A DISSERTATION

Extensions of the Conjugate Residual Method

(共役残差法の拡張)

by
Tomohiro Sogabe

Presented to

Department of Applied Physics, The University of Tokyo

Contents

List of Symbols

$A, \ldots, Z$	matrices
$\boldsymbol{a}, \ldots, \boldsymbol{z}$	vectors
$\alpha, \beta, \ldots, \omega$	scalars
A^{T}	transpose of A
A^{H}	conjugate transpose (Hermitian) of A
A^{-1}	matrix inverse
$A^{-\mathrm{H}}$	the inverse of A^{H}
$\bar{\alpha}$	complex conjugate of the scalar α
$a_{i,j}$	matrix element
a_i	vector element
u_x, u_{xx}	first, second derivative with respect to x
$\boldsymbol{x}_n$	vector $\boldsymbol{x}$ in the nth iteration
$(\boldsymbol{x}, \boldsymbol{y})$	vector dot product (inner product), defined as $\boldsymbol{x}^{\mathrm{H}}\boldsymbol{y}$
$\mathrm{diag}(\alpha, \beta, \ldots)$	diagonal matrix constructed from scalars $\alpha, \beta, \ldots$
$\mathrm{span}\{\boldsymbol{a}, \boldsymbol{b}, \ldots\}$	spanning space of vectors $\boldsymbol{a}, \boldsymbol{b}, \ldots$
$K_n(A, \boldsymbol{u})$	Krylov subspace, defined as $\mathrm{span}\{\boldsymbol{u}, A\boldsymbol{u}, \ldots, A^{n-1}\boldsymbol{u}\}$
$\kappa(A)$	spectral condition number of matrix A
$\rho(A)$	spectral radius of matrix A
$\lambda_{\max}(A), \lambda_{\min}(A)$	the largest (resp. smallest) absolute value of eigenvalues of A
$\max\{S\}, \min\{S\}$	maximum (resp. minimum) value in set S
$R_n(\lambda)$	Lanczos polynomial of degree n
$\mathcal{R}$	set of real numbers
$\mathcal{R}^n$	real n-space
$\mathcal{C}$	set of complex numbers
$\mathcal{C}^n$	complex n-space
$\|X\|$	matrix 2-norm
$\|X\|_F$	Frobenius norm
$\|\boldsymbol{x}\|$	vector 2-norm
$\|\boldsymbol{x}\|_A$	the "A-norm", defined as $(\boldsymbol{x}, A\boldsymbol{x})^{1/2}$
B	bidiagonal matrix
D	diagonal matrix
H	Hessenberg matrix
K	preconditioner
L	(strictly) lower triangular matrix
R, U	(strictly) upper triangular matrix
T	tridiagonal matrix
δ_{ij}	Kronecker delta
Σ	summation
$\prod$	multiplication
$\mathcal{O}(\cdot)$	"big-oh" asymptotic bound

Chapter 1

Introduction

In many fields of scientific computing, we have to face the fact that most computational time is spend in solving systems of linear equations

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{1.1}$$

where $\boldsymbol{x}, \boldsymbol{b} \in \mathcal{C}^N$ and the coefficient matrix $A \in \mathcal{C}^{N\times N}$ is generally large and *sparse*, *i.e.*, the percentage of zero entries of the matrix is very large. Hence, it is of great importance to study efficient solvers for such systems.

Numerical methods for solving (1.1) have been studied by many researchers, and they are roughly classified into the following three groups:

- Direct Methods;
- Stationary Iterative Methods;
- Krylov Subspace Methods (Nonstationary Iterative Methods).

Direct methods are named after the idea of direct computations of $A^{-1}\boldsymbol{b}$. For efficiency, matrix decomposition techniques are used such as the LU decomposition, or Gaussian elimination, and the Cholesky decomposition. In many numerical experiments, direct methods often give high accuracy of the solution; however they require much memory storage and computational cost of $\mathcal{O}(N^3)$ since the decomposed matrices are, generally, not sparse any longer because of so called *fill-in*. Hence, decomposition techniques for sparse matrices have been studied for years [22, 23, 28].

Stationary iterative methods generate approximate solution vectors of (1.1) via the simple form $\boldsymbol{x}_n = B\boldsymbol{x}_{n-1} + \boldsymbol{c}$. Since the matrix B and the vector $\boldsymbol{c}$ do not depend on the iteration number n, this class of methods is called "stationary". We can obtain various methods by the choice of B and $\boldsymbol{c}$ such as the Jacobi method, the Gauss-Seidel method, and the Successive OverRelaxation (SOR) method [37, 96]. Stationary iterative methods have an advantage over direct methods in that they only need sparse matrix-vector product and vector update, and thus they do not require much memory storage. Moreover, these methods can be easily implemented, and have been studied well for their practical uses. However, they have some weaknesses: the class of matrices is limited; in terms of infinite precision arithmetic, the exact solution can not be obtained by finite iteration number, which may be critical when they show slow convergence behavior. For more details, see a recent book by Varga [90] that contains much of the theory and some recent results.

Krylov subspace methods (KS methods hereafter) have been developed recently; their algorithms are usually harder to be understood, but they can be highly effective. KS methods are named after the idea that they generate the nth approximate solution vector $\boldsymbol{x}_n$ of (1.1) over n-dimensional Krylov subspace. Since KS methods differ from stationary iterative methods in that required information for updating $\boldsymbol{x}_n$ changes at each iteration step, they are also called nonstationary iterative methods. One of the most well-known KS methods is Conjugate Gradient (CG) proposed by Hestenes and Stiefel [53]. CG is regarded as an ideal solver for Hermitian positive definite linear systems, and in this case, CG minimizes A-norm of the error $\|\boldsymbol{x} - \boldsymbol{x}_n\|_A$. On the other hand, since CG does not minimize anything if the coefficient matrix A is Hermitian indefinite, MINRES [62], CR [82], and SYMMLQ [62] are often chosen in many practical situations for indefinite systems. MINRES and CR minimize the 2-norm of the residual $\|\boldsymbol{b}-A\boldsymbol{x}_n\|$, and SYMMLQ minimizes the 2-norm of the error $\|\boldsymbol{x} - \boldsymbol{x}_n\|$. For the property, SYMMLQ seems to be ideal; however since it finds approximate solutions in different subspaces from ones used by CG, MINRES, and CR, it is essentially difficult to make comparison with others. Moreover, the 2-norm of the error is not known, and the practical criteria for convergence is the norm of the residual. Hence, it requires additional costs for computing residual vectors. Here is one note that we can also apply CG to Hermitian indefinite systems but in this case it may suffer from *breakdown*, *i.e.*, division by zero.

If the coefficient matrix A is non-Hermitian, the above useful solvers can not be applied directly, and thus it is natural to consider transformed linear systems $A^{\mathrm{H}}A\boldsymbol{x} = A^{\mathrm{H}}\boldsymbol{b}$ or $AA^{\mathrm{H}}\tilde{\boldsymbol{x}} = \boldsymbol{b}, \boldsymbol{x} = A^{\mathrm{H}}\tilde{\boldsymbol{x}}$. $A^{\mathrm{H}}A$ and AA^{H} are Hermitian positive definite, and thus CG should be chosen as their solvers. These ideas lead to CGNR [53] and CGNE [18]. It is known that LSQR [63] gives better results. These methods are effective in certain class of systems; however if the original system (1.1) is ill-conditioned, then these methods may give unreliable solutions because of the use of $A^{\mathrm{H}}A$ or AA^{H} with the square of the condition number of A. Hence, generalized methods of CG, MINRES, and SYMMLQ have been studied to solve the original system. Bi-CG [56, 35] and FOM [66] can be regarded as generalizations of CG; GMRES [71] and GCR [31] are natural generalizations of MINRES and CR respectively, and they minimize the norm of the residual; GMERR [94] is less known but it minimizes the 2-norm of the error. This property is similar to SYMMLQ.

Of various Krylov subspace methods for non-Hermitian linear systems, Bi-CG and GMRES play a very important role in designing more practical variants of them. Since Bi-CG requires transposed matrix-vector products in its algorithm, P. Sonneveld [81] obtained a transpose-free variant, called CGS, by using the square of the Bi-CG residual polynomial. Ideally, CGS would double the convergence rate of Bi-CG, but in many cases, CGS shows much irregular convergence behavior than Bi-CG. Hence, H. A. van der Vorst [86] derived one of the most successful variant of Bi-CG, known as Bi-CGSTAB, by using the product of the Bi-CG residual polynomial and two-term recurrences for smoothing its residual 2-norm. Based on the idea of Bi-CGSTAB, various generalized methods were proposed such as Bi-CGSTAB2 [51], Bi-CGSTAB(ℓ) [79], and GPBi-CG [97]. On the other hand, QMR [41] is an attractive variant of Bi-CG in that it often shows smoother convergence behavior and unlike Bi-CG, it can evade certain (near) breakdowns by using suitable *look-ahead* strategy, *e.g.*, [83, 64]. Similar to CGS, a transposed-free variant of QMR, called TFQMR [39], was proposed. Using the idea of TFQMR, QMRCGSTAB [15] was proposed as the analogy of Bi-CGSTAB, and then QMRCGSTAB(k) [84] was proposed as a generalization of QMRCGSTAB. On the other hand, (full) GMRES is a robust algorithm but the computational cost and memory requirement grow linearly as the iteration step increases. Hence, it would

be impractical for very large systems. To avoid such problem, restarted and truncated variants were proposed, which are called GMRES(m) [71] and DQGMRES [73] respectively; however, these approaches often increase the required number of iterations, and these variants sometimes cause stagnation. Hence, to improve the performance of them, preconditioning technique such as FGMRES [68], GMRESR [89] and deflated restarting technique such as GMRES_DR [60] and DEFLGMRES(m, l) [32] have been developed.

We have described a brief history of the KS methods for Hermitian and non-Hermitian cases. It is also natural to consider an algorithm for a certain class of matrices. Here, we give a brief review of the KS methods for complex symmetric matrices *i.e.*, $A = A^{\mathrm{T}} \neq A^{\mathrm{H}}$ since there is a strong need for the fast solution in some research fields such as numerical acoustics, numerical simulation of electromagnetic high-voltage fields, numerical computation of Green's function, and discretization of the Helmholtz equation. To solve complex symmetric linear systems effectively, some useful algorithms have been proposed such as COCG [88], and QMR [38] (or QMR_SYM called in [87, p.112]). Since these methods have only one matrix-vector product and use short-term recurrences at each iteration step, they have advantage over any Krylov solver for non-Hermitian systems. If A is real symmetric, COCG and QMR are mathematically equivalent to CG and MINRES.

In this thesis, we will provide mainly the following four ideas:

(1) CR is extended to non-Hermitian linear systems with low memory requirement. The main idea for the extension is to relax the condition of CR that the residual vectors are A-orthogonal. The resulting algorithm is named Bi-CR whose residual vectors are A-biorthogonal. Some of derivation processes of Bi-CR are based on [75] and [76]. In exact precision arithmetic, Bi-CR generates an exact solution in finite iteration steps. While Bi-CR does not have a global minimization property, we will show that Bi-CR often gives smoother convergence behavior than Bi-CG in the residual 2-norm. This convergence behavior leads to the following two ideas, (2) and (3), for obtaining useful solvers according to the type of matrices;

(2) Based on [78], CR is extended to complex symmetric linear systems with low memory requirement. The idea is very similar to the above one (1). The resulting algorithm is referred to as COCR whose residual vectors are conjugate A-orthogonal. The computational cost of COCR is about half of the Bi-CR's. We will show that COCR is also obtained by applying Bi-CR to complex symmetric linear systems;

(3) Since Bi-CR requires transpose matrix-vector products, a transpose-free variant of Bi-CR, named CRS, is proposed. The idea comes from the definition of the nth residual vector of CRS as the product of the Bi-CR residual polynomial of order n and the nth Bi-CR residual vector. The relationship between Bi-CR and CRS is similar to that of Bi-CG and CGS. As a generalization of CRS, we will give a framework of product-type methods based on Bi-CR. The framework mainly comes from [77];

(4) To improve the performance of CGS, the convergence behavior of CGS is stabilized by introducing a two-term recurrence which includes the CGS residual polynomial and choosing free parameters so that the residual 2-norm is locally minimized. The algorithm is referred to as SCGS. We will show that in exact precision arithmetic SCGS always converges faster than CGS in terms of the number of iteration steps. This work is based on [74].

The thesis is organized as follows:

◇ Chapter 2: we give mathematical preliminaries sufficient to discuss the theory of the Krylov subspace methods, and then we describe well-known Krylov subspace methods for solving Hermitian, complex symmetric, and non-Hermitian linear systems;

◇ Chapter 3: we give the first idea (1). First, Bi-CR is derived as an extension of CR, and then other derivations are given. Second, we discuss its properties. Finally, we compare Bi-CR with Bi-CG to see whether Bi-CR has a potential to be a basic solver as well as Bi-CG;

◇ Chapter 4: we give the second idea (2). First, CR is extended to complex symmetric linear systems and we show that COCR is also derived from the application of Bi-CR to complex symmetric linear systems. Second, we discuss its properties. Finally, we give some numerical experiments for evaluating its performance;

◇ Chapter 5: we give the third idea (3). We show that a transpose-free variant of Bi-CR, or CRS, can be obtained by the square of the Bi-CR residual polynomial, and then we give a brief framework of product-type methods based on Bi-CR. Finally, we compare CRS with other successful solvers such as CGS, Bi-CGSTAB, and GPBi-CG;

◇ Chapter 6: we give the fourth idea (4). SCGS is proposed for stabilizing the convergence behavior of CGS, and then we prove the theorem that SCGS always converges faster than CGS in terms of the number of iterations. Finally, we report numerical results of CGS and SCGS;

◇ Chapter 7: we make some concluding remarks and give future work.

Since various methods are proposed and described, it may be complicated for readers to see what work is actually done and new in the present thesis. Hence, we give a research flow chart below.

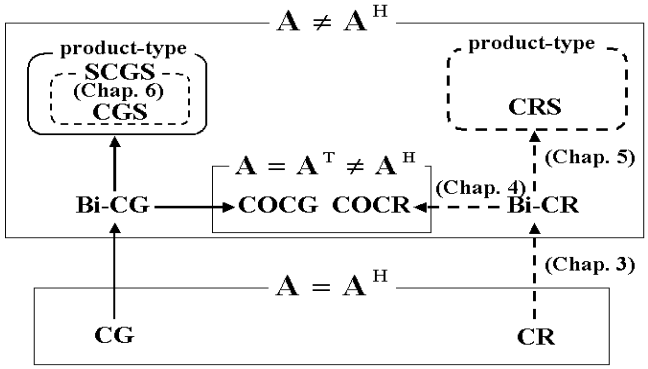

Figure 1.1: Research flow chart.

Chapter 2

Krylov subspace methods

In this chapter, we will give derivations of representative Krylov subspace methods that fall in three categories; the ones for Hermitian, complex symmetric, and non-Hermitian linear systems. We also describe useful preconditioners that have a great potential for accelerating the speed of convergence of these methods. First of all, a framework of Krylov subspace methods is given below.

Let $\boldsymbol{x}_0$ be an initial guess, and let $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$ be the corresponding residual vector. Then, Krylov subspace methods generate approximate solutions over the following affine space:

Krylov subspace condition:

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + \boldsymbol{z}_n, \quad \boldsymbol{z}_n \in K_n(A, \boldsymbol{r}_0), \tag{2.1}$$

where $K_n(A, \boldsymbol{r}_0) := \mathrm{span}\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{n-1}\boldsymbol{r}_0\}$. Then, the corresponding residual vector $\boldsymbol{r}_n$ is given by

$$\boldsymbol{r}_n := \boldsymbol{b} - A\boldsymbol{x}_n = \boldsymbol{r}_0 - A\boldsymbol{z}_n, \quad \boldsymbol{r}_n \in K_{n+1}(A, \boldsymbol{r}_0). \tag{2.2}$$

Since $\boldsymbol{x}_n$ can not be uniquely determined by using the above condition (2.1), one of the following conditions needs to be imposed for obtaining a unique approximate solution:

Ritz-Galerkin approach:

$$\boldsymbol{r}_n \perp K_n(A, \boldsymbol{r}_0), \tag{2.3}$$

Petrov-Galerkin approach:

$$\boldsymbol{r}_n \perp K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*), \tag{2.4}$$

Minimal residual approach:

$$\boldsymbol{r}_n = \min_{\boldsymbol{x} \in \boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)} \|\boldsymbol{b} - A\boldsymbol{x}\|. \tag{2.5}$$

Here, $\boldsymbol{r}_0^*$ is an arbitrary vector such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0$. To achieve one of the above approaches, efficient numerical processes for generating (bi-)orthogonal basis vectors of $K_n(A, \boldsymbol{r}_0)$ are required. If the coefficient matrix A is Hermitian, the Lanczos process plays an important role in meeting the above requirement since only a three-term recurrence

is used for generating one of the orthogonal basis. For non-Hermitian case, the Lanczos process can not be used. Instead, the Arnoldi process or the bi-Lanczos process are useful even though they lose one of the following two properties; three-term recurrences and orthogonality. The Arnoldi process can generate the orthogonal basis at the cost of low computational costs and memory. On the other hand, the bi-Lanczos process generates not orthogonal but bi-orthogonal basis with coupled three-term recurrences. The details of these processes will be discussed later.

Now, let us show the relationship among representative Krylov subspace methods and the above three approaches in Table 2.1 and Table 2.2. Looking at Table 2.2, we can also

Table 2.1: Krylov subspace methods for Hermitian linear systems.

	Ritz-Galerkin approach	Minimal residual approach
Lanczos process	CG [53]	MINRES [62], CR [82]

Table 2.2: Krylov subspace methods for non-Hermitian linear systems.

	Petrov-Galerkin approach	Minimal residual approach
Arnoldi process	†	GMRES [71], GCR [31]
Bi-Lanczos process	Bi-CG [56, 35]	†

consider iterative solvers that belong to †. However, these solvers are useless in terms of computational costs and memory. Comparing Table 2.2 with Table 2.1, we see that Bi-CG, GMRES, and GCR are natural generalizations of CG, MINRES, and CR.

In the next two sections, we describe some processes to generate orthonormal basis of any subspace and give some useful processes to generate basis of the Krylov subspace.

2.1 Gram-Schmidt orthogonalization

Let $A \in \mathcal{C}^{N\times m}$, $N \geq m$ be a matrix of full rank with columns $[\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m]$. We consider obtaining the vectors $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_m$, where $\boldsymbol{q}_j$ is orthogonal to $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}$, lies in span$\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j\}$, and has norm 1. One of the simplest ways for achieving the purpose is known as the Classical Gram-Schmidt (CGS) orthogonalization described below:

Algorithm 2.1: CGS orthogonalization

$$
\begin{aligned}
&\text{set } r_{1,1} = \|\boldsymbol{a}_1\|,\ \boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{r_{1,1}},\\
&\text{for } n = 1, 2, \ldots, m-1, \text{ do:}\\
&\quad r_{i,n+1} = (\boldsymbol{q}_i, \boldsymbol{a}_{n+1}), \quad i = 1, 2, \ldots, n,\\
&\quad \tilde{\boldsymbol{q}}_{n+1} = \boldsymbol{a}_{n+1} - \sum_{i=1}^{n} r_{i,n+1}\boldsymbol{q}_i, \qquad (2.6)\\
&\quad r_{n+1,n+1} = \|\tilde{\boldsymbol{q}}_{n+1}\|,\\
&\quad \boldsymbol{q}_{n+1} = \frac{\tilde{\boldsymbol{q}}_{n+1}}{r_{n+1,n+1}}.\\
&\text{end}
\end{aligned}
$$

By using $Q_n := [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n]$, (2.6) can be written as the following matrix form:

$$\tilde{\boldsymbol{q}}_{n+1} = (I - Q_n Q_n^{\mathrm{H}})\boldsymbol{a}_{n+1}.$$

The matrix $Q_n Q_n^{\mathrm{H}}$ with rank n has n multiple eigenvalues of 1, and thus the matrix $I - Q_n Q_n^{\mathrm{H}}$ has $N-n$ multiple eigenvalues of 1 and the rank is $N-n$. In terms of computation, although the rank of $I - Q_n Q_n^{\mathrm{H}}$ is not full, the matrix can be highly ill-conditioned because of the rounding error. Hence, this matrix-vector multiplication may lead to poor orthogonalities of the basis vectors. On the other hand, an orthogonalization process using the Householder transformation gives much better accuracy of orthogonality of the basis vectors since it requires only unitary matrix-vector products.

It is remarkable that changing order of orthogonalization process of Algorithm 2.1 gives more accurate orthonormal basis vectors. The resulting algorithm is known as the Modified Gram-Schmidt (MGS) orthogonalization shown in Algorithm 2.2.

Algorithm 2.2: MGS orthogonalization

$$
\begin{aligned}
&\text{set } r_{1,1} = \|\boldsymbol{a}_1\|,\ \boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{r_{1,1}},\\
&\text{for } n = 1, 2, \ldots, m-1, \text{ do:}\\
&\quad \text{for } i = 1, 2, \ldots, n, \text{ do:}\\
&\qquad r_{i,n+1} = (\boldsymbol{q}_i, \boldsymbol{a}_{n+1}),\\
&\qquad \boldsymbol{a}_{n+1} \ = \boldsymbol{a}_{n+1} - r_{i,n+1}\boldsymbol{q}_i,\\
&\quad \text{end}\\
&\quad r_{n+1,n+1} = \|\boldsymbol{a}_{n+1}\|,\\
&\quad \boldsymbol{q}_{n+1} = \frac{\boldsymbol{a}_{n+1}}{r_{n+1,n+1}}.\\
&\text{end}
\end{aligned}
\tag{2.7}
$$

CGS and MGS can be regarded as the QR factorization of A. Let $Q \in \mathcal{C}^{N\times m}$ be a matrix with columns $[\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_m]$ and $R \in \mathcal{C}^{m\times m}$ be an upper triangular matrix with the entries $r_{i,n}$. Then, we obtain the following QR factorization of A:

$$A = QR \Longleftrightarrow [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m] = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m] \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,m} \\ & r_{2,2} & \cdots & r_{2,m} \\ & & \ddots & \vdots \\ & & & r_{m,m} \end{pmatrix}.$$

To compare CGS and MGS, we use the Hilbert matrix whose i, j element is defined as $(i + j - 1)^{-1}$. The Hilbert matrix is well known as a highly ill-conditioned matrix, and thus it may clarify the difference between the numerical stabilities of the two methods. The condition number of the Hilbert matrix is shown in Table 2.3, and the result of the comparison between CGS and MGS is shown in Fig. 2.1, where the vertical axis represents accuracy of orthogonalities defined as $\log_{10} \|I - \tilde{Q}^{\mathrm{H}}\tilde{Q}\|$ with a computed matrix $\tilde{Q}$, and the horizontal axis represents the order of the Hilbert matrix.

Table 2.3: Condition number of the Hilbert matrix.

Order	2	4	6	8	10	12
Cond.	1.9×10^1	1.6×10^4	1.5×10^7	1.5×10^{10}	1.6×10^{13}	1.8×10^{16}

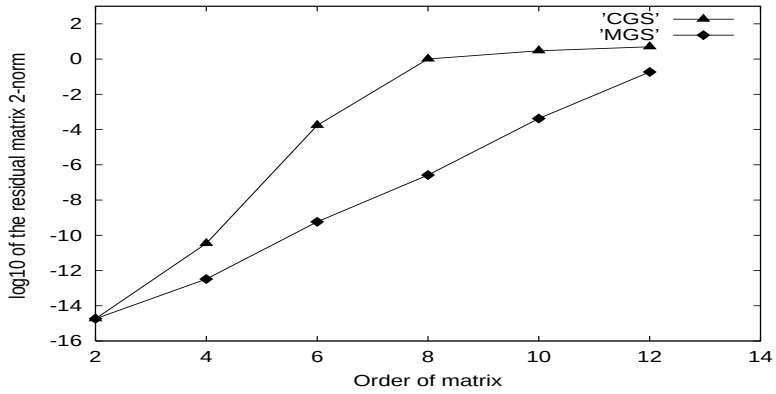

Figure 2.1: Comparison of CGS and MGS.

We can see from Fig. 2.1 that MGS is numerically robuster than CGS. Actually, in terms of rounding error analysis, it is known that CGS generates a matrix $\tilde{Q}$ that satisfies

$$\|I - \tilde{Q}^{\mathrm{H}}\tilde{Q}\| \approx u \cdot \kappa^2(A),$$

and MGS produces a matrix $\tilde{Q}$ that satisfies

$$\|I - \tilde{Q}^{\mathrm{H}}\tilde{Q}\| \approx u \cdot \kappa(A), \tag{2.8}$$

where u is a unit roundoff. See, *e.g.*, [45] for CGS and [48, p.232] for MGS. We can also see from Fig. 2.1 and Table 2.3 that the accuracy of orthogonalities by MGS is nearly proportional to the condition number of A. This result follows the above proposition (2.8).

For more accurate basis, the Householder implementation is preferred since it generates the basis such that

$$\|I - \tilde{Q}^{\mathrm{H}}\tilde{Q}\| \approx u.$$

See [12]. On the other hand, reorthogonalization strategy is competitive with the Householder variant, *e.g.*, it is shown that under some assumptions CGS with only one reorthogonalization guarantees the level of orthogonality which is close to the unit roundoff [45]. However, these implementations cost about twice as much as MGS.

In the next section, we consider several ways to construct orthonormal basis for Krylov subspaces.

2.2 Orthogonalization processes for the Krylov subspace

In the previous section, we have seen two orthogonalization processes of linearly independent vector sequences, and we could see from a simple example that MGS was numerically

robuster than CGS. In this section, we describe orthogonalization processes for the Krylov subspace $K_m(A, \boldsymbol{r}_0)$.

There are several (bi-)orthogonalization processes for $K_m(A, \boldsymbol{r}_0)$ corresponding to the matrix properties:

- $A \neq A^{\mathrm{H}}$: the Arnoldi process [1] or the bi-Lanczos process [55];
- $A \neq A^{\mathrm{H}}, A = A^{\mathrm{T}}$: the complex symmetric Lanczos process [55, 57];
- $A = A^{\mathrm{H}}$: the Lanczos process [55, 57].

The Arnoldi process and the bi-Lanczos process are well known as the Gram-Schmidt-style iterations for transforming a matrix into Hessenberg form and tridiagonal form. The complex symmetric Lanczos process and the Lanczos process are the special cases of the bi-Lanczos process when A is complex symmetric and Hermitian respectively. In that case, the coefficient matrix can be transformed into tridiagonal form that is complex symmetric (respectively, Hermitian). It is well known that they are numerically stabler processes than the (modified) Gram-Schmidt orthogonalization for constructing orthogonal basis of the Krylov subspace. This fact comes from the reason that CGS and MGS use the vector sequence $\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{m-1}\boldsymbol{r}_0$. Hence, for large m, $A^{m-2}\boldsymbol{r}_0$ tends to be close to $A^{m-1}\boldsymbol{r}_0$. More precisely, since these vectors approximately converge to an eigenvector corresponding to the largest eigenvalue in absolute value of A, this leads to severe cancellations.

2.2.1 The Arnoldi process

The Arnoldi process has been regarded as a useful orthonormalization procedure for the Krylov subspace $K_m(A, \boldsymbol{r}_0)$ generated by the non-Hermitian matrix A and the vector $\boldsymbol{r}_0$. There are several variants of the Arnoldi process such as the classical Gram-Schmidt variant and the modified Gram-Schmidt one. First, the classical Gram-Schmidt (CGS) variant is shown below.

Algorithm 2.3: The Arnoldi process with CGS

$$
\begin{aligned}
&\text{set } \boldsymbol{v}_1 = \frac{\boldsymbol{r}_0}{\|\boldsymbol{r}_0\|},\\
&\text{for } n = 1, 2, \ldots, m-1 \text{ do:}\\
&\quad h_{i,n} = (\boldsymbol{v}_i, A\boldsymbol{v}_n), \quad i = 1, 2, \ldots, n,\\
&\quad \tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \sum_{i=1}^{n} h_{i,n}\boldsymbol{v}_i, \qquad (2.9)\\
&\quad h_{n+1,n} = \|\tilde{\boldsymbol{v}}_{n+1}\|,\\
&\quad \boldsymbol{v}_{n+1} = \frac{\tilde{\boldsymbol{v}}_{n+1}}{h_{n+1,n}}.\\
&\text{end}
\end{aligned}
$$

It is readily shown that Algorithm 2.3 generates a vector sequence $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ such that

$$(\boldsymbol{v}_i, \boldsymbol{v}_j) = \delta_{ij},$$

where $K_m(A, \boldsymbol{r}_0) = \mathrm{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$, and δ_{ij} denotes the Kronecker delta.

To produce more accurate orthonormal basis, the following modified Gram-Schmidt (MGS) is often used:

Algorithm 2.4: The Arnoldi process with MGS

$$
\begin{aligned}
&\text{set } \boldsymbol{v}_1 = \frac{\boldsymbol{r}_0}{\|\boldsymbol{r}_0\|},\\
&\text{for } n = 1, 2, \ldots, m-1 \text{ do:}\\
&\quad \boldsymbol{t} = A\boldsymbol{v}_n,\\
&\quad \text{for } i = 1, 2, \ldots, n \text{ do:}\\
&\quad\quad h_{i,n} = (\boldsymbol{v}_i, \boldsymbol{t}),\\
&\quad\quad \boldsymbol{t} = \boldsymbol{t} - h_{i,n}\boldsymbol{v}_i,\\
&\quad \text{end}\\
&\quad h_{n+1,n} = \|\boldsymbol{t}\|,\\
&\quad \boldsymbol{v}_{n+1} = \frac{\boldsymbol{t}}{h_{n+1,n}}.\\
&\text{end}
\end{aligned}
$$

From the previous section, we can expect that the Arnoldi process with MGS is superior to the one with CGS. Actually, we can see from Fig. 2.2 that the Arnoldi process with MGS is the best of four orthogonalization processes, where we used the Hilbert matrix and chose $\boldsymbol{r}_0 = (1, \ldots, 1)^{\mathrm{T}}$ to run Algorithm 2.3 and Algorithm 2.4, we also applied CGS and MGS to the vector sequence $\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{m-1}\boldsymbol{r}_0$. If we need more accurate basis of the

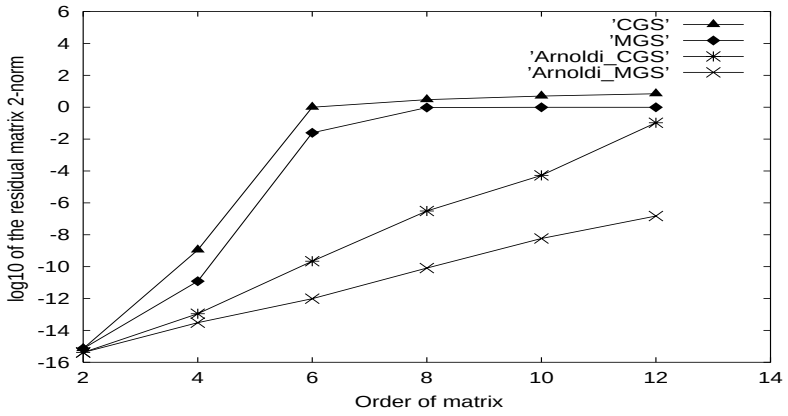

Figure 2.2: Comparison of CGS, MGS, Arnoldi with CGS, and Arnoldi with MGS.

Krylov subspace, we prefer to use the Arnoldi process with Householder reflection suggested by Walker [92] or the Arnoldi process with reorthogonalization technique, *e.g.*, [87, p.30] that is based on the Gram-Schmidt with reorthogonalization technique [20]; however, these algorithms cost about twice as much as the original one.

The Arnoldi process can be expressed in matrix form. This form will give us further understanding of the algorithm. Let V_n be the $N \times n$ matrix whose columns are the first n orthonormal system, *i.e.*,

$$V_n = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n], \quad V_n^{\mathrm{H}} V_n = I_n,$$

and $H_{n+1,n}$ be $(n+1)\times n$ Hessenberg matrix with entries $h_{i,j}=0$ for $i>j+1$, and H_n be $n\times n$ Hessenberg matrix:

$$H_{n+1,n}=\begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n-1} & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n-1} & h_{2,n} \\ & \ddots & \ddots & \vdots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & h_{n,n-1} & h_{n,n} \\ & & & & h_{n+1,n} \end{pmatrix}, H_n=\begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n-1} & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n-1} & h_{2,n} \\ & \ddots & \ddots & \vdots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & h_{n,n-1} & h_{n,n} \end{pmatrix}.$$

Then, from Algorithm 2.3, we have

$$AV_n = V_{n+1}H_{n+1,n} = V_nH_n + h_{n+1,n}\boldsymbol{v}_{n+1}\boldsymbol{e}_n^{\mathrm{T}},$$

where $\boldsymbol{e}_n=(0,0,\ldots,1)^{\mathrm{T}}\in\mathcal{R}^n$. From the above expression and $V_n^{\mathrm{H}}V_n=I_n$, we readily obtain the formula $H_n=V_n^{\mathrm{H}}AV_n$.

2.2.2 The bi-Lanczos process

In the previous section, we saw useful orthonormalization processes of $K_m(A,\boldsymbol{r}_0)$; however, their required computational costs and memory storage grow linearly. This fact can be a bottleneck for computing orthonormal basis of $K_m(A,\boldsymbol{r}_0)$ with a large number of dimensions. The bi-Lanczos process was designed for obtaining bi-orthogonal basis of $K_m(A,\boldsymbol{r}_0)$ with short-term recurrences. Although the process does not generate orthonormal basis vectors, it plays an important role in solving linear systems with low memory requirement. The algorithm is given below.

Algorithm 2.5: The bi-Lanczos process

choose $\boldsymbol{r}_0^*$ such that $(\boldsymbol{r}_0^*,\boldsymbol{r}_0)\neq 0$,
set $\beta_0=\gamma_0=0,\ \boldsymbol{v}_0=\boldsymbol{w}_0=\boldsymbol{0}$,
set $\boldsymbol{v}_1=\boldsymbol{r}_0/\|\boldsymbol{r}_0\|,\ \boldsymbol{w}_1=\boldsymbol{r}_0^*/(\boldsymbol{r}_0^*,\boldsymbol{v}_1)$,
for $n=1,2,\ldots,m-1$ do:
  $\alpha_n=(\boldsymbol{w}_n,A\boldsymbol{v}_n)$,
  $\tilde{\boldsymbol{v}}_{n+1}=A\boldsymbol{v}_n-\alpha_n\boldsymbol{v}_n-\beta_{n-1}\boldsymbol{v}_{n-1}$,
  $\tilde{\boldsymbol{w}}_{n+1}=A^{\mathrm{H}}\boldsymbol{w}_n-\bar{\alpha}_n\boldsymbol{w}_n-\gamma_{n-1}\boldsymbol{w}_{n-1}$,
  $\gamma_n=\|\tilde{\boldsymbol{v}}_{n+1}\|$,
  $\boldsymbol{v}_{n+1}=\tilde{\boldsymbol{v}}_{n+1}/\gamma_n$,
  $\beta_n=(\tilde{\boldsymbol{w}}_{n+1},\boldsymbol{v}_{n+1})$,
  $\boldsymbol{w}_{n+1}=\tilde{\boldsymbol{w}}_{n+1}/\bar{\beta}_n$.
end

If breakdown does not occur, the above algorithm generates bi-orthogonal basis of the two Krylov subspaces such that

$$(\boldsymbol{w}_i,\boldsymbol{v}_j)=\delta_{ij},$$

where $K_m(A, \boldsymbol{r}_0) = \mathrm{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ and $K_m(A^{\mathrm{H}}, \boldsymbol{r}_0^*) = \mathrm{span}\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m\}$.

Similar to the Arnoldi process, the bi-Lanczos process can be also written in matrix form. Let $T_{n+1,n}$ be the $(n+1) \times n$ tridiagonal matrix whose entries are recurrence coefficients of the bi-Lanczos process, *i.e.*,

$$T_{n+1,n} = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \gamma_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \gamma_{n-1} & \alpha_n \\ & & & \gamma_n \end{pmatrix}, \quad T_n = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \gamma_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \gamma_{n-1} & \alpha_n \end{pmatrix}.$$

Then from the bi-Lanczos process, we have

$$\begin{aligned} AV_n = V_{n+1}T_{n+1,n} &= V_nT_n + \gamma_n \boldsymbol{v}_{n+1}\boldsymbol{e}_n^{\mathrm{T}}, \\ A^{\mathrm{H}}W_n = W_{n+1}T_{n+1,n}^{\mathrm{H}} &= W_nT_n^{\mathrm{H}} + \bar{\beta}_n \boldsymbol{w}_{n+1}\boldsymbol{e}_n^{\mathrm{T}}. \end{aligned} \tag{2.10}$$

From the above expression and $W_n^{\mathrm{H}}V_n = I_n$, we readily obtain the formula $T_n = W_n^{\mathrm{H}}AV_n$.

2.2.3 The complex symmetric Lanczos process

The complex symmetric Lanczos process is a special case for the above bi-Lanczos process when the coefficient matrix is complex symmetric, *i.e.* $A = A^{\mathrm{T}} \neq A^{\mathrm{H}}$. If A is complex symmetric, we can readily derive the process from Algorithm 2.5 by setting $\boldsymbol{r}_0^* = \overline{\boldsymbol{r}}_0$.

Algorithm 2.6: The complex symmetric Lanczos process

set $\beta_0 = 0,\ \boldsymbol{v}_0 = \boldsymbol{0}$,
set $\boldsymbol{v}_1 = \boldsymbol{r}_0/(\bar{\boldsymbol{r}}_0, \boldsymbol{r}_0)^{1/2}$,
for $n = 1, 2, \ldots, m-1$ do:
  $\alpha_n = (\bar{\boldsymbol{v}}_n, A\boldsymbol{v}_n)$,
  $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n\boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$,
  $\beta_n = (\bar{\tilde{\boldsymbol{v}}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$,
  $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\beta_n$.
end

From the above algorithm, we see that it is very similar to the Lanczos process. If breakdown does not occur, the above algorithm generates conjugate orthogonal basis of the Krylov subspace such that

$$(\bar{\boldsymbol{v}}_i, \boldsymbol{v}_j) = \delta_{ij},$$

where $K_m(A, \boldsymbol{r}_0) = \mathrm{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$.

Similar to the Arnoldi process and the bi-Lanczos process, Algorithm 2.6 can be also written in matrix form. Let $T_{n+1,n}$ be the $(n+1) \times n$ tridiagonal matrix whose entries are

recurrence coefficients of the complex symmetric Lanczos process, *i.e.*,

$$T_{n+1,n} = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \\ & & & \beta_n \end{pmatrix}, \quad T_n = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix}.$$

Then from Algorithm 2.6, we have

$$AV_n = V_{n+1}T_{n+1,n} = V_nT_n + \beta_n \boldsymbol{v}_{n+1}\boldsymbol{e}_n^{\mathrm{T}}. \tag{2.11}$$

From the above, we see that T_n is also complex symmetric, *i.e.*, $T_n = T_n^{\mathrm{T}} \neq T_n^{\mathrm{H}}$.

2.2.4 The Lanczos process

The Lanczos process [56] is the Arnoldi process specialized to the case where A is Hermitian. Since H_n is both Hermitian and Hessenberg, it is tridiagonal. This means that in the inner loop of Arnoldi process (2.9), the summation from 1 to n can be replaced by $n-1$ to n. Therefore, instead of the $(n+1)$-term recurrence at step n, the Lanczos process only requires a three-term recurrence. As a result of this amazing property, each step of the Lanczos process is much cheaper than the corresponding step of the Arnoldi process or the bi-Lanczos process. The Lanczos process is written as follows:

Algorithm 2.7: The Lanczos process

set $\beta_0 = 0,\ \boldsymbol{v}_0 = \boldsymbol{0}$,
set $\boldsymbol{v}_1 = \boldsymbol{r}_0/(\boldsymbol{r}_0, \boldsymbol{r}_0)^{1/2}$,
for $n = 1, 2, \ldots, m-1$ do:
  $\alpha_n = (\boldsymbol{v}_n, A\boldsymbol{v}_n)$,
  $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n\boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$,
  $\beta_n = (\tilde{\boldsymbol{v}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$,
  $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\beta_n$.
end

We can readily derive the above algorithm from the Arnoldi process or the bi-Lanczos process by setting $\boldsymbol{r}_0^* = \boldsymbol{r}_0$. If breakdown does not occur, the above algorithm generates orthonormal basis of $K_n(A, \boldsymbol{r}_0)$ such that

$$(\boldsymbol{v}_i, \boldsymbol{v}_j) = \delta_{ij},$$

where $K_m(A, \boldsymbol{r}_0) = \mathrm{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$.

Similar to the Arnoldi process and the bi-Lanczos process, the Lanczos process can be also written in matrix form. Let $T_{n+1,n}$ be the $(n+1) \times n$ tridiagonal matrix whose entries

are recurrence coefficients of the Lanczos process, *i.e.*,

$$T_{n+1,n} = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \\ & & & \beta_n \end{pmatrix}, \quad T_n = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix}.$$

Then from the Lanczos process, we have

$$AV_n = V_{n+1}T_{n+1,n} = V_nT_n + \beta_n \boldsymbol{v}_{n+1}\boldsymbol{e}_n^{\mathrm{T}}. \tag{2.12}$$

From the above expression and $V_n^{\mathrm{H}}V_n = I_n$, we readily obtain the following formula:

$$T_n = V_n^{\mathrm{H}}AV_n. \tag{2.13}$$

We see that T_n is also Hermitian (more precisely, real symmetric). This process is widely used for solving Hermitian linear systems.

2.3 Preconditioners

The convergence rate of iterative methods depends strongly on spectral property of the coefficient matrix. It is therefore natural to try to transform the original system into one having the same solution but more favorable spectral properties. If K is a nonsingular matrix, the transformed linear system

$$K^{-1}A\boldsymbol{x} = K^{-1}\boldsymbol{b} \tag{2.14}$$

has the same solution as the original one. Hence, we can obtain the solution by applying KS methods to the above system. The matrix K is called preconditioner and a good preconditioner satisfies the following properties:

- K is close to A. More precisely, $K^{-1}A$ has a low degree minimal polynomial;
- $K^{-1}\boldsymbol{z}$ is readily obtained for any vector $\boldsymbol{z}$.

Left and Right preconditioning

If A is Hermitian positive definite (HPD), the transformed system (2.14) is not useful in practice because of the fact that the matrix $K^{-1}A$ is not HPD any longer. Since the CG method is an iterative method for HPD, an alternative way is to split the preconditioner as $K = K_1K_1^{\mathrm{H}}$ and to transform the system as

$$K_1^{-1}AK_1^{-\mathrm{H}}(K_1^{\mathrm{H}}\boldsymbol{x}) = K_1^{-1}\boldsymbol{b}.$$

The above coefficient matrix $K_1^{-1}AK_1^{-\mathrm{H}}$ is obviously HPD. Hence, the CG method can be applied. If the coefficient matrix A is non-Hermitian, the corresponding preconditioning is

$$K_1^{-1}AK_2^{-1}(K_2\boldsymbol{x}) = K_1^{-1}\boldsymbol{b}, \tag{2.15}$$

where $K = K_1 K_2$. The above form is called *left and right* preconditioning. We will adopt the form (2.15) when we use preconditioners. If we choose $K_1 = I$, then we have

$$AK^{-1}(K\boldsymbol{x}) = \boldsymbol{b}. \tag{2.16}$$

This preconditioing is referred to as right preconditioning. The right preconditioning is used for GMRES and GCR described in Appendix B.

Here is one note that if K^{-1} and A are HPD, then CG can be applied to (2.14). Then, since $K^{-1}A$ is HPD with respect to the inner product $(\boldsymbol{p}, \boldsymbol{q})_K := (\boldsymbol{p}, K\boldsymbol{q})$, the resulting algorithm still minimizes A-norm of the error over $\boldsymbol{x}_0 + K_n(K^{-1}A, K^{-1}\boldsymbol{r}_0)$. Moreover, even if K^{-1} is Hermitian but not positive definite, CG can be also applied. However, in this case, CG does not minimize anything since $(\boldsymbol{p}, \boldsymbol{q})_K$ is not a proper inner product.

2.3.1 Incomplete LU factorization

Many preconditioners have been proposed in the last few decades of the 20th century. Of various preconditioners, most well-known ones fall in a category of incomplete factorizations of the coefficient matrix. Incomplete factorizations can be given in the form of $K = LU$ (with nonsingular triangular matrices L and U).

One of the simplest incomplete factorizations is the D-ILU preconditioner that is proposed by Pommerell [65]. The idea of this method is given as follows: first, split the coefficient matrix into its diagonal, strictly lower triangular, and strictly upper triangular parts as $A = D_A + L_A + U_A$; second, use $K = (D + L_A)D^{-1}(D + U_A)$ as a preconditioner, where D is determined by diag(K)=D_A. Hence, only the computation of D is required. The algorithm of D-ILU is described below.

Algorithm 2.8: D-ILU storing the inverses of the pivots:

let S be the nonzero set $\{(i,j)|a_{i,j} \neq 0\}$,
for $i = 1, 2, \ldots, n$ do:
  $d_{i,i} = a_{i,i}$,
end
for $i = 1, 2, \ldots, n$ do:
  $d_{i,i} = 1/d_{i,i}$,
  for $j = i+1, i+2, \ldots, n$ do:
    if $(i,j) \in S$ and $(j,i) \in S$ then
    $d_{j,j} = d_{j,j} - a_{j,i} d_{i,i} a_{i,j}$.
  end
end

The lower and upper matrices of the preconditioner have only nonzero element in the set S, but this fact is not true in general for the preconditioner K itself, *i.e.*, K is usually not sparse but dense. Hence, this preconditioner is readily available in practice.

Since the D-ILU preconditioner explicitly contains the off-diagonal parts of the original matrix, Eisenstat's trick [30] can be used to give a more efficient implementation of the preconditioned CG method.

One of the most well-known and powerful incomplete LU factorizations is given by Meijerink and van der Vorst [59]. This factorization is referred to as $ILU(0)$. The algorithm

is given below. S denotes an index set such that $a_{i,j} \neq 0$, *i.e.*, $S = \{(i,j)|a_{i,j} \neq 0\}$.

Algorithm 2.9: $ILU(0)$

for $i = 2, \ldots, n$ do:
  for $k = 1, \ldots, i-1$ and for $(i,k) \in S$ do:
    $a_{i,k} = a_{i,k}/a_{k,k}$,
    for $j = k+1, \ldots, n$ and for $(i,j) \in S$ do:
      $a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$.
    end
  end
end

By the above factorization, the number of nonzeros in the factorized matrix is the same as that in the original matrix. Hence, if $ILU(0)$ is applied to a sparse matrix, then it also generates a sparse preconditioning matrix. This fact is important for saving memory. The relationship between $ILU(0)$ and D-ILU is given below.

Theorem 2.3.1 *$ILU(0)$ and D-ILU generate the same preconditioning matrix if the matrix graph contains no triangles.*

To generate a better preconditioning matrix, $ILU(p)$ [59] is also widely used at the cost of memory, where p denotes the level of fill-in. The definition of the level is given as follows: original entries have level zero, original zeros have level ∞, and a fill-in in position (i,j) has level defined by

$$\text{Level}_{ij} = \min_{1 \leq k \leq \min\{i,j\}} \{\text{Level}_{ik} + \text{Level}_{kj} + 1\}. \tag{2.17}$$

The algorithm is shown in [70, p.300].

Another successful variant of ILU is an ILU with threshold approach proposed by Saad [69]. This factorization is referred to as $ILUT(p,\tau)$, where p is used for saving memory and τ is a criterion for dropping elements.

2.4 Krylov subspace methods for Hermitian linear systems

In this section, we give brief derivations of the algorithms of CG, CR, and MINRES. CG is widely used for solving Hermitian positive definite linear systems. On the other hand, CR and MINRES are used for solving Hermitian indefinite linear systems. Although MINRES and CR are mathematically equivalent, MINRES is numerically robuster than CR, and CR is easier to be implemented than MINRES.

2.4.1 The CG method

In this subsection, based on the Lanczos process, let us consider a derivation of the CG method from the Krylov subspace condition (2.1) and Ritz-Galerkin approach (2.3). First, substituting the Lanczos basis into the Krylov subspace condition, we have

$$\boldsymbol{z}_n = V_n \boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathcal{C}^n.$$

From (2.2), it follows that

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n\boldsymbol{y}_n. \tag{2.18}$$

Since $V_n^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0}$ by Ritz-Galerkin approach (2.3), it follows from (2.13) that we obtain

$$\boldsymbol{y}_n = \|\boldsymbol{r}_0\|T_n^{-1}\boldsymbol{e}_1,$$

where $\boldsymbol{e}_1 = (1, 0, \ldots, 0)^{\mathrm{T}} \in \mathcal{C}^n$. Then, from (2.12), the residual vector can be written by

$$\begin{aligned} \boldsymbol{r}_n &= \boldsymbol{r}_0 - (V_nT_n + \beta_n\boldsymbol{v}_{n+1}\boldsymbol{e}_n^{\mathrm{T}})\boldsymbol{y}_n \\ &= \boldsymbol{r}_0 - V_nT_nT_n^{-1}\|\boldsymbol{r}_0\|\boldsymbol{e}_1 - \beta_n\boldsymbol{v}_{n+1}\boldsymbol{e}_n^{\mathrm{T}}\boldsymbol{y}_n \\ &= -\beta_n(\boldsymbol{y}_n, \boldsymbol{e}_n)\boldsymbol{v}_{n+1}. \end{aligned} \tag{2.19}$$

It is clear from the above that the residual vectors are orthogonal,

$$(\boldsymbol{r}_i, \boldsymbol{r}_j) = 0 \quad \text{for} \quad i \neq j. \tag{2.20}$$

Substituting (2.19) for $\tilde{\boldsymbol{v}}_n$ of Algorithm 2.7, we obtain the algorithm of the CG method. For details of the derivation, see Appendix B.

Algorithm 2.10: CG method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\beta_{-1} = 0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \leq \epsilon\|\boldsymbol{b}\|$ do:
$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$,
$\alpha_n = \dfrac{(\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{p}_n, A\boldsymbol{p}_n)}$,
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$,
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_nA\boldsymbol{p}_n$,
$\beta_n = \dfrac{(\boldsymbol{r}_{n+1}, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n, \boldsymbol{r}_n)}$.
end

From (2.20), the nth residual vector $\boldsymbol{r}_n$ should be zero in at most N steps. The convergence of CG is theoretically well known. See, e, g, [48]. We also discuss the convergence rate of CG later.

Looking at Algorithm 2.10, we see that the residual vector $\boldsymbol{r}_n$ and the auxiliary vector $\boldsymbol{p}_n$ can be expressed by using two polynomials R_n and P_n.

$$\boldsymbol{r}_n = R_n(A)\boldsymbol{r}_0, \quad \boldsymbol{p}_n = P_n(A)\boldsymbol{r}_0,$$

where $R_n(\lambda)$ is called the Lanczos polynomial [56] which satisfies the following three-term recurrences:

$$R_0(\lambda) = 1, \tag{2.21}$$

$$R_1(\lambda) = (1 - \alpha_0\lambda)R_0(\lambda), \tag{2.22}$$

$$\begin{aligned} R_n(\lambda) &= (1 + \frac{\beta_{n-2}}{\alpha_{n-2}}\alpha_{n-1} - \alpha_{n-1}\lambda)R_{n-1}(\lambda) \\ &\quad -\frac{\beta_{n-2}}{\alpha_{n-2}}\alpha_{n-1}R_{n-2}(\lambda), \quad n = 2, 3, \ldots \end{aligned} \tag{2.23}$$

and R_n, P_n are written by the following two-term recurrences:

$$R_0(\lambda) = 1, \quad P_0(\lambda) = 1, \tag{2.24}$$

$$R_n(\lambda) = R_{n-1}(\lambda) - \alpha_{n-1}\lambda P_{n-1}(\lambda), \tag{2.25}$$

$$P_n(\lambda) = R_n(\lambda) + \beta_{n-1}P_{n-1}(\lambda), \quad n = 1, 2, \ldots \tag{2.26}$$

Note that there are other approaches to derive the CG method from the following minimizations:

$$\begin{aligned} \boldsymbol{r}_n &= \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)} \|\boldsymbol{b} - A\boldsymbol{x}_n\|_{A^{-1}} \\ &= \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)} \|\boldsymbol{e}_n\|_A, \end{aligned} \tag{2.27}$$

where $\boldsymbol{e}_n$ denotes the error $\boldsymbol{e}_n = \boldsymbol{x} - \boldsymbol{x}_n$. For details, see [40].

The convergence rate of the CG method is estimated by the following theorem:

Theorem 2.4.1 *Let $\boldsymbol{e}_n$ be the error at step n of the CG method applied to the Hermitian positive definite system $A\boldsymbol{x} = \boldsymbol{b}$. Then, the A-norm of the nth error satisfies*

$$\|\boldsymbol{e}_n\|_A \leq 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^n \|\boldsymbol{e}_0\|_A,$$

where κ denotes the 2-norm condition number of A.

Proof. See [49, pp. 51-52]. □

From the above theorem, we see that the condition number of A depends directly on the rate of the convergence. Moreover, we can see that

$$\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \sim 1 - \frac{2}{\sqrt{\kappa}}$$

as $\kappa \to \infty$, and it implies that if κ is large, the convergence to a specified tolerance can be expected in $\mathcal{O}(\sqrt{\kappa})$ iterations. Note that this is only an upper bound. The convergence may be faster for special right-hand sides or if the spectrum is clustered.

2.4.2 The CR method

In this subsection, we derive the Conjugate Residual (CR) method from the CG method obtained in the previous section. Let A be Hermitian positive definite, then there exists a square root of A: the Cholesky factorization of A is obtained as $L^{\mathrm{H}}L$ and L can be decomposed as $U\Sigma V^{\mathrm{H}}$ by using singular value decomposition. Then, we have

$$A = L^{\mathrm{H}}L = (V\Sigma U^{\mathrm{H}})(U\Sigma V^{\mathrm{H}}) = V\Sigma^2 V^{\mathrm{H}} = (V\Sigma V^{\mathrm{H}})(V\Sigma V^{\mathrm{H}}) = A^{\frac{1}{2}}A^{\frac{1}{2}}.$$

Thus, the square root of A is $A^{\frac{1}{2}} = V\Sigma V^{\mathrm{H}}$. Here, we show that if A is Hermitian positive definite, then the algorithm of CR is obtained by applying CG to the following systems:

$$A\tilde{\boldsymbol{x}} = A^{\frac{1}{2}}\boldsymbol{b}, \quad \tilde{\boldsymbol{x}} = A^{\frac{1}{2}}\boldsymbol{x}. \tag{2.28}$$

It follows from the algorithm of CG that

$$\begin{aligned}
\tilde{\boldsymbol{p}}_n &= \tilde{\boldsymbol{r}}_n + \beta_{n-1}\tilde{\boldsymbol{p}}_{n-1},\\
\alpha_n &= \frac{(\tilde{\boldsymbol{r}}_n, \tilde{\boldsymbol{r}}_n)}{(\tilde{\boldsymbol{p}}_n, A\tilde{\boldsymbol{p}}_n)},\\
\tilde{\boldsymbol{x}}_{n+1} &= \tilde{\boldsymbol{x}}_n + \alpha_n\tilde{\boldsymbol{p}}_n,\\
\tilde{\boldsymbol{r}}_{n+1} &= \tilde{\boldsymbol{r}}_n - \alpha_n A\tilde{\boldsymbol{p}}_n,\\
\beta_n &= \frac{(\tilde{\boldsymbol{r}}_{n+1}, \tilde{\boldsymbol{r}}_{n+1})}{(\tilde{\boldsymbol{r}}_n, \tilde{\boldsymbol{r}}_n)}.
\end{aligned}$$

The residual vector of the original system $A\boldsymbol{x} = \boldsymbol{b}$ is $\boldsymbol{r}_n = \boldsymbol{b} - A\boldsymbol{x}_n$, and then from (2.28) we have $\tilde{\boldsymbol{r}}_n = A^{\frac{1}{2}}\boldsymbol{b} - A\tilde{\boldsymbol{x}}_n = A^{\frac{1}{2}}(\boldsymbol{b} - A\boldsymbol{x}_n) = A^{\frac{1}{2}}\boldsymbol{r}_n$. Substituting $\tilde{\boldsymbol{r}}_n = A^{\frac{1}{2}}\boldsymbol{r}_n, \tilde{\boldsymbol{p}}_n = A^{\frac{1}{2}}\boldsymbol{p}_n$, and $\tilde{\boldsymbol{x}}_n = A^{\frac{1}{2}}\boldsymbol{x}_n$ into the above recurrences, we have the algorithm of CR.

Algorithm 2.11: CR method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \beta_{-1} = 0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:

$$\begin{aligned}
&\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},\\
&(A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1},)\\
&\alpha_n = \frac{(\boldsymbol{r}_n, A\boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)},\\
&\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,\\
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,\\
&\beta_n = \frac{(\boldsymbol{r}_{n+1}, A\boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n, A\boldsymbol{r}_n)}.
\end{aligned}$$

end

By induction, it can be shown that the CR method generates iterates $\boldsymbol{r}_i, \boldsymbol{p}_i$ that satisfy

$$(\boldsymbol{r}_i, A\boldsymbol{r}_j) = 0 \quad \text{for} \quad i \neq j, \tag{2.29}$$

$$(A\boldsymbol{p}_i, A\boldsymbol{p}_j) = 0 \quad \text{for} \quad i \neq j. \tag{2.30}$$

This leads us to the fact that CR generates the nth approximate solution $\boldsymbol{x}_n$ such that the 2-norm of the corresponding residual vector is minimized over the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$, *i.e.*,

$$\|\boldsymbol{r}_n\| = \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)} \|\boldsymbol{b} - A\boldsymbol{x}_n\|.$$

As an equivalent approach, let us consider applying CGNR that is discussed in Appendix B (Algorithm B.1) to the following system:

$$A^{\frac{1}{2}}\tilde{\boldsymbol{x}} = \boldsymbol{b}, \quad \tilde{\boldsymbol{x}} = A^{\frac{1}{2}}\boldsymbol{x}.$$

Then, the CR method is obtained.

2.4.3 The MINRES method

The Minimal Residual (MINRES) method [62] is a useful algorithm for solving Hermitian indefinite linear systems. This method generates $\boldsymbol{x}_n$ that minimizes $\|\boldsymbol{b} - A\boldsymbol{x}_n\|$ over the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$. This can be achieved by using the Lanczos process. Here, we give the derivation process of the MINRES method. Let V_n be the orthonormal basis of $K_n(A, \boldsymbol{r}_0)$. Then, since $\boldsymbol{x}_n$ lies in the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$, we have

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n \boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathcal{C}^n. \tag{2.31}$$

The corresponding residual vector is written as

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n \boldsymbol{y}_n.$$

From the matrix form of the Lanczos process, it follows that

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - V_{n+1} T_{n+1,n} \boldsymbol{y}_n = V_{n+1}(\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n),$$

where $\beta := \|\boldsymbol{r}_0\|$. Hence, the 2-norm of the residual vector can be minimized by choosing $\boldsymbol{y}_n$ such that

$$\boldsymbol{y}_n := \arg \min_{\boldsymbol{y} \in \mathcal{C}^n} \|\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}\|$$

By solving the above minimization problem using Givens rotations, we obtain the MINRES method. For details of the above solution, see Appendix B.

Algorithm 2.12: MINRES method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
set $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \ldots, 0)^{\mathrm{T}}$, $\boldsymbol{v}_1 = \boldsymbol{r}_0 / \|\boldsymbol{r}_0\|$,
for $n = 1, 2, \ldots$ do:
(Lanczos process)
$\alpha_n = (\boldsymbol{v}_n, A\boldsymbol{v}_n)$,
$\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n \boldsymbol{v}_n - \beta_{n-1} \boldsymbol{v}_{n-1}$,
$\beta_n = (\tilde{\boldsymbol{v}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$,
$\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1} / \beta_n$,
set $t_{n-1,n} = \beta_{n-1}$, $t_{n,n} = \alpha_n$, $t_{n+1,n} = \beta_n$,
(Givens rotations)
for $i = \max\{1, n-2\}, \ldots, n-1$ do:

$$\begin{pmatrix} t_{i,n} \\ t_{i+1,n} \end{pmatrix} = \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} t_{i,n} \\ t_{i+1,n} \end{pmatrix},$$

end

$$c_n = \frac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}},$$

$$\bar{s}_n = \frac{t_{n+1,n}}{t_{n,n}} c_n,$$

$$t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n},$$

$$t_{n+1,n} = 0,$$

$$\begin{pmatrix} g_n \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{pmatrix} \begin{pmatrix} g_n \\ 0 \end{pmatrix},$$

(Update $\boldsymbol{x}_n$)
$\boldsymbol{p}_n = (\boldsymbol{v}_n - t_{n-2,n} \boldsymbol{p}_{n-2} - t_{n-1,n} \boldsymbol{p}_{n-1}) / t_{n,n}$,
$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + g_n \boldsymbol{p}_n$,
(Check convergence)
if $|g_{n+1}| / \|\boldsymbol{b}\| \leq \epsilon$, then stop.
end

From the derivation of MINRES, it is clear that MINRES generates the nth approximate solution $\boldsymbol{x}_n$ such that the 2-norm of the corresponding residual vector is minimized over the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$, *i.e.*,

$$\|\boldsymbol{r}_n\| = \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)} \|\boldsymbol{b} - A\boldsymbol{x}_n\|.$$

Hence, in exact precision arithmetic MINRES and CR generate the same approximate solutions.

2.5 Krylov subspace methods for complex symmetric linear systems

In this section, we derive the algorithms of COCG and QMR_SYM. COCG is a well-known algorithm for solving complex symmetric linear systems. QMR_SYM is also a useful solver and it often gives smoother convergence behavior than COCG. The remarkable property of these methods is that they require only one matrix-vector multiplication per iteration.

2.5.1 The COCG method

In this subsection, we discuss a way to obtain COCG. Let $\boldsymbol{x}_n$ be the nth approximate solution. Then, the corresponding nth residual vector $\boldsymbol{r}_n(= \boldsymbol{b} - A\boldsymbol{x}_n)$ and search direction $\boldsymbol{p}_n$ are given by the following coupled two-term recurrences:

$$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \quad \boldsymbol{p}_0 = \boldsymbol{r}_0, \tag{2.32}$$

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \alpha_{n-1} A\boldsymbol{p}_{n-1}, \tag{2.33}$$

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \quad n = 1, 2, \ldots \tag{2.34}$$

The computational formulas of α_{n-1} and β_{n-1} in the recurrences (2.33)-(2.34) are determined by the following orthogonality conditions:

$$\boldsymbol{r}_n \perp K_n(\bar{A}, \bar{\boldsymbol{r}}_0) \quad \text{and} \quad A\boldsymbol{p}_n \perp K_n(\bar{A}, \bar{\boldsymbol{r}}_0). \tag{2.35}$$

Then, we obtain the COCG method. For details of the derivation, see Appendix B.

Algorithm 2.13: COCG method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
set $\boldsymbol{p}_{-1} = \boldsymbol{0}$, $\beta_{-1} = 0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \leq \epsilon\|\boldsymbol{b}\|$ do:
$\quad \boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$,
$\quad \alpha_n = \dfrac{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}{(\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n)}$,
$\quad \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$,
$\quad \boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$,
$\quad \beta_n = \dfrac{(\bar{\boldsymbol{r}}_{n+1}, \boldsymbol{r}_{n+1})}{(\bar{\boldsymbol{r}}_n, \boldsymbol{r}_n)}$.
end

If breakdown does not occur, COCG residuals satisfy $(\overline{\boldsymbol{r}}_i, \boldsymbol{r}_j) = 0$ for $i \neq j$. From the above algorithm, we see that if the coefficient matrix is real symmetric, then COCG is equivalent to CG.

2.5.2 The QMR_SYM method

The QMR_SYM method [38] (named by van der Vorst [87, p.112]) is also useful for solving complex symmetric linear systems, and it can be derived from the complex symmetric Lanczos process described in section 2.2.3. Here, we give the derivation process of the

QMR_SYM method. Let V_n be the conjugate orthonormal basis of $K_n(A, \boldsymbol{r}_0)$. Then, since $\boldsymbol{x}_n$ lies in the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$, we have

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n \boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathcal{C}^n. \tag{2.36}$$

The corresponding residual vector is $\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n\boldsymbol{y}_n$. From the complex symmetric Lanczos process, it follows that $\boldsymbol{r}_n = \boldsymbol{r}_0 - V_{n+1}T_{n+1,n}\boldsymbol{y}_n = V_{n+1}(\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}_n)$ with $\beta := \|\boldsymbol{r}_0\|$. We see that the above derivation process is very similar to the one of MINRES; however, in this case if we choose $\boldsymbol{y}_n$ such that the norm of the residual is minimized, then it may lead to large amount of computational costs. Instead, QMR_SYM chooses $\boldsymbol{y}_n$ such that

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y} \in \mathcal{C}^n} \|\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\|. \tag{2.37}$$

Here, we note that if V_n satisfies the relation $V_n^{\mathrm{H}} V_n = I_n$, then the above choice leads to minimization of the norm of the residuals. Unfortunately, when matrix A is complex symmetric, V_n does not satisfy the relation $V_n^{\mathrm{H}} V_n = I_n$ except some special cases. One of the special cases is given in [38], *i.e.*, matrix A has the form

$$A = B + i\sigma I, \quad B = B^{\mathrm{T}} \in \mathcal{R}^{N \times N},\ \sigma \in \mathcal{R}.$$

To achieve the minimization (2.37), Givens rotations described in Appendix B (MINRES) play an important role. Multiplying $\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}$ by $Q_n = G_n \cdots G_1$ such that $Q_n^{\mathrm{H}} Q_n = I_n$ and $Q_n T_{n+1,n} = R_n$, we have

$$\min_{\boldsymbol{y} \in \mathcal{C}^n} \|\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\| = \min_{\boldsymbol{y} \in \mathcal{C}^n} \|\boldsymbol{g}_n - R_n \boldsymbol{y}\|, \text{ where } \boldsymbol{g}_n = \beta Q_n \boldsymbol{e}_1.$$

Thus, we have $\boldsymbol{y}_n = R_n^{-1}\boldsymbol{g}_n$. From which and (2.36), we obtain the following approximate solution $\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n R_n^{-1}\boldsymbol{g}_n$. Similar to MINRES, introducing $P_n := V_n R_n^{-1}$ gives a three-term recurrence relation. Then, we have more practical formula of $\boldsymbol{x}_n$:

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + g_i^{(i)} \boldsymbol{p}_i, \quad i = 1, \ldots, n.$$

The complete algorithm of QMR_SYM is described as follows:

Algorithm 2.14: QMR_SYM method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
set $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \ldots, 0)^{\mathrm{T}}$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$,
for $n = 1, 2, \ldots$ do:
(Complex symmetric Lanczos process)
$\alpha_n = (\overline{\boldsymbol{v}}_n, A\boldsymbol{v}_n)$,
$\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n \boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$,
$\beta_n = (\overline{\tilde{\boldsymbol{v}}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$,
$\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\beta_n$,
set $t_{n-1,n} = \beta_{n-1},\ t_{n,n} = \alpha_n,\ t_{n+1,n} = \beta_n$,
(Givens rotations)
for $i = \max\{1, n-2\}, \ldots, n-1$ do:

$$\begin{pmatrix} t_{i,n} \\ t_{i+1,n} \end{pmatrix} = \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} t_{i,n} \\ t_{i+1,n} \end{pmatrix},$$

end

$$c_n = \frac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}},$$

$$\bar{s}_n = \frac{t_{n+1,n}}{t_{n,n}} c_n,$$

$$t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n},$$

$$t_{n+1,n} = 0,$$

$$\begin{pmatrix} g_n \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{pmatrix} \begin{pmatrix} g_n \\ 0 \end{pmatrix},$$

(Update $\boldsymbol{x}_n$)

$\boldsymbol{p}_n = (\boldsymbol{v}_n - t_{n-2,n}\boldsymbol{p}_{n-2} - t_{n-1,n}\boldsymbol{p}_{n-1})/t_{n,n}$,

$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + g_n\boldsymbol{p}_n$,

(Check convergence)

$A\boldsymbol{p}_n = (A\boldsymbol{v}_n - t_{n-2,n}A\boldsymbol{p}_{n-2}$
$-t_{n-1,n}A\boldsymbol{p}_{n-1})/t_{n,n}$,

$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - g_nA\boldsymbol{p}_n$,

if $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq \epsilon$, then stop.

end

2.6 Krylov subspace methods for non-Hermitian linear systems

In this section, we derive Bi-CG and QMR from the bi-Lanczos process and Petrov-Galerkin approach.

2.6.1 The QMR method

In this subsection, we describe the idea of Quasi-Minimal Residual (QMR) method [41]. The derivation process of QMR is almost the same as that of QMR_SYM. The main difference is not to use the complex symmetric Lanczos process but the bi-Lanczos process. Let V_n be the bi-orthogonal basis of $K_n(A, \boldsymbol{r}_0)$ via the bi-Lanczos process given in Algorithm 2.5. Then, since $\boldsymbol{x}_n$ lies in the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$, we have

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n\boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathcal{C}^n.$$

Then, the corresponding residual vector is obtained below.

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n\boldsymbol{y}_n.$$

From the matrix form of the bi-Lanczos process (2.10), it follows that

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - V_{n+1}T_{n+1,n}\boldsymbol{y}_n = V_{n+1}(\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}_n),$$

where $\beta := \|\boldsymbol{r}_0\|$. From here we see that the above derivation process is very similar to the one of MINRES; however, in this case, if we choose $\boldsymbol{y}_n$ such that the norm of the residual is minimized, then it leads to large amount of computational costs. Instead, QMR chooses $\boldsymbol{y}_n$ such that

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y}\in\mathcal{C}^n} \|\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\|. \tag{2.38}$$

We call $\boldsymbol{y}_n$ quasi-residual vector. This can be solved by Givens rotations. The algorithm of QMR is written as follows:

Algorithm 2.15: QMR method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,

set $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \ldots, 0)^{\mathrm{T}}$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$,

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

(Bi-Lanczos process)

$\alpha_n = (\boldsymbol{w}_n, A\boldsymbol{v}_n)$,

$\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n\boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$,

$\tilde{\boldsymbol{w}}_{n+1} = A^{\mathrm{H}}\boldsymbol{w}_n - \bar{\alpha}_n\boldsymbol{w}_n - \gamma_{n-1}\boldsymbol{w}_{n-1}$,

$\gamma_n = \|\tilde{\boldsymbol{v}}_{n+1}\|$,

$\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\gamma_n$,

$\beta_n = (\tilde{\boldsymbol{w}}_{n+1}, \boldsymbol{v}_{n+1})$,
$\boldsymbol{w}_{n+1} = \bar{\boldsymbol{w}}_{n+1}/\bar{\beta}_n$,
(Givens rotations)
for $i = \max\{1, n-2\}, \ldots, n-1$ do:
$\begin{pmatrix} t_{i,n} \\ t_{i+1,n} \end{pmatrix} = \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} t_{i,n} \\ t_{i+1,n}, \end{pmatrix}$
end
$c_n = \dfrac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}}$,
$\bar{s}_n = \dfrac{t_{n+1,n}}{t_{n,n}} c_n$,
$t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n}$,
$t_{n+1,n} = 0$,
$\begin{pmatrix} g_n \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{pmatrix} \begin{pmatrix} g_n \\ 0 \end{pmatrix}$,
(Update $\boldsymbol{x}_n$)
$\boldsymbol{p}_n = (\boldsymbol{v}_n - t_{n-2,n}\boldsymbol{p}_{n-2} - t_{n-1,n}\boldsymbol{p}_{n-1})/t_{n,n}$,
$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + g_n \boldsymbol{p}_n$,
(Check convergence)
if $|g_{n+1}|/\|\boldsymbol{b}\| \le \epsilon$, then stop.
end

The above stopping criterion is clearly not equivalent to the others that we have seen before because of the relation $\|\boldsymbol{r}_n\| \le \|V_{n+1}\| \cdot |g_n| = \sqrt{n+1} \cdot |g_n|$. However, in many cases, $|g_n|$ and $\|\boldsymbol{r}_n\|$ are of the same order of magnitude. If we need to adopt the conventional stopping criterion, use the following recurrences:

(Check convergence)
$A\boldsymbol{p}_n = (A\boldsymbol{v}_n - t_{n-2,n}A\boldsymbol{p}_{n-2} - t_{n-1,n}A\boldsymbol{p}_{n-1})/t_{n,n}$,
$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - g_n A\boldsymbol{p}_n$,
if $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \le \epsilon$, then stop.

This leads to additional computational costs and memory.

Finally, here is one note that QMR has the possibility of breakdown, division by zero, underling the bi-Lanczos process. In finite precision arithmetic, such breakdown is very rare; however, near breakdown may occur, and this causes numerical instability. Hence, to avoid such problem, QMR in [41] uses the look-ahead Lanczos process. The look-ahead Lanczos process was first proposed by Taylor [83] and Parlett *et al*. [64].

2.6.2 The Bi-CG method

First of all, let us consider an ideal Krylov subspace method for solving non-Hermitian linear systems. An ideal method such as the CG method has two characteristic properties:

- It is based on Ritz-Galerkin approach (2.3) or minimal residual approach (2.5);
- Residual vectors can be updated by short-term recurrences.

Without some exceptions, if A is non-Hermitian, it is shown by Faber and Manteuffel that ideal iterative methods with both of the above two properties can not be obtained. See [33, 34].

Theorem 2.6.1 *Ideal iterative methods can be obtained if the coefficient matrix A has the following ideal structure:*

$$A = \alpha T + \beta I,$$

where $T = T^H$, $\alpha, \beta \in \mathcal{C}$, and I denotes a unit matrix.

The above structure of the matrix for ideal iterative methods has been extended to the following structure [2]:

$$A = \alpha T + \beta I,$$

where $\alpha, \beta \in \mathcal{C}$ and B is Hermitian such that $TB = (TB)^{\mathrm{H}}$. Due to the above reason, iterative methods for solving non-Hermitian linear systems are designed by adopting one of the two ideal properties, and the other one has to be an alternative. Now, we introduce an iterative method based on short recurrences and Petrov-Galerkin condition instead of Ritz-Galerkin condition.

The Conjugate Gradient method is not suitable for non-Hermitian linear systems because the residual vectors can not be made orthogonal by short-term recurrences. Therefore, the Bi-CG method takes another approach. It is based on Krylov subspace condition (2.1), three-term recurrences, and the Petrov-Galerkin approach (2.4). For details of the derivation, see Appendix B. We give the algorithm of Bi-CG below.

Algorithm 2.16: Bi-CG method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \beta_{-1} = 0,$
$\boldsymbol{r}_0^*$ is an arbitrary vector, such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0,\ e.g.,\ \boldsymbol{r}_0^* = \boldsymbol{r}_0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \leq \epsilon \|\boldsymbol{b}\|$ do:

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \quad \boldsymbol{p}_n^* = \boldsymbol{r}_n^* + \bar{\beta}_{n-1}\boldsymbol{p}_{n-1}^*,$$
$$\alpha_n = \frac{(\boldsymbol{r}_n^*, \boldsymbol{r}_n)}{(\boldsymbol{p}_n^*, A\boldsymbol{p}_n)},$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n,$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n, \quad \boldsymbol{r}_{n+1}^* = \boldsymbol{r}_n^* - \bar{\alpha}_n A^{\mathrm{H}}\boldsymbol{p}_n^*,$$
$$\beta_n = \frac{(\boldsymbol{r}_{n+1}^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n^*, \boldsymbol{r}_n)}.$$

end

If breakdown does not occur, residual vector sequences $\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_n$ and $\boldsymbol{r}_0^*, \boldsymbol{r}_1^*, \ldots, \boldsymbol{r}_n^*$ satisfy the bi-orthogonality property:

$$(\boldsymbol{r}_i^*, \boldsymbol{r}_j) = 0 \quad \text{for} \quad i \neq j. \tag{2.39}$$

The Bi-CG method can be described in another form by using two polynomials R_n and P_n which satisfy three-term recurrences (2.21)-(2.23) and two-term recurrences (2.24)-(2.26).

$$\boldsymbol{r}_n = R_n(A)\boldsymbol{r}_0, \qquad \boldsymbol{p}_n = P_n(A)\boldsymbol{r}_0,$$
$$\boldsymbol{r}_n^* = \bar{R}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, \quad \boldsymbol{p}_n^* = \bar{P}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*.$$

Since the Bi-CG algorithm has possibility of breakdown, QMR with look-ahead strategy is preferred to use in practice. QMR is better than Bi-CR in that QMR can avoid most of breakdowns and that the residual 2-norm of QMR is often less than that of Bi-CG at the same iteration step. In exact arithmetic, the Bi-CG residual $\boldsymbol{r}_n$ and the quasi residual of QMR (2.38) are related as

$$\|\boldsymbol{r}_n\| = \frac{\|\boldsymbol{y}_n\|}{\sqrt{1 - (\|\boldsymbol{y}_n\|/\|\boldsymbol{y}_{n-1}\|)^2}}.$$

This result is given by Cullum and Greenbaum [17].

If the breakdown does not occur, it follows from (2.39) that the Bi-CG method converges in at most N iteration steps. The convergence has been proved under some conditions [4, 41]. The Bi-CG method has two shortcomings. One is that the Bi-CG method needs to compute the product of A^{H} and vectors. The other one is that $\boldsymbol{r}_n^*$ is not directly used while $\boldsymbol{r}_n^*$ converges zero as well as $\boldsymbol{r}_n$. Since the Bi-CG method may lead to a rather irregular convergence behavior, smoothing algorithms have been proposed such as the generalized conjugate gradient method [93] and the composite step bi-conjugate gradient method [4].

2.7 Product-type Krylov subspace methods based on Bi-CG

In this section, let us introduce the product-type methods which overcome the drawbacks of Bi-CG. The IDR method [95] by Wesseling and Sonneveld has been proposed earlier. However, since the IDR method may suffer from severe cancellation, it was not given further attention. Nine years later, the CGS method [81] was proposed by Sonneveld, which computes the square of the Bi-CG polynomials without requiring A^{H}. When Bi-CG converges, CGS often converges in the half of the number of iterations. However, CGS often shows more irregular convergence behavior than Bi-CG. To avoid such convergence behavior, the Bi-CGSTAB method was proposed by van der Vorst [86]. It is shown by many numerical examples that Bi-CGSTAB often gives much smoother convergence behavior than CGS. In terms of exact precision arithmetic, IDR and Bi-CGSTAB are mathematically equivalent. Hence, we can see that it is quite important to develop numerically robust algorithm. Recently, CGS, Bi-CGSTAB, and Bi-CGSTAB2 were generalized by Zhang. By using his framework, the GPBi-CG method has been proposed [97]. In the next subsection, we give the framework of Zhang's product type methods.

2.7.1 Definition of the product-type methods

The nth residual vector of Bi-CG is characterized by the product of the n degree Lanczos polynomial $R_n(\lambda)$ and the initial residual vector:

$$\boldsymbol{r}_n^{\mathrm{BiCG}} = R_n(A)\boldsymbol{r}_0.$$

In the product-type methods, the convergence behavior can be improved by the product of a polynomial of degree n and the nth Bi-CG residual:

$$\boldsymbol{r}_n = H_n(A)\boldsymbol{r}_n^{\mathrm{BiCG}} = H_n(A)R_n(A)\boldsymbol{r}_0.$$

To improve the convergence behavior of Bi-CG, the residual vector must meet the following requirements:

(1) The recurrence for computing the residual vectors of the product-type method must be given by the determination of H_n.

(2) Two parameters α_n, β_n in the polynomial R_n have to be determined by the Petrov-Galerkin condition (2.4).

(3) H_n needs to have short-term recurrences for low computational costs per iteration and low memory requirements.

(4) The choice of parameters in H_n is important to accelerate and stabilize the the convergence of Bi-CG.

Restructuring of residual polynomials

Similar to the Lanczos polynomials (2.21)-(2.23), let us introduce three-term recurrences:

$$H_0(\lambda) := 1, \tag{2.40}$$
$$H_1(\lambda) := (1-\zeta_0\lambda)H_0(\lambda), \tag{2.41}$$
$$H_n(\lambda) := (1+\eta_{n-1}-\zeta_{n-1}\lambda)H_{n-1}(\lambda) - \eta_{n-1}H_{n-2}(\lambda), \quad n=2,3,\ldots \tag{2.42}$$

Here, we note that the choice $\zeta_n = \alpha_n, \eta_n = \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n$ leads to the relation $H_n = R_n$. Hence, H_n can be regarded as a natural generalization of the polynomial R_n. Now, we transform the above three-term recurrences into a coupled two-term ones. Since the recurrence (2.42) is of the form $H_n(\lambda) - H_{n-1}(\lambda) = -\zeta_{n-1}\lambda H_{n-1}(\lambda) + \eta_{n-1}(H_{n-1}(\lambda) - H_{n-2}(\lambda))$, by using the definition

$$G_{n-1}(\lambda) := \frac{H_{n-1}(\lambda) - H_n(\lambda)}{\lambda},$$

we obtain the following coupled two-term recurrences:

$$H_0(\lambda) = 1, \quad G_0(\lambda) = \zeta_0,$$
$$H_n(\lambda) = H_{n-1}(\lambda) - \lambda G_{n-1}(\lambda), \tag{2.43}$$
$$G_n(\lambda) = \zeta_n H_n(\lambda) + \eta_n G_{n-1}(\lambda), \quad n=1,2,\ldots \tag{2.44}$$

By the above polynomial, we can derive several well-known iterative methods from the choice of the two parameters ζ_n, η_n such as CGS, Bi-CGSTAB, Bi-CGSTAB2, and GPBi-CG. Let us show the relationships in Table 2.4.

Table 2.4. The choice for the product-type methods.

CGS	$\zeta_n = \alpha_n, \quad \eta_n = \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n \iff H_n = R_n$
Bi-CGSTAB	$\zeta_n = \arg \min_{\zeta_n \in \mathcal{C}} \|\boldsymbol{r}_{n+1}\|, \quad \eta_n = 0$
Bi-CGSTAB2	$\zeta_n, \eta_n = \arg \min_{\zeta_n, \eta_n \in \mathcal{C}} \|\boldsymbol{r}_{n+1}\|$ ($\eta_n = 0$ at even iterations)
GPBi-CG	$\zeta_n, \eta_n = \arg \min_{\zeta_n, \eta_n \in \mathcal{C}} \|\boldsymbol{r}_{n+1}\|$

2.7.2 Derivations of CGS, Bi-CGSTAB, and GPBi-CG

The choice for CGS

If we choose $\zeta_n = \alpha_n$ and $\eta_n = \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n$ in recurrence relations (2.43) and (2.44). Then, the polynomials H_n and G_n become the Lanczos polynomial:

$$\begin{aligned} H_n(\lambda) &= R_n(\lambda), \\ G_n(\lambda) &= \alpha_n P_n(\lambda). \end{aligned}$$

This leads to the algorithm of CGS given below.

Algorithm 2.17: CGS method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \beta_{-1} = 0,$
$\boldsymbol{r}_0^*$ is an arbitrary vector, such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0,\ e.g.,\ \boldsymbol{r}_0^* = \boldsymbol{r}_0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \leq \epsilon\|\boldsymbol{b}\|$ do:

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{z}_{n-1},$$
$$\boldsymbol{u}_n = \boldsymbol{p}_n + \beta_{n-1}(\boldsymbol{z}_{n-1} + \beta_{n-1}\boldsymbol{u}_{n-1}),$$
$$\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{u}_n)},$$
$$\boldsymbol{z}_n = \boldsymbol{p}_n - \alpha_n A\boldsymbol{u}_n,$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n(\boldsymbol{p}_n + \boldsymbol{z}_n),$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A(\boldsymbol{p}_n + \boldsymbol{z}_n),$$
$$\beta_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}.$$

end

The choice for Bi-CGSTAB

In this subsection, we describe the choice for Bi-CGSTAB. If we choose $\eta_n = 0$, and ζ_n such that the 2-norm of $\boldsymbol{r}_{n+1}$ is locally minimized, more precisely, it is determined by the following orthogonalization:

$$\boldsymbol{r}_{n+1} \perp A\boldsymbol{t}_n.$$

Then, Bi-CGSTAB is obtained, which has the simplest recurrences of other product-type Krylov subspace methods.

Algorithm 2.18: Bi-CGSTAB method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \beta_{-1} = 0,$
$\boldsymbol{r}_0^*$ is an arbitrary vector, such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0,\ e.g.,\ \boldsymbol{r}_0^* = \boldsymbol{r}_0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \leq \epsilon\|\boldsymbol{b}\|$ do:

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \zeta_{n-1}A\boldsymbol{p}_{n-1}),$$
$$\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)},$$
$$\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,$$
$$\zeta_n = \frac{(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)},$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n + \zeta_n\boldsymbol{t}_n,$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \zeta_n A\boldsymbol{t}_n,$$
$$\beta_n = \frac{\alpha_n}{\zeta_n} \cdot \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}.$$

end

The choice for GPBi-CG

In this subsection, we describe the choice for GPBi-CG. If we choose ζ_n and η such that the 2-norm of $\boldsymbol{r}_{n+1}$ is locally minimized, more precisely, the two parameters are determined by the following orthogonalization:

$$\boldsymbol{r}_{n+1} \perp \mathrm{span}\{A\boldsymbol{t}_n, \boldsymbol{y}_n\}.$$

Then, the algorithm of GPBi-CG is obtained as follows:

Algorithm 2.19: GPBi-CG method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \boldsymbol{t}_{-1} = \boldsymbol{w}_{-1} = \boldsymbol{0},\ \beta_{-1} = 0,$
$\boldsymbol{r}_0^*$ is an arbitrary vector, such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0,\ e.g.,\ \boldsymbol{r}_0^* = \boldsymbol{r}_0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \leq \epsilon\|\boldsymbol{b}\|$ do:

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \boldsymbol{u}_{n-1}),$$
$$\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)},$$
$$\boldsymbol{y}_n = \boldsymbol{t}_{n-1} - \boldsymbol{r}_n - \alpha_n\boldsymbol{w}_{n-1} + \alpha_n A\boldsymbol{p}_n,$$
$$\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,$$
$$\zeta_n = \frac{(\boldsymbol{y}_n, \boldsymbol{y}_n)(A\boldsymbol{t}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, \boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)},$$
$$\eta_n = \frac{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)},$$
$$(\text{if } n = 0,\ \text{then } \zeta_n = \frac{(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)},\ \eta_n = 0)$$
$$\boldsymbol{u}_n = \zeta_n A\boldsymbol{p}_n + \eta_n(\boldsymbol{t}_{n-1} - \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{u}_{n-1}),$$
$$\boldsymbol{z}_n = \zeta_n\boldsymbol{r}_n + \eta_n\boldsymbol{z}_{n-1} - \alpha_n\boldsymbol{u}_n,$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n + \boldsymbol{z}_n,$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \eta_n\boldsymbol{y}_n - \zeta_n A\boldsymbol{t}_n,$$
$$\beta_n = \frac{\alpha_n}{\zeta_n} \cdot \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)},$$
$$\boldsymbol{w}_n = A\boldsymbol{t}_n + \beta_n A\boldsymbol{p}_n.$$

end

Since Zhang's framework includes the previous product-type methods, it admits also hybrid variants in which one can shift from one of them to another, *e.g.* CGS to BiCGSTAB, at some iteration steps. Even though BiCGSTAB(ℓ) does not fall in this framework, it is also practical and useful for recent scientific computing. However, the optimal choice for ℓ is an open problem.

So far, we have seen some important Krylov subspace methods and described these ideas. However, we have not described some important methods and preconditioners such as the product type methods based on QMR and flexible preconditioner. For details of the Krylov subspace methods and preconditioners. See Freund *et al.* [40], Golub & van der Vorst [47], Gutknecht [52], Saad & van der Vorst [72].

Chapter 3

Bi-CR: a biconjugate residual method

3.1 Introduction

CG, MINRES, and CR given in the previous chapter have been extended to non-Hermitian linear systems. FOM is an extension of CG. GMRES and GCR (given in Appendix B) are extensions of MINRES and CR respectively. There is a similarity among FOM, GMRES, and GCR in that they are based on the Arnoldi process.

On the other hand, based on the bi-Lanczos process, CG was extended to non-Hermitian linear systems. The algorithm is known as Bi-CG. Similarly, in this chapter, based on a bi-Lanczos like process, CR is extended to non-Hermitian linear systems.

The chapter is organized as follows: in the next section, we first describe a simple derivation of Bi-CG. Then, based on the derivation, we extend CR to nonsymmetric linear systems. In §3.3, we discuss some properties of the extended algorithm, and we give other derivations. In §3.4, we report some numerical experiments. Finally, we present conclusions and ideas for future work in §3.5.

3.2 An extension of CR without loss of short-term recurrence property

In this section, we describe one of the simplest derivations of Bi-CG, and then CR is extended to nonsymmetric linear systems by analogously using this derivation process.

3.2.1 H. A. van der Vorst's derivation of Bi-CG

The Bi-CG method is a Krylov subspace method for solving nonsymmetric linear systems

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{3.1}$$

where A is an $N \times N$ real nonsymmetric matrix and $\boldsymbol{b}$ is an N-vector. The algorithm of Bi-CG is well known, and there are several ways to obtain it. Recently, one of the simplest derivations has been given by H. A. van der Vorst [87, pp.97-98], which was inspired by [54]. In this subsection, we give the details of his derivation.

First, using (3.1) and a dual linear system $A^{\mathrm{T}}\boldsymbol{x}^* = \boldsymbol{b}^*$, we consider the following $2N \times 2N$ symmetric linear system:

$$\begin{bmatrix} O & A \\ A^{\mathrm{T}} & O \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b}^* \end{bmatrix}, \text{ or } \tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}. \tag{3.2}$$

If we apply the algorithm of CG with the preconditioner

$$\tilde{M} = \begin{bmatrix} O & I \\ I & O \end{bmatrix}, \quad I : \text{identity matrix}, \tag{3.3}$$

to (3.2), then the resulting algorithm at the nth iteration step can be written as

$$\begin{aligned}
\tilde{\boldsymbol{p}}_n^{\mathrm{CG}} &= \tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{CG}} + \beta_{n-1}\tilde{\boldsymbol{p}}_{n-1}^{\mathrm{CG}}, \\
\alpha_n &= \frac{(\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{CG}}, \tilde{\boldsymbol{r}}_n^{\mathrm{CG}})}{(\tilde{\boldsymbol{p}}_n^{\mathrm{CG}}, \tilde{A}\tilde{\boldsymbol{p}}_n^{\mathrm{CG}})}, \\
\tilde{\boldsymbol{x}}_{n+1}^{\mathrm{CG}} &= \tilde{\boldsymbol{x}}_n^{\mathrm{CG}} + \alpha_n\tilde{\boldsymbol{p}}_n^{\mathrm{CG}}, \\
\tilde{\boldsymbol{r}}_{n+1}^{\mathrm{CG}} &= \tilde{\boldsymbol{r}}_n^{\mathrm{CG}} - \alpha_n\tilde{A}\tilde{\boldsymbol{p}}_n^{\mathrm{CG}}, \\
\beta_n &= \frac{(\tilde{M}^{-1}\tilde{\boldsymbol{r}}_{n+1}^{\mathrm{CG}}, \tilde{\boldsymbol{r}}_{n+1}^{\mathrm{CG}})}{(\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{CG}}, \tilde{\boldsymbol{r}}_n^{\mathrm{CG}})}.
\end{aligned}$$

Substituting $\tilde{M}^{-1} = \tilde{M}$ of (3.3) and the vectors

$$\tilde{\boldsymbol{x}}_n^{\mathrm{CG}} := \begin{bmatrix} \boldsymbol{x}_n^{\mathrm{BiCG}*} \\ \boldsymbol{x}_n^{\mathrm{BiCG}} \end{bmatrix}, \ \tilde{\boldsymbol{r}}_n^{\mathrm{CG}} := \begin{bmatrix} \boldsymbol{r}_n^{\mathrm{BiCG}} \\ \boldsymbol{r}_n^{\mathrm{BiCG}*} \end{bmatrix}, \ \tilde{\boldsymbol{p}}_n^{\mathrm{CG}} := \begin{bmatrix} \boldsymbol{p}_n^{\mathrm{BiCG}*} \\ \boldsymbol{p}_n^{\mathrm{BiCG}} \end{bmatrix}$$

in the previous recurrences, we readily obtain the algorithm of Bi-CG.

Furthermore, van der Vorst also states in [87, p.98] that the preconditioned Bi-CG can be derived from the above framework with the preconditioner

$$\tilde{M} = \begin{bmatrix} O & K \\ K^{\mathrm{T}} & O \end{bmatrix}. \tag{3.4}$$

In the next subsection, we extend the CR method to nonsymmetric linear systems using this framework.

3.2.2 An extension of CR to nonsymmetric linear systems

We see from the previous subsection that the extended algorithm of CG is obtained by applying the preconditioned CG method to (3.2), and that the resulting algorithm (Bi-CG) has attractive coupled two-term recurrences. Analogously, in this subsection, we consider applying the preconditioned CR method to (3.2).

Based on the unpreconditioned CR algorithm found in [70, p.194], we can obtain the preconditioned algorithm described below.

Algorithm 3.1: Preconditioned CR method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,

set $\boldsymbol{p}_{-1} = 0,\ \beta_{-1} = 0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:
$$\boldsymbol{p}_n = M^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},$$
$$\alpha_n = \frac{(M^{-1}\boldsymbol{r}_n, AM^{-1}\boldsymbol{r}_n)}{(M^{-1}A\boldsymbol{p}_n, A\boldsymbol{p}_n)},$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,$$
$$\beta_n = \frac{(M^{-1}\boldsymbol{r}_{n+1}, AM^{-1}\boldsymbol{r}_{n+1})}{(M^{-1}\boldsymbol{r}_n, AM^{-1}\boldsymbol{r}_n)}.$$
end

For actual computations, two recurrences, $A\boldsymbol{p}_n = AM^{-1}\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1}$ and $M^{-1}\boldsymbol{r}_{n+1} = M^{-1}\boldsymbol{r}_n - \alpha_n M^{-1}A\boldsymbol{p}_n$, are added to Algorithm 3.1 to reduce the number of matrix-vector products and solve the preconditioner in each iteration step.
Next, we apply Algorithm 3.1 with the preconditioner (3.3) to (3.2), so that we have

$$\tilde{\boldsymbol{p}}_n = \tilde{M}^{-1}\tilde{\boldsymbol{r}}_n + \beta_{n-1}\tilde{\boldsymbol{p}}_{n-1},$$
$$\alpha_n = \frac{(\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n, \tilde{A}\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n)}{(M^{-1}\tilde{A}\tilde{\boldsymbol{p}}_n, \tilde{A}\tilde{\boldsymbol{p}}_n)},$$
$$\tilde{\boldsymbol{x}}_{n+1} = \tilde{\boldsymbol{x}}_n + \alpha_n\tilde{\boldsymbol{p}}_n,$$
$$\tilde{\boldsymbol{r}}_{n+1} = \tilde{\boldsymbol{r}}_n - \alpha_n\tilde{A}\tilde{\boldsymbol{p}}_n,$$
$$\beta_n = \frac{(\tilde{M}^{-1}\tilde{\boldsymbol{r}}_{n+1}, \tilde{A}\tilde{M}^{-1}\tilde{\boldsymbol{r}}_{n+1})}{(\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n, \tilde{A}\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n)}.$$

Substituting $\tilde{M}^{-1} = \tilde{M}$ of (3.3) and the vectors

$$\tilde{\boldsymbol{x}}_n := \begin{bmatrix} \boldsymbol{x}_n^* \\ \boldsymbol{x}_n \end{bmatrix},\ \tilde{\boldsymbol{r}}_n := \begin{bmatrix} \boldsymbol{r}_n \\ \boldsymbol{r}_n^* \end{bmatrix},\ \tilde{\boldsymbol{p}}_n := \begin{bmatrix} \boldsymbol{p}_n^* \\ \boldsymbol{p}_n \end{bmatrix}$$

in the previous recurrences, we readily obtain the new algorithm. Since Bi-CG is obtained from the preconditioned CG and the algorithm is obtained from the preconditioned CR, we call it Bi-CR.

Algorithm 3.2: Bi-CR method for real systems

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
choose $\boldsymbol{r}_0^*$ (for example, $\boldsymbol{r}_0^* = \boldsymbol{r}_0$),
set $\boldsymbol{p}_{-1}^* = \boldsymbol{p}_{-1} = 0,\ \beta_{-1} = 0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:

(3.5) $$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},$$

(3.6) $$\boldsymbol{p}_n^* = \boldsymbol{r}_n^* + \beta_{n-1}\boldsymbol{p}_{n-1}^*,$$
$$(A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1},)$$

(3.7) $$\alpha_n = \frac{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(A^{\mathrm{T}}\boldsymbol{p}_n^*, A\boldsymbol{p}_n)},$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,$$

$$
\begin{aligned}
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,\\
&\boldsymbol{r}^*_{n+1} = \boldsymbol{r}^*_n - \alpha_n A^{\mathrm{T}}\boldsymbol{p}^*_n,\\
&\beta_n = \frac{(\boldsymbol{r}^*_{n+1}, A\boldsymbol{r}_{n+1})}{(\boldsymbol{r}^*_n, A\boldsymbol{r}_n)}.
\end{aligned} \tag{3.8}
$$

end

Here, $A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1}$ is newly added to reduce the number of matrix-vector products per iteration step. We see from Algorithm 3.2 that if the coefficient matrix is symmetric, Bi-CR reduces to CR.

Furthermore, if we apply Algorithm 3.1 with the preconditioner (3.4) to (3.2), then we obtain the following preconditioned Bi-CR:

Algorithm 3.3: Preconditioned Bi-CR method for real systems

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
choose $\boldsymbol{r}^*_0$ (for example, $\boldsymbol{r}^*_0 = \boldsymbol{r}_0$),
set $\boldsymbol{p}^*_{-1} = \boldsymbol{p}_{-1} = 0,\ \beta_{-1} = 0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:

$$
\begin{aligned}
&\boldsymbol{p}_n = K^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},\\
&\boldsymbol{p}^*_n = K^{-\mathrm{T}}\boldsymbol{r}^*_n + \beta_{n-1}\boldsymbol{p}^*_{n-1},\\
&(A\boldsymbol{p}_n = AK^{-1}\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1},)\\
&\alpha_n = \frac{(K^{-\mathrm{T}}\boldsymbol{r}^*_n, AK^{-1}\boldsymbol{r}_n)}{(K^{-\mathrm{T}}A^{\mathrm{T}}\boldsymbol{p}^*_n, A\boldsymbol{p}_n)},\\
&\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,\\
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,\\
&\boldsymbol{r}^*_{n+1} = \boldsymbol{r}^*_n - \alpha_n A^{\mathrm{T}}\boldsymbol{p}^*_n,\\
&(K^{-\mathrm{T}}\boldsymbol{r}^*_{n+1} = K^{-\mathrm{T}}\boldsymbol{r}^*_n - \alpha_n K^{-\mathrm{T}}A^{\mathrm{T}}\boldsymbol{p}^*_n,)\\
&\beta_n = \frac{(K^{-\mathrm{T}}\boldsymbol{r}^*_{n+1}, AK^{-1}\boldsymbol{r}_{n+1})}{(K^{-\mathrm{T}}\boldsymbol{r}^*_n, AK^{-1}\boldsymbol{r}_n)}.
\end{aligned}
$$

end

Here, let us remark on the name Bi-CR. In [13], a biconjugate residual (BCR) method is introduced with the same name as our method; however, the two algorithms are mathematically different since Algorithm 3.2 generates $\boldsymbol{x}_n - \boldsymbol{x}_0 \in K_n(A, \boldsymbol{r}_0)$ while BCR generates $\boldsymbol{x}^{\mathrm{BCR}}_n - \boldsymbol{x}^{\mathrm{BCR}}_0 \notin K_n(A, \boldsymbol{r}^{\mathrm{BCR}}_0)$ for nonsymmetric linear systems.

At the end of this section, we show the computational cost for Bi-CG and Bi-CR in Table 3.1. "AXPY" denotes addition of scaled vectors, "6 or 7" denotes "6" for the unpreconditioned Bi-CR and "7" for the preconditioned one, and "1+1" denotes 1 multiplication with the matrix and 1 with its transpose.

Table 3.1. Summary of cost per iteration step.

Method	Inner Product	AXPY	Matrix-Vector Product	Preconditioner Solve
Bi-CG	2	5	1+1	1+1
Bi-CR	2	6 or 7	1+1	1+1

From Table 3.1, we can say that Bi-CG and Bi-CR require almost the same memory and computational work per iteration step.

3.2.3 A derivation of Bi-CR for non-Hermitian linear systems

In this subsection, we show that a complex version of Bi-CR can be derived from the preconditioned COCR method given later (Algorithm 4.2) and an extension of van der Vorst's framework. First, we consider the following $2N \times 2N$ complex symmetric linear system:

$$\begin{bmatrix} O & A \\ A^{\mathrm{T}} & O \end{bmatrix} \begin{bmatrix} \overline{\boldsymbol{x}}^* \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \overline{\boldsymbol{b}}^* \end{bmatrix}, \text{ or } \tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}. \tag{3.9}$$

We can see from the system (3.9) that it is mathematically equivalent to

$$A\boldsymbol{x} = \boldsymbol{b} \quad \text{and} \quad A^{\mathrm{T}}\overline{\boldsymbol{x}}^* = \overline{\boldsymbol{b}}^* (\Leftrightarrow A^{\mathrm{H}}\boldsymbol{x}^* = \boldsymbol{b}^*).$$

If we apply the algorithm of COCR with the preconditioner

$$\tilde{M} = \begin{bmatrix} O & I \\ I & O \end{bmatrix}, \quad I : \text{identity matrix}, \tag{3.10}$$

to (3.9), then the resulting algorithm at the nth iteration step can be written as

$$\begin{aligned}
&\tilde{\boldsymbol{p}}_n^{\mathrm{COCR}} = \tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{COCR}} + \beta_{n-1}\tilde{\boldsymbol{p}}_{n-1}^{\mathrm{COCR}}, \\
&\alpha_n = \frac{(\overline{\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{COCR}}}, \tilde{A}\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{COCR}})}{(\overline{\tilde{A}\tilde{\boldsymbol{p}}_n^{\mathrm{COCR}}}, \tilde{M}^{-1}\tilde{A}\tilde{\boldsymbol{p}}_n^{\mathrm{COCR}})}, \\
&\tilde{\boldsymbol{x}}_{n+1}^{\mathrm{COCR}} = \tilde{\boldsymbol{x}}_n^{\mathrm{COCR}} + \alpha_n\tilde{\boldsymbol{p}}_n^{\mathrm{COCR}}, \\
&\tilde{\boldsymbol{r}}_{n+1}^{\mathrm{COCR}} = \tilde{\boldsymbol{r}}_n^{\mathrm{COCR}} - \alpha_n\tilde{A}\tilde{\boldsymbol{p}}_n^{\mathrm{COCR}}, \\
&\beta_n = \frac{(\overline{\tilde{M}^{-1}\tilde{\boldsymbol{r}}_{n+1}^{\mathrm{COCR}}}, \tilde{A}\tilde{M}^{-1}\tilde{\boldsymbol{r}}_{n+1}^{\mathrm{COCR}})}{(\overline{\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{COCR}}}, \tilde{A}\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{COCR}})}.
\end{aligned}$$

Substituting $\tilde{M}^{-1}(= \tilde{M})$ of (3.10) and the vectors

$$\tilde{\boldsymbol{x}}_n^{\mathrm{COCR}} := \begin{bmatrix} \boldsymbol{x}_n \\ \overline{\boldsymbol{x}}_n^* \end{bmatrix}, \ \tilde{\boldsymbol{r}}_n^{\mathrm{COCR}} := \begin{bmatrix} \boldsymbol{r}_n \\ \overline{\boldsymbol{r}}_n^* \end{bmatrix}, \ \tilde{\boldsymbol{p}}_n^{\mathrm{COCR}} := \begin{bmatrix} \overline{\boldsymbol{p}}_n^* \\ \boldsymbol{p}_n \end{bmatrix}$$

in the previous recurrences, and using the following results,

$$\begin{aligned}
(\overline{\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{COCR}}}, \tilde{A}\tilde{M}^{-1}\tilde{\boldsymbol{r}}_n^{\mathrm{COCR}}) &= [\overline{\boldsymbol{r}}_n^{*\,\mathrm{T}}\boldsymbol{r}_n^{\mathrm{T}}] \begin{bmatrix} A\boldsymbol{r}_n \\ A^{\mathrm{T}}\overline{\boldsymbol{r}}_n^* \end{bmatrix} \\
&= \overline{\boldsymbol{r}}_n^{*\,\mathrm{T}} A\boldsymbol{r}_n + \boldsymbol{r}_n^{\mathrm{T}} A^{\mathrm{T}}\overline{\boldsymbol{r}}_n^* \\
&= \boldsymbol{r}_n^{*\,\mathrm{H}} A\boldsymbol{r}_n + (\boldsymbol{r}_n^{\mathrm{T}} A^{\mathrm{T}}\overline{\boldsymbol{r}}_n^*)^{\mathrm{T}} \\
&= (\boldsymbol{r}_n^*, A\boldsymbol{r}_n) + \boldsymbol{r}_n^{*\,\mathrm{H}} A\boldsymbol{r}_n \\
&= 2(\boldsymbol{r}_n^*, A\boldsymbol{r}_n), \\
(\overline{\tilde{A}\tilde{\boldsymbol{p}}_n^{\mathrm{COCR}}}, \tilde{M}^{-1}\tilde{A}\tilde{\boldsymbol{p}}_n^{\mathrm{COCR}}) &= [(A\boldsymbol{p}_n)^{\mathrm{T}}(A^{\mathrm{T}}\overline{\boldsymbol{p}}_n^*)^{\mathrm{T}}] \begin{bmatrix} A^{\mathrm{T}}\overline{\boldsymbol{p}}_n^* \\ A\boldsymbol{p}_n \end{bmatrix}
\end{aligned}$$

$$
\begin{aligned}
&= (A\boldsymbol{p}_n)^{\mathrm{T}} A^{\mathrm{T}} \overline{\boldsymbol{p}}_n^* + (A^{\mathrm{T}} \overline{\boldsymbol{p}}_n^*)^{\mathrm{T}} A\boldsymbol{p}_n \\
&= (\boldsymbol{p}_n^{\mathrm{T}} A^{\mathrm{T}} A^{\mathrm{T}} \overline{\boldsymbol{p}}_n^*)^{\mathrm{T}} + (A^{\mathrm{H}} \boldsymbol{p}_n^*)^{\mathrm{H}} A\boldsymbol{p}_n \\
&= (A^{\mathrm{H}} \boldsymbol{p}_n^*)^{\mathrm{H}} A\boldsymbol{p}_n + (A^{\mathrm{H}} \boldsymbol{p}_n^*, A\boldsymbol{p}_n) \\
&= 2(A^{\mathrm{H}} \boldsymbol{p}_n^*, A\boldsymbol{p}_n),
\end{aligned}
$$

we readily obtain the algorithm of Bi-CR for solving general non-Hermitian linear systems.

Algorithm 3.4: Bi-CR method for complex systems

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
choose $\boldsymbol{r}_0^*$ (for example, $\boldsymbol{r}_0^* = \boldsymbol{r}_0$),
set $\boldsymbol{p}_{-1}^* = \boldsymbol{p}_{-1} = 0,\ \beta_{-1} = 0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon \|\boldsymbol{b}\|$ do:

$$
\begin{aligned}
&\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \\
&\boldsymbol{p}_n^* = \boldsymbol{r}_n^* + \bar{\beta}_{n-1}\boldsymbol{p}_{n-1}^*, \\
&(A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1} A\boldsymbol{p}_{n-1},) \\
&\alpha_n = \frac{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{p}_n^*, A\boldsymbol{p}_n)}, \\
&\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n, \\
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n, \\
&\boldsymbol{r}_{n+1}^* = \boldsymbol{r}_n^* - \bar{\alpha}_n A^{\mathrm{H}} \boldsymbol{p}_n^*, \\
&\beta_n = \frac{(\boldsymbol{r}_{n+1}^*, A\boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}.
\end{aligned}
$$

end

Here, $A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1} A\boldsymbol{p}_{n-1}$ is newly added to reduce the number of matrix-vector products per iteration step.

Furthermore, if we apply the COCR method with the preconditioner (3.4) to (3.2), then we obtain the following preconditioned Bi-CR:

Algorithm 3.5: Preconditioned Bi-CR method for complex systems

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
choose $\boldsymbol{r}_0^*$ (for example, $\boldsymbol{r}_0^* = \boldsymbol{r}_0$),
set $\boldsymbol{p}_{-1}^* = \boldsymbol{p}_{-1} = 0,\ \beta_{-1} = 0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon \|\boldsymbol{b}\|$ do:

$$
\begin{aligned}
&\boldsymbol{p}_n = K^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \\
&\boldsymbol{p}_n^* = K^{-\mathrm{H}}\boldsymbol{r}_n^* + \bar{\beta}_{n-1}\boldsymbol{p}_{n-1}^*, \\
&(A\boldsymbol{p}_n = AK^{-1}\boldsymbol{r}_n + \beta_{n-1} A\boldsymbol{p}_{n-1},) \\
&\alpha_n = \frac{(K^{-\mathrm{H}}\boldsymbol{r}_n^*, AK^{-1}\boldsymbol{r}_n)}{(K^{-\mathrm{H}}A^{\mathrm{H}}\boldsymbol{p}_n^*, A\boldsymbol{p}_n)}, \\
&\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n, \\
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n, \\
&\boldsymbol{r}_{n+1}^* = \boldsymbol{r}_n^* - \bar{\alpha}_n A^{\mathrm{H}} \boldsymbol{p}_n^*,
\end{aligned}
$$

$$(K^{-\mathrm{H}}\boldsymbol{r}^*_{n+1} = K^{-\mathrm{H}}\boldsymbol{r}^*_n - \bar{\alpha}_n K^{-\mathrm{H}}A^{\mathrm{H}}\boldsymbol{p}^*_n,)$$
$$\beta_n = \frac{(K^{-\mathrm{H}}\boldsymbol{r}^*_{n+1}, AK^{-1}\boldsymbol{r}_{n+1})}{(K^{-\mathrm{H}}\boldsymbol{r}^*_n, AK^{-1}\boldsymbol{r}_n)}.$$

end

The derivation processes for Bi-CR that we gave are shown below.

$$\boxed{\text{PCOCR}}\ \text{(complex)} \quad \overset{[\S 3.2.3]}{\longrightarrow} \quad \boxed{\text{PBi-CR}}\ \text{(complex)}$$
$$\Big\downarrow\ [A \in \mathcal{R}^{N\times N}] \qquad\qquad \Big\downarrow\ [A \in \mathcal{R}^{N\times N}]$$
$$\boxed{\text{PCR}}\ \text{(real)} \quad \overset{[\S 3.2.2]}{\longrightarrow} \quad \boxed{\text{PBi-CR}}\ \text{(real)}$$

From the above, we see that the derivation process in this subsection includes the previous one. Hence, we can say that PCOCR plays an essential role in obtaining PBi-CR.

3.3 Some properties and other derivations of Bi-CR

In this section, we discuss some properties of Bi-CR, and then give other derivations.

3.3.1 Some properties

Observing Algorithm 3.2, we see that four iterates $\boldsymbol{r}_n, \boldsymbol{p}_n, \boldsymbol{r}^*_n$, and $\boldsymbol{p}^*_n$ can be expressed as

$$\boldsymbol{r}_n = R_n(A)\boldsymbol{r}_0, \qquad \boldsymbol{p}_n = P_n(A)\boldsymbol{r}_0, \tag{3.11}$$
$$\boldsymbol{r}^*_n = R_n(A^{\mathrm{T}})\boldsymbol{r}^*_0, \qquad \boldsymbol{p}^*_n = P_n(A^{\mathrm{T}})\boldsymbol{r}^*_0, \tag{3.12}$$

where R_n and P_n are polynomials of degree n satisfying

$$\begin{aligned} R_0(\lambda) &:= 1, \quad P_0(\lambda) := 1, \\ R_n(\lambda) &:= R_{n-1}(\lambda) - \alpha_{n-1}\lambda P_{n-1}(\lambda), \\ P_n(\lambda) &:= R_n(\lambda) + \beta_{n-1}P_{n-1}(\lambda), \quad n = 1, 2, \ldots \end{aligned}$$

From (3.11), (3.12), and Algorithm 3.2, the following results are obtained if breakdown does not occur:

Theorem 3.3.1 *For $i \neq j$, the following biorthogonality properties hold:*

$$(\boldsymbol{r}^*_i, A\boldsymbol{r}_j) = 0, \tag{3.13}$$
$$(A^T\boldsymbol{p}^*_i, A\boldsymbol{p}_j) = 0. \tag{3.14}$$

Proof. It follows from (3.11) and (3.12) that

$$(\boldsymbol{r}^*_i, A\boldsymbol{r}_j) = (R_i(A^{\mathrm{T}})\boldsymbol{r}^*_0, AR_j(A)\boldsymbol{r}_0) = (R_j(A^{\mathrm{T}})\boldsymbol{r}^*_0, AR_i(A)\boldsymbol{r}_0) = (\boldsymbol{r}^*_j, A\boldsymbol{r}_i).$$

Similarly, from (3.14) we obtain $(A^{\mathrm{T}}\boldsymbol{p}^*_i, A\boldsymbol{p}_j) = (A^{\mathrm{T}}\boldsymbol{p}^*_j, A\boldsymbol{p}_i)$. Hence, the statements of (3.13) and (3.14) are equivalent to

$$(\boldsymbol{r}^*_i, A\boldsymbol{r}_j) = 0 \quad \text{and} \quad (A^{\mathrm{T}}\boldsymbol{p}^*_i, A\boldsymbol{p}_j) = 0 \quad \text{for all} \ \ j < i. \tag{3.15}$$

Now, we give the proof of (3.15) by induction. Since the trivial case is obvious from Algorithm 3.2, we assume that properties (3.15) hold for $j < i \leq k$. Then, we show that

$$(\boldsymbol{r}^*_{k+1}, A\boldsymbol{r}_j) = 0, \tag{3.16}$$

$$(A^{\mathrm{T}}\boldsymbol{p}^*_{k+1}, A\boldsymbol{p}_j) = 0. \tag{3.17}$$

First, let us show (3.16). For the case $j < k$ it follows from the above assumption that

$$\begin{aligned}(\boldsymbol{r}^*_{k+1}, A\boldsymbol{r}_j) &= (\boldsymbol{r}^*_k, A\boldsymbol{r}_j) - \alpha_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{r}_j)\\ &= -\alpha_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{r}_j)\\ &= -\alpha_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_j) - \alpha_k\beta_{j-1}(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_{j-1})\\ &= 0.\end{aligned}$$

For the case $j = k$ we obtain

$$\begin{aligned}(\boldsymbol{r}^*_{k+1}, A\boldsymbol{r}_k) &= (\boldsymbol{r}^*_k, A\boldsymbol{r}_k) - \alpha_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{r}_k)\\ &= (\boldsymbol{r}^*_k, A\boldsymbol{r}_k) - \alpha_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_k) - \alpha_k\beta_{k-1}(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_{k-1})\\ &= (\boldsymbol{r}^*_k, A\boldsymbol{r}_k) - \alpha_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_k)\\ &= 0\end{aligned}$$

from the computational formula of α_k of (3.7). Next, we show (3.17). For the case $j < k$ it follows from the first result of the proof that

$$(A^{\mathrm{T}}\boldsymbol{p}^*_{k+1}, A\boldsymbol{p}_j) = (A^{\mathrm{T}}\boldsymbol{r}^*_{k+1}, A\boldsymbol{p}_j) + \beta_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_j) = \frac{1}{\alpha_j}(A^{\mathrm{T}}\boldsymbol{r}^*_{k+1}, \boldsymbol{r}_j - \boldsymbol{r}_{j+1}) = 0.$$

For the case $j = k$ we obtain

$$\begin{aligned}(A^{\mathrm{T}}\boldsymbol{p}^*_{k+1}, A\boldsymbol{p}_k) &= (A^{\mathrm{T}}\boldsymbol{r}^*_{k+1}, A\boldsymbol{p}_k) + \beta_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_k)\\ &= \frac{1}{\alpha_k}(A^{\mathrm{T}}\boldsymbol{r}^*_{k+1}, \boldsymbol{r}_k - \boldsymbol{r}_{k+1}) + \beta_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_k)\\ &= -\frac{1}{\alpha_k}(A^{\mathrm{T}}\boldsymbol{r}^*_{k+1}, \boldsymbol{r}_{k+1}) + \beta_k(A^{\mathrm{T}}\boldsymbol{p}^*_k, A\boldsymbol{p}_k)\\ &= 0\end{aligned}$$

from the computational formulas of α_k of (3.7) and β_k of (3.8). □

Corollary 3.3.2 *Some further properties of Bi-CR are*

$$(\boldsymbol{r}^*_i, A\boldsymbol{p}_j) = 0 \quad \textit{for} \;\; i > j, \tag{3.18}$$

$$(\boldsymbol{r}^*_i, A\boldsymbol{r}_i) = (\boldsymbol{r}^*_i, A\boldsymbol{p}_i), \tag{3.19}$$

$$(A^T\boldsymbol{r}^*_i, A\boldsymbol{p}_i) = (A^T\boldsymbol{p}^*_i, A\boldsymbol{p}_i). \tag{3.20}$$

Proof. First, we give the proof of (3.18). From the recurrence (3.5) it follows that $(\boldsymbol{r}^*_i, A\boldsymbol{p}_j) = (\boldsymbol{r}^*_i, A\boldsymbol{r}_j) + \beta_{j-1}(\boldsymbol{r}^*_i, A\boldsymbol{p}_{j-1})$, and thus from the property (3.13) we obtain $(\boldsymbol{r}^*_i, A\boldsymbol{p}_j) = \beta_{j-1}(\boldsymbol{r}^*_i, A\boldsymbol{p}_{j-1})$. Applying this process recursively, we finally obtain $(\boldsymbol{r}^*_i, A\boldsymbol{p}_j)$ $= \beta_{j-1}\beta_{j-2}\cdots\beta_0(\boldsymbol{r}^*_i, A\boldsymbol{p}_0)$. Hence, from $\boldsymbol{p}_0 = \boldsymbol{r}_0$ and (3.13) the property (3.18) is established.

Second, we give the proof of (3.19). From the recurrence (3.5) it follows that $(\boldsymbol{r}_i^*, A\boldsymbol{r}_i) = (\boldsymbol{r}_i^*, A\boldsymbol{p}_i) - \beta_{i-1}(\boldsymbol{r}_i^*, A\boldsymbol{p}_{i-1})$. Since the second term is zero by (3.18), the property (3.19) is established.

Finally, we give the proof of (3.20). From the recurrence (3.6) it follows that $(A^{\mathrm{T}}\boldsymbol{r}_i^*, A\boldsymbol{p}_i) = (A^{\mathrm{T}}\boldsymbol{p}_i^*, A\boldsymbol{p}_i) - \beta_{i-1}(A^{\mathrm{T}}\boldsymbol{p}_{i-1}^*, A\boldsymbol{p}_i)$. Since the second term is zero from (3.14), the property (3.20) is established. □

We see from the algorithm of Bi-CR that it can be also regarded as the algorithm of Bi-CG with formal inner product $(\boldsymbol{y}^*, \boldsymbol{y})_A := (\boldsymbol{y}^*)^{\mathrm{T}} A\boldsymbol{y}$. In this sense, M. Gutknecht have developed Bi-CG for $(\boldsymbol{y}^*, \boldsymbol{y})_B$, where $BA = AB$. The theoretical results are given in [52].

3.3.2 Other derivations

In this subsection, we give other derivations of Bi-CR. Some of the results will be useful for obtaining variants of Bi-CR in chapter 5.

Let us redefine the nth residual vector and the search direction as

$$\boldsymbol{r}_n := R_n(A)\boldsymbol{r}_0, \quad \boldsymbol{p}_n := P_n(A)\boldsymbol{r}_0, \tag{3.21}$$

where R_n and P_n denote the following coupled two-term recurrences:

$$R_0(\lambda) := 1, \quad P_0(\lambda) := 1, \tag{3.22}$$

$$R_n(\lambda) := R_{n-1}(\lambda) - \alpha_{n-1}\lambda P_{n-1}(\lambda), \tag{3.23}$$

$$P_n(\lambda) := R_n(\lambda) + \beta_{n-1}P_{n-1}(\lambda), \quad n = 1, 2, \ldots, \tag{3.24}$$

where, at this point, α_{n-1} and β_{n-1} are free parameters.

From here to the end of this subsection, we show that Algorithm 3.4 is obtained by using (3.21) and the following conditions:

$$\boldsymbol{r}_n \perp A^{\mathrm{H}}K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*) \quad \text{and} \quad A\boldsymbol{p}_n \perp A^{\mathrm{H}}K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*), \tag{3.25}$$

which imply the biorthogonality properties (3.13)-(3.14). It follows from the relation (3.21)-(3.24) that $\boldsymbol{r}_n$ and $\boldsymbol{p}_n$ are updated as follows:

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \alpha_{n-1}A\boldsymbol{p}_{n-1}, \tag{3.26}$$

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}. \tag{3.27}$$

Since the parameters α_n and β_n in (3.26) and (3.27) are not determined, we first give an auxiliary formula for α_{n-1}. From (3.26), the inner product of $(A^{\mathrm{H}})^n\boldsymbol{r}_0^*$ and $\boldsymbol{r}_n$ is computed as

$$((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, \boldsymbol{r}_n) = ((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, \boldsymbol{r}_{n-1}) - \alpha_{n-1}((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1}). \tag{3.28}$$

It follows from $(A^{\mathrm{H}})^n\boldsymbol{r}_0^* \in A^{\mathrm{H}}K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$ and the conditions (3.25) that $((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, \boldsymbol{r}_n) = 0$. Thus from (3.28) it follows that

$$\alpha_{n-1} = \frac{((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, \boldsymbol{r}_{n-1})}{((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1})}. \tag{3.29}$$

Next, we give an auxiliary formula for β_{n-1}. From (3.27), the inner product of $(A^{\mathrm{H}})^n\boldsymbol{r}_0^*$ and $A\boldsymbol{p}_n$ is computed as

$$((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, A\boldsymbol{p}_n) = ((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, A\boldsymbol{r}_n) + \beta_{n-1}((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1}). \tag{3.30}$$

From the conditions (3.25), we obtain $((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{p}_n) = 0$. Thus from (3.30), it follows that

$$\beta_{n-1} = -\frac{((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{r}_n)}{((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1})}.$$

Moreover, from formula (3.29) β_{n-1} can be written as follows:

$$\beta_{n-1} = -\alpha_{n-1}\frac{((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{r}_n)}{((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, \boldsymbol{r}_{n-1})}. \tag{3.31}$$

Here, let us give practical formulas for α_{n-1} and β_{n-1} using the following auxiliary vectors:

$$\boldsymbol{r}_n^* := \bar{R}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, \quad \boldsymbol{p}_n^* := \bar{P}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*. \tag{3.32}$$

It follows from the relation (3.23)-(3.24) that $\boldsymbol{r}_n^*$ and $\boldsymbol{p}_n^*$ in (3.32) are updated as follows:

$$\boldsymbol{r}_n^* = \boldsymbol{r}_{n-1}^* - \bar{\alpha}_{n-1}A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, \tag{3.33}$$

$$\boldsymbol{p}_n^* = \boldsymbol{r}_n^* + \bar{\beta}_{n-1}\boldsymbol{p}_{n-1}^*. \tag{3.34}$$

Note that the auxiliary vectors $\boldsymbol{r}_{n-1}^*$, $\boldsymbol{p}_{n-1}^*$ can be written as

$$\boldsymbol{r}_{n-1}^* = \bar{R}_{n-1}(A^{\mathrm{T}})\boldsymbol{r}_0^* = \ c_{n-1}(A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^* + \boldsymbol{z}_1, \quad \boldsymbol{z}_1 \in K_{n-1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*), \tag{3.35}$$

$$\boldsymbol{p}_{n-1}^* = \bar{P}_{n-1}(A^{\mathrm{T}})\boldsymbol{r}_0^* = \ c_{n-1}(A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^* + \boldsymbol{z}_2, \quad \boldsymbol{z}_2 \in K_{n-1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*), \tag{3.36}$$

where $c_{n-1} = (-1)^{n-1}\prod_{i=0}^{n-2}\bar{\alpha}_i$. Then, it follows from (3.35) and (3.36) that (3.29) satisfies

$$\alpha_{n-1} = \frac{(A^{\mathrm{H}}\boldsymbol{r}_{n-1}^*, \boldsymbol{r}_{n-1}) - (A^{\mathrm{H}}\boldsymbol{z}_1, \boldsymbol{r}_{n-1})}{(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_{n-1}) - (A^{\mathrm{H}}\boldsymbol{z}_2, A\boldsymbol{p}_{n-1})}.$$

Since $(A^{\mathrm{H}}\boldsymbol{z}_1, \boldsymbol{r}_{n-1}) = (A^{\mathrm{H}}\boldsymbol{z}_2, A\boldsymbol{p}_{n-1}) = 0$ by the conditions (3.25), the formula for α_{n-1} is given by

$$\alpha_{n-1} = \frac{(A^{\mathrm{H}}\boldsymbol{r}_{n-1}^*, \boldsymbol{r}_{n-1})}{(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_{n-1})}. \tag{3.37}$$

Similarly, from (3.35) and the conditions (3.25), the auxiliary formula (3.31) is written as

$$\beta_{n-1} = \frac{(A^{\mathrm{H}}\boldsymbol{r}_n^*, \boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{r}_{n-1}^*, \boldsymbol{r}_{n-1})}. \tag{3.38}$$

Compared with α_{n-1} and β_{n-1} in Algorithm 3.4, we rewrite (3.37) and (3.38) in the following form:

$$\alpha_{n-1} = \frac{(\boldsymbol{r}_{n-1}^*, A\boldsymbol{r}_{n-1})}{(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_{n-1})}, \quad \beta_{n-1} = \frac{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(\boldsymbol{r}_{n-1}^*, A\boldsymbol{r}_{n-1})}. \tag{3.39}$$

Finally, we give an update formula for the approximate solution $\boldsymbol{x}_n$. From the relation (3.26) and recalling $\boldsymbol{r}_n = \boldsymbol{b} - A\boldsymbol{x}_n$, it follows that

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \alpha_{n-1}\boldsymbol{p}_{n-1}. \tag{3.40}$$

From the recurrence relations (3.26), (3.27), (3.33), (3.34), the formulas of (3.39), and the approximate solution (3.40), we obtain Algorithm 3.4.

A derivation based on an A-biorthogonalization Process

We give another derivation of Bi-CR. First, we introduce an A-biorthogonalization process. Then, we show that the Bi-CR method is also derived from the process. Similar to the bi-Lanczos process, an A-biorthogonalization process of $K_n(A, \boldsymbol{r}_0)$ and $K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$ is given below.

Algorithm 3.6: A-biorthogonalization process

set $\boldsymbol{v}_0 = \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \qquad \boldsymbol{v}_0^* = \boldsymbol{r}_0^* = \boldsymbol{b}^* - A^{\mathrm{H}}\boldsymbol{x}_0^*,$

set $t_{0,0} = (A^{\mathrm{H}}\boldsymbol{v}_0^*, A\boldsymbol{v}_0)/(\boldsymbol{v}_0^*, A\boldsymbol{v}_0),$

(3.41) set $\boldsymbol{v}_1 = -\alpha_0(A\boldsymbol{v}_0 - t_{0,0}\boldsymbol{v}_0),\ \boldsymbol{v}_1^* = -\alpha_0(A^{\mathrm{H}}\boldsymbol{v}_0^* - \bar{t}_{0,0}\boldsymbol{v}_0^*),$

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

$$t_{n-1,n} = (A^{\mathrm{H}}\boldsymbol{v}_{n-1}^*, A\boldsymbol{v}_n)/(\boldsymbol{v}_{n-1}^*, A\boldsymbol{v}_{n-1}),$$

$$t_{n,n} = (A^{\mathrm{H}}\boldsymbol{v}_n^*, A\boldsymbol{v}_n)/(\boldsymbol{v}_n^*, A\boldsymbol{v}_n),$$

(3.42) $$\boldsymbol{v}_{n+1} = -\alpha_n(A\boldsymbol{v}_n - t_{n,n}\boldsymbol{v}_n - t_{n-1,n}\boldsymbol{v}_{n-1}),$$

(3.43) $$\boldsymbol{v}_{n+1}^* = -\bar{\alpha}_n(A^{\mathrm{H}}\boldsymbol{v}_n^* - \bar{t}_{n,n}\boldsymbol{v}_n^* - \bar{t}_{n-1,n}\boldsymbol{v}_{n-1}^*).$$

end

where the scalar sequences $\alpha_0, \ldots, \alpha_n$ are not determined at this point, and these are used as a condition for deriving approximate solutions from residual vectors. From Algorithm 3.6 it is clear that the vector sequences $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$ and $\boldsymbol{v}_0^*, \boldsymbol{v}_1^*, \ldots$ satisfy the following A-biorthogonality property:

$$(\boldsymbol{v}_i^*, A\boldsymbol{v}_j) = 0 \quad \text{for} \quad i \neq j.$$

Now, we consider generating the nth approximate solution $\boldsymbol{x}_n$ for $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x}_n^*$ for $A^{\mathrm{H}}\boldsymbol{x}^* = \boldsymbol{b}^*$ over the following affine spaces:

(3.44)
$$\begin{aligned} \boldsymbol{x}_n &:= \boldsymbol{x}_0 + \boldsymbol{z}_n, \quad \boldsymbol{z}_n \in K_n(A, \boldsymbol{r}_0), \\ \boldsymbol{x}_n^* &:= \boldsymbol{x}_0^* + \boldsymbol{z}_n^*, \quad \boldsymbol{z}_n^* \in K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*). \end{aligned}$$

Then, the nth corresponding residual vectors $\boldsymbol{r}_n$ and $\boldsymbol{r}_n^*$ are given by

(3.45) $$\boldsymbol{r}_n := \boldsymbol{b} - A\boldsymbol{x}_n = \boldsymbol{r}_0 - A\boldsymbol{z}_n, \qquad \boldsymbol{r}_n \in K_{n+1}(A, \boldsymbol{r}_0),$$

(3.46) $$\boldsymbol{r}_n^* := \boldsymbol{b}^* - A^{\mathrm{H}}\boldsymbol{x}_n^* = \boldsymbol{r}_0^* - A^{\mathrm{H}}\boldsymbol{z}_n^*, \quad \boldsymbol{r}_n^* \in K_{n+1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*).$$

Since $\boldsymbol{v}_n$ of (3.42) is in $K_{n+1}(A, \boldsymbol{r}_0)$ and $\boldsymbol{v}_n^*$ of (3.43) is in $K_{n+1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$, we can use $\boldsymbol{v}_n$, $\boldsymbol{v}_n^*$ as the nth residual vectors, $\boldsymbol{r}_n$ of (3.45) and $\boldsymbol{r}_n^*$ of (3.46), respectively. Then, it follows from the formulas (3.42) and (3.43) that we obtain

(3.47) $$\boldsymbol{r}_{n+1} = -\alpha_n(A\boldsymbol{r}_n - t_{n,n}\boldsymbol{r}_n - t_{n-1,n}\boldsymbol{r}_{n-1}),$$

(3.48) $$\boldsymbol{r}_{n+1}^* = -\bar{\alpha}_n(A^{\mathrm{H}}\boldsymbol{r}_n^* - \bar{t}_{n,n}\boldsymbol{r}_n^* - \bar{t}_{n-1,n}\boldsymbol{r}_{n-1}^*),$$

where $t_{n-1,n} = (A^{\mathrm{H}}\boldsymbol{r}_{n-1}^*, A\boldsymbol{r}_n)/(\boldsymbol{r}_{n-1}^*, A\boldsymbol{r}_{n-1})$ and $t_{n,n} = (A^{\mathrm{H}}\boldsymbol{r}_n^*, A\boldsymbol{r}_n)/(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)$. Substituting (3.45) and (3.46) into (3.47) and (3.48) respectively, the residual vectors are computed as follows:

(3.49) $$\boldsymbol{r}_{n+1} = \alpha_n(t_{n,n} + t_{n-1,n})\boldsymbol{r}_0 - \alpha_n A(\boldsymbol{r}_n + t_{n,n}\boldsymbol{z}_n + t_{n-1,n}\boldsymbol{z}_{n-1}),$$

(3.50) $$\boldsymbol{r}_{n+1}^* = \bar{\alpha}_n(\bar{t}_{n,n} + \bar{t}_{n-1,n})\boldsymbol{r}_0^* - \bar{\alpha}_n A^{\mathrm{H}}(\boldsymbol{r}_n^* + \bar{t}_{n,n}\boldsymbol{z}_n^* + \bar{t}_{n-1,n}\boldsymbol{z}_{n-1}^*).$$

Comparing the coefficient of $\boldsymbol{r}_0$ in (3.45) with the one of $\boldsymbol{r}_0$ in (3.49), α_n must satisfy

$$\alpha_n(t_{n,n} + t_{n-1,n}) = 1 \quad \text{for } n \geq 1 \tag{3.51}$$

to derive an update formula of the approximate solution $\boldsymbol{x}_n$ from the information of $\boldsymbol{r}_n$. In addition, from (3.50) and (3.51) we can also derive the update formula of $\boldsymbol{x}_n^*$. However, we omit the formula.

For the special case $n = 0$, from (3.41), (3.45), (3.46) it follows that

$$\boldsymbol{r}_1 = \boldsymbol{r}_0 - A\boldsymbol{z}_1 = -\alpha_0(A\boldsymbol{r}_0 - t_{0,0}\boldsymbol{r}_0), \tag{3.52}$$

$$\boldsymbol{r}_1^* = \boldsymbol{r}_0^* - A^{\mathrm{H}}\boldsymbol{z}_1^* = -\bar{\alpha}_0(A^{\mathrm{H}}\boldsymbol{r}_0^* - \bar{t}_{0,0}\boldsymbol{r}_0^*), \tag{3.53}$$

and thus α_0 satisfies

$$\alpha_0 = \frac{1}{t_{0,0}} = \frac{(\boldsymbol{r}_0^*, A\boldsymbol{r}_0)}{(A^{\mathrm{H}}\boldsymbol{r}_0^*, A\boldsymbol{r}_0)}. \tag{3.54}$$

Now, we can compute the residual vectors of Bi-CR by using recurrence relations (3.47)-(3.48), and (3.51)-(3.54).

Here, let us give a property of this process. From Algorithm 3.6 it is clear that the sequences $\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots$ and $\boldsymbol{r}_0^*, \boldsymbol{r}_1^*, \ldots$ satisfy the following A-biorthogonality property:

$$(\boldsymbol{r}_i^*, A\boldsymbol{r}_j) = 0 \quad \text{for} \quad i \neq j. \tag{3.55}$$

Next, we consider an efficient way to compute the residual vectors $\boldsymbol{r}_n$ by introducing two auxiliary vectors, and we give a formula for updating the approximate solution $\boldsymbol{x}_n$. Let us define auxiliary vectors $\boldsymbol{p}_n$, $\boldsymbol{p}_n^*$ as

$$\boldsymbol{p}_n := \frac{\boldsymbol{z}_{n+1} - \boldsymbol{z}_n}{\alpha_n}, \quad \boldsymbol{p}_n^* := \frac{\boldsymbol{z}_{n+1}^* - \boldsymbol{z}_n^*}{\bar{\alpha}_n}. \tag{3.56}$$

Multiplying left and right side of (3.56) by $-\alpha_n A$ and $-\bar{\alpha}_n A^{\mathrm{H}}$ respectively, and from the definitions (3.45) and (3.46) it follows that

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n, \quad \boldsymbol{r}_{n+1}^* = \boldsymbol{r}_n^* - \bar{\alpha}_n A^{\mathrm{H}}\boldsymbol{p}_n^*. \tag{3.57}$$

Substituting the relation (3.51) into $t_{n,n}$ of (3.47) and (3.48), then the recurrence relations (3.47) and (3.48) can be written as

$$\boldsymbol{r}_{n+1} - \boldsymbol{r}_n = -\alpha_n A\boldsymbol{r}_n - \alpha_n t_{n-1,n}(\boldsymbol{r}_n - \boldsymbol{r}_{n-1}), \tag{3.58}$$

$$\boldsymbol{r}_{n+1}^* - \boldsymbol{r}_n^* = -\bar{\alpha_n} A^{\mathrm{H}}\boldsymbol{r}_n^* - \bar{\alpha}_n \bar{t}_{n-1,n}(\boldsymbol{r}_n^* - \boldsymbol{r}_{n-1}^*). \tag{3.59}$$

From (3.57) it follows that $\boldsymbol{r}_{n+1} - \boldsymbol{r}_n = -\alpha_n A\boldsymbol{p}_n$, $\boldsymbol{r}_{n+1}^* - \boldsymbol{r}_n^* = -\bar{\alpha}_n A^{\mathrm{H}}\boldsymbol{p}_n^*$, and thus substituting these relations into (3.58) and (3.59) respectively, we obtain

$$\boldsymbol{p}_n = \boldsymbol{r}_n - \alpha_{n-1} t_{n-1,n}\boldsymbol{p}_{n-1}, \quad \boldsymbol{p}_n^* = \boldsymbol{r}_n^* - \bar{\alpha}_{n-1}\bar{t}_{n-1,n}\boldsymbol{p}_{n-1}^*. \tag{3.60}$$

From the inner product of $A^{\mathrm{H}}\boldsymbol{r}_{n+1}^*$ and (3.47), using (3.55) and the relation $(A^{\mathrm{H}}\boldsymbol{r}_n^*, A\boldsymbol{r}_{n-1})$ $=(A^{\mathrm{H}}\boldsymbol{r}_{n-1}^*, A\boldsymbol{r}_n)$, it follows that

$$\alpha_n = -\frac{(\boldsymbol{r}_{n+1}^*, A\boldsymbol{r}_{n+1})}{(A^{\mathrm{H}}\boldsymbol{r}_{n+1}^*, A\boldsymbol{r}_n)}. \tag{3.61}$$

Substituting (3.61) into (3.60), we obtain

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \quad \boldsymbol{p}_n^* = \boldsymbol{r}_n^* + \bar{\beta}_{n-1}\boldsymbol{p}_{n-1}^*, \tag{3.62}$$

where $\beta_{n-1} = (\boldsymbol{r}_n^*, A\boldsymbol{r}_n)/(\boldsymbol{r}_{n-1}^*, A\boldsymbol{r}_{n-1})$. Comparing (3.52) and (3.53) with (3.62), we obtain $\boldsymbol{p}_0 = \boldsymbol{r}_0$ and $\boldsymbol{p}_0^* = \boldsymbol{r}_0^*$. Here, we show that α_n satisfies the following relation:

$$\alpha_n = \frac{1}{t_{n,n} + t_{n-1,n}} = \frac{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{p}_n^*, A\boldsymbol{p}_n)}. \tag{3.63}$$

From (3.51) it is clear that the first equation of (3.63) is satisfied. Next, we show the second equation of (3.63). From (3.60) it follows that

$$\begin{aligned}
\frac{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{p}_n^*, A\boldsymbol{p}_n)} &= \frac{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{r}_n^*, A\boldsymbol{p}_n) + \beta_{n-1}(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_n)} \\
&= \frac{1}{\frac{(A^{\mathrm{H}}\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)} + \beta_{n-1}\frac{(A^{\mathrm{H}}\boldsymbol{r}_n^*, A\boldsymbol{r}_{n-1})}{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)} + \beta_{n-1}\frac{(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_n)}{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}} \\
&= \frac{1}{t_{n,n} + t_{n-1,n}},
\end{aligned}$$

where we used the relation $(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_n) = 0$ because

$$(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_n) = \frac{1}{\alpha_n}(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, \boldsymbol{r}_n) - \frac{1}{\alpha_n}(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, \boldsymbol{r}_{n+1}) = 0,$$

where $(A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, \boldsymbol{r}_n) = (A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, \boldsymbol{r}_{n+1}) = 0$ from $\boldsymbol{p}_{n-1}^* \in \mathrm{span}\{\boldsymbol{r}_0^*, \ldots, \boldsymbol{r}_{n-1}^*\}$ and the A-biorthogonality property (3.55). For computing the approximate solutions, from the relation between (3.44) and (3.56) it follows that

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n. \tag{3.64}$$

From the relations (3.52)-(3.54), (3.57), and (3.62)-(3.64), we obtain the Bi-CR method.

A derivation of Bi-CR in matrix form

Here, we derive the Bi-CR method by using a matrix form of Algorithm 3.6. Let $\boldsymbol{r}_0$ be the initial residual vector. Then, from Algorithm 3.6 A-biorthogonal basis of the Krylov subspace $K_n(A, \boldsymbol{r}_0)$ and $K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$ can be expressed in the following matrix form:

$$A\underbrace{[\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n-1}]}_{R_n} = \underbrace{[\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n-1}, \boldsymbol{r}_n]}_{R_{n+1}} \begin{pmatrix} t_{0,0} & t_{0,1} & & \\ -\alpha_0^{-1} & t_{1,1} & \ddots & \\ & -\alpha_1^{-1} & \ddots & t_{n-2,n-1} \\ & & \ddots & t_{n-1,n-1} \\ & & & -\alpha_{n-1}^{-1} \end{pmatrix},$$

where $t_{k-1,k} = \frac{(A^{\mathrm{H}}\boldsymbol{r}_{k-1}^*, A\boldsymbol{r}_k)}{(\boldsymbol{r}_{k-1}^*, A\boldsymbol{r}_{k-1})}$ and $t_{k,k} = \frac{(A^{\mathrm{H}}\boldsymbol{r}_k^*, A\boldsymbol{r}_k)}{(\boldsymbol{r}_k^*, A\boldsymbol{r}_k)}$. From the above matrix form, we have

$$AR_n = R_{n+1}T_{n+1,n} \quad \text{and} \quad A^{\mathrm{H}}R_n^* = R_{n+1}^*\bar{T}_{n+1,n}. \tag{3.65}$$

Since the matrix R_n is generated via Algorithm 3.6, it is clear that $R_n^{*H} A R_n = D_n$, where D_n is a diagonal matrix of order n. Since the scalar sequence $\alpha_0, \ldots, \alpha_{n-1}$ still remains unknown. We show that if we determine the parameters such that approximate solutions $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}$ can be extracted from information of $\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{n-1}$, then we obtain the desired algorithm of the Bi-CR method. Let X_n be a matrix with columns $[\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}]$ and $\mathbf{1}$ be $(1, \ldots, 1)^{\mathrm{T}}$. Then, we can connect R_n and X_n in the following form:

$$R_n := \boldsymbol{b}\mathbf{1}_n^{\mathrm{T}} - AX_n = [\boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{b} - A\boldsymbol{x}_1, \ldots, \boldsymbol{b} - A\boldsymbol{x}_{n-1}]. \tag{3.66}$$

From the above definition, we can regard the ith column of R_n as the ith residual vector. Substituting (3.66) into (3.65), it follows that

$$AR_n = (\boldsymbol{b}\mathbf{1}_{n+1}^{\mathrm{T}} - AX_{n+1})T_{n+1,n}.$$

Then, we obtain

$$R_n = (\boldsymbol{x}\mathbf{1}_{n+1}^{\mathrm{T}} - X_{n+1})T_{n+1,n},$$

where $\boldsymbol{x}$ is the exact solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$. Hence, to extract X_n from R_n, we need the following condition:

$$\boldsymbol{x}\mathbf{1}_{n+1}^{\mathrm{T}} T_{n+1,n} = O_n.$$

This leads to

$$\begin{aligned} \alpha_0^{-1} &= t_{0,0}, \\ \alpha_k^{-1} &= t_{k,k} + t_{k-1,k}, \quad 1 \le k \le n-1. \end{aligned}$$

Substituting the above recurrence into $T_{n+1,n}$, we have

$$T_{n+1,n} = \begin{pmatrix} \alpha_0^{-1} & t_{0,1} & & \\ -\alpha_0^{-1} & \alpha_1^{-1} - t_{0,1} & \ddots & \\ & -\alpha_1^{-1} & \ddots & t_{n-2,n-1} \\ & & \ddots & \alpha_{n-1}^{-1} - t_{n-2,n-1} \\ & & & -\alpha_{n-1}^{-1} \end{pmatrix}.$$

Then, $T_{n+1,n}$ can be factorized as follows:

$$\begin{aligned} T_{n+1,n} &= \begin{pmatrix} 1 & & \\ -1 & \ddots & \\ & \ddots & 1 \\ & & -1 \end{pmatrix} \begin{pmatrix} \alpha_0^{-1} & & \\ & \ddots & \\ & & \alpha_{n-1}^{-1} \end{pmatrix} \begin{pmatrix} 1 & \alpha_0 t_{0,1} & & \\ & 1 & \ddots & \\ & & \ddots & \alpha_{n-2} t_{n-2,n-1} \\ & & & 1 \end{pmatrix} \\ &= B_{n+1,n}^{(L)} \Omega_n^{-1} B_n^{(U)}. \end{aligned}$$

From (3.65) and the above factorization, we obtain

$$AR_n = R_{n+1} T_{n+1,n} = R_{n+1} B_{n+1,n}^{(L)} \Omega_n^{-1} B_n^{(U)}, \tag{3.67}$$

$$A^{\mathrm{H}} R_n^* = R_{n+1}^* \bar{T}_{n+1,n} = R_{n+1}^* \bar{B}_{n+1,n}^{(L)} \bar{\Omega}_n^{-1} \bar{B}_n^{(U)}. \tag{3.68}$$

Here, we introduce $P_n := R_n(B_n^{(U)})^{-1}$ and $P_n^* := R_n^*(\bar{B}_n^{(U)})^{-1}$. Then from (3.67) and (3.68), we obtain

$$AP_n = AR_n(B_n^{(U)})^{-1} = R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}, \tag{3.69}$$
$$A^{\mathrm{H}}P_n^* = A^{\mathrm{H}}R_n^*(\bar{B}_n^{(U)})^{-1} = R_{n+1}^*\bar{B}_{n+1,n}^{(L)}\bar{\Omega}_n^{-1}.$$

The above matrix form are equivalent to the following recurrences:

$$\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_{k-1}A\boldsymbol{p}_{k-1}, \quad 1 \le k \le n, \tag{3.70}$$
$$\boldsymbol{r}_k^* = \boldsymbol{r}_{k-1} - \bar{\alpha}_{k-1}A^{\mathrm{H}}\boldsymbol{p}_{k-1}^*, \quad 1 \le k \le n. \tag{3.71}$$

From $P_n = R_n(B_n^{(U)})^{-1}$ and $P_n^* = R_n^*(\bar{B}_n^{(U)})^{-1}$, it follows that

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_{k-1}\boldsymbol{p}_{k-1}, \quad 1 \le k \le n, \tag{3.72}$$
$$\boldsymbol{p}_k^* = \boldsymbol{r}_k^* + \bar{\beta}_{k-1}\boldsymbol{p}_{k-1}^*, \quad 1 \le k \le n, \tag{3.73}$$

where $\beta_{k-1} := -\alpha_{k-1}t_{k-1,k}$. Here, we give the computational formula of approximate solution. From (3.69), it follows that

$$\begin{aligned} AP_n &= R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1} \\ &= (\boldsymbol{b}\mathbf{1}^{\mathrm{T}} - AX_{n+1})B_{n+1,n}^{(L)}\Omega_n^{-1} \\ &= -AX_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}. \end{aligned}$$

Thus, we obtain

$$P_n\Omega_n = X_{n+1}(-B_{n+1,n}^{(L)})$$

and it is equivalent to

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_{k-1}\boldsymbol{p}_{k-1}, \quad 1 \le k \le n. \tag{3.74}$$

Now, we give more practical computational formulas of α_k and β_k. From (3.65), it follows that

$$(A^{\mathrm{H}}R_n^*)^{\mathrm{H}}AR_n = R_n^{*\mathrm{H}}AR_{n+1}T_{n+1,n}.$$

From the $(k+1,k)$ entry of the above matrices, we have $(A^{\mathrm{H}}\boldsymbol{r}_k^*, A\boldsymbol{r}_{k-1}) = -\alpha_{k-1}^{-1}(\boldsymbol{r}_k^*, A\boldsymbol{r}_k)$. Thus from $\beta_{k-1} = -\alpha_{k-1}t_{k-1,k}$, we obtain

$$\begin{aligned} \beta_k &= -\alpha_k t_{k,k+1} \\ &= \frac{(\boldsymbol{r}_{k+1}^*, A\boldsymbol{r}_{k+1})}{(A^{\mathrm{H}}\boldsymbol{r}_{k+1}^*, A\boldsymbol{r}_k)} \cdot \frac{(A^{\mathrm{H}}\boldsymbol{r}_k^*, A\boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k^*, A\boldsymbol{r}_k)} \\ &= \frac{(\boldsymbol{r}_{k+1}^*, A\boldsymbol{r}_{k+1})}{(\boldsymbol{r}_k^*, A\boldsymbol{r}_k)}. \end{aligned} \tag{3.75}$$

We used the relation $(\boldsymbol{r}_k^*, A\boldsymbol{r}_{k+1}) = (\boldsymbol{r}_{k+1}^*, A\boldsymbol{r}_k)$ since $\boldsymbol{r}_n$ is written as the product of matrix polynomial and initial residual vector, *i.e.*,

$$\begin{aligned} (\boldsymbol{r}_k^*, A\boldsymbol{r}_{k+1}) &= (\bar{R}_k(A^{\mathrm{T}})\boldsymbol{r}_0^*, AR_{k+1}(A)\boldsymbol{r}_0) \\ &= (\bar{R}_{k+1}(A^{\mathrm{T}})\bar{R}_k(A^{\mathrm{T}})\boldsymbol{r}_0^*, A\boldsymbol{r}_0) \\ &= (\bar{R}_k(A^{\mathrm{T}})\bar{R}_{k+1}(A^{\mathrm{T}})\boldsymbol{r}_0^*, A\boldsymbol{r}_0) \\ &= (\bar{R}_{k+1}(A^{\mathrm{T}})\boldsymbol{r}_0^*, AR_k(A)\boldsymbol{r}_0) \\ &= (\boldsymbol{r}_{k+1}^*, A\boldsymbol{r}_k). \end{aligned}$$

From (3.69) and recalling $P_n^* = R_n^*(\bar{B}_n^{(U)})^{-1}$, it follow that

$$(A^{\mathrm{H}}P_n^*)^{\mathrm{H}}AP_n = (A^{\mathrm{H}}P_n^*)^{\mathrm{H}}R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1} = (\bar{B}_n^{(U)})^{-\mathrm{H}}R_n^{*H}AR_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}.$$

This leads to

$$(A^{\mathrm{H}}P_n^*)^{\mathrm{H}}AP_n = \begin{pmatrix} d_0 & & \\ * & \ddots & \\ * & * & d_{n-1} \end{pmatrix},$$

where $d_k = (\boldsymbol{r}_k^*, A\boldsymbol{r}_k)\alpha_k^{-1}$. Thus, we obtain

$$\alpha_k = \frac{(\boldsymbol{r}_k^*, A\boldsymbol{r}_k)}{(A^{\mathrm{H}}\boldsymbol{p}_k^*, A\boldsymbol{p}_k)}. \tag{3.76}$$

Moreover, since $(A^{\mathrm{H}}\boldsymbol{p}_i, A\boldsymbol{p}_j) = (A^{\mathrm{H}}\boldsymbol{p}_j, A\boldsymbol{p}_i)$, $A^{\mathrm{H}}\boldsymbol{p}_i^*$ and $A\boldsymbol{p}_j$ are orthogonal, *i.e.*,

$$(A^{\mathrm{H}}\boldsymbol{p}_i^*, A\boldsymbol{p}_j) = 0 \quad \text{for} \quad i \neq j.$$

From (3.70)-(3.76), we obtain the algorithm of Bi-CR.

3.4 Numerical experiments

In this section, we report the results of numerical experiments on a range of Matrix Market problems from the Harwell-Boeing collection [26], the NEP collection [3], the SPARSKIT collection [67], and Tim Davis's collection [21]. The iterative solvers used in the experiments are Bi-CG and Bi-CR, and we evaluate the two methods with respect to the number of iterations (Its), computational time (Time), and $\log_{10}$ of the true relative residual 2-norm (TRR) defined as $\log_{10} \|\boldsymbol{b} - A\boldsymbol{x}_n\|/|\boldsymbol{b}\|$. All experiments were performed on an ALPHA work station with a 750MHz processor using double precision arithmetic. Codes were written in Fortran 77 and compiled with the optimization option -$O4$. In all cases the iteration was started with $\boldsymbol{x}_0 = \boldsymbol{0}$ and $\boldsymbol{r}_0^* = \boldsymbol{r}_0$ in both methods, the right-hand side $\boldsymbol{b}$ was chosen as a vector with random entries from -1 to 1, and the stopping criterion was $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq 10^{-12}$. The convergence histories show the number of iterations (on the horizontal axis) versus $\log_{10}$ of the relative residual 2-norm, $\log_{10} \|\boldsymbol{r}_n\|/\|\boldsymbol{b}\|$ (on the vertical axis).

Matrices in the experiments come from electronic circuit design (ADD20, ADD32, MEMPLUS), electrical engineering (BFW782A), finite element modeling (CAVITY05, CAV ITY10, FIDAP036), fluid dynamics (CDDE1, E20R0000, E30R0000), oil reservoir simulation (ORSIRR1, ORSIRR2, ORSREG1, SHERMAN1, SHERMAN5), partial differential equations (PDE2961), aeroelasticity (TOLS4000), and petroleum engineering (WATT1, WATT2).

◇ Comparison of Bi-CG and Bi-CR

We evaluate the performance of Bi-CG and Bi-CR with no preconditioning. The numerical results are shown in Table 3.2.

Looking at Its and Time, we see that Bi-CR required only about 90% of the iteration steps and computational time of Bi-CG in BFW782A, FIDAP036, TOLS4000, and WATT1.

Notably in WATT2, Bi-CR performed much better than Bi-CG in that Bi-CR required only about 46% of the iteration steps and computational time of Bi-CG. In other problems, Bi-CG and Bi-CR required almost the same number of iteration steps and computational time.

In terms of TRR, accuracy of both approximate solutions was worse than the stopping criterion in E30R0000, TOLS4000, and WATT2; however, Bi-CR generated better approximate solutions than Bi-CG in TOLS4000 and in WATT2.

Residual 2-norm histories of Bi-CG and Bi-CR for ADD20, E30R0000, and WATT2 are shown in Figs. 3.1, 3.2, and 3.3 respectively. We see from Fig. 3.1 and Fig. 3.2 that Bi-CG gives many peaks in the residual norm, whereas Bi-CR gives much smoother convergence behavior. In Fig. 3.3, Bi-CR gave much smoother convergence behavior and converged much faster than Bi-CG.

$\diamond$ Comparison of Bi-CG and Bi-CR with $ILU(0)$ preconditioning

We evaluate the performance of Bi-CG and Bi-CR with $ILU(0)$ preconditioning. The numerical results are shown in Table 3.3, where the result for TOLS4000 is not listed since $ILU(0)$ caused breakdown.

With respect to Its and Time, Bi-CR required about 90% of the iteration steps and computational time of Bi-CG in CAVITY10, E20R0000, and E30R0000. There was little difference in the performance of Bi-CG and Bi-CR in other problems except TOLS4000, since the preconditioner was quite effective in improving the convergence behavior.

In the terms of TRR, the accuracy of both approximate solutions was worse than the stopping criterion in E30R0000 and WATT2. In other problems, the two methods generated almost the same accuracy of the approximate solutions as one of the stopping criterion.

Residual 2-norm histories of Bi-CG and Bi-CR with $ILU(0)$ preconditioning for ADD20, E30R0000, and WATT2 are shown in Figs. 3.4, 3.5, and 3.6 respectively. As seen in Fig. 3.4, the convergence behavior with Bi-CG was jagged, whereas that with Bi-CR was smoother. These histories were similar to the ones without preconditioning in Fig. 3.1. We see from Fig. 3.5 that Bi-CR showed fairly attractive convergence behavior in the last phase; however the smoothness that we see in Fig. 3.2 was not observed. In Fig. 3.6, the two methods showed similar convergence behavior.

3.5 Concluding remarks

Based on H. A. van der Vorst's derivation of Bi-CG, we extended CR to nonsymmetric linear systems without loss of its short-term recurrences. Then, we discussed some properties of Bi-CR and found that iterates $\boldsymbol{r}_i^*$ and $\boldsymbol{r}_j$ are A-orthogonal. On the other hand, we also gave other derivations of Bi-CR based on an A-biorthogonalization process, and its matrix form.

From the numerical experiments we have learned that Bi-CR tends to show smoother convergence behavior and often converges faster than Bi-CG. Since Bi-CG is a basic solver for Bi-CGSTAB(ℓ) and GPBi-CG, it may take the place of Bi-CG for attractive variants, *i.e.*, similar to product-type methods based on Bi-CG, we can consider product-type methods based on Bi-CR. This framework will be given in chapter 5.

In the next chapter, we will consider the Bi-CR method specialized to the case where the coefficient matrix is complex symmetric.

Table 3.2. Matrices, their sizes (N), and numerical results of Bi-CG and Bi-CR without preconditioning.

		Its		Time [sec]		TRR	
Matrix	N	Bi-CG	Bi-CR	Bi-CG	Bi-CR	Bi-CG	Bi-CR
ADD20	2395	665	644	0.90	0.88	-11.98	-11.97
ADD32	4960	130	129	0.39	0.39	-12.08	-12.12
BFW782A	782	435	390	0.18	0.17	-11.50	-11.52
CAVITY05	1182	801	764	1.09	1.05	-12.04	-12.04
CAVITY10	2597	1236	1155	3.70	3.53	-12.50	-12.03
CDDE1	961	173	173	0.09	0.09	-12.07	-12.05
E20R0000	4241	1667	1597	9.01	8.74	-11.68	-11.69
E30R0000	9661	2565	2526	36.83	36.74	-10.21	-10.25
FIDAP036	3079	7263	6410	18.52	16.97	-11.48	-11.41
MEMPLUS	17758	1913	1849	21.74	21.57	-11.73	-11.75
ORSIRR1	1030	1646	1599	1.19	1.18	-11.87	-11.89
ORSIRR2	886	1192	1191	0.75	0.77	-12.11	-11.97
ORSREG1	2205	630	582	1.12	1.05	-12.00	-11.95
PDE2961	2961	335	319	0.60	0.58	-12.00	-12.55
SHERMAN1	1000	709	704	0.31	0.31	-12.01	-12.01
SHERMAN5	3312	2668	2664	3.83	3.88	-11.73	-11.02
TOLS4000	4000	6248	5577	5.72	5.19	-9.00	-10.10
WATT1	1856	544	492	0.80	0.74	-12.05	-12.01
WATT2	1856	1489	689	2.14	1.01	-6.78	-7.74

Table 3.3. Matrices, their sizes (N), and numerical results of Bi-CG and Bi-CR with $ILU(0)$ preconditioning.

		Its		Time [sec]		TRR	
Matrix	N	Bi-CG	Bi-CR	Bi-CG	Bi-CR	Bi-CG	Bi-CR
ADD20	2395	274	274	0.72	0.73	-12.10	-12.14
ADD32	4960	63	63	0.30	0.31	-12.05	-12.17
BFW782A	782	128	118	0.14	0.13	-11.23	-11.26
CAVITY05	1182	168	168	0.48	0.49	-11.30	-11.36
CAVITY10	2597	275	238	1.92	1.72	-12.29	-11.95
CDDE1	961	55	54	0.04	0.04	-12.14	-12.12
E20R0000	4241	185	164	2.47	2.22	-12.16	-12.07
E30R0000	9661	303	266	11.10	9.90	-11.03	-11.06
FIDAP036	3079	177	177	1.07	1.09	-12.51	-12.00
MEMPLUS	17758	492	488	12.07	12.18	-11.90	-11.89
ORSIRR1	1030	76	72	0.10	0.10	-12.23	-11.99
ORSIRR2	886	77	73	0.09	0.09	-12.17	-12.05
ORSREG1	2205	95	93	0.29	0.29	-12.03	-12.01
PDE2961	2961	75	77	0.23	0.24	-12.00	-13.06
SHERMAN1	1000	63	62	0.04	0.04	-12.43	-12.10
SHERMAN5	3312	43	43	0.12	0.12	-12.17	-12.26
WATT1	1856	60	59	0.14	0.14	-12.25	-12.06
WATT2	1856	112	110	0.28	0.28	-8.09	-8.14

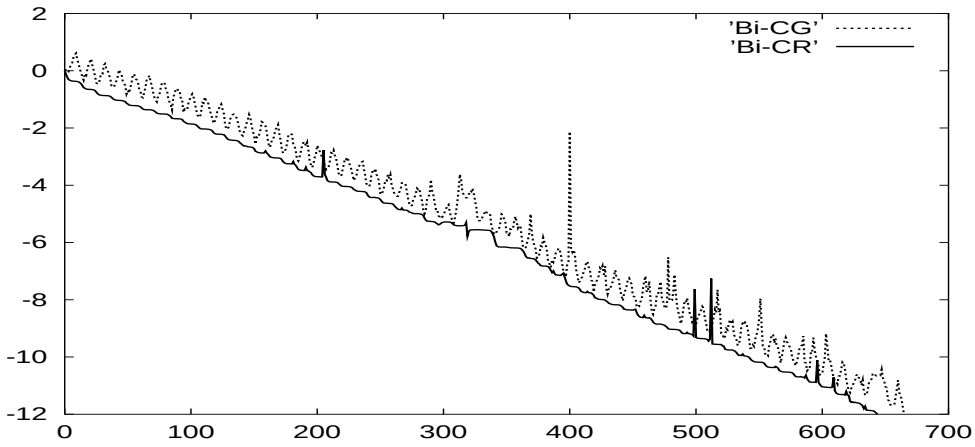

Figure 3.1: Residual 2-norm histories of Bi-CG and Bi-CR for ADD20.

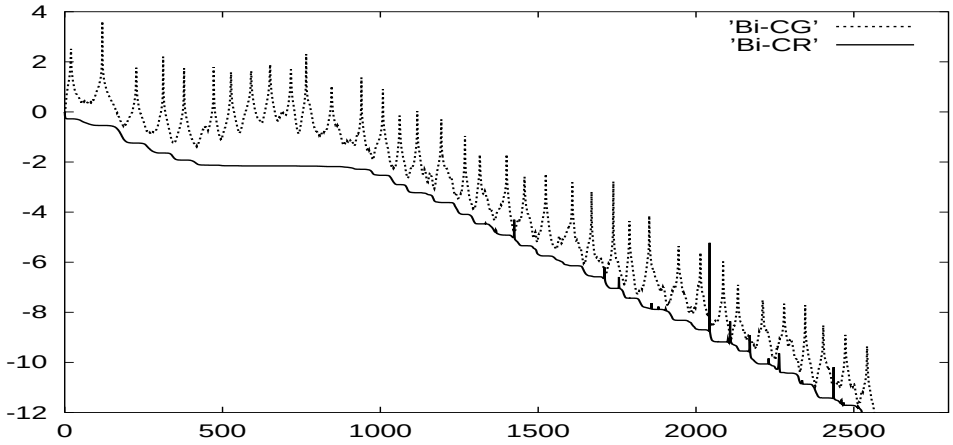

Figure 3.2: Residual 2-norm histories of Bi-CG and Bi-CR for E30R0000.

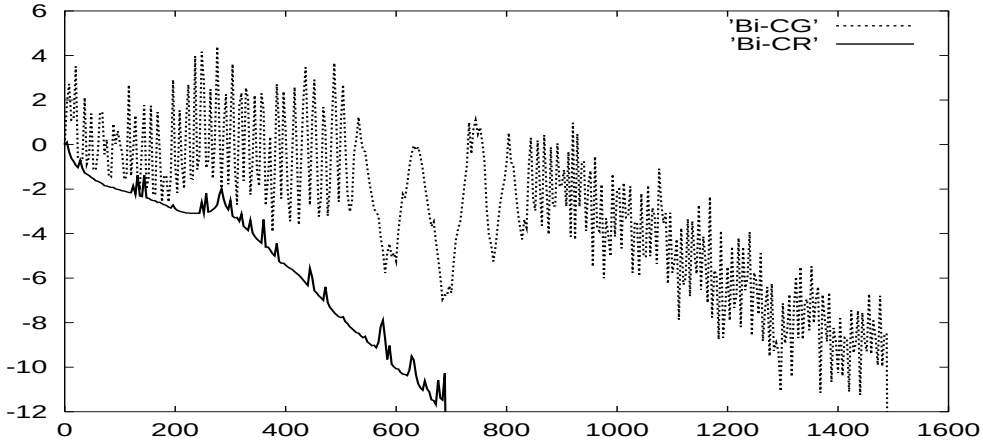

Figure 3.3: Residual 2-norm histories of Bi-CG and Bi-CR for WATT2.

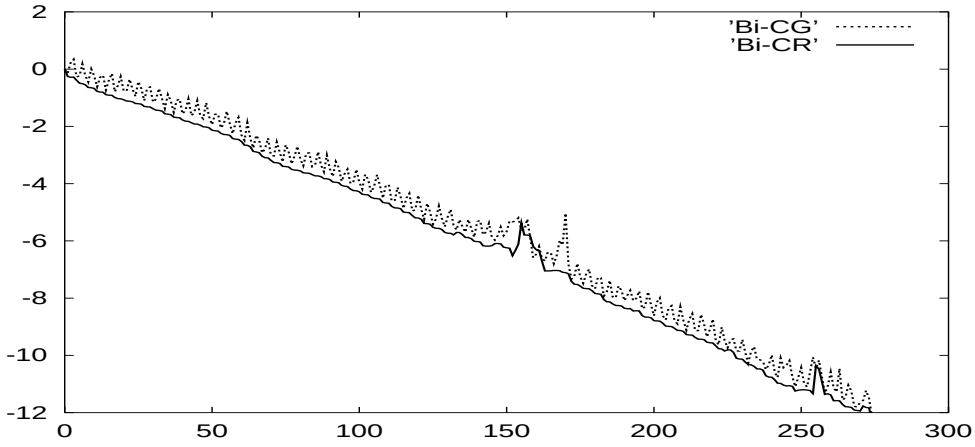

Figure 3.4: Residual 2-norm histories of Bi-CG and Bi-CR with $ILU(0)$ for ADD20.

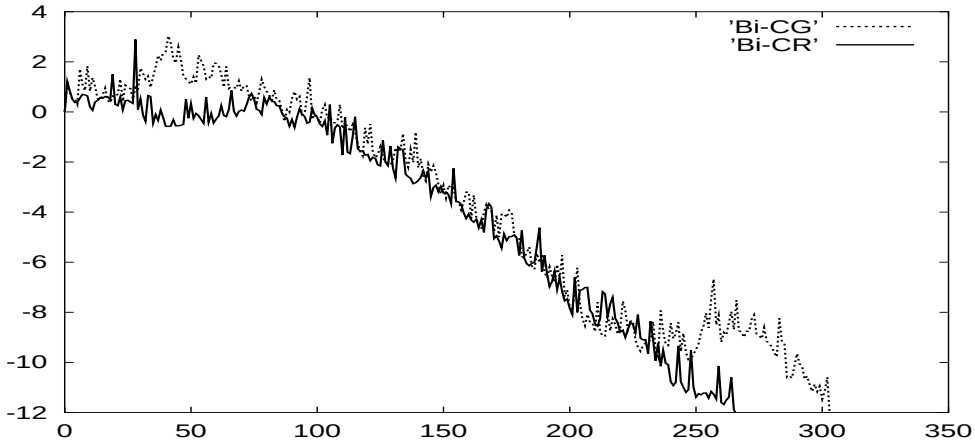

Figure 3.5: Residual 2-norm histories of Bi-CG and Bi-CR with $ILU(0)$ for E30R0000.

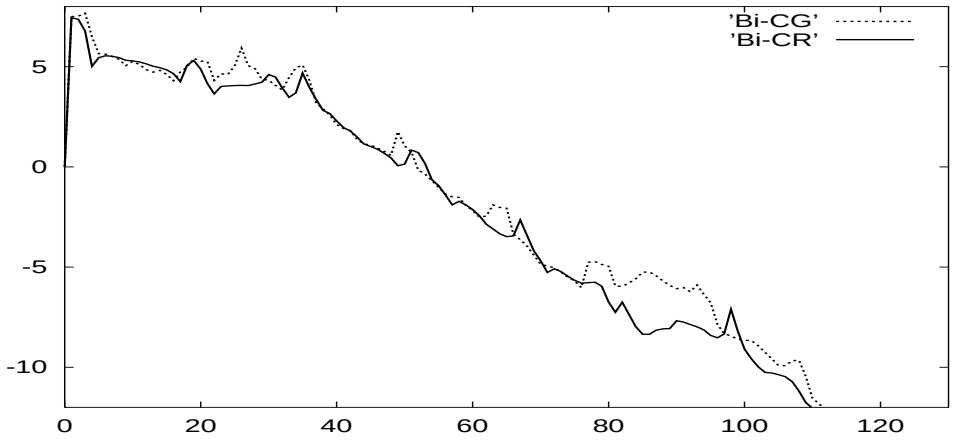

Figure 3.6: Residual 2-norm histories of Bi-CG and Bi-CR with $ILU(0)$ for WATT2.

Chapter 4

COCR: a conjugate orthogonal conjugate residual method

4.1 Introduction

We consider the solution of nonsingular complex symmetric linear systems of the form $A\boldsymbol{x} = \boldsymbol{b}$, where A is an $N \times N$ non-Hermitian but symmetric matrix ($A \neq \bar{A}^{\mathrm{T}}$, $A = A^{\mathrm{T}}$). Such systems arise in many important applications such as numerical computations in quantum chemistry, eddy current problems, and numerical solutions of the complex Helmholtz equation. Hence, there is a strong need for the fast solution of complex symmetric linear systems. For solving such systems efficiently, van der Vorst and Melissen [88] proposed the Conjugate Orthogonal Conjugate Gradient (COCG) method, which is regarded as an extension of the Conjugate Gradient (CG) method [53]. QMR_SYM [41], CSYM [14], and Bi-CGCR [16] are also useful Krylov subspace methods. QMR_SYM (QMR hereafter) is derived from the complex symmetric Lanczos algorithm, CSYM is obtained from the idea of QMR and tridiagonalization of A by Householder reflections, and Bi-CGCR is derived from a particular case in Bi-CG [56, 35] for solving non-Hermitian linear systems. These methods are often numerically robuster than COCG. However, these implementations are more complicated.

In this chapter, we extend the Conjugate Residual (CR) method [82] to complex symmetric linear systems based on an observation of derivations of CG, CR, and COCG. Since CR satisfies a minimal residual property, the extended algorithm, named COCR, can be expected to give smoother convergence behavior than COCG in the residual norm. From a more general point of view, the algorithm of COCR is also obtained from a special case of Bi-CR which is given in the previous chapter, and this is similar to the relation between COCG and Bi-CG.

This chapter is organized as follows: in the next section, first, we observe a way to derive the algorithms of CG, CR, and COCG. Second, we derive COCR from the observation, and its orthogonality properties are discussed. We also give other derivations of COCR based on a conjugate A-orthogonalization process. In §4.3, we report the results of some numerical examples. Finally, we make some concluding remarks in §4.4.

4.2 An extension of CR to complex symmetric linear systems

4.2.1 An observation of deriving CG, CR, and COCG

In this subsection, we discuss a way to obtain CG, CR, and COCG. Let $\boldsymbol{x}_n$ be the nth approximate solution in the methods. Then, the corresponding nth residual vector $\boldsymbol{r}_n (:= \boldsymbol{b} - A\boldsymbol{x}_n)$ and search direction $\boldsymbol{p}_n$ are given by the following coupled two-term recurrences:

$$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \quad \boldsymbol{p}_0 = \boldsymbol{r}_0, \tag{4.1}$$

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \alpha_{n-1} A\boldsymbol{p}_{n-1}, \tag{4.2}$$

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \quad n = 1, 2, \ldots \tag{4.3}$$

The differences among the algorithms of three methods are computational formulas of α_{n-1} and β_{n-1} in the recurrences (4.2)-(4.3), and these parameters are determined by the following orthogonality conditions:

$$\boldsymbol{r}_n \perp W \quad \text{and} \quad A\boldsymbol{p}_n \perp W. \tag{4.4}$$

When A is Hermitian (positive definite),

- $W = K_n(A, \boldsymbol{r}_0)$ leads to CG;
- $W = AK_n(A, \boldsymbol{r}_0)$ leads to CR.

When A is complex symmetric,

- $W = K_n(\bar{A}, \bar{\boldsymbol{r}}_0)$ leads to COCG.

4.2.2 A derivation of COCR

In this subsection, we derive the algorithm of COCR, and discuss its orthogonality properties.

Let $\boldsymbol{r}_n$ and $\boldsymbol{p}_n$ be the nth residual vector and search direction of COCR, and also given by the recurrences (4.1)-(4.3). Then, $\boldsymbol{r}_n$ and $\boldsymbol{p}_n$ can be computed by determining α_{n-1} and β_{n-1}. Hence, as we see in (4.4), a choice of subspace W is needed to determine these parameters. Compared with the subspaces for CG and COCG, the main difference is the complex conjugate, and thus it is natural from the subspace for CR that we take the following choice:

- $W = \bar{A}K_n(\bar{A}, \bar{\boldsymbol{r}}_0)$.

Hence, the following orthogonality conditions are chosen for COCR:

$$\boldsymbol{r}_n \perp \bar{A}K_n(\bar{A}, \bar{\boldsymbol{r}}_0) \quad \text{and} \quad A\boldsymbol{p}_n \perp \bar{A}K_n(\bar{A}, \bar{\boldsymbol{r}}_0). \tag{4.5}$$

Now, we show a process for obtaining α_{n-1} and β_{n-1} using the recurrences (4.1)-(4.3) and the orthogonality conditions (4.5). For determining α_{n-1}, it follows from (4.2) that the inner product of $\bar{A}^n\bar{\boldsymbol{r}}_0$ and $\boldsymbol{r}_n$ is computed as

$$(\bar{A}^n\bar{\boldsymbol{r}}_0, \boldsymbol{r}_n) = (\bar{A}^n\bar{\boldsymbol{r}}_0, \boldsymbol{r}_{n-1}) - \alpha_{n-1}(\bar{A}^n\bar{\boldsymbol{r}}_0, A\boldsymbol{p}_{n-1}).$$

Since $\bar{A}^n\bar{\boldsymbol{r}}_0$ belongs to $\bar{A}K_n(\bar{A},\bar{\boldsymbol{r}}_0)$, it follows $(\bar{A}^n\bar{\boldsymbol{r}}_0,\boldsymbol{r}_n)=0$ from the conditions (4.5). Hence, we obtain

$$\alpha_{n-1}=\frac{(\bar{A}^n\bar{\boldsymbol{r}}_0,\boldsymbol{r}_{n-1})}{(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{p}_{n-1})}. \tag{4.6}$$

Next, for determining β_{n-1} it follows from (4.3) that the inner product of $\bar{A}^n\bar{\boldsymbol{r}}_0$ and $A\boldsymbol{p}_n$ is computed as

$$(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{p}_n)=(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{r}_n)+\beta_{n-1}(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{p}_{n-1}).$$

From the conditions (4.5) it follows $(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{p}_n)=0$, thus we obtain

$$\beta_{n-1}=-\frac{(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{r}_n)}{(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{p}_{n-1})}=-\alpha_{n-1}\frac{(\bar{A}^n\bar{\boldsymbol{r}}_0,A\boldsymbol{r}_n)}{(\bar{A}^n\bar{\boldsymbol{r}}_0,\boldsymbol{r}_{n-1})}. \tag{4.7}$$

Here, let us consider obtaining practical formulas for α_{n-1} and β_{n-1} from (4.6) and (4.7). Note that from the recurrences (4.1)-(4.3) two vectors $\bar{A}\bar{\boldsymbol{r}}_{n-1}$ and $\bar{A}\bar{\boldsymbol{p}}_{n-1}$ can be written as

$$\bar{A}\bar{\boldsymbol{r}}_{n-1} = \bar{c}_{n-1}\bar{A}^n\bar{\boldsymbol{r}}_0+\bar{A}\bar{\boldsymbol{z}}_1,\quad \bar{A}\bar{\boldsymbol{z}}_1\in\bar{A}K_{n-1}(\bar{A},\bar{\boldsymbol{r}}_0), \tag{4.8}$$

$$\bar{A}\bar{\boldsymbol{p}}_{n-1} = \bar{c}_{n-1}\bar{A}^n\bar{\boldsymbol{r}}_0+\bar{A}\bar{\boldsymbol{z}}_2,\quad \bar{A}\bar{\boldsymbol{z}}_2\in\bar{A}K_{n-1}(\bar{A},\bar{\boldsymbol{r}}_0), \tag{4.9}$$

where $c_{n-1}=(-1)^{n-1}\prod_{i=0}^{n-2}\alpha_i$. Then, from (4.8), (4.9), and the conditions (4.5), the formula α_{n-1} in (4.6) can be rewritten by

$$\alpha_{n-1}=\frac{(\bar{A}\bar{\boldsymbol{r}}_{n-1},\boldsymbol{r}_{n-1})-(\bar{A}\bar{\boldsymbol{z}}_1,\boldsymbol{r}_{n-1})}{(\bar{A}\bar{\boldsymbol{p}}_{n-1},A\boldsymbol{p}_{n-1})-(\bar{A}\bar{\boldsymbol{z}}_2,A\boldsymbol{p}_{n-1})}=\frac{(\bar{A}\bar{\boldsymbol{r}}_{n-1},\boldsymbol{r}_{n-1})}{(\bar{A}\bar{\boldsymbol{p}}_{n-1},A\boldsymbol{p}_{n-1})}. \tag{4.10}$$

Similarly, the formula β_{n-1} in (4.7) can be rewritten by

$$\beta_{n-1}=\frac{(\bar{A}\bar{\boldsymbol{r}}_n,\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{r}}_{n-1},\boldsymbol{r}_{n-1})}. \tag{4.11}$$

Finally, we give an update formula of the nth approximate solution $\boldsymbol{x}_n$. From the relation (4.2), and recall $\boldsymbol{r}_n=\boldsymbol{b}-A\boldsymbol{x}_n$, we obtain

$$\boldsymbol{x}_n=\boldsymbol{x}_{n-1}+\alpha_{n-1}\boldsymbol{p}_{n-1}. \tag{4.12}$$

From (4.1)-(4.3) and (4.10)-(4.12), the algorithm of COCR is obtained as follows:

Algorithm 4.1: COCR method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0=\boldsymbol{b}-A\boldsymbol{x}_0$,
set $\boldsymbol{p}_{-1}=\boldsymbol{0}$, $\beta_{-1}=0$,
for $n=0,1,\ldots,$ until $\|\boldsymbol{r}_n\|\le\epsilon\|\boldsymbol{b}\|$ do:
  $\boldsymbol{p}_n=\boldsymbol{r}_n+\beta_{n-1}\boldsymbol{p}_{n-1}$,
  $(A\boldsymbol{p}_n=A\boldsymbol{r}_n+\beta_{n-1}A\boldsymbol{p}_{n-1},)$
  $\alpha_n=\dfrac{(\bar{\boldsymbol{r}}_n,A\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{p}}_n,A\boldsymbol{p}_n)}$,
  $\boldsymbol{x}_{n+1}=\boldsymbol{x}_n+\alpha_n\boldsymbol{p}_n$,
  $\boldsymbol{r}_{n+1}=\boldsymbol{r}_n-\alpha_nA\boldsymbol{p}_n$,
  $\beta_n=\dfrac{(\bar{\boldsymbol{r}}_{n+1},A\boldsymbol{r}_{n+1})}{(\bar{\boldsymbol{r}}_n,A\boldsymbol{r}_n)}$.
end

When COCR is applied to the linear systems $K_1^{-1}AK_1^{-\mathrm{T}}\tilde{\boldsymbol{x}} = K_1^{-1}\boldsymbol{b}$, the preconditioned COCR can be obtained by the following rewrites:

$$\tilde{\boldsymbol{x}}_n \Rightarrow K_1\boldsymbol{x}_n, \quad \tilde{\boldsymbol{p}}_n \Rightarrow K_1^{\mathrm{T}}\boldsymbol{p}_n, \quad \tilde{\boldsymbol{r}}_n \Rightarrow K_1^{-1}\boldsymbol{r}_n,$$

Then, we have preconditioend COCR method.

Algorithm 4.2: Preconditioned COCR method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
set $\boldsymbol{p}_{-1} = \boldsymbol{0}$, $\beta_{-1} = 0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:
$\boldsymbol{p}_n = K^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$,
$(A\boldsymbol{p}_n = AK^{-1}\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1},)$
$\alpha_n = \dfrac{(\bar{K}^{-1}\bar{\boldsymbol{r}}_n, AK^{-1}\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{p}}_n, K^{-1}A\boldsymbol{p}_n)}$,
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$,
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$,
$(K^{-1}\boldsymbol{r}_{n+1} = K^{-1}\boldsymbol{r}_n - \alpha_n K^{-1}A\boldsymbol{p}_n,)$
$\beta_n = \dfrac{(\bar{K}^{-1}\bar{\boldsymbol{r}}_{n+1}, AK^{-1}\boldsymbol{r}_{n+1})}{(\bar{K}^{-1}\bar{\boldsymbol{r}}_n, AK^{-1}\boldsymbol{r}_n)}$.
end

It follows from Algorithm 4.1 that we obtain the three results: first, it is clear that COCR is equivalent to CR when A is real symmetric, and thus COCR can be considered as an extension of CR for symmetric linear systems to complex symmetric ones; second, similar to COCG, COCR may break down, *i.e.*, $(\bar{A}\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n) = 0$ or $(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n) = 0$ with $\boldsymbol{r}_n \neq \boldsymbol{0}$. On the other hand, it is possible for QMR to evade such breakdowns by using *look-ahead* strategy [41, §4]; third, the following theorem is obtained:

Theorem 4.2.1 *If breakdown does not occur, iterates of COCR satisfy*

$$(\bar{\boldsymbol{r}}_i, A\boldsymbol{r}_j) = 0 \quad \textit{for} \quad i \neq j, \tag{4.13}$$

$$(\bar{A}\bar{\boldsymbol{p}}_i, A\boldsymbol{p}_j) = 0 \quad \textit{for} \quad i \neq j. \tag{4.14}$$

Proof. For the proof of (4.13) and (4.14), it is sufficient to consider the case $j < i$, and the proof is given by induction. Since the trivial case is obvious, we assume that properties (4.13) and (4.14) hold for $j < i \le k$. Then, we show

$$(\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_j) = 0, \tag{4.15}$$

$$(\bar{A}\bar{\boldsymbol{p}}_{k+1}, A\boldsymbol{p}_j) = 0. \tag{4.16}$$

First, we show (4.15). For the case $j < k$ it follows from the assumption that

$$\begin{aligned}(\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_j) &= (\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_j) - \alpha_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{r}_j) \\ &= -\alpha_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{r}_j) \\ &= -\alpha_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_j - \beta_{j-1}A\boldsymbol{p}_{j-1}) = 0.\end{aligned}$$

For the case $j = k$, from the formula of α_k we obtain

$$\begin{aligned}(\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_k) &= (\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k) - \alpha_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{r}_k) \\ &= (\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k) - \alpha_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_k - \beta_{k-1}A\boldsymbol{p}_{k-1}) \\ &= (\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k) - \alpha_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_k) \\ &= 0.\end{aligned}$$

Next, we show (4.16). For the case $j < k$ it follows from the first result of the proof that

$$(\bar{A}\bar{\boldsymbol{p}}_{k+1}, A\boldsymbol{p}_j) = (\bar{A}\bar{\boldsymbol{r}}_{k+1} + \bar{\beta}_k\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_j) = (\bar{A}\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{p}_j) = \frac{1}{\alpha_j}(\bar{A}\bar{\boldsymbol{r}}_{k+1}, \boldsymbol{r}_j - \boldsymbol{r}_{j+1}) = 0.$$

For the case $j = k$, we obtain

$$\begin{aligned}(\bar{A}\bar{\boldsymbol{p}}_{k+1}, A\boldsymbol{p}_k) &= (\bar{A}\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{p}_k) + \beta_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_k) \\ &= \frac{1}{\alpha_k}(\bar{A}\bar{\boldsymbol{r}}_{k+1}, \boldsymbol{r}_k - \boldsymbol{r}_{k+1}) + \beta_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_k) \\ &= -\frac{1}{\alpha_k}(\bar{A}\bar{\boldsymbol{r}}_{k+1}, \boldsymbol{r}_{k+1}) + \beta_k(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_k) \\ &= 0\end{aligned}$$

from the formulas of α_k and β_k. □

Since Algorithm 4.1 satisfies a conjugate A-orthogonality property (4.13) and it is similar to the algorithm of CR, we named it Conjugate A-Orthogonal Conjugate Residual (COCR) method. The computational costs for QMR, COCG, and COCR at each iteration step are shown in Table 4.1. In COCR, four AXPYs are required for Algorithm 4.1 and five for Algorithm 4.2.

Table 4.1. Summary of operations per iteration step, where AXPY: $a\boldsymbol{x} + \boldsymbol{y}$.

Method	Inner Product	AXPY	Matrix-Vector Product	Preconditioner Solve
QMR	2	6	1	1
COCG	2	3	1	1
COCR	2	4 or 5	1	1

At the end of this subsection, let us consider another set of formulas for α_n and β_n in Algorithm 4.1. From (4.13) and (4.14), it is easily verified that $\alpha_n = (\bar{A}\bar{\boldsymbol{p}}_n, \boldsymbol{r}_n)/(\bar{A}\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n)$ and $\beta_n = -(\bar{A}\bar{\boldsymbol{p}}_n, A\boldsymbol{r}_{n+1})/(\bar{A}\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n)$. This set of formulas leads to the algorithm of BiCGCR, but causes one more inner product per iteration step.

A derivation based on conjugate A-orthogonalization process

We give another derivation of COCR. First, we introduce a conjugate A-orthogonalization process. Then, we show that the COCR method is also derived from the process. Similar to the complex symmetric Lanczos process, conjugate A-orthogonalization process of $K_n(A, \boldsymbol{r}_0)$ and $K_n(\bar{A}, \bar{\boldsymbol{r}}_0)$ is given below.

Algorithm 4.3: Conjugate A-orthogonalization process

(4.17) set $\boldsymbol{v}_0 = \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,

set $t_{0,0} = (\bar{A}\bar{\boldsymbol{v}}_0, A\boldsymbol{v}_0)/(\bar{\boldsymbol{v}}_0, A\boldsymbol{v}_0)$,

(4.18) set $\boldsymbol{v}_1 = -\alpha_0(A\boldsymbol{v}_0 - t_{0,0}\boldsymbol{v}_0)$,

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

$t_{n-1,n} = (\bar{A}\bar{\boldsymbol{v}}_{n-1}, A\boldsymbol{v}_n)/(\bar{\boldsymbol{v}}_{n-1}, A\boldsymbol{v}_{n-1})$,

$t_{n,n} = (\bar{A}\bar{\boldsymbol{v}}_n, A\boldsymbol{v}_n)/(\bar{\boldsymbol{v}}_n, A\boldsymbol{v}_n)$,

(4.19) $\boldsymbol{v}_{n+1} = -\alpha_n(A\boldsymbol{v}_n - t_{n,n}\boldsymbol{v}_n - t_{n-1,n}\boldsymbol{v}_{n-1})$.

end

where the scalar parameters $\alpha_0, \ldots, \alpha_n$ are not determined at this point, and these are used as a condition for deriving approximate solutions from residual vectors. From Algorithm 4.3, it is clear that if breakdown does not occur, the vector sequences $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$ satisfy the following conjugate A-orthogonality property:

$$(\bar{\boldsymbol{v}}_i, A\boldsymbol{v}_j) = 0 \quad \text{for} \quad i \neq j.$$

Now, let us consider generating the nth approximate solution $\boldsymbol{x}_n$ for $A\boldsymbol{x} = \boldsymbol{b}$ and $\bar{\boldsymbol{x}}_n$ for $\bar{A}\bar{\boldsymbol{x}} = \bar{\boldsymbol{b}}$ over the following affine spaces:

(4.20)
$$\begin{aligned} \boldsymbol{x}_n &:= \boldsymbol{x}_0 + \boldsymbol{z}_n, \quad \boldsymbol{z}_n \in K_n(A, \boldsymbol{r}_0), \\ \bar{\boldsymbol{x}}_n &:= \bar{\boldsymbol{x}}_0 + \bar{\boldsymbol{z}}_n, \quad \bar{\boldsymbol{z}}_n \in K_n(\bar{A}, \bar{\boldsymbol{r}}_0). \end{aligned}$$

Then, the nth corresponding residual vectors $\boldsymbol{r}_n$ and $\bar{\boldsymbol{r}}_n$ are given by

(4.21) $$\boldsymbol{r}_n := \boldsymbol{b} - A\boldsymbol{x}_n = \boldsymbol{r}_0 - A\boldsymbol{z}_n, \quad \boldsymbol{r}_n \in K_{n+1}(A, \boldsymbol{r}_0),$$

(4.22) $$\bar{\boldsymbol{r}}_n := \bar{\boldsymbol{b}} - \bar{A}\bar{\boldsymbol{x}}_n = \bar{\boldsymbol{r}}_0 - \bar{A}\bar{\boldsymbol{z}}_n, \quad \bar{\boldsymbol{r}}_n \in K_{n+1}(\bar{A}, \bar{\boldsymbol{r}}_0).$$

Since $\boldsymbol{v}_n$ of (4.19) is in $K_{n+1}(A, \boldsymbol{r}_0)$ and $\bar{\boldsymbol{v}}_n$ of (4.19) is in $K_{n+1}(\bar{A}, \bar{\boldsymbol{r}}_0)$, we can use $\boldsymbol{v}_n$, $\bar{\boldsymbol{v}}_n$ as the nth residual vectors, $\boldsymbol{r}_n$ of (4.21) and $\bar{\boldsymbol{r}}_n$ of (4.22), respectively. Then, it follows from the formula (4.19) that we obtain

(4.23) $$\boldsymbol{r}_{n+1} = -\alpha_n(A\boldsymbol{r}_n - t_{n,n}\boldsymbol{r}_n - t_{n-1,n}\boldsymbol{r}_{n-1}),$$

(4.24) $$\bar{\boldsymbol{r}}_{n+1} = -\bar{\alpha}_n(\bar{A}\bar{\boldsymbol{r}}_n - \bar{t}_{n,n}\bar{\boldsymbol{r}}_n - \bar{t}_{n-1,n}\bar{\boldsymbol{r}}_{n-1}),$$

where $t_{n-1,n} = (\bar{A}\bar{\boldsymbol{r}}_{n-1}, A\boldsymbol{r}_n)/(\bar{\boldsymbol{r}}_{n-1}, A\boldsymbol{r}_{n-1})$ and $t_{n,n} = (\bar{A}\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)/(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)$. Substituting (4.21) and (4.22) into (4.23) and (4.24) respectively, the residual vectors are computed as follows:

(4.25)
$$\begin{aligned} \boldsymbol{r}_{n+1} &= \alpha_n(t_{n,n} + t_{n-1,n})\boldsymbol{r}_0 - \alpha_n A(\boldsymbol{r}_n + t_{n,n}\boldsymbol{z}_n + t_{n-1,n}\boldsymbol{z}_{n-1}), \\ \bar{\boldsymbol{r}}_{n+1} &= \bar{\alpha}_n(\bar{t}_{n,n} + \bar{t}_{n-1,n})\bar{\boldsymbol{r}}_0 - \bar{\alpha}_n\bar{A}(\bar{\boldsymbol{r}}_n + \bar{t}_{n,n}\bar{\boldsymbol{z}}_n + \bar{t}_{n-1,n}\bar{\boldsymbol{z}}_{n-1}). \end{aligned}$$

Comparing the coefficient of $\boldsymbol{r}_0$ in (4.21) with the one of $\boldsymbol{r}_0$ in (4.25), α_n must satisfy

(4.26) $$\alpha_n(t_{n,n} + t_{n-1,n}) = 1, \quad n \geq 1$$

to derive an update formula of the approximate solution $\boldsymbol{x}_n$ from the information of $\boldsymbol{r}_n$.

For the special case $n = 0$, from (4.18), (4.21), (4.22) it follows that

$$\boldsymbol{r}_1 = \boldsymbol{r}_0 - A\boldsymbol{z}_1 = -\alpha_0(A\boldsymbol{r}_0 - t_{0,0}\boldsymbol{r}_0), \tag{4.27}$$

$$\bar{\boldsymbol{r}}_1 = \bar{\boldsymbol{r}}_0 - \bar{A}\bar{\boldsymbol{z}}_1 = -\bar{\alpha}_0(\bar{A}\bar{\boldsymbol{r}}_0 - \bar{t}_{0,0}\bar{\boldsymbol{r}}_0), \tag{4.28}$$

and thus α_0 satisfies

$$\alpha_0 = \frac{1}{t_{0,0}} = \frac{(\bar{\boldsymbol{r}}_0, A\boldsymbol{r}_0)}{(\bar{A}\bar{\boldsymbol{r}}_0, A\boldsymbol{r}_0)}. \tag{4.29}$$

Now, we can compute the residual vectors of COCR by using recurrence relations (4.23)-(4.24), and (4.26)-(4.29).

Here, let us give a property of this process. From Algorithm 4.3 it is clear that the sequences $\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots$ satisfy the following conjugate A-orthogonality property:

$$(\bar{\boldsymbol{r}}_i, A\boldsymbol{r}_j) = 0 \quad \text{for} \quad i \neq j. \tag{4.30}$$

Next, we consider an efficient way to compute the residual vectors $\boldsymbol{r}_n$ by introducing two auxiliary vectors, and we give a formula for updating the approximate solution $\boldsymbol{x}_n$. Let us define auxiliary vectors $\boldsymbol{p}_n$, $\bar{\boldsymbol{p}}_n$ as

$$\boldsymbol{p}_n := \frac{\boldsymbol{z}_{n+1} - \boldsymbol{z}_n}{\alpha_n}, \quad \bar{\boldsymbol{p}}_n := \frac{\bar{\boldsymbol{z}}_{n+1} - \bar{\boldsymbol{z}}_n}{\bar{\alpha}_n}. \tag{4.31}$$

Multiplying left and right side of (4.31) by $-\alpha_n A$ and $-\bar{\alpha}_n \bar{A}$ respectively, and from the definitions (4.21) and (4.22) it follows that

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n, \quad \bar{\boldsymbol{r}}_{n+1} = \bar{\boldsymbol{r}}_n - \bar{\alpha}_n \bar{A}\bar{\boldsymbol{p}}_n. \tag{4.32}$$

Substituting the relation (4.26) into $t_{n,n}$ of (4.23) and (4.24), then the recurrence relations (4.23) and (4.24) can be written as

$$\boldsymbol{r}_{n+1} - \boldsymbol{r}_n = -\alpha_n A\boldsymbol{r}_n - \alpha_n t_{n-1,n}(\boldsymbol{r}_n - \boldsymbol{r}_{n-1}), \tag{4.33}$$

$$\bar{\boldsymbol{r}}_{n+1} - \bar{\boldsymbol{r}}_n = -\bar{\alpha_n} \bar{A}\bar{\boldsymbol{r}}_n - \bar{\alpha}_n \bar{t}_{n-1,n}(\bar{\boldsymbol{r}}_n - \bar{\boldsymbol{r}}_{n-1}). \tag{4.34}$$

From (4.32) it follows that $\boldsymbol{r}_{n+1} - \boldsymbol{r}_n = -\alpha_n A\boldsymbol{p}_n$, $\bar{\boldsymbol{r}}_{n+1} - \bar{\boldsymbol{r}}_n = -\bar{\alpha}_n \bar{A}\bar{\boldsymbol{p}}_n$, and thus substituting these relations into (4.33) and (4.34) respectively, we obtain

$$\boldsymbol{p}_n = \boldsymbol{r}_n - \alpha_{n-1} t_{n-1,n}\boldsymbol{p}_{n-1}, \quad \bar{\boldsymbol{p}}_n = \bar{\boldsymbol{r}}_n - \bar{\alpha}_{n-1}\bar{t}_{n-1,n}\bar{\boldsymbol{p}}_{n-1}. \tag{4.35}$$

From the inner product of $\bar{A}\bar{\boldsymbol{r}}_{n+1}$ and (4.23), and using (4.30) and $(\bar{A}\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_{n-1}) = (\bar{A}\bar{\boldsymbol{r}}_{n-1}, A\boldsymbol{r}_n)$, it follows that

$$\alpha_n = -\frac{(\bar{\boldsymbol{r}}_{n+1}, A\boldsymbol{r}_{n+1})}{(\bar{A}\bar{\boldsymbol{r}}_{n+1}, A\boldsymbol{r}_n)}. \tag{4.36}$$

Substituting (4.36) into (4.35), we obtain

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \quad \bar{\boldsymbol{p}}_n = \bar{\boldsymbol{r}}_n + \bar{\beta}_{n-1}\bar{\boldsymbol{p}}_{n-1}, \tag{4.37}$$

where $\beta_{n-1} = (\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)/(\bar{\boldsymbol{r}}_{n-1}, A\boldsymbol{r}_{n-1})$. Comparing (4.27) and (4.28) with (4.37), we obtain $\boldsymbol{p}_0 = \boldsymbol{r}_0$ and $\bar{\boldsymbol{p}}_0 = \bar{\boldsymbol{r}}_0$. Here, we show that α_n satisfies the following relation:

$$\alpha_n = \frac{1}{t_{n,n} + t_{n-1,n}} = \frac{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n)}. \tag{4.38}$$

From (4.26) it is clear that the first equation of (4.38) is obtained. Next, we show the second equality of (4.38). From (4.35) it follows that

$$\begin{aligned}\frac{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n)} &= \frac{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{r}}_n, A\boldsymbol{p}_n) + \beta_{n-1}(\bar{A}\bar{\boldsymbol{p}}_{n-1}, A\boldsymbol{p}_n)} \\ &= \frac{1}{\frac{(\bar{A}\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)} + \beta_{n-1}\frac{(\bar{A}\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_{n-1})}{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)} + \beta_{n-1}\frac{(\bar{A}\bar{\boldsymbol{p}}_{n-1}, A\boldsymbol{p}_n)}{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}} \\ &= \frac{1}{t_{n,n} + t_{n-1,n}},\end{aligned}$$

where we used the relation $(\bar{A}\bar{\boldsymbol{p}}_{n-1}, A\boldsymbol{p}_n) = 0$ because

$$(\bar{A}\bar{\boldsymbol{p}}_{n-1}, A\boldsymbol{p}_n) = \frac{1}{\alpha_n}(\bar{A}\bar{\boldsymbol{p}}_{n-1}, \boldsymbol{r}_n) - \frac{1}{\alpha_n}(\bar{A}\bar{\boldsymbol{p}}_{n-1}, \boldsymbol{r}_{n+1}) = 0,$$

where $(\bar{A}\bar{\boldsymbol{p}}_{n-1}, \boldsymbol{r}_n) = (\bar{A}\bar{\boldsymbol{p}}_{n-1}, \boldsymbol{r}_{n+1}) = 0$ from $\bar{\boldsymbol{p}}_{n-1} \in \mathrm{span}\{\bar{\boldsymbol{r}}_0, \ldots, \bar{\boldsymbol{r}}_{n-1}\}$ and the conjugate A-orthogonality property (4.30). For computing the approximate solutions, it follows from the relation between (4.20) and (4.31) that

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n. \tag{4.39}$$

From the relations (4.27)-(4.29), (4.32), and (4.37)-(4.39), we obtain the COCR method.

A derivation of COCR in matrix form

Here, we derive the COCR method by using a matrix form of Algorithm 4.3. Let $\boldsymbol{r}_0$ be the initial residual vector. Then, Algorithm 4.3 can be expressed in the following matrix form:

$$A\underbrace{[\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n-1}]}_{R_n} = \underbrace{[\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n-1}, \boldsymbol{r}_n]}_{R_{n+1}} \begin{pmatrix} t_{0,0} & t_{0,1} & & \\ -\alpha_0^{-1} & t_{1,1} & \ddots & \\ & -\alpha_1^{-1} & \ddots & t_{n-2,n-1} \\ & & \ddots & t_{n-1,n-1} \\ & & & -\alpha_{n-1}^{-1} \end{pmatrix},$$

where $t_{k-1,k} = \frac{(\bar{A}\bar{\boldsymbol{r}}_{k-1}, A\boldsymbol{r}_k)}{(\bar{\boldsymbol{r}}_{k-1}, A\boldsymbol{r}_{k-1})}$ and $t_{k,k} = \frac{(\bar{A}\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)}{(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)}$. From the above matrix form, we obtain

$$AR_n = R_{n+1}T_{n+1,n}. \tag{4.40}$$

Since the matrix R_n is generated via Algorithm 4.3, it is clear that $R_n^{\mathrm{T}} A R_n = D_n$, where D_n is a diagonal matrix of order n. Since the scalar parameters $\alpha_0, \ldots, \alpha_{n-1}$ still remain unknown. We show that if we determine the parameters such that approximate solutions $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}$ can be extracted from information of $\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{n-1}$, then we obtain the desired algorithm of the COCR method. Let X_n be a matrix with columns $[\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}]$ and $\mathbf{1}$ be $(1, \ldots, 1)^{\mathrm{T}}$. Then we can connect R_n and X_n in the following form:

$$R_n := \boldsymbol{b}\mathbf{1}_n^{\mathrm{T}} - AX_n = [\boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{b} - A\boldsymbol{x}_1, \ldots, \boldsymbol{b} - A\boldsymbol{x}_{n-1}]. \tag{4.41}$$

From the above definition, we can regard the ith column of R_n as the ith residual vector. Substituting (4.41) into (4.40), it follows that

$$AR_n = (\boldsymbol{b}\mathbf{1}_{n+1}^{\mathrm{T}} - AX_{n+1})T_{n+1,n}.$$

Then, we obtain

$$R_n = (\boldsymbol{x}\mathbf{1}_{n+1}^{\mathrm{T}} - X_{n+1})T_{n+1,n},$$

where $\boldsymbol{x}$ is the exact solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$. Hence, to extract X_n from R_n, we need the following condition:

$$\boldsymbol{x}\mathbf{1}_{n+1}^{\mathrm{T}} T_{n+1,n} = O_n.$$

This leads to

$$\begin{aligned} \alpha_0^{-1} &= t_{0,0}, \\ \alpha_k^{-1} &= t_{k,k} + t_{k-1,k}, \quad 1 \le k \le n-1. \end{aligned}$$

Substituting the above recurrence into $T_{n+1,n}$, we have

$$T_{n+1,n} = \begin{pmatrix} \alpha_0^{-1} & t_{0,1} & & \\ -\alpha_0^{-1} & \alpha_1^{-1} - t_{0,1} & \ddots & \\ & -\alpha_1^{-1} & \ddots & t_{n-2,n-1} \\ & & \ddots & \alpha_{n-1}^{-1} - t_{n-2,n-1} \\ & & & -\alpha_{n-1}^{-1} \end{pmatrix}.$$

Then, $T_{n+1,n}$ can be factorized as follows:

$$\begin{aligned} T_{n+1,n} &= \begin{pmatrix} 1 & & \\ -1 & \ddots & \\ & \ddots & 1 \\ & & -1 \end{pmatrix} \begin{pmatrix} \alpha_0^{-1} & & \\ & \ddots & \\ & & \alpha_{n-1}^{-1} \end{pmatrix} \begin{pmatrix} 1 & \alpha_0 t_{0,1} & & \\ & 1 & \ddots & \\ & & \ddots & \alpha_{n-2} t_{n-2,n-1} \\ & & & 1 \end{pmatrix} \\ &= B_{n+1,n}^{(L)} \Omega_n^{-1} B_n^{(U)}. \end{aligned}$$

From (4.40) and the above factorization, we obtain

$$AR_n = R_{n+1}T_{n+1,n} = R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}B_n^{(U)}. \tag{4.42}$$

Here, we introduce $P_n := R_n(B_n^{(U)})^{-1}$. Then from (4.42), we obtain

$$AP_n = AR_n(B_n^{(U)})^{-1} = R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}. \tag{4.43}$$

The above matrix form are equivalent to the following recurrences:

$$\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_{k-1}A\boldsymbol{p}_{k-1}, \quad 1 \le k \le n. \tag{4.44}$$

From $P_n = R_n(B_n^{(U)})^{-1}$ it follows that

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_{k-1}\boldsymbol{p}_{k-1}, \quad 1 \le k \le n, \tag{4.45}$$

where $\beta_{k-1} := -\alpha_{k-1}t_{k-1,k}$. Here, we give the computational formula of approximate solution. From (4.43), it follows that

$$\begin{aligned} AP_n &= R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1} \\ &= (\boldsymbol{b}\mathbf{1}^{\mathrm{T}} - AX_{n+1})B_{n+1,n}^{(L)}\Omega_n^{-1} \\ &= -AX_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}. \end{aligned}$$

Thus, we obtain

$$P_n\Omega_n = X_{n+1}(-B_{n+1,n}^{(L)}),$$

and it is equivalent to

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_{k-1}\boldsymbol{p}_{k-1}, \quad 1 \le k \le n. \tag{4.46}$$

Now, we give more practical computational formulas of α_k and β_k. From (4.40), it follows that

$$(AR)_n^{\mathrm{T}}AR_n = R_n^{\mathrm{T}}AR_{n+1}T_{n+1,n}.$$

From the (k+1,k) entry of the above relation, we have $(\bar{A}\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k-1}) = -\alpha_{k-1}^{-1}(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)$. Thus from $\beta_{k-1} = -\alpha_{k-1}t_{k-1,k}$, we obtain

$$\begin{aligned} \beta_k &= -\alpha_k t_{k,k+1} \\ &= \frac{(\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_{k+1})}{(\bar{A}\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_k)} \cdot \frac{(\bar{A}\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k+1})}{(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)} \\ &= \frac{(\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_{k+1})}{(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)}. \end{aligned} \tag{4.47}$$

We used the relation $(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k+1}) = (\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_k)$ since $\boldsymbol{r}_n$ is written as the product of matrix polynomial and initial residual vector, *i.e.*,

$$\begin{aligned} (\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k+1}) &= (\bar{R}_k(A)\bar{\boldsymbol{r}}_0, AR_{k+1}(A)\boldsymbol{r}_0) \\ &= (\bar{R}_{k+1}(A)\bar{R}_k(A)\bar{\boldsymbol{r}}_0, A\boldsymbol{r}_0) \\ &= (\bar{R}_k(A)\bar{R}_{k+1}(A)\bar{\boldsymbol{r}}_0, A\boldsymbol{r}_0) \\ &= (\bar{R}_{k+1}(A)\bar{\boldsymbol{r}}_0, AR_k(A)\boldsymbol{r}_0) \\ &= (\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_k). \end{aligned}$$

From (4.43) and recalling $P_n = R_n(B_n^{(U)})^{-1}$, it follow that

$$(AP_n)^{\mathrm{T}}AP_n = (AP_n)^{\mathrm{T}}R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1} = (B_n^{(U)})^{-\mathrm{T}}R_n^{\mathrm{T}}AR_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}.$$

This leads to

$$(AP_n)^{\mathrm{T}}AP_n = \begin{pmatrix} d_0 & & \\ * & \ddots & \\ * & * & d_{n-1} \end{pmatrix},$$

where $d_k = (\boldsymbol{r}_k, \boldsymbol{r}_k)\alpha_k^{-1}$. Thus, we obtain

$$\alpha_k = \frac{(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)}{(\bar{A}\bar{\boldsymbol{p}}_k, A\boldsymbol{p}_k)}. \tag{4.48}$$

Moreover, since $(AP_n)^{\mathrm{T}}AP_n$ is complex symmetric, $(AP_n)^{\mathrm{T}}AP_n = \mathrm{diag}(d_0, \ldots, d_{n-1})$. Thus $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are conjugate A^2-orthogonal, *i.e.*,

$$(\bar{A}\bar{\boldsymbol{p}}_i, A\boldsymbol{p}_j) = 0 \quad \text{for} \quad i \neq j.$$

From (4.44)-(4.48), we obtain the algorithm of COCR.

4.2.3 Relationship between COCR and Bi-CR

In this subsection, we show that if matrix A is complex symmetric, Bi-CR with the choice $\boldsymbol{r}_0^* = \bar{\boldsymbol{r}}_0$ reduces to COCR. It follows from Algorithm 3.4 that we have the following recurrences:

$$\begin{aligned}
&\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},\\
&\boldsymbol{p}_n^* = \boldsymbol{r}_n^* + \bar{\beta}_{n-1}\boldsymbol{p}_{n-1}^*,\\
&(A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1},)\\
&\alpha_n = \frac{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{p}_n^*, A\boldsymbol{p}_n)},\\
&\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,\\
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,\\
&\boldsymbol{r}_{n+1}^* = \boldsymbol{r}_n^* - \bar{\alpha}_n A^{\mathrm{H}}\boldsymbol{p}_n^*,\\
&\beta_n = \frac{(\boldsymbol{r}_{n+1}^*, A\boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n^*, A\boldsymbol{r}_n)}.
\end{aligned}$$

If matrix A is complex symmetric, then we have the relation $A^{\mathrm{H}} = \bar{A}$. On the other hand, the choice $\boldsymbol{r}_0^* = \bar{\boldsymbol{r}}_0$ leads to $\boldsymbol{p}_n^* = \bar{\boldsymbol{p}}_n$ and $\boldsymbol{r}_n^* = \bar{\boldsymbol{r}}_n$. Hence, from the above recurrences, it follow that

$$\begin{aligned}
&\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},\\
&\bar{\boldsymbol{p}}_n = \bar{\boldsymbol{r}}_n + \bar{\beta}_{n-1}\bar{\boldsymbol{p}}_{n-1},\\
&(A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1},)\\
&\alpha_n = \frac{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n)},\\
&\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,\\
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,
\end{aligned} \tag{4.49}$$

$$\begin{aligned}
&\bar{\boldsymbol{r}}_{n+1} = \bar{\boldsymbol{r}_n} - \bar{\alpha}_n\bar{A}\bar{\boldsymbol{p}}_n,\\
&\beta_n = \frac{(\bar{\boldsymbol{r}}_{n+1}, A\boldsymbol{r}_{n+1})}{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}.
\end{aligned} \tag{4.50}$$

From the above, the recurrences (4.49) and (4.50) can be deleted since $\bar{\boldsymbol{p}}_n$ and $\bar{\boldsymbol{r}}_n$ are readily obtained by $\boldsymbol{p}_n$ and $\boldsymbol{r}_n$. Hence, we can see that the resulting algorithm is the same as Algorithm 4.1 and that Bi-CR is regarded as an extension of COCR.

4.3 Numerical experiments

In this section, we report some numerical examples with QMR, COCG, and COCR. We evaluate both methods in aspects of the number of iterations (Its) and $\log_{10}$ of true relative residual 2-norm (TRR) defined as $\log_{10} \|\boldsymbol{b} - A\boldsymbol{x}_n\|/\|\boldsymbol{b}\|$. All tests were performed on an ALPHA work station with a 750MHz processor using double precision arithmetic. Codes were written in Fortran 77 and compiled with the optimization option -$O4$. In all cases the iteration was started with $\boldsymbol{x}_0 = \boldsymbol{0}$, and the stopping criterion was $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq 10^{-6}$. The preconditioner was IC(0) [59]. For the complex symmetric structure of A, IC(0) produces LDL^{T}. If the diagonal matrix D and the lower triangular matrix L are nonsingular, then the preconditioned matrix $D^{-1/2}L^{-1}AL^{-\mathrm{T}}D^{-1/2}$ is also a nonsingular complex symmetric matrix. Thus, in this case, we can use LDL^{T} as a preconditioner. The convergence plots show $\log_{10}$ of the relative residual 2-norm, $\log_{10} \|\boldsymbol{r}_n\|/\|\boldsymbol{b}\|$, (on the vertical axis) versus Its (on the horizontal axis).

◇ Example 1

In the first numerical example, we consider problems from NEP collection [3]. We chose two matrices from electrical engineering (DWG961B) and quantum chemistry (QC2534). Numerical results for each test problem are given in Table 4.2. The right-hand side $\boldsymbol{b}$ was chosen as $(1+i, \ldots, 1+i)^{\mathrm{T}}$.

Table 4.2. Numerical results for Example 1, where N: order of matrix, Its: number of iterations, TRR: $\log_{10}$ of final true relative residual 2-norm.

		Its			TRR		
Matrix	N	QMR	COCG	COCR	QMR	COCG	COCR
DWG961B	961	1523	1365	1388	-6.00	-6.10	-6.05
QC2534	2534	1008	628	403	-6.14	-6.06	-6.41

Convergence histories for DWG961B and QC2534 are shown in Fig. 4.1 and Fig. 4.2. We see from Fig. 4.1 that COCG and COCR gave similar convergence behavior, and the residual norm of COCR was often less than that of COCG and QMR at each iteration step. On the other hand, QMR often gave much smoother convergence behavior than COCG and COCR; however QMR required a larger number of iterations than the other two methods. Fig. 4.2 shows that COCR converged considerably faster than COCG and QMR. In the two cases, the three methods gave almost the same accuracy on TRR at each final iteration step (see Table 4.2).

◇ Example 2

In the second numerical example, we consider a complex symmetric linear system arising from the 200×200 central difference discretization of the Helmholtz equation described in [6]: $u_{xx} + u_{yy} + \sigma^2 u = 0$ over $[0, \pi] \times [0, \pi]$, with Dirichlet condition $u = 0$ along $y = \pi$, Neumann conditions $u_x = i\sqrt{\sigma^2 - \frac{1}{4}}\cos(\frac{y}{2})$ along $x = 0$ and $u_y = 0$ along $y = 0$, and radiation condition $u_x - i\sqrt{\sigma^2 - \frac{1}{4}}u = 0$ along $x = \pi$. This leads to a linear system with 201×200 unknowns. Here, we consider the two cases of $\sigma = 2.0, 4.0$. These numerical results are shown in Fig. 4.3 and Fig. 4.4.

Table 4.3. Numerical results for Example 2, where N: order of matrix, Its: number of iterations, TRR: $\log_{10}$ of final true relative residual 2-norm.

		Its			TRR		
Matrix	N	QMR	COCG	COCR	QMR	COCG	COCR
$\sigma = 2.0$	40200	276	288	278	-6.01	-6.03	-6.04
$\sigma = 4.0$	40200	453	473	458	-6.06	-6.01	-6.01

From Fig. 4.3 and Fig. 4.4 we observe that COCR gave smoother convergence behavior than COCG in the early phase, and then the two methods showed similar convergence behavior. We also see that QMR has an advantage over COCG and COCR in that its residual 2-norm decreased almost monotonically. In the two cases ($\sigma = 2.0, 4.0$), they gave almost the same accuracy on TRR at each final iteration step (see Table 4.3).

4.4 Concluding remarks

In this chapter, first, we observed a way for obtaining CG, CR, and COCG. Second, we derived the algorithm of COCR from the observation and showed some orthogonality properties. We gave other derivations of COCR from matrix form of a conjugate A-orthogonalization process, and the algorithm of Bi-CR. From numerical examples we have learned that COCR tends to give smoother convergence behavior than COCG, and it sometimes converges faster than QMR in terms of the number of iterations. Hence, we conclude that COCR as well as QMR and COCG may become a useful tool for solving complex symmetric linear systems.

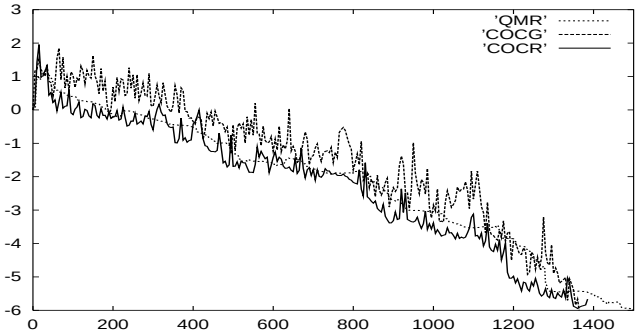

Figure 4.1: Residual 2-norm histories for Example 1 (DW961B).

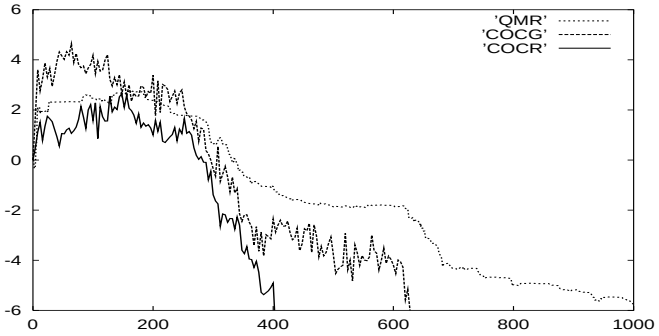

Figure 4.2: Residual 2-norm histories for Example 1 (QC2534).

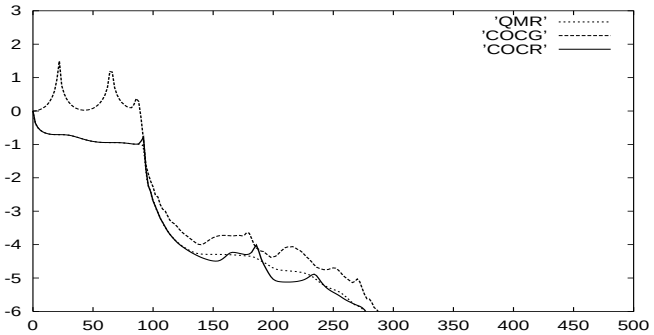

Figure 4.3: Residual 2-norm histories for Example 2 ($\sigma = 2.0$).

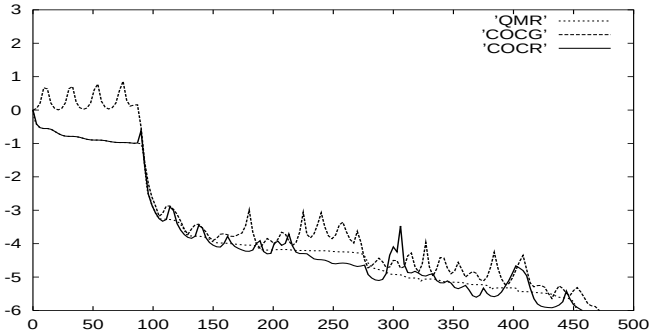

Figure 4.4: Residual 2-norm histories for Example 2 ($\sigma = 4.0$).

Chapter 5

CRS: a conjugate residual squared method

In chapter 3, we have learned that Bi-CR often gives smoother convergence behavior than Bi-CG. However, Bi-CR needs one matrix-vector multiplication and one transposed matrix-vector multiplication even though the operations increase only one dimension of Krylov subspace for updating the approximate solution. In this chapter, to improve the performance of Bi-CR, we give a general framework of the product-type methods and then we propose a more efficient algorithm by defining a residual vector as the product of the Bi-CR polynomial squared and the initial residual vector. The algorithm is referred to as a Conjugate Residual Squared (CRS) method. The expected convergence rate is about twice that of Bi-CR.

5.1 A general framework of product-type methods based on Bi-CR

As we saw in chapter 3, the residual vector of Bi-CR at the nth iteration step is characterized by $R_n(\lambda)$ and the initial residual vector $\boldsymbol{r}_0$:

$$\boldsymbol{r}_n^{\mathrm{BiCR}} = R_n(A)\boldsymbol{r}_0.$$

Similar to the product-type methods based on Bi-CG described in chapter 2, we consider the product of a polynomial of degree n and the nth residual vector of Bi-CR:

$$\boldsymbol{r}_n = H_n(A)\boldsymbol{r}_n^{\mathrm{BiCR}} = H_n(A)R_n(A)\boldsymbol{r}_0.$$

The choice of the polynomial H_n plays an important role in improving the convergence behavior of Bi-CR. Hence, we define H_n such that it meets the following requirements:

(1) The recurrence for computing residual vectors of the product-type methods are given by the determination of H_n.

(2) Two parameters α_n, β_n in the polynomial R_n are equivalent to those used in the Bi-CR method, and they are readily computed by the nth residual vector.

(3) H_n needs to satisfy short-term recurrences for low computational costs per iteration and low memory requirements.

(4) Parameters in H_n should be chosen so that they accelerate and stabilize the convergence behavior of Bi-CR.

Restructuring of residual polynomials

To meet the previous requirements, let us introduce three-term recurrences relations similar to Lanczos polynomial (2.21)-(2.23):

$$H_0(\lambda) = 1, \tag{5.1}$$
$$H_1(\lambda) = (1-\zeta_0\lambda)H_0(\lambda), \tag{5.2}$$
$$H_n(\lambda) = (1+\eta_{n-1}-\zeta_{n-1}\lambda)H_{n-1}(\lambda) - \eta_{n-1}H_{n-2}(\lambda), \quad n=2,3,\ldots \tag{5.3}$$

Here, we transform the above three-term recurrences into coupled two-term ones to give useful update formulas for the residual vector. Let G_{n-1} be a polynomial of degree $n-1$ defined by

$$G_{n-1}(\lambda) := \frac{H_{n-1}(\lambda) - H_n(\lambda)}{\lambda}.$$

Then, rewriting the three-term recurrence relations (5.3) as

$$H_n(\lambda) - H_{n-1}(\lambda) = -\zeta_{n-1}\lambda H_{n-1}(\lambda) + \eta_{n-1}(H_{n-1}(\lambda) - H_{n-2}(\lambda))$$

and using (5.1)-(5.3), we obtain the following recurrences:

$$\begin{aligned}
&H_0(\lambda) = 1, \quad G_0(\lambda) = \zeta_0,\\
&H_n(\lambda) = H_{n-1}(\lambda) - \lambda G_{n-1}(\lambda),\\
&G_n(\lambda) = \zeta_n H_n(\lambda) + \eta_n G_{n-1}(\lambda), \quad n=1,2,\ldots
\end{aligned}$$

Next, by using the above recurrences, we give update formulas for the residual vector $\boldsymbol{r}_n$.

Recurrence formulas for iterates

Since R_n and H_n have been already given, recurrence relations for updating $\boldsymbol{r}_n$ can be obtained. We make use of a new set of recurrence relations among the products of polynomials H_nR_n, H_nR_{n+1},
$\lambda G_{n-1}R_{n+1}$, H_nP_n, λH_nP_{n+1}, λG_nP_n, and G_nR_{n+1} in order to compute the product of polynomials H_nR_n.

$$H_{n+1}R_{n+1} = H_nR_{n+1} - \eta_n\lambda G_{n-1}R_{n+1} - \zeta_n\lambda H_nR_{n+1} \tag{5.4}$$
$$= H_nR_n - \alpha_n\lambda H_nP_n - \lambda G_nR_{n+1}, \tag{5.5}$$
$$H_nR_{n+1} = H_nR_n - \alpha_n\lambda H_nP_n, \tag{5.6}$$
$$\lambda G_nR_{n+2} = H_nR_{n+1} - H_{n+1}R_{n+1} - \alpha_{n+1}\lambda H_nP_{n+1} + \alpha_{n+1}\lambda H_{n+1}P_{n+1}, \tag{5.7}$$
$$H_{n+1}P_{n+1} = H_{n+1}R_{n+1} + \beta_nH_nP_n - \beta_n\lambda G_nP_n, \tag{5.8}$$
$$\lambda H_nP_{n+1} = \lambda H_nR_{n+1} + \beta_n\lambda H_nP_n, \tag{5.9}$$
$$\lambda G_nP_n = \zeta_n\lambda H_nP_n + \eta_n(H_{n-1}R_n - H_nR_n + \beta_{n-1}\lambda G_{n-1}P_{n-1}), \tag{5.10}$$
$$G_nR_{n+1} = \zeta_nH_nR_n + \eta_nG_{n-1}R_n - \alpha_n\lambda G_nP_n. \tag{5.11}$$

Let us define new auxiliary vectors as

$$\begin{aligned} \boldsymbol{t}_n &:= H_n(A)\boldsymbol{r}_{n+1}^{\mathrm{BiCR}}, & \boldsymbol{y}_n &:= AG_{n-1}(A)\boldsymbol{r}_{n+1}^{\mathrm{BiCR}}, \\ \boldsymbol{p}_n &:= H_n(A)\boldsymbol{p}_n^{\mathrm{BiCR}}, & \boldsymbol{w}_n &:= AH_n(A)\boldsymbol{p}_{n+1}^{\mathrm{BiCR}}, \\ \boldsymbol{u}_n &:= AG_n(A)\boldsymbol{p}_n^{\mathrm{BiCR}}, & \boldsymbol{z}_n &:= G_n(A)\boldsymbol{r}_{n+1}^{\mathrm{BiCR}}. \end{aligned}$$

Then, we have new recurrence formulas from (5.4)-(5.11):

$$\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \eta_n\boldsymbol{y}_n - \zeta_n A\boldsymbol{t}_n \tag{5.12}$$

$$= \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n - A\boldsymbol{z}_n, \tag{5.13}$$

$$\begin{aligned} \boldsymbol{t}_n &= \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n, \\ \boldsymbol{y}_{n+1} &= \boldsymbol{t}_n - \boldsymbol{r}_{n+1} - \alpha_{n+1}\boldsymbol{w}_n + \alpha_{n+1}A\boldsymbol{p}_{n+1}, \\ \boldsymbol{p}_{n+1} &= \boldsymbol{r}_{n+1} + \beta_n(\boldsymbol{p}_n - \boldsymbol{u}_n), \\ \boldsymbol{w}_n &= A\boldsymbol{t}_n + \beta_n A\boldsymbol{p}_n, \\ \boldsymbol{u}_n &= \zeta_n A\boldsymbol{p}_n + \eta_n(\boldsymbol{t}_{n-1} - \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{u}_{n-1}), \\ \boldsymbol{z}_n &= \zeta_n\boldsymbol{r}_n + \eta_n\boldsymbol{z}_{n-1} - \alpha_n\boldsymbol{u}_n. \end{aligned}$$

From (5.13), the following new formula for the approximate solution can be obtained:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n + \boldsymbol{z}_n.$$

The recurrence (5.12) is still of great importance to achieve local minimization of $\boldsymbol{r}_{n+1}$ by the two parameters ζ_n, η_n.

Computational formulas for α_n, β_n

Since the coefficient of the highest-order term of H_n is $(-1)^n\prod_{i=0}^{n-1}\zeta_i$, we obtain

$$(A^{\mathrm{H}}\boldsymbol{r}_0^*, \boldsymbol{r}_n) = (A^{\mathrm{H}}\bar{H}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{BiCR}}) = \left((-1)^n\prod_{i=0}^{n-1}\zeta_i\right)((A^{\mathrm{H}})^{n+1}\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{BiCR}}),$$

$$(A^{\mathrm{H}}\boldsymbol{r}_0^*, A\boldsymbol{p}_n) = (A^{\mathrm{H}}\bar{H}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, A\boldsymbol{p}_n^{\mathrm{BiCR}}) = \left((-1)^n\prod_{i=0}^{n-1}\zeta_i\right)((A^{\mathrm{H}})^{n+1}\boldsymbol{r}_0^*, A\boldsymbol{p}_n^{\mathrm{BiCR}}).$$

Thus, it follows from (3.29) and (3.31) that

$$\alpha_n = \frac{(A^{\mathrm{H}}\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{r}_0^*, A\boldsymbol{p}_n)}, \quad \beta_n = \frac{\alpha_n}{\zeta_n}\cdot\frac{(A^{\mathrm{H}}\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(A^{\mathrm{H}}\boldsymbol{r}_0^*, \boldsymbol{r}_n)}.$$

Similar to the product-type methods based on Bi-CG, we can derive several product-type methods based on Bi-CR by the choice of the two parameters ζ_n, η_n. The choice and these names are given in Table 5.1.

Table 5.1. The choice for the product-type methods based on Bi-CR.

CRS	$\zeta_n = \alpha_n, \quad \eta_n = \dfrac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n \iff H_n = R_n$
Bi-CRSTAB	$\zeta_n = \arg\min_{\zeta_n\in\mathcal{C}}\|\boldsymbol{r}_{n+1}\|, \quad \eta_n = 0$
Bi-CRSTAB2	$\zeta_n, \eta_n = \arg\min_{\zeta_n,\eta_n\in\mathcal{C}}\|\boldsymbol{r}_{n+1}\|$ ($\eta_n = 0$ at even iterations)
GPBi-CR	$\zeta_n, \eta_n = \arg\min_{\zeta_n,\eta_n\in\mathcal{C}}\|\boldsymbol{r}_{n+1}\|$

5.2 Definition of the residual vector of CRS

By the choice for CRS given in Table 5.1, we have the residual vector of CRS as the product of the Bi-CR polynomial squared and an initial residual vector, *i.e.*

$$\boldsymbol{r}_n := R_n(A)^2 \boldsymbol{r}_0. \tag{5.14}$$

Since $R_n(A)\boldsymbol{r}_0$ is the nth residual vector of Bi-CR (3.21), $\boldsymbol{r}_n$ can be also written as the product of Bi-CR polynomial and $\boldsymbol{r}_n^{\mathrm{BiCR}}$:

$$\boldsymbol{r}_n = R_n(A)\boldsymbol{r}_n^{\mathrm{BiCR}}.$$

If Bi-CR gives better convergence behavior, the matrix 2-norm of $R_n(A)$ will decrease faster. Hence, we can expect that the value of $\|\boldsymbol{r}_n\|$ converges to zero faster than that of Bi-CR.

Here, we give more quantitative evaluation of convergence for CRS and Bi-CR. The residual 2-norm of Bi-CR is bounded by $\|R_n(A)\| \cdot \|\boldsymbol{r}_0\|$. On the other hand, the residual 2-norm of CRS is bounded by $\|R_n(A)\|^2 \cdot \|\boldsymbol{r}_0\|$. Hence, assuming $\|R_n(A)\| \leq 1$, we can expect that the convergence rate of CRS is about twice that of Bi-CR.

5.3 Recurrence formulas for CRS iterates

In this section, we give useful recurrence formulas for updating $\boldsymbol{r}_n$. From the definition (5.14) and Bi-CR polynomial (3.22)-(3.24), we obtain the following recurrences:

$$\begin{aligned}
R_{n+1}R_{n+1} &= R_nR_{n+1} - \alpha_n\lambda P_nR_{n+1} \\
&= R_nR_n - \alpha_n\lambda(R_nP_n + P_nR_{n+1}), \\
R_{n+1}P_{n+1} &= R_{n+1}R_{n+1} + \beta_nP_nR_{n+1}, \\
P_nR_{n+1} &= P_nR_n - \alpha_n\lambda P_nP_n, \\
\lambda P_{n+1}P_{n+1} &= \lambda P_{n+1}R_{n+1} + \beta_n\lambda P_{n+1}P_n \\
&= \lambda P_{n+1}R_{n+1} + \beta_n(\lambda R_{n+1}P_n + \beta_n\lambda P_nP_n).
\end{aligned}$$

Here, we introduce auxiliary vector

$$\boldsymbol{e}_n := P_nR_n\boldsymbol{r}_0, \quad \boldsymbol{h}_n := P_nR_{n+1}\boldsymbol{r}_0, \quad \boldsymbol{p}_n := P_nP_n\boldsymbol{r}_0.$$

Then, from the previous recurrences, we have the following formulas:

$$\begin{aligned}
A\boldsymbol{p}_n &= A\boldsymbol{e}_n + \beta_{n-1}(A\boldsymbol{h}_{n-1} + \beta_{n-1}A\boldsymbol{p}_{n-1}), \\
\boldsymbol{h}_n &= \boldsymbol{e}_n - \alpha_nA\boldsymbol{p}_n, \\
\boldsymbol{r}_{n+1} &= \boldsymbol{r}_n - \alpha_nA(\boldsymbol{e}_n + \boldsymbol{h}_n), \\
\boldsymbol{e}_{n+1} &= \boldsymbol{r}_{n+1} + \beta_n\boldsymbol{h}_n.
\end{aligned}$$

The parameters α_n and β_n of CRS are mathematically equivalent to those of Bi-CR. Based on the formulas given in (3.39), we see that α_n and β_n satisfy the following relations:

$$\begin{aligned}
\alpha_n &= \frac{(\boldsymbol{r}_n^{\mathrm{BiCR}*}, A\boldsymbol{r}_n^{\mathrm{BiCR}})}{(A^{\mathrm{H}}\boldsymbol{p}_n^{\mathrm{BiCR}*}, A\boldsymbol{p}_n^{\mathrm{BiCR}})} = \frac{(\bar{R}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, AR_n(A)\boldsymbol{r}_0)}{(A^{\mathrm{H}}\bar{P}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, AP_n(A)\boldsymbol{r}_0)} = \frac{(\boldsymbol{r}_0^*, AR_n(A)R_n(A)\boldsymbol{r}_0)}{(\boldsymbol{r}_0^*, A^2P_n(A)P_n(A)\boldsymbol{r}_0)}, \\
\beta_n &= \frac{(\boldsymbol{r}_n^{\mathrm{BiCR}*}, A\boldsymbol{r}_n^{\mathrm{BiCR}})}{(\boldsymbol{r}_{n+1}^{\mathrm{BiCR}*}, A\boldsymbol{r}_{n+1}^{\mathrm{BiCR}})} = \frac{(\bar{R}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, AR_n(A)\boldsymbol{r}_0)}{(\bar{R}_{n+1}(A^{\mathrm{T}})\boldsymbol{r}_0^*, AR_{n+1}(A)\boldsymbol{r}_0)} = \frac{(\boldsymbol{r}_0^*, AR_n(A)R_n(A)\boldsymbol{r}_0)}{(\boldsymbol{r}_0^*, AR_{n+1}(A)R_{n+1}(A)\boldsymbol{r}_0)}.
\end{aligned}$$

Hence, the same values given in (3.39) can be generated by the residual vector of CRS. It follows from the above results and the definitions $\boldsymbol{r}_n = R_n(A)R_n(A)\boldsymbol{r}_0,\ \boldsymbol{p}_n = P_n(A)P_n(A)\boldsymbol{r}_0$ that we have

$$\alpha_n = \frac{(\boldsymbol{r}_0^*, A\boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A^2\boldsymbol{p}_n)}, \quad \beta_n = \frac{(\boldsymbol{r}_0^*, A\boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{r}_{n+1})}.$$

We see from the computational formulas for α_n and β_n that they require four matrix vector multiplications. Hence, to reduce the operations, we introduce the following recurrences:

$$\begin{aligned} A\boldsymbol{h}_n &= A\boldsymbol{e}_n - \alpha_n A^2\boldsymbol{p}_n, \\ A\boldsymbol{e}_{n+1} &= A\boldsymbol{r}_{n+1} + \beta_n A\boldsymbol{h}_n. \end{aligned}$$

Using the notation $\boldsymbol{q}_n := A\boldsymbol{p}_n$, $\boldsymbol{d}_n := A\boldsymbol{e}_n$, and $\boldsymbol{f}_n := A\boldsymbol{h}_n$, we have

$$\begin{aligned} \boldsymbol{q}_n &= \boldsymbol{d}_n + \beta_{n-1}(\boldsymbol{f}_{n-1} + \beta_{n-1}\boldsymbol{q}_{n-1}), \\ \boldsymbol{f}_n &= \boldsymbol{d}_n - \alpha_n A\boldsymbol{q}_n, \\ \boldsymbol{d}_{n+1} &= A\boldsymbol{r}_{n+1} + \beta_n \boldsymbol{f}_n. \end{aligned}$$

By using the above recurrences, we can reduce two matrix vector multiplications per iteration step. Finally, the residual vector and the approximate solution are written as

$$\begin{aligned} \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \alpha_n(\boldsymbol{e}_n + \boldsymbol{h}_n), \\ \boldsymbol{r}_{n+1} &= \boldsymbol{r}_n - \alpha_n(\boldsymbol{d}_n + \boldsymbol{f}_n). \end{aligned}$$

From the above results, we obtain the algorithm of CRS below.

Algorithm 5.1: CRS method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
choose $\boldsymbol{r}_0^*$ (for example, $\boldsymbol{r}_0^* = \boldsymbol{r}_0$),
set $\boldsymbol{e}_0 = \boldsymbol{r}_0,\ \boldsymbol{d}_0 = A\boldsymbol{e}_0,\ \beta_{-1} = 0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:
$\boldsymbol{q}_n = \boldsymbol{d}_n + \beta_{n-1}(\boldsymbol{f}_{n-1} + \beta_{n-1}\boldsymbol{q}_{n-1})$,
$\alpha_n = \dfrac{(\boldsymbol{r}_0^*, A\boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{q}_n)}$,
$\boldsymbol{h}_n = \boldsymbol{e}_n - \alpha_n\boldsymbol{q}_n$,
$\boldsymbol{f}_n = \boldsymbol{d}_n - \alpha_n A\boldsymbol{q}_n$,
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n(\boldsymbol{e}_n + \boldsymbol{h}_n)$,
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n(\boldsymbol{d}_n + \boldsymbol{f}_n)$,
$\beta_n = \dfrac{(\boldsymbol{r}_0^*, A\boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, A\boldsymbol{r}_n)}$,
$\boldsymbol{e}_{n+1} = \boldsymbol{r}_{n+1} + \beta_n\boldsymbol{h}_n$,
$\boldsymbol{d}_{n+1} = A\boldsymbol{r}_{n+1} + \beta_n\boldsymbol{f}_n$.
end

We can see from Algorithm 5.1 that it requires two matrix-vector multiplications per iteration step. Since the product-type methods based on Bi-CG such as CGS, Bi-CGSTAB,

and GPBi-CG also require two matrix-vector multiplications, we can say that CRS is a practical algorithm in terms of operations.

When CRS is applied to the linear systems (2.15), a preconditioned CRS method is obtained by the following rewrites:

$$\tilde{\boldsymbol{x}}_n \Rightarrow K_2\boldsymbol{x}_n, \quad \tilde{\boldsymbol{h}}_n \Rightarrow K_1^{-1}\boldsymbol{h}_n, \quad \tilde{\boldsymbol{e}}_n \Rightarrow K_1^{-1}\boldsymbol{e}_n, \quad \tilde{\boldsymbol{r}}_0^* \Rightarrow K_1^{-\mathrm{H}}\boldsymbol{r}_0^*,$$
$$\tilde{\boldsymbol{r}}_n \Rightarrow K_1^{-1}\boldsymbol{r}_n, \quad \tilde{\boldsymbol{q}}_n \Rightarrow K_1^{-1}\boldsymbol{q}_n, \quad \tilde{\boldsymbol{f}}_n \Rightarrow K_1^{-1}\boldsymbol{f}_n, \quad \tilde{\boldsymbol{d}}_n \Rightarrow K_1^{-1}\boldsymbol{d}_n.$$

The resulting algorithm is given as follows:

Algorithm 5.2: Preconditioned CRS method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
choose $\boldsymbol{r}_0^*$ (for example, $\boldsymbol{r}_0^* = \boldsymbol{r}_0$),
set $\boldsymbol{e}_0 = \boldsymbol{r}_0,\ \boldsymbol{d}_0 = AK^{-1}\boldsymbol{e}_0,\ \beta_{-1} = 0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:
$\boldsymbol{q}_n = \boldsymbol{d}_n + \beta_{n-1}(\boldsymbol{f}_{n-1} + \beta_{n-1}\boldsymbol{q}_{n-1}),$
$\alpha_n = \dfrac{(\boldsymbol{r}_0^*, AK^{-1}\boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, AK^{-1}\boldsymbol{q}_n)},$
$\boldsymbol{h}_n = \boldsymbol{e}_n - \alpha_n\boldsymbol{q}_n,$
$(K^{-1}\boldsymbol{h}_n = K^{-1}\boldsymbol{e}_n - \alpha_n K^{-1}\boldsymbol{q}_n,)$
$\boldsymbol{f}_n = \boldsymbol{d}_n - \alpha_n AK^{-1}\boldsymbol{q}_n,$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n(K^{-1}\boldsymbol{e}_n + K^{-1}\boldsymbol{h}_n),$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n(\boldsymbol{d}_n + \boldsymbol{f}_n),$
$\beta_n = \dfrac{(\boldsymbol{r}_0^*, AK^{-1}\boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, AK^{-1}\boldsymbol{r}_n)},$
$\boldsymbol{e}_{n+1} = \boldsymbol{r}_{n+1} + \beta_n\boldsymbol{h}_n,$
$(K^{-1}\boldsymbol{e}_{n+1} = K^{-1}\boldsymbol{r}_{n+1} + \beta_n K^{-1}\boldsymbol{h}_n,)$
$\boldsymbol{d}_{n+1} = AK^{-1}\boldsymbol{r}_{n+1} + \beta_n\boldsymbol{f}_n.$
end

If we can easily use a transposed matrix-vector multiplication, then we can delete three auxiliary vectors $\boldsymbol{q}_n$, $\boldsymbol{f}_n$, and $\boldsymbol{d}_n$ from the Algorithm 5.1. Then, we have a simpler variant of the CRS method.

Algorithm 5.3: Simpler CRS method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
choose $\boldsymbol{r}_0^*$ (for example, $\boldsymbol{r}_0^* = \boldsymbol{r}_0$),
set $\boldsymbol{e}_0 = \boldsymbol{r}_0,\ \boldsymbol{d}_0 = A\boldsymbol{e}_0,\ \beta_{-1} = 0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:
$\boldsymbol{p}_n = \boldsymbol{e}_n + \beta_{n-1}(\boldsymbol{h}_{n-1} + \beta_{n-1}\boldsymbol{p}_{n-1}),$
$\alpha_n = \dfrac{(A^{\mathrm{H}}\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(A^{\mathrm{H}}\boldsymbol{r}_0^*, A\boldsymbol{p}_n)},$
$\boldsymbol{h}_n = \boldsymbol{e}_n - \alpha_n A\boldsymbol{p}_n,$

$$
\begin{aligned}
&\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n(\boldsymbol{e}_n + \boldsymbol{h}_n),\\
&\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n(\boldsymbol{d}_n + \boldsymbol{f}_n),\\
&\beta_n = \frac{(A^{\mathrm{H}}\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(A^{\mathrm{H}}\boldsymbol{r}_0^*, \boldsymbol{r}_n)},\\
&\boldsymbol{e}_{n+1} = \boldsymbol{r}_{n+1} + \beta_n \boldsymbol{h}_n.\\
&\text{end}
\end{aligned}
$$

The computational cost for CGS, CRS, and simpler CRS is given in Table 5.2. The difference between CGS and CRS is the number of AXPY. CRS requires 4 or 6 more additional AXPYs. If matrix-vector multiplications are bottle-neck in operations per iteration, then this factor is not a threat to the total computational cost per iteration. In such case, CGS and CRS have almost the same computational cost. On the other hand, CGS and simpler CRS require the same operations per iteration. If the transposed matrix-vector multiplication is readily available, a simpler CRS may be preferred because of the lower operations than CRS.

Table 5.2. Summary of operations per iteration step.

Method	Inner Product	AXPY	Matrix-Vector Product	Preconditioner Solve
CGS	2	8	2	2
CRS	2	10 or 12	2	2
Simpler CRS	2	8	2	2

5.4 Numerical experiments

In this section, we report the results of numerical experiments on a range of Matrix Market problems from the Harwell-Boeing collection [26], the NEP collection [3], the SPARSKIT collection [67], and Tim Davis's collection [21]. The iterative solvers used in the experiments are CGS and CRS, and we evaluate the two methods with respect to the number of iterations (Its), computational time (Time), and $\log_{10}$ of the true relative residual 2-norm (TRR) defined as $\log_{10} \|\boldsymbol{b} - A\boldsymbol{x}_n\|/|\boldsymbol{b}\|$. All experiments were performed on a work station with a 2.0GHz Opteron processor 846 using double precision arithmetic. Codes were written in Fortran 77 and compiled with g77 option -*O*3. In all cases the iteration was started with $\boldsymbol{x}_0 = 0$ and $\boldsymbol{r}_0^* = \boldsymbol{r}_0$ in both methods, the right-hand side $\boldsymbol{b}$ was chosen as a vector with random entries from -1 to 1, and the stopping criterion was $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq 10^{-12}$. The convergence histories show the number of iterations (on the horizontal axis) versus $\log_{10}$ of the relative residual 2-norm, $\log_{10} \|\boldsymbol{r}_n\|/\|\boldsymbol{b}\|$ (on the vertical axis).

Matrices in the experiments come from electronic circuit design (ADD20, ADD32, MEMPLUS), electrical engineering (BFW782A), finite element modeling (CAVITY05, CAV ITY10, FIDAP036), fluid dynamics (CDDE1, E20R0000, E30R0000), oil reservoir simulation (ORSIRR1, ORSIRR2, ORSREG1, SHERMAN1, SHERMAN5), partial differential equations (PDE2961), aeroelasticity (TOLS4000), and petroleum engineering (WATT1, WATT2).

Table 5.3. Matrices, their sizes (N), and numerical results of CGS and CRS.

		Its		Time [sec]		TRR	
Matrix	N	CGS	CRS	CGS	CRS	CGS	CRS
ADD20	2395	390	370	1.87E-1	1.83E-1	-12.39	-12.13
ADD32	4960	72	70	5.71E-2	5.74E-2	-12.12	-12.00
BFW782A	782	365	329	7.03E-2	6.46E-2	-10.23	-11.87
CAVITY05	1182	607	567	4.28E-1	4.08E-1	-9.69	-8.13
CAVITY10	2597	1177	878	2.24E 0	1.69E 0	-8.70	-11.46
CDDE1	961	147	116	1.88E-2	1.56E-2	-11.57	-12.00
E20R0000	4241	1851	1254	6.29E 0	4.29E 0	-7.00	-10.85
E30R0000	9661	2875	1970	2.37E 1	1.64E 1	-5.62	-10.72
FIDAP036	3079	6145	4719	8.83E 0	6.86E 0	-4.99	-9.94
MEMPLUS	17758	1357	1139	6.24E 0	5.52E 0	-9.65	-11.77
ORSIRR1	1030	1137	1030	2.03E-1	1.89E-1	-9.87	-10.58
ORSIRR2	886	890	786	1.37E-1	1.25E-1	-9.81	-12.06
ORSREG1	2205	348	317	1.27E-1	1.21E-1	-11.98	-11.98
PDE2961	2961	191	173	7.67E-2	7.41E-2	-11.79	-10.61
SHERMAN1	1000	521	500	6.62E-2	6.50E-2	-12.00	-12.02
SHERMAN5	3312	2059	1737	1.22E 0	1.08E 0	-6.86	-9.95
TOLS4000	4000	—	—	—	—	—	—
WATT1	1856	485	470	1.52E-1	1.52E-1	-6.47	-12.04
WATT2	1856	903	*667	2.88E-1	*2.19E-1	-5.05	-6.64

Table 5.4. Matrices, their sizes (N), and numerical results of CGS and CRS with $ILU(0)$.

		Its		Time [sec]		TRR	
Matrix	N	CGS	CRS	CGS	CRS	CGS	CRS
ADD20	2395	170	151	1.55E-1	1.44E-1	-12.39	-11.99
ADD32	4960	34	34	5.65E-2	6.12E-2	-12.16	-12.04
BFW782A	782	85	85	3.25E-2	3.35E-2	-12.34	-12.25
CAVITY05	1182	123	98	1.83E-1	1.51E-1	-9.52	-11.61
CAVITY10	2597	186	181	9.26E-1	9.28E-1	-10.77	-11.57
CDDE1	961	36	37	9.02E-3	9.91E-3	-12.01	-12.38
E20R0000	4241	134	127	1.39E 0	1.34E 0	-10.04	-11.90
E30R0000	9661	257	212	6.08E 0	5.06E 0	-8.62	-10.39
FIDAP036	3079	152	160	5.69E-1	6.27E-1	-9.62	-11.89
MEMPLUS	17758	327	310	2.57E 0	3.60E 0	-9.60	-11.61
ORSIRR1	1030	47	46	1.72E-2	1.76E-2	-12.54	-12.63
ORSIRR2	886	47	47	1.47E-2	1.51E-2	-12.39	-11.99
ORSREG1	2205	53	53	3.88E-2	4.08E-2	-12.32	-12.42
PDE2961	2961	42	43	3.38E-2	3.74E-2	-12.29	-12.51
SHERMAN1	1000	41	41	9.74E-3	1.06E-2	-12.06	-12.12
SHERMAN5	3312	32	32	4.35E-2	4.65E-2	-12.46	-12.47
WATT1	1856	60	59	2.59E-2	2.75E-2	-12.18	-12.39
WATT2	1856	112	110	7.67E-2	6.14E-2	-4.52	-5.92

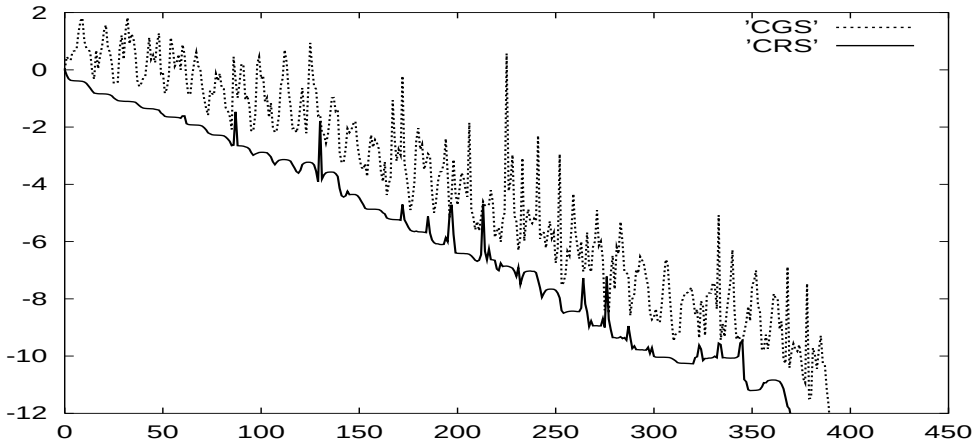

Figure 5.1: Residual 2-norm histories of CGS and CRS for ADD20.

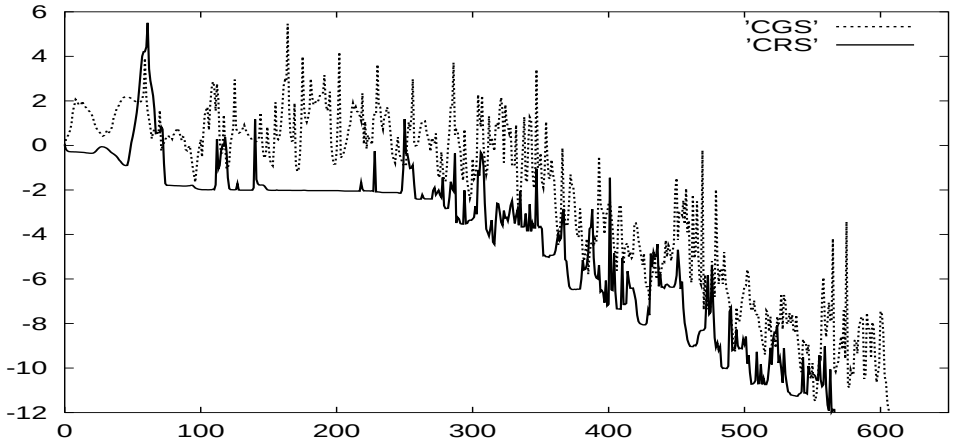

Figure 5.2: Residual 2-norm histories of CGS and CRS for CAVITY05.

◇ Comparison of CGS and CRS

We evaluate the performance of CGS and CRS with no preconditioning. The numerical results are shown in Table 5.3, where the result for TOLS4000 is not listed since CGS diverged and CRS stagnated and the two methods did not generated approximate solutions with the accuracy less than initial relative residual 2-norm.

Looking at Its, we see that CRS required only about 70-90% of the iteration steps of CGS in CAVITY10, CDDE1, E20R0000, E30R0000, FIDAP036, MEMPLUS, ORSIRR2, and SHERMAN5. However, CRS did not converge in WATT2. In the other problems, CGS and CRS required almost the same number of iteration steps.

In terms of TIME, CRS took only about 70-90% of the iteration steps of CGS in CAVITY10, CDDE1, E20R0000, E30R0000, FIDAP036, MEMPLUS, and SHERMAN5. Since the computational cost per iteration of CRS is slightly larger than that of CGS, it slightly increases the required cpu time per itaration even if the two methods need the same iteration steps.

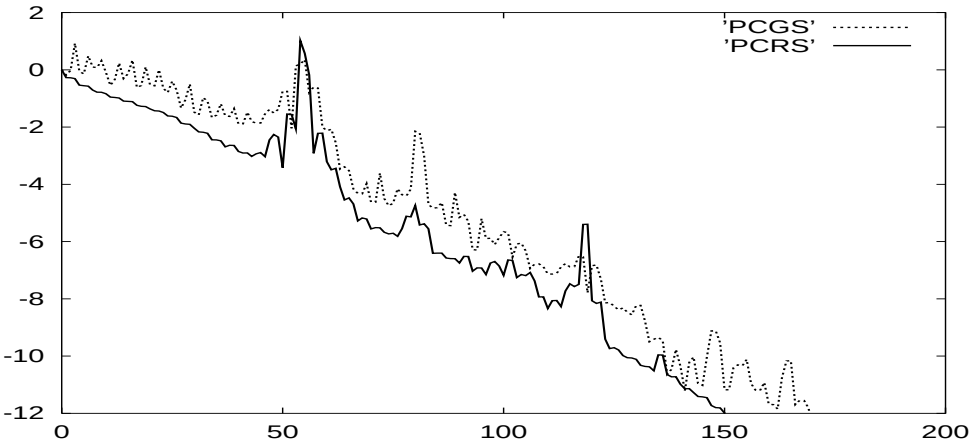

Figure 5.3: Residual 2-norm histories of CGS and CRS with $ILU(0)$ for ADD20.

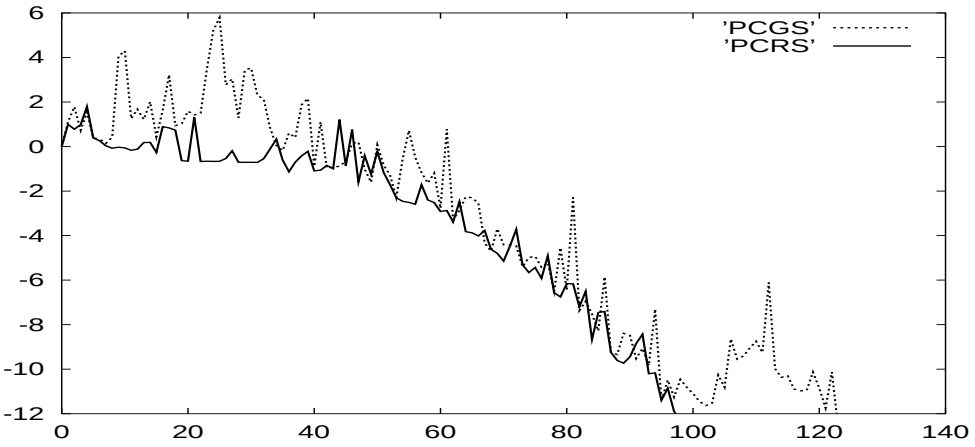

Figure 5.4: Residual 2-norm histories of CGS and CRS with $ILU(0)$ for CAVITY05.

In terms of TRR, CRS tended to give approximate solutions with much higher accuracy. Notably, in CAVITY10, MEMPLUS, ORSIRR2, and WATT1, each TRR of CRS was almost the same as the stopping criterion. On the other hand, corresponding TRRs of CGS were greater than -10. In CAVITY05 and PDE2961, TRRs of CGS were slightly better than those of CRS.

Residual 2-norm histories of CGS and CRS for ADD20 and CAVITY05, and ones with $ILU(0)$ for ADD20 and CAVITY05 are shown in Figs. 5.1, and 5.2.

◇ Comparison of CGS and CRS with $ILU(0)$ preconditioning

We evaluate the performance of CGS and CRS with $ILU(0)$ preconditioning. The numerical results are shown in Table 5.4, where the result for TOLS4000 is not listed since $ILU(0)$ caused breakdown.

With respect to Its and Time, CRS only required about 80% of the iteration steps and computational time of CGS in CAVITY05, E30R0000, and WATT2. There was little

difference in the performance of CGS and CRS in other problems except TOLS4000, since the preconditioner was quite effective in improving the convergence behavior. In terms of TRR, TRRs of CRS were much better than those of CGS in CAVITY05, E30R0000, FIDAP036, and MEMPLUS. In WATT2, TRRs of the two methods were extremely less than the stopping criterion. Hence, we see from this case that CRS does not always give an approximate solution close to stopping criterion. In other problems, the two methods generated almost the same accuracy of the approximate solutions as the stopping criterion.

Residual 2-norm histories of Bi-CG and Bi-CR with $ILU(0)$ preconditioning for ADD20, CAVITY05 are shown in Figs. 5.3, and 5.4. In Fig. 5.3, CGS gave irregular convergence behavior while CRS converged smoother and faster than CGS. In Fig. 5.4, CRS also showed smoother convergence behavior in the initial phase than CGS.

◇ The Helmholtz equation

We consider the following boundary value problem:

$$\begin{array}{ll} u_{xx} + u_{yy} + \sigma^2 u = 0, & (x,y) \in [0,\pi] \times [0,\pi], \\ u_x|_{x=0} = i\sqrt{\sigma^2 - \frac{1}{4}}\cos\frac{y}{2}, & \text{Neumann Condition (1)}, \\ u_x - i\sqrt{\sigma^2 - \frac{1}{4}}u|_{x=\pi} = 0, & \text{Radiation Condition}, \\ u_y|_{y=0} = 0, & \text{Neumann Condition (2)}, \\ u|_{y=\pi} = 0, & \text{Dirichlet Condition}. \end{array}$$

The above elliptic problem is known as the Helmholtz equation [6]. Here we show that the approximate solution is obtained by solving sparse linear systems arising from the finite difference discretization of the above equation. The first step is to choose approximate formulas for the derivatives, we use the standard formula

$$u_{xx}(x,y) \approx \frac{1}{h^2}\Big\{u(x+h,y) - 2u(x,y) + u(x-h,y)\Big\},$$
$$u_{yy}(x,y) \approx \frac{1}{h^2}\Big\{u(x,y+h) - 2u(x,y) + u(x,y+h)\Big\}.$$

Now, it is convenient to introduce an abbreviated notation $u_{i,j} = u(x_i, y_j)$. Then we obtain five-point formula

$$u_{xx}(x_i,y_j) + u_{yy}(x_i,y_j) \approx \frac{1}{h^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}).$$

If this approximation is made in without boundary condition, the result is

$$-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + (4 - h^2\sigma^2)u_{i,j} = 0. \tag{5.15}$$

The coefficient of this equation can be illustrated by a five-point star in which each point correspond to the coefficient of u in the grid (Fig. 5.5). To be specific, we consider the region is square $(x,y) \in [0,\pi] \times [0,\pi]$ and the grid has spacing $h = \pi/2$ (See Fig. 5.6). We obtain a single linear equation of the form (5.15) for each of the six grid points. These six equations are shown as follows:

$$\begin{aligned} -u_2 - g_4 - u_4 - g_1 + (4 - h^2\sigma^2)u_1 &= 0, \\ -u_3 - u_1 - u_5 - g_2 + (4 - h^2\sigma^2)u_2 &= 0, \\ -g_5 - u_2 - u_6 - g_3 + (4 - h^2\sigma^2)u_3 &= 0, \end{aligned} \tag{5.16}$$

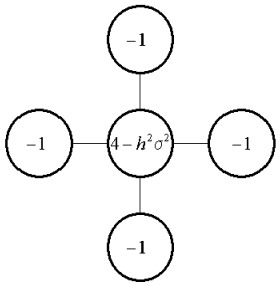

Figure 5.5: Five-point star.

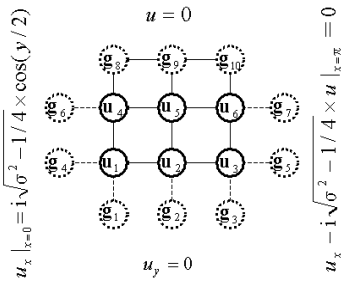

Figure 5.6: Mesh points on the region.

$$\begin{aligned} -u_5 - g_6 - g_8 - u_1 + (4 - h^2\sigma^2)u_4 &= 0, \\ -u_6 - u_4 - g_9 - u_2 + (4 - h^2\sigma^2)u_5 &= 0, \\ -g_7 - u_5 - g_{10} - u_3 + (4 - h^2\sigma^2)u_3 &= 0, \end{aligned}$$

where $u_1 = u_{1,1},\ u_2 = u_{2,1},\ u_3 = u_{3,1},\ u_4 = u_{2,1},\ u_5 = u_{2,2},\ u_6 = u_{2,3}$. Unknowns g_k for $1 \le k \le 10$ can be computed by the four boundary conditions: first, from Neumann Condition (2) we obtain $g_1,\ g_2$, and g_3 below.

$$\frac{u_4 - g_1}{2h} = \frac{u_5 - g_2}{2h} = \frac{u_6 - g_3}{2h} = 0,$$

or equivalently,

$$g_1 = u_4, \quad g_2 = u_5, \quad g_3 = u_6.$$

Second, from Neumann Condition (1), the right-hand side directly depends on the condition, g_4 and g_6 are

$$\frac{u_2 - g_4}{2h} = id\cos(\frac{0}{2}), \quad \frac{u_5 - g_6}{2h} = id\cos(\frac{h}{2}).$$

or equivalently,

$$g_4 = u_2 - 2idh\cos(\frac{0}{2}), \quad g_6 = u_5 - 2idh\cos(\frac{h}{2}),$$

where $d = \sqrt{\sigma^2 - 1/4}$. Third, from Radiation Condition it follows that

$$\frac{g_5 - u_2}{2h} - idu_3 = \frac{g_7 - u_5}{2h} - idu_6 = 0$$

or equivalently,

$$g_5 = u_2 + 2idhu_3, \quad g_7 = u_5 + 2idhu_6.$$

Finally, from Dirichlet Condition $g_8,\ g_9$ and g_{10} become automatically

$$g_8 = g_9 = g_{10} = 0.$$

Substituting $g_1, g_2, \ldots, g_{10}$ into the six equations (5.16) the following linear systems are obtained as follows:

$$\begin{pmatrix} a & -2 & 0 & -2 & 0 & 0 \\ -1 & a & -1 & 0 & -2 & 0 \\ 0 & -2 & a - idh & 0 & 0 & -2 \\ -1 & 0 & 0 & a & -2 & 0 \\ 0 & -1 & 0 & -1 & a & -1 \\ 0 & 0 & -1 & 0 & -2 & a - idh \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} = \begin{pmatrix} f_1 \\ 0 \\ 0 \\ f_4 \\ 0 \\ 0 \end{pmatrix}, \tag{5.17}$$

where $a = 4 - h^2\sigma^2$, $f_1 = -2idh\cos(\frac{0}{2})$, and $f_4 = -2idh\cos(\frac{h}{2})$. The above linear systems (5.17) is not symmetric. Generally speaking, it is harder to solve unsymmetric linear systems compared with symmetric ones. Therefore, we symmetrize the coefficient matrix by scaling the form (5.17):

$$(5.18)\quad \begin{pmatrix} a/4 & -1/2 & 0 & -1/2 & 0 & 0 \\ -1/2 & a/2 & -1/2 & 0 & -1 & 0 \\ 0 & -1/2 & (a-idh)/4 & 0 & 0 & -1/2 \\ -1/2 & 0 & 0 & a/2 & -1 & 0 \\ 0 & -1 & 0 & -1 & a & -1 \\ 0 & 0 & -1/2 & 0 & -1 & (a-idh)/2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} = \begin{pmatrix} f_1/4 \\ 0 \\ 0 \\ f_4/2 \\ 0 \\ 0 \end{pmatrix}.$$

The coefficient matrix of the form (5.18) is non-Hermitian but symmetric.

In our experiments a square grid ($M \times M$) was used. Taking boundary condition into account, the size of the linear system is $M \times (M+1)$. The experiments were made on the same conditions as previous subsection. Note that the right-hand side is determined by the Neumann condition (1).

We tested four iterative methods (CGS, CRS, Bi-CGSTAB, and GPBi-CG) with no preconditioning in the case of $M = 50$ for $\sigma = 2.27, 4.16$ and $M = 100$ for $\sigma = 2.27, 4.16$. Numerical results are shown in Table 5.5. "STAB" and "GPB" are short for Bi-CGSTAB and GPBi-CG.

We see from Table 5.5 that the larger the number of dimension of the matrix or the value of σ becomes, the larger the required number of iterations tends to increase. In every case, CRS was the fastest of all. The runner-up is CGS. However, in terms of TRR, CGS gave an approximate solution with worse accuracy. Especially in $M = 100, \sigma = 4.16$, TRR of CGS is -5.24. The accuracy of the solution may be too far away from that often required in scientific computations. On the other hand, CRS gave much better solutions than CGS. In the case of which Bi-CGSTAB converged, it also gave better solution than CGS. GPBi-CG showed well performances in all cases while it generated a worse solution for the case $M = 100, \sigma = 2.27$.

Residual 2-norm histories of the four methods are shown in Figs. 5.7, 5.8, 5.9, 5.10. We see from the all Figs that CRS showed smoother convergence behavior than CGS in the

Table 5.5. Results for the elliptic problems (M=grid size, N= order of matrix) and convergence results (Its = number of iterations, TRR = $\log_{10}$ of the final true relative residual 2-norm) for the iterative methods.

			Its				TRR			
M	N	σ	CGS	CRS	STAB	GPB	CGS	CRS	STAB	GPB
50	2550	2.27	485	429	1025	574	-7.68	-11.34	-11.64	-10.84
50	2550	4.16	933	704	3283	1016	-7.96	-11.65	-11.92	-10.10
100	10100	2.27	1185	908	3157	987	-5.66	-10.56	-11.33	-7.38
100	10100	4.16	2124	1572	†	2336	-5.24	-10.32	-5.53	-10.04

initial stage of convergence. In the middle and the last stage of convergence, CGS and CRS gave similar convergence behavior. On the other hand, Bi-CGSTAB and GPBi-CG showed much smoother convergence behavior than CGS and CRS while they required more iteration steps. Such convergence behavior of Bi-CGSTAB and GPBi-CG comes from the local minimization of the residual 2-norm. Since Bi-CGSTAB and GPBi-CG use one and two dimensional local minimization techniques respectively, GPBi-CG tends to give smoother convergence behavior than Bi-CGSTAB.

5.5 Concluding remarks

We see from the results of many numerical experiments that CRS often gives smoother convergence behavior and generates better approximate solutions than CGS. The main reason for the smoothness could come from the convergence behavior of Bi-CR whose residual 2 norm is often less than that of Bi-CG. This smoother behavior is remarkable since this approach is not based on local minimization of the residual vectors. From another angle, the product-type method based on Bi-CR may give a paradigm shift; recent main theme on the product-type methods based on Bi-CG is finding an appropriate polynomial H_n to give better residual vectors $\boldsymbol{r}_n (= H_n \boldsymbol{r}_n^{\mathrm{BiCG}})$. However, we focus on not finding H_n but an alternative basic solver for Bi-CG.

The reason why CRS generates better approximate solutions than CGS could be the smoother convergence behavior. In [80], it is shown that irregular convergence tends to lead worse approximate solutions. More precisely, the accuracy depends largely on the maximum residual 2-norm in the convergence process. Actually, we can see that log_{10} of the maximum residual 2-norm of CGS is about 8 in Figs. 5.7, 5.8 and about 10 in Figs. 5.9, 5.10. Hence, the difference is about 2. We can see from Table 5.5 that the difference between the corresponding TRRs is also about 2.

From the results of numerical experiments and the numerical property, we can say that CRS may be a useful tool for solving non-Hermitian linear systems arising from scientific computing.

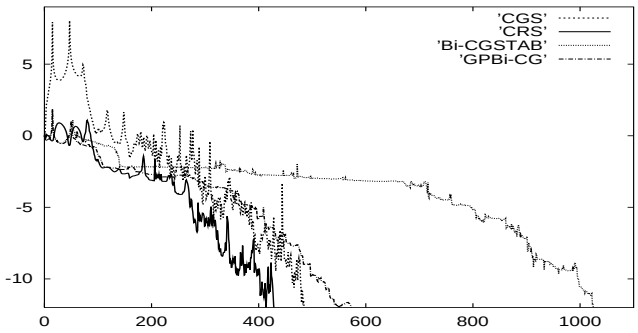

Figure 5.7: Residual 2-norm histories for $M = 50$, $\sigma = 2.27$.

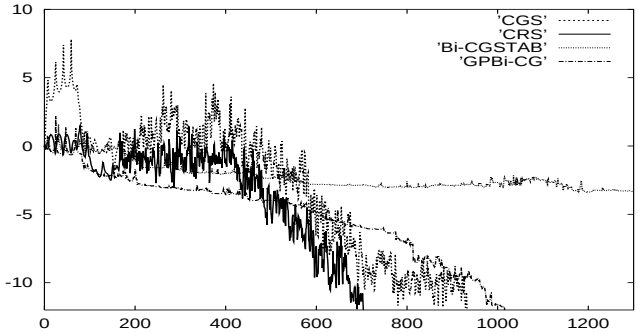

Figure 5.8: Residual 2-norm histories for $M = 50$, $\sigma = 4.16$.

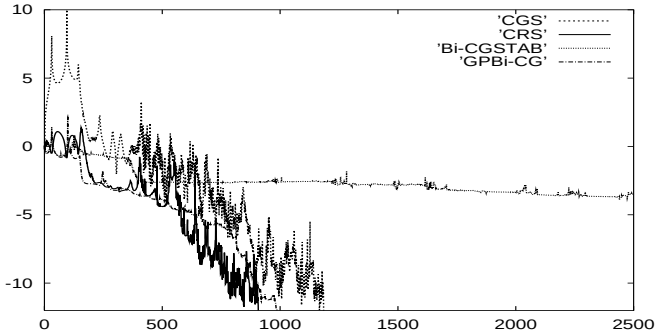

Figure 5.9: Residual 2-norm histories for $M = 100$, $\sigma = 2.27$.

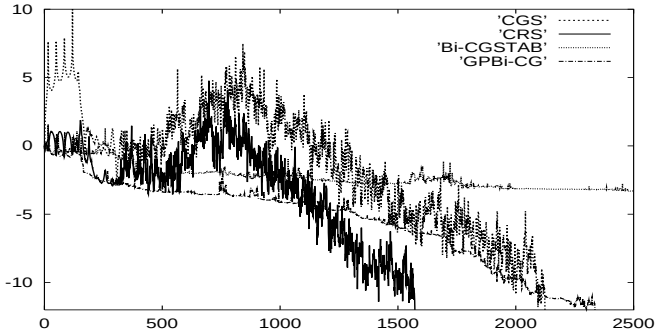

Figure 5.10: Residual 2-norm histories for $M = 100$, $\sigma = 4.16$.

Chapter 6

SCGS: a stabilized CGS method

The CGS method is a well-known Krylov subspace method for solving large and sparse non-Hermitian linear systems. Although there are some practical situations where CGS converges faster than the other product-type methods such as Bi-CGSTAB and GPBi-CG, even in such case it often shows rather irregular convergence behavior and gives unreliable solutions. In this chapter, to improve its convergence behavior, a new iterative method (Stabilized CGS, or SCGS) is proposed by a simple generalization of CGS.

6.1 Definition of SCGS

Let us define the new residual vector $\boldsymbol{r}_n^{\mathrm{SCGS}}$ by the product of the Bi-CG polynomial and a polynomial H_n of degree n as follows:

$$\boldsymbol{r}_n^{\mathrm{SCGS}} := H_n(A)R_n(A)\boldsymbol{r}_0,$$

where H_n denotes

$$\begin{aligned} H_0(\lambda) &:= 1, \\ H_n(\lambda) &:= R_{n-1}(\lambda) - \omega_{n-1}\lambda P_{n-1}(\lambda), \quad n = 1, 2, \ldots \end{aligned}$$

Here, R_n and P_n are polynomials described in (2.25) and (2.26). From the above, the choice $\omega_{n-1} = \alpha_{n-1}$ leads to CGS. Hence, SCGS can be regarded as a generalization of CGS.

6.2 Recurrence formulas for iterates

Since R_n and H_n are given, recurrence relations for updating the residual vector can be obtained. By using polynomials $H_{n+1}R_{n+1}$, $R_{n+1}R_{n+1}$, $R_{n+1}P_{n+1}$, P_nR_{n+1}, and $P_{n+1}P_{n+1}$, the residual polynomial $H_{n+1}R_{n+1}$ can be expanded as follows:

$$\begin{aligned} H_{n+1}R_{n+1} &= (R_n - \omega_n\lambda P_n)R_{n+1} \\ &= R_nR_n - \alpha_n\lambda R_nP_n - \omega_n\lambda P_nR_{n+1}, \qquad (6.1) \\ R_{n+1}R_{n+1} &= R_nR_n - \alpha_n\lambda R_nP_n - \alpha_n\lambda P_nR_{n+1}, \qquad (6.2) \\ R_{n+1}P_{n+1} &= R_{n+1}(R_{n+1} + \beta_nP_n) \\ &= R_{n+1}R_{n+1} + \beta_nP_nR_{n+1}, \qquad (6.3) \\ P_nR_{n+1} &= P_nR_n - \alpha_n\lambda P_nP_n, \qquad (6.4) \end{aligned}$$

$$
\begin{aligned}
P_{n+1}P_{n+1} &= P_{n+1}R_{n+1} + \beta_n P_{n+1}P_n \\
&= P_{n+1}R_{n+1} + \beta_n(P_nR_{n+1} + \beta_n P_nP_n), \qquad (6.5)
\end{aligned}
$$

where $\boldsymbol{r}_n^{\mathrm{CGS}}$ denotes the residual vector of CGS. Let us define new auxiliary vectors by

$$
\begin{aligned}
&\boldsymbol{r}_n^{\mathrm{CGS}} := R_n(A)R_n(A)\boldsymbol{r}_0, \quad && \boldsymbol{e}_n := R_n(A)P_n(A)\boldsymbol{r}_0, \\
&\boldsymbol{h}_n := P_n(A)R_{n+1}(A)\boldsymbol{r}_0, && \boldsymbol{p}_n := P_n(A)P_n(A)\boldsymbol{r}_0, \\
&\boldsymbol{r}_n^{\mathrm{SCGS}} := H_n(A)R_n(A)\boldsymbol{r}_0. &&
\end{aligned}
$$

Then, it follows from (6.1)-(6.5) that

$$
\begin{aligned}
\boldsymbol{p}_n &= \boldsymbol{e}_n + \beta_{n-1}(\boldsymbol{h}_{n-1} + \beta_{n-1}\boldsymbol{p}_{n-1}), \\
\boldsymbol{h}_n &= \boldsymbol{e}_n - \alpha_n A\boldsymbol{p}_n, \\
\boldsymbol{r}_{n+1}^{\mathrm{SCGS}} &= \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n A\boldsymbol{e}_n - \omega_n A\boldsymbol{h}_n, \qquad (6.6) \\
\boldsymbol{r}_{n+1}^{\mathrm{CGS}} &= \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n A\boldsymbol{e}_n - \alpha_n A\boldsymbol{h}_n, \qquad (6.7) \\
\boldsymbol{e}_{n+1} &= \boldsymbol{r}_{n+1}^{\mathrm{CGS}} + \beta_n \boldsymbol{h}_n.
\end{aligned}
$$

From (6.6) and (6.7), the following formula for the approximate solution can be obtained:

$$
\begin{aligned}
\boldsymbol{x}_{n+1} &= \boldsymbol{z}_n + \alpha_n\boldsymbol{e}_n + \omega_n\boldsymbol{h}_n, \\
\boldsymbol{z}_{n+1} &= \boldsymbol{z}_n + \alpha_n\boldsymbol{e}_n + \alpha_n\boldsymbol{h}_n,
\end{aligned}
$$

where $\boldsymbol{x}_{n+1}$ denotes the approximate solution of SCGS and $\boldsymbol{z}_{n+1}$ denotes that of CGS.

6.3 Computational formulas for α_n and β_n

In the previous section, we obtained computational formulas for updating residual vectors of SCGS. In this section, based on α_n and β_n used in the Bi-CG polynomial, we consider computational formulas for α_n and β_n using SCGS iterates.

We see from (B.18), (B.20) that the following inner products are needed for computing α_n and β_n:

$$
((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, R_n(A)\boldsymbol{r}_0), \quad ((A^{\mathrm{H}})^n\boldsymbol{r}_0^*, AP_n(A)\boldsymbol{r}_0). \qquad (6.8)
$$

Here, let us introduce an n degree polynomial of the form

$$
G_n(\lambda) := \sum_{k=0}^{n} c_k\lambda^k,
$$

where the coefficient $c_n \neq 0$. Then, the inner products (6.8) can be computed as follows:

$$
\begin{aligned}
(\bar{G}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*, R_n(A)\boldsymbol{r}_0) &= (\sum_{k=0}^{n}\bar{c}_k(A^{\mathrm{H}})^k\boldsymbol{r}_0^*, R_n(A)\boldsymbol{r}_0) \\
&= (\bar{c}_n(A^{\mathrm{H}})^n\boldsymbol{r}_0^*, R_n(A)\boldsymbol{r}_0) + (\sum_{k=0}^{n-1}\bar{c}_k(A^{\mathrm{H}})^k\boldsymbol{r}_0^*, R_n(A)\boldsymbol{r}_0).
\end{aligned}
$$

Since we see from the Petrov-Galerkin condition (2.4) that the second term of the right-hand side is zero, we obtain

$$
\begin{aligned}
((A^{\mathrm{H}})^n\boldsymbol{r}_0^*,\ R_n(A)\boldsymbol{r}_0) &= \frac{1}{c_n}\cdot(\bar{G}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*,\ R_n(A)\boldsymbol{r}_0) \qquad (6.9) \\
&= \frac{1}{c_n}\cdot(\boldsymbol{r}_0^*,\ G_n(A)R_n(A)\boldsymbol{r}_0).
\end{aligned}
$$

From (B.19), we also have

$$
\begin{aligned}
(6.10) \qquad ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*,\ AP_n(A)\boldsymbol{r}_0) &= \frac{1}{c_n}\cdot(\bar{G}_n(A^{\mathrm{T}})\boldsymbol{r}_0^*,\ AP_n(A)\boldsymbol{r}_0) \\
&= \frac{1}{c_n}\cdot(\boldsymbol{r}_0^*,\ AG_n(A)P_n(A)\boldsymbol{r}_0).
\end{aligned}
$$

In the SCGS method, there are three kinds of polynomials R_n, P_n, and H_n, and the coefficients of the highest-order term of R_n and P_n are

$$
\sigma_n = (-1)^n \prod_{k=0}^{n-1} \alpha_k.
$$

Similarly, that of H_n is

$$
\tau_n = (-1)^n \omega_{n-1} \prod_{k=0}^{n-2} \alpha_k.
$$

Thus from (6.9) and (6.10), the inner products (6.8) can be computed as follows:

$$
\begin{aligned}
(6.11) \qquad ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*,\ R_n(A)\boldsymbol{r}_0) &= \frac{1}{\sigma_n}\cdot(\boldsymbol{r}_0^*, R_n(A)R_n(A)\boldsymbol{r}_0) \\
&= \frac{1}{\sigma_n}\cdot(\boldsymbol{r}_0^*, P_n(A)R_n(A)\boldsymbol{r}_0) \\
&= \frac{1}{\tau_n}\cdot(\boldsymbol{r}_0^*, H_n(A)R_n(A)\boldsymbol{r}_0), \\
(6.12) \qquad ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*,\ AP_n(A)\boldsymbol{r}_0) &= \frac{1}{\sigma_n}\cdot(\boldsymbol{r}_0^*, AR_n(A)P_n(A)\boldsymbol{r}_0) \\
&= \frac{1}{\sigma_n}\cdot(\boldsymbol{r}_0^*, AP_n(A)P_n(A)\boldsymbol{r}_0).
\end{aligned}
$$

To compute (6.11) and (6.12), the following auxiliary vectors can be used:

$$
\begin{aligned}
\boldsymbol{r}_n^{\mathrm{CGS}} &= R_n(A)R_n(A)\boldsymbol{r}_0, \quad \boldsymbol{e}_n = R_n(A)P_n(A)\boldsymbol{r}_0, \\
\boldsymbol{h}_n &= P_n(A)R_{n+1}(A)\boldsymbol{r}_0, \quad \boldsymbol{p}_n = P_n(A)P_n(A)\boldsymbol{r}_0.
\end{aligned}
$$

Therefore, there are six patterns for computing α_n and ten patterns for computing β_n.

$$
\begin{aligned}
\alpha_n &= \rho_0 \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)} = \rho_0 \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{e}_n)} \\
&= \frac{(\boldsymbol{r}_0^*, \boldsymbol{e}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)} = \frac{(\boldsymbol{r}_0^*, \boldsymbol{e}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{e}_n)} \\
&= \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)} = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{e}_n)},
\end{aligned}
$$

$$
\begin{aligned}
\beta_n &= \frac{1}{\omega_n}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)} = \frac{1}{\omega_n}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{e}_n)} \\
&= \frac{1}{\alpha_n}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)} = \frac{1}{\alpha_n}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{e}_n)} \\
&= \frac{1}{\rho_0}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{SCGS}})} = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{e}_n)} = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{CGS}})} \\
&= \frac{\rho_1}{\rho_0}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{SCGS}})} = \rho_1\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{e}_n)} = \rho_1\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{CGS}})},
\end{aligned}
$$

where

$$\rho_0 = \frac{\alpha_{n-1}}{\omega_{n-1}}, \quad \rho_1 = \frac{\alpha_n}{\omega_n}.$$

Although these parameters are mathematically equivalent, these convergence histories are quite different due to a rounding error. Hence, by many numerical experiments, we found a pair of parameters, α_n and β_n, that were often numerically stabler than the others.

6.4 Implementation

In the SCGS method, ω_{n-1} is determined by minimizing 2-norm of the residual vector $\boldsymbol{r}_n^{\mathrm{SCGS}}$, i.e.

$$\omega_{n-1} = \arg\min_{\omega \in \mathbb{C}} \|\boldsymbol{r}_n^{\mathrm{SCGS}}\|.$$

From the above choice, we have the following result:

Theorem 6.4.1 *In exact precision arithmetic, SCGS always converges faster than CGS:*

$$\|\boldsymbol{r}_n^{SCGS}\| \leq \|\boldsymbol{r}_n^{CGS}\|.$$

Proof. Let $f(\omega)$ be a function of ω

$$f(\omega) = \|\boldsymbol{r}_{n-1}^{\mathrm{CGS}} - \alpha_{n-1} A\boldsymbol{e}_{n-1} - \omega A\boldsymbol{h}_{n-1}\|.$$

Then, from the definitions of $\boldsymbol{r}_n^{\mathrm{SCGS}}$ and ω_{n-1}, it follows that

$$\|\boldsymbol{r}_n^{\mathrm{SCGS}}\| = \min_{\omega} \|f(\omega)\| \leq \|f(\alpha_{n-1})\| = \|\boldsymbol{r}_n^{\mathrm{CGS}}\|.$$

Thus, if CGS converges, SCGS does in fewer number of iterations. □

Now, let us introduce the new auxiliary vector

$$\boldsymbol{q}_n = AP_n(A)P_n(A)\boldsymbol{r}_0, \tag{6.13}$$

then we obtain the SCGS method as follows:

Algorithm 6.1: SCGS method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0^{\mathrm{SCGS}} = \boldsymbol{b} - A\boldsymbol{x}_0,\ \boldsymbol{z}_0 = \boldsymbol{x}_0,$
set $\boldsymbol{r}_0^* = \boldsymbol{e}_0 = \boldsymbol{r}_0^{\mathrm{CGS}} = \boldsymbol{r}_0^{\mathrm{SCGS}},\ \beta_{-1} = 0,\ \rho_0 = 1,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n^{\mathrm{SCGS}}\| \leq \epsilon\|\boldsymbol{b}\|$ do:

$$\begin{aligned}
&\boldsymbol{q}_n = A\boldsymbol{e}_n + \beta_{n-1}(A\boldsymbol{h}_{n-1} + \beta_{n-1}\boldsymbol{q}_{n-1}),\\
&\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, A\boldsymbol{e}_n)},\\
&\boldsymbol{h}_n = \boldsymbol{e}_n - \alpha_n \boldsymbol{q}_n,\\
&A\boldsymbol{h}_n = A(\boldsymbol{e}_n + \boldsymbol{h}_n) - A\boldsymbol{e}_n,\\
&\omega_n = \frac{(A\boldsymbol{h}_n, \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n A\boldsymbol{e}_n)}{(A\boldsymbol{h}_n, A\boldsymbol{h}_n)}, \quad \rho_1 = \frac{\alpha_n}{\omega_n},
\end{aligned}$$

$$
\begin{aligned}
&\boldsymbol{r}_{n+1}^{\mathrm{SCGS}} = \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n A\boldsymbol{e}_n - \omega_n A\boldsymbol{h}_n,\\
&\quad \text{if } \|\boldsymbol{r}_{n+1}^{\mathrm{SCGS}}\| \le \epsilon\|\boldsymbol{r}_0^{\mathrm{SCGS}}\|, \text{ then}\\
&\qquad \boldsymbol{x}_{n+1} = \boldsymbol{z}_n + \alpha_n\boldsymbol{e}_n + \omega_n\boldsymbol{h}_n,\\
&\quad \text{end if and stop}\\
&\boldsymbol{r}_{n+1}^{\mathrm{CGS}} = \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n A(\boldsymbol{e}_n + \boldsymbol{h}_n),\\
&\boldsymbol{z}_{n+1} = \boldsymbol{z}_n + \alpha_n(\boldsymbol{e}_n + \boldsymbol{h}_n),\\
&\beta_n = \frac{\rho_1}{\rho_0}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{SCGS}})}, \quad \rho_0 = \rho_1,\\
&\boldsymbol{e}_{n+1} = \boldsymbol{r}_{n+1}^{\mathrm{CGS}} + \beta_n\boldsymbol{h}_n.\\
&\text{end}
\end{aligned}
$$

When SCGS is applied to the linear system (2.15), the rewrites

$$
\begin{array}{lll}
\tilde{\boldsymbol{x}}_n \Rightarrow K_2\boldsymbol{x}_n, & \tilde{\boldsymbol{z}}_n \Rightarrow K_2\boldsymbol{z}_n, & \tilde{\boldsymbol{e}}_n \Rightarrow K_1^{-1}\boldsymbol{e}_n,\\
\tilde{\boldsymbol{h}}_n \Rightarrow K_1^{-1}\boldsymbol{h}_n, & \tilde{\boldsymbol{q}}_n \Rightarrow K_1^{-1}\boldsymbol{q}_n, & \tilde{\boldsymbol{r}}_n^{\mathrm{SCGS}} \Rightarrow K_1^{-1}\boldsymbol{r}_n^{\mathrm{SCGS}},\\
\tilde{\boldsymbol{r}}_n^{\mathrm{CGS}} \Rightarrow K_1^{-1}\boldsymbol{r}_n^{\mathrm{CGS}}, & \tilde{\boldsymbol{r}}_0^* \Rightarrow K_1^{\mathrm{H}}\boldsymbol{r}_0^*. &
\end{array}
$$

lead to the preconditioned SCGS method.

Algorithm 6.2: Preconditioned SCGS method

$$
\begin{aligned}
&\boldsymbol{x}_0 \text{ is an initial guess, } \boldsymbol{r}_0^{\mathrm{SCGS}} = \boldsymbol{b} - A\boldsymbol{x}_0,\ \boldsymbol{z}_0 = \boldsymbol{x}_0,\\
&\text{set } \boldsymbol{r}_0^* = \boldsymbol{e}_0 = \boldsymbol{r}_0^{\mathrm{CGS}} = \boldsymbol{r}_0^{\mathrm{SCGS}},\ \beta_{-1} = 0,\ \rho_0 = 1,\\
&\text{for } n = 0, 1, \ldots, \text{until } \|\boldsymbol{r}_n^{\mathrm{SCGS}}\| \le \epsilon\|\boldsymbol{b}\| \text{ do:}\\
&\quad \boldsymbol{q}_n = AK^{-1}\boldsymbol{e}_n + \beta_{n-1}(AK^{-1}\boldsymbol{h}_{n-1} + \beta_{n-1}\boldsymbol{q}_{n-1}),\\
&\quad \alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{CGS}})}{(\boldsymbol{r}_0^*, AK^{-1}\boldsymbol{e}_n)},\\
&\quad \boldsymbol{h}_n = \boldsymbol{e}_n - \alpha_n\boldsymbol{q}_n,\\
&\quad AK^{-1}\boldsymbol{h}_n = AK^{-1}(\boldsymbol{e}_n + \boldsymbol{h}_n) - AK^{-1}\boldsymbol{e}_n,\\
&\quad \omega_n = \frac{(AK^{-1}\boldsymbol{h}_n, \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n AK^{-1}\boldsymbol{e}_n)}{(AK^{-1}\boldsymbol{h}_n, AK^{-1}\boldsymbol{h}_n)}, \quad \rho_1 = \frac{\alpha_n}{\omega_n},\\
&\quad \boldsymbol{r}_{n+1}^{\mathrm{SCGS}} = \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n AK^{-1}\boldsymbol{e}_n - \omega_n AK^{-1}\boldsymbol{h}_n,\\
&\qquad \text{if } \|\boldsymbol{r}_{n+1}^{\mathrm{SCGS}}\| \le \epsilon\|\boldsymbol{r}_0^{\mathrm{SCGS}}\|, \text{ then}\\
&\qquad\quad K^{-1}\boldsymbol{h}_n = K^{-1}(\boldsymbol{h}_n + \boldsymbol{e}_n) - K^{-1}\boldsymbol{e}_n,\\
&\qquad\quad \boldsymbol{x}_{n+1} = \boldsymbol{z}_n + \alpha_n K^{-1}\boldsymbol{e}_n + \omega_n K^{-1}\boldsymbol{h}_n,\\
&\qquad \text{end if and stop}\\
&\quad \boldsymbol{r}_{n+1}^{\mathrm{CGS}} = \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n AK^{-1}(\boldsymbol{e}_n + \boldsymbol{h}_n),\\
&\quad \boldsymbol{z}_{n+1} = \boldsymbol{z}_n + \alpha_n K^{-1}(\boldsymbol{e}_n + \boldsymbol{h}_n),\\
&\quad \beta_n = \frac{\rho_1}{\rho_0}\cdot\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}^{\mathrm{SCGS}})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\mathrm{SCGS}})}, \quad \rho_0 = \rho_1,\\
&\quad \boldsymbol{e}_{n+1} = \boldsymbol{r}_{n+1}^{\mathrm{CGS}} + \beta_n\boldsymbol{h}_n.\\
&\text{end}
\end{aligned}
$$

Let us show the computational costs of product-type methods per iteration in Table. 6.1.

Table 6.1. Summary of operations for iterations.

Method	Inner Product	AXPY	Matrix-Vector Product	Preconditioner Solve
CGS	2	6	2	2
SCGS	5	8	2	2

where AXPY is short for operations + vector scaling, and the approximate solution $\boldsymbol{x}_{n+1}$ does not need to update at every step but only one time when 2-norm of relative residual $\|\boldsymbol{r}_{n+1}^{\mathrm{SCGS}}\|/\|\boldsymbol{b}\|$ satisfies a stopping criterion.

A generalization of the SCGS method

From the idea of SCGS, we can naturally consider a generalization of it. To achieve this, let us define the polynomial H_n by

$$\begin{aligned} &H_0(\lambda) := 1, \quad G_0(\lambda) = \zeta_0, \\ &H_n(\lambda) := R_{n-1}(\lambda) - \lambda G_{n-1}(\lambda), \\ &G_n(\lambda) := \zeta_n R_n(\lambda) + \eta_n P_{n-1}(\lambda), \quad n = 1, 2, \ldots \end{aligned}$$

Then, we can see that the above polynomial is similar to the one used by product-type methods based on Bi-CG and Bi-CR. If we define the nth residual vector by

$$\boldsymbol{r}_n := H_n(A)\boldsymbol{r}_n^{\mathrm{BiCG}},$$

then we readily obtain CGS from the choice $\zeta_n = \alpha_n,\ \eta_n = \alpha_n\beta_{n-1}$ and obtain SCGS from the choice $\eta = \beta_{n-1}\zeta_n,\ \zeta_n = \arg\min\|\boldsymbol{r}_n\|$. Hence, the above framework is regarded as a generalization of CGS and SCGS. Similar to the choice for SCGS, we adopt the following choice:

$$\zeta_n, \eta_n = \arg \min_{\zeta_n, \eta_n \in \mathcal{C}} \|\boldsymbol{r}_n\|.$$

We call the resulting algorithm SCGS2. The relationship among CGS, SCGS, and SCGS2 is given in Table 6.2.

Table 6.2. The choice for CGS, SCGS, and SCGS2.

CGS	$\zeta_n = \alpha_n, \quad \eta_n = \alpha_n\beta_{n-1} \iff H_n = R_n$
SCGS	$\zeta_n = \arg \min_{\zeta_n \in \mathcal{C}} \|\boldsymbol{r}_{n+1}\|, \quad \eta_n = \beta_{n-1}\zeta_n$
SCGS2	$\zeta_n, \eta_n = \arg \min_{\zeta_n, \eta_n \in \mathcal{C}} \|\boldsymbol{r}_{n+1}\|$

By the choice for SCGS2, we obtain the following theorem:

Theorem 6.4.2 *In exact precision arithmetic, SCGS2 always converges faster than CGS and SCGS:*

$$\|\boldsymbol{r}_n\| \leq \|\boldsymbol{r}_n^{SCGS}\| \leq \|\boldsymbol{r}_n^{CGS}\|.$$

Proof. Let $f(\zeta, \eta)$ be a function of ζ and η

$$f(\zeta, \eta) = \|R_{n-1}\boldsymbol{r}_n^{\mathrm{BiCG}} - \zeta A R_{n-1}\boldsymbol{r}_n^{\mathrm{BiCG}} - \eta A P_{n-2}\boldsymbol{r}_n^{\mathrm{BiCG}}\|.$$

From the choice for CGS, SCGS, and SCGS2 given in Table 6.2, it follows that

$$\min_{\zeta,\eta} f(\zeta, \eta) \le \min_{\zeta} f(\zeta, \beta_{n-2}\zeta) \le f(\alpha_{n-1}, \alpha_{n-1}\beta_{n-2}) \Longleftrightarrow \|\boldsymbol{r}_n\| \le \|\boldsymbol{r}_n^{\mathrm{SCGS}}\| \le \|\boldsymbol{r}_n^{\mathrm{CGS}}\|.$$

Thus if CGS converges, SCGS2 does in fewer number of iterations than CGS and SCGS. □

Recurrence formulas for iterates

We see from Theorem 6.4.2 that in exact arithmetic SCGS2 gives smoother convergence behavior than CGS and SCGS. Hence, it is worth designing the algorithm of SCGS2. In the following, we give update formulas for the residual vector and the approximate solution of SCGS2. From the definition of SCGS2 and CGS polynomial, we have

$$\begin{aligned}
H_{n+1}R_{n+1} &= R_nR_{n+1} - \zeta_n\lambda R_nR_{n+1} - \eta_n\underline{\lambda P_{n-1}R_{n+1}},\\
R_nR_{n+1} &= R_nR_n - \alpha_n\lambda R_nP_n,\\
R_{n+1}R_{n+1} &= R_nR_{n+1} - \alpha_n\lambda\underline{P_nR_{n+1}},\\
R_nP_n &= R_nR_n + \beta_{n-1}P_{n-1}R_n,\\
P_nR_{n+1} &= R_nP_n - \alpha_n\underline{\lambda P_nP_n},\\
R_nP_{n+1} &= R_nR_{n+1} + \beta_nR_nP_n,\\
P_{n+1}P_{n+1} &= R_{n+1}P_{n+1} + \beta_n(P_nR_{n+1} + \beta_nP_nP_n).
\end{aligned}$$

To reduce the number of matrix-vector multiplications, the term underlined is expanded below.

$$\begin{aligned}
\lambda P_nR_{n+2} &= \lambda P_nR_{n+1} - \frac{\alpha_{n+1}}{\alpha_n}(\underline{\lambda R_nP_{n+1}} - \lambda R_{n+1}P_{n+1}),\\
\lambda P_nR_{n+1} &= \lambda R_nR_{n+1} + \beta_{n-1}\lambda P_{n-1}R_{n+1},\\
\lambda P_{n+1}P_{n+1} &= \lambda R_{n+1}P_{n+1} + \beta_n(\lambda P_nR_{n+1} + \beta_n\lambda P_nP_n).
\end{aligned}$$

Similarly, λR_nP_{n+1} is expanded as

$$\lambda R_nP_{n+1} = \lambda R_nR_{n+1} + \beta_n\lambda R_nP_n.$$

From the above, using the following notations:

$$\begin{array}{llll}
\boldsymbol{t}_n := R_nR_{n+1}\boldsymbol{r}_0, & \boldsymbol{y}_n := P_{n-1}R_{n+1}\boldsymbol{r}_0, & \boldsymbol{r}_n^{\mathrm{CGS}} := R_nR_n\boldsymbol{r}_0, & \boldsymbol{e}_n := R_nP_n\boldsymbol{r}_0,\\
\boldsymbol{h}_n := P_nR_{n+1}\boldsymbol{r}_0, & \boldsymbol{f}_n := R_nP_{n+1}\boldsymbol{r}_0, & \boldsymbol{p}_n := P_nP_n\boldsymbol{r}_0, & \boldsymbol{v}_n := \lambda P_{n-1}R_{n+1}\boldsymbol{r}_0,\\
\boldsymbol{s}_n := \lambda P_nR_{n+1}\boldsymbol{r}_0, & \boldsymbol{q}_n := \lambda P_nP_n\boldsymbol{r}_0, & \boldsymbol{w}_n := \lambda P_nR_{n+1}\boldsymbol{r}_0, & \boldsymbol{z}_n := \lambda R_nP_{n+1}\boldsymbol{r}_0,
\end{array}$$

then we can update the residual vector with only two matrix-vector multiplications. The preliminary algorithm is given as follows:

Algorithm 6.3: Preliminary SCGS2 method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \boldsymbol{z}_0 = \boldsymbol{x}_0,$
set $\boldsymbol{r}_0 = \boldsymbol{r}_0^* = \boldsymbol{e}_0 = \boldsymbol{r}_0^{\mathrm{CGS}},\ \boldsymbol{s}_{-1} = \boldsymbol{h}_{-1} = \boldsymbol{0},\ \beta_{-1} = 0,$

for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \leq \epsilon \|\boldsymbol{b}\|$ do:

$\boldsymbol{e}_n = \boldsymbol{r}_n^{\mathrm{CGS}} + \beta_{n-1}\boldsymbol{h}_{n-1}$,

$\boldsymbol{p}_n = \boldsymbol{e}_n + \beta_{n-1}(\boldsymbol{t}_{n-1} + \beta_{n-1}(\boldsymbol{e}_{n-1} + \boldsymbol{p}_{n-1}))$,

$\boldsymbol{q}_n = A\boldsymbol{e}_n + \beta_{n-1}(\boldsymbol{s}_{n-1} + \beta_{n-1}\boldsymbol{q}_{n-1})$,

compute α_n,

$\boldsymbol{t}_n = \boldsymbol{r}_n^{\mathrm{CGS}} - \alpha_n A\boldsymbol{e}_n$

$\boldsymbol{v}_n = \boldsymbol{h}_{n-1} - \dfrac{\alpha_n}{\alpha_{n-1}}(\boldsymbol{t}_{n-1} + \beta_{n-1}\boldsymbol{e}_{n-1} - \boldsymbol{e}_n)$,

$\boldsymbol{y}_n = \boldsymbol{s}_{n-1} - \dfrac{\alpha_n}{\alpha_{n-1}}(A\boldsymbol{t}_{n-1} + \beta_{n-1}A\boldsymbol{e}_{n-1} - A\boldsymbol{e}_n)$,

$\boldsymbol{s}_n = A\boldsymbol{t}_n + \beta_{n-1}\boldsymbol{y}_n$,

$\zeta_n,\ \eta_n = \arg\min \|\boldsymbol{r}_{n+1}\|$,

$\boldsymbol{x}_{n+1}^{\mathrm{CGS}} = \boldsymbol{x}_n^{\mathrm{CGS}} + \alpha_n\boldsymbol{e}_n + \alpha_n\boldsymbol{t}_n + \alpha_n\beta_{n-1}\boldsymbol{v}_n$,

$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{e}_n + \zeta_n\boldsymbol{t}_n + \eta_n\boldsymbol{v}_n$,

$\boldsymbol{r}_{n+1}^{\mathrm{CGS}} = \boldsymbol{t}_n - \alpha_n(A\boldsymbol{t}_n + \beta_{n-1}\boldsymbol{y}_n)$,

$\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \zeta_n A\boldsymbol{t}_n - \eta_n\boldsymbol{y}_n$,

compute β_n,

$\boldsymbol{h}_n = \boldsymbol{e}_n - \alpha_n\boldsymbol{q}_n$.

end

Similar to SCGS, there are many variants of computational formulas for α_n and β_n. Hence, we need to find optimal formulas by many numerical experiments for obtaining a numerically stable algorithm.

6.5 Numerical experiments

◇ Matrix market problems

In this section, we present the results of numerical experiments on a range of problems from Harwell-Boeing collection [26], NEP collection [3], SPARSKIT collection [67], and from Tim Davis's collection [21]. The data of all matrices were transformed into CRS format. We chose a random vector for the right-hand side of each linear system.

The matrices used in the experiments come from reservoir simulation (ORS* and SHERMAN*), chemical kinetics (FS5414), circuit simulation (MEMPLUS and ADD*), finite element modeling (CAVITY05 and FIDAP*), electrical engineering (DW2048), fluid dynamics (CDDE1), partial differential equations (PDE2961), and nuclear physics (UTM1700). The order N and number NNZ of nonzeros for each test problem are given in Table 6.3, together with the number of iterations, computational times, and $\log_{10}$ of true relative residual 2-norm for the unpreconditioned iterative methods. "†" denotes that a solver did not converge in 2000 iterations, and "$*$" denotes that preconditioning could not be implemented because of the breakdown.

All tests were performed on an ALPHA work station with a 750MHz processor using double precision arithmetic. Codes were written in standard Fortran 77 and compiled with the optimization option -$O4$. CPU time was given in seconds and it was measured by using the standard linux command *time*.

In all cases, the iteration was started with $\boldsymbol{x}_0 = \mathbf{0}$. The stopping criterion used was $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq 10^{-12}$, where $\boldsymbol{r}_n$ is the (unpreconditioned) residual vector.

Table 6.4 represents the results of experiments with the $ILU(0)$ preconditioner. Split preconditioning was used for all the experiments.

In terms of the number of iterations, the results support Theorem 6.4.1 without exceptions CAVITY05, ORSIRR1, ORSIRR2, and UTM1700A. The exceptions may be caused by a round-off error and highly ill-conditioned matrices. CDDE1 and DW2048, on the contrary, are suitable for SCGS from the fact that the number of iterations decreased by more than ten percent.

Looking at computational time in Table 6.3, CGS converged a little faster than SCGS, which implies that even if SCGS satisfies a stopping criterion in fewer steps than CGS, SCGS required more computational time because of the additional costs per iteration step. If a coefficient matrix has a large number of nonzeros, the cost of matrix-vector product is much expensive. Since the two methods have 2 matrix-vector product per iteration, they have almost same computational costs, and thus such tendency will disappear.

TRR shows how high the accuracy of an approximate solution is. CAVITY05, DW2048, FS5414 and ORSIRR1 give us some light into the property of SCGS. The true relative residual of SCGS was much better than that of CGS even though the computational time is almost of the same order.

In Table 6.4 we give iteration counts, computational time, and true relative residual 2-norm for the two iterative solvers preconditioned with $ILU(0)$. It appears from these results that the convergence behavior was similar to that for unpreconditioned systems. As regards iteration number, CGS was better than SCGS in the problems of FS5414 and MEMPLUS. The other problems were a little favorable for SCGS. However, clear differences did not occur since each of the preconditioned methods had high performance in all problems except for DW2048.

◇ The Helmholtz equation

We consider the following boundary value problem:

$$
\begin{array}{ll}
u_{xx} + u_{yy} + \sigma^2 u = 0, & (x, y) \in [0, \pi] \times [0, \pi], \\
u_x|_{x=0} = i\sqrt{\sigma^2 - \frac{1}{4}} \cos\frac{y}{2}, & \text{Neumann Condition}, \\
u_x - i\sqrt{\sigma^2 - \frac{1}{4}} u|_{x=\pi} = 0, & \text{Radiation Condition}, \\
u_y|_{y=0} = 0, & \text{Neumann Condition}, \\
u|_{y=\pi} = 0, & \text{Dirichlet Condition}.
\end{array}
$$

In Table 6.5, the number of iterations required for convergence, computing time and $\log_{10}$ of true relative residual 2-norm are shown for two fixed values $\sigma = 2.27$ and $\sigma = 4.16$, and for different grid size. The result is shown for CGS and SCGS with $ILU(0)$ preconditioning.

It appears from the results that SCGS was much more effective and robuster than CGS. From the point of view of the number of iterations, SCGS converged faster in all cases except for $M = 100$, $\sigma = 4.16$. Since computational costs of CGS per iteration are a little more expensive, SCGS makes up for the extra time by the convergence in fewer iteration steps. SCGS also improved the true relative residual. Notably, SCGS found much better approximate solutions in the problem of $M = 100$, $\sigma = 2.27$ while CGS caused stagnation. The two methods did not converge by 2000 iterations in the problem of $M = 100$, $\sigma = 4.16$.

Table 6.3. Test problems (N= order of matrix, NNZ= nonzeros in matrix) and results (Its = number of iterations, Time = computational time, TRR = $\log_{10}$ of the final true relative residual 2-norm) for the iterative methods without preconditioning.

			Its		Time		TRR	
Matrix	N	NNZ	CGS	SCGS	CGS	SCGS	CGS	SCGS
ADD20	2395	17319	399	380	0.61	0.61	-12.01	-12.07
ADD32	4960	23884	73	70	0.27	0.27	-12.34	-12.19
CAVITY05	1182	32747	722	723	1.07	1.10	-8.69	-10.99
CDDE1	961	4681	146	125	0.11	0.10	-11.92	-11.91
DW2048	2048	10114	2315	2076	3.31	3.09	-6.00	-8.58
FIDAP020	2203	69579	†	†	†	†	†	†
FIDAP037	3565	67591	71	68	0.41	0.43	-12.10	-12.44
FS5414	541	4285	1339	1334	0.44	0.46	-6.24	-8.55
MENPLUS	17758	126150	1397	1332	17.70	17.93	-8.24	-8.93
ORSIRR1	1030	6858	1103	1129	1.00	1.05	-6.80	-9.54
ORSIRR2	886	5970	789	807	0.61	0.63	-7.65	-8.00
ORSREG1	2205	14133	344	337	0.73	0.75	-12.07	-12.21
PDE2961	2961	14585	191	190	0.45	0.45	-10.88	-11.49
SHERMAN1	1000	3750	537	513	0.28	0.28	-11.09	-11.50
SHERMAN3	5005	20033	†	†	†	†	†	†
SHERMAN4	1104	3786	131	127	0.08	0.08	-12.39	-12.40
SHERMAN5	3312	20793	2169	2167	3.43	3.73	-8.13	-7.77
UTM1700A	1700	21313	1673	1706	2.39	2.62	-6.53	-6.85

Table 6.4. Test problems and results for the iterative methods with $ILU(0)$.

			Its		Time		TRR	
Matrix	N	NNZ	CGS	SCGS	CGS	SCGS	CGS	SCGS
ADD20	2395	17319	173	169	0.49	0.50	-12.06	-12.15
ADD32	4960	23884	34	34	0.25	0.25	-12.09	-12.20
CAVITY05	1182	32747	*	*	*	*	*	*
CDDE1	961	4681	37	37	0.05	0.05	-12.57	-12.60
DW2048	2048	10114	†	†	†	†	†	†
FIDAP020	2203	69579	153	154	1.07	1.10	-10.78	-9.49
FIDAP037	3565	67591	9	9	0.28	0.29	-12.81	-12.81
FS5414	541	4285	725	844	0.40	0.48	-6.07	-6.20
MENPLUS	17758	126150	329	323	7.89	8.27	-10.94	-11.96
ORSIRR1	1030	6858	46	43	0.08	0.08	-12.71	-12.37
ORSIRR2	886	5970	45	45	0.07	0.07	-12.29	-12.42
ORSREG1	2205	14133	55	51	0.22	0.20	-12.51	-12.45
PDE2961	2961	14585	43	43	0.18	0.19	-12.84	-12.92
SHERMAN1	1000	3750	44	43	0.05	0.05	-12.10	-12.65
SHERMAN3	5005	20033	93	92	0.49	0.51	-12.38	-12.23
SHERMAN4	1104	3786	32	32	0.04	0.04	-12.47	-12.79
SHERMAN5	3312	20793	57	53	0.24	0.23	-10.11	-10.00
UTM1700A	1700	21313	133	135	0.40	0.43	-8.86	-8.90

Let us show the convergence behaviors in Figs 6.1 - 6.4. From Fig. 6.1 we observe that SCGS smoothed the convergence behavior of CGS and the 2-norms of relative residuals at each step were smaller than those of CGS. As a result, SCGS converged faster than CGS. In Fig. 6.2, the 2-norms of the relative residuals of SCGS were larger around at the step 170. Though round-off errors had a great influence on SCGS, SCGS converged faster than CGS. The most remarkable thing is that the convergence behavior was completely different after the iterative step about 100 in Fig. 6.3. From Fig. 6.4, the behavior of SCGS seems to be still better though the two methods did not converge at step 2000.

Table 6.5. Results for the elliptic problems (M=grid size, N= order of matrix) and convergence results (Its = number of iterations, Time = computational time, TRR = $\log_{10}$ of the final true relative residual 2-norm) for the iterative methods with $ILU(0)$.

			Its		Time		TRR	
M	N	σ	CGS	SCGS	CGS	SCGS	CGS	SCGS
50	2550	2.27	128	115	0.12	0.12	-9.97	-9.99
50	2550	4.16	222	179	0.20	0.18	-11.5	-11.5
100	10100	2.27	†	450	†	2.01	†	-10.54
100	10100	4.16	†	†	†	†	†	†

6.6 Concluding remarks

In this chapter, we have developed the algorithm of CGS for stabilizing the irregular convergence behavior. In terms of theory, while CGS uses the Bi-CG polynomial squared, we devised a new polynomial with one free parameter which includes the Bi-CG polynomial and defined a new residual vector by the product of the new polynomial and the Bi-CG polynomial. By choosing the parameter such that the residual 2-norm is locally minimized, we could obtain an algorithm called SCGS. We proved that in exact precision arithmetic SCGS always converges faster than CGS in terms of the number of iterations.

We have learned from various numerical experiments that SCGS often gives smoother convergence behavior and gives more reliable solutions than CGS. Hence, we can say that SCGS may be an alternative solver for the fields where CGS is still the method of choice.

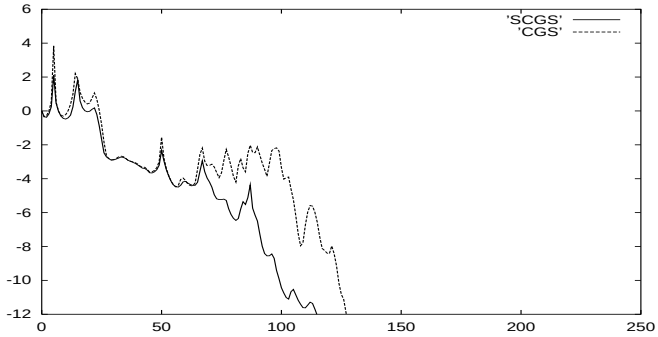

Figure 6.1: Residual 2-norm histories for $M = 50$, $\sigma = 2.27$.

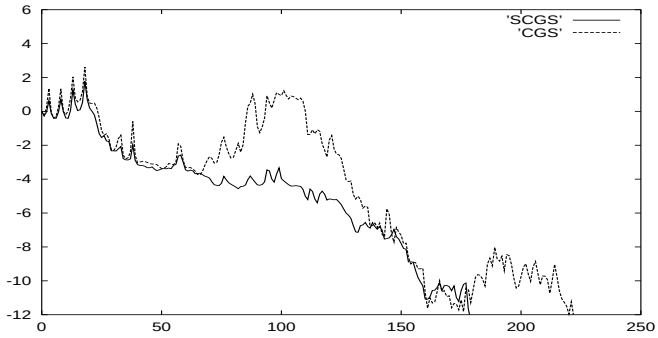

Figure 6.2: Residual 2-norm histories for $M = 50$, $\sigma = 4.16$.

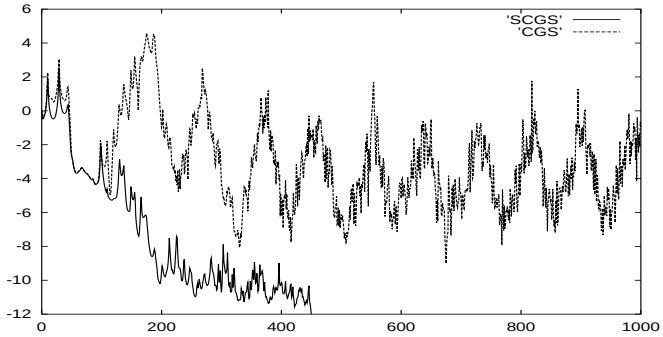

Figure 6.3: Residual 2-norm histories for $M = 100$, $\sigma = 2.27$.

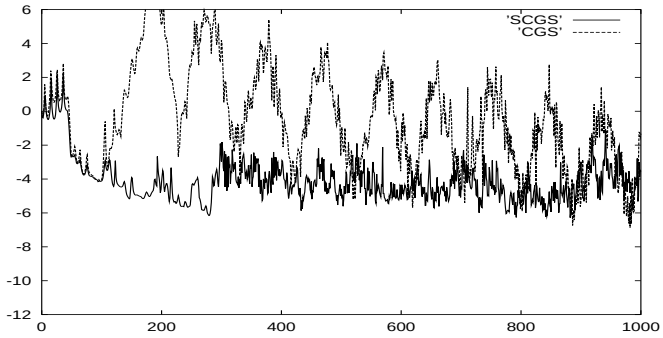

Figure 6.4: Residual 2-norm histories for $M = 100$, $\sigma = 4.16$.

Chapter 7

Conclusion

In this thesis, we focused on the A-orthogonalization property of the conjugate residual (CR) method for solving Hermitian (indefinite) linear systems. Based on the property of CR, we extended CR to general non-Hermitian linear systems and obtained a bi-conjugate residual (Bi-CR) method. From the results of numerical experiments, Bi-CR can be an attractive basic solver, i.e. various developments from Bi-CR can be considered. As a special case of Bi-CR, we derived a conjugate orthogonal conjugate residual (COCR) method for solving complex symmetric linear systems. Based on the algorithm of Bi-CR, we derived a conjugate residual squared (CRS) method as a result of one of the product-type methods. On the other hand, we improved the performance of the conjugate gradient squared (CGS) method that is one of product-type methods based on Bi-CG.

In chapter 3, based on the property of CR, we obtained the algorithm of Bi-CR whose residual vectors satisfy $\boldsymbol{r}_n \perp A^{\mathrm{H}} K_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$. We observed from the algorithm of Bi-CR that Bi-CR reduces to CR if the coefficient matrix is Hermitian and that Bi-CG and Bi-CR require almost the same computational costs per iteration. We have learned from many numerical experiments that Bi-CR often gives smoother convergence behavior and converges faster than Bi-CG. Since this smoother behavior is similar to CR, Bi-CR may have inherited something from the property of CR. Therefore, the analysis of this convergence behavior needs to be done for future work.

In chapter 4, we extended CR to complex symmetric linear systems. The main idea was to consider a procedure which generates residual vectors satisfying $\boldsymbol{r}_n \perp \overline{AK_n(A, \boldsymbol{r}_0)}$. As a result of the extension, we obtained an algorithm of COCR. We showed that Bi-CR reduces to COCR if the coefficient matrix is complex symmetric and that COCR reduces to CR if the coefficient matrix is real symmetric. From the algorithm, we have learned that COCR has almost the same computational costs per iteration as COCG and QMR_SYM which are regarded as successful Krylov subspace methods for complex symmetric linear systems. We have learned from some numerical experiments that COCR as well as QMR_SYM often shows smoother convergence behavior than COCG. Hence, COCR could be a competitive solver for this class of matrices.

In chapter 5, we obtained CRS as one of product-type methods based on Bi-CR. This framework came from an analogy of product-type methods based on Bi-CG which were first introduced by S.-L. Zhang (1997). CGS and CRS have almost the same computational costs per iteration. We have learned that CRS often gives smoother convergence behavior than CGS and often generates more accurate solutions than CGS. In the examples of Helmholtz equations, when CGS worked well, CRS converged faster than the other well-known methods such as CGS, Bi-CGSTAB, and GPBi-CG. Hence, CRS may become a

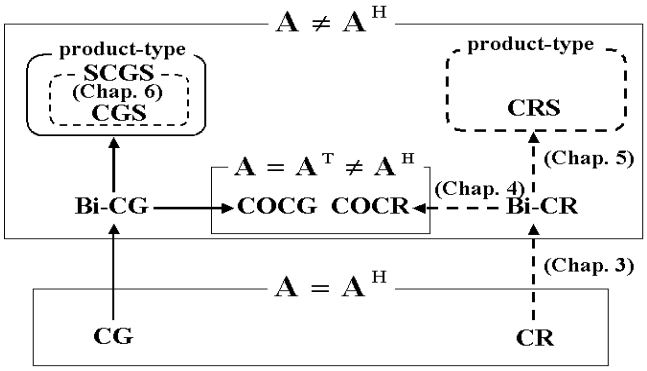

Figure 7.1: Research flow chart.

useful and reliable tool for solving non-Hermitian linear systems especially in the case where CGS works well. However, there are some examples where CRS did not converge, this will motivate us to look for more appropriate matrix polynomials for future work.

In chapter 6, we proposed a stabilized conjugate gradient (SCGS) method to improve the convergence behavior of CGS. The idea was based on the introduction of a polynomial which includes one of the Bi-CG polynomials used in CGS. We proved that in exact precision arithmetic the nth residual 2-norm of SCGS is always less than that of CGS. From the numerical experiments, we have learned that SCGS produced much more reliable solutions. Since the convergence of SCGS theoretically depends strongly on that of CGS, SCGS is preferred to use for the problems where CGS works well.

The research flow chart is shown in Fig. 7.1. We can see from Fig. 7.1 that the present thesis concerns systematic research on the conjugate residual method and that the figure directly gives us future work.

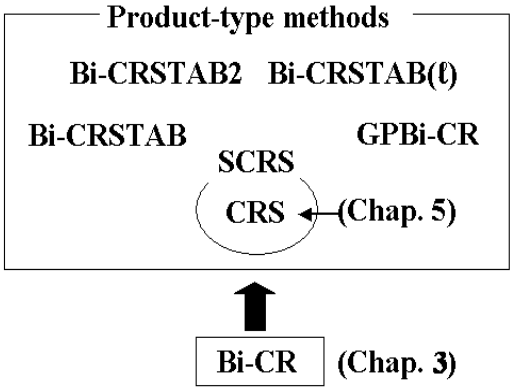

Figure 7.2: Future work.

Finally, we describe the remaining topics that will be future work.

(1) We can apply the idea of SCGS to CRS. Choosing a suitable polynomial will enable

us to obtain SCRS that always converges faster than CRS in terms of the number of iteration steps in exact precision arithmetic.

(2) From the framework of product-type methods based on Bi-CR, we will derive Bi-CRSTAB, Bi-CRSTAB2, Bi-CRSTAB(ℓ), and GPBi-CR. Moreover, we will find a suitable polynomial for the product-type methods since well-known polynomials for product-type methods based on Bi-CG may not be always useful for those based on Bi-CR.

From (2), we can also consider one of the product-type methods by using the Bi-CG and Bi-CR polynomial. The above future work is shown in Fig. 7.2.

On the other hand, QMR-like approach may be a good approach. Since it is known that QMR is closely related to Bi-CG, QMR-like approach may be also closely related to Bi-CR. If this QMR-like method and Bi-CR have close relationship, then a QMR-like method will be derived for non-Hermitian linear systems, and for complex symmetric linear systems. Moreover, product-type methods such as QMRCGSTAB-like approach will be also available for solving non-Hermitian linear systems.

Appendix A

Data structures

Devising data structure gives us lots of benefits in saving memory and computational time. For example, if the coefficient matrix is large and sparse, it is reasonable to consider schemes to retain only non-zero elements. There are many methods for storing data (see, *e.g.*, Saad [67] and Eijkhout [29]) such as Compressed Row and Column Storage (CRS and CCS), Block Compressed Row Storage, Compressed Diagonal Storage (CDS), Jagged Diagonal Storage, and Skyline Storage. In this chapter, we will discuss CRS and CDS that were used in our numerical experiments. For details on other data structures, see, e.g., Templates [5].

A.1 Compressed row storage (CRS)

The compressed row storage makes no assumptions about the sparsity structure of the matrix, and it doesn't store nonzero elements. On the other hand, it requires an indirect addressing step for every operation with regard to the coefficient matrix. We show an example of CRS format. Let us consider the nonsymmetric matrix defined by

$$A = \begin{pmatrix} 11 & 0 & 13 & 14 & 0 \\ 21 & 22 & 0 & 0 & 0 \\ 31 & 0 & 33 & 0 & 35 \\ 0 & 42 & 0 & 44 & 45 \\ 51 & 52 & 0 & 0 & 55 \end{pmatrix}.$$

Then, the CRS format for this matrix is given by the following array {IA,JA,A}:

IA	1	4	6	9	12	15								
JA	1	3	4	1	2	1	3	5	2	4	5	1	2	5
A	11	13	14	21	22	31	33	35	42	44	45	51	52	55

.

IA is a row pointer vector that stores the starting location of each rows. The JA vector stores the column indices of the elements in the matrix A, and A stores the values of the nonzero elements of the matrix A. This approach needs only 2NNZ+n+1 instead of storing n^2 elements, where NNZ is the number of nonzeros in the matrix A. If the matrix A is symmetric, we need to store only an upper (or lower) triangular matrix.

◇ Matrix-vector product for CRS

The matrix vector product $\boldsymbol{y} = A\boldsymbol{x}$ using CRS format can be expressed in the usual way:

$$y_i = \sum_j a_{i,j} x_j.$$

From the above, the matrix-vector multiplication is given by

```
for i = 1, n
  y(i) = 0,
  for j = IA(i), IA(i + 1) − 1
    y(i) = y(i) + A(j) ∗ x(JA(j)).
  end
end
```

We see from the above algorithm that the operation count is 2 times the number of nonzero elements in A, which is a significant saving over the dense operation requirement of $2n^2$.

◇ $ILU(0)$ preconditioner for CRS

Here, we consider $ILU(0)$ using the CRS format. Assuming that a graph of the matrix A contains no triangles. It follows from the assumption that $ILU(0)$ has the form $A \approx K = (D + L_A)D^{-1}(D + U_A)$, where L_A and U_A are a strictly lower (upper) triangular part of A. Hence, we only need to store a diagonal matrix D containing the pivots of the factorization. In fact, we will store the inverses of the pivots rather than the pivots themselves. This implies that during the system solution no divisions have to be performed.

The factorization begins with copying the matrix diagonal

```
for i = 1, n
  d(i) = A(d(i)),
end
```

where we introduced an array of diagonal pointer, *i.e*, A(d(i))=$a_{i,i}$. Each elimination step starts by inverting the pivot

```
for i = 1, n
d(i) = 1/d(i).
```

For all nonzero elements $a_{i,j}$ with $j > i$, we next check whether $a_{j,i}$ is a nonzero element.

```
 for j =d(i) + 1, IA(i + 1) − 1
   found=FALSE,
   for k =IA(JA(j)), d(JA(j) − 1)
     if (JA(k)=i) then
       found=TRUE,
       element=A(k),
     end if
   end
   {If so, we update a_{j,j}}
   If(found=TRUE) then
     d(JA(j))=d(JA(j))-element∗d(i)∗A(j).
   end if
 end
end
```

In preconditioned iterative methods we need to solve the system $K\boldsymbol{y} = \boldsymbol{x}$. If the preconditioner is factorized as $K = LU$, and then the system $LU\boldsymbol{y} = \boldsymbol{x}$ can be solved in the usual manner by introducing a temporary vector $\boldsymbol{z}$:

$$L\boldsymbol{z} = \boldsymbol{x}, \quad U\boldsymbol{y} = \boldsymbol{z}.$$

It follows from the incomplete factorization of the form

$$K = (D + L_A)D^{-1}(D + U_A) = (D + L_A)(I + D^{-1}U_A).$$

that the previous systems can be solved by

$$(D + L_A)\boldsymbol{z} = \boldsymbol{x}, \quad (I + D^{-1}U_A)\boldsymbol{y} = \boldsymbol{z}.$$

We show a CRS-based precondition solve below. The former part of the following code is for obtaining $\boldsymbol{z}$ and the latter for $\boldsymbol{y}$.

```
for i = 1, n
  sum=0,
  for j =IA(i), d(i)-1
    sum = sum + A(j) * z(JA(j)),
  end
  z(i)=d(i) * (x(i)-sum),
end
for i = 1, n, (step -1)
  sum=0,
  for j =d(i)+1, IA(i + 1)-1
    sum = sum + A(j) * y(JA(j)),
  end
  y(i)=z(i)−d(i)∗sum.
end
```

A.2 Compressed diagonal storage (CDS)

In this section, we describe the CDS format which is often used in the case where the matrix A is banded. To explain the CDS format we give a simple example. Let A be of the form

$$A = \begin{pmatrix} 11 & 12 & 0 & 0 & 15 & 0 & 0 \\ 21 & 22 & 23 & 0 & 0 & 26 & 0 \\ 0 & 32 & 33 & 34 & 0 & 0 & 37 \\ 0 & 0 & 43 & 44 & 45 & 0 & 0 \\ 51 & 0 & 0 & 54 & 55 & 56 & 0 \\ 0 & 62 & 0 & 0 & 65 & 66 & 67 \\ 0 & 0 & 73 & 0 & 0 & 76 & 77 \end{pmatrix}.$$

Then, we need to prepare five vectors to store diagonal elements of the coefficient matrix.

SL2	0	0	0	0	51	62	73
SL1	0	21	32	43	54	65	76
SDA	11	22	33	44	55	66	77
SU1	12	23	34	45	56	67	0
SU2	15	26	37	0	0	0	0

,

where SL1 and SL2 store the lower part of the coefficient matrix A, SDA the diagonal part, and SU1 and SU2 the upper part.

◇ Matrix-vector product for CDS

The coefficient matrix $A \in \mathcal{C}^{n\times n}$ given by a five-point central finite difference discretization of second order PDEs is generally written as follows:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & & & a_{1,m} & & \\ a_{2,1} & \ddots & \ddots & & & \ddots & \\ & \ddots & \ddots & \ddots & & & a_{n-m+1,n} \\ a_{m,1} & & \ddots & \ddots & \ddots & & \\ & \ddots & & \ddots & \ddots & a_{n-1,n} & \\ & & a_{n,n-m+1} & & a_{n,n-1} & a_{n,n} & \end{pmatrix}.$$

Since CDS format does not need list vectors, which means there is no indirect addressing, during matrix-vector product, it is quite useful to decrease computing time per iteration. Here, we consider the matrix vector product and $ILU(0)$ for CRS format. We first show the sample program of a matrix-vector product for CDS below.

for $i = 1,\ n$
  $y(i) = \mathrm{SL2}(i) * x(i-m) + \mathrm{SL1}(i) * x(i-1) + \mathrm{SDA}(i) * x(i)$
    $+\ \mathrm{SU1}(i) * x(i+1) + \mathrm{SU2}(i) * x(i+m)$
end

The costs for the product reach $\mathcal{O}(8n)$, which is almost of the same order as five inner products. Another advantage of this algorithm is that it is vectorizable with vector length of essentially the matrix order n. Because of the regular access, most machines can perform this algorithm efficiently by keeping five base registers and using simple offset addressing.

◇ $ILU(0)$ preconditioner for CDS

Next, we show the code of $ILU(0)$ preconditioning for CDS format. Matrices arising from finite difference discretization regularly have a zero-free diagonal and furthermore these elements are not even nearly equal to zeros. This implies that it is usually possible to carry out an incomplete factorization without any difficulties. First, we show the code for storing the inverses of the pivots:

$\mathrm{PDA}(1) = 1/\mathrm{SDA}(1)$,
for $i = 2,\ m$
  $\mathrm{PDA}(i) = 1/(\mathrm{SDA}(i) - \mathrm{SU1}(i-1) * \mathrm{SL1}(i) * \mathrm{PDA}(i-1))$,
end

for $i = m+1,\ n$
$\quad \mathrm{PDA}(i) = 1/(\mathrm{SDA}(i) - \mathrm{SU1}(i-1) * \mathrm{SL1}(i) * \mathrm{PDA}(i-1)$
$\qquad\qquad - \mathrm{SU2}(i-m) * \mathrm{SL2}(i) * \mathrm{PDA}(i-m)).$
end

For solving the system $(D+L)D^{-1}(D+U)\boldsymbol{y} = \boldsymbol{x}$ we consider the form $(D+L)\boldsymbol{z} = \boldsymbol{x}$ for forward substitution and the form $(I + D^{-1}U)\boldsymbol{y} = \boldsymbol{z}$ for backward substitution:

for $i = 1,\ n$
$\quad y(i) = (x(i) - \mathrm{SL1}(i) * y(i-1) - \mathrm{SL2}(i) * y(i-m)) * \mathrm{PDA}(i),$
end
for $i = n-1,\ i,$ (step -1)
$\quad y(i) = y(i) - \mathrm{PDA(i)}^*(\mathrm{SU1}(i) * y(i+1) + \mathrm{SU2}(i) * y(i+m)).$
end

The above code is used for the 5-point central difference discretization of partial differential equations with a rectangular mesh.

Appendix B

Derivations of successful Krylov subspace methods

B.1 KS methods for Hermitian linear systems

◇ The CG method

We give details of the derivation for CG. Substituting (2.19) for $\tilde{\boldsymbol{v}}_n$ of Algorithm 2.7, the residual vector $\boldsymbol{r}_n$ can be computed by the following three-term recurrences:

$$\begin{aligned} \boldsymbol{r}_1 &= \alpha_0 \frac{(\boldsymbol{r}_0, A\boldsymbol{r}_0)}{(\boldsymbol{r}_0, \boldsymbol{r}_0)} \boldsymbol{r}_0 - \alpha_0 A\boldsymbol{r}_0, \\ \boldsymbol{r}_n &= \alpha_{n-1} \Big\{ \frac{(\boldsymbol{r}_{n-1}, A\boldsymbol{r}_{n-1})}{(\boldsymbol{r}_{n-1}, \boldsymbol{r}_{n-1})} \boldsymbol{r}_{n-1} \\ &\quad + \frac{(\boldsymbol{r}_{n-2}, A\boldsymbol{r}_{n-1})}{(\boldsymbol{r}_{n-2}, \boldsymbol{r}_{n-2})} \boldsymbol{r}_{n-2} - A\boldsymbol{r}_{n-1} \Big\}, \end{aligned} \tag{B.1}$$

where $\alpha_{n-1} = -(\boldsymbol{y}_n, \boldsymbol{e}_n)/h_{n,n-1}(\boldsymbol{y}_{n-1}, \boldsymbol{e}_{n-1})$. Since the inverse matrix $H_n^{-1} \in \mathcal{R}^{n \times n}$, it follows that the parameter $\alpha_{n-1} \in \mathcal{R}$. From the inner product of $\boldsymbol{r}_n$ and (B.1), we obtain

$$\alpha_{n-1} = -\frac{(\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{r}_n, A\boldsymbol{r}_{n-1})} = -\frac{(\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{r}_{n-1}, A\boldsymbol{r}_n)}.$$

Substituting (2.18) into (B.1), we have

$$\boldsymbol{r}_n = \alpha_{n-1} \Big\{ \frac{(\boldsymbol{r}_{n-1}, A\boldsymbol{r}_{n-1})}{(\boldsymbol{r}_{n-1}, \boldsymbol{r}_{n-1})} + \frac{(\boldsymbol{r}_{n-2}, A\boldsymbol{r}_{n-1})}{(\boldsymbol{r}_{n-2}, \boldsymbol{r}_{n-2})} \Big\} \boldsymbol{r}_0 - AV_n \hat{\boldsymbol{y}}_n, \tag{B.2}$$

where $\hat{\boldsymbol{y}}_n \in \mathcal{C}^n$. Comparing the coefficients of $\boldsymbol{r}_0$ in (2.18) and (B.2), we have the following recurrences:

$$\alpha_0 = \frac{(\boldsymbol{r}_0, \boldsymbol{r}_0)}{(\boldsymbol{r}_0, A\boldsymbol{r}_0)}, \tag{B.3}$$

$$\begin{aligned} \alpha_n &= \frac{1}{\frac{(\boldsymbol{r}_n, A\boldsymbol{r}_n)}{(\boldsymbol{r}_n, \boldsymbol{r}_n)} + \frac{(\boldsymbol{r}_{n-1}, A\boldsymbol{r}_n)}{(\boldsymbol{r}_{n-1}, \boldsymbol{r}_{n-1})}} \\ &= \frac{1}{\frac{(\boldsymbol{r}_n, A\boldsymbol{r}_n)}{(\boldsymbol{r}_n, \boldsymbol{r}_n)} - \frac{1}{\alpha_{n-1}} \frac{(\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{r}_{n-1}, \boldsymbol{r}_{n-1})}}. \end{aligned} \tag{B.4}$$

Here, let us introduce an auxiliary vector $\boldsymbol{p}_{n-1}$ as

$$\boldsymbol{p}_{n-1} = \frac{\boldsymbol{z}_n - \boldsymbol{z}_{n-1}}{\alpha_{n-1}} \quad \in K_n(A, \boldsymbol{r}_0). \tag{B.5}$$

Then, it follows from (2.18) that the residual vector $\boldsymbol{r}_n$ can be described by using $\boldsymbol{r}_{n-1}$ and $\boldsymbol{p}_{n-1}$.

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - A\boldsymbol{z}_n = \boldsymbol{r}_0 - A\boldsymbol{z}_{n-1} - \alpha_{n-1}A\boldsymbol{p}_{n-1} = \boldsymbol{r}_{n-1} - \alpha_{n-1}A\boldsymbol{p}_{n-1}. \tag{B.6}$$

Substituting the above formula for (2.18) and using (B.4), it follows that

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}, \tag{B.7}$$

where $\beta_{n-1} = (\boldsymbol{r}_n, \boldsymbol{r}_n)/(\boldsymbol{r}_{n-1}, \boldsymbol{r}_{n-1})$. On the other hand, α_{n-1} can be determined in the following way: $\boldsymbol{r}_n$ is orthogonal to $\boldsymbol{p}_{n-1}$ by Ritz-Galerkin condition (2.3) and (B.5). From (B.6), α_{n-1} can be written as

$$\alpha_{n-1} = \frac{(\boldsymbol{p}_{n-1}, \boldsymbol{r}_{n-1})}{(\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})}.$$

Since it follows from (B.7) that $(\boldsymbol{p}_n, \boldsymbol{r}_n) = (\boldsymbol{r}_n, \boldsymbol{r}_n)$, we have

$$\alpha_{n-1} = \frac{(\boldsymbol{r}_{n-1}, \boldsymbol{r}_{n-1})}{(\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})}.$$

The three-term recurrence relation is obtained by using α_{n-1} and β_{n-1}.

$$\begin{aligned}
\boldsymbol{r}_1 &= \boldsymbol{r}_0 - \alpha_0 A\boldsymbol{r}_0, \\
\boldsymbol{r}_n &= (1 + \frac{\beta_{n-2}}{\alpha_{n-2}}\alpha_{n-1})\boldsymbol{r}_{n-1} - \alpha_{n-1}A\boldsymbol{r}_{n-1} \\
&\quad -\frac{\beta_{n-2}}{\alpha_{n-2}}\alpha_{n-1}\boldsymbol{r}_{n-2}, \quad n = 2, 3, \ldots.
\end{aligned}$$

From (B.7), the approximate solution $\boldsymbol{x}_n$ can be calculated as follows:

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + \boldsymbol{z}_n = \boldsymbol{x}_0 + \boldsymbol{z}_{n-1} + \alpha_{n-1}\boldsymbol{p}_{n-1} = \boldsymbol{x}_{n-1} + \alpha_{n-1}\boldsymbol{p}_{n-1}.$$

From the above we obtain the CG algorithm.

◇ The MINRES method

The following problem can be efficiently solved by Givens rotations.

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y} \in \mathcal{C}^n} \|\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\|$$

For simplicity, here we consider the case $n = 4$. In matrix form $\beta\boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}$ is written as

$$\beta\boldsymbol{e}_1 - T_{5,4}\boldsymbol{y} = \begin{pmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} t_{11} & t_{12} & & \\ t_{21} & t_{22} & t_{23} & \\ & t_{32} & t_{33} & t_{34} \\ & & t_{43} & t_{44} \\ & & & t_{54} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix},$$

where $t_{ij} = t_{ji},\ 1 \le i, j \le 4$. Let G_1 be a matrix of the Givens rotation, *i.e.*,

$$G_1 = \begin{pmatrix} c_1 & s_1 & & & \\ -\bar{s}_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} \quad \text{with} \quad c_1 = \frac{|t_{11}|}{\sqrt{|t_{11}|^2 + |t_{21}|^2}}, \quad \bar{s}_1 = \frac{t_{21}}{t_{11}} c_1.$$

Then, G_1 is a unitary matrix, and thus $\|\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\| = \|G_1(\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y})\|$. This leads to

$$G_1(\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}) = \begin{pmatrix} g_1^{(1)} \\ g_2^{(1)} \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} t_{11}^{(1)} & t_{12}^{(1)} & t_{13}^{(1)} & & \\ & t_{22}^{(1)} & t_{23}^{(1)} & & \\ & t_{32} & t_{33} & t_{34} & \\ & & t_{43} & t_{44} & \\ & & & t_{54} & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix},$$

where $g_2^{(1)} = -\bar{s}_1 \beta$. Next, let G_2 be a unitary matrix defined by

$$G_2 = \begin{pmatrix} 1 & & & & \\ & c_2 & s_2 & & \\ & -\bar{s}_2 & c_2 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} \quad \text{with} \quad c_2 = \frac{|t_{22}^{(1)}|}{\sqrt{|t_{22}^{(1)}|^2 + |t_{32}|^2}}, \quad \bar{s}_2 = \frac{t_{32}}{t_{22}^{(1)}} c_2.$$

Then, G_2 and $G_2 G_1$ are unitary matrices, and thus $\|\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\| = \|G_2 G_1(\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y})\|$. This leads to

$$G_2 G_1(\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}) = \begin{pmatrix} g_1^{(1)} \\ g_2^{(2)} \\ g_3^{(2)} \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} t_{11}^{(1)} & t_{12}^{(1)} & t_{13}^{(1)} & & \\ & t_{22}^{(2)} & t_{23}^{(2)} & t_{24}^{(2)} & \\ & & t_{33}^{(2)} & t_{34}^{(2)} & \\ & & t_{43} & t_{44} & \\ & & & t_{54} & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix},$$

where $g_3^{(2)} = (-1)^2 \bar{s}_2 \bar{s}_1 \beta$. Similarly, using G_3 and G_4, we finally obtain

$$\begin{aligned} Q_4(\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}) &= \begin{pmatrix} g_1^{(1)} \\ g_2^{(2)} \\ g_3^{(3)} \\ g_4^{(4)} \\ g_5^{(5)} \end{pmatrix} - \begin{pmatrix} t_{11}^{(1)} & t_{12}^{(1)} & t_{13}^{(1)} & & \\ & t_{22}^{(2)} & t_{23}^{(2)} & t_{24}^{(2)} & \\ & & t_{33}^{(3)} & t_{34}^{(3)} & \\ & & & t_{44}^{(4)} & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{g}_4 \\ g_5^{(5)} \end{pmatrix} - \begin{pmatrix} R_4 \\ 0 \end{pmatrix} \boldsymbol{y}_4, \end{aligned}$$

where $Q_4 = G_4G_3G_2G_1$ and $g_5^{(5)} = (-1)^4\bar{s}_4\bar{s}_3\bar{s}_2\bar{s}_1\beta$. Hence, we obtain $\boldsymbol{y}$ by solving the following equation:

$$R_4\boldsymbol{y}_4 = \boldsymbol{g}_4 \tag{B.8}$$

using backward substitution. From the above form, we can know the value of $\min_{\boldsymbol{y}} \|\beta\boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}\|$ without solving the equation $R_4\boldsymbol{y} = \boldsymbol{g}_4$ since the value satisfies

$$\min_{\boldsymbol{y}_4\in\mathcal{C}^4} \|\beta\boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}_4\| = \min_{\boldsymbol{y}_4\in\mathcal{C}^4} \|Q_4(\beta\boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}_4)\| = |g_5^{(5)}|.$$

Thus, $|g_5^{(5)}|$ can be used as the stopping criterion. To obtain the approximate solution, it follows from (2.31) and (B.8) that

$$\boldsymbol{x}_4 = \boldsymbol{x}_0 + V_4R_4^{-1}\boldsymbol{g}_4.$$

Here, we introduce a matrix $P_4 := V_4R_4^{-1}$ with columns $[\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4]$. Then, we have the following recurrences:

$$\begin{aligned}
\boldsymbol{p}_1 &= \boldsymbol{v}_1/t_{11}^{(1)},\\
\boldsymbol{p}_2 &= (\boldsymbol{v}_2 - t_{12}^{(1)}\boldsymbol{p}_1)/t_{22}^{(2)},\\
\boldsymbol{p}_3 &= (\boldsymbol{v}_3 - t_{13}^{(1)}\boldsymbol{p}_1 - t_{23}^{(2)}\boldsymbol{p}_2)/t_{33}^{(3)},\\
\boldsymbol{p}_4 &= (\boldsymbol{v}_4 - t_{24}^{(2)}\boldsymbol{p}_2 - t_{34}^{(3)}\boldsymbol{p}_3)/t_{44}^{(4)}.
\end{aligned}$$

From the above recurrences, it follows that

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + g_i^{(i)}\boldsymbol{p}_i, \quad i = 1, \ldots, 4.$$

The above computations can be easily generalized, and then Algorithm 2.12 is obtained.

B.2 KS methods for complex symmetric linear systems

◇ The COCG method

We show a process for obtaining α_{n-1} and β_{n-1} using the recurrences (2.32)-(2.34) and the orthogonality conditions (2.35). For determining α_{n-1}, it follows from (2.33) that the inner product of $\bar{A}^n\bar{\boldsymbol{r}}_0$ and $\boldsymbol{r}_n$ is computed as

$$(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, \boldsymbol{r}_n) = (\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, \boldsymbol{r}_{n-1}) - \alpha_{n-1}(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{p}_{n-1}).$$

Since $\bar{A}^{n-1}\bar{\boldsymbol{r}}_0$ belongs to $K_n(\bar{A}, \bar{\boldsymbol{r}}_0)$, it follows that $(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, \boldsymbol{r}_n) = 0$ by the conditions (2.35). Hence, we obtain

$$\alpha_{n-1} = \frac{(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, \boldsymbol{r}_{n-1})}{(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{p}_{n-1})}. \tag{B.9}$$

Next, for determining β_{n-1} it follows from (2.34) that the inner product of $\bar{A}^{n-1}\bar{\boldsymbol{r}}_0$ and $A\boldsymbol{p}_n$ is computed as

$$(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{p}_n) = (\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{r}_n) + \beta_{n-1}(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{p}_{n-1}).$$

From the conditions (2.35), we have $(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{p}_n) = 0$. Thus we obtain

$$\beta_{n-1} = -\frac{(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{r}_n)}{(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{p}_{n-1})} = -\alpha_{n-1}\frac{(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, A\boldsymbol{r}_n)}{(\bar{A}^{n-1}\bar{\boldsymbol{r}}_0, \boldsymbol{r}_{n-1})}. \tag{B.10}$$

Here, let us consider deriving practical formulas for α_{n-1} and β_{n-1} from (B.9) and (B.10). Note that from the recurrences (2.32)-(2.34) two vectors $\bar{\boldsymbol{r}}_{n-1}$ and $\bar{\boldsymbol{p}}_{n-1}$ can be written as

$$\bar{\boldsymbol{r}}_{n-1} = \bar{c}_{n-1}\bar{A}^{n-1}\bar{\boldsymbol{r}}_0 + \bar{\boldsymbol{z}}_1, \quad \bar{\boldsymbol{z}}_1 \in K_{n-1}(\bar{A}, \bar{\boldsymbol{r}}_0), \tag{B.11}$$

$$\bar{\boldsymbol{p}}_{n-1} = \bar{c}_{n-1}\bar{A}^{n-1}\bar{\boldsymbol{r}}_0 + \bar{\boldsymbol{z}}_2, \quad \bar{\boldsymbol{z}}_2 \in K_{n-1}(\bar{A}, \bar{\boldsymbol{r}}_0), \tag{B.12}$$

where $c_{n-1} = (-1)^{n-1}\prod_{i=0}^{n-2}\alpha_i$. Then, from (B.11), (B.12), and the conditions (2.35), the formula α_{n-1} in (B.9) can be rewritten by

$$\alpha_{n-1} = \frac{(\bar{\boldsymbol{r}}_{n-1}, \boldsymbol{r}_{n-1}) - (\bar{\boldsymbol{z}}_1, \boldsymbol{r}_{n-1})}{(\bar{\boldsymbol{p}}_{n-1}, A\boldsymbol{p}_{n-1}) - (\bar{\boldsymbol{z}}_2, A\boldsymbol{p}_{n-1})} = \frac{(\bar{\boldsymbol{r}}_{n-1}, \boldsymbol{r}_{n-1})}{(\bar{\boldsymbol{p}}_{n-1}, A\boldsymbol{p}_{n-1})}. \tag{B.13}$$

Similarly, the formula β_{n-1} in (B.10) can be rewritten by

$$\beta_{n-1} = \frac{(\bar{\boldsymbol{r}}_n, \boldsymbol{r}_n)}{(\bar{\boldsymbol{r}}_{n-1}, \boldsymbol{r}_{n-1})}. \tag{B.14}$$

Finally, we give an update formula of the nth approximate solution $\boldsymbol{x}_n$. From the relation (2.33), and recalling $\boldsymbol{r}_n = \boldsymbol{b} - A\boldsymbol{x}_n$, we obtain

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \alpha_{n-1}\boldsymbol{p}_{n-1}. \tag{B.15}$$

From (2.32)-(2.34) and (B.13)-(B.15), the algorithm of COCG is obtained

B.3 KS methods for non-Hermitian linear systems

◇ The Bi-CG method

We begin with the following recurrences:

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \alpha_{n-1}A\boldsymbol{p}_{n-1}, \tag{B.16}$$

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}. \tag{B.17}$$

Then, we show that Bi-CG is obtained by (B.16), (B.17), and the Petrov-Galerkin approach (2.4). Since $((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, \boldsymbol{r}_n) = 0$ by the Petrov-Galerkin approach (2.4), it follows from (B.16) that α_{n-1} is given as follows:

$$\alpha_{n-1} = \frac{((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, \boldsymbol{r}_{n-1})}{((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1})}. \tag{B.18}$$

The inner product of $\boldsymbol{z} \in K_{n-1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$ and (B.16) gives the form

$$(\boldsymbol{z}, \boldsymbol{r}_n) = (\boldsymbol{z}, \boldsymbol{r}_{n-1}) - \alpha_{n-1}(\boldsymbol{z}, A\boldsymbol{p}_{n-1}).$$

From the condition (2.4), it follows that $(\boldsymbol{z}, \boldsymbol{r}_n) = (\boldsymbol{z}, \boldsymbol{r}_{n-1}) = 0$. Thus, we have the orthogonality

$$A\boldsymbol{p}_{n-1} \perp K_{n-1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*). \tag{B.19}$$

To obtain the computational formula for β_{n-1}, let us consider the inner product of $(A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*$ and (B.17) multiplied by A that gives the form

$$\begin{aligned}((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{p}_n) &= ((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{r}_n) + \beta_{n-1}((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1}) \\ &= ((A^{\mathrm{H}})^{n}\boldsymbol{r}_0^*, \boldsymbol{r}_n) + \beta_{n-1}((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1}) \\ &= 0.\end{aligned}$$

Then, the original form of β_{n-1} can be given as follows:

$$\beta_{n-1} = -\frac{((A^{\mathrm{H}})^{n}\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1})}.$$

Moreover, making use of the formula (B.18), we obtain another original form for β_{n-1}.

$$\beta_{n-1} = -\alpha_{n-1}\frac{((A^{\mathrm{H}})^{n}\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, \boldsymbol{r}_{n-1})}. \tag{B.20}$$

Similarly, let us define residual vectors $\boldsymbol{r}_n^* \in K_{n+1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$ as the two-term recurrences

$$\begin{aligned}\boldsymbol{r}_n^* &= \boldsymbol{r}_{n-1}^* - \bar{\alpha}_{n-1}A^{\mathrm{H}}\boldsymbol{p}_{n-1}^*, \\ \boldsymbol{p}_n^* &= \boldsymbol{r}_n^* + \bar{\beta}_{n-1}\boldsymbol{p}_{n-1}^*,\end{aligned}$$

where $\boldsymbol{p}_0^* = \boldsymbol{r}_0^*$. From the above, the vectors $\boldsymbol{r}_{n-1}^*$ and $\boldsymbol{p}_{n-1}^*$ can be expanded as

$$\begin{aligned}\boldsymbol{r}_{n-1}^* &= \bar{R}_{n-1}(A^{\mathrm{T}})\boldsymbol{r}_0^* = \Big((-1)^{n-1}\prod_{i=0}^{n-2}\bar{\alpha}_i\Big)(A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^* + \boldsymbol{z}_1, \\ \boldsymbol{p}_{n-1}^* &= \bar{P}_{n-1}(A^{\mathrm{T}})\boldsymbol{r}_0^* = \Big((-1)^{n-1}\prod_{i=0}^{n-2}\bar{\alpha}_i\Big)(A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^* + \boldsymbol{z}_2,\end{aligned}$$

where $\boldsymbol{z}_1, \boldsymbol{z}_2 \in K_{n-1}(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$. Thus the α_{n-1} and β_{n-1} can be computed by Petrov-Galerkin condition (2.4) and orthogonality property (B.19) and the following relations:

$$\begin{aligned}(\boldsymbol{r}_{n-1}^*, \boldsymbol{r}_{n-1}) &= (\bar{c}(A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, \boldsymbol{r}_{n-1}) + (\boldsymbol{z}_1, \boldsymbol{r}_{n-1}) \\ &= c((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, \boldsymbol{r}_{n-1}), \\ (\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_{n-1}) &= (\bar{c}(A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1}) + (\boldsymbol{z}_2, A\boldsymbol{p}_{n-1}) \\ &= c((A^{\mathrm{H}})^{n-1}\boldsymbol{r}_0^*, A\boldsymbol{p}_{n-1}),\end{aligned}$$

where $\bar{c} = (-1)^{n-1}\prod_{i=0}^{n-2}\bar{\alpha}_i$. Substituting the above formulas for (B.18) and (B.20), we obtain

$$\alpha_{n-1} = \frac{(\boldsymbol{r}_{n-1}^*, \boldsymbol{r}_{n-1})}{(\boldsymbol{p}_{n-1}^*, A\boldsymbol{p}_{n-1})}, \quad \beta_{n-1} = \frac{(\boldsymbol{r}_n^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_{n-1}^*, \boldsymbol{r}_{n-1})}.$$

From the above, we obtain Algorithm 2.16.

◇ KS methods based on normal equations

Here, we consider how to use CG for solving non-Hermitian linear systems. Since the

coefficient matrix A is non-Hermitian, CG can not be applied to the system $A\boldsymbol{x} = \boldsymbol{b}$; however, if we consider the following normal equations:

$$A^{\mathrm{H}}A\boldsymbol{x} = A^{\mathrm{H}}\boldsymbol{b} \tag{B.21}$$

or

$$AA^{\mathrm{H}}\boldsymbol{y} = \boldsymbol{b},\ \boldsymbol{x} = A^{\mathrm{H}}\boldsymbol{y}, \tag{B.22}$$

then it is natural to solve the above equations by CG since $A^{\mathrm{H}}A$ and AA^{H} are Hermitian positive definite. If we apply CG to (B.21), then we have the CGNR method (or CGLS) [53] given below:

Algorithm B.1: CGNR method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \beta_{-1} = 0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:
$\boldsymbol{p}_n = A^{\mathrm{H}}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},$
$\alpha_n = \dfrac{(A^{\mathrm{H}}\boldsymbol{r}_n, A^{\mathrm{H}}\boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)},$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,$
$\beta_n = \dfrac{(A^{\mathrm{H}}\boldsymbol{r}_{n+1}, A^{\mathrm{H}}\boldsymbol{r}_{n+1})}{(A^{\mathrm{H}}\boldsymbol{r}_n, A^{\mathrm{H}}\boldsymbol{r}_n)}.$
end

It is clear from the algorithm of CGNR and (2.27) that it minimizes the $A^{\mathrm{H}}A$-norm of the error, or the 2-norm of the residual vector:

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A^{\mathrm{H}}A, A^{\mathrm{H}}\boldsymbol{r}_0)} \|\boldsymbol{x}^* - \boldsymbol{x}_n\|_{A^{\mathrm{H}}A} = \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A^{\mathrm{H}}A, A^{\mathrm{H}}\boldsymbol{r}_0)} \|\boldsymbol{r}_n\|.$$

On the other hand, if we apply CG to (B.22), then we have the CGNE method [18] (or Craig's method) given below:

Algorithm B.2: CGNE method

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \beta_{-1} = 0,$
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:
$\boldsymbol{p}_n = A^{\mathrm{H}}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1},$
$\alpha_n = \dfrac{(\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{p}_n, \boldsymbol{p}_n)},$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n,$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,$
$\beta_n = \dfrac{(\boldsymbol{r}_{n+1}, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n, \boldsymbol{r}_n)}.$
end

It is clear from the algorithm of CGNE and (2.27) that it minimizes the AA^{H}-norm of the error:

$$\min_{\boldsymbol{y}_n \in \boldsymbol{y}_0 + K_n(AA^{\mathrm{H}}, \boldsymbol{r}_0)} \|\boldsymbol{y}^* - \boldsymbol{y}_n\|_{A^{\mathrm{H}}A}.$$

It follows from the above fact and $\boldsymbol{y}_n = A^{-\mathrm{H}}\boldsymbol{x}_n$ that CGNE generates $\boldsymbol{x}_n$ such that

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + A^{\mathrm{H}} K_n(AA^{\mathrm{H}}, \boldsymbol{r}_0)} \|\boldsymbol{x}^* - \boldsymbol{x}_n\| = \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A^{\mathrm{H}}A, A^{\mathrm{H}}\boldsymbol{r}_0)} \|\boldsymbol{x}^* - \boldsymbol{x}_n\|.$$

The convergence of CGNR and CGNE is optimal in some special cases [40, 61]. However, since the condition number of AA^{H} is twice as that of A, it may lead to deteriorate convergence behavior by Theorem 2.4.1.

If the coefficient matrix is ill-conditioned, LSQR [63] is often preferred. This algorithm is based on the Golub-Kahan bidiagonalization process [46] and the process is obtained by applying the Lanczos process to

$$\begin{pmatrix} I & A \\ A^{\mathrm{H}} & O \end{pmatrix} \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{x} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ 0 \end{pmatrix} \quad \text{or} \quad \tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}.$$

The process begins with a unit vector:

$$\boldsymbol{w}_1 := \begin{pmatrix} \boldsymbol{u}_1 \\ 0 \end{pmatrix} = \frac{1}{\|\boldsymbol{b}\|} \begin{pmatrix} \boldsymbol{b} \\ 0 \end{pmatrix}.$$

Let us define $h_{0,1} := \|\boldsymbol{b}\|$, then we have $h_{01}\boldsymbol{u}_1 = \boldsymbol{b}$. For the first step, it follows from the Lanczos process that

$$\begin{aligned} \tilde{\boldsymbol{w}}_2 &= \tilde{A}\boldsymbol{w}_1 - \alpha_1 \boldsymbol{w}_1 = \begin{pmatrix} 0 \\ A^{\mathrm{H}}\boldsymbol{u}_1 \end{pmatrix}, \\ \boldsymbol{w}_2 &= \begin{pmatrix} 0 \\ \boldsymbol{v}_1 \end{pmatrix} = \frac{1}{\|A^{\mathrm{H}}\boldsymbol{u}_1\|} \begin{pmatrix} 0 \\ A^{\mathrm{H}}\boldsymbol{u}_1 \end{pmatrix}, \end{aligned}$$

where using $h_{1,1} := \|A^{\mathrm{H}}\boldsymbol{u}_1\|$, it follows that $h_{1,1}\boldsymbol{v}_1 = A^{\mathrm{H}}\boldsymbol{u}_1$. For the second step, we have

$$\begin{aligned} \tilde{\boldsymbol{w}}_3 &= \tilde{A}\boldsymbol{w}_2 - \alpha_2 \boldsymbol{w}_2 - \beta_1 \boldsymbol{w}_1 = \begin{pmatrix} A\boldsymbol{v}_1 - \beta_1 \boldsymbol{u}_1 \\ 0 \end{pmatrix}, \\ \boldsymbol{w}_3 &= \begin{pmatrix} \boldsymbol{u}_2 \\ 0 \end{pmatrix} = \frac{1}{\|A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1\|} \begin{pmatrix} A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1 \\ 0 \end{pmatrix}. \end{aligned}$$

Similar to the above definitions, $h_{2,1} := \|A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1\|$ gives $h_{2,1}\boldsymbol{u}_2 = A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1$. For the third step, it follows that

$$\begin{aligned} \tilde{\boldsymbol{w}}_4 &= \tilde{A}\boldsymbol{w}_3 - \alpha_3 \boldsymbol{w}_3 - \beta_2 \boldsymbol{w}_2 = \begin{pmatrix} 0 \\ A^{\mathrm{H}}\boldsymbol{u}_2 - \beta_2 \boldsymbol{v}_1 \end{pmatrix}, \\ \boldsymbol{w}_4 &= \begin{pmatrix} 0 \\ \boldsymbol{v}_2 \end{pmatrix} = \frac{1}{\|A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1\|} \begin{pmatrix} 0 \\ A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1 \end{pmatrix}. \end{aligned}$$

From the above, we have $h_{2,2}\boldsymbol{v}_2 = A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1$ where $h_{2,2} := \|A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1\|$. Hence, from the above recurrences, we have the following matrix form:

$$\begin{aligned} &A\boldsymbol{v}_1 = [\boldsymbol{u}_1, \boldsymbol{u}_2] \begin{pmatrix} h_{1,1} \\ h_{2,1} \end{pmatrix} \Longleftrightarrow AV_1 = U_2 B_{2,1}, \\ &A^{\mathrm{H}}[\boldsymbol{u}_1, \boldsymbol{u}_2] = [\boldsymbol{v}_1, \boldsymbol{v}_2] \begin{pmatrix} h_{1,1} & h_{2,1} \\ & h_{2,2} \end{pmatrix} \Longleftrightarrow A^{\mathrm{H}}U_2 = V_1 B_{2,1}^{\mathrm{T}} + h_{2,2}\boldsymbol{v}_2 \boldsymbol{e}_2^{\mathrm{T}}. \end{aligned}$$

From the above results, it can be easily generalized, and the generalization leads to the following Golub-Kahan bidiagonalization process [46]:

Algorithm B.3: The Golub-Kahan bidiagonalization process

set $h_{0,1} = \|\boldsymbol{b}\|, \qquad \boldsymbol{u}_1 = \boldsymbol{b}/h_{0,1},$
set $h_{1,1} = \|A^{\mathrm{H}}\boldsymbol{u}_1\|, \ \boldsymbol{v}_1 = A^{\mathrm{H}}\boldsymbol{u}_1/h_{1,1},$
for $n = 1, 2, \ldots$
$\quad \tilde{\boldsymbol{u}}_{n+1} = A\boldsymbol{v}_n - h_{n,n}\boldsymbol{u}_n,$
$\quad h_{n+1,n} = \|\tilde{\boldsymbol{u}}_{n+1}\|,$
$\quad \boldsymbol{u}_{n+1} = \tilde{\boldsymbol{u}}_{n+1}/h_{n+1,n},$
$\quad \tilde{\boldsymbol{v}}_{n+1} = A^{\mathrm{H}}\boldsymbol{u}_{n+1} - h_{n+1,n}\boldsymbol{v}_n,$
$\quad h_{n+1,n+1} = \|\tilde{\boldsymbol{v}}_{n+1}\|,$
$\quad \boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/h_{n+1,n+1}.$
end

The above process can be written in matrix form

$$\begin{aligned} AV_n &= U_{n+1}B_{n+1,n}, \\ A^{\mathrm{H}}U_{n+1} &= V_nB_{n+1,n}^{\mathrm{T}} + h_{n+1,n+1}\boldsymbol{v}_{n+1}\boldsymbol{e}_{n+1}^{\mathrm{T}}, \end{aligned} \tag{B.23}$$

where

$$U_n := [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n], \ V_n := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n], \ B_{n+1,n} := \begin{pmatrix} h_{1,1} & & \\ h_{2,1} & \ddots & \\ & \ddots & h_{n,n} \\ & & h_{n+1,n} \end{pmatrix}.$$

It is clear from the bidiagonalization process that U_n and V_n satisfy the following properties:

$$U_n^{\mathrm{H}}U_n = V_n^{\mathrm{H}}V_n = I_n. \tag{B.24}$$

Now, we are ready for the derivation of LSQR. The LSQR method generates the nth approximate solution with V_n and $\boldsymbol{x}_0 = \boldsymbol{0}$, *i.e.*, $\boldsymbol{x}_n = V_n\boldsymbol{y}_n$, where $\boldsymbol{y}$ is determined by minimizing the 2-norm of the corresponding residual vector

$$\boldsymbol{r}_n = \boldsymbol{b} - AV_n\boldsymbol{y}_n. \tag{B.25}$$

The minimization can be achieved by using the property (B.24). Substituting (B.23) into (B.25) leads to

$$\begin{aligned} \boldsymbol{r}_n &= \boldsymbol{b} - U_{n+1}B_n\boldsymbol{y}_n \\ &= U_{n+1}(h_{0,1}\boldsymbol{e}_1 - B_{n+1,n}\boldsymbol{y}_n). \end{aligned}$$

From the property (B.24) it follows that

$$\min_{\boldsymbol{y}_n \in \mathcal{C}^n} \|\boldsymbol{r}_n\| = \min_{\boldsymbol{y}_n \in \mathcal{C}^n} \|h_{0,1}\boldsymbol{e}_1 - B_{n+1,n}\boldsymbol{y}_n\|.$$

Hence, similar to MINRES, $\boldsymbol{y}_n$ can be obtained by using Givens rotations and triangular-solve, and by analogy with the derivation of MINRES, we readily have the following LSQR algorithm:

Algorithm B.4: LSQR method

set $\boldsymbol{x}_0 = \boldsymbol{0},\ h_{0,1} = \|\boldsymbol{b}|,\ \boldsymbol{u}_1 = \boldsymbol{b}/h_{0,1},$
set $h_{1,1} = \|A^{\mathrm{H}}\boldsymbol{u}_1\|,\ \boldsymbol{v}_1 = A^{\mathrm{H}}\boldsymbol{u}_1/h_{1,1},$
for $n = 1, 2, \ldots,$ do:
(G-K bidiagonalization process)
$\tilde{\boldsymbol{u}}_{n+1} = A\boldsymbol{v}_n - h_{n,n}\boldsymbol{u}_n,$
$h_{n+1,n} = \|\tilde{\boldsymbol{u}}_{n+1}\|,$
$\boldsymbol{u}_{n+1} = \tilde{\boldsymbol{u}}_{n+1}/h_{n+1,n},$
$\tilde{\boldsymbol{v}}_{n+1} = A^{\mathrm{H}}\boldsymbol{u}_{n+1} - h_{n+1,n}\boldsymbol{v}_n,$
$h_{n+1,n+1} = \|\tilde{\boldsymbol{v}}_{n+1}\|,$
$\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/h_{n+1,n+1},$
(Givens rotations)
for $i = \max\{1, n-1\}, \ldots, n-1$ do:
$$\begin{pmatrix} t_{i,n} \\ t_{i+1,n} \end{pmatrix} = \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} t_{i,n} \\ t_{i+1,n} \end{pmatrix},$$
end

$$c_n = \frac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}},$$
$$\bar{s}_n = \frac{t_{n+1,n}}{t_{n,n}} c_n,$$
$t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n},$
$t_{n+1,n} = 0,$
$$\begin{pmatrix} g_n \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{pmatrix} \begin{pmatrix} g_n \\ 0 \end{pmatrix},$$
(Update $\boldsymbol{x}_n$)
$\boldsymbol{p}_n = (\boldsymbol{v}_n - t_{n-1,n}\boldsymbol{p}_{n-1})/t_{n,n},$
$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + g_n\boldsymbol{p}_n,$
(Check convergence)
if $|g_{n+1}|/\|\boldsymbol{b}\| \le \epsilon,$ then stop.
end

◇ KS methods based on the Arnoldi process

Here, we describe algorithms that are based on the Arnoldi process and minimal residual approach. One of the most successful algorithms based on the Arnoldi process is the Generalized Minimal RESidual (GMRES) method proposed by Saad and Schultz [71].

The derivation process of GMRES is closely related to MINRES. This method generates $\boldsymbol{x}_n$ that minimizes $\|\boldsymbol{b} - A\boldsymbol{x}_n\|$ over the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$. It is known that this can be efficiently achieved by using the Arnoldi process. Now, we give the derivation process of the GMRES method. Let V_n be the orthonormal basis of $K_n(A, \boldsymbol{r}_0)$. Then, since $\boldsymbol{x}_n$ lies in the affine space $\boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)$, we have

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n\boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathcal{C}^n.$$

The corresponding residual vector is written as

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n\boldsymbol{y}_n.$$

From the Arnoldi process it follows that

$$\boldsymbol{r}_n = V_{n+1}(\beta\boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}_n),$$

where $\beta := \|\boldsymbol{r}_0\|$. Hence, the 2-norm of the residual vector can be minimized by choosing $\boldsymbol{y}_n$ such that

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y} \in \mathcal{C}^n} \|\beta\boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}\|.$$

The vector $\boldsymbol{y}_n$ can be readily obtained by using Givens rotations. Since V_{n+1} satisfies $V_{n+1}^{\mathrm{H}}V_{n+1} = I_{n+1}$, we have the relation $\|\boldsymbol{r}_n\| = \|\boldsymbol{y}_n\|$. Hence, we can use the value of $\|\boldsymbol{y}_n\|$ as the stopping criterion instead of the direct computation of $\|\boldsymbol{r}_n\|$.

Now, we describe the algorithm of GMRES below. The following algorithm is based on the Arnoldi process with modified Gram-Schmidt.

Algorithm B.5: GMRES method with MGS

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
set $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \ldots, 0)^{\mathrm{T}}$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$,
for $n = 1, 2, \ldots,$ do:
(Arnoldi process with MGS)
  $\boldsymbol{t} = A\boldsymbol{v}_n$,
  for $i = 1, 2, \ldots, n$ do:
    $h_{i,n} = (\boldsymbol{v}_i, \boldsymbol{t})$,
    $\boldsymbol{t} = \boldsymbol{t} - h_{i,n}\boldsymbol{v}_i$,
  end
  $h_{n+1,n} = \|\boldsymbol{t}\|$,
  $\boldsymbol{v}_{n+1} = \dfrac{\boldsymbol{t}}{h_{n+1,n}}$,
(Givens rotations)
  for $i = 1, \ldots, n-1$ do:
    $\begin{pmatrix} h_{i,n} \\ h_{i+1,n} \end{pmatrix} = \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} h_{i,n} \\ h_{i+1,n} \end{pmatrix}$,
  end
  $c_n = \dfrac{|h_{n,n}|}{\sqrt{|h_{n,n}|^2 + |h_{n+1,n}|^2}}$,
  $\bar{s}_n = \dfrac{h_{n+1,n}}{h_{n,n}} c_n$,
  $h_{n,n} = c_n h_{n,n} + s_n h_{n+1,n}$,
  $h_{n+1,n} = 0$,
  $\begin{pmatrix} g_n \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{pmatrix} \begin{pmatrix} g_n \\ 0 \end{pmatrix}$,
(Update $\boldsymbol{x}_n$)
(Check convergence)
  if $|g_{n+1}| \le \epsilon \|\boldsymbol{b}\|$, then
    $\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n H_n^{-1} \boldsymbol{g}_n$.
  end if and stop
end

The above algorithm is also referred to as the full GMRES method. This method finds approximate solution $\boldsymbol{x}_n$ such that

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + K_n(A, \boldsymbol{r}_0)} \|\boldsymbol{r}_n\| \quad \text{or} \quad \boldsymbol{r}_n \perp AK_n(A, \boldsymbol{r}_0).$$

From the above property, we see that the residual 2-norm decreases monotonically and that in exact arithmetic this method converges in at most N iterations. However, computational work and memory increase linearly with the number of iterations. To remedy this difficulty, Saad and Schultz proposed restarted version of GMRES denoted by GMRES(m).

Algorithm B.6: GMRES(m) method with MGS

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
set $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \ldots, 0)^{\mathrm{T}}$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$,
for $n = 1, 2, \ldots, m$ do:
(Arnoldi process with MGS)
  $\boldsymbol{t} = A\boldsymbol{v}_n$,
  for $i = 1, 2, \ldots, n$ do:
    $h_{i,n} = (\boldsymbol{v}_i, \boldsymbol{t})$,
    $\boldsymbol{t} = \boldsymbol{t} - h_{i,n}\boldsymbol{v}_i$,
  end
  $h_{n+1,n} = \|\boldsymbol{t}\|$,
  $\boldsymbol{v}_{n+1} = \dfrac{\boldsymbol{t}}{h_{n+1,n}}$,
(Givens rotations)
  for $i = 1, \ldots, n-1$ do:
    $\begin{pmatrix} h_{i,n} \\ h_{i+1,n} \end{pmatrix} = \begin{pmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{pmatrix} \begin{pmatrix} h_{i,n} \\ h_{i+1,n} \end{pmatrix}$,
  end
  $c_n = \dfrac{|h_{n,n}|}{\sqrt{|h_{n,n}|^2 + |h_{n+1,n}|^2}}$,

$$
\begin{aligned}
&\bar{s}_n = \frac{h_{n+1,n}}{h_{n,n}} c_n, \\
&h_{n,n} = c_n h_{n,n} + s_n h_{n+1,n}, \\
&h_{n+1,n} = 0, \\
&\begin{pmatrix} g_n \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{pmatrix} \begin{pmatrix} g_n \\ 0 \end{pmatrix}, \\
&\text{(Update } \boldsymbol{x}_n\text{)} \\
&\text{(Check convergence)}
\end{aligned}
\quad\Bigg|\quad
\begin{aligned}
&\text{if } |g_{n+1}| \le \epsilon \|\boldsymbol{b}\|, \text{ then} \\
&\quad \boldsymbol{x}_n = \boldsymbol{x}_0 + V_n H_n^{-1} \boldsymbol{g}_n, \\
&\text{end if and stop} \\
&\text{end} \\
&\quad \boldsymbol{x}_n = \boldsymbol{x}_0 + V_n H_n^{-1} \boldsymbol{g}_n, \\
&\quad \boldsymbol{x}_0 = \boldsymbol{x}_n. \\
&\text{repeat}
\end{aligned}
$$

Althogh GMRES(m), in exact arithmetic, does not guarantee the convergence in at most N iterations, it is practical and useful for solving non-Hermitian linear systems.

To improve the accuracy of orthogonality of the Arnoldi basis, a Householder variant of the Arnoldi process was developed by Walker [92]. GMRES with the Householder orthogonalization is numerically, as we can expect, better than GMRES with CGS and MGS though the double of the computational cost is required for orthogonalization procedure. The comparison of computational costs and required memory is made in [70, p.158].

The GCR method

The Generalized Conjugate Residual (GCR) method [31] is another Krylov solver based on the Arnoldi process. The algorithm is obtained by the following Arnoldi process with the weight $A^{\mathrm{H}}A$:

$$
\begin{aligned}
&\text{set } \boldsymbol{v}_1 = \frac{\boldsymbol{r}_0}{\|\boldsymbol{r}_0\|_{A^{\mathrm{H}}A}}, \\
&\text{for } n = 1, 2, \ldots, \text{ do:} \\
&\quad h_{i,n} = (\boldsymbol{v}_i, A\boldsymbol{v}_n)_{A^{\mathrm{H}}A}, \quad i = 1, 2, \ldots, n, \\
&\quad \tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \sum_{i=1}^{n} h_{i,n} \boldsymbol{v}_i, \\
&\quad h_{n+1,n} = \|\tilde{\boldsymbol{v}}_{n+1}\|_{A^{\mathrm{H}}A}, \\
&\quad \boldsymbol{v}_{n+1} = \frac{\tilde{\boldsymbol{v}}_{n+1}}{h_{n+1,n}}. \\
&\text{end}
\end{aligned}
\tag{B.26}
$$

This generates $A^{\mathrm{H}}A$-orthonormal vectors, *i.e.*,

$$
(A\boldsymbol{v}_i, A\boldsymbol{v}_j) = 0 \quad \text{for} \quad i \neq j.
$$

Let V_n be a matrix with columns $[\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{n-1}]$. Then, GCR generates $\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n \boldsymbol{y}_n$ such that $\min \|\boldsymbol{r}_n\|$. The corresponding residual vector can be written as

$$
\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n \boldsymbol{y}_n = \boldsymbol{r}_0 - \sum_{i=1}^{n} y_i A\boldsymbol{v}_i. \tag{B.27}
$$

From the above relation, we have

$$
\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - y_n A\boldsymbol{v}_n. \tag{B.28}
$$

Since $A\boldsymbol{v}_i$ is an orthonormal vector, to minimize the residual 2-norm, y_i is determined by

$$y_i = (A\boldsymbol{v}_i, \boldsymbol{r}_0), \quad i = 1, \ldots, n.$$

Here, we define the vector $\boldsymbol{p}_n := -y_n \hat{\boldsymbol{v}}_{n+1}$. Then, it follows from (B.26) that

$$\boldsymbol{p}_n = -y_n A\boldsymbol{v}_n - \sum_{i=1}^{n} (\boldsymbol{v}_i, -y_n A\boldsymbol{v}_n)_{A^{\mathrm{H}}A} \boldsymbol{v}_i.$$

Substituting (B.28) in the precious recurrence, we have

$$\begin{aligned} \boldsymbol{p}_n &= \boldsymbol{r}_n - \boldsymbol{r}_{n-1} - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_n - \boldsymbol{r}_{n-1})_{A^{\mathrm{H}}A} \boldsymbol{v}_i \\ &= \boldsymbol{r}_n - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_n)_{A^{\mathrm{H}}A} \boldsymbol{v}_i - \left(\boldsymbol{r}_{n-1} - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_{n-1})_{A^{\mathrm{H}}A} \boldsymbol{v}_i \right). \end{aligned} \tag{B.29}$$

Since $\boldsymbol{r}_{n-1}$ lies in $K_n(A, \boldsymbol{r}_0)$, it can be expanded as

$$\boldsymbol{r}_{n-1} = \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_{n-1})_{A^{\mathrm{H}}A} \boldsymbol{v}_i. \tag{B.30}$$

Hence, from (B.29) and (B.30), the vector $\boldsymbol{p}_n$ can be written as

$$\boldsymbol{p}_n = \boldsymbol{r}_n - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_n)_{A^{\mathrm{H}}A} \boldsymbol{v}_i. \tag{B.31}$$

On the other hand, $\boldsymbol{p}_n$ and $\boldsymbol{v}_{n+1}$ are related with

$$\boldsymbol{p}_n = -\frac{y_n}{|y_n|} \|A\boldsymbol{p}_n\| \boldsymbol{v}_{n+1}$$

since it follows that

$$\begin{aligned} \boldsymbol{p}_n &= -y_n \hat{\boldsymbol{v}}_{n+1} \\ &= -y_n \|\hat{\boldsymbol{v}}_{n+1}\|_{A^{\mathrm{H}}A} \boldsymbol{v}_{n+1} \\ &= -y_n \sqrt{(A\hat{\boldsymbol{v}}_{n+1}, A\hat{\boldsymbol{v}}_{n+1})}\ \boldsymbol{v}_{n+1} \\ &= -y_n \sqrt{(-\frac{1}{y_n} A\boldsymbol{p}_n, -\frac{1}{y_n} A\boldsymbol{p}_n)}\ \boldsymbol{v}_{n+1} \\ &= -\frac{y_n}{|y_n|} \sqrt{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}\ \boldsymbol{v}_{n+1}. \end{aligned}$$

Hence, using $\boldsymbol{v}_n = -\frac{|y_{n-1}|}{y_{n-1}} \cdot \frac{\boldsymbol{p}_{n-1}}{\|A\boldsymbol{p}_{n-1}\|}$, we have the following relations:

$$\begin{aligned} y_n A\boldsymbol{v}_n &= (A\boldsymbol{v}_n, \boldsymbol{r}_0) A\boldsymbol{v}_n = \frac{(A\boldsymbol{p}_{n-1}, \boldsymbol{r}_0)}{(A\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})} A\boldsymbol{p}_{n-1}, \\ (\boldsymbol{v}_i, \boldsymbol{r}_n)_{A^{\mathrm{H}}A} \boldsymbol{v}_i &= \frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_n)}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})} \boldsymbol{p}_{i-1}. \end{aligned}$$

Therefore, substituting the above results for (B.28) and (B.31), we have

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \frac{(A\boldsymbol{p}_{n-1}, \boldsymbol{r}_0)}{(A\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})} A\boldsymbol{p}_{n-1},$$
$$\boldsymbol{p}_n = \boldsymbol{r}_n - \sum_{i=1}^{n} \frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_n)}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})} \boldsymbol{p}_{i-1}.$$

From (B.27) and the fact that $\boldsymbol{p}_{n-1} \in \mathrm{span}\{\boldsymbol{v}_n\}$ is $A^{\mathrm{H}}A$-orthogonal to $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1}$, it follows that

$$(A\boldsymbol{p}_{n-1}, \boldsymbol{r}_{n-1}) = (A\boldsymbol{p}_{n-1}, \boldsymbol{r}_0) - \sum_{i=1}^{n-1} y_i (A\boldsymbol{p}_{n-1}, A\boldsymbol{v}_i) = (A\boldsymbol{p}_{n-1}, \boldsymbol{r}_0).$$

Now, let us define α_{n-1} and $\beta_{n-1,i}$ as

$$\alpha_{n-1} = \frac{(A\boldsymbol{p}_{n-1}, \boldsymbol{r}_{n-1})}{(A\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})}, \tag{B.32}$$

$$\beta_{n-1,i} = -\frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_n)}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}. \tag{B.33}$$

Then, we have

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \alpha_{n-1} A\boldsymbol{p}_{n-1}, \tag{B.34}$$

$$\boldsymbol{p}_n = \boldsymbol{r}_n + \sum_{i=1}^{n} \beta_{n-1,i} \boldsymbol{p}_{i-1}. \tag{B.35}$$

From the recurrence relation of the residual vector, we obtain

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \alpha_{n-1} \boldsymbol{p}_{n-1}. \tag{B.36}$$

To reduce the number of matrix-vector multiplications, use the following recurrences:

$$A\boldsymbol{p}_n = A\boldsymbol{r}_n + \sum_{i=1}^{n} \beta_{n-1,i} A\boldsymbol{p}_{i-1}. \tag{B.37}$$

From (B.32)-(B.37), the algorithm of GCR is obtained below:

Algorithm B.7: GCR

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon \|\boldsymbol{b}\|$ do:
$\alpha_n = \dfrac{(A\boldsymbol{p}_n, \boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}$,
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$,
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$,
$\beta_{n,i} = -\dfrac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_{n+1})}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}, \quad 1 \le i \le n+1,$
$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i} \boldsymbol{p}_{i-1},$
$(A\boldsymbol{p}_{n+1} = A\boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i} A\boldsymbol{p}_{i-1}.)$
end

The properties of GCR are given below [31, Theorem 3.1].

(G1) $(A\boldsymbol{p}_i, A\boldsymbol{p}_j) = 0, \qquad i \neq j,$

(G2) $(\boldsymbol{r}_i, A\boldsymbol{p}_j) = 0, \qquad i > j,$

(G3) $(\boldsymbol{r}_i, A\boldsymbol{p}_i) = (\boldsymbol{r}_i, A\boldsymbol{r}_i),$

(G4) $(\boldsymbol{r}_i, A\boldsymbol{r}_j) = 0, \qquad i > j,$

(G5) $(\boldsymbol{r}_i, A\boldsymbol{p}_i) = (\boldsymbol{r}_0, A\boldsymbol{p}_i), \quad i \geq j.$

From the above properties, we can see that the previous derivation is based on the property (G1). We can also derive the algorithm of GCR based on (G4) by using an A-orthogonalization process of the Krylov subspace.

In exact arithmetic, GCR as well as GMRES converges in at most N iterations. However, GCR has the same difficulty as GMRES in that the computational work and the required memory increase linearly with the number of iterations. Hence, restarted version of GCR which is referred to as GCR(k) was proposed [31].

Algorithm B.8: GCR(k)

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \boldsymbol{p}_0 = \boldsymbol{r}_0,$
for $n = 0, 1, \ldots, k$ until $\|\boldsymbol{r}_n\| \leq \epsilon\|\boldsymbol{b}\|$ do:
$\alpha_n = \dfrac{(A\boldsymbol{p}_n, \boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)},$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n,$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,$
$\beta_{n,i} = -\dfrac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_{n+1})}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}, \quad 1 \leq i \leq n+1,$
$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i}\boldsymbol{p}_{i-1},$
$(A\boldsymbol{p}_{n+1} = A\boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i} A\boldsymbol{p}_{i-1},)$
end
$\boldsymbol{x}_0 = \boldsymbol{x}_{k+1}.$
repeat

Another alternative is to save only k direction vectors:

$$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{i=1}^{\min\{k,n+1\}} \beta_{n,i}\boldsymbol{p}_{i-1}.$$

This method is known as Orthomin(k)[31]. Note that the Orthomin method was originally proposed by Vinsome [91].

Algorithm B.9: Orthomin(k)

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0,\ \boldsymbol{p}_0 = \boldsymbol{r}_0$,
for $n = 0, 1, \ldots,$ until $\|\boldsymbol{r}_n\| \le \epsilon\|\boldsymbol{b}\|$ do:

$$\alpha_n = \frac{(A\boldsymbol{p}_n, \boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)},$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n,$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n,$$
$$\beta_{n,i} = -\frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_{n+1})}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}, \quad n-k+2 \le i \le n+1,$$
$$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{i=1}^{\min\{k,n+1\}} \beta_{n,i}\boldsymbol{p}_{i-1}.$$
$$(A\boldsymbol{p}_{n+1} = A\boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i} A\boldsymbol{p}_{i-1},)$$

end
$\boldsymbol{x}_0 = \boldsymbol{x}_{n+1}$.
repeat

The following theorem for GCR, GCR(k), and Orthomin(k) is given by Eisenstat *et al.*:

Theorem B.3.1 (Eisenstat *et al.* **[31, Theorem 4.4])** *Let M be the Hermitian part of A and let R be the skew-Hermitian part of A. If $\{\boldsymbol{r}_n\}$ is the sequence of residuals generated by GCR, GCR(k), or Orthomin(k), then*

$$\|\boldsymbol{r}_n\| \le \left(1 - \frac{\lambda_{\min}(M)^2}{\lambda_{\max}(A^H A)}\right)^{n/2} \|\boldsymbol{r}_0\|,$$

and

$$\|\boldsymbol{r}_n\| \le \left(1 - \frac{\lambda_{\min}(M)^2}{\lambda_{\min}(M)\lambda_{\max}(M) + \rho(R)^2}\right)^{n/2} \|\boldsymbol{r}_0\|.$$

We can see from the above theorem that if A is close to I, then the three methods converge fast since $\lambda_{\min}(M) \approx \lambda_{\max}(M) \approx 1$ and $R \approx O$.

Appendix C

Other preconditioners

C.1 Preconditioners based on stationary iterative methods

The Jacobi (or diagonal) preconditioner is the simplest preconditioner and usually more effective than solving original linear systems $A\boldsymbol{x} = \boldsymbol{b}$. The Jacobi preconditioner is constructed by using only diagonal entries of the coefficient matrix as follows:

$$K_J := \mathrm{diag}(a_{1,1}, a_{2,2}, \ldots, a_{n,n}).$$

Under some conditions, it is shown that the Jacobi preconditioning is optimal, or close to optimal, in the sense of reducing the condition number of a matrix A. This was proved by Forsythe and Strauss [36], and van der Sluis [85].

The symmetric Gause-Seidel (SGS) preconditioner uses more information of A than the Jacobi preconditioner. SGS is named after the Gause-Seidel method widely known as one of the stationary iterative methods. The preconditioner is defined as

$$K_{SGS} := (D - L)D^{-1}(D - U),$$

where $A := D - L - U$. If the diagonal elements of A are scaled to all one, then we have simpler preconditioner:

$$K_{SGS} = (I - L)(I - U).$$

In this case, a very efficient implementation can be used. From the above preconditioner, we solve

$$(I - L)^{-1}A(I - U)^{-1}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \quad \tilde{\boldsymbol{x}} = (I - U)\boldsymbol{x}, \quad \tilde{\boldsymbol{b}} = (I - L)^{-1}\boldsymbol{b}.$$

Hence, when we use preconditioned Krylov subspace methods, we need to compute the following matrix-vector multiplication:

$$(I - L)^{-1}A(I - U)^{-1}\boldsymbol{z}.$$

When the cost of the matrix-vector product $A\boldsymbol{z}$ is dominant in one iteration, this usually leads to about double computational cost per iteration. However, recalling $A = I - L - U$, we have

$$(I - L)^{-1}A(I - U)^{-1} \quad = \quad (I - L)^{-1}(I - L - U)(I - U)^{-1}$$

$$
\begin{aligned}
&= (I-L)^{-1}\Big\{(I-L)+(I-U-I)\Big\}(I-U)^{-1} \\
&= (I-L)^{-1}\Big\{(I-L)(I-U)^{-1}+I-(I-U)^{-1}\Big\} \\
&= (I-U)^{-1}+(I-L)^{-1}\Big\{I-(I-U)^{-1}\Big\}.
\end{aligned}
$$

Thus, we obtain

$$
\begin{aligned}
(I-L)^{-1}A(I-U)^{-1}\boldsymbol{z} &= (I-U)^{-1}\boldsymbol{z}+(I-L)^{-1}\Big\{I-(I-U)^{-1}\Big\}\boldsymbol{z} \\
&= \boldsymbol{t}+(I-L)^{-1}(\boldsymbol{z}-\boldsymbol{t}),
\end{aligned}
$$

where $\boldsymbol{t} := (I-U)^{-1}\boldsymbol{z}$. Hence, we see that the cost of the above operation is only about one matrix-vector multiplication. This implementation is one of Eisenstat's tricks [30].

The Symmetric Successive OverRelaxation (SSOR) preconditioner is regarded as an extension of the SGS preconditioner. The preconditioner is defined as

$$
K_{SSOR} := (\tilde{D}-L)\tilde{D}^{-1}(\tilde{D}-U), \quad \tilde{D} = D/\omega.
$$

Note that the choice $\omega = 1$ leads to K_{SGS}. The Eisenstat's trick is then given as follows: from $A = D-L-U$ and $\tilde{A} = (\tilde{D}-L)^{-1}A(\tilde{D}-U)^{-1}$, we have

$$
\begin{aligned}
\tilde{A}\boldsymbol{z} &= (\tilde{D}-L)^{-1}A(\tilde{D}-U)^{-1}\boldsymbol{z} \\
&= (\tilde{D}-L)^{-1}\Big\{(\tilde{D}-L)+(D-2\tilde{D})+(\tilde{D}-U)\Big\}(\tilde{D}-U)^{-1}\boldsymbol{z} \\
&= (\tilde{D}-U)^{-1}\boldsymbol{z}+(\tilde{D}-L)^{-1}(D-2\tilde{D})(\tilde{D}-U)^{-1}\boldsymbol{z}+(\tilde{D}-L)^{-1}\boldsymbol{z} \\
&= \boldsymbol{t}+(\tilde{D}-L)^{-1}\Big\{(D-2\tilde{D})\boldsymbol{t}+\boldsymbol{z}\Big\},
\end{aligned}
$$

where $\boldsymbol{t} = (\tilde{D}-U)^{-1}\boldsymbol{z}$.

C.2 Approximate inverses and polynomial preconditioners

In the previous preconditioners, their performances depend on how close to A the product of the factorized matrices $\tilde{L}\tilde{U}$ is. Here, we describe another criterion for a good preconditioner. That is how close to A^{-1} the preconditioning matrix is. Based on the criterion, Grote and Huckle [50] attempt to minimize the following Frobenius norm:

$$
\min_{K} \|I-AK\|_F,
$$

where $\|A\|_F = \sqrt{\sum_{i,j} a_{i,j}^2}$. Since the above minimization can be written as

$$
\min_{K} \Big\| \Big[\boldsymbol{e}_1 - A\boldsymbol{k}_1, \boldsymbol{e}_2 - A\boldsymbol{k}_2, \ldots, \boldsymbol{e}_N - A\boldsymbol{k}_N\Big] \Big\|_F,
$$

we have N independent least squares problems

$$
\min_{\boldsymbol{k}_i} \|\boldsymbol{e}_i - A\boldsymbol{k}_i\|, \quad i = 1, \ldots, N.
$$

Hence, the construction of this proconditioner is very suitable for parallel computing.

Another outstanding idea for approximate inverses was given by Benzi and Tůma [9]. This idea is finding nonsingular matrices V and W such that

$$W^{\mathrm{H}}AV = D. \tag{C.1}$$

Then, it follows from $(W^{\mathrm{H}}AV)^{-1} = V^{-1}A^{-1}W^{-H} = D^{-1}$ that the inverse of the matrix A is given as

$$A^{-1} = VD^{-1}W^{\mathrm{H}}.$$

Since the equation (C.1) implies $(\boldsymbol{w}_i, A\boldsymbol{v}_j) = 0$ for $i \neq j$, V and W are computed by an A-biorthogonalization process. In the following algorithm, $\boldsymbol{a}_i$ and $\boldsymbol{c}_i$ denote the ith column of A and A^{H} respectively.

Algorithm C.1: Approximate inverse (AINV)

set $\boldsymbol{w}_i^{(0)} = \boldsymbol{v}_i^{(0)} = \boldsymbol{e}_i,\ 1 \le i \le N,$
for $i = 1, \ldots, N$ do:
  for $j = i, \ldots, N$ do:
    $p_j^{(i-1)} = \boldsymbol{a}_i^{\mathrm{H}}\boldsymbol{v}_j^{(i-1)}, \quad q_j^{(i-1)} = \boldsymbol{c}_i^{\mathrm{H}}\boldsymbol{w}_j^{(i-1)},$
  end
  if $i = N$ exit
  for $j = i+1, \ldots, N$
    $\boldsymbol{v}_j^{(i)} = \boldsymbol{v}_j^{(i-1)} - \left(\dfrac{p_j^{(i-1)}}{p_i^{(i-1)}}\right)\boldsymbol{v}_i^{(i-1)}, \quad \boldsymbol{w}_j^{(i)} = \boldsymbol{w}_j^{(i-1)} - \left(\dfrac{q_j^{(i-1)}}{q_i^{(i-1)}}\right)\boldsymbol{w}_i^{(i-1)},$
    $v_{k,j}^{(i)} = 0$ if $|\boldsymbol{v}_{k,j}^{(i)}| < \mathrm{Tol},\ 1 \le k \le N, \quad w_{k,j}^{(i)} = 0$ if $|\boldsymbol{w}_{k,j}^{(i)}| < \mathrm{Tol},\ 1 \le k \le N,$
  end
end
set $\boldsymbol{z}_i = \boldsymbol{z}_i^{(i-1)},\ \boldsymbol{w}_i = \boldsymbol{w}_i^{(i-1)},\ p_i = p_i^{(i-1)},\ \ 1 \le i \le N,$
set $V = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N],\ V = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_N],\ D = \mathrm{diag}(p_1, p_2, \ldots, p_N).$

From the above algorithm, we see that it uses a MGS-style A-biorthogonalization process and is used for a general non-Hermitian matrix. For Hermitian and normal matrices, the corresponding approximate inverse preconditioners have been developed. See [7] and [11].

Polynomial preconditioner

We have described two types of preconditioners. One of which is approximating the coefficient matrix and the preconditioner can be easily solved. The other one is approximating the inverse of the coefficient matrix. Here, we describe another approach that is closely related to sparse approximate inverses. Since this approach is based on matrix polynomials, it is referred to as polynomial preconditioner.

One of the most well-known polynomial preconditioners is a Neumann expansion and a Euler expansion. Let A be of the form $I - B$ with $\rho(B) < 1$. Then, we can write the inverse matrix of A as

$$A^{-1} \;=\; \sum_{i=0}^{\infty} B^i \qquad \text{(Neumann expansion)}$$

$$= \prod_{i=0}^{\infty}(I + B^{2^i}). \quad \text{(Euler expansion)}$$

Hence, from the above expansions, we can readily obtain and moreover control an approximate inverse of A by using the low order terms of a Neumann (or Euler) expansion. The results were reported in [24].

C.3 Reorderings for preconditioners

In the previous subsections, we have considered some preconditioning techniques for solving original systems effectively. On the other hand, it is natural to find effective reorderings to improving the performance of preconditioners, *e.g.*, apply Krylov subspace methods to the following systems with a reordering matrix P:

$$K^{-1}PAP^{\mathrm{T}}\tilde{x} = \tilde{\boldsymbol{b}},$$

where $\tilde{x} = P\boldsymbol{x}$, $\tilde{\boldsymbol{b}} = K^{-1}P\boldsymbol{b}$. From this idea, we can use many reordering techniques for the efficient implementation of the LU decomposition such as Cuthill-Mckee (CM) [19], Reverse Cuthill-Mckee (RCM) [42], Nested Dissection (ND) [43], and Minimum Degree (MD) [44]. For description of algorithms see, *e.g.*, [25, 58, 70]. CM and RCM decrease the bandwidth of a matrix, MD reduces the number of fill-in of an original matrix, and ND generates an approximate block diagonal matrix and is suitable for parallel computing.

For symmetric definite case (preconditioned CG), Duff and Meurant gave important results on the effects of reorderings that reordering techniques for direct solvers did not enhance the quality of preconditioners [27].

For nonsymmetric case, Saad experimentally showed that MD and ND reorderings before $ILUT$ preconditioning gave no advantage over the original systems and only RCM was the most likely to yield an improvement [70, p.334]. On the other hand, Benzi *et al.* [8] studied the effects of reorderings on ILU-type preconditioner and gave the result that RCM dramatically enhanced the preconditioner in the case where the coefficient matrix is highly nonsymmetric. Similar results of the effects on approximate inverse preconditioners are also discussed in [10].

Acknowledgements

First of all, I would like to give my great thanks to my adviser Prof. S.-L. Zhang at Nagoya University. He introduced me into this very interesting field of Krylov subspace methods and offered many opportunities of my research talks at various conferences. Needless to say, were it not for his continuous support, I could not make my research so smooth.

I am grateful to Prof. M. Sugihara at the University of Tokyo for the careful reading and some important comments on a paper of the Bi-CR method which is one of the main topics of the present thesis. I am also grateful to Prof. M. H. Gutknecht at ETH Zürich for providing some references and fruitful discussion on Bi-CR.

I am thankful to Prof. K. Abe at Gifu Shotoku Gakuen University and Prof. C.-H. Jin at the University of Tokushima. They gave me some important comments on the SCGS method.

I would like to thank Prof. S. Fujino at Kyushu University. He suggested to me that I should investigate the dependency of a shadow vector on the convergence behavior of product-type methods, and introduced me to many researchers for obtaining complex symmetric matrices. I would like to thank Prof. T. Sakurai at the University of Tsukuba for discussion on eigenvalue problems. He informed me of the strong need for solving complex symmetric linear systems efficiently. I appreciate discussion with Prof. Y. Yamamoto at Nagoya University and the use of his work station with Opteron processor. I am also thankful to Dr. T. Hoshi at the University of Tokyo for providing complex symmetric matrices arising from large-scale electronic structure theory.

I am grateful to Prof. T. Fujiwara, Prof. K. Murota, and Prof. Y. Hatsugai at the University of Tokyo for the careful reading of the thesis and useful comments that improved its quality.

Finally, I wish to thank my loving wife, Li Yuan, for her day-to-day support and thank my parents, Tsutao and Chiaki, for financial support.

Bibliography

[1] W. E. Arnoldi, The principle of minimized iteration in the solution of the matrix eigenvalue problem, Quart. Appl. Math., 9(1951), pp. 17-29.

[2] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, A taxonomy for conjugate gradient methods, SIAM J. Numer. Anal., 27(1990), pp. 1542-1568.

[3] Z. Bai, D. Day, J. Demmel, and J. Dongarra, A test matrix collection for non-Hermitian eigenvalue problems, Technical Report CS-97-355, Department of Computer Science, University of Tennessee, Knoxville, TN, March, 1997.

[4] R. E. Bank and T. F. Chan, An analysis of the composite step biconjugate gradient method, Numer. Math., 66(1994), pp. 295-319.

[5] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, Templates for the solution of linear systems: building blocks for iterative methods, 2nd ed., SIAM, Philadelphia, PA, 1994.

[6] A. Bayliss, C. I. Goldstein, and E. Turkel, An iterative method for the Helmholtz equation, J. Comput. Phys., 49(1983), pp. 443-457.

[7] M. Benzi, C. D. Meyer, and M. Tůma, A sparse approximate inverse preconditioner for the conjugate gradient method, SIAM J. Sci. Comput., 17(1996), pp. 1135-1149.

[8] M. Benzi, D. B. Szyld, and A. van Duin, Orderings for incomplete factorization preconditioning of nonsymmetric problems, SIAM J. Sci. Comput., 20(1999), pp. 1652-1670.

[9] M. Benzi and M. Tůma, A sparse approximate inverse preconditioner for nonsymmetric linear systems, SIAM J. Sci. Comput., 19(1998), pp. 968-994.

[10] M. Benzi and M. Tůma, Orderings for factorized sparse approximate inverse preconditioners, SIAM J. Sci. Comput., 21(2000), pp. 1851-1868.

[11] M. Benzi and M. Tůma, A robust preconditioner with low memory requirements for large sparse least squares problems, SIAM J. Sci. Comput., 25(2003), pp. 499-512.

[12] Å. Björck, Solving linear least squares problems by Gram-Schmidt orthogonalization, BIT, 7(1967), pp. 1-21.

[13] C. G. Broyden and M. A. Boschetti, A comparison of three basic conjugate direction methods, Num. Lin. Alg. Appl., 3(1996), pp. 473-489.

[14] A. Bunse-Gerstner and R. Stöver, On a conjugate gradient-type method for solving complex symmetric linear systems, Lin. Alg. Appl., 287(1999), pp. 105-123.

[15] T. F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, and C. H. Tong, A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems, SIAM J. Sci. Comput., 15(1994), pp. 338-347.

[16] M. Clemens and T. Weiland, Comparison of Krylov-type methods for complex linear systems applied to high-voltage problems, IEEE Trans. Mag., 34:5(1998), pp. 3335-3338.

[17] J. Collum and A. Greenbaum, Relations between Galerkin and norm-minimizing iterative methods for solving linear systems, SIAM J. Matrix Anal. Appl., 17(1996), pp. 223-247.

[18] E. J. Craig, The N-step iteration procedures, J. Math. and Phys., 34(1955), pp. 64-73.

[19] E. Cuthill and J. McKee, Reducing the bandwidth of sparse symmetric matrices, in: Proceedings of the ACM National Conference, Association for Computing Machinery, New York, NY, (1969), pp. 157–172.

[20] J. W. Daniel, W. B. Gragg, L. Kaufmann, and G. W. Stewart, Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization, Math. Comp., 30(1976), pp. 772-795.

[21] T. Davis, University of Florida sparse matrix collection, NA Digest, 97(23), June 7, 1997.

[22] T. A. Davis and I. S. Duff, A combined unifrontal/multifrontal method for unsymmetric sparse matrices, Technical Report TR-95-020, Computer and Information Sciences Department, University of Florida, Gainesville, 1995.

[23] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, A supernodal approach to sparse partial pivoting, SIAM J. Matrix Anal. Appl., 20(1999), pp. 720-755.

[24] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, Approximating the inverse of a matrix for use in iterative algorithms on vector processors, Computing, 22(1979), pp. 257-268.

[25] I. S. Duff, A. M. Erisman and J. K. Reid, Direct methods for sparse matrices, Oxford University Press, Oxford, 1986.

[26] I. S. Duff, R. G. Grimes, and J. G. Lewis, User's guide for the Harwell-Boeing sparse matrix collection, Technical Report RAL-92-086, Rutherford Appleton Laboratory, Chilton, UK, 1992.

[27] I. S. Duff and G. A. Meurant, The effect of ordering on preconditioned conjugate gradients, BIT, 29(1989), pp. 635–657.

[28] I. S. Duff and J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Trans. Math. Software, 9(1983), pp. 302-325.

[29] V. Eijkhout, LAPACK working note 50: distributed sparse data structures for linear algebra operations, Tech. Rep. CS 92-169, Computer Science Department, University of Tennessee, Knoxville, TN, 1992.

[30] S. C. Eisenstat, Efficient implementation of a class of preconditioned conjugate gradient methods, SIAM J. Sci. Stat. Comput., 2(1981), pp. 1-4.

[31] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, SIAM J. Numer. Anal., 20(1983), pp. 345-357.

[32] J. Erhel, K. Burrage, and B. Pohl, Restarted GMRES preconditioned by deflation, J. Comput. Appl. Math., 69(1996), pp. 303-318.

[33] V. Faber and T. A. Manteuffel, Necessary and sufficient conditions for the existence of a conjugate method, SIAM J. Numer. Anal., 21(1984), pp. 352-362.

[34] V. Faber and T. A. Manteuffel, Orthogonal error methods, SIAM J, Numer. Anal., 24(1987), pp. 170-187.

[35] R. Fletcher, Conjugate gradient methods for indefinite systems, Lecture Notes in Mathematics 506, 1976, pp. 73-89.

[36] G. Forsythe and E. Strauss, On best conditioned matrices, Proc. Amer. Math. Soc., 6(1955), pp. 340-345.

[37] S. Frankel, Convergence rates of iterative treatments of partial differential equations, MTAC, 4(1950), pp. 65-75.

[38] R. W. Freund, Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices, SIAM J. Sci. Stat. Comput., 13(1992), pp. 425-448.

[39] R. W. Freund, A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems, SIAM J. Sci. Comput., 14(1993), pp. 470-482.

[40] R. W. Freund, G. H. Golub, and N. M. Nachtigal, Iterative solution of linear systems, Acta Numerica, 1(1992), pp. 57-100.

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

[42] A. George, Computer implementation of the finite element method, Tech. Rep. 208, Department of 20 Computer Science, Stanford University, Stanford, CA, 1971.

[43] A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal., 10(1973), pp. 345–363.

[44] J. A. George and J. W. Liu, The evolution of the minimum degree algorithm, SIAM Rev., 31(1989), pp. 1-19.

[45] L. Giraud, J. Langou, M. Rozložník, and J. van der Eshof, Rounding error analysis of the classical Gram-Schmidt orthogonalization process, Numer. Math., 101(2005), pp. 87-100.

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

[47] G. Golub and H. A. van der Vorst, Closer to the solution: iterative linear solvers, the State of Art in Numerical Analysis (edited by I. S. Duff and G. A. Watson), Clarendon Press, 1997, pp. 63-92.

[48] G. Golub and C. F. Van Loan, Matrix computations, 3rd ed., The Johns Hopkins University Press, Baltimore, 1996.

[49] A. Greenbaum, Iterative methods for solving linear systems, SIAM, Philaledelphia 1997.

[50] M. Grote and T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM J. Sci. Comput., 18(1997), pp. 838-853.

[51] M. H. Gutknecht, Variants of BiCGSTAB for matrices with complex spectrum, SIAM J. Sci. Comput., 14(1993), pp. 1020-1033.

[52] M. H. Gutknecht, Lanczos-type solvers for nonsymmetric linear systems of equations, Acta Numerica, 6(1997), pp. 271-397.

[53] M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Standards, 49(1952), pp. 409-436.

[54] C. P. Jackson and P. C. Robinson, A numerical study of various algorithms related to the preconditioned conjugate gradient method, Int. J. Num. Meth. Eng., 21(1985), pp. 1315-1338.

[55] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Nat. Bur. Standards, 45(1950), pp. 255-282.

[56] C. Lanczos, Solution of systems of linear equations by minimized iterations, J. Res. Nat. Bur. Standards, 49(1952), pp. 33-53.

[57] C. Lanczos, Applied analysis, Prentice-Hall, Englewood Cliffs, NJ, 1956.

[58] J. W. Liu, Modification of the minimum-degree algorithm by multiple elimination, ACM Trans. Math. Software, 11:2(1985), pp. 141-153.

[59] J. A. Meijerink and H. A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Math. Comp., 31(1977), pp. 148-162.

[60] R. B. Morgan, GMRES with deflated restarting, SIAM J. Sci. Comput., 24(2002), pp. 20-37.

[61] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, How fast are nonsymmetric matrix iterations?, SIAM J. Matrix Anal. Appl., 13(1992), pp. 778-795.

[62] C. Paige and M. Saunders, Solution of sparse indefinite systems of linear equations, SIAM J. Numer. Anal., 12(1975), pp. 617-629.

[63] C. Paige and M. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares, ACM Trans. Math. Software, 8(1982), pp. 43-71.

[64] B. N. Parlett, D. R. Taylor, and Z. A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, Math. Comput., 44(1985), pp. 105-124.

[65] C. Pommerell, Solution of large unsymmetric systems of linear equations, vol. 17 of series in micro-electronics, volume 17, Hartung-Gorre Verlag, Konstantz, 1992.

[66] Y. Saad, Krylov subspace methods for solving large unsymmetric linear systems, Math. Comp., 37(1981), pp. 105-126.

[67] Y. Saad, SPARSKIT: a basic tool kit for sparse matrix computation, Tech. Rep. CSRD TR 1029, CSRD, University of Illinois, Urbana, IL, 1990.

[68] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM J. Sci. Comput., 14(1993), pp. 461-469.

[69] Y. Saad, ILUT: a dual threshold incomplete LU factorization, Num. Lin. Alg. Appl., 4(1994), pp. 387-402.

[70] Y. Saad, Iterative methods for sparse linear systems, 2nd ed., SIAM, Philadelphia, PA, 2003.

[71] Y. Saad and M. H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput., 7(1986), pp. 856-869.

[72] Y. Saad and H. A. van der Vorst, Iterative solution of linear systems in the 20th century, J. Comput. Appl. Math., 123(2000), pp. 1-33.

[73] Y. Saad and K. Wu, DQGMRES: a direct quasi-minimal residual algorithm based on incomplete orthogonalization, Num. Lin. Alg. Appl., 3(1996), pp. 329-343.

[74] T. Sogabe, C.-H. Jin, K. Abe, and S.-L. Zhang, On an improvement of the CGS method, Trans. JSIAM, 14:1(2004), pp. 1-12. (in Japanese)

[75] T. Sogabe, M. Sugihara, and S.-L. Zhang, An extension of the conjugate residual method for solving nonsymmetric linear systems, Trans. JSIAM, 15:3(2005), pp. 445-459. (in Japanese)

[76] T. Sogabe, M. Sugihara, and S.-L. Zhang, An extension of the conjugate residual method to nonsymmetric linear systems, manuscript, 2005.

[77] T. Sogabe and S.-L. Zhang, On product-type methods based on the Bi-CR method, RIMS Kokyuroku 1362, 4(2004), pp. 22-30. (in Japanese)

[78] T. Sogabe and S.-L. Zhang, A COCR method for solving complex symmetric linear systems, J. Comput. Appl. Math., to appear.

[79] G. L. G. Sleijpen and D. R. Fokkema, BICGSTAB(ℓ) for linear equations involving unsymmetric matrices with complex spectrum, Elec. Trans. Numer. Anal., 1(1993), pp. 11-32.

[80] G. L. G. Sleijpen and H. A. van der Vorst, An overview of approaches for the stable computation of hybrid BiCG method, Appl. Numer. Math., 19(1995), pp. 235-254.

[81] P. Sonneveld, CGS: a fast Lanczos-type solver for nonsymmetric linear systems, SIAM J. Sci. Stat. Comput., 10(1989), pp. 36-52.

[82] E. Stiefel, Relaxationsmethoden bester Strategie zur Lösung linearer Gleichungssystems, Comment. Math. Helv., 29(1955), pp. 157-179.

[83] D. R. Taylor, Analysis of the look ahead Lanczos algorithm, Ph.D. Dissertation, University of California, Berkley, 1982.

[84] C. H. Tong, A family of quasi-minimal residual methods for nonsymmetric linear systems, SIAM J. Sci. Comput., 15(1994), pp. 89-105.

[85] A. van der Sluis, Condition numbers and equilibration of matrices, Numer. Math., 14(1969), pp. 14-23.

[86] 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. Stat. Comput., 13(1992), pp. 631-644.

[87] H. A. van der Vorst, Iterative Krylov methods for large linear systems, Cambridge University Press, Cambridge, 2003.

[88] H. A. van der Vorst and J. B. M. Melissen, A Petrov-Galerkin type method for solving $A\boldsymbol{x} = \boldsymbol{b}$, where A is symmetric complex, IEEE Trans. Mag., 26(1990), pp. 706-708.

[89] H. A. van der Vorst and C. Vuik, GMRESR: A family of nested GMRES methods, Num. Lin. Alg. Appl., 1(1994), pp. 369-386.

[90] R. S. Varga, Matrix iterative analysis, 2nd ed., Springer-Verlag, New York, 2000.

[91] P. K. W. Vinsome, ORTHOMIN: an iterative method for solving sparse sets of simultaneous linear equations, In Proc. Fourth Symposium on Reservoir Simulation, Society of Petroleum of Engineering of AIME, (1976), pp. 149-159.

[92] H. F. Walker, Implementation of the GMRES method using Householder transformation, SIAM J. Sci. Stat. Comput., 9(1988), pp. 152-163.

[93] R. Weiss, Properties of generalized conjugate gradient methods, Num. Lin. Alg. Appl., 1(1994), pp. 45-63.

[94] R. Weiss, Error-minimizing Krylov subspace methods, SIAM J. Sci. Comput., 15(1994), pp. 511-527.

[95] P. Wesseling and P. Sonneveld, Numerical experiments with a multiple grid and a preconditioned Lanczos type method, in: R. Rauntmann(ed.), Approximation methods for Navier-Stokes problems, Lecture Notes in Math., 1980, pp. 543-562.

[96] D. M. Young, Iterative methods for solving partial differential equations of ellipic type, Ph.D. thesis, Harvard University, Cambridge (MA, USA), 1950.

[97] S.-L. Zhang, GPBi-CG: generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems, SIAM J. Sci. Comput., 18(1997), pp. 537-551.