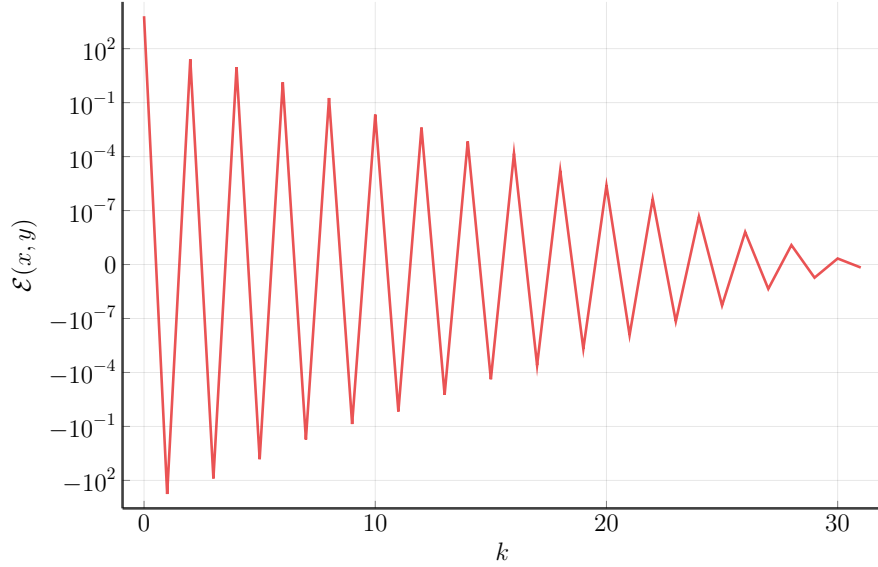


Figure 5.1 $\mathcal{E}(x, y)$ history of TRICG on illc1850, where A is 1850×712 with 8636 nonzeros.

5.3.2 Derivation of TRIMR

In the same way as TRICG is related to BLOCK-CG, the minimum residual variant TRIMR developed below is related to BLOCK-MINRES. The k -th TRIMR iterate is defined as the solution of the linear least-squares problem

$$\underset{(x_k, y_k) \in \text{Range}(W_k)}{\text{minimize}} \|r_k\|_{H^{-1}} \iff \underset{z_k \in \mathbb{R}^{2k}}{\text{minimize}} \|S_{k+1,k} z_k - \beta_1 e_1 - \gamma_1 e_2\|, \quad (5.32)$$

where the equivalence follows from (5.17). We now outline the main stages of the subproblem solution.

A QR factorization

The solution of (5.32) can be determined via the QR factorization

$$S_{k+1,k} = Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \quad (5.33)$$

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

$$R_k = \begin{bmatrix} \delta_1 & \sigma_1 & \eta_1 & \lambda_1 & \mu_1 & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \mu_{2k-4} & \\ & & & \ddots & \ddots & \ddots & \lambda_{2k-3} & \\ & & & & \ddots & \ddots & \eta_{2k-2} & \\ & & & & & \ddots & \sigma_{2k-1} & \\ & & & & & & \delta_{2k} & \end{bmatrix} \in \mathbb{R}^{(2k) \times (2k)}. \quad (5.34)$$

Below, we outline the main steps of the update and commit all details for reference.

At iteration k , four reflections are necessary to update (5.33). Together, they affect four rows and six columns of $S_{k+1,k}$. We denote their product $Q_{2k-1,2k+2}$ in (5.39) so that $Q_k^T = Q_{2k-1,2k+2} \cdots Q_{1,4}$. If we initialize $\bar{\theta}_1 := \alpha_1$, $\bar{\delta}_1 := 1$, $\bar{\delta}_2 := -1$, $\bar{\sigma}_1 := \alpha_1$, $\bar{\eta}_1 := 0$, $\bar{\lambda}_1 := \gamma_2$ and $\bar{\sigma}_2 := \beta_2$, individual factorization steps may be represented as an application of $Q_{2k-1,2k+2}$ to $Q_{k-1}^T S_{k+1,k}$:

$$\begin{array}{c} 2k-1 \\ 2k \\ 2k+1 \\ 2k+2 \end{array} \left[\begin{array}{cc|cc} 2k-1 & 2k & 2k+1 & 2k+2 & 2k+3 & 2k+4 \\ \hline \bar{\delta}_{2k-1} & \bar{\sigma}_{2k-1} & \bar{\eta}_{2k-1} & \bar{\lambda}_{2k-1} & 0 & 0 \\ \bar{\theta}_k & \bar{\delta}_{2k} & \bar{\sigma}_{2k} & 0 & 0 & 0 \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} & 0 & \gamma_{k+2} \\ \beta_{k+1} & 0 & \alpha_{k+1} & -1 & \beta_{k+2} & 0 \end{array} \right].$$

Because α_{k+1} , γ_{k+2} and β_{k+2} are not yet available at iteration k , we apply the last four reflections at iteration $k+1$ to determine all remaining coefficients of rows $2k-1$ and $2k$ of R_{k+1} and R_{k+2} :

$$\begin{array}{c} 2k-1 \\ 2k \\ 2k+1 \\ 2k+2 \end{array} \left[\begin{array}{cc|cc} 2k-1 & 2k & 2k+1 & 2k+2 & 2k+3 & 2k+4 \\ \hline \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} & \mu_{2k-1} & 0 \\ 0 & \delta_{2k} & \sigma_{2k} & \eta_{2k} & \lambda_{2k} & \mu_{2k} \\ 0 & 0 & \bar{\delta}_{2k+1} & \bar{\sigma}_{2k+1} & \bar{\eta}_{2k+1} & \bar{\lambda}_{2k+1} \\ 0 & 0 & \bar{\theta}_{k+1} & \bar{\delta}_{2k+2} & \bar{\sigma}_{2k+2} & 0 \end{array} \right].$$

Additional details about the four reflections that compose $Q_{2k-1,2k+4}$ and the factorization are available in (5.40)–(5.43).

Update of the TRIMR iterate

We can avoid computing z_k in (5.32) for the same reasons as with TRICG by updating instead $p_k := (\pi_1, \dots, \pi_{2k})$, which is defined via $R_k z_k = p_k$:

$$\bar{p}_0 := (\beta_1, \gamma_1), \quad (5.35a)$$

$$\bar{p}_k := (p_k, \bar{\pi}_{2k+1}, \bar{\pi}_{2k+2}) = Q_k^T(\beta_1 e_1 + \gamma_1 e_2), \quad k \geq 1. \quad (5.35b)$$

$\bar{p}_k$ can be easily determined from $\bar{p}_{k-1}$ because $\bar{p}_k = Q_{2k-1, 2k+2}(\bar{p}_{k-1}, 0, 0)$. Details are given in (5.44) and (5.45). We set

$$G_k := W_k R_k^{-1} \iff R_k G_k^T = W_k^T, \quad (5.36)$$

similarly to (5.25). The columns of G_k are obtained from the recursion

$$\begin{aligned} g_{2k-1}^x &= (v_k - \mu_{2k-5} g_{2k-5}^x - \lambda_{2k-4} g_{2k-4}^x - \eta_{2k-3} g_{2k-3}^x - \sigma_{2k-2} g_{2k-2}^x) / \delta_{2k-1} \\ g_{2k-1}^y &= (-\mu_{2k-5} g_{2k-5}^y - \lambda_{2k-4} g_{2k-4}^y - \eta_{2k-3} g_{2k-3}^y - \sigma_{2k-2} g_{2k-2}^y) / \delta_{2k-1} \\ g_{2k}^x &= (-\mu_{2k-4} g_{2k-4}^x - \lambda_{2k-3} g_{2k-3}^x - \eta_{2k-2} g_{2k-2}^x - \sigma_{2k-1} g_{2k-1}^x) / \delta_{2k} \\ g_{2k}^y &= (u_k - \mu_{2k-4} g_{2k-4}^y - \lambda_{2k-3} g_{2k-3}^y - \eta_{2k-2} g_{2k-2}^y - \sigma_{2k-1} g_{2k-1}^y) / \delta_{2k}, \end{aligned} \quad (5.37)$$

where we set $\eta_j, \lambda_j, \mu_j, g_j^x$ and g_j^y to zero if $j \leq 0$. Analogously to TRICG, $(x_k, y_k) = W_k z_k = G_k R_k z_k = G_k p_k$ and the solution may be updated efficiently as (5.27).

Residual computation

The definition of $\bar{p}_k$, (5.17) and (5.33) yield

$$\|r_k\|_{H^{-1}} = \|S_{k+1,k} z_k - (\beta_1 e_1 + \gamma_1 e_2)\| = \left\| \begin{bmatrix} R_k \\ 0 \end{bmatrix} z_k - \bar{p}_k \right\| = \sqrt{\bar{\pi}_{2k+1}^2 + \bar{\pi}_{2k+2}^2}. \quad (5.38)$$

The complete algorithm is stated as [Algorithm 9](#).

TriMR details

$$Q_{2k-1,2k+2} := \begin{bmatrix} 1 & & & \\ & c_{4,k} & s_{4,k} & \\ & s_{4,k} & -c_{4,k} & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & c_{3,k} & s_{3,k} \\ & & 1 \\ & s_{3,k} & -c_{3,k} \end{bmatrix} \begin{bmatrix} c_{2,k} & s_{2,k} \\ s_{2,k} & -c_{2,k} \\ & & 1 \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & c_{1,k} & s_{1,k} \\ & & 1 \\ & s_{1,k} & -c_{1,k} \end{bmatrix} \quad (5.39)$$

$$\begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 \\ 2k-1 & \left[\begin{array}{cccc} 1 & & & \\ & c_{1,k} & & s_{1,k} \\ & & 1 & \\ & s_{1,k} & & -c_{1,k} \end{array} \right] & \left[\begin{array}{cc|cc} \bar{\delta}_{2k-1} & \bar{\sigma}_{2k-1} & \bar{\eta}_{2k-1} & \bar{\lambda}_{2k-1} \\ \bar{\theta}_k & \bar{\delta}_{2k} & \bar{\sigma}_{2k} & 0 \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ \beta_{k+1} & 0 & \alpha_{k+1} & -1 \end{array} \right] & \left[\begin{array}{cc|cc} \bar{\eta}_{2k-1} & \bar{\lambda}_{2k-1} & 0 & 0 \\ \bar{\sigma}_{2k} & 0 & 0 & 0 \\ 1 & \alpha_{k+1} & 0 & \gamma_{k+2} \\ \alpha_{k+1} & -1 & \beta_{k+2} & 0 \end{array} \right] \\ 2k & & & & \\ 2k+1 & & & & \\ 2k+2 & & & & \end{array} = \begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 & 2k+3 & 2k+4 \\ \left[\begin{array}{cc|cc} \bar{\delta}_{2k-1} & \bar{\sigma}_{2k-1} & \bar{\eta}_{2k-1} & \bar{\lambda}_{2k-1} \\ \theta_k & \tilde{\delta}_{2k} & \tilde{\sigma}_{2k} & \tilde{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & g_k & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] & \left[\begin{array}{cc|cc} \tilde{\eta}_{2k-1} & \tilde{\lambda}_{2k-1} & 0 & 0 \\ \tilde{\sigma}_{2k} & \tilde{\eta}_{2k} & \tilde{\lambda}_{2k} & 0 \\ 1 & \alpha_{k+1} & 0 & \gamma_{k+2} \\ \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} & \tilde{\sigma}_{2k+2} & 0 \end{array} \right] & \left[\begin{array}{cc|cc} \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} \\ 0 & \hat{\delta}_{2k} & \hat{\sigma}_{2k} & \hat{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & g_k & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] & \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k-1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+2} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+2} & 0 \end{array} \right] \\ 2k-1 & & & & \\ 2k & & & & \\ 2k+1 & & & & \\ 2k+2 & & & & \end{array}$$

$$\theta_k = \sqrt{\bar{\theta}_k^2 + \beta_{k+1}^2}, \quad c_{1,k} = \theta_k / \bar{\theta}_k, \quad s_{1,k} = \beta_{k+1} / \bar{\theta}_k \quad (5.40a)$$

$$\tilde{\delta}_{2k} = c_{1,k} \bar{\delta}_{2k}, \quad \tilde{\sigma}_{2k} = c_{1,k} \bar{\sigma}_{2k} + s_{1,k} \alpha_{k+1}, \quad \tilde{\eta}_{2k} = -s_{1,k}, \quad \tilde{\lambda}_{2k} = s_{1,k} \beta_{k+2}, \quad (5.40b)$$

$$g_k = s_{1,k} \bar{\delta}_{2k}, \quad \tilde{\theta}_{k+1} = s_{1,k} \bar{\sigma}_{2k} - c_{1,k} \alpha_{k+1}, \quad \tilde{\delta}_{2k+2} = c_{1,k}, \quad \tilde{\sigma}_{2k+2} = -c_{1,k} \beta_{k+2}. \quad (5.40c)$$

$$\begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 \\ 2k-1 & \left[\begin{array}{cccc} c_{2,k} & s_{2,k} & & \\ s_{2,k} & -c_{2,k} & & \\ & & 1 & \\ & & & 1 \end{array} \right] & \left[\begin{array}{cc|cc} \bar{\delta}_{2k-1} & \bar{\sigma}_{2k-1} & \bar{\eta}_{2k-1} & \bar{\lambda}_{2k-1} \\ \theta_k & \tilde{\delta}_{2k} & \tilde{\sigma}_{2k} & \tilde{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & g_k & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] & \left[\begin{array}{cc|cc} \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} \\ 0 & \hat{\delta}_{2k} & \hat{\sigma}_{2k} & \hat{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & g_k & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] \\ 2k & & & & \\ 2k+1 & & & & \\ 2k+2 & & & & \end{array} = \begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 & 2k+3 & 2k+4 \\ \left[\begin{array}{cc|cc} \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} \\ 0 & \hat{\delta}_{2k} & \hat{\sigma}_{2k} & \hat{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & g_k & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] & \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k-1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+2} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+2} & 0 \end{array} \right] & \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k-1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+2} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+2} & 0 \end{array} \right] & \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k-1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+2} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+2} & 0 \end{array} \right] \\ 2k-1 & & & & \\ 2k & & & & \\ 2k+1 & & & & \\ 2k+2 & & & & \end{array}$$

$$\delta_{2k-1} = \sqrt{\bar{\delta}_{2k-1}^2 + \theta_k^2}, \quad c_{2,k} = \bar{\delta}_{2k-1}/\delta_{2k-1}, \quad s_{2,k} = \theta_k/\delta_{2k-1} \quad (5.41a)$$

$$\sigma_{2k-1} = c_{2,k}\bar{\sigma}_{2k-1} + s_{2,k}\tilde{\delta}_{2k}, \quad \eta_{2k-1} = c_{2,k}\bar{\eta}_{2k-1} + s_{2,k}\tilde{\sigma}_{2k}, \quad \lambda_{2k-1} = c_{2,k}\bar{\lambda}_{2k-1} + s_{2,k}\tilde{\eta}_{2k}, \quad \mu_{2k-1} = s_{2,k}\tilde{\lambda}_{2k}, \quad (5.41b)$$

$$\hat{\delta}_{2k} = s_{2,k}\bar{\sigma}_{2k-1} - c_{2,k}\tilde{\delta}_{2k}, \quad \hat{\sigma}_{2k} = s_{2,k}\bar{\eta}_{2k-1} - c_{2,k}\tilde{\sigma}_{2k}, \quad \tilde{\eta}_{2k} = s_{2,k}\bar{\lambda}_{2k-1} - c_{2,k}\tilde{\eta}_{2k}, \quad \tilde{\lambda}_{2k} = -c_{2,k}\tilde{\lambda}_{2k}. \quad (5.41c)$$

$$\begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 \\ 2k-1 & \left[\begin{array}{cccc} 1 & & & \\ & c_{3,k} & & s_{3,k} \\ & & 1 & \\ & s_{3,k} & & -c_{3,k} \end{array} \right] & \begin{array}{cccc} 2k-1 & 2k & 2k+1 & 2k+2 \\ \left[\begin{array}{cc|cc} \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} \\ 0 & \hat{\delta}_{2k} & \hat{\sigma}_{2k} & \hat{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & g_k & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] & \begin{array}{cccc} 2k+3 & 2k+4 & 2k+5 & 2k+6 \\ \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k+1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k+2} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+4} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+4} & 0 \end{array} \right] \end{array} \end{array} = \begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 & 2k+3 & 2k+4 \\ \left[\begin{array}{cc|cc} \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} \\ 0 & \hat{\delta}_{2k} & \hat{\sigma}_{2k} & \hat{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & 0 & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] & \begin{array}{cccc} 2k+5 & 2k+6 & 2k+7 & 2k+8 \\ \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k+1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k+2} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+4} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+4} & 0 \end{array} \right] \end{array} \end{array}$$

$$\hat{\delta}_{2k} = \sqrt{\hat{\delta}_{2k}^2 + g_k^2}, \quad c_{3,k} = \hat{\delta}_{2k}/\hat{\delta}_{2k}, \quad s_{3,k} = g_k/\hat{\delta}_{2k} \quad (5.42a)$$

$$\hat{\sigma}_{2k} = c_{3,k}\hat{\sigma}_{2k} + s_{3,k}\tilde{\theta}_{k+1}, \quad \hat{\eta}_{2k} = c_{3,k}\hat{\eta}_{2k} + s_{3,k}\tilde{\delta}_{2k+2}, \quad \hat{\lambda}_{2k} = c_{3,k}\hat{\lambda}_{2k} + s_{3,k}\tilde{\sigma}_{2k+2}, \quad (5.42b)$$

$$\bar{\theta}_{k+1} = s_{3,k}\hat{\sigma}_{2k} - c_{3,k}\tilde{\theta}_{k+1}, \quad \bar{\delta}_{2k+2} = s_{3,k}\hat{\eta}_{2k} - c_{3,k}\tilde{\delta}_{2k+2}, \quad \bar{\sigma}_{2k+2} = s_{3,k}\hat{\lambda}_{2k} - c_{3,k}\tilde{\sigma}_{2k+2}. \quad (5.42c)$$

$$\begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 \\ 2k-1 & \left[\begin{array}{cccc} 1 & & & \\ & c_{4,k} & & s_{4,k} \\ & s_{4,k} & & -c_{4,k} \\ & & & 1 \end{array} \right] & \begin{array}{cccc} 2k-1 & 2k & 2k+1 & 2k+2 \\ \left[\begin{array}{cc|cc} \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} \\ 0 & \hat{\delta}_{2k} & \hat{\sigma}_{2k} & \hat{\eta}_{2k} \\ 0 & \gamma_{k+1} & 1 & \alpha_{k+1} \\ 0 & 0 & \tilde{\theta}_{k+1} & \tilde{\delta}_{2k+2} \end{array} \right] & \begin{array}{cccc} 2k+3 & 2k+4 & 2k+5 & 2k+6 \\ \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k+1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k+2} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+4} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+4} & 0 \end{array} \right] \end{array} \end{array} = \begin{array}{cccc} & 2k-1 & 2k & 2k+1 & 2k+2 & 2k+3 & 2k+4 \\ \left[\begin{array}{cc|cc} \delta_{2k-1} & \sigma_{2k-1} & \eta_{2k-1} & \lambda_{2k-1} \\ 0 & \delta_{2k} & \sigma_{2k} & \eta_{2k} \\ 0 & 0 & \bar{\delta}_{2k+1} & \bar{\sigma}_{2k+1} \\ 0 & 0 & \bar{\theta}_{k+1} & \bar{\delta}_{2k+2} \end{array} \right] & \begin{array}{cccc} 2k+5 & 2k+6 & 2k+7 & 2k+8 \\ \left[\begin{array}{cc|cc} \mu_{2k-1} & 0 & \mu_{2k+1} & 0 \\ \hat{\lambda}_{2k} & 0 & \hat{\lambda}_{2k+2} & 0 \\ 0 & \gamma_{k+2} & 0 & \gamma_{k+4} \\ \tilde{\sigma}_{2k+2} & 0 & \tilde{\sigma}_{2k+4} & 0 \end{array} \right] \end{array} \end{array}$$

$$\delta_{2k} = \sqrt{\delta_{2k}^{\circ 2} + \gamma_{k+1}^2}, \quad c_{4,k} = \delta_{2k}^{\circ 2} / \delta_{2k}, \quad s_{4,k} = \gamma_{k+1} / \delta_{2k} \quad (5.43a)$$

$$\sigma_{2k} = c_{4,k} \mathring{\sigma}_{2k} + s_{4,k}, \quad \eta_{2k} = c_{4,k} \mathring{\eta}_{2k} + s_{4,k} \alpha_{k+1}, \quad \lambda_{2k} = c_{4,k} \mathring{\lambda}_{2k}, \quad \mu_{2k} = s_{4,k} \gamma_{k+2}, \quad (5.43b)$$

$$\bar{\delta}_{2k+1} = s_{4,k} \mathring{\sigma}_{2k} - c_{4,k}, \quad \bar{\sigma}_{2k+1} = s_{4,k} \mathring{\eta}_{2k} - c_{4,k} \alpha_{k+1}, \quad \bar{\eta}_{2k+1} = s_{4,k} \mathring{\lambda}_{2k}, \quad \bar{\lambda}_{2k+1} = -c_{4,k} \gamma_{k+2}. \quad (5.43c)$$

$$\begin{bmatrix} 1 & & & \\ & c_{1,k} & & s_{1,k} \\ & & 1 & \\ & s_{1,k} & & -c_{1,k} \end{bmatrix} \begin{bmatrix} \bar{\pi}_{2k-1} \\ \bar{\pi}_{2k} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \bar{\pi}_{2k-1} \\ \tilde{\pi}_{2k} \\ 0 \\ \tilde{\pi}_{2k+2} \end{bmatrix} \quad \begin{bmatrix} c_{2,k} & s_{2,k} & & \\ s_{2,k} & -c_{2,k} & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \bar{\pi}_{2k-1} \\ \tilde{\pi}_{2k} \\ 0 \\ \tilde{\pi}_{2k+2} \end{bmatrix} = \begin{bmatrix} \pi_{2k-1} \\ \hat{\pi}_{2k} \\ 0 \\ \tilde{\pi}_{2k+2} \end{bmatrix}$$

$$\tilde{\pi}_{2k} = c_{1,k} \bar{\pi}_{2k}, \quad \tilde{\pi}_{2k+2} = s_{1,k} \bar{\pi}_{2k}, \quad \pi_{2k-1} = c_{2,k} \bar{\pi}_{2k-1} + s_{2,k} \tilde{\pi}_{2k}, \quad \hat{\pi}_{2k} = s_{2,k} \bar{\pi}_{2k-1} - c_{2,k} \tilde{\pi}_{2k} \quad (5.44)$$

$$\begin{bmatrix} 1 & & & \\ & c_{3,k} & & s_{3,k} \\ & & 1 & \\ & s_{3,k} & & -c_{3,k} \end{bmatrix} \begin{bmatrix} \pi_{2k-1} \\ \hat{\pi}_{2k} \\ 0 \\ \tilde{\pi}_{2k+2} \end{bmatrix} = \begin{bmatrix} \pi_{2k-1} \\ \mathring{\pi}_{2k} \\ 0 \\ \bar{\pi}_{2k+2} \end{bmatrix} \quad \begin{bmatrix} 1 & & & \\ & c_{4,k} & & s_{4,k} \\ & s_{4,k} & & -c_{4,k} \\ & & & 1 \end{bmatrix} \begin{bmatrix} \pi_{2k-1} \\ \mathring{\pi}_{2k} \\ 0 \\ \bar{\pi}_{2k+2} \end{bmatrix} = \begin{bmatrix} \pi_{2k-1} \\ \pi_{2k} \\ \bar{\pi}_{2k+1} \\ \bar{\pi}_{2k+2} \end{bmatrix}$$

$$\mathring{\pi}_{2k} = c_{3,k} \hat{\pi}_{2k} + s_{3,k} \tilde{\pi}_{2k+2}, \quad \bar{\pi}_{2k+2} = s_{3,k} \hat{\pi}_{2k} - c_{3,k} \tilde{\pi}_{2k+2}, \quad \pi_{2k} = c_{4,k} \mathring{\pi}_{2k}, \quad \bar{\pi}_{2k+1} = s_{4,k} \mathring{\pi}_{2k} \quad (5.45)$$

Scalars decorated by a hat, a tilde or a ring are updated at the current iteration. Scalars decorated by a bar will be updated at the next iteration. Scalars without any decoration have been updated to their final value.

Algorithm 9 TRIMR

Require: $A, b, c, v \mapsto M \setminus v, u \mapsto N \setminus u$

```

1:  $x_0 = 0, y_0 = 0$ 
2:  $g_{-3}^x = 0, g_{-2}^x = 0, g_{-1}^x = 0, g_0^x = 0$ 
3:  $g_{-3}^y = 0, g_{-2}^y = 0, g_{-1}^y = 0, g_0^y = 0$ 
4:  $u_0 = 0, v_0 = 0$  begin orthogonal triorthogonalization
5:  $\beta_1 M v_1 = b, \gamma_1 N u_1 = c$   $(\beta_1, \gamma_1) > 0$  so that  $\|v_1\|_M = \|u_1\|_N = 1$ 
6:  $\|r_0\|_{H^{-1}} = (\beta_1^2 + \gamma_1^2)^{\frac{1}{2}}$  compute  $\|r_0\|_{H^{-1}}$ 
7: for  $k = 1, 2, \dots$  do
8:    $q = A u_k - \gamma_k M v_{k-1}, \alpha_k = v_k^T q$  continue orthogonal triorthogonalization
9:    $p = A^T v_k - \beta_k N u_{k-1}$ 
10:   $\beta_{k+1} M v_{k+1} = q - \alpha_k M v_k$   $\beta_{k+1} > 0$  so that  $\|v_{k+1}\|_M = 1$ 
11:   $\gamma_{k+1} N u_{k+1} = p - \alpha_k N u_k$   $\gamma_{k+1} > 0$  so that  $\|u_{k+1}\|_N = 1$ 
12:  if  $k == 1$  then
13:     $\bar{\theta}_1 = \alpha_1, \bar{\delta}_1 = 1, \bar{\delta}_2 = -1$  initialize the QR factorization
14:     $\bar{\sigma}_1 = \alpha_1, \bar{\eta}_1 = 0, \bar{\lambda}_1 = \beta_2, \bar{\sigma}_2 = \gamma_2$ 
15:  else
16:    Compute  $\eta_{2k-3}, \lambda_{2k-3}, \mu_{2k-3}, \sigma_{2k-2}, \eta_{2k-2}, \lambda_{2k-2}, \mu_{2k-2}$  update  $R_k$ 
17:    Compute  $\theta_k, \delta_{2k-1}, \delta_{2k}, \bar{\sigma}_{2k-1}, \bar{\eta}_{2k-1}, \lambda_{2k-1}, \bar{\sigma}_{2k}$ 
18:  end if
19:  Compute  $Q_{2k-1, 2k+2}, \delta_{2k-1}, \sigma_{2k-1}, \delta_{2k}$  continue the QR factorization
20:  Compute  $\pi_{2k-1}, \pi_{2k}, \bar{\pi}_{2k+1}, \bar{\pi}_{2k+2}$  update  $\bar{p}_k$ 
21:   $g_{2k-1}^x = (v_k - \mu_{2k-5} g_{2k-5}^x - \lambda_{2k-4} g_{2k-4}^x - \eta_{2k-3} g_{2k-3}^x - \sigma_{2k-2} g_{2k-2}^x) / \delta_{2k-1}$ 
22:   $g_{2k}^x = -(\mu_{2k-4} g_{2k-4}^x + \lambda_{2k-3} g_{2k-3}^x + \eta_{2k-2} g_{2k-2}^x + \sigma_{2k-1} g_{2k-1}^x) / \delta_{2k}$  update  $G_k^x$ 
23:   $g_{2k-1}^y = -(\mu_{2k-5} g_{2k-5}^y + \lambda_{2k-4} g_{2k-4}^y + \eta_{2k-3} g_{2k-3}^y + \sigma_{2k-2} g_{2k-2}^y) / \delta_{2k-1}$ 
24:   $g_{2k}^y = (u_k - \mu_{2k-4} g_{2k-4}^y - \lambda_{2k-3} g_{2k-3}^y - \eta_{2k-2} g_{2k-2}^y - \sigma_{2k-1} g_{2k-1}^y) / \delta_{2k}$  update  $G_k^y$ 
25:   $x_k = x_{k-1} + \pi_{2k-1} g_{2k-1}^x + \pi_{2k} g_{2k}^x$  update  $x_k$ 
26:   $y_k = y_{k-1} + \pi_{2k-1} g_{2k-1}^y + \pi_{2k} g_{2k}^y$  update  $y_k$ 
27:   $\|r_k\|_{H^{-1}} = (\bar{\pi}_{2k+1}^2 + \bar{\pi}_{2k+2}^2)^{\frac{1}{2}}$  compute  $\|r_k\|_{H^{-1}}$ 
28: end for

```

Storage

TRIMR has the same storage requirements as TRICG plus two n -vectors (g_{2k-2}^y, g_{2k-3}^y) and two m -vectors (g_{2k-2}^x, g_{2k-3}^x). All other vectors are identical to those in TRICG.

5.4 Implementation and numerical experiments

We evaluate the performance of TRICG and TRIMR on SQD systems generated from rectangular matrices A obtained from the UFL collection of Davis and Hu [15].¹ We implemented

1. Now the SuiteSparse Matrix Collection sparse.tamu.edu.

Algorithm 8 and Algorithm 9 in Julia² [5], version 1.5. Both algorithms are available as part of the `Krylov.jl` collection of Krylov methods [60].

Because standard CG may break down when applied to (5.1), we compare the evolution of the TRICG residual to that of SYMMLQ, whose iterates are always well defined. Similarly, we compare the evolution of the TRIMR residual to that of MINRES. In order to evaluate benefits of TRICG and TRIMR in terms of loss of orthogonality along the iterations, we also compare the evolution of TRICG and TRIMR residuals to those of BLOCK-CG and BLOCK-MINRES, respectively, applied to K with block right-hand side B where the two approximate solutions are summed at the last iteration. SYMMLQ, BLOCK-CG, MINRES and BLOCK-MINRES are run with preconditioner H .

In our first set of experiments, we set M and N to the identity. Thus the H^{-1} -norm is simply the Euclidean norm. The right-hand side (b, c) is generated such that the exact solution of (5.1) is the vector of ones. Residuals $r_k = b - Ax_k$ are calculated explicitly at each iteration in order to evaluate $\|r_k\|$ instead of using (5.28) or (5.38). Each algorithm stops as soon as $\|r_k\| \leq \varepsilon_a + \|(b, c)\|\varepsilon_r$ with absolute tolerance $\varepsilon_a = 10^{-12}$ and relative tolerance $\varepsilon_r = 10^{-10}$.

Figures 5.2 and 5.3 report residual histories on matrices arising from linear optimization. In all cases, the TRICG and TRIMR residuals attain the required tolerance in around half the number of iterations of SYMMLQ and MINRES, respectively. We also note that the TRICG and BLOCK-CG residuals are close, but not quite superposed, as are the MINRES and BLOCK-MINRES residuals. These results are encouraging if ones wishes to employ TRICG or TRIMR to solve the linear systems arising at each iteration of a numerical method for constrained optimization, including interior-point methods, where the systems have the form of those just tested. The results also suggest that orthogonality is not lost quite as fast in TRICG and TRIMR as it is in BLOCK-CG and BLOCK-MINRES.

Figures 5.4 and 5.5 report residual histories on matrices arising from least-squares problems. In all cases, TRICG and TRIMR require fewer iterations than SYMMLQ and MINRES. On these two problems, the residuals of the block methods are nearly superposed. We observe on our test problems that TRICG and TRIMR perform fewer iterations when the singular values of A are clustered. However, a deeper analysis is required to confirm this empirical observation.

We simulate the behavior of the six methods in exact arithmetic in hopes to compare loss of orthogonality empirically. Figure 5.6 reports residual histories obtained when running each method on `lp_osa_07` entirely in quadruple and octuple precision, and should be compared

2. julialang.org

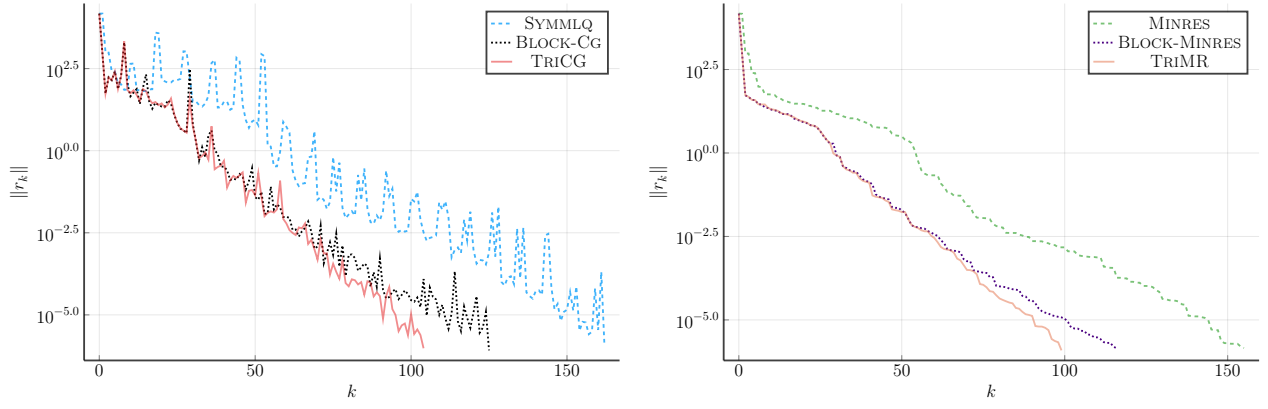


Figure 5.2 Residual history of SYMMLQ, BLOCK-CG, TriCG, MINRES, BLOCK-MINRES and TriMR on lp_czprob, where A is 929×3562 with 10708 nonzeros.

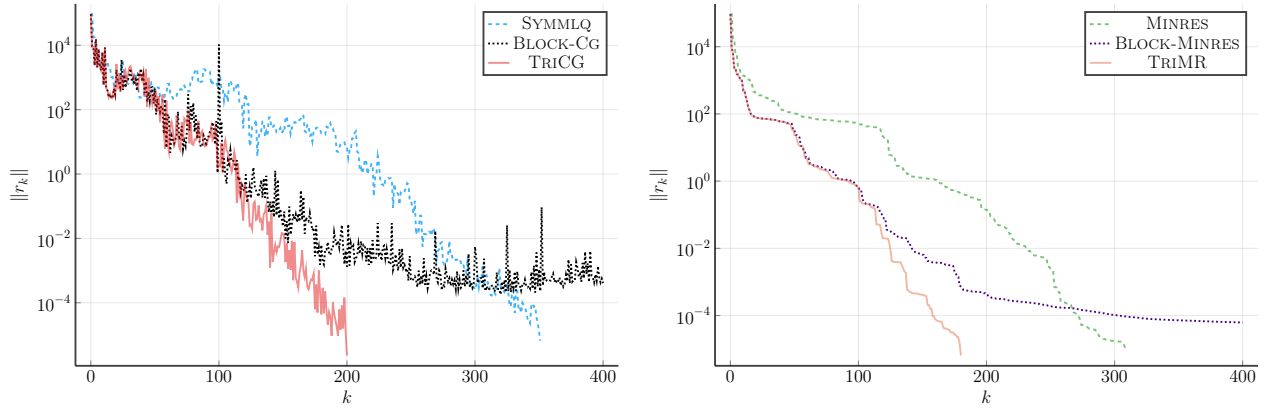


Figure 5.3 Residual history of SYMMLQ, BLOCK-CG, TriCG, MINRES, BLOCK-MINRES and TriMR on lp_osa_07, where A is 1108×25067 with 144812 nonzeros.

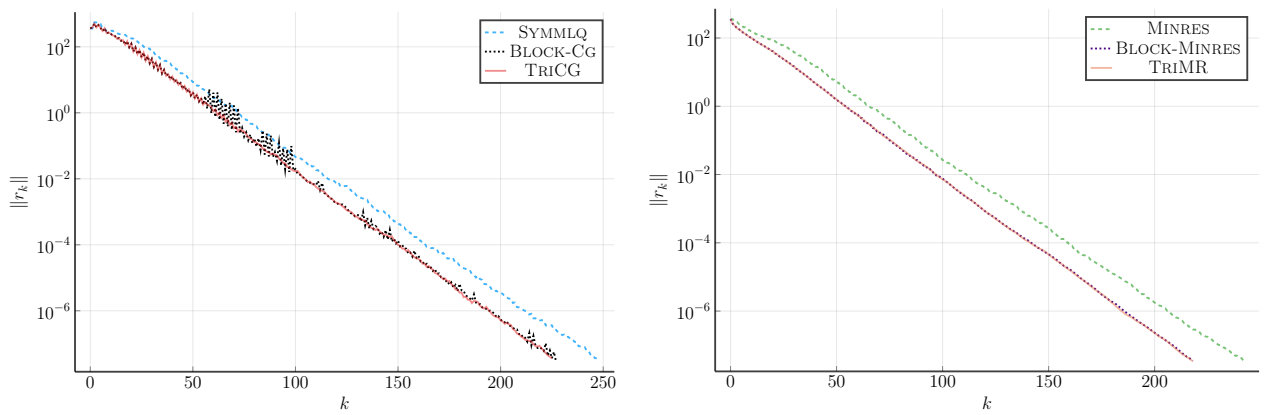


Figure 5.4 Residual history of SYMMLQ, BLOCK-CG, TriCG, MINRES, BLOCK-MINRES and TriMR on Maragal_6, where A is 21255×10152 with 537694 nonzeros.

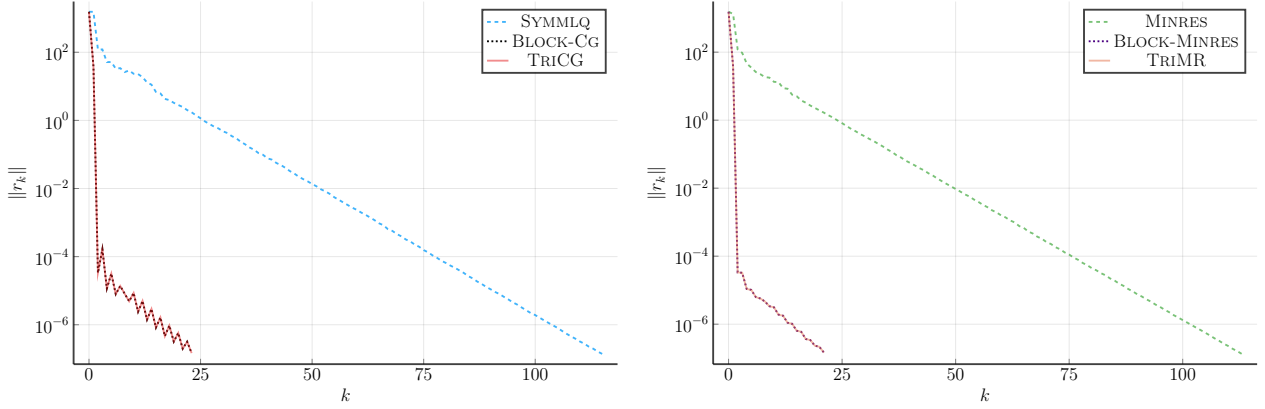


Figure 5.5 Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR on landmark, where A is 71952×2704 with 1146848 nonzeros.

with Figure 5.3. This time, the TRICG and BLOCK-CG residuals are nearly superposed, as are the TRIMR and BLOCK-MINRES residuals as the increased accuracy of arithmetic mitigates loss of orthogonality. The block methods continue to require around half as many iterations as the standard methods. However, we note an additional phenomenon: doubling the number of digits reduces the number of iterations by a factor of approximately two.

In a second set of experiments, we run all six methods on discretized and stabilized Stokes equations generated by the MATLAB package IFISS, version 3.6, of Elman *et al.* [19]. Whenever the discrete velocity and pressure belong to finite-element spaces that do not satisfy the *inf-sup*, or Ladyzhenskaya-Babuška-Brezzi (LBB), stability conditions [7], a nonzero and negative semi-definite stabilization term $-N$ is inserted in the bottom block of (5.6). It is the case with the unstable 2D finite-element pairs Q_1-P_0 and Q_1-Q_1 , which we use on a test problem from IFISS. In order to obtain an SQD system, we add $10^{-5}I$ to N . For this set of problems, M and N are not identity operators, and each algorithm stops as soon as $\|r_k\|_{H^{-1}} \leq \varepsilon_a + \|(b, c)\|_{H^{-1}}\varepsilon_r$ with the same tolerances as above.

Figures 5.7 and 5.8 report residual histories. TRICG and TRIMR reach the prescribed tolerance before SYMMLQ and MINRES with around 25% fewer iteration. These results suggest that TRICG and TRIMR may be of interest to solve other PDEs whose discretization leads to (5.1), such as the Reissner-Mindlin plate model in linear elasticity [8].

Additional numerical illustrations of TriCG and TriMR

This subsection contains further numerical comparisons between TRICG, TRIMR, BLOCK-CG, BLOCK-MINRES, SYMMLQ and MINRES. Figure 5.9 reports residual histories in dou-

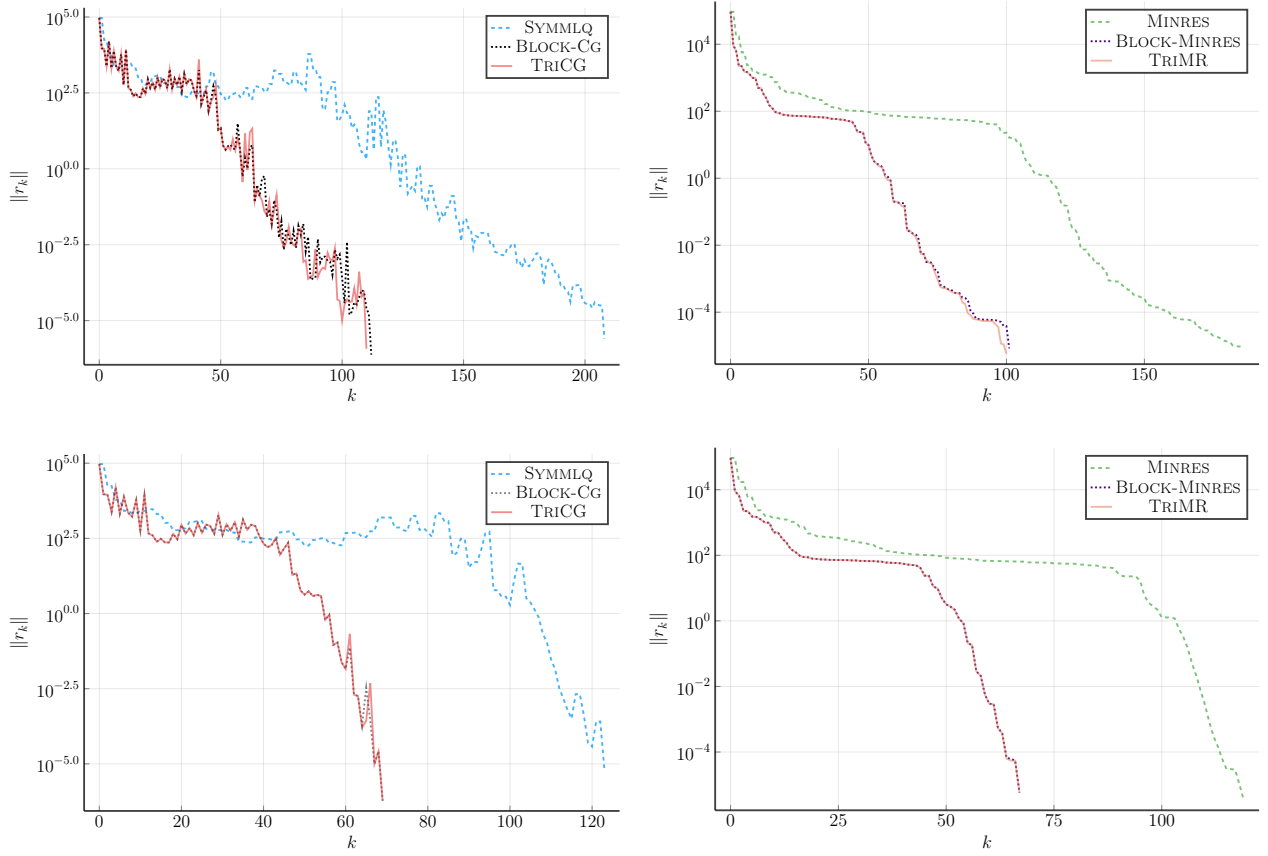


Figure 5.6 Residual history of SYMMLQ, BLOCK-CG, TriCG, MINRES, BLOCK-MINRES and TriMR iterates on `lp_osa_07` in quadruple (top) and octuple precision (bottom).

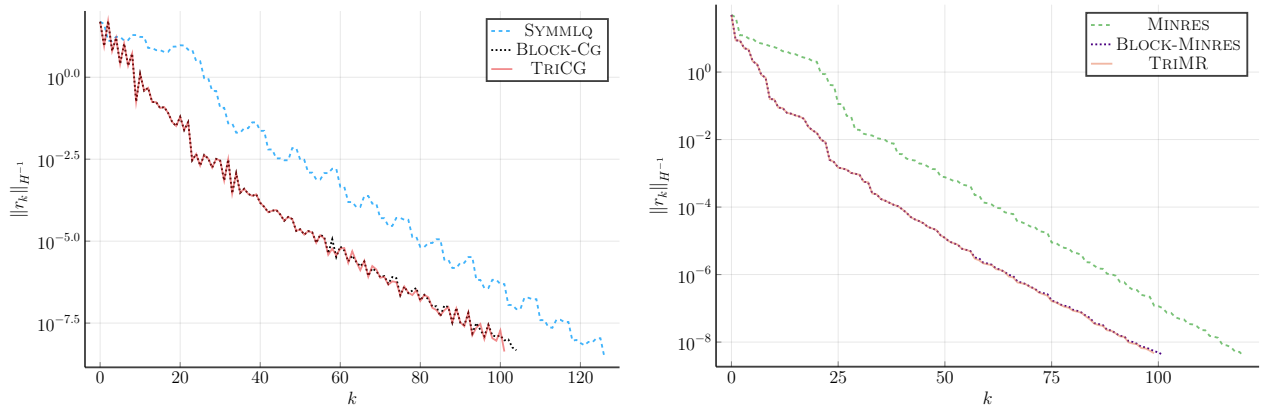


Figure 5.7 Residual history of SYMMLQ, BLOCK-CG, TriCG, MINRES, BLOCK-MINRES and TriMR on `channel_domain` problem with Q_1 - P_0 discretization. The discretized linear system has size 12546×12546 with 147742 nonzeros.

ble precision on another underdetermined system from optimization, where BLOCK-CG and BLOCK-MINRES do not converge, presumably due to excessive loss of orthogonality. Fig-

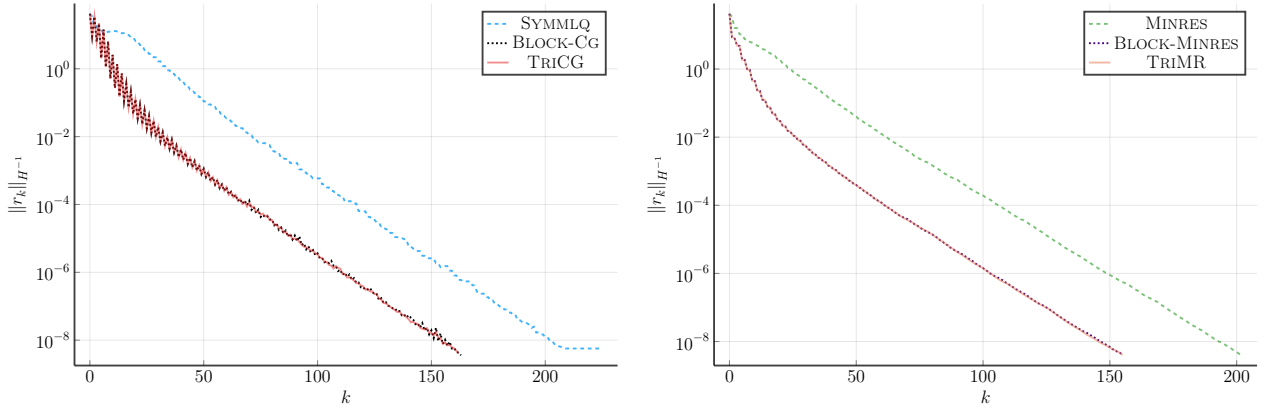


Figure 5.8 Residual history of SYMMLQ, BLOCK-CG, TriCG, MINRES, BLOCK-MINRES and TriMR on channel_domain problem with Q_1 - Q_1 discretization. The discretized linear system has size 12675×12675 with 242381 nonzeros.

Figure 5.10 corresponds to a well-conditioned overdetermined system from a least-squares application, where the residuals of the block methods nearly coincide. Figures 5.11 and 5.12 are Stokes systems. Finally, Figure 5.13 is a rather dramatic example of an underdetermined system from optimization where only TriCG and TriMR converge in double precision. As the accuracy increases, BLOCK-CG and BLOCK-MINRES converge and nearly coincide with TriCG and TriMR. Moreover, as the number of digits doubles, the number of iterations to converge is roughly halved.

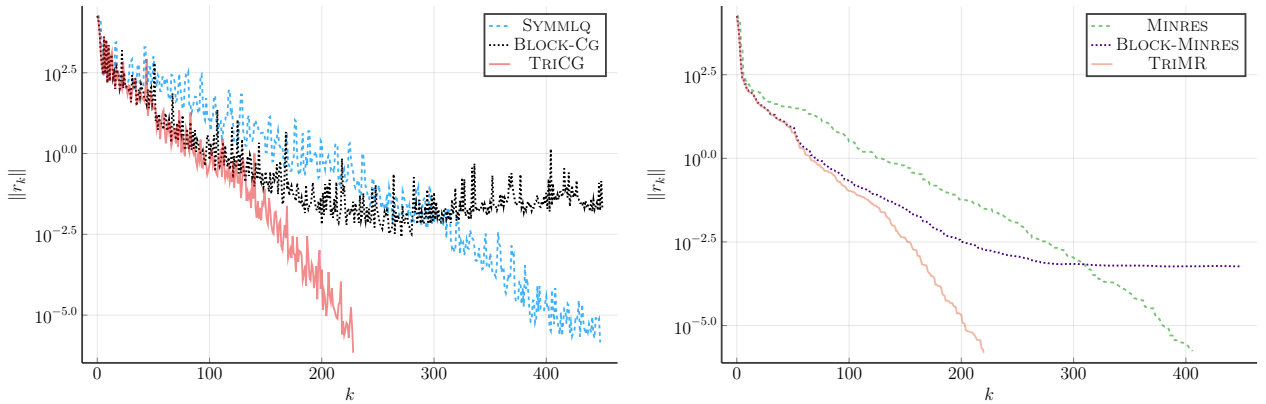


Figure 5.9 Residual history of SYMMLQ, BLOCK-CG, TriCG, MINRES, BLOCK-MINRES and TriMR on lp_d6cube, where A is 415×6184 with 37704 nonzeros.

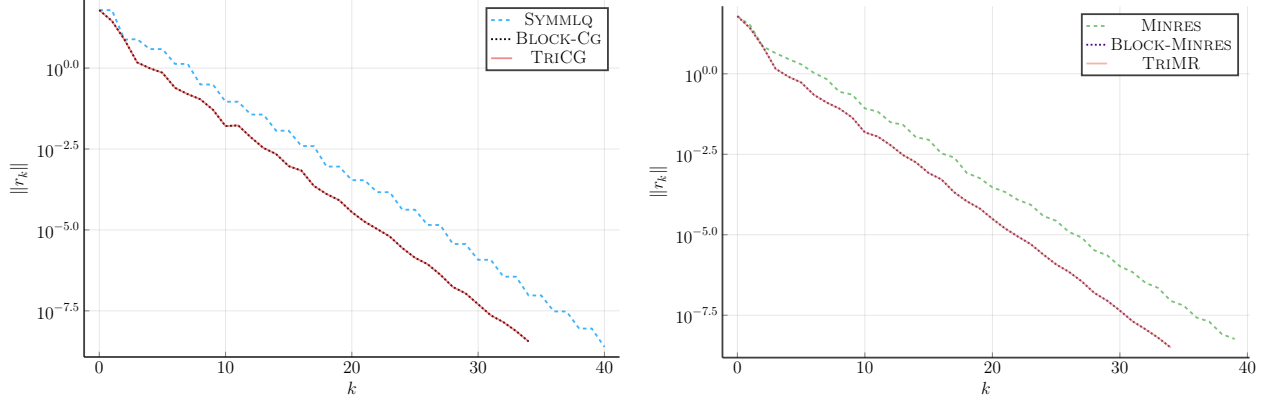


Figure 5.10 Residual history of SYMMLQ, BLOCK-CG, TRiCG, MINRES, BLOCK-MINRES and TRIMR on well1033, where A is 1033×320 with 4732 nonzeros.

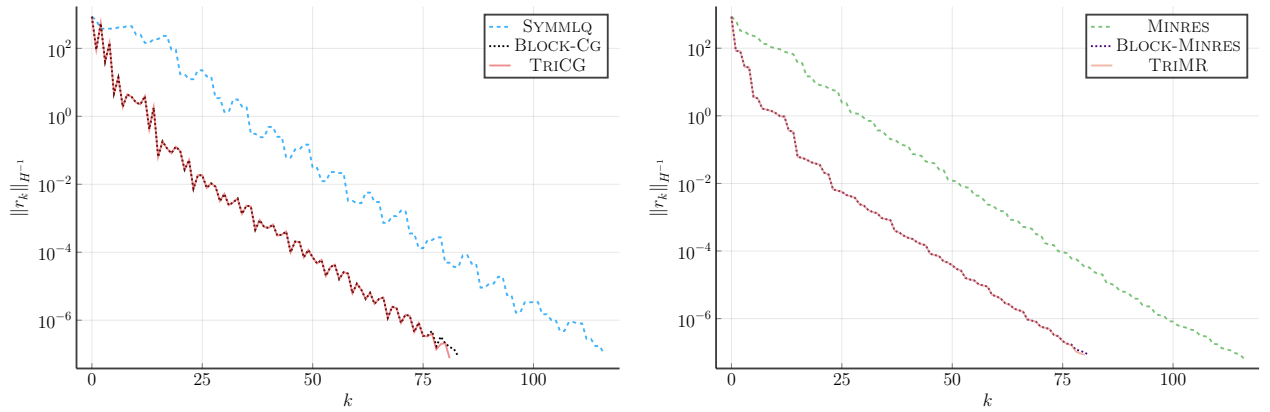


Figure 5.11 Residual history of SYMMLQ, BLOCK-CG, TRiCG, MINRES, BLOCK-MINRES and TRIMR iterates on colliding_flow problem with $Q_1 - P_0$ discretization. The discretized linear system has size 12546×12546 with 146241 nonzeros.

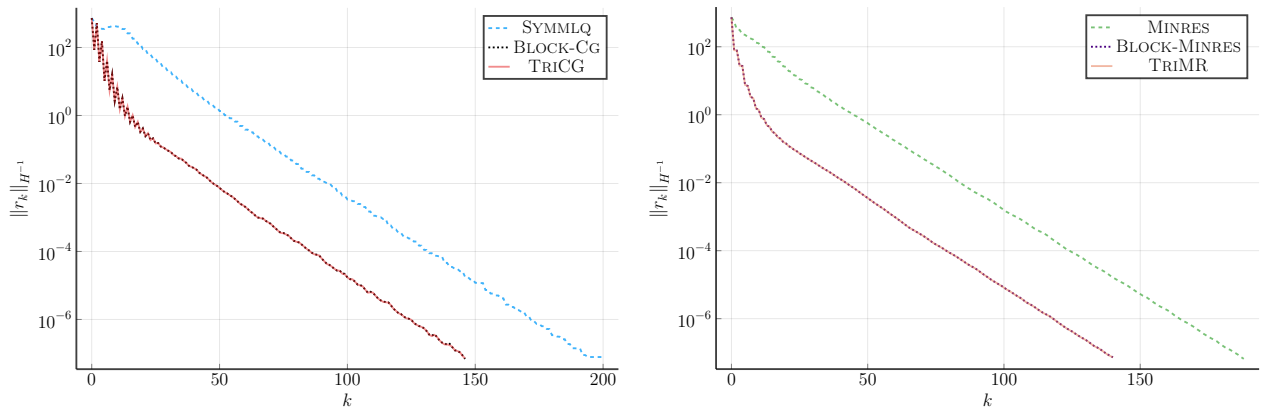


Figure 5.12 Residual history of SYMMLQ, BLOCK-CG, TRiCG, MINRES, BLOCK-MINRES and TRIMR iterates on colliding_flow problem with $Q_1 - Q_1$ discretization. The discretized linear system has size 12675×12675 with 239873 nonzeros.

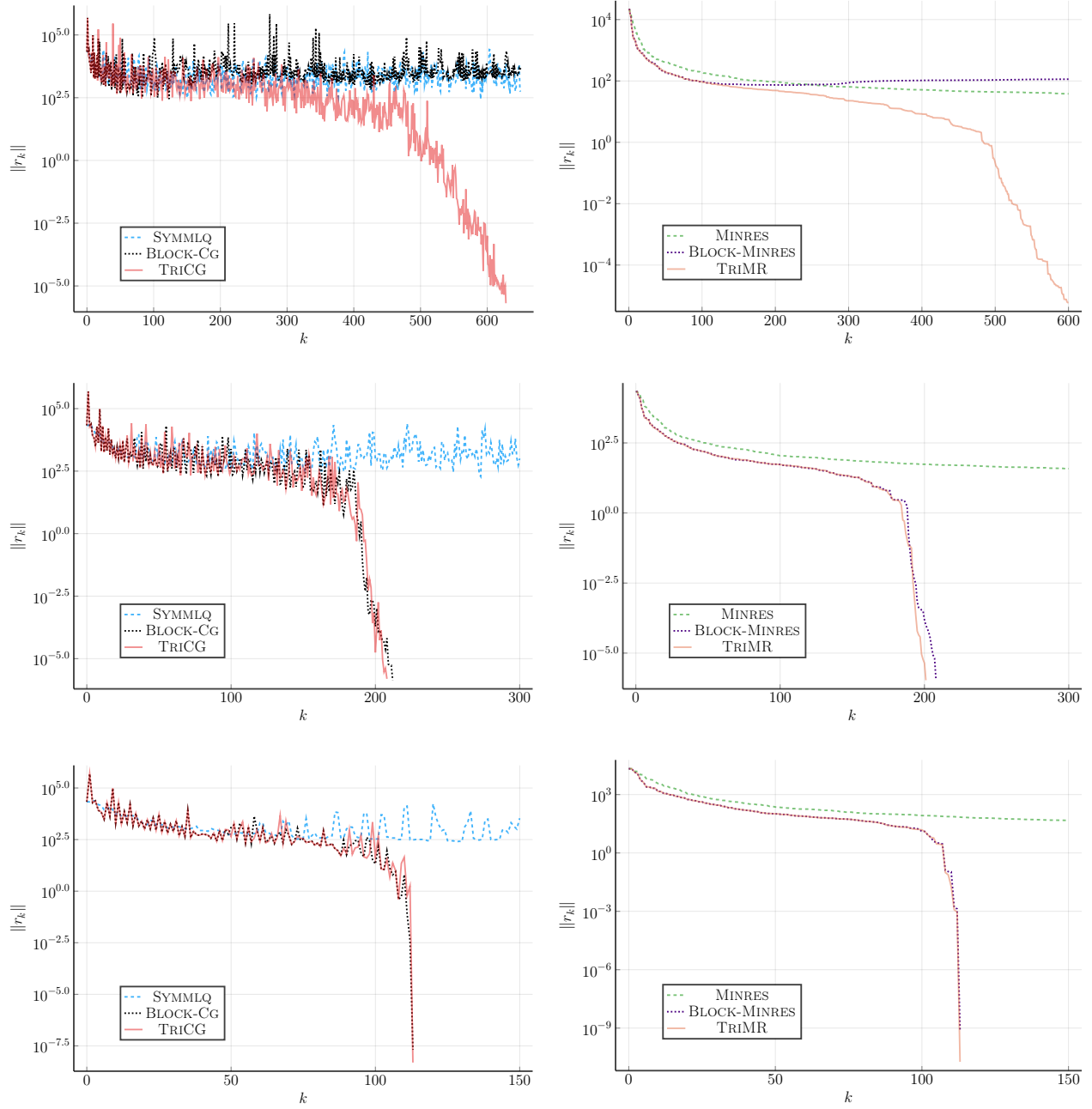


Figure 5.13 Residual history of SYMMLQ, BLOCK-CG, TRICG, MINRES, BLOCK-MINRES and TRIMR iterates on lpi_klein3 in double (top), quadruple (center) and octuple precision (bottom) where A is 994×1082 with 13101 nonzeros.

5.5 Discussion and extensions

Although we develop TRICG and TRIMR for SQD systems, both can be generalized to handle any linear system of the form

$$\begin{bmatrix} \pm M & A \\ A^T & \pm N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}, \quad (5.46)$$

where M and N are symmetric positive definite. Symmetric definite systems can always be written in the form (5.46). For instance, one could solve any definite system by first partitioning any symmetric permutation of it as (5.46) and applying TRICG or TRIMR to the resulting A . There are multiple ways to perform such partitioning, and it is not clear whether such strategy might lead to improved solution processes for SPD systems. With minor modifications, the blocks $\pm M$ and $\pm N$ in (5.46) can be replaced by λM and μN . Thus, TRIMR supports the saddle-point system (5.6) because the scalars μ and ν can be safely set to zero. In this case, N can be replaced by any SPD matrix to define an elliptic norm in Algorithm 6, which extends the possibilities for preconditioning the linear system. For instance, TRIMR could use the preconditioner $\text{blkdiag}(M, S)$ [66] for Stokes problems discretized by LBB-stable finite element pairs where S is the Schur complement $A^T M^{-1} A$ or an approximation to it. Our implementations of TRICG and TRIMR take into account all these extensions, are applicable in any floating-point system supported by Julia, and run on GPUs.

TRICG and TRIMR perform substantially better than SYMMLQ and MINRES in our experiments and often terminate in about half as many iterations on a residual-based stopping condition. Nevertheless more extensive testing is required to properly assess their performance in practice. Although theoretically equivalent, TRICG and TRIMR appear to preserve orthogonality of the Krylov basis better than their counterparts BLOCK-CG and BLOCK-MINRES.

Based upon Algorithm 6, it is possible to develop a third method in the spirit of SYMMLQ that we could name TRILQ. The TRILQ subproblem selects z_k in (5.16) as the solution of the minimum-norm subproblem

$$\underset{z_k \in \mathbb{R}^{2k}}{\text{minimize}} \quad \|z_k\| \quad \text{subject to} \quad S_{k-1,k} z_k = \beta_1 e_1 + \gamma_1 e_2,$$

where $S_{k-1,k}$ is the leading $(2k-2) \times (2k)$ submatrix of $S_{k+1,k}$ in (5.13). The subproblem can be solved via the LQ factorization of $S_{k-1,k}$. Much of TRILQ would be similar to block-SYMMLQ: iterates are updated along orthogonal directions, the H -norm of the iterates increases monotonically, and the H -norm of the error decreases monotonically. At each

iteration, TRILQ allows the user to transfer to the TRICG iterate. Because the TRICG iterate always exists for (5.1), TRILQ might not have any advantage in practice, other than completing the family of numerical methods based on Algorithm 6. However, the TRILQ iterate remains well defined for the saddle-point system (5.6), whereas TRICG may break down in that case.

The strong connection between TRICG and BLOCK-CG with blocks of size 2 suggests that TRICG might also be useful to approximate eigenvalues. We leave the investigation of such extensions to future work.

CHAPITRE 6 ARTICLE 3: GPMR: AN ITERATIVE METHOD FOR UNSYMMETRIC PARTITIONED LINEAR SYSTEMS

ALEXIS MONTISON AND DOMINIQUE ORBAN

Manuscript published on March 10, 2023, in the *SIAM Journal on Matrix Analysis and Applications* (SIMAX)

Abstract

We introduce an iterative method named GPMR for solving 2×2 block unsymmetric linear systems. GPMR is based on a new process that simultaneously reduces two rectangular matrices to upper Hessenberg form and 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 its storage and work per iteration are similar to those of GMRES. We compare the performance of GPMR with 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.

6.1 Introduction

Consider the partitioned linear system

$$\begin{bmatrix} M & A_\star \\ B_\star & N \end{bmatrix} \begin{bmatrix} x_\star \\ y_\star \end{bmatrix} = \begin{bmatrix} b_\star \\ c_\star \end{bmatrix}, \quad (6.1)$$

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 (6.1) occurs, among others, in the discretization of systems of partial-differential equations, including the Navier-Stokes equations by way of the finite element method [21]. A prime example is domain decomposition with no overlap, also known as iterative substructuring [17], which consists of splitting a domain into k non-overlapping subregions and leads to structured matrices with *arrowhead* form [26]. Let $\mathcal{I}$ be the set of 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

$$\begin{bmatrix} A_{\mathcal{I}\mathcal{I}} & A_{\mathcal{I}\Gamma} \\ A_{\Gamma\mathcal{I}} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_{\mathcal{I}} \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_{\mathcal{I}} \\ f_\Gamma \end{bmatrix} \iff \begin{bmatrix} A_{11} & & A_{1\Gamma} \\ & \ddots & \vdots \\ & & A_{kk} & A_{k\Gamma} \\ A_{\Gamma 1} & \dots & A_{\Gamma k} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_k \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_k \\ f_\Gamma \end{bmatrix}, \quad (6.2)$$

where $u = (u_{\mathcal{I}}, u_\Gamma)$ is the vector of nodal displacements and f is the vector of nodal forces. For a tour of applications leading to (6.1), we refer to [4]. We assume that there exist nonsingular P_ℓ and P_r with inexpensive inverses such that

$$K := P_\ell^{-1} \begin{bmatrix} M & A_\star \\ B_\star & N \end{bmatrix} P_r^{-1} = \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix}, \quad \lambda, \mu \in \mathbb{R}, \quad (6.3)$$

so that the equivalent preconditioned system

$$\begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}, \quad \begin{bmatrix} x_\star \\ y_\star \end{bmatrix} = P_r^{-1} \begin{bmatrix} x \\ y \end{bmatrix}, \quad \begin{bmatrix} b \\ c \end{bmatrix} = P_\ell^{-1} \begin{bmatrix} b_\star \\ c_\star \end{bmatrix} \quad (6.4)$$

can be solved instead of (6.1). Note that λ and/or μ may be zero. For example, the ideal preconditioners of [Murphy *et al.* \[66\]](#) and [Ipsen \[49\]](#) lead to (6.3). Although ideal preconditioners are typically not practical because they require the solution of systems with the Schur complement $S = N - B_\star M^{-1} A_\star$, viable preconditioners such that $P_\ell P_r = \text{blkdiag}(M, N)$ can be employed when M and N are both nonsingular.

Given an unstructured matrix C , a practical approach to recovering the matrix of (6.1) is to permute its rows and columns with orderings determined by graph partitioning tools such as METIS [52]. This reordering also provides a uniform partitioning to compute a parallel block-Jacobi preconditioner for (6.3).

When $\lambda \neq 0$, (6.4) can be reduced to the Schur complement system

$$(BA - \lambda \mu I)y = Bb - \lambda c, \quad x = \lambda^{-1}(b - Ay), \quad (6.5)$$

or, when $\mu \neq 0$, to

$$(AB - \lambda \mu I)x = Ac - \mu b, \quad y = \mu^{-1}(c - Bx). \quad (6.6)$$

The above systems are attractive because of their smaller size, but may have worse conditioning than (6.4), e.g., when $B = A^T$, $M = M^T \succ 0$ and $N = N^T \preceq 0$, though not always, e.g, when (6.1) is symmetric and positive definite. We focus on applying an iterative method

to (6.4) 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 (6.4), and (iii) an efficient software implementation to solve (6.4) in arbitrary floating-point arithmetic on CPU and GPU.

Related research

Numerous Krylov methods have been developed for solving general unsymmetric linear systems, including GMRES [80], QMR [30] or BiLQ [61]. Few are tailored specifically to the block structure of (6.1).

Specialized iterative methods have been developed for special cases of (6.1). Estrin and Greif [25] developed SPMR, a family of methods for (6.1) that exploit its block structure when $N = 0$ and b or c is zero. Buttari *et al.* [9] developed USYMLQR, an interlacing of the methods USYMLQ and USYMQR of Saunders *et al.* [85], applicable when $A = B^T$, $M = M^T \succ 0$ and $N = 0$. Greif and Wathen [41] 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 [70] 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 [62] 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.* [85], 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

$$K_0 := \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}, \quad \text{blkdiag}(\lambda I, \mu I) := \begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix}, \quad d := \begin{bmatrix} b \\ c \end{bmatrix}, \quad D := \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix}. \quad (6.7)$$

For a matrix C and a vector t , $\mathcal{K}_k(C, t)$ is the Krylov subspace $\text{Span}\{t, Ct, \dots, C^{k-1}t\}$. For a matrix T with p columns and as many rows as C has columns, $\mathcal{K}_k(C, T)$ is the block-Krylov subspace [44]

$$\mathcal{K}_k^\square(C, T) := \left\{ \sum_{i=0}^{k-1} C^i T \Omega_i \mid \Omega_i \in \mathbb{C}^{p \times p}, 0 \leq i \leq k-1 \right\}.$$

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 $l = \begin{bmatrix} l_1 & \dots & l_n \end{bmatrix}^T$, respectively.

6.2 A Hessenberg reduction process

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 2. *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 $F \in \mathbb{R}^{p \times p}$ with nonnegative subdiagonal such that*

$$V^T A U = H, \quad (6.8a)$$

$$U^T B V = F. \quad (6.8b)$$

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

$$\beta_{k+1} v_{k+1} = A u_k - \sum_{i=1}^k (v_i^T A u_k) v_i, \quad (6.9a)$$

$$\gamma_{k+1} u_{k+1} = B v_k - \sum_{i=1}^k (u_i^T B v_k) u_i, \quad (6.9b)$$

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 $A u_k \in \text{Span}\{v_1, \dots, v_k\}$ or $B v_k \in \text{Span}\{u_1, \dots, u_k\}$, we choose an arbitrary unit $v_{k+1} \perp \text{Span}\{v_1, \dots, v_k\}$ or $u_{k+1} \perp \text{Span}\{u_1, \dots, u_k\}$ 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:

$$v_j^T v_{k+1} = 0 \quad \text{and} \quad u_j^T u_{k+1} = 0 \quad (j = 1, \dots, k). \quad (6.10)$$

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 (6.9) 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 $j = 1, \dots, k$, (6.9) implies

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

so that $\mathcal{P}(k)$ also holds. For $j = 1, \dots, k-1$, we have from (6.9) and $\mathcal{P}(k)$ that

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

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

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

satisfy (6.8a)–(6.8b) and have the properties announced. $\square$

Algorithm 10 formalizes a Hessenberg reduction process derived from **Theorem 2**.

Algorithm 10 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 $k = 1, 2, \dots$ do | |
| 3: for $i = 1, \dots, k$ 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: $h_{k+1,k} v_{k+1} = A u_k - \sum_{i=1}^k h_{i,k} v_i$ | $h_{k+1,k} > 0$ 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 $V_k := [v_1 \ \dots \ v_k]$ and $U_k := [u_1 \ \dots \ u_k]$. After k iterations of **Algorithm 10**, the

situation may be summarized as

$$AU_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T = V_{k+1} H_{k+1,k}, \quad (6.11a)$$

$$BV_k = U_k F_k + f_{k+1,k} u_{k+1} e_k^T = U_{k+1} F_{k+1,k}, \quad (6.11b)$$

$$V_k^T V_k = U_k^T U_k = I_k, \quad (6.11c)$$

where

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

and

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

If $B = A^T$, [Algorithm 10](#) reduces to the orthogonal tridiagonalization process of [Saunders et al. \[85\]](#), H_k and F_k are tridiagonal and $H_k = F_k^T$. [Algorithm 10](#) uses the Gram-Schmidt method [\[6\]](#) for computing ℓ_2 -orthonormal bases V_k and U_k for simplicity. In a practical implementation, including ours, detailed in [section 6.4](#), the modified Gram-Schmidt algorithm would be used instead. Note that (6.11a)–(6.11b) hold to within machine precision despite a potential numerical loss of orthogonality of (6.11c). We refer the interested reader to [\[11\]](#) for a stability analysis of (6.11c).

In exact arithmetic, (6.11) 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 \[2\]](#), which can be used to approximate eigenvalues.

In the same vein that the Arnoldi process and the Gram-Schmidt orthogonalization are related, [Algorithm 10](#) and the block Gram-Schmidt orthogonalization [\[88\]](#) are analogous. In the case of two blocks A and B , [Algorithm 10](#) reduces the blocks to Hessenberg form, while the block Gram-Schmidt orthogonalization triangularizes the blocks by a succession of triangular operations. They both have the advantage that they can be interrupted early, leading to partial reductions to block-Hessenberg form or block QR factorizations.

6.2.1 Relation with the block-Arnoldi process

For $k \geq 1$,

$$v_{2k} \in \text{Span}\{b, \dots, (AB)^{k-1}b, Ac, \dots, (AB)^{k-1}Ac\}, \quad (6.12a)$$

$$v_{2k+1} \in \text{Span}\{b, \dots, (AB)^k b, Ac, \dots, (AB)^{k-1}Ac\}, \quad (6.12b)$$

$$u_{2k} \in \text{Span}\{c, \dots, (BA)^{k-1}c, Bb, \dots, (BA)^{k-1}Bb\}, \quad (6.12c)$$

$$u_{2k+1} \in \text{Span}\{c, \dots, (BA)^k c, Bb, \dots, (BA)^{k-1}Bb\}. \quad (6.12d)$$

The subspaces generated by [Algorithm 10](#) 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 $\begin{bmatrix} c & Bb \end{bmatrix}$. V_{2k} and U_{2k} are bases of $\mathcal{K}_k^\square(AB, \begin{bmatrix} b & Ac \end{bmatrix})$ and $\mathcal{K}_k^\square(BA, \begin{bmatrix} c & Bb \end{bmatrix})$, respectively. Merging (6.11a)–(6.11b) gives

$$\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} = \begin{bmatrix} V_{k+1} & 0 \\ 0 & U_{k+1} \end{bmatrix} \begin{bmatrix} 0 & H_{k+1,k} \\ F_{k+1,k} & 0 \end{bmatrix}, \quad (6.13)$$

which is reminiscent of the relation one would obtain from applying an orthogonalization process to K_0 . Note the similarity between (6.13) and a Krylov process in which basis vectors have been permuted. Let

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

denote the permutation introduced by [Paige \[71\]](#) that restores the order in which [Algorithm 10](#) generates basis vectors, i.e.,

$$W_k := \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} P_k = \begin{bmatrix} w_1 & \cdots & w_k \end{bmatrix}, \quad w_k = \begin{bmatrix} v_k & 0 \\ 0 & u_k \end{bmatrix}, \quad W_k^T W_k = I_{2k}. \quad (6.14)$$

The projection of K_0 into the basis W_k is also shuffled to block-Hessenberg form with blocks of size 2. Indeed, if we multiply (6.13) on the right with P_k and use (6.14), we obtain

$$K_0 W_k = \begin{bmatrix} V_{k+1} & 0 \\ 0 & U_{k+1} \end{bmatrix} P_{k+1} P_{k+1}^T \begin{bmatrix} 0 & H_{k+1,k} \\ F_{k+1,k} & 0 \end{bmatrix} P_k = W_{k+1} G_{k+1,k}, \quad (6.15)$$

where

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

The two relations at line 1 of [Algorithm 10](#) can be rearranged as

$$\begin{bmatrix} v_1 & 0 \\ 0 & u_1 \end{bmatrix} \begin{bmatrix} \beta & 0 \\ 0 & \gamma \end{bmatrix} = \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} \iff w_1 \Gamma = D. \quad (6.16)$$

Identities (6.15) and (6.16) characterize the block-Arnoldi process applied to K_0 with initial block D . W_k is therefore a basis of $\mathcal{K}_k^\square(K_0, D)$. We summarize the process as [Algorithm 11](#), 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 11 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**
-

6.2.2 Regularization of the block-Arnoldi process

Because $K = K_0 + \text{blkdiag}(\lambda I, \mu I)$, (6.13) yields

$$\begin{aligned} \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} &= \left(\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} + \begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix} \right) \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} \\ &= \begin{bmatrix} V_k & 0 \\ 0 & U_k \end{bmatrix} \begin{bmatrix} \lambda I & H_k \\ F_k & \mu I \end{bmatrix} + \begin{bmatrix} v_{k+1} & 0 \\ 0 & u_{k+1} \end{bmatrix} \begin{bmatrix} 0 & h_{k+1,k} e_k^T \\ f_{k+1,k} e_k^T & 0 \end{bmatrix}. \end{aligned} \quad (6.17)$$

The same reasoning applied to (6.15) yields the following result, which parallels [Montoisson and Orban \[62, Theorem 2.1\]](#).

Theorem 3. *Given the matrix K defined in (6.3) and the block right-hand side D defined in (6.7), the block Krylov basis $W_k = [w_1 \ \cdots \ w_k]$ generated by Algorithm 11 with regularization has the form (6.14), where the vectors u_k and v_k are the same as those generated by Algorithm 10 with initial vectors b and c . In addition,*

$$KW_k = W_{k+1}S_{k+1,k}, \quad S_{k+1,k} := \begin{bmatrix} \Theta_{1,1} & \Psi_{1,2} & \cdots & \Psi_{1,k} \\ \Psi_{2,1} & \Theta_{2,2} & \ddots & \vdots \\ & \ddots & \ddots & \Psi_{k-1,k} \\ & & \ddots & \Theta_{k,k} \\ & & & \Psi_{k+1,k} \end{bmatrix}, \quad (6.18)$$

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 10 applied to A and B with initial vectors b and c .

Proof. Algorithm 11 applied to K_0 generates sparse pairs w_k as in (6.14) because of the equivalence with Algorithm 10. The term $\text{blkdiag}(\lambda I, \mu I)$ can be seen as a regularization term:

$$\begin{bmatrix} \lambda I & 0 \\ 0 & \mu I \end{bmatrix} w_k = w_k \Lambda \quad \text{with} \quad \Lambda := \begin{bmatrix} \lambda & 0 \\ 0 & \mu \end{bmatrix}. \quad (6.19)$$

Identities (6.15) and (6.19) allow us to write

$$KW_k = W_{k+1} \begin{bmatrix} \Psi_{1,1} + \Lambda & \Psi_{1,2} & \cdots & \Psi_{1,k} \\ \Psi_{2,1} & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \Psi_{k-1,k} \\ & & \Psi_{k,k-1} & \Psi_{k,k} + \Lambda \\ & & & \Psi_{k+1,k} \end{bmatrix}, \quad (6.20)$$

which amounts to (6.18) because $\Theta_{k,k} = \Psi_{k,k} + \Lambda$. □

Note that (6.18) is identical to (6.17) with the order of the w_k permuted according to P_k .

Because of Theorem 3, the block Krylov basis W_k generated by Algorithm 11 must have the sparsity structure (6.14), so that only u_k and v_k need be generated, and they may be generated

directly from [Algorithm 10](#). The key point is that generating orthonormal bases of $\mathcal{K}_k(K, d)$ and $\mathcal{K}_k^\square(K, D)$ by the Arnoldi process and [Algorithm 10](#), respectively, requires exactly the same amount of storage, and $\mathcal{K}_k(K, d) \subset \mathcal{K}_k^\square(K, D)$. 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 $\mathcal{K}_k^\square(K, D)$. Such a method is the subject of the next section.

6.3 Derivation of GPMR

We develop the method GPMR based upon [Algorithm 10](#) with regularization to solve (6.4), in which the iterate k has the form

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = W_k z_k, \quad (6.21)$$

where $z_k \in \mathbb{R}^{2k}$. From (6.16) and (6.18), the residual can be written

$$\begin{aligned} r_k &= \begin{bmatrix} b \\ c \end{bmatrix} - \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} \\ &= w_1 \begin{bmatrix} \beta \\ \gamma \end{bmatrix} - W_{k+1} S_{k+1,k} z_k \\ &= W_{k+1} (\beta e_1 + \gamma e_2 - S_{k+1,k} z_k). \end{aligned} \quad (6.22)$$

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

$$\underset{z_k \in \mathbb{R}^{2k}}{\text{minimize}} \|S_{k+1,k} z_k - (\beta e_1 + \gamma e_2)\|. \quad (6.23)$$

6.3.1 Relation between GPMR and BLOCK-GMRES

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

$$\underset{z_k \in \mathbb{R}^{2k}}{\text{minimize}} \left\| \begin{bmatrix} b & 0 \\ 0 & c \end{bmatrix} - \begin{bmatrix} \lambda I & A \\ B & \mu I \end{bmatrix} \begin{bmatrix} x_k^b & x_k^c \\ y_k^b & y_k^c \end{bmatrix} \right\|_F, \quad (6.24)$$

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

$$\underset{z_k^b, z_k^c \in \mathbb{R}^{2k}}{\text{minimize}} \left\| S_{k+1,k} \begin{bmatrix} z_k^b & z_k^c \end{bmatrix} - \begin{bmatrix} \beta e_1 & \gamma e_2 \end{bmatrix} \right\|_F, \quad (6.25)$$

so that z_k^b and z_k^c solve the subproblems associated with right-hand sides βe_1 and γe_2 . In exact arithmetic, the solutions of (6.23) and (6.25) 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 (6.23).

6.3.2 A QR factorization

The solution of (6.23) can be determined via the QR factorization

$$S_{k+1,k} = Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \quad (6.26)$$

which can be updated at each iteration, where $Q_k \in \mathbb{R}^{(2k+2) \times (2k+2)}$ is a product of reflections, and $R_k \in \mathbb{R}^{(2k) \times (2k)}$ is upper triangular. At each iteration, four new reflections are necessary to update (6.26). 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

$$\begin{matrix}
& 1 & \dots & 2i-2 & 2i-1 & 2i & 2i+1 & 2i+2 & 2i+3 & \dots & 2k+2 \\
1 & 1 & & & & & & & & & \\
\vdots & & \ddots & & & & & & & & \\
2i-2 & & & 1 & & & & & & & \\
2i-1 & & & & \times & \times & \times & \times & & & \\
2i & & & & \times & \times & \times & \times & & & \\
2i+1 & & & & \times & \times & \times & \times & & & \\
2i+2 & & & & \times & \times & \times & \times & & & \\
2i+3 & & & & & & & & 1 & & \\
\vdots & & & & & & & & & \ddots & \\
2k+2 & & & & & & & & & & 1
\end{matrix}$$

where the diagonal block extracted from rows and columns $2i - 1, \dots, 2i + 2$ is the following product of four 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 12](#).

Algorithm 12 Procedure ref

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

- | | |
|--|--------------------------|
| 1: $t = c_{1,i}a_1^{\text{in}} + s_{1,i}a_4^{\text{in}}, a_4^{\text{out}} = s_{1,i}a_1^{\text{in}} - c_{1,i}a_4^{\text{in}}, a_1^{\text{out}} = t$ | <i>first reflection</i> |
| 2: $t = c_{2,i}a_1^{\text{out}} + s_{2,i}a_2^{\text{in}}, a_2^{\text{out}} = s_{2,i}a_1^{\text{out}} - c_{2,i}a_2^{\text{in}}, a_1^{\text{out}} = t$ | <i>second reflection</i> |
| 3: $t = c_{3,i}a_2^{\text{out}} + s_{3,i}a_4^{\text{out}}, a_4^{\text{out}} = s_{3,i}a_2^{\text{out}} - c_{3,i}a_4^{\text{out}}, a_2^{\text{out}} = t$ | <i>third reflection</i> |
| 4: $t = c_{4,i}a_2^{\text{out}} + s_{4,i}a_3^{\text{in}}, a_3^{\text{out}} = s_{4,i}a_2^{\text{out}} - c_{4,i}a_3^{\text{in}}, a_2^{\text{out}} = t$ | <i>fourth reflection</i> |
-

At iteration k , [Algorithm 10](#) generates two new columns, and to update the QR decomposition we need first to apply all previous reflections:

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

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 sines and cosines and finalize the QR factorization of $S_{k+1,k}$ is described as [Algorithm 13](#). Note that the first parameter of [Algorithm 12](#) and [Algorithm 13](#) is used to define which sines and cosines are read from or written to memory.

Algorithm 13 Procedure qr

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

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

6.3.3 GPMR iterate and residual norm computation

We have from (6.22) and (6.26):

$$\|r_k\| = \left\| Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix} z_k - (\beta e_1 + \gamma e_2) \right\| = \left\| \begin{bmatrix} R_k \\ 0 \end{bmatrix} z_k - \bar{t}_k \right\|, \quad (6.27)$$

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 (6.23) 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 (6.27) yield

$$\|r_k\| = \sqrt{\bar{\tau}_{2k+1}^2 + \bar{\tau}_{2k+2}^2}. \quad (6.28)$$

As in GMRES, we only compute z_k when $\|r_k\|$ is smaller than a user-provided tolerance.

From (6.21), the solution may be computed efficiently as

$$x_k = \sum_{i=1}^k \zeta_{2i-1} v_i, \quad (6.29a)$$

$$y_k = \sum_{i=1}^k \zeta_{2i} u_i. \quad (6.29b)$$

We summarize the complete procedure as [Algorithm 14](#).

Algorithm 14 GPMR

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

```

1:  $\beta v_1 = b, \gamma u_1 = c$   $(\beta, \gamma) > 0$  so that  $\|v_1\| = \|u_1\| = 1$ 
2:  $\bar{\tau}_1 = \beta, \bar{\tau}_2 = \gamma$  Initialize  $\bar{t}_0$ 
3:  $\|r_0\| = (\bar{\tau}_1^2 + \bar{\tau}_2^2)^{\frac{1}{2}}$  compute  $\|r_0\|$ 
4:  $k = 0$ 
5: while  $\|r_k\| > \epsilon$  and  $k < k_{\max}$  do
6:    $k \leftarrow k + 1$ 
7:    $q = Au_k$  Orthogonal Hessenberg reduction
8:    $p = Bv_k$ 
9:   for  $i = 1, \dots, k$  do
10:     $h_{i,k} = v_i^T q$ 
11:     $f_{i,k} = u_i^T p$ 
12:     $q = q - h_{i,k} v_i$ 
13:     $p = p - f_{i,k} u_i$ 
14:   end for
15:    $h_{k+1,k} v_{k+1} = q$   $h_{k+1,k} > 0$  so that  $\|v_{k+1}\| = 1$ 
16:    $f_{k+1,k} u_{k+1} = p$   $f_{k+1,k} > 0$  so that  $\|u_{k+1}\| = 1$ 
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  $i = 1, \dots, k-1$  do Apply  $Q_{2k-5, 2k-2}, \dots, Q_{1,4}$ 
20:     if  $i \neq k-1$  then  $(\rho, \delta) = (0, 0)$  else  $(\rho, \delta) = (\lambda, \mu)$ 
21:        $r_{2i-1, 2k-1}, r_{2i, 2k-1}, \bar{r}_{2i+1, 2k-1}, \bar{r}_{2i+2, 2k-1} = \text{ref}(i, \bar{r}_{2i-1, 2k-1}, \bar{r}_{2i, 2k-1}, \rho, f_{i+1, k})$ 
22:        $r_{2i-1, 2k}, r_{2i, 2k}, \bar{r}_{2i+1, 2k}, \bar{r}_{2i+2, 2k} = \text{ref}(i, \bar{r}_{2i-1, 2k}, \bar{r}_{2i, 2k}, h_{i+1, k}, \delta)$ 
23:     end for
24:      $r_{2k-1, 2k-1}, r_{2k-1, 2k}, r_{2k, 2k} =$  Compute and apply  $Q_{2k-1, 2k+2}$ 
        $\text{qr}(k, \bar{r}_{2k-1, 2k-1}, \bar{r}_{2k-1, 2k}, \bar{r}_{2k, 2k-1}, \bar{r}_{2k, 2k}, h_{k+1, k}, f_{k+1, k})$ 
25:      $\tau_{2k-1}, \tau_{2k}, \bar{\tau}_{2k+1}, \bar{\tau}_{2k+2} = \text{ref}(k, \bar{\tau}_{2k-1}, \bar{\tau}_{2k}, 0, 0)$  update  $\bar{t}_k$ 
26:      $\|r_k\| = (\bar{\tau}_{2k+1}^2 + \bar{\tau}_{2k+2}^2)^{\frac{1}{2}}$  compute  $\|r_k\|$ 
27:   end while
28:    $\zeta_{2k} = \tau_{2k} / r_{2k, 2k}$  compute  $z_k$ 
29:   for  $i = 2k-1, \dots, 1$  do
30:      $\zeta_i = (\tau_i - \sum_{j=i+1}^{2k} r_{i,j} \zeta_j) / r_{i,i}$ 
31:   end for
32:    $x_k = \sum_{i=1}^k \zeta_{2i-1} v_i$  compute  $x_k$ 
33:    $y_k = \sum_{i=1}^k \zeta_{2i} u_i$  compute  $y_k$ 

```

6.3.4 Memory and computational requirements

Table 6.1 summarizes the storage costs of k iterations of GPMR, GMRES and BLOCK-GMRES.

Table 6.1 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 6.1 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.

The computational expense of GPMR, GMRES and BLOCK-GMRES is dominated by the operator-vector products ($y \leftarrow Kx$) and vector operations (dot products, vector norms, $y \leftarrow \alpha x + \beta y$). GPMR and GMRES require one operator-vector product with A and B per iteration, whereas a standard implementation of BLOCK-GMRES requires two (one per right-hand side). GMRES and BLOCK-GMRES also need products with λI and μI but their costs are insignificant compared to those with A and B . If we regroup vector operations in GPMR, exactly the same number of $(n + m)$ -vector operations are needed by GPMR and GMRES to build Krylov bases and compute the final solution. BLOCK-GMRES performs twice as many operations of the form $y \leftarrow \alpha x + \beta y$, and four times as many dot products as GPMR and GMRES. Norm operations used to normalize Arnoldi vectors in GMRES and the orthogonal Hessenberg reduction process in GPMR are replaced by a thin QR factorization to orthonormalize basis matrices in BLOCK-GMRES. The remaining operations concern the subproblem solves: QR factorization of (block)-Hessenberg matrices and backward substitutions. GPMR and BLOCK-GMRES perform and apply four Givens reflections per iteration, while GMRES performs only one. After k iterations, GMRES requires one triangular solve with a matrix of size k , while GPMR and BLOCK-GMRES require one and two triangular solves with a matrix of size $2k$, respectively. These operations related to subproblems are negligible as long as $k \ll n + m$.

6.4 Implementation and numerical experiments

We implemented Algorithm 14 in Julia [5], version 1.7, as part of our Krylov.jl collection of Krylov methods [60]. Our implementation of GPMR is applicable in any floating-point

system supported by Julia, including complex numbers, and runs on CPU and GPU. The GPU support can be particularly relevant for (6.2) because, as a Krylov method, GPMR only requires linear operators that model $A_{\mathcal{IT}}u$, $B_{\mathcal{IT}}v$, $u \mapsto M_{\mathcal{IT}} \setminus u$ and $v \mapsto N_{\mathcal{IT}} \setminus v$. For instance, $v \mapsto N_{\mathcal{IT}} \setminus v$ can be forward and backward substitutions with the factors of an LU decomposition of $N_{\mathcal{IT}}$. The use of abstract linear operators allows us to store $A_{\mathcal{IT}}$ and $B_{\mathcal{IT}}$ as well as decompositions of the diagonal blocks of (6.2) on distinct compute nodes and leverage parallel architectures, such as GPUs. When the matrices are unstructured, Duff and Scott [18] propose a robust arrowhead reordering such that each diagonal block is nonsingular and recovers a system of arrowhead form (6.2).

We evaluate the performance of GPMR on systems generated from unsymmetric matrices in the SuiteSparse Matrix Collection [15]. 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 (6.1) is identical to that of (6.4). The right-hand side $(b_\star, c_\star)$ is generated so the exact solution of (6.1) 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 $\|r_k\| \leq \varepsilon_a + \|(b, c)\| \varepsilon_r$ with absolute tolerance $\varepsilon_a = 10^{-12}$ and relative tolerance $\varepsilon_r = 10^{-10}$. Residuals are calculated explicitly at each iteration in order to evaluate $\|r_k\|$, (6.27) is not used to get a fair comparison. Table 6.2 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 6.1 reports residual histories of GPMR and GMRES on problems *scircuit*, *sme3Dc*, *PR02R* and *sherman5*. Note that residual histories of BLOCK-GMRES applied to K with block right-hand side D where the two approximate solutions are summed are superposed to those of GPMR. The curves of BLOCK-GMRES are therefore not displayed to improve readability. Our implementation of GPMR, GMRES and BLOCK-GMRES use the modified Gram-Schmidt procedure for computing ℓ_2 -orthonormal Krylov bases.

We also compare GPMR and GMRES on two generalized saddle point systems built with different shapes of A and B . The first problem is built with matrices *well1033* as A and *illc1033* as B . The second problem is built with the transpose of matrices *well1850* as A and *illc1850* as B . Both systems use $M = N = I$, $\lambda = 1$ and $\mu = 0$. Figure 6.2 reports residual histories of GPMR and GMRES on the two generalized saddle point systems. We observe that GPMR performs better than GMRES regardless the shape of A and B .

When K in (6.3) is symmetric, Algorithm 10 coincides with the orthogonal tridiagonalization process of Saunders *et al.* [85] 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, $M = N = I$,

Table 6.2 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%
Freescall1	3428755	18920347	456	344	25%

$\lambda = 1$ 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 6.3 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.

The GPMR and GMRES storage requirements increase with the iterations. For very large problems, storage may become prohibitive after a few iterations and restarted variants $\text{GPMR}(\ell)$ and $\text{GMRES}(\ell)$ may be preferable. Table 6.3 summarizes the number of iterations of $\text{GPMR}(\ell)$ and $\text{GMRES}(\ell)$ for $\ell = 20$ and 50 on the systems of Table 6.2. Figure 6.4 reports residual histories of $\text{GPMR}(\ell)$ and $\text{GMRES}(\ell)$ on problems *scircuit* and *thermomech_dK*. After ℓ iterations, the residuals of $\text{GPMR}(\ell)$ and $\text{GMRES}(\ell)$ are different, which leads to restarts from distinct right-hand sides, and explains the difference between the two methods. $\text{GPMR}(\ell)$ is therefore not ensured to require fewer iterations than $\text{GMRES}(\ell)$, although we observe that it does most of the time.

We close with experiments comparing GPMR to GMRES applied to the Schur complements (6.5) and (6.6), denoted GMRES-SCHUR(BA) and GMRES-SCHUR(AB), respectively. We selected systems generated during the iterations of the interior-point method of Friedlander and Orban [32] for convex quadratic optimization. The systems use the symmetric block 3×3 formu-

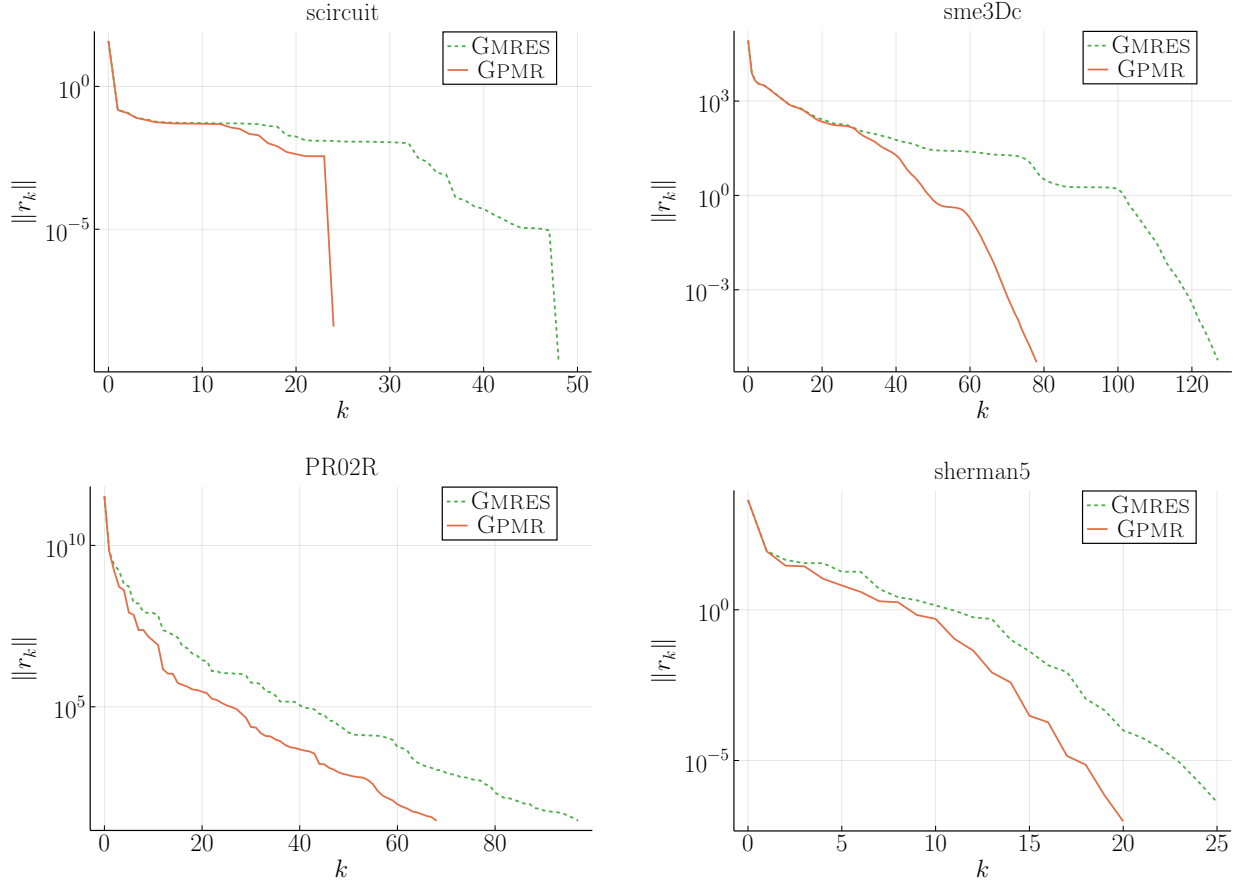


Figure 6.1 Residual history of GPMR and GMRES.

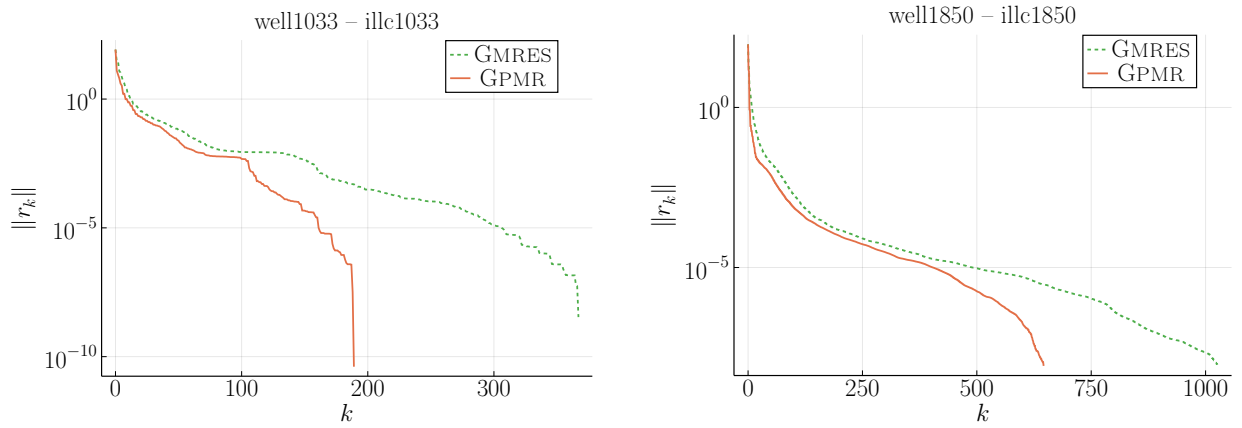


Figure 6.2 Residual history of GPMR and GMRES.

lation proposed in [40], and are part of the collection described in [69]. Even though K becomes increasingly ill conditioned as the interior-point iterations progress, its condition

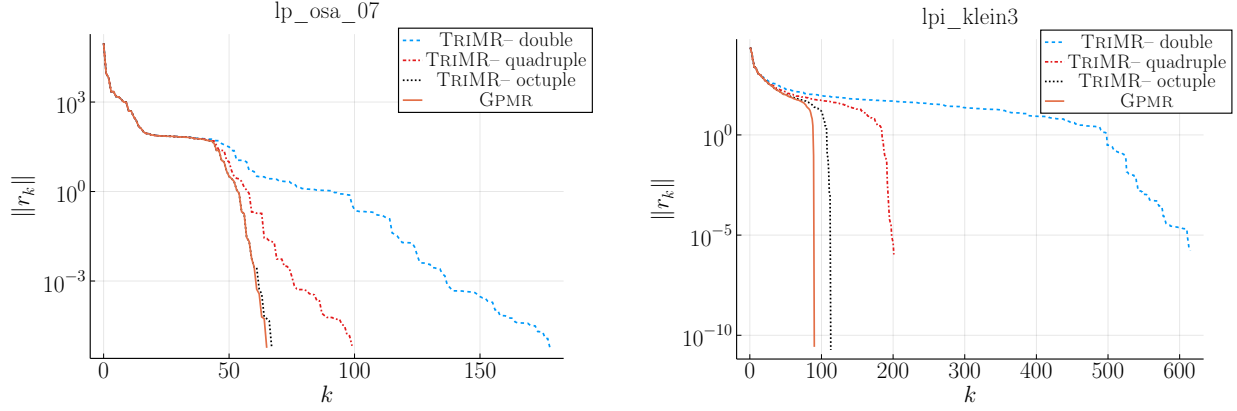


Figure 6.3 Residual history of GPMR and TRIMR.

Table 6.3 Number of iterations of $\text{GPMR}(\ell)$ and $\text{GMRES}(\ell)$ on systems from the SuiteSparse Matrix Collection.

name	GMRES(20)	GPMR(20)	GMRES(50)	GPMR(50)
sherman5	35	20	25	20
powersim	5540	2758	1040	349
Ill_Stokes	63	59	61	56
sme3Dc	1257	2237	706	229
rma10	63	51	48	41
ecl32	372	137	78	42
venkat50	55	50	48	35
poisson3Db	69	61	58	50
ifiss_mat	62	42	42	33
hcircuit	57	47	47	37
PR02R	353	238	180	99
scircuit	194	208	48	24
transient	18967	5618	2233	1471
ohne2	72	59	50	39
thermomech_dK	2269	1783	918	205
marine1	158	152	132	72
Freescale1	2844	2598	1360	1330

number remains theoretically bounded, while that of the Schur complements is provably unbounded [40]. Figure 6.5 reports the norm of the residual of (6.4). The two top plots correspond to well-conditioned K generated at the first interior-point iteration. We observe that GMRES-SCHUR(BA) and GMRES-SCHUR(AB) terminate earlier than both GPMR and GMRES. However, on the two bottom plots, which correspond to systems with worse conditioning generated at iteration 5 of the interior-point method, the residual of both GMRES-SCHUR(BA) and GMRES-SCHUR(AB) plateaus while that of GPMR and GMRES continues

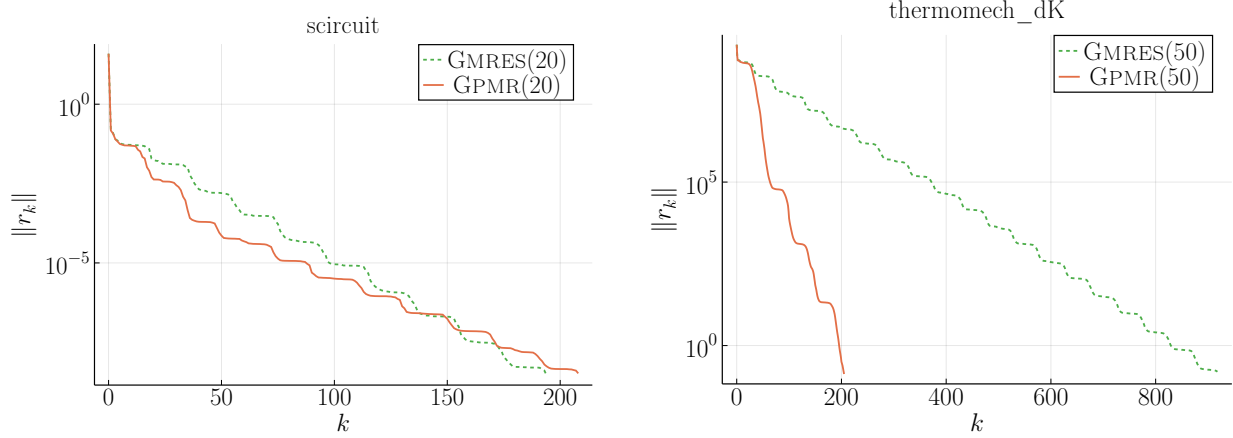


Figure 6.4 Residual history of $\text{GPMR}(\ell)$ and $\text{GMRES}(\ell)$.

to progress.

6.5 Discussion and extensions

In view of [Theorem 3](#), it is possible to specialize BLOCK-GMRES to the particular structure of the saddle-point system and devise a customized implementation. However, GPMR is more than simply a specialization of BLOCK-GMRES in the following respects:

- block Krylov methods are typically used to solve systems with multiple right-hand sides, whereas GPMR solves a system with a single right-hand side by adding together the solution estimates of BLOCK-GMRES at each iteration;
- GPMR is derived from the novel orthogonalization process [Algorithm 10](#) that generalizes that of [Saunders *et al.* \[85\]](#) (SSY), and, to the best of our knowledge, does not appear in the literature;
- on symmetric systems, GPMR reduces to the recently discovered TRIMR method [\[62\]](#), which is also based on the SSY process and shares similar relationships with BLOCK-MINRES.

Thus, one might have derived GPMR in a more contrived way by applying BLOCK-GMRES with blocks of size 2 to [\(6.4\)](#) with the block right-hand side D given in [\(6.7\)](#), exploiting sparsity to avoid performing unnecessary operations, and summing the solution approximations at each iteration. A benefit of this view is that known properties of the block Arnoldi and block Gram-Schmidt processes such as those of [\[11\]](#) apply to [Algorithm 10](#). However, the derivation of GPMR in the previous sections establishes a connection between the well-studied block Lanczos and block Arnoldi processes, and the far less studied SSY process.

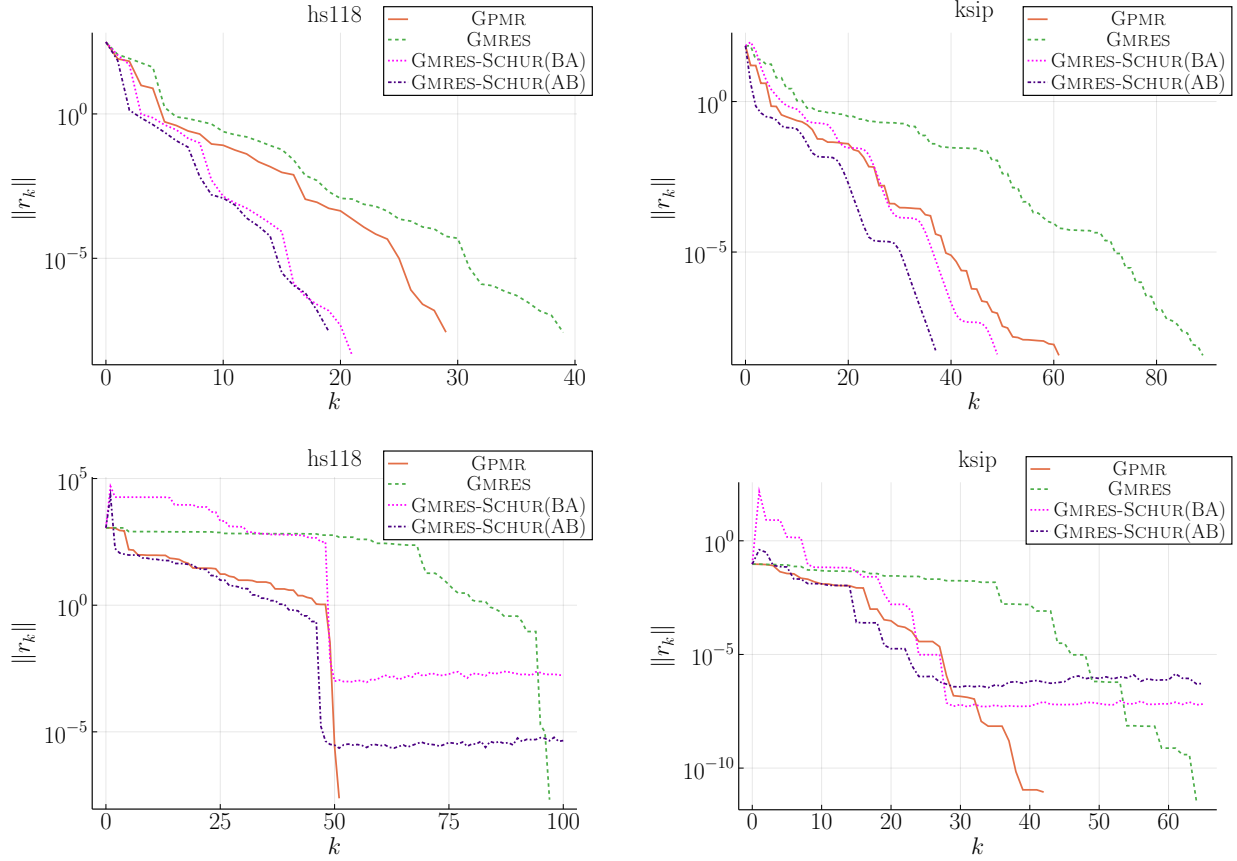


Figure 6.5 Residual history of GPMR, GMRES, GMRES-SCHUR(BA) and GMRES-SCHUR(AB) on systems generated from an interior-point method. Top: well-conditioned systems generated at the first interior-point iteration. Bottom: ill-conditioned systems generated at interior-point iteration 5.

Based upon [Algorithm 10](#), it is possible to develop another method, GPCG, in the spirit of FOM [\[81\]](#). The k -th GPCG iterate is defined by the Galerkin condition $W_k^T r_k = 0$. Its subproblem selects z_k in [\(6.21\)](#) as the solution of the square system

$$S_k z_k = \beta e_1 + \gamma e_2, \quad (6.30)$$

where S_k denotes the leading $(2k) \times (2k)$ submatrix of $S_{k+1,k}$ in [\(6.18\)](#). 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 [\[95\]](#), 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 (6.30) has a unique solution. The same observation holds for FOM, which should be restricted to certain classes of linear systems to avoid breakdowns.

Figure 6.6 puts processes, and the minimum-residual and Galerkin-based methods deriving from them in relation with one another and should help position Algorithm 10, denoted by the abbreviation OHR, and Algorithm 14 in a broad picture. Relations between processes translate to relations between methods, although sometimes with subtle differences such as those just outlined between GPMR and BLOCK-GMRES.

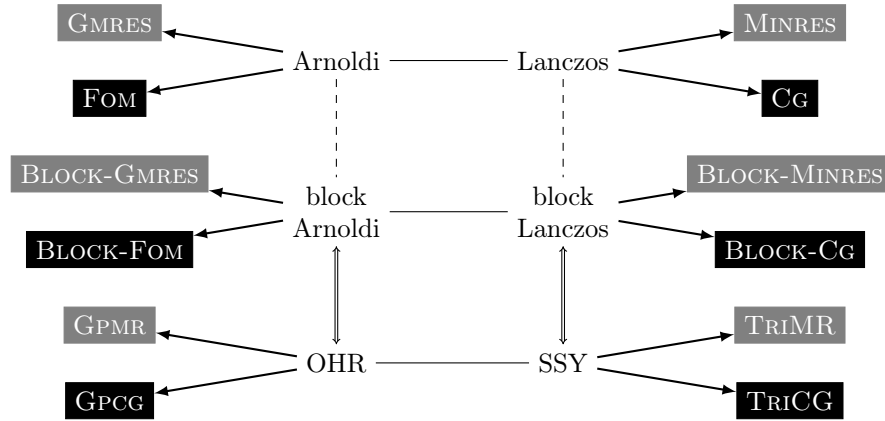


Figure 6.6 Relations between the relevant standard and block processes, and methods based on them. Minimum-residual methods appear in white on gray, while Galerkin methods appear in white on black. Thick arrows indicate that a method is based on a process. Thin lines between processes indicate equivalence when the operator is symmetric. Dashed lines between processes indicate a block relationship. Double stroke arrows between processes indicate equivalence.

Although the focus of GPMR is on unsymmetric linear systems, Figure 6.3 shows that it is also relevant for ill-conditioned symmetric linear systems. Moreover, GPMR applies to symmetric partitioned systems with symmetric indefinite blocks M and N , whereas TRIMR requires them to be zero or definite matrices.

A limited-memory variant of GPMR in the spirit of DQGMRES [84] can be developed and might provide an adequate balance of performance and storage cost. We leave the investigation of such extensions to future work.

CHAPITRE 7 ARTICLE 4: MINARES: AN ITERATIVE SOLVER FOR SYMMETRIC LINEAR SYSTEMS

ALEXIS MONTISON, DOMINIQUE ORBAN, AND MICHAEL A. SAUNDERS

Manuscript submitted on September 28, 2023, to the *SIAM Journal on Matrix Analysis
and Applications* (SIMAX)

Abstract

We introduce an iterative solver named MINARES for symmetric linear systems $Ax \approx b$, where A is possibly singular. MINARES is based on the symmetric Lanczos process, like MINRES and MINRES-QLP, but it minimizes $\|Ar_k\|$ in each Krylov subspace rather than $\|r_k\|$, where r_k is the current residual vector. When A is symmetric, MINARES minimizes the same quantity $\|Ar_k\|$ as LSMR, but in more relevant Krylov subspaces, and it requires only one matrix-vector product Av per iteration, whereas LSMR would need two. Our numerical experiments with MINRES-QLP and LSMR show that MINARES is a pertinent alternative on consistent symmetric systems and the most suitable Krylov method for inconsistent symmetric systems. We derive properties of MINARES from an equivalent solver named CAR that is to MINARES as CR is to MINRES, is not based on the Lanczos process, and minimizes $\|Ar_k\|$ in the same Krylov subspace as MINARES. We establish that MINARES and CAR generate monotonic $\|x_k - x^*\|$, $\|x_k - x^*\|_A$ and $\|r_k\|$ when A is positive definite.

7.1 Introduction

Suppose $A \in \mathbb{R}^{n \times n}$ is a large symmetric matrix for which matrix-vector products Av can be computed efficiently for any vector $v \in \mathbb{R}^n$. We present a Krylov subspace method called MINARES for computing a solution to the following problems:

$$\text{Symmetric linear systems:} \quad Ax = b, \quad (7.1)$$

$$\text{Symmetric least-squares problems:} \quad \min \|Ax - b\|, \quad (7.2)$$

$$\text{Symmetric nullspace problems:} \quad Ar = 0, \quad (7.3)$$

$$\text{Symmetric eigenvalue problems:} \quad Ar = \lambda r, \quad (7.4)$$

$$\text{Singular value problems for rectangular } B: \quad \begin{bmatrix} & B \\ B^T & \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \sigma \begin{bmatrix} u \\ v \end{bmatrix}. \quad (7.5)$$

If A is nonsingular, problems (7.1)–(7.2) have a unique solution x^* . When A is singular, if b is not in the range of A then (7.1) has no solution; otherwise, (7.1)–(7.2) have an infinite number of solutions, and we seek the unique solution x^* that minimizes $\|x\|$. Whenever x^* exists, it solves the problem

$$\min \frac{1}{2}\|x\|^2 \quad \text{subject to} \quad A^2x = Ab. \quad (7.6)$$

Let x_k be an approximation to x^* with residual $r_k = b - Ax_k$. If A were unsymmetric or rectangular, applicable solvers for (7.1)–(7.2) would be LSQR [73] and LSMR [28], which reduce $\|r_k\|$ and $\|A^T r_k\|$ respectively within the k th Krylov subspace $\mathcal{K}_k(A^T A, A^T b)$ generated by the Golub-Kahan bidiagonalization on (A, b) [34].

For (7.1)–(7.5), we propose an algorithm MINARES that solves (7.6) by reducing $\|Ar_k\|$ within the k th Krylov subspace $\mathcal{K}_k(A, b)$ generated by the symmetric Lanczos process on (A, b) [55]. Thus when A is symmetric, MINARES minimizes the same quantity $\|Ar_k\|$ as LSMR, but in different (more effective) subspaces, and it requires only one matrix-vector product Av per iteration, whereas LSMR would need two.

Qualitatively, certain residual norms decrease smoothly for these iterative methods, but other norms are more erratic as they approach zero. It is ideal if stopping criteria involve the smooth quantities. For LSQR and LSMR on general (possibly rectangular) systems, $\|r_k\|$ decreases smoothly for both methods. We observe that while LSQR is always ahead by construction, it is never by very much. Thus on consistent systems $Ax = b$, LSQR may terminate slightly sooner. On inconsistent systems $Ax \approx b$, the comparison is more striking. $\|A^T r_k\|$ decreases erratically for LSQR but smoothly for LSMR, and there is usually a significance difference between the two. Thus LSMR may terminate significantly sooner [28].

Similarly for MINRES [72] and MINARES, $\|r_k\|$ decreases smoothly for both methods, and on consistent symmetric systems $Ax = b$, MINRES may have a small advantage. On inconsistent symmetric systems $Ax \approx b$, $\|Ar_k\|$ decreases erratically for MINRES and its variant MINRES-QLP [13] but smoothly for MINARES, and there is usually a significant difference between them. Thus MINARES may terminate sooner.

We introduce CAR, a new conjugate direction method similar to CG and CR and equivalent to MINARES when A is SPD. We prove that $\|r_k\|$, $\|x_k - x^*\|$ and $\|x_k - x^*\|_A$ decrease monotonically for CAR and hence MINARES when A is positive definite.

7.1.1 Notation

A symmetric positive definite matrix is said to be SPD. For a vector v_k , $\|v_k\|$ denotes the Euclidean norm of v_k , and for an SPD matrix A , the A -norm of v_k is $\|v_k\|_A^2 = v_k^T A v_k$. For a matrix V_k , $\|V_k\|$ may be any norm. Vector e_j is the j th column of an identity matrix I_k of size dictated by the context. An approximate solution x_k has residual $r_k = b - Ax_k$, and x^* is the unique solution of $Ax = b$ if A is nonsingular, or the minimum-norm solution of $A^2x = Ab$ otherwise. $\mathcal{K}_k(A, b)$ is the *Krylov subspace* $\{b, Ab, \dots, A^{k-1}b\}$. We abusively write $z = (\zeta_1, \dots, \zeta_n)$ to represent the column vector $z = \begin{bmatrix} \zeta_1 & \dots & \zeta_n \end{bmatrix}^T$. If H is SPD and $\{d_1, \dots, d_k\}$ is a set of non-zero vectors, the vectors are H -conjugate if $d_i^T H d_j = 0$ for $i \neq j$. If $H = I$, conjugacy is equivalent to the usual notion of orthogonality.

7.2 Applications

7.2.1 Null vector, eigenvector, and singular value problems

Given a symmetric A and nonzero b , MINARES solves $A^2x = Ab$ even if A is singular. If b is random and A is singular, $r = b - Ax$ is unlikely to be zero, but it will be a nonzero nullvector of A because $Ar = 0$.

If an eigenvalue λ of A is known, we can use it as a shift in the Lanczos process with a random starting vector b to find a null vector r such that $(A - \lambda I)r = 0$. Then r is an eigenvector because $Ar = \lambda r$. MINARES is effectively implementing the inverse power method [37, 90] to obtain the eigenvector in one iteration. If λ is approximate, MINARES can implement Rayleigh quotient iteration [37, 90] to obtain increasingly accurate eigenpair estimates.

Similarly, if a singular value σ is known for a rectangular matrix B , the singular value problem $Bv = \sigma u$, $B^T u = \sigma v$ may be reformulated as a null vector problem or eigenvalue problem:

$$\left(\begin{bmatrix} & B \\ B^T & \end{bmatrix} - \sigma I \right) \begin{bmatrix} u \\ v \end{bmatrix} = 0 \quad \Longleftrightarrow \quad \begin{bmatrix} & B \\ B^T & \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \sigma \begin{bmatrix} u \\ v \end{bmatrix},$$

for which MINARES may be used to implement inverse iteration or Rayleigh quotient iteration (although an algorithm based on the Golub-Kahan bidiagonalization of B would be preferable).

7.2.2 Singular systems with semi-positive definite matrices

Inconsistent (singular) symmetric systems could arise from discretized semidefinite Neumann boundary value problems [51, sect. 4]. Measurement errors will be random, so b is unlikely

to be in the range of singular A .

Another potential application is large, singular, symmetric, indefinite Toeplitz least-squares problems as described in [33, sec. 5]. Rank-deficient Toeplitz matrices arise in image reconstruction and system identification problems. In both cases, A is a semi-positive definite matrix and MINARES is a suitable solver.

7.3 Symmetric systems

Algorithm 15 Lanczos process

Require: A, b

```

1:  $v_0 = 0$ 
2:  $\beta_1 v_1 = b$   $\beta_1 > 0$  so that  $\|v_1\| = 1$ 
3: for  $k = 1, 2, \dots$  do
4:    $q_k = Av_k - \beta_k v_{k-1}$ 
5:    $\alpha_k = v_k^T q_k$ 
6:    $q_k = q_k - \alpha_k v_k$ 
7:    $\beta_{k+1} = \|q_k\|$ 
8:   if  $\beta_{k+1} = 0$  then
9:      $\ell = k$ ; return  $\ell$ 
10:  else
11:     $v_{k+1} = q_k / \beta_{k+1}$   $\beta_{k+1} > 0$  so that  $\|v_{k+1}\| = 1$ 
12:  end if
13: end for
```

With A symmetric and starting vector b , we make use of the symmetric Lanczos process [55] of Algorithm 15. After k iterations the situation may be summarized as

$$AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T = V_{k+1} T_{k+1,k}, \quad (7.7a)$$

$$V_k^T V_k = I_k, \quad (7.7b)$$

where

$$V_k := [v_1 \ \dots \ v_k], \quad T_k = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \beta_k & \alpha_k \end{bmatrix}, \quad T_{k+1,k} = \begin{bmatrix} T_k \\ \beta_{k+1} e_k^T \end{bmatrix}.$$

In exact arithmetic, V_k is an orthonormal basis of $\mathcal{K}_k(A, b)$. The Lanczos process terminates after $\ell \leq n$ iterations when $\beta_{\ell+1} = 0$, and we then have $AV_\ell = V_\ell T_\ell$, where square T_ℓ is nonsingular if and only if $b \in \text{range}(A)$ [13, sec. 2.1 property 4]. $T_{k+1,k}$ has full column rank k for all $k < \ell$ [13, sec. 2.1 property 2] and the rank of T_ℓ is ℓ or $\ell - 1$ but no less (because the first $\ell - 1$ columns of T_ℓ are independent).

In finite arithmetic, (7.7a) holds to machine precision. Reorthogonalization would be needed for (7.7b) to hold accurately, but it is enough to note that we always have $\|V_k\| = O(1)$.

7.3.1 CG, SYMMLQ, MINRES, MINARES

As with CG [48], SYMMLQ [72], and MINRES [72], the goal of MINARES is to solve symmetric problems $Ax \approx b$. All methods define an approximate solution $x_k = V_k y_k$ at iteration k (where y_k is different for each method). MINARES chooses y_k to minimize $\|Ar_k\|$ in $\mathcal{K}_k(A, b)$, so that $\|Ar_k\|$ is monotonically decreasing towards zero. MINARES is therefore well suited to singular inconsistent symmetric systems. This case is difficult for the other methods because $\|x_k - x^*\|_A$, $\|x_k - x^*\|$ and $\|r_k\|$ do not converge to zero and they are the quantities minimized respectively by CG, SYMMLQ, and both MINRES and MINRES-QLP.

7.4 Derivation of MINARES

7.4.1 Subproblems of MINARES

From Algorithm 15 we have $Ab = \beta_1 \alpha_1 v_1 + \beta_1 \beta_2 v_2$ because $\beta_2 v_2 = Av_1 - \alpha_1 v_1$. Hence

$$\begin{aligned} Ar_k &= A(b - AV_k y_k) \\ &= Ab - AV_{k+1} T_{k+1,k} y_k \\ &= \beta_1 \alpha_1 v_1 + \beta_1 \beta_2 v_2 - V_{k+2} T_{k+2,k+1} T_{k+1,k} y_k \\ &= V_{k+2} (\beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2 - T_{k+2,k+1} T_{k+1,k} y_k), \quad k \leq \ell - 2, \end{aligned} \tag{7.8a}$$

$$Ar_{\ell-1} = V_\ell (\beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2 - T_\ell T_{\ell,\ell-1} y_{\ell-1}), \tag{7.8b}$$

$$Ar_\ell = V_\ell (\beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2 - T_\ell^2 y_\ell). \tag{7.8c}$$

Theoretically, V_k has orthonormal columns ($1 \leq k \leq \ell$), so that $\|x_k\| = \|y_k\|$ and $\|Ar_k\|$ is minimized with $\|x_k\|$ of minimal norm if we define y_k as the unique solution of the following subproblems:

$$\underset{y_k \in \mathbb{R}^k}{\text{minimize}} \quad \|T_{k+2,k+1} T_{k+1,k} y_k - \beta_1 \alpha_1 e_1 - \beta_1 \beta_2 e_2\|, \quad k \leq \ell - 2, \tag{7.9a}$$

$$\underset{y_{\ell-1} \in \mathbb{R}^{\ell-1}}{\text{minimize}} \quad \|T_\ell T_{\ell,\ell-1} y_{\ell-1} - \beta_1 \alpha_1 e_1 - \beta_1 \beta_2 e_2\|, \tag{7.9b}$$

$$\underset{y_\ell \in \mathbb{R}^\ell}{\text{minimize}} \quad \|y_\ell\|^2 \quad \text{subject to} \quad T_\ell^2 y_\ell = \beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2. \tag{7.9c}$$

We define y_k from these subproblems even though V_k does not remain orthonormal numerically. In practice, we expect $\|Ar_k\| \leq \|Ar_{k-1}\|$ unless k becomes too large.

To be sure that the subproblems have unique solutions, we need to verify that $T_{k+2,k+1}T_{k+1,k}$ has rank k ($k \leq \ell - 2$), $T_\ell T_{\ell,\ell-1}$ has rank $\ell - 1$, and $T_\ell^2 y_\ell = \beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2$ is consistent even if T_ℓ is singular. These results are proved in [Theorem 4](#), [Theorem 5](#) and [Theorem 6](#).

Theorem 4. *For $k \leq \ell - 2$, $T_{k+2,k+1}T_{k+1,k}$ has rank k .*

Proof. From (7.13a) and (7.15) we have

$$T_{k+2,k+1}T_{k+1,k} = \begin{bmatrix} R_k^T R_k \\ (\varepsilon_{k-1} e_{k-1}^T + \gamma_k e_k^T) R_k \\ \varepsilon_k e_k^T R_k \end{bmatrix},$$

where $R_k^T R_k$ has rank k because $T_{k+1,k}$ and hence R_k have full column rank. $\square$

Theorem 5. *$T_\ell T_{\ell,\ell-1}$ has rank $\ell - 1$.*

Proof. From (7.13b) and (7.15) we have

$$T_\ell T_{\ell,\ell-1} = \begin{bmatrix} R_{\ell-1}^T R_{\ell-1} \\ (\varepsilon_{\ell-1} e_{\ell-1}^T + \gamma_\ell e_\ell^T) R_{\ell-1} \end{bmatrix},$$

where $R_{\ell-1}^T R_{\ell-1}$ has rank $\ell - 1$ because $T_{\ell,\ell-1}$ and $R_{\ell-1}$ have full column rank. $\square$

Theorem 6. *$T_\ell^2 y_\ell = \beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2$ is consistent even if T_ℓ is singular.*

Proof. If T_ℓ is singular, the symmetry of T_ℓ and its complete orthogonal decomposition give

$$T_\ell = Q \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} P = P^T \begin{bmatrix} L^T & 0 \\ 0 & 0 \end{bmatrix} Q^T \quad \text{and} \quad T_\ell^2 = P^T \begin{bmatrix} L^T L & 0 \\ 0 & 0 \end{bmatrix} P,$$

where Q and P are orthogonal and $\text{rank}(L) = \ell - 1$. Thus,

$$\begin{aligned} T_\ell^2 y_\ell - \beta_1 \alpha_1 e_1 - \beta_1 \beta_2 e_2 &= T_\ell^2 y_\ell - \beta_1 T_\ell e_1 \\ &= P^T \left(\begin{bmatrix} L^T L & 0 \\ 0 & 0 \end{bmatrix} P y_\ell - \beta_1 \begin{bmatrix} L^T & 0 \\ 0 & 0 \end{bmatrix} Q^T e_1 \right) \\ &= P^T \begin{bmatrix} L^T L t_{\ell-1} - L^T u_{\ell-1} \\ 0 \end{bmatrix}, \end{aligned}$$

where $t_{\ell-1}$ and $u_{\ell-1}$ are the first $\ell - 1$ components of Py_ℓ and $\beta_1 Q^T e_1$. Because L has full rank, $L^T L t_{\ell-1} = L^T u_{\ell-1}$ has a unique solution. Then, $y_\ell = P^T \begin{bmatrix} t_{\ell-1} \\ \omega \end{bmatrix}$ is a solution of $T_\ell^2 y_\ell = \beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2$ for any ω , which means the system is consistent. $\square$

From (7.8c) and Theorem 6, $Ar_\ell = V_\ell(T_\ell^2 y_\ell - \beta_1 \alpha_1 e_1 - \beta_1 \beta_2 e_2) = 0$. Hence with definition (7.9c) we can conclude that x_ℓ is the solution x^* of (7.6).

7.4.2 QR factorization of T_k

To solve (7.9), we first need the QR factorization used by MINRES:

$$T_{k+1,k} = Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \quad R_k = \begin{bmatrix} \lambda_1 & \gamma_1 & \varepsilon_1 & & & \\ & \lambda_2 & \gamma_2 & \ddots & & \\ & & \lambda_3 & \ddots & \varepsilon_{k-2} & \\ & & & \ddots & \gamma_{k-1} & \\ & & & & \lambda_k & \end{bmatrix}, \quad (7.10)$$

where $Q_k^T = Q_{k+1,k} \dots Q_{3,2} Q_{2,1}$ is an orthogonal matrix defined as a product of 2×2 reflections with the structure

$$Q_{i+1,i} = \begin{matrix} & \begin{matrix} 1 & \dots & i-1 & i & i+1 & i+2 & \dots & k \end{matrix} \\ \begin{matrix} 1 \\ \vdots \\ i-1 \\ i \\ i+1 \\ i+2 \\ \vdots \\ k \end{matrix} & \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c_i & s_i & & & \\ & & & s_i & -c_i & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \end{matrix}.$$

If we initialize $Q_0 := I$, $\bar{\lambda}_1 := \alpha_1$, $\bar{\gamma}_1 := \beta_2$, individual factorization steps may be represented as an application of $Q_{k+1,k}$ to $Q_{k-1}^T T_{k+1,k}$:

$$\begin{matrix} & \begin{matrix} k & k+1 \end{matrix} \\ \begin{matrix} k \\ k+1 \end{matrix} & \begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \end{matrix} \begin{bmatrix} \bar{\lambda}_k \\ \beta_{k+1} \end{bmatrix} = \begin{matrix} & \begin{matrix} k+1 & k+2 \end{matrix} \\ \begin{matrix} k \\ k+1 \end{matrix} & \begin{bmatrix} \bar{\gamma}_k & 0 \\ \alpha_{k+1} & \beta_{k+2} \end{bmatrix} \end{matrix} = \begin{matrix} & \begin{matrix} k & k+1 & k+2 \end{matrix} \\ \begin{matrix} k \\ k+1 \end{matrix} & \begin{bmatrix} \lambda_k & \gamma_k & \varepsilon_k \\ 0 & \bar{\lambda}_{k+1} & \bar{\gamma}_{k+1} \end{bmatrix} \end{matrix}.$$

The reflection $Q_{k+1,k}$ zeroes β_{k+1} on the subdiagonal of $T_{k+1,k}$ and affects three columns and two rows. It is defined by

$$\lambda_k = \sqrt{\bar{\lambda}_k^2 + \beta_{k+1}^2}, \quad c_k = \bar{\lambda}_k / \lambda_k, \quad s_k = \beta_{k+1} / \lambda_k, \quad (7.11)$$

and yields the following recursion for $k \geq 1$:

$$\gamma_k = c_k \bar{\gamma}_k + s_k \alpha_{k+1}, \quad (7.12a)$$

$$\bar{\lambda}_{k+1} = s_k \bar{\gamma}_k - c_k \alpha_{k+1}, \quad (7.12b)$$

$$\varepsilon_k = s_k \beta_{k+2}, \quad (7.12c)$$

$$\bar{\gamma}_{k+1} = -c_k \beta_{k+2}. \quad (7.12d)$$

7.4.3 Definition of N_k

Let us define

$$N_k := T_{k+2,k+1} Q_k \begin{bmatrix} I_k \\ 0 \end{bmatrix}, \quad \text{where} \quad N_k R_k = T_{k+2,k+1} T_{k+1,k}, \quad k \leq \ell - 2, \quad (7.13a)$$

$$N_{\ell-1} := T_{\ell,\ell-1} Q_{\ell-1} \begin{bmatrix} I_{\ell-1} \\ 0 \end{bmatrix}, \quad \text{where} \quad N_{\ell-1} R_{\ell-1} = T_{\ell} T_{\ell,\ell-1}, \quad (7.13b)$$

$$N_{\ell} := T_{\ell} Q_{\ell}, \quad \text{where} \quad N_{\ell} R_{\ell} = T_{\ell}^2. \quad (7.13c)$$

Because $Q_k = Q_{2,1} Q_{3,2} \dots Q_{k+1,k}$, we have

$$e_k^T Q_k = e_k^T Q_{k,k-1} Q_{k+1,k} = s_{k-1} e_{k-1}^T - c_{k-1} c_k e_k^T - c_{k-1} s_k e_{k+1}^T, \quad (7.14a)$$

$$e_{k+1}^T Q_k = e_{k+1}^T Q_{k+1,k} = s_k e_k^T - c_k e_{k+1}^T. \quad (7.14b)$$

Moreover, $T_{k+2,k+1} = \begin{bmatrix} T_{k+1,k}^T \\ \beta_{k+1} e_k^T + \alpha_{k+1} e_{k+1}^T \\ \beta_{k+2} e_{k+1}^T \end{bmatrix}$ and the product $T_{k+2,k+1} Q_k$ can be determined in three parts. From (7.10), $T_{k+1,k}^T Q_k = (Q_k^T T_{k+1,k})^T = \begin{bmatrix} R_k^T & 0 \end{bmatrix}$, and from (7.14) we have

$$\begin{aligned} (\beta_{k+1} e_k^T + \alpha_{k+1} e_{k+1}^T) Q_k &= \beta_{k+1} s_{k-1} e_{k-1}^T + (\alpha_{k+1} s_k - \beta_{k+1} c_{k-1} s_k) e_k^T \\ &\quad - (\alpha_{k+1} c_k + \beta_{k+1} c_{k-1} s_k) e_{k+1}^T \\ &= \varepsilon_{k-1} e_{k-1}^T + \gamma_k e_k^T - (\alpha_{k+1} c_k + \beta_{k+1} c_{k-1} s_k) e_{k+1}^T, \\ \beta_{k+2} e_{k+1}^T Q_k &= s_k \beta_{k+2} e_k^T - c_k \beta_{k+2} e_{k+1}^T \\ &= \varepsilon_k e_k^T - c_k \beta_{k+2} e_{k+1}^T. \end{aligned}$$

Thus, for $k \leq \ell - 2$ we obtain

$$N_k = \begin{bmatrix} R_k^T \\ \varepsilon_{k-1}e_{k-1}^T + \gamma_k e_k^T \\ \varepsilon_k e_k^T \end{bmatrix}, \quad N_{\ell-1} = \begin{bmatrix} R_{\ell-1}^T \\ \varepsilon_{\ell-1}e_{\ell-1}^T + \gamma_\ell e_\ell^T \end{bmatrix}, \quad N_\ell = R_\ell^T. \quad (7.15)$$

7.4.4 QR factorization of N_k

$$N_k = \tilde{Q}_k \begin{bmatrix} U_k \\ 0 \end{bmatrix}, \quad U_k = \begin{bmatrix} \mu_1 & \phi_1 & \rho_1 & & \\ & \mu_2 & \phi_2 & \ddots & \\ & & \mu_3 & \ddots & \rho_{k-2} \\ & & & \ddots & \phi_{k-1} \\ & & & & \mu_k \end{bmatrix}, \quad (7.16)$$

where $\tilde{Q}_k^T = \tilde{Q}_{k+2,k} \tilde{Q}_{k+1,k} \dots \tilde{Q}_{3,1} \tilde{Q}_{2,1}$ for $k \leq \ell - 2$, and $\tilde{Q}_\ell^T = \tilde{Q}_{\ell-1}^T = \tilde{Q}_{\ell,\ell-1} \tilde{Q}_{\ell-2}^T$ are orthogonal matrices defined as a product of reflections. If we initialize $\bar{\mu}_1 := \lambda_1$, $\hat{\gamma}_1 := \gamma_1$ and $\hat{\lambda}_2 := \lambda_2$, individual factorization steps may be represented as an application of $\tilde{Q}_{k+1,k}$ to $\tilde{Q}_{k-1}^T N_k$:

$$\begin{array}{c} k \\ k+1 \\ k+2 \end{array} \begin{bmatrix} & k & k+1 & k+2 \\ \tilde{c}_{2k-1} & & \tilde{s}_{2k-1} & \\ \tilde{s}_{2k-1} & -\tilde{c}_{2k-1} & & \\ & & & 1 \end{bmatrix} \begin{bmatrix} k & k+1 & k+2 \\ \bar{\mu}_k & & \\ \hat{\gamma}_k & \hat{\lambda}_{k+1} & \\ \varepsilon_k & \gamma_{k+1} & \lambda_{k+2} \end{bmatrix} = \begin{bmatrix} k & k+1 & k+2 \\ \bar{\mu}_k & \bar{\phi}_k & \\ \bar{\mu}_{k+1} & & \\ \varepsilon_k & \gamma_{k+1} & \lambda_{k+2} \end{bmatrix},$$

followed by an application of $Q_{k+2,k}$ to the result:

$$\begin{array}{c} k \\ k+1 \\ k+2 \end{array} \begin{bmatrix} & k & k+1 & k+2 \\ \tilde{c}_{2k} & & \tilde{s}_{2k} & \\ & 1 & & \\ \tilde{s}_{2k} & & -\tilde{c}_{2k} & \end{bmatrix} \begin{bmatrix} k & k+1 & k+2 \\ \bar{\mu}_k & \bar{\phi}_k & \\ \bar{\mu}_{k+1} & & \\ \varepsilon_k & \gamma_{k+1} & \lambda_{k+2} \end{bmatrix} = \begin{bmatrix} k & k+1 & k+2 \\ \mu_k & \phi_k & \rho_k \\ & \bar{\mu}_{k+1} & \\ & \hat{\gamma}_{k+1} & \hat{\lambda}_{k+2} \end{bmatrix}.$$

The reflections $\tilde{Q}_{k+1,k}$ and $\tilde{Q}_{k+2,k}$ zero γ_k and ε_k on the subdiagonals of N_k :

$$\bar{\mu}_k = \sqrt{\bar{\mu}_k^2 + \hat{\gamma}_k^2}, \quad \tilde{c}_{2k-1} = \bar{\mu}_k / \bar{\mu}_k, \quad \tilde{s}_{2k-1} = \hat{\gamma}_k / \bar{\mu}_k, \quad k \leq \ell - 1, \quad (7.17a)$$

$$\mu_k = \sqrt{\bar{\mu}_k^2 + \varepsilon_k^2}, \quad \tilde{c}_{2k} = \bar{\mu}_k / \mu_k, \quad \tilde{s}_{2k} = \varepsilon_k / \mu_k, \quad k \leq \ell - 2, \quad (7.17b)$$

and they yield the recursion

$$\bar{\phi}_k = \tilde{s}_{2k-1}\hat{\lambda}_{k+1}, \quad 1 \leq k \leq \ell-1, \quad (7.18a)$$

$$\bar{\mu}_{k+1} = -\tilde{c}_{2k-1}\hat{\lambda}_{k+1}, \quad 1 \leq k \leq \ell-1, \quad (7.18b)$$

$$\phi_k = \tilde{c}_{2k}\bar{\phi}_k + \tilde{s}_{2k}\gamma_{k+1}, \quad 1 \leq k \leq \ell-2, \quad (7.18c)$$

$$\hat{\gamma}_{k+1} = \tilde{s}_{2k}\bar{\phi}_k - \tilde{c}_{2k}\gamma_{k+1}, \quad 1 \leq k \leq \ell-2, \quad (7.18d)$$

$$\rho_k = \tilde{s}_{2k}\lambda_{k+2}, \quad 1 \leq k \leq \ell-2, \quad (7.18e)$$

$$\hat{\lambda}_{k+2} = -\tilde{c}_{2k}\lambda_{k+2}, \quad 1 \leq k \leq \ell-2, \quad (7.18f)$$

$$\mu_{\ell-1} = \bar{\mu}_{\ell-1}, \quad (7.18g)$$

$$\phi_{\ell-1} = \bar{\phi}_{\ell-1}, \quad (7.18h)$$

$$\mu_{\ell} = \bar{\mu}_{\ell}. \quad (7.18i)$$

From (7.8) and (7.16) we have

$$\|Ar_k\| = \|N_k R_k y_k - \beta_1 \alpha_1 e_1 - \beta_1 \beta_2 e_2\| = \left\| \begin{bmatrix} U_k \\ 0 \end{bmatrix} R_k y_k - \bar{z}_k \right\|, \quad (7.19)$$

where $\bar{z}_k := \tilde{Q}_k^T(\beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2) = (z_k, \bar{\bar{\zeta}}_{k+1}, \bar{\zeta}_{k+2})$, $k \leq \ell-2$, $z_k = (\zeta_1, \dots, \zeta_k)$ represents the first k components of $\bar{z}_k$, and the recurrence starts with $\bar{z}_0 := (\bar{\bar{\zeta}}_1, \bar{\zeta}_2) = (\beta_1 \alpha_1, \beta_1 \beta_2)$. We can determine $\bar{z}_k$ from $\bar{z}_{k-1}$ because $\bar{z}_k = \tilde{Q}_{k+2,k} \tilde{Q}_{k+1,k}(\bar{z}_{k-1}, 0)$ for $k \leq \ell-2$:

$$\begin{array}{c} k & k+1 & k+2 \\ \begin{array}{c} k \\ k+1 \\ k+2 \end{array} & \begin{bmatrix} \tilde{c}_{2k} & & \tilde{s}_{2k} \\ & 1 & \\ \tilde{s}_{2k} & & -\tilde{c}_{2k} \end{bmatrix} & \begin{array}{c} k & k+1 & k+2 \\ \begin{bmatrix} \tilde{c}_{2k-1} & \tilde{s}_{2k-1} & \\ \tilde{s}_{2k-1} & -\tilde{c}_{2k-1} & \\ & & 1 \end{bmatrix} & \begin{bmatrix} \bar{\bar{\zeta}}_k \\ \bar{\zeta}_{k+1} \\ 0 \end{bmatrix} & = & \begin{bmatrix} \zeta_k \\ \bar{\bar{\zeta}}_{k+1} \\ \bar{\zeta}_{k+2} \end{bmatrix} \end{array}$$

and $z_{\ell} = z_{\ell-1} = \tilde{Q}_{\ell,\ell-1} \bar{z}_{\ell-2}$. The elements are updated according to

$$\overset{\circ}{\zeta}_k = \tilde{c}_{2k-1}\bar{\bar{\zeta}}_k + \tilde{s}_{2k-1}\bar{\zeta}_{k+1}, \quad k \leq \ell-1, \quad (7.20a)$$

$$\bar{\bar{\zeta}}_{k+1} = \tilde{s}_{2k-1}\bar{\bar{\zeta}}_k - \tilde{c}_{2k-1}\bar{\zeta}_{k+1}, \quad k \leq \ell-1, \quad (7.20b)$$

$$\zeta_k = \tilde{c}_{2k}\overset{\circ}{\zeta}_k, \quad k \leq \ell-2, \quad (7.20c)$$

$$\bar{\zeta}_{k+2} = \tilde{s}_{2k}\overset{\circ}{\zeta}_k, \quad k \leq \ell-2, \quad (7.20d)$$

$$\zeta_{\ell-1} = \overset{\circ}{\zeta}_{\ell-1}, \quad (7.20e)$$

$$\zeta_{\ell} = \bar{\bar{\zeta}}_{\ell}. \quad (7.20f)$$

For $k \leq \ell - 1$, U_k and R_k are nonsingular, and from (7.19), $\|Ar_k\|$ is minimized when $U_k R_k y_k = z_k$, giving

$$\|Ar_k\| = \sqrt{\bar{\zeta}_{k+1}^2 + \bar{\zeta}_{k+2}^2}, \quad k \leq \ell - 2, \quad \|Ar_{\ell-1}\| = |\zeta_\ell|. \quad (7.21)$$

7.4.5 Computation of x_k

Suppose R_k and U_k are nonsingular. If we were to update x_k directly from $x_k = V_k y_k$, all components of y_k would have to be recomputed because of the backward substitutions required to solve $U_k R_k y_k = z_k$, which would require us to store V_k entirely. To avoid such drawbacks, we employ the strategy of Paige and Saunders [72]. Thus, we define W_k and D_k by the lower triangular systems $R_k^T W_k^T = V_k^T$ and $U_k^T D_k^T = W_k^T$. Then

$$x_k = V_k y_k = W_k R_k y_k = D_k U_k R_k y_k = D_k z_k. \quad (7.22)$$

The columns of W_k and D_k are obtained from the recursions

$$\begin{aligned} w_1 &= v_1/\lambda_1, & w_2 &= (v_2 - \gamma_1 w_1)/\lambda_2, \\ w_k &= (v_k - \gamma_{k-1} w_{k-1} - \varepsilon_{k-2} w_{k-2})/\lambda_k, & k &\geq 3, \\ d_1 &= w_1/\mu_1, & d_2 &= (w_2 - \phi_1 d_1)/\mu_2, \\ d_k &= (w_k - \phi_{k-1} d_{k-1} - \rho_{k-2} d_{k-2})/\mu_k, & k &\geq 3, \end{aligned}$$

and the solution $x_k = D_k z_k$ may be updated efficiently via $x_0 = 0$ and

$$x_k = x_{k-1} + \zeta_k d_k. \quad (7.23)$$

This is possible for all $k \leq \ell$ if $Ax = b$ is consistent, and $k \leq \ell - 1$ otherwise. If $Ax = b$ is consistent, from Theorem 7, the final MINARES iterate x_ℓ satisfies $r_\ell = 0$ and is the minimum-length solution. If $Ax = b$ is inconsistent, from Theorem 8, $Ar_{\ell-1} = 0$. We obtain a solution x that satisfies $A^2 x = Ab$ in both cases.

Theorem 7. *If $b \in \text{range}(A)$, the final MINARES iterate x_ℓ is the minimum-length solution of $Ax = b$ (and $r_\ell = b - Ax_\ell = 0$).*

Proof. The final MINARES subproblem is $T_\ell^2 y_\ell = \beta_1 \alpha_1 e_1 + \beta_1 \beta_2 e_2 = T_\ell \beta_1 e_1$. Because $b \in \text{range}(A)$, T_ℓ is nonsingular, and the latter system is equivalent to $T_\ell y_\ell = \beta_1 e_1$, the subproblem solved by MINRES and MINRES-QLP. The final iterate generated by these methods is the minimum-length solution of $Ax = b$ [13, sec. 3.2 theorem 3.1]. $\square$

Theorem 8. *If $Ax = b$ is inconsistent, $\zeta_\ell = 0$ and $Ar_{\ell-1} = 0$.*

Proof. From (7.13c), (7.16) and Theorem 6:

$$z_\ell = \tilde{Q}_\ell^T(\beta_1\alpha_1e_1 + \beta_1\beta_2e_2) = \tilde{Q}_\ell^TT_\ell^2y_\ell = \tilde{Q}_\ell^TN_\ell R_\ell y_\ell = U_\ell R_\ell y_\ell.$$

When $Ax = b$ is inconsistent, T_ℓ has rank $\ell - 1$ and $r_{\ell\ell} = 0$. Because R_ℓ and U_ℓ are upper triangular matrices, $\zeta_\ell = u_{\ell\ell}r_{\ell\ell}v_\ell = 0$, where v_ℓ is the last component of y_ℓ . From (7.21), $Ar_{\ell-1} = 0$ when $\zeta_\ell = 0$. $\square$

If the minimum-norm solution is not required, such as problems (7.3)–(7.5), we can stop with $x_{\ell-1}$ and avoid the computation of $x_\ell = x^*$. We can also stop with $x_{\ell-1}$ if a preconditioner is used because the minimum-norm solution is determined in a non-Euclidean norm.

We summarize the complete procedure as Algorithm 16.

7.5 Stopping rules

The end of Algorithm 16 shows how $\|r_k\|$ and $\|Ar_k\|$ are estimated. They are needed for use within stopping rules. The required norm estimates are derived next.

7.5.1 Estimating $\|r_k\|$

To compute $\|r_k\|$, we need an LQ factorization

$$U_k = \hat{L}_k \hat{P}_k, \quad \hat{L}_k = \begin{bmatrix} \psi_1 & & & & & & \\ \theta_1 & \psi_2 & & & & & \\ \omega_1 & \theta_2 & \psi_3 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \psi_{k-2} & \\ & & & & \ddots & \theta_{k-2} & \bar{\psi}_{k-1} \\ & & & & & \omega_{k-2} & \bar{\theta}_{k-1} & \bar{\psi}_k \end{bmatrix}, \quad (7.24)$$

where $\hat{P}_1^T = I$, $\hat{P}_2^T = \hat{P}_{1,2}$, and $\hat{P}_k^T = \hat{P}_{k-1}^T \hat{P}_{k-2,k} \hat{P}_{k-1,k}$ ($k \geq 3$) are orthogonal. Note that $\hat{L}_k$ is the L factor of a QLP decomposition of N_k . If we initialize $\bar{\psi}_1 := \mu_1$, $\hat{P}_{1,2}$ is defined to zero ϕ_1 :

$$\begin{bmatrix} \bar{\psi}_1 & \phi_1 \\ & \mu_2 \end{bmatrix} \begin{bmatrix} \hat{c}_1 & \hat{s}_1 \\ \hat{s}_1 & -\hat{c}_1 \end{bmatrix} = \begin{bmatrix} \bar{\bar{\psi}}_1 & \\ \bar{\theta}_1 & \bar{\bar{\psi}}_2 \end{bmatrix},$$

Algorithm 16 MINARES

Require: $A, b, \epsilon_r > 0, \epsilon_{Ar} > 0, k_{\max} > 0$

```

1:  $k = 0, x_0 = 0$ 
2:  $w_{-1} = w_0 = 0, d_{-1} = d_0 = 0$ 
3:  $\varepsilon_{-1} = \varepsilon_0 = \gamma_0 = 0, \rho_{-1} = \rho_0 = \phi_0 = 0$ 
4:  $\beta_1 v_1 = b, q_1 = Av_1, \alpha_1 = v_1^T q_1$ 
5:  $\bar{q}_1 = q_1 - \alpha_1 v_1, \beta_2 v_2 = q_1$ 
6:  $\bar{\zeta}_1 = \beta_1 \alpha_1, \bar{\zeta}_2 = \beta_1 \beta_2$ 
7:  $\bar{\chi}_1 = \beta_1, \bar{\lambda}_1 = \alpha_1, \bar{\gamma}_1 = \beta_2$ 
8:  $\|r_0\| = \bar{\chi}_1, \|Ar_0\| = (\bar{\zeta}_1^2 + \bar{\zeta}_2^2)^{\frac{1}{2}}$ 
9: while  $\|r_k\| > \epsilon_r$  and  $\|Ar_k\| > \epsilon_{Ar}$  and  $k \leq k_{\max}$  do
10:    $k \leftarrow k + 1$ 
11:    $q_{k+1} = Av_{k+1} - \beta_{k+1} v_k, \alpha_{k+1} = v_{k+1}^T q_{k+1}$ 
12:    $q_{k+1} = q_{k+1} - \alpha_{k+1} v_{k+1}, \beta_{k+2} v_{k+2} = q_{k+1}$ 
13:    $\lambda_k = (\bar{\lambda}_k^2 + \beta_{k+1}^2)^{\frac{1}{2}}, c_k = \bar{\lambda}_k / \lambda_k, s_k = \beta_{k+1} / \lambda_k$ 
14:    $\gamma_k = c_k \bar{\gamma}_k + s_k \alpha_{k+1}, \varepsilon_k = s_k \beta_{k+2}$ 
15:    $\bar{\lambda}_{k+1} = s_k \bar{\gamma}_k - c_k \alpha_{k+1}, \bar{\gamma}_{k+1} = -c_k \beta_{k+2}$ 
16:   if  $k == 1$  then
17:      $\bar{\mu}_k = \lambda_k, \hat{\gamma}_k = \gamma_k$ 
18:   else
19:     if  $k == 2$  then
20:        $\hat{\lambda}_k = \lambda_k$ 
21:     else
22:        $\rho_{k-2} = \tilde{s}_{2k-4} \lambda_k, \hat{\lambda}_k = -\tilde{c}_{2k-4} \lambda_k$ 
23:     end if
24:      $\bar{\phi}_{k-1} = \tilde{s}_{2k-3} \hat{\lambda}_k, \phi_{k-1} = \tilde{c}_{2k-2} \bar{\phi}_{k-1} + \tilde{s}_{2k-2} \gamma_k$ 
25:      $\bar{\mu}_k = -\tilde{c}_{2k-3} \hat{\lambda}_k, \hat{\gamma}_k = \tilde{s}_{2k-2} \bar{\phi}_{k-1} - \tilde{c}_{2k-2} \gamma_k$ 
26:   end if
27:    $\bar{\bar{\mu}}_k = (\bar{\mu}_k^2 + \hat{\gamma}_k^2)^{\frac{1}{2}}, \tilde{c}_{2k-1} = \bar{\mu}_k / \bar{\bar{\mu}}_k, \tilde{s}_{2k-1} = \hat{\gamma}_k / \bar{\bar{\mu}}_k$ 
28:    $\mu_k = (\bar{\mu}_k^2 + \varepsilon_k^2)^{\frac{1}{2}}, \tilde{c}_{2k} = \bar{\mu}_k / \mu_k, \tilde{s}_{2k} = \varepsilon_k / \mu_k$ 
29:    $\bar{\zeta}_k = \tilde{c}_{2k-1} \bar{\zeta}_k + \tilde{s}_{2k-1} \bar{\zeta}_{k+1}, \zeta_k = \tilde{c}_{2k} \bar{\zeta}_k$ 
30:    $\bar{\zeta}_{k+1} = \tilde{s}_{2k-1} \bar{\zeta}_k - \tilde{c}_{2k-1} \bar{\zeta}_{k+1}, \bar{\zeta}_{k+2} = \tilde{s}_{2k} \zeta_k$ 
31:    $w_k = (v_k - \gamma_{k-1} w_{k-1} - \varepsilon_{k-2} w_{k-2}) / \lambda_k$ 
32:    $d_k = (w_k - \phi_{k-1} d_{k-1} - \rho_{k-2} d_{k-2}) / \mu_k$ 
33:    $x_k = x_{k-1} + \zeta_k d_k$ 
34:    $\|Ar_k\| = (\bar{\zeta}_{k+1}^2 + \bar{\zeta}_{k+2}^2)^{\frac{1}{2}}$ 

```

where

$$\bar{\bar{\psi}}_1 = \sqrt{\bar{\psi}_1^2 + \phi_1^2}, \quad \hat{c}_1 = \bar{\psi}_1 / \bar{\bar{\psi}}_1, \quad \hat{s}_1 = \phi_1 / \bar{\bar{\psi}}_1, \quad \bar{\theta}_1 = \hat{s}_1 \mu_2, \quad \bar{\psi}_2 = -\hat{c}_1 \mu_2. \quad (7.25)$$

```

35:    $\chi_k = c_k \bar{\chi}_k, \quad \bar{\chi}_{k+1} = s_k \bar{\chi}_k$ 
36:   if  $k == 1$  then
37:      $\bar{\psi}_k = \mu_k, \quad \bar{\pi}_{k-1} = 0, \quad \bar{\pi}_k = \chi_k$ 
38:      $\xi_k = \zeta_k, \quad \bar{\tau}_{k-1} = 0, \quad \bar{\tau}_k = \xi_k / \bar{\psi}_k$ 
39:   else if  $k == 2$  then
40:      $\bar{\psi}_{k-1} = (\bar{\psi}_{k-1}^2 + \phi_{k-1}^2)^{\frac{1}{2}}, \quad \hat{c}_{k-1} = \bar{\psi}_{k-1} / \bar{\psi}_{k-1}, \quad \hat{s}_{k-1} = \phi_{k-1} / \bar{\psi}_{k-1}$ 
41:      $\bar{\theta}_{k-1} = \hat{s}_{2k-3} \mu_k, \quad \bar{\psi}_k = -\hat{c}_{2k-3} \mu_k$ 
42:      $\bar{\pi}_{k-1} = \hat{c}_{2k-3} \bar{\pi}_{k-1} + \hat{s}_{2k-3} \chi_k, \quad \bar{\pi}_k = \hat{s}_{2k-3} \bar{\pi}_{k-1} - \hat{c}_{2k-3} \chi_k$ 
43:      $\xi_k = \zeta_k, \quad \bar{\tau}_{k-1} = \xi_{k-1} / \bar{\psi}_{k-1}, \quad \bar{\tau}_k = (\xi_k - \bar{\theta}_{k-1} \bar{\tau}_{k-1}) / \bar{\psi}_k$ 
44:   else
45:      $\bar{\psi}_{k-2} = (\bar{\psi}_{k-2}^2 + \rho_{k-2}^2)^{\frac{1}{2}}, \quad \hat{c}_{2k-4} = \bar{\psi}_{k-2} / \bar{\psi}_{k-2}, \quad \hat{s}_{2k-4} = \rho_{k-2} / \bar{\psi}_{k-2}$ 
46:      $\bar{\psi}_{k-1} = (\bar{\psi}_{k-1}^2 + \delta_k^2)^{\frac{1}{2}}, \quad \hat{c}_{2k-3} = \bar{\psi}_{k-1} / \bar{\psi}_{k-1}, \quad \hat{s}_{2k-3} = \delta_k / \bar{\psi}_{k-1}$ 
47:      $\theta_{k-2} = \hat{c}_{2k-4} \bar{\theta}_{k-2} + \hat{s}_{2k-4} \phi_{k-1}, \quad \omega_{k-2} = \hat{s}_{2k-4} \mu_k$ 
48:      $\delta_k = \hat{s}_{2k-4} \bar{\theta}_{k-2} - \hat{c}_{2k-4} \phi_{k-1}, \quad \eta_k = -\hat{c}_{2k-4} \mu_k$ 
49:      $\bar{\theta}_{k-1} = \hat{s}_{2k-3} \eta_k, \quad \bar{\psi}_k = -\hat{c}_{2k-3} \eta_k, \quad v_k = \hat{s}_{2k-4} \bar{\pi}_{k-2} - \hat{c}_{2k-4} \chi_k$ 
50:      $\bar{\pi}_{k-1} = \hat{c}_{2k-3} \bar{\pi}_{k-1} + \hat{s}_{2k-3} v_k, \quad \bar{\pi}_k = \hat{s}_{2k-3} \bar{\pi}_{k-1} - \hat{c}_{2k-3} v_k$ 
51:      $\tau_{k-2} = \bar{\tau}_{k-2} \bar{\psi}_{k-2} / \bar{\psi}_{k-2}, \quad \xi_k = \zeta_k - \omega_{k-2} \tau_{k-2}$ 
52:      $\bar{\tau}_{k-1} = (\xi_{k-1} - \theta_{k-2} \tau_{k-2}) / \bar{\psi}_{k-1}, \quad \bar{\tau}_k = (\xi_k - \bar{\theta}_{k-1} \bar{\tau}_{k-1}) / \bar{\psi}_k$ 
53:   end if
54:    $\|r_k\| = ((\bar{\pi}_{k-1} - \bar{\tau}_{k-1})^2 + (\bar{\pi}_k - \bar{\tau}_k)^2 + \bar{\chi}_{k+1}^2)^{\frac{1}{2}}$ 
55: end while

```

For $k \geq 3$, individual factorization steps may be represented as an application of $\hat{P}_{k-2,k}$ to $U_k \hat{P}_{k-1}^T$:

$$\begin{array}{c} k-2 \\ k-1 \\ k \end{array} \begin{array}{ccc} k-2 & k-1 & k \\ \left[\begin{array}{ccc} \bar{\psi}_{k-2} & & \rho_{k-2} \\ \bar{\theta}_{k-2} & \bar{\psi}_{k-1} & \phi_{k-1} \\ & & \mu_k \end{array} \right] \end{array} \begin{array}{ccc} k-2 & k-1 & k \\ \left[\begin{array}{ccc} \hat{c}_{2k-4} & & \hat{s}_{2k-4} \\ & 1 & \\ \hat{s}_{2k-4} & & -\hat{c}_{2k-4} \end{array} \right] \end{array} = \begin{array}{ccc} k-2 & k-1 & k \\ \left[\begin{array}{ccc} \psi_{k-2} & & \\ \theta_{k-2} & \bar{\psi}_{k-1} & \delta_k \\ \omega_{k-2} & & \eta_k \end{array} \right] \end{array},$$

followed by an application of $\hat{P}_{k-1,k}$ to the result:

$$\begin{array}{c} k-2 \\ k-1 \\ k \end{array} \begin{array}{ccc} k-2 & k-1 & k \\ \left[\begin{array}{ccc} \psi_{k-2} & & \\ \theta_{k-2} & \bar{\psi}_{k-1} & \delta_k \\ \omega_{k-2} & & \eta_k \end{array} \right] \end{array} \begin{array}{ccc} k-2 & k-1 & k \\ \left[\begin{array}{ccc} 1 & & \\ & \hat{c}_{2k-3} & \hat{s}_{2k-3} \\ \hat{s}_{2k-3} & & -\hat{c}_{2k-3} \end{array} \right] \end{array} = \begin{array}{ccc} k-2 & k-1 & k \\ \left[\begin{array}{ccc} \psi_{k-2} & & \\ \theta_{k-2} & \bar{\psi}_{k-1} & \\ \omega_{k-2} & \bar{\theta}_{k-1} & \bar{\psi}_k \end{array} \right] \end{array}.$$

The reflections $\hat{P}_{k-2,k}$ and $\hat{P}_{k-1,k}$ zero ρ_{k-2} and δ_k on the superdiagonals of U_k :

$$\psi_{k-2} = \sqrt{\bar{\bar{\psi}}_{k-2}^2 + \rho_{k-2}^2}, \quad \hat{c}_{2k-4} = \bar{\bar{\psi}}_{k-2}/\psi_{k-2}, \quad \hat{s}_{2k-4} = \rho_{k-2}/\psi_{k-2}, \quad (7.26a)$$

$$\bar{\bar{\psi}}_{k-1} = \sqrt{\bar{\bar{\psi}}_{k-1}^2 + \delta_k^2}, \quad \hat{c}_{2k-3} = \bar{\bar{\psi}}_{k-1}/\bar{\bar{\psi}}_{k-1}, \quad \hat{s}_{2k-3} = \delta_k/\bar{\bar{\psi}}_{k-1}, \quad (7.26b)$$

and for $k \geq 3$ they yield the recursion

$$\theta_{k-2} = \hat{c}_{2k-4}\bar{\bar{\theta}}_{k-2} + \hat{s}_{2k-4}\phi_{k-1}, \quad (7.27a)$$

$$\delta_k = \hat{s}_{2k-4}\bar{\bar{\theta}}_{k-2} - \hat{c}_{2k-4}\phi_{k-1}, \quad (7.27b)$$

$$\omega_{k-2} = \hat{s}_{2k-4}\mu_k, \quad (7.27c)$$

$$\eta_k = -\hat{c}_{2k-4}\mu_k, \quad (7.27d)$$

$$\bar{\bar{\theta}}_{k-1} = \hat{s}_{2k-3}\eta_k, \quad (7.27e)$$

$$\bar{\bar{\psi}}_k = -\hat{c}_{2k-3}\eta_k. \quad (7.27f)$$

Assuming orthonormality of V_{k+1} , we have

$$\begin{aligned} \|r_k\| &= \|\beta_1 e_1 - T_{k+1,k} y_k\| = \left\| Q_k^T \beta_1 e_1 - \begin{bmatrix} R_k \\ 0 \end{bmatrix} y_k \right\| \\ &= \left\| \begin{bmatrix} \hat{P}_k \\ 1 \end{bmatrix} Q_k^T \beta_1 e_1 - \begin{bmatrix} \hat{P}_k R_k y_k \\ 0 \end{bmatrix} \right\| \\ &= \left\| p_{k+1} - \begin{bmatrix} t_k \\ 0 \end{bmatrix} \right\|, \end{aligned} \quad (7.28)$$

where

$$(\chi_1, \dots, \chi_k, \bar{\chi}_{k+1}) := Q_k^T \beta_1 e_1, \quad (7.29a)$$

$$p_{k+1} := (\pi_1, \dots, \pi_{k-2}, \bar{\bar{\pi}}_{k-1}, \bar{\pi}_k, \bar{\chi}_{k+1}) = \begin{bmatrix} \hat{P}_k \\ 1 \end{bmatrix} Q_k^T \beta_1 e_1, \quad (7.29b)$$

$$t_k := (\tau_1, \dots, \tau_{k-2}, \bar{\bar{\tau}}_{k-1}, \bar{\tau}_k) \quad \text{solves} \quad \hat{L}_k t_k = z_k. \quad (7.29c)$$

The components of $Q_k^T \beta_1 e_1$ can be updated with the relations

$$\bar{\chi}_1 = \beta_1, \quad \chi_k = c_k \bar{\chi}_k, \quad \bar{\chi}_{k+1} = s_k \bar{\chi}_k, \quad (7.30)$$

the components of p_{k+1} are updated with

$$\bar{\pi}_1 = \chi_1, \quad (7.31a)$$

$$v_2 = \chi_2, \quad (7.31b)$$

$$\pi_{k-2} = \hat{c}_{2k-4} \bar{\pi}_{k-2} + \hat{s}_{2k-4} \chi_k, \quad k \geq 3, \quad (7.31c)$$

$$v_k = \hat{s}_{2k-4} \bar{\pi}_{k-2} - \hat{c}_{2k-4} \chi_k, \quad k \geq 3, \quad (7.31d)$$

$$\bar{\pi}_{k-1} = \hat{c}_{2k-3} \bar{\pi}_{k-1} + \hat{s}_{2k-3} v_k, \quad k \geq 2, \quad (7.31e)$$

$$\bar{\pi}_k = \hat{s}_{2k-3} \bar{\pi}_{k-1} - \hat{c}_{2k-3} v_k, \quad k \geq 2, \quad (7.31f)$$

and with $\omega_{-1} = \omega_0 = \theta_0 = \bar{\theta}_0 = 0$ the components of t_k are updated with

$$\xi_k = \zeta_k - \omega_{k-2} \tau_{k-2}, \quad (7.32a)$$

$$\bar{\tau}_k = (\xi_k - \bar{\theta}_{k-1} \bar{\tau}_{k-1}) / \bar{\psi}_k, \quad (7.32b)$$

$$\bar{\tau}_k = (\xi_k - \theta_{k-1} \tau_{k-1}) / \bar{\psi}_k, \quad (7.32c)$$

$$\tau_k = \bar{\tau}_k \bar{\psi}_k / \psi_k. \quad (7.32d)$$

Using [Lemma 1](#) we can estimate $\|r_k\|$ from the last three elements of p_{k+1} and the last two of t_k :

$$\|r_1\| = \sqrt{(\bar{\pi}_1^2 - \bar{\tau}_1^2) + \bar{\chi}_2^2}, \quad (7.33a)$$

$$\|r_k\| = \sqrt{(\bar{\pi}_{k-1} - \bar{\tau}_{k-1})^2 + (\bar{\pi}_k - \bar{\tau}_k)^2 + \bar{\chi}_{k+1}^2}, \quad k \geq 2. \quad (7.33b)$$

Lemma 1. In [\(7.28\)](#), $\pi_i = \tau_i$ for $i = 1, \dots, k-2$.

Proof. Let L_{k-2} be the leading $(k-2) \times (k-2)$ submatrix of $\hat{L}_k$, and $J_{m,n}$ be the first m rows of I_n . Then

$$\begin{aligned} L_{k-2} J_{k-2,k+1} p_{k+1} &= J_{k-2,k} \hat{L}_k J_{k,k+1} \begin{bmatrix} \hat{P}_k & 0 \\ 0 & 1 \end{bmatrix} Q_k^T \beta_1 e_1 \\ &= J_{k-2,k} U_k J_{k,k+1} Q_k^T \beta_1 e_1 \\ &= J_{k-2,k+2} \tilde{Q}_k^T N_k J_{k,k+1} Q_k^T \beta_1 e_1 \\ &= J_{k-2,k+2} \tilde{Q}_k^T T_{k+2,k+1} Q_k J_{k,k+1}^T J_{k,k+1} Q_k^T \beta_1 e_1 \\ &= J_{k-2,k+2} \tilde{Q}_k^T T_{k+2,k+1} Q_k (I_{k+1} - e_{k+1} e_{k+1}^T) Q_k^T \beta_1 e_1 \\ &= J_{k-2,k+2} \tilde{Q}_k^T (\beta_1 \alpha_1 e_1 + \beta_1 \beta_2 - \bar{\chi}_{k+1} T_{k+2,k+1} Q_k e_{k+1}) \\ &= J_{k-2,k+2} (\bar{z}_k - \bar{\chi}_{k+1} \tilde{Q}_k^T T_{k+2,k+1} Q_k e_{k+1}). \end{aligned}$$

We now have $T_{k+2,k+1}Q_k e_{k+1} = -(\alpha_{k+1}c_k + \beta_{k+1}c_{k-1}s_k)e_{k+1} - c_k\beta_{k+2}e_{k+2}$. Further, from the structure of the reflections composing $\tilde{Q}_k^T$, the first $k-2$ elements of $\tilde{Q}_k^T T_{k+2,k+1}Q_k e_{k+1}$ are zero. Thus,

$$L_{k-2}(\pi_1, \dots, \pi_{k-2}) = z_{k-2}.$$

Because L_{k-2} is always nonsingular,

$$L_{k-2} \begin{bmatrix} \pi_1 - \tau_1 \\ \vdots \\ \pi_{k-2} - \tau_{k-2} \end{bmatrix} = 0 \quad \implies \quad \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_{k-2} \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_{k-2} \end{bmatrix}.$$

□

7.5.2 Estimating $\|Ar_k\|$

From (7.21) we have

$$\|Ar_k\| = \sqrt{\bar{\zeta}_{k+1}^2 + \bar{\zeta}_{k+2}^2}, \quad k \leq \ell - 2, \quad \|Ar_{\ell-1}\| = |\zeta_\ell|. \quad (7.34)$$

7.6 CAR

We now introduce CAR, a conjugate direction method in the vein of CG and CR of Hestenes and Stiefel [48, 89] for solving $Ax = b$ when A is SPD. By design, CAR is equivalent to MINARES in exact arithmetic as both methods minimize the same quantities in the same subspace, and generate the same iterates. The name CAR stems from the property that successive A-residuals are conjugate with respect to A . The three methods generate sequences of approximate solutions x_k in the Krylov subspaces $\mathcal{K}_k(A, b)$ by minimizing a quadratic function $f(x)$:

$$\begin{aligned} f_{\text{CG}}(x) &= \tfrac{1}{2}x^T Ax - b^T x, & \nabla f_{\text{CG}}(x) &= -r, & \nabla^2 f_{\text{CG}}(x) &= A, \\ f_{\text{CR}}(x) &= \tfrac{1}{2}\|Ax - b\|^2, & \nabla f_{\text{CR}}(x) &= -Ar, & \nabla^2 f_{\text{CR}}(x) &= A^2, \\ f_{\text{CAR}}(x) &= \tfrac{1}{2}\|A^2x - Ab\|^2, & \nabla f_{\text{CAR}}(x) &= -A^3r, & \nabla^2 f_{\text{CAR}}(x) &= A^4. \end{aligned}$$

Note that all three quadratic functions satisfy $A\nabla f(x) = -\nabla^2 f(x)r$, where $r = b - Ax$. Because CAR minimizes $\|Ar_k\|$ in $\mathcal{K}_k(A, b)$, it is an alternative version of MINARES restricted to SPD A . We can derive it as a descent method with exact linesearch. From initial vectors $x_0 = 0$ and $r_0 = p_0 = b$, we update the iterates with $x_{k+1} = x_k + \alpha_k p_k$. From the Taylor

expansion, we can determine α_k that minimizes $f(x_k + \alpha p_k)$:

$$f(x_k + \alpha p_k) = f(x_k) + \alpha \nabla f(x_k)^T p_k + \frac{1}{2} \alpha^2 p_k^T \nabla^2 f(x_k) p_k, \quad \alpha_k = -\frac{\nabla f(x_k)^T p_k}{p_k^T \nabla^2 f(x_k) p_k}.$$

Afterwards we update the residuals with $r_{k+1} = r_k - \alpha_k A p_k$ and the directions with $p_{k+1} = r_{k+1} - \sum_{j=0}^k \gamma_{k+1,j} p_j$ such that $\text{Span}\{p_0, \dots, p_{k+1}\}$ forms a basis of $\mathcal{K}_{k+2}(A, b)$. We could apply a Gram–Schmidt process to orthogonalize p_{k+1} against all previous directions, but a more relevant approach is to H -conjugate them to derive a shorter recurrence, where $H = \nabla^2 f(x)$ is constant. H -conjugacy also ensures that the vectors are linearly independent. For $i = 0, \dots, k$, $p_i^T H p_{k+1} = 0$ implies $\gamma_{k+1,i} = p_i^T H r_{k+1} / p_i^T H p_i$. Let $\mathcal{P}_k := \text{Span}\{p_0, \dots, p_k\} = \text{Span}\{r_0, \dots, r_k\}$. The exact linesearch property yields $\nabla f(x_{k+1})^T p_k$ but also $\nabla f(x_{k+1}) \perp \mathcal{P}_k$ — see, e.g., [67, proof of Theorem 5.2]. Because $A p_i = (r_i - r_{i+1}) / \alpha_i \in \text{Span}\{r_i, r_{i+1}\} \subset \mathcal{P}_k$ for $i = 0, \dots, k-1$, we have $p_i^T A \nabla f(x_{k+1}) = -p_i^T \nabla^2 f(x_{k+1}) r_{k+1} = -p_i^T H r_{k+1} = 0$ and $\gamma_{k+1,i} = 0$. With $\beta_k = -\gamma_{k+1,k} = -p_k^T H r_{k+1} / p_k^T H p_k$, we obtain $p_{k+1} = r_{k+1} + \beta_k p_k$.

Theorem 9. *For CG, CR and CAR, we have:*

$$\alpha_k = \frac{\rho_k}{p_k^T H p_k} \quad \text{and} \quad \beta_k = \frac{\rho_{k+1}}{\rho_k} \quad \text{with} \quad \rho_k = -\nabla f(x_k)^T r_k.$$

Proof. Let $\rho_k = -\nabla f(x_k)^T r_k$. Because $p_k = r_k + \beta_{k-1} p_{k-1}$ and $\nabla f(x_k) \perp p_{k-1}$ (exact linesearch property), $\nabla f(x_k)^T p_k = \nabla f(x_k)^T r_k$. Therefore,

$$\alpha_k = -\frac{\nabla f(x_k)^T p_k}{p_k^T H p_k} = -\frac{\nabla f(x_k)^T r_k}{p_k^T H p_k} = \frac{\rho_k}{p_k^T H p_k}.$$

Because the directions p_i are H -conjugate, $p_k^T H p_k = p_k^T H (r_k + \beta_{k-1} p_{k-1}) = p_k^T H r_k$. With the relations $H r_i = -A \nabla f(x_i)$ and $A p_k = (r_k - r_{k+1}) / \alpha_k$, we have:

$$\beta_k = -\frac{p_k^T H r_{k+1}}{p_k^T H p_k} = -\frac{p_k^T H r_{k+1}}{p_k^T H r_k} = -\frac{\nabla f(x_{k+1})^T (r_k - r_{k+1})}{\nabla f(x_k)^T (r_k - r_{k+1})} = \frac{\nabla f(x_{k+1})^T r_{k+1}}{\nabla f(x_k)^T r_k} = \frac{\rho_{k+1}}{\rho_k},$$

where we used the fact that $\nabla f(x_{k+1})^T r_k = -r_{k+1}^T A^i r_k = \nabla f(x_k)^T r_{k+1} = 0$, ($i = 0$ for CG, $i = 1$ for CR and $i = 3$ for CAR). $\square$

CG, CR and CAR require A to be SPD because we then have $\alpha_k > 0$ until $r_k = 0$. The formulations of CG (Algorithm 17), CR (Algorithm 18) and CAR (Algorithm 19) compare the methods and suggest efficient implementations. The vectors $s_k = A r_k$, $q_k = A p_k$, $t_k = A s_k = A^2 r_k$ and $u_k = A q_k = A^2 p_k$ ultimately involve just one matrix-vector product with A per iteration. Properties of CAR are summarized in Theorem 10. By virtue of its equivalence to

MINARES in exact arithmetic, CAR allows us to establish monotonicity of relevant quantities for MINARES ([Theorem 11](#)) on SPD systems. The proofs are strongly inspired by those in [\[29, 57\]](#) for similar properties of CR and MINRES.

Algorithm 17 CG	Algorithm 18 CR	Algorithm 19 CAR
Require: $A, b, \epsilon > 0$ $k = 0, x_0 = 0$ $r_0 = b, p_0 = r_0$ $q_0 = Ap_0$ $\rho_0 = r_0^T r_0$ while $\ r_k\ > \epsilon$ do $\alpha_k = \rho_k / p_k^T q_k$ $x_{k+1} = x_k + \alpha_k p_k$ $r_{k+1} = r_k - \alpha_k q_k$ $\rho_{k+1} = r_{k+1}^T r_{k+1}$ $\beta_k = \rho_{k+1} / \rho_k$ $p_{k+1} = r_{k+1} + \beta_k p_k$ $q_{k+1} = Ap_{k+1}$ $k \leftarrow k + 1$ end while	Require: $A, b, \epsilon > 0$ $k = 0, x_0 = 0$ $r_0 = b, p_0 = r_0$ $s_0 = Ar_0, q_0 = s_0$ $\rho_0 = r_0^T s_0$ while $\ r_k\ > \epsilon$ do $\alpha_k = \rho_k / \ q_k\ ^2$ $x_{k+1} = x_k + \alpha_k p_k$ $r_{k+1} = r_k - \alpha_k q_k$ $s_{k+1} = Ar_{k+1}$ $\rho_{k+1} = r_{k+1}^T s_{k+1}$ $\beta_k = \rho_{k+1} / \rho_k$ $p_{k+1} = r_{k+1} + \beta_k p_k$ $q_{k+1} = s_{k+1} + \beta_k q_k$ $k \leftarrow k + 1$ end while	Require: $A, b, \epsilon > 0$ $k = 0, x_0 = 0$ $r_0 = b, p_0 = r_0$ $s_0 = Ar_0, q_0 = s_0$ $t_0 = As_0, u_0 = t_0$ $\rho_0 = s_0^T t_0$ while $\ r_k\ > \epsilon$ do $\alpha_k = \rho_k / \ u_k\ ^2$ $x_{k+1} = x_k + \alpha_k p_k$ $r_{k+1} = r_k - \alpha_k q_k$ $s_{k+1} = s_k - \alpha_k u_k$ $t_{k+1} = As_{k+1}$ $\rho_{k+1} = s_{k+1}^T t_{k+1}$ $\beta_k = \rho_{k+1} / \rho_k$ $p_{k+1} = r_{k+1} + \beta_k p_k$ $q_{k+1} = s_{k+1} + \beta_k q_k$ $u_{k+1} = t_{k+1} + \beta_k u_k$ $k \leftarrow k + 1$ end while

Lemma 2. *Let A be SPD. The following properties hold for CAR and MINARES for all $k \geq 0$:*

- (a) $\zeta_{k+1} d_{k+1} = \alpha_k p_k$
- (b) $s_k = Ar_k$
- (c) $q_k = Ap_k$
- (d) $t_k = As_k$
- (e) $u_k = Aq_k$.

Proof. (a) follows by direct comparison of [Algorithm 16](#) and [Algorithm 19](#).

(b)–(e) all hold by construction at $k = 0$. By induction, assume that they also hold at index $k \geq 0$. Then, $s_{k+1} = s_k - \alpha_k u_k = Ar_k - \alpha_k Aq_k = Ar_{k+1}$, which establishes (b). The remaining properties follow similarly. $\square$

Theorem 10. Let A be SPD. For $(i, j) \in \{0, \dots, n-1\}^2$, the following properties hold for CAR:

- (a) $p_i^T A^4 p_j = 0$ ($i \neq j$)
- (b) $r_i^T A^3 p_j = 0$ ($i > j$)
- (c) $r_i^T A^3 r_j = 0$ ($i \neq j$)
- (d) $\alpha_i \geq 0$
- (e) $\beta_i \geq 0$
- (f) $q_i^T u_j = p_i^T A^3 p_j \geq 0$
- (g) $q_i^T q_j = p_i^T A^2 p_j \geq 0$
- (h) $q_i^T p_j = p_i^T A p_j \geq 0$
- (i) $p_i^T p_j \geq 0$
- (j) $x_i^T p_j \geq 0$
- (k) $r_i^T q_j = r_i^T A p_j \geq 0$.

Proof. Because $\nabla^2 f_{\text{CAR}}(x) = A^4$, we A^4 -conjugate the vectors p_i by construction and (a) is satisfied.

Because $\nabla f_{\text{CAR}}(x_i) = -A^3 r_i$, the exact linesearch property yields (b) as in [67, proof of Theorem 5.2].

If $i > j$, $r_i^T A^3 r_j = r_i^T A^3 (p_j - \beta_{j-1} p_{j-1}) = 0$ by (b). If $i < j$, $r_i^T A^3 r_j = (p_i - \beta_{i-1} p_{i-1})^T A^3 r_j = 0$, again thanks to (b), which proves (c).

First note that $\rho_i = s_i^T t_i = r_i^T A^3 r_i \geq 0$ because A is SPD. Thus $\alpha_i = \rho_i / \|u_i\|^2 \geq 0$ and $\beta_i = \rho_{i+1} / \rho_i \geq 0$, which proves (d) and (e).

We now establish (f) by induction. If $i = j$, $q_i^T u_i = q_i^T A q_i \geq 0$ because A is SPD. Assuming $q_i^T u_j \geq 0$ when $|i - j| = k - 1 \geq 0$, we want to show the result for $|i - j| = k$. If $i - j = k > 0$ then $q_i^T u_j = q_i^T u_{i-k}$. Otherwise we have $j - i = k > 0$ and $q_i^T u_j = q_i^T u_{i+k}$. Lemma 2 yields

$$\begin{aligned}
 q_i^T u_{i-k} &= (s_i + \beta_{i-1} q_{i-1})^T u_{i-k} & q_i^T u_{i+k} &= q_i^T A q_{i+k} \\
 &= s_i^T u_{i-k} + \beta_{i-1} q_{i-1}^T u_{i-k} & &= q_i^T A (s_{i+k} + \beta_{i+k-1} q_{i+k-1}) \\
 &= r_i^T A^3 p_{i-k} + \beta_{i-1} q_{i-1}^T u_{i-k} & &= p_i^T A^3 r_{i+k} + \beta_{i+k-1} u_i^T q_{i+k-1} \\
 &= \beta_{i-1} q_{i-1}^T u_{i-k} & &= \beta_{i+k-1} q_{i+k-1}^T u_i
 \end{aligned}$$

$\beta_{i-1} \geq 0$ and $\beta_{i+k-1} \geq 0$ by (e). $q_{i-1}^T u_{i-k} \geq 0$ and $q_{i+k-1}^T u_i \geq 0$ by induction assumption. Thus, $q_i^T u_j \geq 0$ for $|i-j| = k$, which completes the proof of (f).

At termination, define $\mathcal{P} = \text{Span}\{p_0, \dots, p_{\ell-1}\}$, $\mathcal{Q} = \text{Span}\{q_0, \dots, q_{\ell-1}\} = A\mathcal{P}$ and $\mathcal{U} = \text{Span}\{u_0, \dots, u_{\ell-1}\} = A\mathcal{Q}$. By construction, $\mathcal{P} = \text{Span}\{b, \dots, A^{\ell-1}b\}$, $\mathcal{Q} = \text{Span}\{Ab, \dots, A^{\ell}b\}$ and $\mathcal{U} = \text{Span}\{A^2b, \dots, A^{\ell+1}b\}$. Again by construction, $x_{\ell} \in \mathcal{P}$, and since $r_{\ell} = 0$, we have $Ax_{\ell} = b \in \mathcal{Q}$ and $A^2x_{\ell} = Ab \in \mathcal{U}$. We see that $\mathcal{P} \subset \mathcal{Q} \subset \mathcal{U}$.

(a) and Lemma 2 (c)–(e) imply that $u_i^T u_j = 0$ for $i \neq j$, and therefore, $\{u_k / \|u_k\|\}_{k=0, \dots, \ell-1}$ forms an orthonormal basis for $\mathcal{U}$. Thus, if we project p_i and q_i into $\mathcal{U}$, we have

$$p_i = \sum_{k=0}^{\ell-1} \frac{p_i^T u_k}{u_k^T u_k} u_k \quad \text{and} \quad q_i = \sum_{k=0}^{\ell-1} \frac{q_i^T u_k}{u_k^T u_k} u_k.$$

Scalar products between these vectors can be expressed as

$$q_i^T q_j = \sum_{k=0}^{\ell-1} \frac{(q_i^T u_k)(q_j^T u_k)}{\|u_k\|^2}, \quad p_i^T q_j = \sum_{k=0}^{\ell-1} \frac{(p_i^T u_k)(q_j^T u_k)}{\|u_k\|^2} \quad \text{and} \quad p_i^T p_j = \sum_{k=0}^{\ell-1} \frac{(p_i^T u_k)(p_j^T u_k)}{\|u_k\|^2}.$$

Thus $q_i^T q_j \geq 0$ by (f), proving (g). Because $p_i^T u_k = p_i^T A q_k = q_i^T q_k$, $p_i^T q_j \geq 0$ and $p_i^T p_j \geq 0$ by (f) and (g), which proves (h) and (i).

By construction, $x_i = \sum_{k=0}^i \alpha_k p_k$ and so $x_i^T p_j \geq 0$ by (d) and (i), proving (j).

Finally, $r_i^T q_j = \sum_{k=i}^{\ell-1} \alpha_k q_k^T q_j \geq 0$ by (d) and (g), proving (k). $\square$

Theorem 11. *For CAR (and hence MINARES) applied to $Ax = b$ when A is SPD, the following properties are satisfied:*

- $\|x_k\|$ increases monotonically
- $\|x^* - x_k\|$ decreases monotonically
- $\|x^* - x_k\|_A$ decreases monotonically
- $\|r_k\|$ decreases monotonically.

Proof. From Theorem 10 (d) and (j),

$$\begin{aligned} \|x_k\|^2 - \|x_{k-1}\|^2 &= (x_{k-1} + \alpha_k p_k)^T (x_{k-1} + \alpha_k p_k) - x_{k-1}^T x_{k-1} \\ &= 2\alpha_k p_k^T x_{k-1} + \alpha_k^2 \|p_k\|^2 \geq 0. \end{aligned}$$

From [Theorem 10 \(d\)](#) and [\(i\)](#),

$$\begin{aligned}\|x^\star - x_{k-1}\|^2 - \|x^\star - x_k\|^2 &= \left(\sum_{i=k}^{\ell-1} \alpha_i p_i\right)^T \left(\sum_{i=k}^{\ell-1} \alpha_i p_i\right) - \left(\sum_{i=k+1}^{\ell-1} \alpha_i p_i\right)^T \left(\sum_{i=k+1}^{\ell-1} \alpha_i p_i\right) \\ &= 2\alpha_k p_k^T \left(\sum_{i=k+1}^{\ell-1} \alpha_i p_i\right) + \alpha_k^2 \|p_k\|^2 \geq 0.\end{aligned}$$

From [Theorem 10 \(d\)](#) and [\(h\)](#),

$$\begin{aligned}\|x^\star - x_{k-1}\|_A^2 - \|x^\star - x_k\|_A^2 &= \left(\sum_{i=k}^{\ell-1} \alpha_i p_i\right)^T A \left(\sum_{i=k}^{\ell-1} \alpha_i p_i\right) - \left(\sum_{i=k+1}^{\ell-1} \alpha_i p_i\right)^T A \left(\sum_{i=k+1}^{\ell-1} \alpha_i p_i\right) \\ &= 2\alpha_k q_k^T \left(\sum_{i=k+1}^{\ell-1} \alpha_i p_i\right) + \alpha_k^2 q_k^T p_k \geq 0.\end{aligned}$$

From [Theorem 10 \(d\)](#) and [\(k\)](#),

$$\begin{aligned}\|r_{k-1}\|^2 - \|r_k\|^2 &= r_{k-1}^T r_{k-1} - r_k^T r_k \\ &= (r_k + \alpha_{k-1} q_{k-1})^T (r_k + \alpha_{k-1} q_{k-1}) - r_k^T r_k \\ &= 2\alpha_{k-1} q_{k-1}^T r_k + \alpha_{k-1}^2 \|q_{k-1}\|^2 \geq 0.\end{aligned}$$

□

7.7 Implementation and numerical experiments

We implemented [Algorithm 16](#) and [Algorithm 19](#) in Julia [5], version 1.9, as part of our `Krylov.jl` collection of Krylov methods [64]. These implementations of MINARES and CAR are applicable in any floating-point system supported by Julia, including complex numbers, and they run on CPU and GPU. They also support preconditioners.

We evaluate the performance of MINARES on systems generated from symmetric matrices A in the SuiteSparse Matrix Collection [15]. In each case we first scale A to be A/α with $\alpha = \max |A_{ij}|$, so that $\|A\| \approx 1$.

In our first set of experiments, we compare MINARES to our Julia implementation of MINRES-QLP in terms of number of iterations on consistent systems when the stopping criterion is $\|r_k\| \leq 10^{-10}$, then when it is $\|Ar_k\| \leq 10^{-10}$. The right-hand side $b = Ae$ (with e a vector of ones) ensures that the system is consistent even if A is singular. The residual and A-residual are calculated explicitly at each iteration in order to evaluate $\|r_k\|$ and $\|Ar_k\|$.

(To get a fair comparison, (7.33) and (7.34) are not used.) Figure 7.1 reports residual and A-residual histories for MINARES and MINRES-QLP on problems *rail_5177* and *bcsstm36*. We observe that MINRES-QLP's $\|Ar_k\|$ is erratic, whereas MINARES's $\|Ar_k\|$ and $\|r_k\|$ are both smooth. Also, MINRES-QLP's $\|Ar_k\|$ lags further behind MINARES's than MINARES's $\|r_k\|$ does behind MINRES-QLP's. When the system is consistent, we have similar behavior whether A is singular or not.

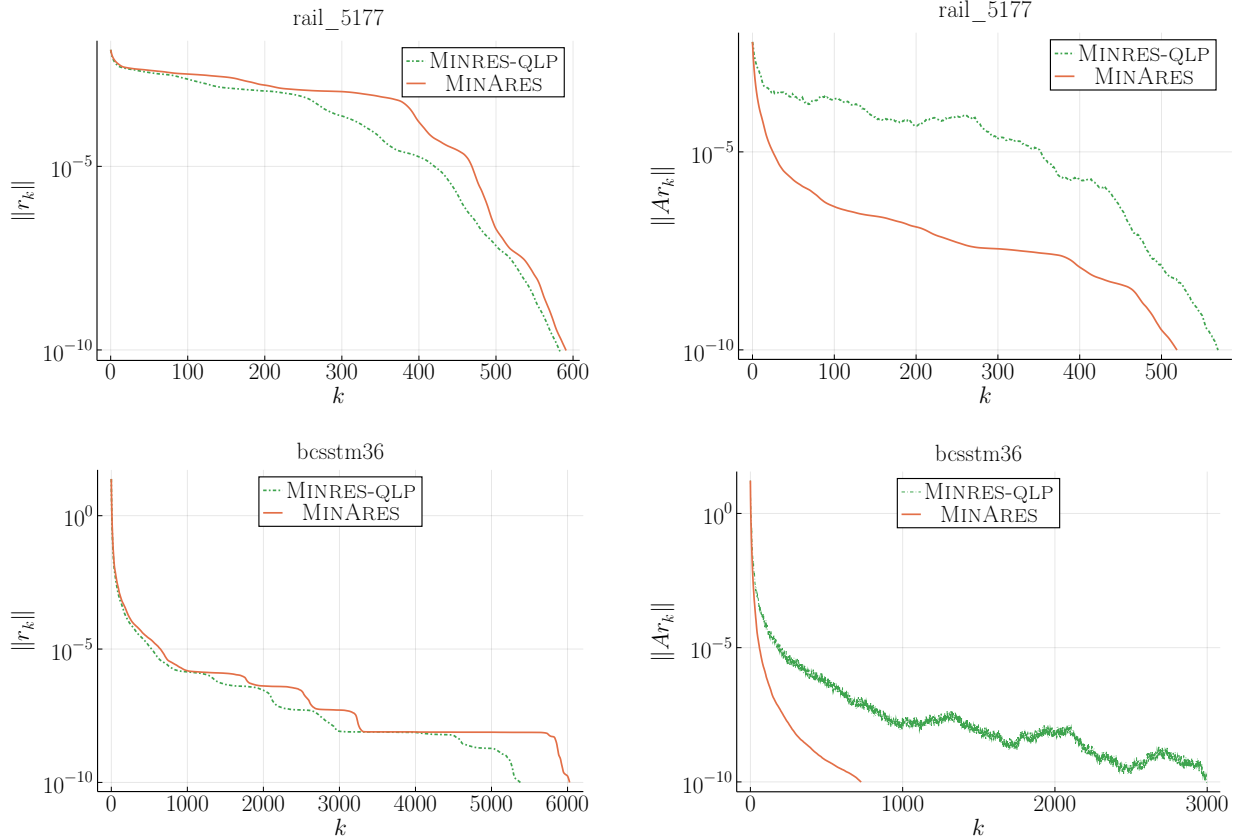


Figure 7.1 Residual and A-residual histories for MINARES and MINRES-QLP on consistent systems generated from the SuiteSparse Matrix Collection. Top: System based on the nonsingular matrix *rail_5177* ($n = 5177$). Bottom: System based on the singular matrix *bcsstm36* ($n = 23052$).

In a second set of experiments, we compare MINARES to our Julia implementations of MINRES-QLP and LSMR in terms of number of matrix-vector products Av on singular inconsistent systems with $b = e$ when the stopping criterion is $\|Ar_k\| \leq 10^{-6}$ for the problem *zenios* and $\|Ar_k\| \leq 10^{-10}$ for *laser*. Figure 7.2 shows that MINRES-QLP has difficulty reaching the specified $\|Ar_k\|$, but MINARES performs well and converges much faster than LSMR, the only other Krylov method that minimizes $\|Ar_k\|$.

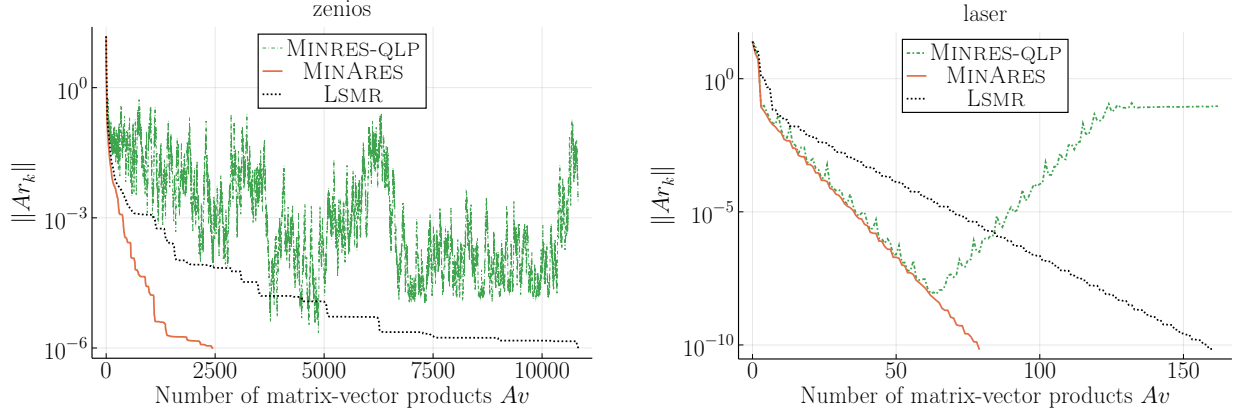


Figure 7.2 A-residual history for MINARES, MINRES-QLP and LSMR on singular inconsistent systems generated from the SuiteSparse Matrix Collection. Left: System based on the singular matrix zenios ($n = 2873$). Right: System based on the singular matrix laser ($n = 3002$).

7.8 Summary

MINARES completes the family of Krylov methods based on the symmetric Lanczos process. By minimizing $\|Ar_k\|$ (which always converges to zero), MINARES can be applied safely to any symmetric system. For SPD systems, CAR is equivalent to MINARES and extends the conjugate directions family CG and CR. For such systems we prove that $\|r_k\|$, $\|x_k - x^*\|$ and $\|x_k - x^*\|_A$ decrease monotonically for CAR and hence MINARES.

On consistent symmetric systems, MINARES is a relevant alternative to MINRES and MINRES-QLP because it converges in a similar number of iterations if the stopping condition is based on $\|r_k\|$, and much faster if the stopping condition is based on $\|Ar_k\|$. On singular inconsistent symmetric systems, MINARES outperforms MINRES-QLP and LSMR, and should be the preferred method.

CHAPITRE 8 ARTICLE 5: KRYLOV.JL: A JULIA BASKET OF HAND-PICKED KRYLOV METHODS

ALEXIS MONTOISON AND DOMINIQUE ORBAN

Manuscript published on September 26, 2023, in the *Journal of Open Source Software*
(JOSS)

Abstract

This paper presents `Krylov.jl`, a Julia package that implements a collection of Krylov processes and methods for solving a variety of linear problems.

8.1 Summary

`Krylov.jl` is a Julia [5] package that implements a collection of Krylov processes and methods for solving a variety of linear problems:

Square systems	Linear least-squares problems	Linear least-norm problems
$Ax = b$	$\min \ b - Ax\ $	$\min \ x\ $ subject to $Ax = b$

Adjoint systems	Saddle-point and Hermitian quasi-definite systems	Generalized saddle-point and non-Hermitian partitioned systems
$Ax = b$ $A^H y = c$	$\begin{bmatrix} M & A \\ A^H & -N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$	$\begin{bmatrix} M & A \\ B & N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$

A^H denotes the conjugate transpose of A . It coincides with A^T , the transpose of A , if A is real. Krylov methods are iterative methods based on [53] subspaces. They are an alternative to direct methods such as Gaussian elimination or QR decomposition when storage requirements or computational costs become prohibitive, which is often the case for large and sparse linear problems. Contrary to direct methods, which require storing A explicitly, Krylov methods support linear operators to model operator-vector products $u \leftarrow Av$, and in some instances $u \leftarrow A^H w$ because Krylov processes only require those operations to build Krylov subspaces. The same goes with preconditioners, i.e., transformations that modify a linear system into an equivalent form with favorable spectral properties that may yield faster convergence in finite-precision arithmetic. We refer interested readers to [50] for an introduction to Krylov methods along with [38] and [82] for more details.

8.2 Statement of need

8.2.1 Largest collection of Krylov processes and methods

Krylov.jl aims to provide a user-friendly and unified interface for the largest collection of Krylov processes and methods, all programming languages taken together, with six and thirty-five implementations, respectively:

- **Krylov processes:** ARNOLDI, GOLUB-KAHAN, HERMITIAN LANCZOS, MONTISON-ORBAN, NON-HERMITIAN LANCZOS, SAUNDERS-SIMON-YIP;
- **Krylov methods:** BICGSTAB, BILQ, BILQR, CAR, CG, CG-LANCZOS, CG-LANCZOS-SHIFT, CGLS, CGNE, CGS, CR, CRAIG, CRAIGMR, CRLS, CRMR, DIOM, DQGMRES, FGMRES, FOM, GMRES, GPMR, LNLQ, LSLQ, LSMR, LSQR, MINARES, MINRES, MINRES-QLP, QMR, SYMMLQ, TRICG, TRILQR, TRIMR, USYMLQ, USYMR.

Hence Krylov.jl is a suitable toolbox for easily comparing existing methods with each other as well as new ones. The number of distinct Krylov methods is twenty-two for PETSc [3], eleven for [58] and [KrylovMethods.jl](#), nine for [IterativeSolvers.jl](#) and three for [KrylovKit.jl](#). However Krylov.jl does not have implementations of recycling Krylov methods nor block Krylov methods, unlike some alternatives, except for special cases, including TRICG, TRIMR, and GPMR. Note that we only consider the number of Krylov methods that generate different iterates without preconditioning. Variants with preconditioning are not counted except flexible ones such as FGMRES.

Some processes and methods are not available elsewhere and are the product of our own research. References for each process and method are available in the extensive [documentation](#). Beyond the number of methods, Krylov.jl is the only package that offers all of the features that we describe below.

8.2.2 Support for any floating-point system supported by Julia

Krylov.jl works with real and complex data in any floating-point system supported by Julia, which means that Krylov.jl handles any precision T and `Complex{T}` where $T <: \text{AbstractFloat}$. Although most personal computers offer IEEE 754 single and double precision computations, new architectures implement native computations in other floating-point systems. In addition, software libraries such as the GNU MPFR, shipped with Julia, let users experiment with computations in variable, extended precision at the software level with the `BigFloat` data type. Working in high precision has obvious benefits in terms of accuracy.

8.2.3 Support for Nvidia, AMD and Intel GPUs

Krylov methods are well suited for GPU computations because they only require operator-vector products ($u \leftarrow Av$, $u \leftarrow A^H w$) and vector operations ($\|v\|$, $u^H v$, $v \leftarrow \alpha u + \beta v$), which are highly parallelizable. The implementations in Krylov.jl are generic so as to take advantage of the multiple dispatch and broadcast features of Julia. Those allow the implementations to be specialized automatically by the compiler for both CPU and GPU. Thus, Krylov.jl works with GPU backends that build on [GPUArrays.jl](#), including [CUDA.jl](#), [AMDGPU.jl](#), and [oneAPI.jl](#), the Julia interfaces to NVIDIA, AMD, and Intel GPUs.

8.2.4 Support for linear operators

The input arguments of all Krylov.jl solvers that model A , B , M , N and preconditioners can be any object that represents a linear operator. Krylov methods combined with linear operators allow to reduce computation time and memory requirements considerably by avoiding building and storing matrices. In nonlinear optimization, finding a critical point of a continuous function frequently involves linear systems where A is a Hessian or a Jacobian. Materializing such operators as matrices is expensive in terms of operations and memory consumption and is unreasonable for high-dimensional problems. However, it is often possible to implement efficient Hessian-vector and Jacobian-vector products, for example with the help of automatic differentiation tools.

8.2.5 In-place methods

All solvers in Krylov.jl have an in-place variant that allows to solve multiple linear systems with the same dimensions, precision and architecture. Optimization methods such as the Newton and Gauss-Newton methods can take advantage of this functionality by allocating workspace for the solve only once. The in-place variants only require a Julia structure that contains all the storage needed by a Krylov method as additional argument. In-place methods limit memory allocations and deallocations, which are particularly expensive on GPUs.

8.2.6 Performance optimizations and storage requirements

Operator-vector products and vector operations are the most expensive operations in Krylov.jl. The vectors in Krylov.jl are always dense. One may then expect that taking advantage of an optimized BLAS library when one is available on CPU and when the problem data is stored in a supported representation should improve performance. Thus, we dispatch vector-vector operations to BLAS1 routines, and operator-vector operations to BLAS2 routines

when the operator is a dense matrix. By default, Julia ships with OpenBLAS and provides multithreaded routines. Since Julia 1.6, users can also switch dynamically to other BLAS backends, such as the Intel MKL, BLIS or Apple Accelerate, thanks to the BLAS demuxing library `libblastrampoline`, if an optimized BLAS is available.

A “Storage Requirements” section is available in the documentation to provide the theoretical number of bytes required by each method. Our implementations are storage-optimal in the sense that they are guaranteed to match the theoretical storage amount. The match is verified in the unit tests by way of functions that return the number of bytes allocated by our implementations.

8.2.7 Examples

Our first example is a simple implementation of the Gauss-Newton method without line-search for nonlinear least squares. It illustrates several of the facilities of `Krylov.jl`: solver preallocation and reuse, genericity with respect to data types, and linear operators. Another example based on a simplistic Newton method without linesearch for convex optimization is also available in the documentation, and illustrates the same concepts in the sections “In-place methods” and “Factorization-free operators”.

```
using LinearAlgebra    # Linear algebra library of Julia
using SparseArrays     # Sparse library of Julia
using Test            # Test library of Julia
using Krylov          # Krylov methods and processes
using LinearOperators  # Linear operators
using ForwardDiff     # Automatic differentiation
using Quadmath        # Quadruple precision
using MKL             # Intel BLAS

"The Gauss-Newton method for Nonlinear Least Squares"
function gauss_newton(F, JF, x₀::AbstractVector{T}; itmax = 200, tol = √eps(T)) where T
    n = length(x₀)
    x = copy(x₀)
    Fx = F(x)
    m = length(Fx)
    iter = 0
    S = typeof(x)          # precision and architecture
    solver = LsmrSolver(m, n, S) # structure that contains the workspace of LSMR
    solved = tired = false
    while !(solved || tired)
```

```

        Jx = JF(x)                # Compute  $J(x_k)$ 
        lsqr!(solver, Jx, -Fx)    # Minimize  $\|J(x_k)\Delta x + F(x_k)\|$ 
        x .+= solver.x            # Update  $x_{k+1} = x_k + \Delta x$ 
        Fx_old = Fx               #  $F(x_k)$ 
        Fx = F(x)                 #  $F(x_{k+1})$ 
        iter += 1
        solved = norm(Fx - Fx_old) / norm(Fx) ≤ tol
        tired = iter ≥ itmax
    end
    return x
end

T = Float128 # IEEE quadruple precision
x_exact = T[8, 0.25]
x_0 = ones(T, 2)
t = T[1, 2, 3, 4, 5, 6, 7, 8]
y = [trunc(x_exact[1] * exp(x_exact[2] * t[i]), digits=3) for i=1:8]
F(x) = [x[1] * exp(x[2] * t[i]) - y[i] for i=1:8] # F(x)
J(y, x, v) = ForwardDiff.derivative!(y, h -> F(x + h * v), 0) #  $y \leftarrow JF(x)v$ 
JT(y, x, w) = ForwardDiff.gradient!(y, x -> dot(F(x), w), x) #  $y \leftarrow JF^T(x)w$ 
symmetric = hermitian = false
JF(x) = LinearOperator(T, 8, 2, symmetric, hermitian,
    (y, v) -> J(y, x, v), # non-transpose
    (y, w) -> JT(y, x, w), # transpose
    (y, w) -> JT(y, x, w)) # conjugate transpose

x = gauss_newton(F, JF, x_0)

# Check the solution returned by the Gauss-Newton method
@test norm(x - x_exact) ≤ 1e-4

```

Our second example concerns the solution of a complex Hermitian linear system from the SuiteSparse Matrix Collection [15] with an incomplete Cholesky factorization preconditioner on GPU. The preconditioner is implemented as an in-place linear operator that performs the forward and backward sweeps with the Cholesky factor of the incomplete decomposition. Because the system matrix is Hermitian and positive definite, we use the conjugate gradient method. However, other methods for Hermitian systems could be used, including SYMMLQ, CR, and MINRES.

```

using LinearAlgebra                # Linear algebra library of Julia
using SparseArrays                 # Sparse library of Julia

```

```

using Test                                # Test library of Julia
using Krylov                              # Krylov methods and processes
using LinearOperators                     # Linear operators
using MatrixMarket                       # Reader of matrices stored in the Matrix Market format
using SuiteSparseMatrixCollection        # Interface to the SuiteSparse Matrix Collection
using CUDA                               # Interface to NVIDIA GPUs
using CUDA.CUSPARSE                      # NVIDIA CUSPARSE library

if CUDA.functional()
    ssmc = ssmc_db(verbose=false)
    matrices = ssmc_matrices(ssmc, "Sinclair", "3Dspectralwave2")
    paths = fetch_ssmc(matrices, format="MM")
    path_A = joinpath(paths[1], "3Dspectralwave2.mtx")

    # A is an Hermitian and positive definite matrix of size 292008 x 292008
    A_cpu = MatrixMarket.mmread(path_A) + 50I
    m, n = size(A_cpu)
    x_exact = ones(ComplexF64, m)
    b_cpu = A_cpu * x_exact

    # Transfer the linear system from the CPU to the GPU
    A_gpu = CuSparseMatrixCSR(A_cpu)
    b_gpu = CuVector(b_cpu)

    # Incomplete Cholesky factorization  $LL^H \approx A$  with zero fill-in
    P = ic02(A_gpu)

    # Additional vector required for solving triangular systems
    z = CUDA.zeros(ComplexF64, n)

    # Solve  $P y = x$ 
    function ldiv_ic0!(P, x, y, z)
        L = LowerTriangular(P)
        LH = adjoint(L)
        ldiv!(z, L, x)    # Forward substitution with L
        ldiv!(y, LH, z)  # Backward substitution with LH
        return y
    end

    # Linear operator that approximates the preconditioner  $P^{-1}$  in floating-point arithmetic
    T = ComplexF64
    symmetric = false
    hermitian = true
    P-1 = LinearOperator(T, m, n, symmetric, hermitian, (y, x) -> ldiv_ic0!(P, x, y, z))

```

```
# Solve an Hermitian positive definite system with an incomplete Cholesky preconditioner
x_gpu, stats = cg(A_gpu, b_gpu, M=P-1)

# Check the solution returned by the conjugate gradient method
x_cpu = Vector{ComplexF64}(x_gpu)
@test norm(x_cpu - x_exact) ≤ 1e-5
end
```

CHAPITRE 9 DISCUSSION GÉNÉRALE

9.1 Synthèse des travaux

Les principales contributions de cette thèse sont le développement de huit nouvelles méthodes de Krylov, BiLQ, BiLQR, TriLQR, TriCG, TriMR, GPMR, MINARES, CAR, ainsi qu'un nouveau processus de Krylov appelé ORTHOGONAL HESSENBERG REDUCTION. Chaque méthode permet de combler un vide dans la littérature et de fournir un algorithme plus adapté à certains systèmes linéaires. La [table 9.1](#) récapitule les contributions.

TABLEAU 9.1 Synthèse des nouvelles méthodes de Krylov.

Méthodes de Krylov	Caractéristiques des systèmes linéaires
BiLQ	$Ax = b, A \neq A^H$
BiLQR, TriLQR	$Ax = b$ et $A^H y = c$
TriCG, TriMR	$\begin{bmatrix} M & A \\ A^H & -N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$
GPMR	$\begin{bmatrix} M & A \\ B & -N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$
MINARES	$Ax = b, A = A^H$
CAR	$Ax = b, A = A^H, A \succ 0$

L'ensemble des méthodes fait partie d'une bibliothèque numérique Julia Krylov.jl qui possède trente-cinq méthodes différentes ainsi que six processus de Krylov.

Il existe d'autres bibliothèques numériques contenant des méthodes de Krylov, comme MATLAB [58], Scilab [10], NumPy [45], SciPy [93], PETSc [3], Trilinos [46] ou Eigen [42]. Cependant, aucune d'entre elles n'est considérée comme la référence en matière de méthodes de Krylov. Krylov.jl pourrait devenir l'outil de référence dans ce domaine pour plusieurs raisons.

Tout d'abord, Krylov.jl fournit une interface simple et unifiée de la plus grande collection de méthodes et processus de Krylov. Krylov.jl se distingue par l'implémentation polymorphe de chaque méthode, ce qui permet le support de divers systèmes à virgule flottante, offrant ainsi une flexibilité pour la précision des calculs. De plus, Krylov.jl supporte les cartes graphiques Nvidia, AMD et Intel ainsi que l'unité graphique des processeurs Apple M pour accélérer les calculs sur ces architectures. Les méthodes de Krylov présentes dans Krylov.jl offrent

également des variantes *in-place* pour optimiser l'utilisation de la mémoire lors de la résolution successive de systèmes linéaires, ce qui est fréquemment rencontré dans les méthodes d'optimisation. Krylov.jl est optimisé pour atteindre des performances et des exigences de stockage optimales. Enfin, la documentation exhaustive de Krylov.jl, accompagnée de multiples exemples, facilite l'utilisation de Krylov.jl par tout utilisateur.

Krylov.jl est classé parmi les 200 meilleures bibliothèques Julia¹, avec un total de 284 étoiles sur GitHub en date du 13 novembre 2023, lorsque l'on utilise le nombre d'étoiles comme métrique de popularité.

9.2 Limitations de la solution proposée

Les nouvelles méthodes de Krylov montrent qu'il n'existe pas de méthode capable de résoudre de manière optimale tous les systèmes linéaires. On peut faire la même analogie avec les méthodes dédiées à la résolution de problèmes d'optimisation. On peut toujours prendre en compte des caractéristiques additionnelles au problème qu'on rencontre pour développer des méthodes plus adaptées.

Par la nature du langage de programmation dans lequel la bibliothèque numérique Krylov.jl est implémentée, son utilisation est limitée aux utilisateurs Julia. Le langage de programmation Julia peut être appelé depuis d'autres langages mais cela requiert le développement d'une interface C. Bien que la communauté Julia soit en expansion, soutenue par la syntaxe intuitive du langage et les performances d'un langage compilé, elle reste encore petite, comparé à celles des langages établis comme Python ou C++. Malgré cela, Julia a trouvé sa place dans divers domaines, comme le calcul haute performance, démontrant sa valeur même face à une concurrence bien établie.

9.3 Améliorations futures

Les méthodes de Krylov.jl sont fréquemment combinées à des préconditionneurs afin d'accélérer la convergence. L'existence d'une implémentation polymorphe des préconditionneurs améliorerait la robustesse des méthodes de Krylov dans divers contextes. La bibliothèque numérique **KrylovPreconditioners.jl** a été créée dans cette perspective, bien qu'elle en soit encore au début de son développement. Accorder de l'attention à cette bibliothèque numérique ne peut qu'encourager l'utilisation de Krylov.jl.

Les méthodes de Krylov disponible dans Krylov.jl permettent de résoudre uniquement des

1. <https://juliapackages.com/packages>

systèmes linéaires avec un vecteur comme second membre. Si le second membre est une matrice, aucune méthode n'est disponible. Le développement de méthodes par blocs permettrait de combler ce manque, mais apporte aussi de nouveaux défis en matière d'implémentation.

Enfin le développement de TRICG, TRIMR et GPMR a révélé que certaines méthodes par blocs de taille deux pouvaient permettre de résoudre plus efficacement des systèmes linéaires structurés avec un seul second membre. Bien que ceci soit contre-intuitif, des méthodes de Krylov par blocs de taille n , plus coûteuses en mémoire et en opérations, pourraient résoudre des systèmes linéaires non structurés avec un vecteur comme second membre plus rapidement sur cartes graphiques grâce à une plus grande richesse d'opérations parallélisables.

CHAPITRE 10 CONCLUSION ET RECOMMANDATIONS

Aujourd’hui, les architectures de nos ordinateurs évoluent continuellement. Prendre en compte la portabilité de nos codes devient un impératif dans un environnement aussi hétérogène pour ne pas réimplémenter ce qui a déjà été fait. Nous ne savons pas si demain la programmation FPGA (Field Programmable Gate Arrays) ou encore le calcul quantique seront la norme mais nous devons nous préparer à cette éventuelle transition.

Ma collaboration avec le laboratoire de recherche Argonne dans le cadre du projet *Exascale Computing Project* a mis en évidence cette problématique et renforcé la pertinence de Krylov.jl. La portabilité, la flexibilité de développement et les performances de Krylov.jl en font une solution appropriée pour la résolution de systèmes linéaires creux sur les nouvelles architectures des cartes graphiques AMD et Intel, équipant les deux supercalculateurs les plus puissants au monde. Ces supercalculateurs, capables d’effectuer 10^{18} opérations par seconde, à savoir *Frontier* au laboratoire national d’Oak Ridge et *Aurora* au laboratoire national d’Argonne, utilisent Krylov.jl pour résoudre les systèmes linéaires générés par un solveur d’optimisation non linéaire dédié aux problèmes d’optimisation contraints [86].

RÉFÉRENCES

- [1] A. ALTMAN et J. GONDZIO : *Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization*. *Optim. Method Softw.*, 11(1-4):275–302, 1999.
- [2] W. E. ARNOLDI : *The principle of minimized iterations in the solution of the matrix eigenvalue problem*. *Q. Appl. Math.*, 9:17–29, 1951.
- [3] S. BALAY, S. ABHYANKAR, M. F. ADAMS, S. BENSON, J. BROWN, P. BRUNE, K. BUSCHELMAN, E. CONSTANTINESCU, L. DALCIN, A. DENER, V. EIJKHOUT, J. FAIBUSSOWITSCH, W. D. GROPP, V. HAPLA, T. ISAAC, P. JOLIVET, D. KARPEEV, D. KAUSHIK, M. G. KNEPLEY, F. KONG, S. KRUGER, D. A. MAY, L. C. MCINNES, R. T. MILLS, L. MITCHELL, T. MUNSON, J. E. ROMAN, K. RUPP, P. SANAN, J. SARICH, B. F. SMITH, S. ZAMPINI, H. ZHANG, H. ZHANG et J. ZHANG : *PETSc/TAO users manual*. Rap. tech. ANL-21/39 - Revision 3.19, Argonne National Laboratory, 2023.
- [4] M. BENZI, G. H. GOLUB et J. LIESEN : *Numerical solution of saddle point problems*. *Acta Numer.*, 14:1–137, 2005.
- [5] J. BEZANSON, A. EDELMAN, S. KARPINSKI et V. B. SHAH : *Julia: A Fresh Approach to Numerical Computing*. *SIAM Rev.*, 59(1):65–98, 2017.
- [6] Å. BJÖRCK : *Numerics of Gram-Schmidt orthogonalization*. *Linear Algebra Appl.*, 197:297–316, 1994.
- [7] D. BOFFI, F. BREZZI et M. FORTIN : *Mixed Finite Element Methods and Applications*, vol. 44. Springer-Verlag Berlin Heidelberg, 2013.
- [8] D. BRAESS : *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. Cambridge University Press, 2007.
- [9] A. BUTTARI, D. ORBAN, D. RUIZ et D. TITLEY-PELOQUIN : *A tridiagonalization method for symmetric saddle-point systems*. *SIAM J. Sci. Comput.*, 41(5):409–432, 2019.
- [10] S. L. CAMPBELL, J.-P. CHANCELIER, R. NIKOUKHAH, S. L. CAMPBELL, J.-P. CHANCELIER et R. NIKOUKHAH : *Modeling and Simulation in SCILAB*. Springer, 2010.
- [11] E. CARSON, K. LUND, M. ROZLOŽNÍK et S. THOMAS : *Block Gram-Schmidt algorithms and their stability properties*. *Linear Algebra Appl.*, 638:150–195, 2022.
- [12] T. T. CHISHOLM et D. W. ZINGG : *A Jacobian-free Newton-Krylov algorithm for compressible turbulent fluid flows*. *J. Comput. Phys.*, 228:3490–3507, 2009.
- [13] S.-C. CHOI, C. C. PAIGE et M. A. SAUNDERS : *MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems*. *SIAM J. Sci. Comput.*, 33(4):1810–1836, 2011.
- [14] J. E. CRAIG : *The N-step iteration procedures*. *J. Math. Phys. Camb.*, 34(1-4):64–73, 1955.
- [15] T. DAVIS et Y. HU : *The University of Florida sparse matrix collection*. *ACM Trans. Math. Software*, 38(1):1–25, 2011.
- [16] T. A. DAVIS et E. P. NATARAJAN : *Sparse matrix methods for circuit simulation problems*. In *Scientific Computing in Electrical Engineering SCEE 2010*, p. 3–14. Springer, 2012.
- [17] V. DOLEAN, P. JOLIVET et F. NATAF : *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*. SIAM, 2015.

- [18] I. S. DUFF et J. A. SCOTT : **Stabilized bordered block diagonal forms for parallel sparse solvers.** *Parallel Comput.*, 31(3-4):275–289, 2005.
- [19] H. C. ELMAN, A. RAMAGE et D. J. SILVESTER : Incompressible Flow and Iterative Solver Software (IFISS), version 3.6. <http://www.manchester.ac.uk/ifiss>, 2019.
- [20] H. C. ELMAN, D. J. SILVESTER et A. J. WATHEN : *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics.* Oxford University Press, USA, 2014.
- [21] H. C. ELMAN : **Preconditioners for saddle point problems arising in computational fluid dynamics.** *Appl. Numer. Math.*, 43(1-2):75–89, 2002.
- [22] R. ESTRIN, D. ORBAN et M. A. SAUNDERS : **Euclidean-norm error bounds for SYMMLQ and CG.** *SIAM J. Matrix Anal. Appl.*, 40(1):235–253, 2019.
- [23] R. ESTRIN, D. ORBAN et M. A. SAUNDERS : **LNLQ: An Iterative Method for Least-Norm Problems with an Error Minimization Property.** *SIAM J. Matrix Anal. Appl.*, 40(3):1102–1124, 2019.
- [24] R. ESTRIN, D. ORBAN et M. A. SAUNDERS : **LSLQ: An Iterative Method for Least-Squares with an Error Minimization Property.** *SIAM J. Matrix Anal. Appl.*, 40(1):254–275, 2019.
- [25] R. ESTRIN et C. GREIF : **SPMR: A family of saddle-point minimum residual solvers.** *SIAM J. Sci. Comput.*, 40(3):1884–1914, 2018.
- [26] M. C. FERRIS et J. D. HORN : **Partitioning mathematical programs for parallel solution.** *Math. Program.*, 80(1):35–61, 1998.
- [27] R. FLETCHER : **Conjugate gradient methods for indefinite systems.** In *Numerical Analysis*, p. 73–89. Springer, 1976.
- [28] D. C.-L. FONG et M. A. SAUNDERS : **LSMR: An iterative algorithm for sparse least-squares problems.** *SIAM J. Sci. Comput.*, 33(5):2950–2971, 2011.
- [29] D. C.-L. FONG et M. A. SAUNDERS : **CG versus MINRES: an empirical comparison.** *Sultan Qaboos University Journal for Science*, 17(1):44–62, 2012.
- [30] R. W. FREUND et N. M. NACHTIGAL : **QMR: a quasi-minimal residual method for non-Hermitian linear systems.** *Numer. Math.*, 60(1):315–339, 1991.
- [31] R. W. FREUND : **A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems.** *SIAM J. Sci. Comput.*, 14(2):470–482, 1993.
- [32] M. P. FRIEDLANDER et D. ORBAN : **A primal–dual regularized interior-point method for convex quadratic programs.** *Math. Program. Comp.*, 4(1):71–107, 2012.
- [33] K. A. GALLIVAN, S. THIRUMALAI, P. V. DOOREN et V. VERMAUT : **High performance algorithms for Toeplitz and block Toeplitz matrices.** *Linear Algebra and its Applications*, 241:343–388, 1996.
- [34] G. H. GOLUB et W. KAHAN : **Calculating the Singular Values and Pseudo-Inverse of a Matrix.** *SIAM J. Numer. Anal.*, 2(2):205–224, 1965.
- [35] G. H. GOLUB, M. STOLL et A. WATHEN : **Approximation of the scattering amplitude and linear systems.** *ETNA*, 31(2008):178–203, 2008.
- [36] G. H. GOLUB et R. UNDERWOOD : **The block Lanczos method for computing eigenvalues.** In *Mathematical software*, p. 361–377. Elsevier, 1977.
- [37] G. H. GOLUB et C. F. VAN LOAN : *Matrix Computations.* The Johns Hopkins University Press, fourth éd., 2013.

- [38] A. GREENBAUM : *Iterative methods for solving linear systems*. SIAM, 1997.
- [39] A. GREENBAUM, V. PTÁK et Z. STRAKOŠ : *Any nonincreasing convergence curve is possible for GMRES*. *SIAM J. Matrix Anal. Appl.*, 17(3):465–469, 1996.
- [40] C. GREIF, E. MOULDING et D. ORBAN : *Bounds on the Eigenvalues of Matrices Arising from Interior-Point Methods*. *SIAM J. Optim.*, 24(1):49–83, 2014.
- [41] C. GREIF et M. WATHEN : *Conjugate gradient for nonsingular saddle-point systems with a maximally rank-deficient leading block*. *J. Comput. Appl. Math.*, 358:1–11, 2019.
- [42] G. GUENNEBAUD, B. JACOB, P. AVERY, A. BACHRACH, S. BARTHELEMY *et al.* : *Eigen*, 2010.
- [43] A. E. GUENNOUNI, K. JBILOU et H. SADOK : *The block Lanczos method for linear systems with multiple right-hand sides*. *Appl. Numer. Math.*, 51(2-3):243–256, 2004.
- [44] M. H. GUTKNECHT et T. SCHMELZER : *The block grade of a block Krylov space*. *Linear Algebra Appl.*, 430(1):174–185, 2009.
- [45] C. R. HARRIS, K. J. MILLMAN, S. J. van der WALT, R. GOMMERS, P. VIRTANEN, D. COURNAPEAU, E. WIESER, J. TAYLOR, S. BERG, N. J. SMITH, R. KERN, M. PICUS, S. HOYER, M. H. van KERKWIJK, M. BRETT, A. HALDANE, J. F. del RÍO, M. WIEBE, P. PETERSON, P. GÉRARD-MARCHANT, K. SHEPARD, T. REDDY, W. WECKESSER, H. ABBASI, C. GOHLKE et T. E. OLIPHANT : *Array programming with NumPy*. *Nature*, 585(7825):357–362, 2020.
- [46] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS *et al.* : *An overview of the Trilinos project*. *ACM Trans. Math. Software*, 31(3):397–423, 2005.
- [47] R. HERZOG et K. SOODHALTER : *A Modified Implementation of MINRES to Monitor Residual Subvector Norms for Block Systems*. *SIAM J. Sci. Comput.*, 39(6):A2645–A2663, 2017.
- [48] M. R. HESTENES et E. STIEFEL : *Methods of Conjugate Gradients for Solving Linear Systems*. *J. Res. Natl. Bur. Stand.*, 49(6):409–436, 1952.
- [49] I. C. IPSEN : *A note on preconditioning nonsymmetric matrices*. *SIAM J. Sci. Comput.*, 23(3):1050–1051, 2001.
- [50] I. C. IPSEN et C. D. MEYER : *The idea behind Krylov methods*. *The American mathematical monthly*, 105(10):889–899, 1998.
- [51] E. F. KAASSCHIETER : *Preconditioned conjugate gradients for solving singular systems*. *J. Computational and Applied Mathematics*, 24(1-2):265–275, 1988.
- [52] G. KARYPIS et V. KUMAR : *A fast and high quality multilevel scheme for partitioning irregular graphs*. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.
- [53] A. N. KRYLOV : *On the numerical solution of the equation by which, in technical matters, frequencies of small oscillations of material systems are determined*. *Izvestija AN SSSR (News of Academy of Sciences of the USSR), Otdel. mat. i estest. nauk*, 7(4):491–539, 1931.
- [54] M. LAI : *A note on finite difference discretizations for Poisson equation on a disk*. *Numer. Meth. Part. D. E.*, 17(3):199–203, 2001.
- [55] C. LANCZOS : *An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators*. *J. Res. Natl. Bur. Stand.*, 45:225–280, 1950.
- [56] J. LU et D. DARMOFAL : *A Quasi-Minimal Residual Method for Simultaneous Primal-Dual Solutions and Superconvergent Functional Estimates*. *SIAM J. Sci. Comput.*, 24(5):1693–1709, 2003.

- [57] D. G. LUENBERGER : **The conjugate residual method for constrained minimization problems.** *SIAM J. Numer. Anal.*, 7(3):390–398, 1970.
- [58] MATLAB : *version 9.13.0 (R2022b)*. The MathWorks Inc., Natick, Massachusetts, 2022.
- [59] P. L. MONTGOMERY : **A block Lanczos algorithm for finding dependencies over GF(2).** In *International Conference on the Theory and Applications of Cryptographic Techniques*, p. 106–120. Springer, 1995.
- [60] A. MONTOISON, D. ORBAN et CONTRIBUTORS : **Krylov.jl: A Julia Basket of Hand-Picked Krylov Methods.** <https://github.com/JuliaSmoothOptimizers/Krylov.jl>, 2020.
- [61] A. MONTOISON et 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.
- [62] A. MONTOISON et D. ORBAN : **TriCG and TriMR: Two Iterative Methods for Symmetric Quasi-definite Systems.** *SIAM J. Sci. Comput.*, 43(4):2502–2525, 2021.
- [63] A. MONTOISON et D. ORBAN : **GPMP: An Iterative Method for Unsymmetric Partitioned Linear Systems.** *SIAM J. Matrix Anal. Appl.*, 44(1):293–311, 2023.
- [64] A. MONTOISON et D. ORBAN : **Krylov.jl: A Julia basket of hand-picked Krylov methods.** *Journal of Open Source Software*, 8(89):5187, 2023.
- [65] A. MONTOISON, D. ORBAN et M. A. SAUNDERS : **MinAres: An iterative solver for symmetric linear systems.** Cahier du GERAD G-2023-40, GERAD, Montréal, QC, Canada.
- [66] M. F. MURPHY, G. H. GOLUB et A. J. WATHEN : **A note on preconditioning for indefinite linear systems.** *SIAM J. Sci. Comput.*, 21(6):1969–1972, 2000.
- [67] J. NOCEDAL et S. J. WRIGHT : *Numerical Optimization*. Springer New York, NY, 2 éd., 2006.
- [68] D. P. O’LEARY : **The block conjugate gradient algorithm and related methods.** *Linear Algebra Appl.*, 29:293–322, 1980.
- [69] D. ORBAN : A Collection of Linear Systems Arising from Interior-Point Methods for Quadratic Optimization. Cahier du GERAD G-2015-117, GERAD, Montréal, QC, Canada, 2015.
- [70] D. ORBAN et M. ARIOLI : *Iterative Solution of Symmetric Quasi-Definite Linear Systems*, vol. 3 de *Spotlights*. SIAM, Philadelphia, PA, 2017.
- [71] C. C. PAIGE : **Bidiagonalization of matrices and solution of linear equations.** *SIAM J. Numer. Anal.*, 11(1):197–209, 1974.
- [72] C. C. PAIGE et M. A. SAUNDERS : **Solution of Sparse Indefinite Systems of Linear Equations.** *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.
- [73] C. C. PAIGE et M. A. SAUNDERS : **LSQR: An algorithm for sparse linear equations and sparse least squares.** *ACM Trans. Math. Software*, 8(1):43–71, 1982.
- [74] C. C. PAIGE, I. PANAYOTOV et J.-P. M. ZEMKE : **An augmented analysis of the perturbed two-sided Lanczos tridiagonalization process.** *Linear Algebra Appl.*, 447:119–132, 2014.
- [75] B. N. PARLETT, D. R. TAYLOR et Z. A. LIU : **A look-ahead Lanczos algorithm for unsymmetric matrices.** *Math. Comp.*, 44(169):105–124, 1985.
- [76] N. A. PIERCE et M. B. GILES : **Adjoint Recovery of Superconvergent Functionals from PDE Approximations.** *SIAM Rev.*, 42(2):247–264, 2000.
- [77] L. REICHEL et Q. YE : **A generalized LSQR algorithm.** *Numer. Linear Algebra Appl.*, 15(7):643–660, 2008.

- [78] Y. SAAD : Practical use of some Krylov subspace methods for solving indefinite and nonsymmetric linear systems. *SIAM J. Sci. and Statist. Comput.*, 5(1):203–228, 1984.
- [79] Y. SAAD : A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.
- [80] Y. SAAD et 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.
- [81] Y. SAAD : Krylov subspace methods for solving large unsymmetric linear systems. *Math. Comp.*, 37(155):105–126, 1981.
- [82] Y. SAAD : *Iterative methods for sparse linear systems*. SIAM, 2003.
- [83] Y. SAAD : The Origin and Development of Krylov Subspace Methods. *Computing in Science & Engineering*, 24(4):28–39, 2022.
- [84] Y. SAAD et K. WU : DQGMRES: A direct quasi-minimal residual algorithm based on incomplete orthogonalization. *Numer. Linear Algebra Appl.*, 3(4):329–343, 1996.
- [85] M. A. SAUNDERS, H. D. SIMON et E. L. YIP : Two Conjugate-Gradient-Type Methods for Unsymmetric Linear Equations. *SIAM J. Numer. Anal.*, 25(4):927–940, 1988.
- [86] M. SCHANEN, D. A. MALDONADO, F. PACAUD, A. MONTISON, M. ANITESCU, K. KIM, Y. KIM, V. RAO et A. SUBRAMANYAM : Julia as a portable high-level language for numerical solvers of power flow equations on GPU architectures. Cahier du GERAD G-2020-74, GERAD, Montréal, QC, Canada, 2020.
- [87] P. SONNEVELD : CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear systems. *SIAM J. Sci. and Statist. Comput.*, 10(1):36–52, 1989.
- [88] G. STEWART : Block Gram–Schmidt orthogonalization. *SIAM J. Sci. Comput.*, 31(1):761–775, 2008.
- [89] E. STIEFEL : Relaxationsmethoden bester Strategie zur Lösung linearer Gleichungssysteme. *Commentarii Mathematici Helvetici*, 29(1):157–179, 1955.
- [90] L. N. TREFETHEN et D. BAU III : *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.
- [91] H. A. van der VORST : Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J. Sci. and Statist. Comput.*, 13(2):631–644, 1992.
- [92] R. J. VANDERBEI : Symmetric quasidefinite matrices. *SIAM J. Optim.*, 5(1):100–113, 1995.
- [93] P. VIRTANEN, R. GOMMERS, T. E. OLIPHANT, M. HABERLAND, T. REDDY, D. COURNAPEAU, E. BURROVSKI, P. PETERSON, W. WECKESSER, J. BRIGHT, S. J. VAN DER WALT, M. BRETT, J. WILSON, K. J. MILLMAN, N. MAYOROV, A. R. J. NELSON, E. JONES, R. KERN, E. LARSON, C. J. CAREY, İ. POLAT, Y. FENG, E. W. MOORE, J. VANDERPLAS, D. LAXALDE, J. PERKTOLD, R. CIRMANN, I. HENRIKSEN, E. A. QUINTERO, C. R. HARRIS, A. M. ARCHIBALD, A. H. RIBEIRO, F. PEDREGOSA, P. VAN MULBREGT et SciPy 1.0 CONTRIBUTORS : SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
- [94] R. WEISS : Error-Minimizing Krylov Subspace Methods. *SIAM J. Sci. Comput.*, 15:511–527, 1994.
- [95] T. XUEMIN et J. LI : BDDC for Nonsymmetric positive definite and symmetric indefinite problems. In *Domain Decomposition Methods in Science and Engineering XVIII*, p. 75–86. Springer Verlag, 2009.