Springer Series in Computational Mathematics 60

Tomohiro Sogabe

Krylov Subspace Methods for Linear Systems

Principles of Algorithms

Springer

Springer Series in Computational Mathematics

Volume 60

Tomohiro Sogabe

Krylov Subspace Methods for Linear Systems

Principles of Algorithms

Springer

Tomohiro Sogabe
Department of Applied Physics
Nagoya University
Nagoya, Japan

ISSN 0179-3632 ISSN 2198-3712 (electronic)
Springer Series in Computational Mathematics
ISBN 978-981-19-8531-7 ISBN 978-981-19-8532-4 (eBook)
https://doi.org/10.1007/978-981-19-8532-4

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

In many fields of scientific computing and data science, we frequently face the problem of solving large and sparse linear systems of the form $A\boldsymbol{x} = \boldsymbol{b}$, which is one of the most time-consuming parts of all computations. From this fact, many researchers have devoted themselves to developing efficient numerical algorithms for solving the linear systems, and Krylov subspace methods are nowadays popular numerical algorithms and are known as one of the top ten algorithms of the twentieth century, others including fast Fourier transform and Quick Sort [39]. Though the basic theory was established in the twentieth century, Krylov subspace methods have been developed by mathematicians, engineers, physicists, and many others.

There are many excellent books on Krylov subspace methods, including those by:

- Owe Axelsson, 1994 [10],
- Richard Barrett et al., 1994 [18],
- Wolfgang Hackbusch, 1994 [92],
- Are Magnus Bruaset, 1995 [29],
- Rüdiger Weiss, 1996 [203],
- Anne Greenbaum, 1997 [81],
- Yousef Saad, 2003 [151],
- Henk A. van der Vorst, 2003 [196],
- Jörg Liesen and Zdeněk Strakoš, 2012 [122],
- Gérard Meurant and Jurjen Duintjer Tebbens, 2020 [129].

In [129], detailed historical notes of Krylov subspace methods are described, which form a masterpiece (around 700 pages) of Krylov subspace methods for non-Hermitian linear systems. The features of this book are listed as follows:

(1) Many applications of linear systems from computational science and data science;
(2) Krylov subspace methods for complex symmetric linear systems such as the COCG method and the COCR method;
(3) Krylov subspace methods for non-Hermitian linear systems such as the BiCR method, the GPBiCG method, and the (block) IDR(s) method;

(4) Krylov subspace methods for shifted linear systems such as the shifted IDR(s) method;
(5) Matrix functions as applications of shifted linear systems.

Feature (1) corresponds to Chap. 2, and linear systems are derived from various applications: partial differential equations (finite difference discretization methods and the finite element method); computational physics: condensed matter physics (computation of Green's function) and lattice quantum chromodynamics (Wilson fermion matrix); machine learning (least-squares problems); matrix equations (Sylvester-type matrix equations); optimization (Hessian matrix over Euclidean space and Riemannian manifold using tensor computation notations).

Features (2) and (3) correspond to Chap. 3. In this chapter, Krylov subspace methods are classified into three groups: Hermitian linear systems, complex symmetric linear systems, and non-Hermitian linear systems. For Hermitian linear systems, the CG method is derived from the matrix form of the Lanczos process, the CR method is derived from the CG method, and the MINRES method is derived from the Lanczos process. For complex symmetric linear systems, the COCG method, the COCR method, and the QMR_SYM method are described as extensions of the CG method, the CR method, and the MINRES method, respectively. For non-Hermitian linear systems, the BiCG method, the BiCR method, and the QMR method are described as extensions of the CG method, the CR method, and the MINRES method, respectively. The detailed derivations of the GPBiCG method and the (block) IDR(s) method, one of the features of this book, are described in this chapter. In addition, some preconditioning techniques are briefly described.

Feature (4) corresponds to Chap. 4. In this chapter, Krylov subspace methods for shifted linear systems are classified into three groups: Hermitian, complex symmetric, and non-Hermitian linear systems. The detailed derivations of these algorithms are described systematically.

Feature (5) corresponds to Chap. 5. If one needs a large matrix function, then Krylov subspace methods and Krylov subspace methods for shifted linear systems are methods of choice since these algorithms can produce any element of the matrix function. The definitions of matrix functions and well-known algorithms for matrix functions are also described.

An additional feature of this book is that there are no numerical experiments except some typical numerical examples for further understanding the convergence behavior of Krylov subspace methods. The convergence of Krylov subspace methods depends highly on the coefficient matrix, and the best algorithm changes if the coefficient matrix changes. So, if the reader wants to solve linear systems, I recommend the reader to apply several Krylov subspace methods (including the BiCGSTAB method and the GMRES method) to their problem and choose the best one among them.

This book is suitable for anyone who studied linear algebra and needs to solve large and sparse linear systems. I hope this book is helpful for the reader to understand the principles and properties of Krylov subspace methods and to correctly use Krylov subspace methods to solve their problems.

Nagoya, Japan Tomohiro Sogabe
September 2022

Acknowledgements

I would like to thank Prof. Dr. Lei Du at Dalian University of Technology for kindly sending me the LaTeX code of his block IDR(s) method. I appreciate Masayuki Nakamura's work. I could raise the priority of writing this book from his timely reminders, together with his kind seasonal greetings. I thank Lokeshwaran Manickavasagam and Poojitha Ravichandran for their help with the production of this book. I am grateful to three anonymous reviewers for their careful reading of the manuscript and highly constructive comments that enhanced the quality of the manuscript. I would like to express my sincere thanks to three editors from the Springer Series in Computational Mathematics (SSCM) editorial board: Prof. Dr. Randolph E. Bank, Prof. Dr. Wolfgang Hackbusch, Prof. Dr. Josef Stoer, and Prof. Dr. Harry Yserentant for providing the author with fruitful comments from broad and distinguished professional viewpoints.

Finally, I appreciate Prof. Dr. Shao-Liang Zhang who introduced me to the field of Krylov subspace methods when I was a Ph.D. student at the University of Tokyo about 20 years ago, and I wish to thank my loving wife, Yuan Li, for her day-to-day support and her kindness. She recommended me to write this book on weekends, and she was taking care of two active sons: Kazuma and Takuma, and her old father.

Contents

Chapter 1
Introduction to Numerical Methods for Solving Linear Systems

Abstract Numerical methods for solving linear systems are classified into two groups: direct methods and iterative methods. Direct methods solve linear systems within a finite number of arithmetic operations, and the best-known direct method is the LU decomposition. Iterative methods produce a sequence of approximate solutions, and the iterative methods are roughly classified into stationary iterative methods and Krylov subspace methods. Multigrid methods also fall into an important class of iterative methods. This chapter aims to describe the principles of the direct methods, the stationary iterative methods, and a brief introduction to the theory of Krylov subspace methods. A brief explanation of multigrid methods is also given.

1.1 Linear Systems

In this book, we consider linear systems of the following form:

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

where $A \in \mathbb{C}^{N\times N}$, $\boldsymbol{b} \in \mathbb{C}$, and the coefficient matrix A is assumed to be nonsingular, i.e., there exists the inverse of A.

Let $\tilde{\boldsymbol{x}}$ be an approximate solution of linear systems (1.1). Then a quantitative way to know the distance between the solution $\boldsymbol{x}$ and the approximate solution $\tilde{\boldsymbol{x}}$ is to compute a vector norm of $\boldsymbol{x} - \tilde{\boldsymbol{x}}$. Further, a quantitative way to measure the distance between two matrices A, B is given by a matrix norm of $A - B$. The definitions of a vector norm and a matrix norm are described in the next two subsections. The explanations are based on [170], and for the details of numerical linear algebra, see [45, 79, 107, 191].

T. Sogabe, *Krylov Subspace Methods for Linear Systems*,
Springer Series in Computational Mathematics 60,
https://doi.org/10.1007/978-981-19-8532-4_1

1.1.1 Vector Norm

The length of the vector $\boldsymbol{x} := (x_1, x_2)^\top \in \mathbb{R}^2$ is given by $(x_1^2 + x_2^2)^{1/2}$. The *norm* is the extension of the notion of the "length" and, the map $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ is called a *vector norm* if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$, the map satisfies the following rules:[1]

(Nv1) $\|\boldsymbol{x}\| \geq 0;\quad \|\boldsymbol{x}\| = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{0}$, (Positivity)
(Nv2) $\|\alpha\boldsymbol{x}\| = |\alpha|\ \|\boldsymbol{x}\|$, (Homogeneity)
(Nv3) $\|\boldsymbol{x}+\boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$. (Triangle inequality)

For example, a p-norm ($p \geq 1$) is defined by

$$\|\boldsymbol{x}\|_p := (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}.$$

For $p = 1, 2, \infty$, the vector p-norm is written as

- $\|\boldsymbol{x}\|_1 = \sum_{i=1}^n |x_i|$,
- $\|\boldsymbol{x}\|_2 = (\sum_{i=1}^n |x_i|^2)^{\frac{1}{2}}\ (= \sqrt{\boldsymbol{x}^\top\boldsymbol{x}})$,
- $\|\boldsymbol{x}\|_\infty = \max\limits_{1\leq i\leq n} |x_i|$.

Let A be a symmetric positive definite matrix, i.e., A is symmetric and all the eigenvalues are positive. Then $\|\boldsymbol{x}\|_A := \boldsymbol{x}^\top A\boldsymbol{x}$ is called an A-norm of $\boldsymbol{x}$; this will be used in Section 3.1.1.

Let $\tilde{\boldsymbol{x}}$ be an approximate solution of linear systems (1.1). Then $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_p$ is the distance between the exact solution and approximate solution of (1.1).

Throughout this book, unless otherwise stated, the symbol $\|\cdot\|$ means 2-norm, i.e., $\|\cdot\| = \|\cdot\|_2$.

1.1.2 Matrix Norm

Similar to the definition of vector norm, *matrix norm* is defined as follows: a map $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ is called a matrix norm if for all $A, B \in \mathbb{R}^{m\times n}$, $\alpha \in \mathbb{R}$ the map satisfies the following three items:

(Nm1) $\|A\| \geq 0;\quad \|A\| = 0 \Leftrightarrow A = O$, ($O$: zero matrix) (Positivity)
(Nm2) $\|\alpha A\| = |\alpha|\|A\|$, (Homogeneity)
(Nm3) $\|A + B\| \leq \|A\| + \|B\|$. (Triangle inequality)

In addition, if the matrix norm satisfies

(Nm4) $\|AB\| \leq \|A\|\|B\|,\quad A \in \mathbb{R}^{m\times n},\ B \in \mathbb{R}^{n\times q}$,

[1] In general, given a vector space V, a map $\|\cdot\| : \mathbb{V} \to \mathbb{R}$ is called a *norm* if the map satisfies (Nv1)–(Nv3) for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$, $\alpha \in \mathbb{K}$ ($\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$).

then the matrix norm is called *submultiplicative*. Note that in some books the map satisfying (Nm1)–(Nm4) is also called a matrix norm. As an example of a submultiplicative matrix norm, the following *subordinate matrix norm* is widely used:[2]

$$\|A\|_p := \sup_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|A\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}. \tag{1.2}$$

If $p = 1, 2, \infty$, $\|A\|_p$ is given as follows:

- $\|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$,
- $\|A\|_2 = \sqrt{\text{maximum eigenvalue of } A^\top A}$,
- $\|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$.

A *Frobenius norm* is also submultiplicative:

- $\|A\|_\mathrm{F} := \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$.

Let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ be the column vectors of matrix $A \in \mathbb{R}^{m\times n}$, and let $\boldsymbol{a} = [\boldsymbol{a}_1^\top, \ldots, \boldsymbol{a}_n^\top]^\top$, i.e., all the columns are connected to get one long vector $\boldsymbol{a}$. Then, the Frobenius norm of A equals the 2-norm of $\boldsymbol{a}$, i.e., $\|A\|_\mathrm{F} = \|\boldsymbol{a}\|_2$.

The *max norm* of the form

$$\|A\|_{\max} := \max_{i,j} |a_{ij}|$$

is not submultiplicative. For example, consider a 2-by-2 matrix A with all the elements being 1. Then $\|AA\|_{\max} = \|A^2\|_{\max} = 2$ and $\|A\|_{\max}\|A\|_{\max} = 1$. Thus the max norm does not hold (Nm4) because $\|AA\|_{\max} > \|A\|_{\max}\|A\|_{\max}$.

Some properties of the matrix norm are given next.

Theorem 1.1 *The following properties hold true:*

(1) $\|A\boldsymbol{x}\|_p \leq \|A\|_p \|\boldsymbol{x}\|_p, \quad \|A\boldsymbol{x}\|_2 \leq \|A\|_\mathrm{F} \|\boldsymbol{x}\|_2,$
(2) $\|AB\|_p \leq \|A\|_p \|B\|_p, \quad \|AB\|_\mathrm{F} \leq \|A\|_\mathrm{F} \|B\|_\mathrm{F},$
(3) $\|QA\tilde{Q}\|_2 = \|A\|_2, \quad \|QA\tilde{Q}\|_\mathrm{F} = \|A\|_\mathrm{F}.$

Here, Q and $\tilde{Q}$ are orthogonal matrices.[3]

Theorem 1.2 *Let A be an $n \times n$ matrix. Then,*

(1) $n^{-1/2}\|A\|_2 \leq \|A\|_1 \leq n^{1/2}\|A\|_2,$
(2) $n^{-1/2}\|A\|_2 \leq \|A\|_\infty \leq n^{1/2}\|A\|_2,$
(3) $n^{-1}\|A\|_\infty \leq \|A\|_1 \leq n\|A\|_\infty,$
(4) $\|A\|_1 \leq \|A\|_\mathrm{F} \leq n^{1/2}\|A\|_2.$

[2] "sup" can be replaced with "max".

[3] A real square matrix Q is called an orthogonal matrix if $Q^\top Q = I$, where I is the identity matrix.

1.2 Condition Number

Consider $A\boldsymbol{x} = \boldsymbol{b}$; if the matrix A and right-hand side vector $\boldsymbol{b}$ are slightly changed, then by how much will the solution be changed? To answer this question, let ΔA and $\Delta\boldsymbol{b}$ be perturbations of A and $\boldsymbol{b}$. Then the change of the solution denoted by $\Delta\boldsymbol{x}$ satisfies the following relation:

$$(A + \Delta A)(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b}.$$

Using $A\boldsymbol{x} = \boldsymbol{b}$ gives

$$\Delta\boldsymbol{x} = A^{-1}[-\Delta A(\boldsymbol{x} + \Delta\boldsymbol{x}) + \Delta\boldsymbol{b}].$$

Using Theorem 1.1-(1) and the triangle inequality (Nv3), the p-norm of $\boldsymbol{x}$ can be evaluated by

$$\|\Delta\boldsymbol{x}\|_p \le \|A^{-1}\|_p(\|\Delta A\|_p\|\boldsymbol{x} + \Delta\boldsymbol{x}\|_p + \|\Delta\boldsymbol{b}\|_p).$$

Thus, we have

$$\frac{\|\Delta\boldsymbol{x}\|_p}{\|\boldsymbol{x} + \Delta\boldsymbol{x}\|_p} \le \|A\|_p\|A^{-1}\|_p\left(\frac{\|\Delta A\|_p}{\|A\|_p} + \frac{\|\Delta\boldsymbol{b}\|_p}{\|A\|_p\|\boldsymbol{x} + \Delta\boldsymbol{x}\|_p}\right). \tag{1.3}$$

Here,

$$\kappa_p(A) := \|A\|_p\|A^{-1}\|_p \tag{1.4}$$

is called the *p-norm condition number*. Inequality (1.3) implies that if the condition number is large, the computed approximate solution may be far from the exact solution. Note that this fact does not depend on what algorithms are used for solving linear systems.

If the condition number is large, then the linear systems are called *ill-conditioned*. We will see in Chap. 3 that the condition number also affects the speed of convergence of Krylov subspace methods.

In this book, we often use a 2-norm condition number $\kappa_2(A)$ that will be simply denoted by κ or $\mathrm{cond}(A)$.

1.3 Direct Methods

For simplicity of presentation, we only consider the case $A\boldsymbol{x} = \boldsymbol{b}$ where $A \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$. In Sect. 1.3.1, the LU decomposition is introduced to solve linear systems, and then in Sect. 1.3.2 the LU decomposition with pivoting techniques is explained, which is numerically stable and widely used in practice. A way to improve the accuracy of the approximate solution is described in Sect. 1.3.3.

1.3.1 LU Decomposition

Let A be a square matrix. Then the *LU decomposition* of A is achieved by decomposing A into the multiplication of a *lower triangular matrix* L and an *upper triangular matrix* U of the form

$$A = LU, \tag{1.5}$$

where

$$L = \begin{pmatrix} l_{11} & & \\ \vdots & \ddots & \\ l_{n1} & \cdots & l_{nn} \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & \cdots & u_{1n} \\ & \ddots & \vdots \\ & & u_{nn} \end{pmatrix}.$$

The blanks in matrices L and U mean that all the corresponding elements are zero.

If the diagonal elements of L or U are one, then the LU decomposition is uniquely determined. The LU decomposition with $l_{ii} = 1$ for all i is referred to as Doolittle's method, and the one with $u_{ii} = 1$ for all i is referred to as Crout's method. Once we obtain the LU decomposition of A, it is easy to obtain the solution $\boldsymbol{x}$ of the linear systems via

$$A = LU, \quad L\boldsymbol{y} = \boldsymbol{b}, \quad U\boldsymbol{x} = \boldsymbol{y}. \tag{1.6}$$

To be specific, forward substitution:

$$y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right) \quad (i = 1, \dots, n)$$

gives $\boldsymbol{y}$, and then back substitution

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \quad (i = n, \dots, 1)$$

yields the solution $\boldsymbol{x}$, where $y_1 = b_1/l_{11}$ and $x_n = y_n/u_{nn}$.

The necessary and sufficient condition for the existence of the LU decomposition is that all the *leading principal minors*[4] are not zero. Below are some examples of cases where the LU decomposition exists.

[4] Using MATLAB notation, "all the leading principal minors" of A are defined by det(A(1:1,1:1)), det(A(1:2,1:2)), …, det(A(1:n,1:n)).

1. Strictly diagonally dominant matrix: $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ $(1 \leq i \leq n)$.
2. M-matrix: $a_{ij} \leq 0$ $(i \neq j)$ and all the elements of A^{-1} are non-negative.
3. Symmetric part of matrix $A \in \mathbb{R}^{n \times n}$ is positive definite. The term "symmetric part" means $(A + A^\top)/2$ and the term "positive definite" means $\boldsymbol{v}^\top A \boldsymbol{v} > 0$ for any nonzero vector $\boldsymbol{v}$. For any square matrix A, matrix A can be written as the sum of the symmetric part $(A + A^\top)/2$ and the skew symmetric part $(A - A^\top)/2$.

The algorithm of the LU decomposition (Doolittle's method) is shown in Algorithm 1.1, which produces L and U such that $A = LU$.

Algorithm 1.1 The LU decomposition

```
Input: an $n \times n$ matrix $A$
Output: a unit lower triangular matrix $L$ and an upper triangular matrix $U$
1: for $i = 1, \dots, n$ do
2:    $a_{ji} = a_{ji}/a_{ii}$   $(j = i+1, \dots, n)$
3:    for $j = i+1, \dots, n$ do
4:       $a_{jk} = a_{jk} - a_{ji} \times a_{ik}$   $(k = i+1, \dots, n)$
5:    end for
6: end for
7: $l_{ii} = 1$   $(i = 1, \dots, n)$
8: $l_{ij} = a_{ij}$   $(i > j)$
9: $u_{ij} = a_{ij}$   $(i \leq j)$
```

1.3.2 LU Decomposition with Pivoting

The LU decomposition may not exist for some A. For example, consider

$$A = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}.$$

Then it is easy to see that the LU decomposition (Algorithm 1.1) fails due to zero division by $a_{11} = 0$. On the other hand, if the first row and the second row are swapped, i.e.,

$$PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 0 & 1 \end{pmatrix},$$

then the LU decomposition does not fail, and we have the LU decomposition of the form

$$PA = LU,$$

where

$$L = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 3 \\ 0 & 1 \end{pmatrix}.$$

This technique is called *pivoting*.

The LU decomposition with the *partial pivoting* is shown in Algorithm 1.2. Here, the term “partial” pivoting corresponds to lines 2–6 of Algorithm 1.2.

Algorithm 1.2 The LU decomposition with partial pivoting

```
Input: an $n \times n$ matrix $A$ and the $n \times n$ identity matrix $P(= I)$
Output: a unit lower triangular matrix $L$ and an upper triangular matrix $U$
Output: a permutation matrix $P$
 1: for $i = 1, \dots, n$ do
 2:   Find the maximum element $a_{p_i i}$ from $\{|a_{ii}|, \dots, |a_{ni}|\}$.
 3:   Swap $a_{ij}$ and $a_{p_i j}$ for $j = 1, \dots, n$.
 4:   if $i \neq n$ then
 5:     Swap $i$th row and $p_i$th row of matrix $P$.
 6:   end if
 7:   $a_{ji} = a_{ji}/a_{ii} \quad (j = i+1, \dots, n)$
 8:   for $j = i+1, \dots, n$ do
 9:     $a_{jk} = a_{jk} - a_{ji} \times a_{ik} \quad (k = i+1, \dots, n)$
10:   end for
11: end for
12: $l_{ii} = 1 \quad (i = 1, \dots, n)$
13: $l_{ij} = a_{ij} \quad (i > j)$
14: $u_{ij} = a_{ij} \quad (i \leq j)$
```

Algorithm 1.2 produces L, U, and permutation matrix P such that

$$PA = LU.$$

Unlike the LU decomposition, the LU decomposition with the partial pivoting never suffers from breakdown (zero-division).

Next, the LU decomposition with the *complete pivoting* is shown in Algorithm 1.3. The term “complete” pivoting corresponds to lines 2–8 of Algorithm 1.3.

Notice that at the ith step, the partial pivoting finds the maximum value from $\{|a_{k,i}| : k \in \{i, \dots, n\}\}$ and the complete pivoting finds the maximum value from $\{|a_{k,l}| : k, l \in \{i, \dots, n\}\}$.

Algorithm 1.3 produces decomposed matrices L and U and permutation matrices P and Q such that

$$PAQ = LU.$$

The LU decomposition with partial pivoting is usually used, and if the accuracy of the decomposition is not satisfactory, then the complete pivoting may be used.

Algorithm 1.3 The LU decomposition with complete pivoting

```
Input: an $n \times n$ matrix $A$ and the $n \times n$ identity matrices $P(= I)$, $Q(= I)$
Output: a unit lower triangular matrix $L$ and an upper triangular matrix $U$
Output: permutation matrices $P$, $Q$
1: for $i = 1, \dots, n$ do
2:   Find the maximum element $a_{p_i q_i}$ from $\{|a_{kl}| : i \le k \le n,\ i \le l \le n\}$.
3:   Swap $a_{ij}$ and $a_{p_i j}$ for $j = 1, \dots, n$.
4:   Swap $a_{ji}$ and $a_{jq_i}$ for $j = 1, \dots, n$.
5:   if $i \ne n$ then
6:     Swap $i$th row and $p_i$th row of matrix $P$.
7:     Swap $i$th column and $q_i$th row of matrix $P$.
8:   end if
9:   $a_{ji} = a_{ji}/a_{ii} \quad (j = i + 1, \dots, n)$
10:  for $j = i + 1, \dots, n$ do
11:    $a_{jk} = a_{jk} - a_{ji} \times a_{ik} \quad (k = i + 1, \dots, n)$
12:  end for
13: end for
14: $l_{ii} = 1 \quad (i = 1, \dots, n)$
15: $l_{ij} = a_{ij} \quad (i > j)$
16: $u_{ij} = a_{ij} \quad (i \le j)$
```

If we need to solve the set of linear systems of the form

$$A\boldsymbol{x}_i = \boldsymbol{b}_i, \quad i = 1, 2, \dots, m,$$

then the LU decomposition is very attractive because once we obtain L and U, then all we have to do is to conduct forward and back substitutions, see the explanation after (1.6).

1.3.3 Iterative Refinement

In this subsection, we consider the case where the approximate solution $\tilde{\boldsymbol{x}}$ of the linear systems (1.1) by the LU decomposition is not accurate enough.

In such a case, we can obtain the exact solution $\boldsymbol{x}$ if we know the correction vector $\Delta\boldsymbol{x}$ such that $\boldsymbol{x} = \tilde{\boldsymbol{x}} + \Delta\boldsymbol{x}$. It is easy to see that the correction vector $\Delta\boldsymbol{x}$ corresponds to the solution of the following linear systems:

$$A\Delta\boldsymbol{x} = \boldsymbol{r}, \tag{1.7}$$

where $\boldsymbol{r} := \boldsymbol{b} - A\tilde{\boldsymbol{x}}$. After solving (1.7) to obtain an approximate solution $\Delta\tilde{\boldsymbol{x}}$ of (1.7), the corrected approximate solution $\tilde{\boldsymbol{x}} + \Delta\tilde{\boldsymbol{x}}$ is expected to be better in accuracy than $\tilde{\boldsymbol{x}}$. This process can be repeated until the required accuracy is obtained, which is called *iterative refinement*. The algorithm of the iterative refinement is shown in Algorithm 1.4.

Algorithm 1.4 Iterative refinement for $A\boldsymbol{x} = \boldsymbol{b}$

```
Input: $A$, $\boldsymbol{b}$, and an approximate solution $\tilde{\boldsymbol{x}}$ for $A\boldsymbol{x} = \boldsymbol{b}$
Output: a refined approximate solution $\tilde{\boldsymbol{x}}$
1: Compute $\boldsymbol{r} = \boldsymbol{b} - A\tilde{\boldsymbol{x}}$ using "double" precision arithmetic.
2: Solve $A\Delta\boldsymbol{x} = \boldsymbol{r}$ to obtain the approximate solution $\Delta\tilde{\boldsymbol{x}}$.
3: Update $\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}} + \Delta\tilde{\boldsymbol{x}}$.
4: Repeat steps 1–3 until required accuracy is obtained.
```

If the LU decomposition is used for $A\boldsymbol{x} = \boldsymbol{b}$, then it is easy to obtain the approximate solution $\Delta\tilde{\boldsymbol{x}}$ because one can reuse the decomposed factors L and U, and thus only forward and back substitutions are required.

Notice that "double" precision arithmetic at step 1 in Algorithm 1.4 means that if we use single precision arithmetic, then compute $\boldsymbol{r} = \boldsymbol{b} - A\tilde{\boldsymbol{x}}$ using double precision arithmetic, and if we use double precision arithmetic, then compute it using quadruple precision arithmetic.

Algorithm 1.4 can be interpreted as Newton's method by letting $\boldsymbol{f}(\boldsymbol{x}) := \boldsymbol{b} - A\boldsymbol{x}$ and applying Newton's method to $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$.

1.4 Direct Methods for Symmetric Linear Systems

When matrix A is symmetric, *Cholesky decomposition* and *LDL*$^\top$ *decomposition* are efficient direct methods that are variants of the LU decomposition. As a preliminary, let us define the notion of *positive definite*:

1. Positive Definite
   Matrix A is called *positive definite* if $\boldsymbol{x}^\top A\boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$.
2. Symmetric Positive Definite
   Matrix A is called *symmetric positive definite* if positive definite matrix A is symmetric.

It is known that matrix A is symmetric positive definite if and only if all the eigenvalues of symmetric matrix A are positive.

1.4.1 Cholesky Decomposition

If matrix A is symmetric positive definite, there exists a lower triangular matrix L such that

$$A = LL^\top.$$

The decomposition is referred to as *Cholesky decomposition*. The algorithm of the Cholesky decomposition is shown in Algorithm 1.5.

Algorithm 1.5 Cholesky decomposition

```
Input: $A$
Output: $L$
1: for $i = 1, \dots, n$ do
2:     $l_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)^{1/2}$
3:     for $j = i+1, \dots, n$ do
4:         $l_{ji} = \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} \times l_{ik}\right) / l_{ii}$
5:     end for
6: end for
```

The computational cost of the Cholesky decomposition is about half of that of the LU decomposition because of the use of the symmetric property.

It is known that from the positive definiteness of A the Cholesky decomposition never suffers from breakdown and does not need the pivoting technique as described in Sect. 1.3.2.

1.4.2 $LDL^\top$ Decomposition

When symmetric matrix A is not positive definite, then the Cholesky decomposition cannot be used. On the other hand, if all the leading principal minors are non-negative, then there exists a diagonal matrix D (i.e., all the off-diagonal elements are zero) and a unit lower triangular matrix L (i.e., lower triangular matrix with all its diagonal elements being one) such that

$$A = LDL^\top.$$

The decomposition is referred to as $LDL^\top decomposition$ (or modified Cholesky decomposition) whose algorithm is shown in Algorithm 1.6.

Algorithm 1.6 $LDL^\top$ decomposition

```
Input: $A$
Output: $L$, $D$
1: for $i = 2, \dots, n$ do
2:     for $j = 1, \dots, i-1$ do
3:         $l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} \times l_{jk} \times d_{kk}\right) / d_{jj}$
4:     end for
5:     $d_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \times d_{kk}$
6: end for
```

Algorithm 1.6 produces diagonal elements of diagonal matrix D, and l_{ij} for lower triangular matrix L with $l_{ii} = 1$ for all i. Unlike the Cholesky decomposition, the square root does not appear in the $LDL^\top$ decomposition.

The $LDL^\top$ decomposition is not only useful for solving linear systems but also gives a rough distribution of the eigenvalues of symmetric matrix A. To see this, let S be a nonsingular matrix, then *Sylvester's law of inertia* ensures that the number of positive/zero/negative eigenvalues of symmetric matrix A is the same as that of positive/zero/negative eigenvalues of $SAS^\top$. Thus, if we have $A = LDL^\top$, then the number of positive/zero/negative diagonal elements tells us the number of positive/zero/negative eigenvalues of A.

Further, we can know the number of eigenvalues on a given closed interval $[\sigma_1, \sigma_2]$ by computing $LDL^\top$ factorizations of $A - \sigma_1 I = L_1 D_1 L_1^\top$ and $A - \sigma_2 I = L_2 D_2 L_2^\top$. To be specific, let $n_1 (\geq 0)$ be the number of nonnegative diagonal elements of D_1 and let $n_2^{(>0)}$ be the number of positive diagonal elements of D_2. Then the number of eigenvalues on the closed interval $[\sigma_1, \sigma_2]$ is given by $n_1^{(\geq 0)} - n_2^{(>0)}$.

The fact is useful for the kth (generalized) eigenvalue problem:

$$Ax = \lambda Bx, \tag{1.8}$$

where A is symmetric and B is symmetric positive definite. It is known that all the eigenvalues are real numbers, and thus they can be written as $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k \leq \cdots \leq \lambda_n$. The kth (generalized) eigenvalue problem is to find λ_k and the corresponding eigenvector of (1.8). The generalized eigenvalue problem (1.8) is equivalent to the following shifted problem:

$$(A - \sigma B)x = (\lambda - \sigma)Bx. \tag{1.9}$$

Since B is symmetric positive definite, B has the Cholesky decomposition, i.e., $B = L_B L_B^\top$. Thus from (1.9) we have

$$L_B^{-1}(A - \sigma B)L_B^{-\top}\tilde{x} = (\lambda - \sigma)\tilde{x},$$

where $\tilde{x} = L_B^\top x$. Now, let D be the diagonal matrix of the $LDL^\top$ decomposition of $(A - \sigma B)$. Then,

$$L_B^{-1}LDL^\top L_B^{-\top}\tilde{x} = (\lambda - \sigma)\tilde{x} \Leftrightarrow GDG^\top\tilde{x} = (\lambda - \sigma)\tilde{x},$$

where $G = L_B^{-1}L$. This indicates that using Sylvester's law of inertia, the number of positive (or negative) diagonal elements of D equals the number of the eigenvalues that are greater (or less) than σ. Thus, if the number of negative diagonal elements of D is, e.g., 10, then we can know that the eigenvalue λ closest to σ with the condition $\lambda \leq \sigma$ is the 10th eigenvalue, i.e., $\lambda = \lambda_{10}$. Based on this idea, together with several accelerating techniques, an algorithm to obtain the kth eigenvalue and eigenvector of the (generalized) eigenvalue problems was proposed in [120]. For the kth eigenvalue problems, see also [156].

1.5 Direct Methods for Large and Sparse Linear Systems

Consider the following matrix:

$$A = \begin{pmatrix} 2 & -1 & -1 & -1 \\ 1 & 1 & \mathbf{0} & \mathbf{0} \\ 1 & \mathbf{0} & 1 & \mathbf{0} \\ 1 & \mathbf{0} & \mathbf{0} & 1 \end{pmatrix}. \tag{1.10}$$

If the number of zeros in a matrix is relatively large, then the matrix is called *sparse*. Conversely, if the number of zeros in a matrix is relatively small, it is called *dense*.

Applying the LU decomposition (Algorithm 1.1) to (1.10) yields

$$A = LU = \begin{pmatrix} 1 & & & \\ 1/2 & 1 & & \\ 1/2 & \mathbf{1/3} & 1 & \\ 1/2 & \mathbf{1/3} & \mathbf{1/4} & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & -1 & -1 \\ & 3/2 & \mathbf{1/2} & \mathbf{1/2} \\ & & 4/3 & \mathbf{1/3} \\ & & & 5/4 \end{pmatrix}.$$

For the lower part of A, we see that $a_{32} = a_{42} = a_{43} = 0$ but the corresponding lower part of L, we have $l_{32} = l_{42} = l_{43} \neq 0$. Similarly, for the upper part of A, we see that $a_{23} = a_{24} = a_{34} = 0$ but $u_{23} = u_{24} = u_{34} \neq 0$ for the upper part of U. The nonzero elements in L and U are called *fill-ins* if the elements in the original matrix A are zero but the corresponding elements in L and U are not zero.

The fill-ins can be a bottleneck to computing the LU decomposition when matrix A is large and sparse. For a large and sparse matrix, we usually only use nonzero elements and do not store zero elements in memory. However, after the LU decomposition, the L and U may become dense, i.e., at worst n^2 elements should be stored in the memory. This indicates that if the matrix size n is $n =$1,000,000 and we use real and double precision arithmetic, then we need $n^2 \times 8 \approx 8$ Terabytes of memory. If $n =$10,000,000, then a computer with 800 Terabytes of memory is required.

As we will see in Chap. 2, matrices arising from scientific computing are often large and sparse. Thus, for solving large and sparse linear systems by the LU decomposition, it is of prime importance to reduce the fill-ins. In what follows, the key idea to reduce the fill-ins is described.

The key idea is to permute some rows and columns of sparse matrix A to reduce fill-ins, and then apply the LU decomposition to the permuted matrix. To be specific, we apply the LU decomposition to PAQ, where P and Q are prescribed permutation matrices. Then we solve

$$PAQ\boldsymbol{y} = P\boldsymbol{b}$$

to obtain the solution $\boldsymbol{x} = Q\boldsymbol{y}$. Here, P and Q are chosen so that the number of fill-ins of L and U are small as possible.

Below is an example to reduce fill-ins of A in (1.10). Consider

$$P = Q = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Then, from (1.10) it follows that

$$PAQ = \begin{pmatrix} 1 & \mathbf{0} & \mathbf{0} & 1 \\ \mathbf{0} & 1 & \mathbf{0} & 1 \\ \mathbf{0} & \mathbf{0} & 1 & 1 \\ -1 & -1 & -1 & 2 \end{pmatrix}.$$

The LU decomposition of PAQ yields

$$PAQ = LU = \begin{pmatrix} 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} \\ -1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0} & \mathbf{0} & 1 \\ \mathbf{0} & 1 & \mathbf{0} & 1 \\ \mathbf{0} & \mathbf{0} & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 5 \end{pmatrix}.$$

In this case, fortunately, fill-ins do not occur. The result is memory-efficient by storing only nonzero elements in memory.

Although it is difficult to choose optimal permutation matrices P and Q so that the number of fill-ins is minimum, approximate optimization methods have been developed. Well-known methods are the minimum degree method and the reverse Cuthill–McKee method. For the references of these methods see Section 3.5.5. For the details of sparse direct methods, see, e.g., [44].

1.6 Stationary Iterative Methods

Stationary iterative methods produce approximate solutions by the following recurrences:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k), \quad k = 0, 1, \dots \tag{1.11}$$

with a prescribed initial guess $\boldsymbol{x}_0$ for $A\boldsymbol{x} = \boldsymbol{b}$. Here the vector-valued function $\boldsymbol{f}$ is defined by

$$\boldsymbol{f}(\boldsymbol{y}) := M^{-1}N\boldsymbol{y} + M^{-1}\boldsymbol{b}, \tag{1.12}$$

where M is a nonsingular matrix and N is a matrix such that matrix A in (1.1) satisfies $A = M - N$. The solution $\boldsymbol{x}$ corresponds to a fixed–point of $\boldsymbol{f}$, i.e., $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$. Therefore, the stationary iterative methods can be regarded as iterative methods finding the fixed–point of $\boldsymbol{f}$ via the recurrences in (1.11). Since $\boldsymbol{f}$ does not depend

Table 1.1 Classification of three stationary iterative methods

	M	N
The Jacobi method	D	$-(L+U)$
The Gauss–Seidel method	$D+L$	$-U$
The SOR method	$(D+\omega L)/\omega$	$N=[(1-\omega)D-\omega U]/\omega$

on the number of iterations k, the iterative methods are called "stationary". From (1.11) and (1.12), the iterates $\boldsymbol{x}_{k+1}$ are written as

$$\boldsymbol{x}_{k+1} = M^{-1}N\boldsymbol{x}_k + M^{-1}\boldsymbol{b} \tag{1.13}$$

for $k = 0, 1, \dots$.

Stationary iterative methods are used for solving large and sparse linear systems because unlike the direct methods such as the LU decomposition, we do not need to consider fill-ins and only need to store nonzero elements of the coefficient matrix and some working vectors in memory for computing the approximate solutions.

In what follows, the best-known stationary iterative methods are described, i.e., the Jacobi method, the Gauss–Seidel (GS) method, and the Successive Over-Relaxation (SOR) method. Let $A = (a_{ij})$ be the coefficient matrix for $A\boldsymbol{x} = \boldsymbol{b}$, and

$$\begin{aligned} L &= (a_{ij}), \quad a_{ij} = 0 \text{ for } i \le j, \\ D &= (a_{ij}), \quad a_{ij} = 0 \text{ for } i \ne j, \\ U &= (a_{ij}), \quad a_{ij} = 0 \text{ for } i \ge j. \end{aligned}$$

Then we have

$$A = L + D + U.$$

L, D, and U are referred to as the *strictly lower triangular matrix*, *diagonal matrix*, and *strictly upper triangular matrix*.

Choices of M and N using L, D, and U yield the Jacobi, GS, and SOR methods. Table 1.1 shows the choices for the representative stationary iterative methods.

1.6.1 The Jacobi Method

From (1.13) and Table 1.1, the iterates $\boldsymbol{x}_k$ of the Jacobi method are given as follows:

$$\boldsymbol{x}_{k+1} = -D^{-1}(L+U)\boldsymbol{x}_k + D^{-1}\boldsymbol{b}.$$

In practice, $\boldsymbol{x}_{k+1}$ is computed by the following recurrence relation:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) \tag{1.14}$$

for $i = 1, 2, \dots, n$, where $x_i^{(k)}$ and b_i are the ith element of $\boldsymbol{x}_k$ and $\boldsymbol{b}$ respectively, and the symbol $\sum_{j<i}$ and $\sum_{j>i}$ denotes $\sum_{j=1}^{i-1}$ and $\sum_{j=i+1}^{n}$ respectively.

1.6.2 The Gauss–Seidel Method

From (1.13) and Table 1.1, the iterates $\boldsymbol{x}_k$ of the Gauss–Seidel (GS) method are given as follows:

$$\boldsymbol{x}_{k+1} = -(D+L)^{-1} U \boldsymbol{x}_k + (D+L)^{-1} \boldsymbol{b}.$$

A derivation of the GS method is given below. Let us reconsider the Jacobi method. When $x_i^{(k+1)}$ of the Jacobi method is computed by (1.14), $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$ have already been computed, which can be regarded as new information. Thus, a faster convergence may be expected if we use the new information $\sum_{j<i} a_{ij} x_j^{(k+1)}$ instead of the old information $\sum_{j<i} a_{ij} x_j^{(k)}$. This is the key idea of the Gauss–Seidel method, which is written as follows:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) \tag{1.15}$$

for $i = 1, 2, \dots, n$.

1.6.3 The SOR Method

From (1.13) and Table 1.1, the iterates $\boldsymbol{x}_k$ of the SOR method are given as follows:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \left[\frac{1}{\omega}(D+\omega L) \right]^{-1} \left\{ \frac{1}{\omega}[(1-\omega)D - \omega U] \right\} \boldsymbol{x}_k + \left[\frac{1}{\omega}(D+\omega L) \right]^{-1} \boldsymbol{b} \\ &= (D+\omega L)^{-1}[(1-\omega)D - \omega U]\boldsymbol{x}_k + (D+\omega L)^{-1}\omega \boldsymbol{b}, \end{aligned}$$

or equivalently $\boldsymbol{x}_{k+1}$ of the SOR method is given by solving the following linear systems:

$$(D + \omega L)\boldsymbol{x}_{k+1} = [(1-\omega)D - \omega U]\boldsymbol{x}_k + \omega \boldsymbol{b}.$$

The element-wise iterates of the SOR method are described below:

$$\begin{aligned}
\tilde{x}_i^{(k+1)} &= \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right), \\
x_i^{(k+1)} &= x_i^{(k)} + \omega(\tilde{x}_i^{(k+1)} - x_i^{(k)})
\end{aligned}$$

for $i = 1, 2, \dots, n$. From this, it is easy to see that the SOR method is a one-parameter generalization of the Gauss–Seidel method since the right-hand side of the first equation is the same as the Gauss–Seidel method (1.15). If $\omega = 1$, the SOR method reduces to the Gauss–Seidel method. It is known that the necessary condition for convergence is $0 < \omega < 2$.

When A is symmetric, the symmetric SOR (SSOR) method is also well known, since it can be used as a preconditioner for Krylov subspace methods to solve symmetric linear systems. For the SSOR preconditioner, see Section 3.5.4. The iterates of the SSOR method are given as follows:

$$\begin{aligned}
(D + \omega L)\tilde{\boldsymbol{x}}_k &= [(1-\omega)D - \omega U]\boldsymbol{x}_k + \omega \boldsymbol{b}, \\
(D + \omega U)\boldsymbol{x}_k &= [(1-\omega)D - \omega L]\tilde{\boldsymbol{x}}_k + \omega \boldsymbol{b}.
\end{aligned}$$

For the SSOR method, M and N in (1.13) are given by

$$M_{\mathrm{SSOR}} = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega U), \tag{1.16}$$

$$N_{\mathrm{SSOR}} = \frac{1}{\omega(2-\omega)}[(1-\omega)D - \omega L]D^{-1}[(1-\omega)D - \omega U]. \tag{1.17}$$

Since A is symmetric, we have $U = L^\top$. Thus, it follows from (1.16) and (1.17) that M_{SSOR} and N_{SSOR} are symmetric.

Many basic results of the SOR method including the SSOR method are summarized in [93]. For recent developments of the SOR methods, see, e.g., [128, 130, 131] and the references therein. In [131], an adaptive SOR method is proposed based on a novel connection between the SOR method and discrete gradient methods for gradient systems in [130], which gives a geometric view of the SOR method.

1.6.4 Convergence of the Stationary Iterative Methods

A remarkable property of the stationary iterative methods is that we can estimate the rate of error reduction by computing $\|M^{-1}N\|$, where $\|\cdot\|$ is a subordinate matrix norm (1.2). To see this, from (1.13) and the relation $A\boldsymbol{x} = \boldsymbol{b} \Leftrightarrow \boldsymbol{x} = M^{-1}N\boldsymbol{x} +$

$M^{-1}\boldsymbol{b}$, the error is given by

$$\begin{aligned}\boldsymbol{e}_{k+1} &:= \boldsymbol{x} - \boldsymbol{x}_{k+1}\\ &= (M^{-1}N\boldsymbol{x} + M^{-1}\boldsymbol{b}) - (M^{-1}N\boldsymbol{x}_k + M^{-1}\boldsymbol{b})\\ &= M^{-1}N(\boldsymbol{x} - \boldsymbol{x}_k)\\ &= M^{-1}N\boldsymbol{e}_k. \end{aligned} \tag{1.18}$$

Thus the norm of the error is given by

$$\|\boldsymbol{e}_{k+1}\| \le \|M^{-1}N\|\|\boldsymbol{e}_k\|.$$

The rate of the reduction of the error is $\|M^{-1}N\|$, which shows linear convergence. Let λ_i $(i = 1, \dots, n)$ be the eigenvalues of an n-by-n matrix G and let $\rho(G) = \max\{|\lambda_1|, \dots, |\lambda_n|\}$. Then, $\rho(G)$ is referred to as the *spectral radius* of G. It is easy to see from (1.18) and the Jordan decomposition of $M^{-1}N$ that the necessary and sufficient condition of the convergence for an arbitrary initial guess is

$$\rho(M^{-1}N) < 1. \tag{1.19}$$

In what follows, the best-known fact for satisfying (1.19) is described.

Definition 1.1 (Regular splitting) $A = M - N$ is called *regular splitting* if $M^{-1} \ge O$ and $N \ge O$, i.e., all the elements of M^{-1} and N are greater than or equal to zero.

Theorem 1.3 *If $A = M - N$ is regular splitting, then*

$$\rho(M^{-1}N) < 1 \Leftrightarrow A^{-1} \ge O.$$

Proof We first show $\rho(M^{-1}N) < 1 \Rightarrow A^{-1} \ge O$. Let $G := M^{-1}N$ and λ_i be the eigenvalues of G. Then, the eigenvalues of $I - G$ are nonzeros (i.e., nonsingular), because the eigenvalues of $I - G$ are $1 - \lambda_i$, and $|\lambda_i| < 1$ from the assumption $\rho(G) < 1$. Since M and $I - G$ are nonsingular, $A = M(I - G)$ is also nonsingular, i.e., there exists A^{-1}. Since $\rho(G) < 1$, we have

$$A^{-1} = (I - G)^{-1}M^{-1} = (I + G + G^2 + \cdots)M^{-1}.$$

Since $A = M - N$ is assumed to be regular splitting, $G \ge O$ and $M^{-1} \ge O$. Thus $A^{-1} \ge O$.

Next, we show the converse. Since $A = M - N$ is regular splitting, $M^{-1} \ge O$ and $N \ge O$. Thus $G \ge O$. It follows from the Perron–Frobenius theorem that there exists a nonnegative eigenvector $\boldsymbol{x} \ge \boldsymbol{0}$ such that $G\boldsymbol{x} = \rho(G)\boldsymbol{x}$. It is easy to see that $A^{-1}N = (I - G)^{-1}G$, and thus we have

$$A^{-1}N\boldsymbol{x} = (I - G)^{-1}G\boldsymbol{x} = \frac{\rho(G)}{1 - \rho(G)}\boldsymbol{x}.$$

Since $A^{-1} \geq O$ and $N \geq O$, we have $A^{-1}N\boldsymbol{x} \geq \mathbf{0}$. This implies $\rho(G)/(1 - \rho(G)) \geq 0$, which means $(0 \leq)\rho(G) \leq 1$. Since there exists $(I - G)^{-1}$, we have $\rho(G) \neq 1$. □

For further convergence theorems of the stationary iterative methods, see, e.g., an excellent book by Varga [200].

The stationary iterative methods can also be used as a preconditioner for Krylov subspace methods in Section 3.5.4.

1.7 Multigrid Methods

Multigrid methods fall into a class of iterative methods that have been developed for solving linear systems arising from elliptic partial differential equations. Multigrid methods find a good approximate solution of the linear systems from the information of some rough approximate solutions of smaller linear systems (arising from hierarchical coarser meshes) and an accurate approximate solution of the smallest linear systems (arising from the coarsest mesh). For obtaining the rough approximate solutions, stationary iterative methods (especially the Gauss–Seidel method) are often used, which are called *smoothers*. The name *smoother* comes from the fact that the Gauss–Seidel method damps quickly the high frequency components of the error or residuals for the linear systems, and the multigrid methods make the most of this property.

The notable feature of the multigrid methods is that the convergence rate does not depend on the mesh size, though stationary iterative methods and Krylov subspace methods do. The multigrid methods are later extended to solving other linear systems arising from not only elliptic ones but other models.

Multigrid methods are the subject of active research, and there are many excellent books on multigrid methods, some of them by:

- Hackbusch [91],
- Wesseling [204],
- Bramble [27],
- Briggs et al. [28],
- Trottenberg et al.[193],
- Lottes [125].

Wolfgang Hackbusch's book presents not only the basic concepts of multigrid methods using simple one-dimensional differential equations but also describes algorithms, their program codes, and detailed convergence analysis. James Lottes's book focuses mainly on algebraic multigrid methods.

1.8 Krylov Subspace Methods

Krylov subspace methods have a long history, dating from the mid-20th century, and the best-known method is the Conjugate Gradient (CG) method by Hestenes and Stiefel.[5]

Krylov subspace methods are iterative methods for solving linear systems (1.1) by utilizing Krylov subspace as defined next.

Definition 1.2 (Krylov subspace) Let vectors $\boldsymbol{v}, A\boldsymbol{v}, \dots, A^{n-1}\boldsymbol{v} \in \mathbb{C}^N$ be linearly independent, then

$$\mathcal{K}_n(A, \boldsymbol{v}) = \left\{c_0\boldsymbol{v} + c_1 A\boldsymbol{v} + \cdots + c_{n-1}A^{n-1}\boldsymbol{v} \,:\, c_0, c_1, \dots, c_{n-1} \in \mathbb{C}\right\} \quad (1.20)$$
$$(= \operatorname{span}\{\boldsymbol{v}, A\boldsymbol{v}, \dots, A^{n-1}\boldsymbol{v}\})$$

is called an *(n dimensional) Krylov subspace*.

The dimension of $\mathcal{K}_n(A, \boldsymbol{v})$ depends on $\boldsymbol{v}$, and the largest dimension with respect to a given $\boldsymbol{v}$ is known as *the grade of a Krylov subspace*, as defined next.

Definition 1.3 (Grade of Krylov subspace) The smallest number n such that

$$\dim(\mathcal{K}_n(A, \boldsymbol{v})) = \dim(\mathcal{K}_{n+1}(A, \boldsymbol{v})) \quad (1.21)$$

is called *the grade of A with respect to $\boldsymbol{v}$*.

Consider $A\boldsymbol{x} = \boldsymbol{b}$ and let m be the grade of A with respect to $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$, where $\boldsymbol{x}_0$ is an initial guess to the solution of $A\boldsymbol{x} = \boldsymbol{b}$. Then

$$\dim(\mathcal{K}_m(A, \boldsymbol{r}_0)) = \dim(\mathcal{K}_{m+1}(A, \boldsymbol{r}_0)).$$

This means that there exist $c_0, c_1, \dots, c_{m-1}$ such that

$$A^m\boldsymbol{r}_0 = c_0\boldsymbol{r}_0 + c_1 A\boldsymbol{r}_0 + \cdots + c_{m-1}A^{m-1}\boldsymbol{r}_0.$$

Multiplying the equation by A^{-1} yields

$$A^{m-1}\boldsymbol{r}_0 = c_0 A^{-1}\boldsymbol{r}_0 + c_1\boldsymbol{r}_0 + c_2 A\boldsymbol{r}_0 + \cdots + c_{m-1}A^{m-2}\boldsymbol{r}_0.$$

Since $A^{-1}\boldsymbol{r}_0 = A^{-1}(\boldsymbol{b} - A\boldsymbol{x}_0) = \boldsymbol{x} - \boldsymbol{x}_0$, the solution can be written as

$$\boldsymbol{x} = \boldsymbol{x}_0 + \frac{1}{c_0}\left(-c_1\boldsymbol{r}_0 - c_2 A\boldsymbol{r}_0 - \cdots - c_{m-1}A^{m-2}\boldsymbol{r}_0 + A^{m-1}\boldsymbol{r}_0\right)$$

[5] After the proposal of the CG method, Stiefel proposed the Conjugate Residual (CR) method in 1955, and he is also known for the Stiefel manifold that is an extension of the unit circle.

if $c_0 \neq 0$. Let $z_m := \left(-c_1 \boldsymbol{r}_0 - \cdots - c_{m-1} A^{m-2} \boldsymbol{r}_0 + A^{m-1} \boldsymbol{r}_0\right)/c_0$. Then, $z_m \in \mathcal{K}_m(A, \boldsymbol{r}_0)$, which means that the solution $\boldsymbol{x}$ satisfies

$$\boldsymbol{x} = \boldsymbol{x}_0 + z_m, \quad z_m \in \mathcal{K}_m(A, \boldsymbol{r}_0).$$

From this we see that the solution $\boldsymbol{x}$ belongs to the affine space $\boldsymbol{x}_0 + \mathcal{K}_m(A, \boldsymbol{r}_0)$. This fact leads to a natural idea to compute approximate solution $\boldsymbol{x}_n$ over the affine space $\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)$, or equivalently

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + z_n, \quad z_n \in \mathcal{K}_n(A, \boldsymbol{r}_0)$$

for $n = 1, 2, \ldots$. If z_n is appropriately determined, then this approach gives the exact solution at m iteration steps, which is the basis of Krylov subspace methods.

Now the framework of Krylov subspace methods is described below. Let $\boldsymbol{x}_0 \in \mathbb{C}^N$ be an initial guess, and let $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$ be the corresponding residual vector. Then, Krylov subspace methods produce the nth approximate solution over the following affine space:

Krylov subspace condition

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + z_n, \quad z_n \in \mathcal{K}_n(A, \boldsymbol{r}_0). \tag{1.22}$$

Then, the corresponding residual vector $\boldsymbol{r}_n$ is given by

$$\boldsymbol{r}_n := \boldsymbol{b} - A\boldsymbol{x}_n = \boldsymbol{r}_0 - A z_n, \quad \boldsymbol{r}_n \in \mathcal{K}_{n+1}(A, \boldsymbol{r}_0). \tag{1.23}$$

Since $\boldsymbol{x}_n$ cannot be uniquely determined by utilizing the condition (1.22) above, one of the following conditions needs to be imposed for obtaining the unique approximate solution:

Ritz–Galerkin approach

$$\boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0). \tag{1.24}$$

Petrov–Galerkin approach

$$\boldsymbol{r}_n \perp W_n \subset \mathbb{C}^N, \quad \dim(W_n) = n. \tag{1.25}$$

Minimal residual approach

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)} \|\boldsymbol{b} - A\boldsymbol{x}_n\|. \tag{1.26}$$

The Petrov–Galerkin approach includes the Ritz–Galerkin approach and the minimal residual approach as described below.

Proposition 1.1 *The following statements holds true:*

- *The Ritz–Galerkin approach* (1.24) *is the Petrov–Galerkin approach* (1.25) *with the choice*

$$W_n = \mathcal{K}_n(A, \boldsymbol{r}_0).$$

- *The minimal residual approach* (1.26) *is the Petrov–Galerkin approach* (1.25) *with the choice*

$$W_n = A\mathcal{K}_n(A, \boldsymbol{r}_0), \tag{1.27}$$

where $A\mathcal{K}_n(A, \boldsymbol{r}_0) := \left\{\sum_{k=1}^{n} c_k A\boldsymbol{v}_k : c_1, c_2, \ldots, c_n \in \mathbb{C}\right\}$*, and* $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ *are the basis vectors of* $\mathcal{K}_n(A, \boldsymbol{r}_0)$.

Proof It is obvious for the Ritz–Galerkin approach, so the equivalence of (1.26) and (1.27) will only be proved. Let $V_n = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n]$ be an N-by-n matrix whose column vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ are the basis vectors of $\mathcal{K}_n(A, \boldsymbol{r}_0)$. Then $\boldsymbol{z}_n \in \mathcal{K}_n(A, \boldsymbol{r}_0)$ in (1.22) can be written as $\mathcal{K}_n(A, \boldsymbol{r}_0) \ni \boldsymbol{z}_n = \sum_{k=1}^{n} c_k \boldsymbol{v}_k = V_n \boldsymbol{c}_n$, where $\boldsymbol{c}_n = [c_1, c_2, \ldots, c_n]^\top \in \mathbb{C}^n$. Substitution of $\boldsymbol{z}_n = V_n \boldsymbol{c}_n$ into (1.23) yields

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n\boldsymbol{c}_n. \tag{1.28}$$

Note that if $\boldsymbol{r}_n \perp A\boldsymbol{v}_k (k = 1, 2, \ldots, n)$, then it holds that $\boldsymbol{r}_n \perp A\mathcal{K}_n(A, \boldsymbol{r}_0)$ because it is easy to see that $\boldsymbol{r}_n \perp \sum_{k=1}^{n} c_k A\boldsymbol{v}_k$ for all $c_i \in \mathbb{C}$. The condition $\boldsymbol{r}_n \perp A\boldsymbol{v}_k (k = 1, 2, \ldots, n)$ is equivalent to $(AV_n)^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0}$ $(:= [0, \ldots, 0]^\top)$, and thus it follows from (1.28) that $\boldsymbol{0} = (AV_n)^{\mathrm{H}}\boldsymbol{r}_n = (AV_n)^{\mathrm{H}}\boldsymbol{r}_0 - (AV_n)^{\mathrm{H}}AV_n\boldsymbol{c}_n$. Thus, the vector $\boldsymbol{c}$ is determined by

$$\boldsymbol{c}_n = [(AV_n)^{\mathrm{H}}AV_n]^{-1}(AV_n)^{\mathrm{H}}\boldsymbol{r}_0. \tag{1.29}$$

We now show that $\boldsymbol{c}_n$ is the solution of the minimization problem (1.26). Since the minimized solution of $\|\boldsymbol{b} - A\boldsymbol{x}_n\|^2$ is the same as that of (1.26), we consider

$$\begin{aligned}\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)} \|\boldsymbol{b} - A\boldsymbol{x}_n\|^2 &= \min_{\boldsymbol{c}_n \in \mathbb{C}^n} \|\boldsymbol{r}_0 - AV_n\boldsymbol{c}_n\|^2 \\ &= \min_{\boldsymbol{c}_n \in \mathbb{C}^n} (\boldsymbol{r}_0 - AV_n\boldsymbol{c}_n)^{\mathrm{H}}(\boldsymbol{r}_0 - AV_n\boldsymbol{c}_n). \end{aligned} \tag{1.30}$$

The following decomposition is useful for finding the solution of (1.30):

$$\|\boldsymbol{d} - M\boldsymbol{x}\|^2 = (\boldsymbol{d} - M\boldsymbol{x})^{\mathrm{H}}(\boldsymbol{d} - M\boldsymbol{x}) = f(\boldsymbol{x}) + c, \tag{1.31}$$

Table 1.2 Krylov subspace methods for Hermitian linear systems

Ritz–Galerkin approach	Minimal residual approach
CG	MINRES, CR

Table 1.3 Some Krylov subspace methods for non-Hermitian linear systems

Ritz–Galerkin approach	Petrov–Galerkin approach	Minimal residual approach
FOM	BiCG, BiCR	GMRES, GCR

where

$$\begin{aligned} f(\boldsymbol{x}) &= (M^{\mathrm{H}}M\boldsymbol{x} - M^{\mathrm{H}}\boldsymbol{d})^{\mathrm{H}}(M^{\mathrm{H}}M)^{-1}(M^{\mathrm{H}}M\boldsymbol{x} - M^{\mathrm{H}}\boldsymbol{d}),\\ c &= -\boldsymbol{d}^{\mathrm{H}}M(M^{\mathrm{H}}M)^{-1}M^{\mathrm{H}}\boldsymbol{d} + \boldsymbol{d}^{\mathrm{H}}\boldsymbol{d}. \end{aligned}$$

If M is column full rank, then $M^{\mathrm{H}}M$ is Hermitian positive definite,[6] and thus $f(\boldsymbol{x}) \geq 0$. This means the minimization of (1.31) is achieved by minimizing $f(\boldsymbol{x})$. It is easy to see that

$$\boldsymbol{x} = (M^{\mathrm{H}}M)^{-1}M^{\mathrm{H}}\boldsymbol{d} \tag{1.32}$$

gives $f(\boldsymbol{x}) = 0$, which is the solution of the minimization problem of (1.31). Letting $\boldsymbol{x} = (M^{\mathrm{H}}M)^{-1}M^{\mathrm{H}}\boldsymbol{d}$ with $M = AV_n$ and $\boldsymbol{d} = \boldsymbol{r}_0$ yields the same result as in (1.29). □

Remark 1 The equation (1.31) is also useful for solving the following minimization:

$$\min_{\omega \in \mathbb{C}} \|\boldsymbol{a} - \omega\boldsymbol{b}\| \tag{1.33}$$

because setting $\boldsymbol{d} = \boldsymbol{a}$, $M = \boldsymbol{b}$, $\boldsymbol{x} = \omega$ in (1.32) yields the solution of (1.33) as follows:

$$\omega = \frac{\boldsymbol{b}^{\mathrm{H}}\boldsymbol{a}}{\boldsymbol{b}^{\mathrm{H}}\boldsymbol{b}}. \tag{1.34}$$

The result will be used for Krylov subspace methods in Chap. 3.

Now, the relationship among representative Krylov subspace methods and three approaches (1.24)–(1.26) is described in Tables 1.2 and 1.3.

[6] Matrix $G \in \mathbb{C}^{N\times N}$ is called *Hermitian positive definite* if G is Hermitian and $\boldsymbol{v}^{\mathrm{H}}G\boldsymbol{v} > 0$ for all $\boldsymbol{v}(\neq \boldsymbol{0}) \in \mathbb{C}^N$, or equivalently, all the eigenvalues of Hermitian matrix G are positive. From this, it follows that if G is Hermitian positive definite, then G^{-1} is also Hermitian positive definite. For the positive definiteness, see also Sect. 1.4.

The relationship among Krylov subspace methods for Hermitian and non-Hermitian linear systems is as follows: the BiCG method and the FOM are extensions of the CG method; the GMRES method is an extension of the MINRES method; the BiCR method and the GCR method are extensions of the CR method. The details of these Krylov subspace methods are described in Chap. 3.

1.9 Orthogonalization Methods for Krylov Subspaces

For satisfying Ritz–Galerkin or Petrov–Galerkin or minimal residual approaches in Sect. 1.8, constructing basis vectors of Krylov subspace is required. For a non-Hermitian matrix, the Arnoldi and the bi-Lanczos processes are used for Krylov subspace basis vectors. For a complex symmetric matrix, the complex symmetric Lanczos process is used, and for a Hermitian matrix, the Lanczos process is used.

The Arnoldi process produces orthonormal basis vectors of Krylov subspace $\mathcal{K}_n(A, \boldsymbol{v})$. The Arnoldi process never suffers from breakdown (i.e., zero-division), but the computational costs grow linearly as n increases. The bi-Lanczos process produces bi-orthogonal basis vectors of the Krylov subspace. The computational costs do not grow linearly as n increases and thus the costs are less than those of the Arnoldi method. However, it may suffer from breakdown, and near breakdown will lead to loss of orthogonality in the basis vectors. The complex symmetric Lanczos process and the Lanczos process are special cases of the bi-Lanczos process. In particular, since the Lanczos process is also a special case of the Arnoldi process, the Lanczos process has favorable features of the Arnoldi process and the bi-Lanczos process, i.e., the Lanczos process generates orthonormal basis vectors and the computational costs do not grow linearly as n increases.

In the following subsections, these methods are described, together with algorithms and matrix representations that will be used for the derivations of Krylov subspace methods.

1.9.1 The Arnoldi Process

The Arnoldi process produces orthogonalized basis vectors of Krylov subspace $\mathcal{K}_n(A, \boldsymbol{v})$, i.e., the basis vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n$ satisfy

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

where $(\boldsymbol{x}, \boldsymbol{y}) := \boldsymbol{x}^{\mathrm{H}}\boldsymbol{y} = \sum_{i=1}^{N} \bar{x}_i y_i$ is the standard dot product and $\delta_{i,j}$ is the Kronecker delta, i.e., $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ if $i \neq j$.

For simplicity, consider producing orthonormal vectors of $\mathcal{K}_n(A, \boldsymbol{v})$ for the case $n = 3$, i.e., producing $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ from $\mathrm{span}\{\boldsymbol{v}, A\boldsymbol{v}, A^2\boldsymbol{v}\}$.

A method to obtain orthonormal vectors from span$\{\boldsymbol{v}, A\boldsymbol{v}, A^2\boldsymbol{v}\}$ is to use the Gram–Schmidt process, which computes $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}_3$ as follows:

$$\begin{aligned}\boldsymbol{v}_1 &= z_1\boldsymbol{v},\\ \boldsymbol{v}_2 &= z_2(\underline{A\boldsymbol{v}} - c_1\boldsymbol{v}_1),\\ \boldsymbol{v}_3 &= z_3(\underline{A^2\boldsymbol{v}} - d_1\boldsymbol{v}_1 - d_2\boldsymbol{v}_2).\end{aligned}$$

Here z_i $(i = 1, 2, 3)$ are scalar values for satisfying the normalizations $\|\boldsymbol{v}_i\| = 1$. Scalar value c_1 is determined so that $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are orthogonal, i.e., $(\boldsymbol{v}_1, \boldsymbol{v}_2) = 0$. Similarly, scalar values d_1 and d_2 are determined so that $\boldsymbol{v}_3 \perp \boldsymbol{v}_2$ and $\boldsymbol{v}_3 \perp \boldsymbol{v}_1$, i.e., $(\boldsymbol{v}_1, \boldsymbol{v}_3) = (\boldsymbol{v}_2, \boldsymbol{v}_3) = 0$. However, it is known that this approach is numerically unstable, since the angle between $A^n\boldsymbol{v}$ and $A^{n-1}\boldsymbol{v}$ may be close to zero for large n, leading to severe cancellation. In fact, under a suitable condition, $A^n\boldsymbol{v}$ converges to an eigenvector of A whose eigenvalue is the maximum in magnitude among eigenvalues of A.

The Arnoldi process slightly modifies the (classical) Gram-Schmidt process as follows:

$$\boldsymbol{v}_1 = z_1\boldsymbol{v}, \tag{1.35}$$

$$\boldsymbol{v}_2 = z_2(\underline{A\boldsymbol{v}_1} - c_1\boldsymbol{v}_1), \tag{1.36}$$

$$\boldsymbol{v}_3 = z_3(\underline{A\boldsymbol{v}_2} - d_1\boldsymbol{v}_1 - d_2\boldsymbol{v}_2). \tag{1.37}$$

Unlike the Gram–Schmidt process, $A^2\boldsymbol{v}$ is not computed. In the following, the scalar values are determined. Let $z_1 = 1/(\boldsymbol{v}, \boldsymbol{v})^{1/2} \in \mathbb{R}$, or equivalently $1/\|\boldsymbol{v}\|$. Then $\|\boldsymbol{v}_1\| = 1$ because $\|\boldsymbol{v}_1\| = (z_1\boldsymbol{v}, z_1\boldsymbol{v})^{1/2} = (z_1^2(\boldsymbol{v}, \boldsymbol{v}))^{1/2} = z_1(\boldsymbol{v}, \boldsymbol{v})^{1/2} = 1$. Next consider determining c_1 such that $(\boldsymbol{v}_1, \boldsymbol{v}_2) = 0$, i.e.,

$$\begin{aligned}0 &= (\boldsymbol{v}_1, \boldsymbol{v}_2)\\ &= (\boldsymbol{v}_1, z_2(A\boldsymbol{v}_1 - c_1\boldsymbol{v}_1))\\ &= z_2((\boldsymbol{v}_1, A\boldsymbol{v}_1) - c_1(\boldsymbol{v}_1, \boldsymbol{v}_1)) = z_2((\boldsymbol{v}_1, A\boldsymbol{v}_1) - c_1).\end{aligned}$$

Thus $c_1 = (\boldsymbol{v}_1, A\boldsymbol{v}_1)$, and z_2 is determined by $\|\boldsymbol{v}_2\| = 1$, i.e., $z_2 = 1/\|A\boldsymbol{v}_1 - c_1\boldsymbol{v}_1\|$. Similarly, scaler values d_1, d_2 in (1.37) are determined by $(\boldsymbol{v}_1, \boldsymbol{v}_3) = (\boldsymbol{v}_2, \boldsymbol{v}_3) = 0$, and z_3 is determined by $\|\boldsymbol{v}_3\| = 1$.

The Arnoldi process (classical Gram–Schmidt type) is given in Algorithm 1.7 and a more accurate variant of the Arnoldi process (modified Gram–Schmidt type) is described in Algorithm 1.8.

The Arnoldi process can be expressed in matrix form. As an example, we consider a matrix form of the recurrences (1.36) and (1.37). From (1.36) and (1.37) we have

$$\begin{aligned}A\boldsymbol{v}_1 &= z_2^{-1}\boldsymbol{v}_2 + c_1\boldsymbol{v}_1,\\ A\boldsymbol{v}_2 &= z_3^{-1}\boldsymbol{v}_3 + d_1\boldsymbol{v}_1 + d_2\boldsymbol{v}_2,\end{aligned}$$

Algorithm 1.7 The Arnoldi process of classical Gram–Schmidt type

```
Input: Non-Hermitian matrix $A \in \mathbb{C}^{N \times N}$ and choose $\boldsymbol{v}_1$ such that $\|\boldsymbol{v}_1\| = 1$
Output: $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_N$
1: for $n = 1, 2, \dots, N-1$ do
2:     $h_{i,n} = (\boldsymbol{v}_i, A\boldsymbol{v}_n), \quad i = 1, 2, \dots, n$
3:     $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \sum_{i=1}^{n} h_{i,n}\boldsymbol{v}_i$
4:     $h_{n+1,n} = \|\tilde{\boldsymbol{v}}_{n+1}\|$
5:     $\boldsymbol{v}_{n+1} = \frac{\tilde{\boldsymbol{v}}_{n+1}}{h_{n+1,n}}$
6: end for
```

Algorithm 1.8 The Arnoldi process of modified Gram–Schmidt type

```
Input: Non-Hermitian matrix $A \in \mathbb{C}^{N \times N}$ and choose $\boldsymbol{v}_1$ such that $\|\boldsymbol{v}_1\| = 1$
Output: $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_N$
1: for $n = 1, 2, \dots, N-1$ do
2:     $\boldsymbol{t} = A\boldsymbol{v}_n$
3:     for $i = 1, 2, \dots, n$ do
4:         $h_{i,n} = (\boldsymbol{v}_i, \boldsymbol{t})$
5:         $\boldsymbol{t} = \boldsymbol{t} - h_{i,n}\boldsymbol{v}_i$
6:     end for
7:     $h_{n+1,n} = \|\boldsymbol{t}\|$
8:     $\boldsymbol{v}_{n+1} = \frac{\boldsymbol{t}}{h_{n+1,n}}$
9: end for
```

or equivalently

$$A[\boldsymbol{v}_1, \boldsymbol{v}_2] = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3] \begin{bmatrix} c_1 & d_1 \\ z_2^{-1} & d_2 \\ 0 & z_3^{-1} \end{bmatrix}.$$

Note that $A[\boldsymbol{v}_1, \boldsymbol{v}_2] = [A\boldsymbol{v}_1, A\boldsymbol{v}_2]$. Similarly, the matrix form of Algorithm 1.7 (or Algorithm 1.8) can be obtained as described next.

Let V_n be the $N \times n$ matrix whose columns are the first n orthonormal vectors given in Algorithm 1.7 (or Algorithm 1.8), i.e.,

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

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

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

and

$$H_n = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n-1} & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n-1} & h_{2,n} \\ & \ddots & \ddots & \vdots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & h_{n,n-1} & h_{n,n} \end{bmatrix},$$

where $h_{i,j}$'s are scalar values that are produced by Algorithm 1.7 (or Algorithm 1.8). Then, we have

$$AV_n = V_{n+1}H_{n+1,n} = V_nH_n + h_{n+1,n}\boldsymbol{v}_{n+1}\boldsymbol{e}_n^\top, \tag{1.38}$$

where $\boldsymbol{e}_n = (0, 0, \dots, 0, 1)^\top \in \mathbb{R}^n$. From the above matrix form and $V_n^{\mathrm{H}}V_n = I_n$, we obtain the relation of the form

$$H_n = V_n^{\mathrm{H}}AV_n. \tag{1.39}$$

If A is Hermitian, from $H_n^{\mathrm{H}} = (V_n^{\mathrm{H}}AV_n)^{\mathrm{H}} = V_n^{\mathrm{H}}A^{\mathrm{H}}V_n = V_n^{\mathrm{H}}AV_n = H_n$ we see that the Hessenberg matrix H_n is Hermitian. Thus, in this case, H_n is a tridiagonal matrix. This means that if A is Hermitian, the Arnoldi process has a short-term recurrences relation, which leads to the Lanczos process. The details of the Lanczos process will be described in Sect. 1.9.4.

1.9.2 The Bi-Lanczos Process

The bi-Lanczos process is an algorithm for obtaining a bi-orthogonal basis of $K_n(A, \boldsymbol{v}_0)$ 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 requirements. The algorithm is given in Algorithm 1.9.

If breakdown does not occur, Algorithm 1.9 generates bi-orthogonal basis vectors of the two Krylov subspaces such that

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

where $K_n(A, \boldsymbol{v}_1) = \mathrm{span}\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ and $K_n(A^{\mathrm{H}}, \boldsymbol{w}_1) = \mathrm{span}\{\boldsymbol{w}_1, \dots, \boldsymbol{w}_n\}$.

Algorithm 1.9 The bi-Lanczos process

```
Input: Non-Hermitian matrix $A \in \mathbb{C}^{N\times N}$ and choose $\boldsymbol{v}_0$ and $\boldsymbol{w}_0$ such that $(\boldsymbol{v}_0, \boldsymbol{w}_0) \neq 0$
Input: $\boldsymbol{v}_1 = \boldsymbol{v}_0/\|\boldsymbol{v}_0\|$, $\boldsymbol{w}_1 = \boldsymbol{w}_0/(\boldsymbol{w}_0, \boldsymbol{v}_1)$
Output: $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_N$ and $\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_N$
1: for $n = 1, 2, \dots, N-1$ do
2:    $\alpha_n = (\boldsymbol{w}_n, A\boldsymbol{v}_n)$,
3:    $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n\boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$
4:    $\tilde{\boldsymbol{w}}_{n+1} = A^{\mathrm{H}}\boldsymbol{w}_n - \overline{\alpha}_n\boldsymbol{w}_n - \gamma_{n-1}\boldsymbol{w}_{n-1}$
5:    $\gamma_n = \|\tilde{\boldsymbol{v}}_{n+1}\|$
6:    $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\gamma_n$
7:    $\beta_n = (\tilde{\boldsymbol{w}}_{n+1}, \boldsymbol{v}_{n+1})$
8:    $\boldsymbol{w}_{n+1} = \tilde{\boldsymbol{w}}_{n+1}/\overline{\beta}_n$
9: end for
```

Similar to the Arnoldi process, the bi-Lanczos process can also be 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.,

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

Then from the bi-Lanczos process, we have

$$\begin{aligned} AV_n &= V_{n+1}T_{n+1,n} = V_nT_n + \gamma_n\boldsymbol{v}_{n+1}\boldsymbol{e}_n^\top, \\ A^{\mathrm{H}}W_n &= W_{n+1}T_{n+1,n}^{\mathrm{H}} = W_nT_n^{\mathrm{H}} + \overline{\beta}_n\boldsymbol{w}_{n+1}\boldsymbol{e}_n^\top. \end{aligned} \tag{1.40}$$

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

1.9.3 The Complex Symmetric Lanczos Process

The complex symmetric Lanczos process is a special case for the bi-Lanczos process (Algorithm 1.9) when the coefficient matrix is complex symmetric, i.e., $A = A^\top \neq A^{\mathrm{H}}$. If A is complex symmetric, we can readily derive the process from Algorithm 1.9 by setting $\boldsymbol{w}_0 = \overline{\boldsymbol{v}}_0$, and the resulting algorithm is given in Algorithm 1.10.

If breakdown does not occur, Algorithm 1.10 generates conjugate orthogonal basis vectors of the Krylov subspace such that

Algorithm 1.10 The complex symmetric Lanczos process (note: $(\overline{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: Complex symmetric matrix $A \in \mathbb{C}^{N\times N}$ and choose $\boldsymbol{v}_1$ such that $(\overline{\boldsymbol{v}}_1, \boldsymbol{v}_1) \neq 0$
Input: $\boldsymbol{v}_1 = \boldsymbol{v}_1/(\overline{\boldsymbol{v}}_1, \boldsymbol{v}_1)^{1/2}$
Output: $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N$
1: for $n = 1, 2, \ldots, N-1$ do
2:   $\alpha_n = (\overline{\boldsymbol{v}}_n, A\boldsymbol{v}_n)$
3:   $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n \boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$
4:   $\beta_n = (\overline{\tilde{\boldsymbol{v}}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$
5:   $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\beta_n$
6: end for
```

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

where $K_n(A, \boldsymbol{v}_1) = \mathrm{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$.

Similar to the Arnoldi process and the bi-Lanczos process, Algorithm 1.10 can also be 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.,

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

Then from Algorithm 1.10, we have

$$AV_n = V_{n+1}T_{n+1,n} = V_nT_n + \beta_n \boldsymbol{v}_{n+1}\boldsymbol{e}_n^\top. \tag{1.41}$$

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

1.9.4 The Lanczos Process

The Lanczos process 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 the Arnoldi process (Algorithm 1.7), 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 more concise than the corresponding step of the Arnoldi process or the bi-Lanczos process. The Lanczos process is listed in Algorithm 1.11.

Algorithm 1.11 The Lanczos process

```
Input: Hermitian matrix $A \in \mathbb{C}^{N\times N}$ and choose $\boldsymbol{v}_1$ such that $\|\boldsymbol{v}_1\| = 1$
Input: $\beta_0 = 0$, $\boldsymbol{v}_0 = \mathbf{0}$
Output: $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N$
1: for $n = 1, 2, \ldots, N-1$ do
2:     $\alpha_n = (\boldsymbol{v}_n, A\boldsymbol{v}_n)$
3:     $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n \boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$
4:     $\beta_n = (\tilde{\boldsymbol{v}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$
5:     $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\beta_n$
6: end for
```

We can readily derive the Lanczos process from the Arnoldi process or the bi-Lanczos process with setting $\boldsymbol{w}_0 = \boldsymbol{v}_0$. If breakdown does not occur, the above algorithm generates an orthonormal basis of $K_n(A, \boldsymbol{v}_1)$ such that

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

where $K_n(A, \boldsymbol{v}_1) = \mathrm{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$.

Similar to the Arnoldi process and the bi-Lanczos process, the Lanczos process can also be 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.,

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

Then from the Lanczos process, we have

$$AV_n = V_{n+1}T_{n+1,n} = V_nT_n + \beta_n \boldsymbol{v}_{n+1}\boldsymbol{e}_n^\top. \tag{1.42}$$

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

$$T_n = V_n^{\mathrm{H}} A V_n. \tag{1.43}$$

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

Chapter 2
Some Applications to Computational Science and Data Science

Abstract Linear systems of the form $Ax = b$ arise in a rich variety of applications such as computational science and data science. For computational science, physical phenomena are often described as partial differential equations. For solving partial differential equations, the finite-difference methods and the finite element method are widely used, leading to a problem of solving linear systems. Large-scale electronic structure calculation for condensed matter physics and lattice quantum chromodynamics for particle physics require solving large-scale linear systems. For data science, regression and classification are fundamental tasks that also require solving linear systems. Minimizing or maximizing functions is one of the most important optimization problems, which requires solving linear systems when Newton's method is used. We will see in this chapter how the linear systems arise in these applications by using simple and concrete examples.

2.1 Partial Differential Equations

In this section, the finite difference method and the finite element method are described using some partial differential equations, and we will see that the more accurate solution we need, the larger the linear systems we need to solve.

2.1.1 Finite Difference Methods

This subsection describes the finite difference methods using several simple partial differential equations. The notion of tensor products is also explained, which comes in handy to describe the coefficient matrix of the linear systems arising from the finite difference discretization of the partial differential equations. First of all, a mathematical preliminary is given next.

T. Sogabe, *Krylov Subspace Methods for Linear Systems*,
Springer Series in Computational Mathematics 60,
https://doi.org/10.1007/978-981-19-8532-4_2

2.1.1.1 Preliminary

This subsection describes several formulas for approximating derivatives $f'(x), f''(x)$ of a smooth function $f(x)$ at x using $f(x), f(x-h)$, and $f(x+h)$ with $0 < h < 1$.

Consider how to approximate $f'(x)$. The Taylor series of $f(x)$ is given by

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2!}f''(x)h^2 + \frac{1}{3!}f'''(x)h^3 + \cdots . \tag{2.1}$$

Then, $f'(x)$ is written as

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x)}{h} - \frac{1}{2!}f''(x)h - \frac{1}{3!}f'''(x)h^2 - \cdots \\ &= \frac{f(x+h) - f(x)}{h} + O(h). \end{aligned}$$

$O(\cdot)$ is the big O notation. Roughly speaking, $O(h)$ in this case means that $-\frac{1}{2!}f''(x)$ $h - \frac{1}{3!}f'''(x)h^2 - \cdots$ can be written as ch for a suitable constant value c as $h \to 0$. The approximation to $f'(x)$ by the recurrence

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{2.2}$$

is referred to as the *forward difference*. The accuracy of the forward difference is of order h. A similar approximation

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \tag{2.3}$$

is referred to as the *backward difference*. The accuracy of the backward difference is also of order h. For a more accurate formula, it follows from (2.1) that we have

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2),$$

from which, we have

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}, \tag{2.4}$$

and this is referred to as the *central difference* for the first derivative of $f(x)$, whose accuracy is of order h^2.

We now consider approximating the second derivative $f''(x)$. From the Taylor series (2.1) of $f(x+h)$ and $f(x-h)$, we have

$$f(x+h) - 2f(x) + f(x-h) = f''(x)h^2 + O(h^4).$$

Thus

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2).$$

The following approximation

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{2.5}$$

is referred to as the *central difference* for the second derivative of $f(x)$, whose accuracy is of order h^2.

2.1.1.2 Example 1 (Symmetric Matrix)

Consider the Laplace equation of the form

$$\begin{aligned} &u_{xx} + u_{yy} = 0, \qquad (x, y) \in (0, 1) \times (0, 1), \\ &u(x, y) = g(x, y), \qquad (x, y) \in \Gamma, \end{aligned}$$

where Γ is the boundary of $(0, 1) \times (0, 1)$. To be specific, the boundary condition is given by $u(0, y) = g(0, y)$, $u(1, y) = g(1, y)$, $u(x, 0) = g(x, 0)$, and $u(x, 1) = g(x, 1)$.

We now approximately solve the Laplace equation by using central difference (2.5). To this end, an equispaced grid of $N \times N$ points in $(0, 1) \times (0, 1)$ is used. The case $N = 3$ ($h = 1/4$) is shown in Fig. 2.1.

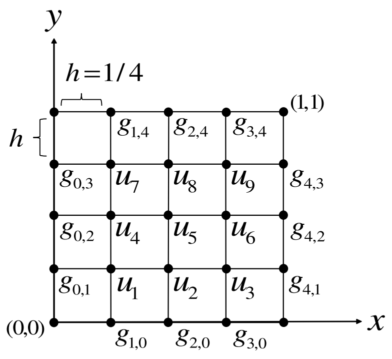


Fig. 2.1 Mesh grid on the region for example 1 (the Laplace equation)

In the following, we use the mesh in Fig. 2.1 and we will see how linear systems are obtained. Central difference (2.5) for approximating u_{xx} and u_{yy} is given by

$$u_{xx}(x,y) \approx \frac{1}{h^2}(u(x+h,y) - 2u(x,y) + u(x-h,y)), \tag{2.6}$$
$$u_{yy}(x,y) \approx \frac{1}{h^2}(u(x,y+h) - 2u(x,y) + u(x,y-h)). \tag{2.7}$$

It is convenient to introduce an abbreviated notation $u_{i,j} = u(x_i, y_j)$. Then we obtain the following five-point formula:

$$u_{xx}(x_i,y_j) + u_{yy}(x_i,y_j) \approx \frac{1}{h^2}(u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1}), \tag{2.8}$$

where $u(x_i,y_j) = g(x_i,y_j)$ on the boundary. Let $u_1 := u(x_1,y_1)$, $u_2 := u(x_1,y_2)$,..., $u_9 := u(x_3,y_3)$ and $g_{i,j} := u(x_i,y_j)$ on the boundary. Then from Fig. 2.1 and (2.8), unknowns $u_1, u_2, \ldots, u_9$ have the following relations:

$$\begin{aligned}
&u_2 + g_{0,1} - 4u_1 + u_4 + g_{1,0} = 0,\\
&u_3 + u_1 - 4u_2 + u_5 + g_{2,0} = 0,\\
&g_{4,1} + u_2 - 4u_3 + u_6 + g_{3,0} = 0,\\
&u_5 + g_{0,2} - 4u_4 + u_7 + u_1 = 0,\\
&\vdots\\
&g_{4,3} + u_8 - 4u_9 + g_{3,4} + u_6 = 0.
\end{aligned}$$

The corresponding matrix form is

$$\underbrace{\left[\begin{array}{ccc|ccc|ccc}
-4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\
1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0\\
0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0\\
\hline
1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0\\
0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0\\
0 & 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1\\
\hline
0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4
\end{array}\right]}_{A}
\underbrace{\begin{bmatrix} u_1\\ u_2\\ u_3\\ u_4\\ u_5\\ u_6\\ u_7\\ u_8\\ u_9 \end{bmatrix}}_{x}
= \underbrace{-\begin{bmatrix} g_{1,0}\\ g_{2,0}\\ g_{3,0}\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0 \end{bmatrix}
- \begin{bmatrix} 0\\ 0\\ 0\\ 0\\ 0\\ 0\\ g_{1,4}\\ g_{2,4}\\ g_{3,4} \end{bmatrix}
- \begin{bmatrix} g_{0,1}\\ 0\\ 0\\ g_{0,2}\\ 0\\ 0\\ g_{0,3}\\ 0\\ 0 \end{bmatrix}
- \begin{bmatrix} 0\\ 0\\ g_{4,1}\\ 0\\ 0\\ g_{4,2}\\ 0\\ 0\\ g_{4,3} \end{bmatrix}}_{b}. \tag{2.9}$$

By solving the linear systems, we obtain the approximate solution $u_1, \ldots, u_9$ on the mesh grid.

The matrix of the linear systems (2.9) has a special nonzero structure, which can be rewritten in a simple form using Kronecker product. Let $A(= \{a_{ij}\})$ and B are matrices. Then, the symbol $A \otimes B$ is called the *Kronecker product* (or *Tensor product*), defined by

$$A \otimes B := \begin{bmatrix} a_{11}B & \cdots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nm}B \end{bmatrix}. \tag{2.10}$$

Below is an example for A, B being 2-by-2 matrices:

$$A \otimes B = \left[\begin{array}{c|c} a_{11}B & a_{12}B \\ \hline a_{21}B & a_{22}B \end{array}\right] = \left[\begin{array}{cc|cc} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{array}\right].$$

Now, let

$$M = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}, \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Then, the coefficient matrix A in (2.9) can be written as

$$A = I \otimes M + M \otimes I. \tag{2.11}$$

The size of the matrix A is $3^2 \times 3^2$. In general, $N \times N$ grid yields an $N^2 \times N^2$ matrix, which can be very large and symmetric. Krylov subspace methods for symmetric linear systems are described in Sect. 3.1.

Finally, some properties of the Kronecker product are listed below.

Theorem 2.1 *The Kronecker product has the following properties:*

(1) *For* $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{p\times q}$, $C \in \mathbb{C}^{r\times s}$,

$$(A \otimes B) \otimes C = A \otimes (B \otimes C).$$

(2) *For* $A, B \in \mathbb{C}^{m\times n}$, $C \in \mathbb{C}^{p\times q}$,

$$(A + B) \otimes C = A \otimes C + B \otimes C.$$

(3) *For* $A \in \mathbb{C}^{m\times n}$, $B, C \in \mathbb{C}^{p\times q}$,

$$A \otimes (B + C) = A \otimes B + A \otimes C.$$

(4) *For* $c \in \mathbb{C}, A \in \mathbb{C}^{m\times n}, B \in \mathbb{C}^{p\times q}$,

$$(cA) \otimes B = A \otimes (cB) = c(A \otimes B).$$

(5) *For* $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{p\times q}$,

$$(A \otimes B)^{\mathrm{H}} = A^{\mathrm{H}} \otimes B^{\mathrm{H}}.$$

(6) *For* $A \in \mathbb{C}^{m \times n}, B \in \mathbb{C}^{p \times q}, C \in \mathbb{C}^{n \times r}, D \in \mathbb{C}^{q \times s}$,

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD).$$

Proof We only give a proof of (6). The (i, k) block element of $A \otimes B$ is $a_{i,k}B$, and the (k, j) block element of $C \otimes D$ is $c_{k,j}D$. Thus the (i, j) block element of $(A \otimes B)(C \otimes D)$ is given as

$$\begin{aligned}((A \otimes B)(C \otimes D))_{i,j} &= \sum_{k=1}^{n}(a_{i,k}B)(c_{k,j}D) = \left(\sum_{k=1}^{n} a_{i,k}c_{k,j}\right) BD \\ &= (AC)_{i,j}BD = ((AC) \otimes (BD))_{i,j},\end{aligned}$$

which concludes the proof. □

Note that (i, j) "block" element is not a scalar but a matrix. For example, consider the following 2-by-2 block matrices of the form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

Then the (1,1) block element of A is matrix A_{11}, and the (1,2) block element of B is matrix B_{12}. Furthermore, we can calculate AB as if the submatrices $A_{i,j}$ and $B_{i,j}$ are scalars when the matrix multiplication $A_{i,j}B_{j,k}$ is defined. For example, The (1,1) block element of AB is given by $A_{11}B_{11} + A_{12}B_{21}$.

From Theorem 2.1-(6), the following properties hold true.

Corollary 2.1 *Let A and B be invertible matrices. Then*

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}.$$

Proof From Theorem 2.1-(6), we have $(A^{-1} \otimes B^{-1})(A \otimes B) = (A^{-1}A) \otimes (B^{-1}B) = I \otimes I$, which is the Kronecker product of the two identity matrices. It is easy to see that the Kronecker product of the two identity matrices is the identity matrix. □

Corollary 2.2 *Let* $\boldsymbol{p}_i^{(A)}$ *and* $\boldsymbol{p}_j^{(B)}$ *be eigenvectors of A and B, and the corresponding eigenvalues are* $\lambda_i^{(A)}$ *and* $\lambda_j^{(B)}$. *Then an eigenvector of* $A \otimes B$ *is* $\boldsymbol{p}_i^{(A)} \otimes \boldsymbol{p}_j^{(B)}$, *and the corresponding eigenvalue is* $\lambda_i^{(A)}\lambda_j^{(B)}$, *i.e.,*

$$(A \otimes B)\left(\boldsymbol{p}_i^{(A)} \otimes \boldsymbol{p}_j^{(B)}\right) = \lambda_i^{(A)}\lambda_j^{(B)}\left(\boldsymbol{p}_i^{(A)} \otimes \boldsymbol{p}_j^{(B)}\right).$$

Proof From Theorem 2.1-(4) and (6), we have

$$(A \otimes B)\left(\boldsymbol{p}_i^{(A)} \otimes \boldsymbol{p}_j^{(B)}\right) = \left(A\boldsymbol{p}_i^{(A)}\right) \otimes \left(B\boldsymbol{p}_j^{(B)}\right) = \lambda_i\lambda_j \left(\boldsymbol{p}_i^{(A)} \otimes \boldsymbol{p}_j^{(B)}\right),$$

which concludes the proof. □

Corollary 2.3 *Using the same notations as in Corollary 2.2, it follows that*

$$(A \otimes I + I \otimes B)\left(\boldsymbol{p}_i^{(A)} \otimes \boldsymbol{p}_j^{(B)}\right) = (\lambda_i^{(A)} + \lambda_j^{(B)})\left(\boldsymbol{p}_i^{(A)} \otimes \boldsymbol{p}_j^{(B)}\right),$$

where I is the identity matrix.

The matrix $A \otimes I + I \otimes B$ is referred to as a *Kronecker sum* (or *Tensor sum*).

It follows from Corollary 2.3 that eigenvectors and eigenvalues of matrix A in (2.11) can be given by $(\boldsymbol{p}_i \otimes \boldsymbol{p}_j, \lambda_i + \lambda_j)$, where $\boldsymbol{p}_i$ and λ_i is the eigenvector and eigenvalue of M. This indicates that matrix A in (2.11) is not invertible if and only if there exist i, j such that $\lambda_i = -\lambda_j$. Thus, the distribution of eigenvalues of much smaller matrices M than A determines whether matrix A is invertible or not.

In general, the Kronecker product does not commute, i.e.,

$$A \otimes B \neq B \otimes A,$$

where $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{p\times q}$. However, there are permutation matrices $P_{mp} \in \mathbb{R}^{mp\times mp}$ and $P_{nq} \in \mathbb{R}^{nq\times nq}$ such that

$$P_{mp}^\top (A \otimes B) P_{nq} = B \otimes A.$$

We describe this fact in detail as given next.

Theorem 2.2 *Let $P_{mn} \in \mathbb{R}^{mn\times mn}$ be a matrix of the form*

$$P_{mn} = \begin{bmatrix} (E_{11})^\top & (E_{12})^\top & \dots & (E_{1n})^\top \\ (E_{21})^\top & (E_{22})^\top & \dots & (E_{2n})^\top \\ \vdots & \vdots & \ddots & \vdots \\ (E_{m1})^\top & (E_{m2})^\top & \dots & (E_{mn})^\top \end{bmatrix}, \tag{2.12}$$

where $E_{i,j} \in \mathbb{R}^{m\times n}$ is a matrix whose (i, j) element is one and the other elements are zeros. Then,

$$P_{mn} = P_{nm}^\top = P_{nm}^{-1}$$

and for $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{p\times q}$,

$$\underbrace{P_{mp}^\top}_{mp\times mp} \underbrace{(A \otimes B)}_{mp\times nq} \underbrace{P_{nq}}_{nq\times nq} = B \otimes A. \tag{2.13}$$

If A and B are square matrices (i.e., $m = n$, $p = q$),

$$P_{mp}^{\top}(A \otimes B)P_{mp} = B \otimes A.$$

Proof We give a proof of (2.13). The (i, j) block element of $P_{m,n}$ in (2.12) is $E_{ij}^{\top}$, and E_{ij} can be factorized as

$$E_{ij} = \boldsymbol{e}_i \boldsymbol{e}_j^{\top},$$

where $\boldsymbol{e}_i$ is the ith unit vector, i.e., the ith element is one, and the others are zeros. Let a_{ij} and b_{ij} be the (i, j) elements of A and B. Then, from (2.12), the (i, j) block element of $P_{mp}^{\top}(A \otimes B)P_{nq}$ is calculated as

$$\begin{aligned}
\left(P_{mp}^{\top}(A \otimes B)P_{nq}\right)_{ij} &= \sum_{k,\ell} E_{ki}(a_{k\ell}B)E_{\ell j}^{\top} = \sum_{k,\ell} a_{k\ell}\left(E_{ki}BE_{j\ell}\right) \\
&= \sum_{k,\ell} a_{k\ell}\left(\boldsymbol{e}_k \boldsymbol{e}_i^{\top} B \boldsymbol{e}_j \boldsymbol{e}_\ell^{\top}\right) = \sum_{k,\ell} a_{k\ell}\left(\boldsymbol{e}_k b_{ij} \boldsymbol{e}_\ell^{\top}\right) \\
&= b_{ij}\sum_{k,\ell} a_{k\ell}\left(\boldsymbol{e}_k \boldsymbol{e}_\ell^{\top}\right) = b_{ij}\left(\sum_{k,\ell} a_{k\ell}E_{k\ell}\right) = b_{ij}A \\
&= (B \otimes A)_{ij},
\end{aligned}$$

which concludes the proof. □

2.1.1.3 Example 2 (Complex Symmetric Matrix)

We consider the following boundary value problem:

$$\begin{aligned}
&u_{xx} + u_{yy} + \sigma^2 u = 0, && (x, y) \in (0, \pi) \times (0, \pi), \\
&u_x|_{x=0} = i\sqrt{\sigma^2 - \tfrac{1}{4}}\cos\tfrac{y}{2}, && \text{Neumann Condition (1)}, \\
&u_x|_{x=\pi} - i\sqrt{\sigma^2 - \tfrac{1}{4}}u|_{x=\pi} = 0, && \text{Radiation Condition}, \\
&u_y|_{y=0} = 0, && \text{Neumann Condition (2)}, \\
&u|_{y=\pi} = 0, && \text{Dirichlet Condition}.
\end{aligned}$$

The elliptic partial differential equation is known as the Helmholtz equation, see, e.g., [19]. For simplicity, we use a 3-by-3 mesh grid as shown in Figs. 2.2 and 2.3.

Similar to Example 1 in the previous section, applying (2.6)–(2.8) to the Helmholtz equation yields

$$-u_{i+1,j} - u_{i-1,j} + (4 - h^2\sigma^2)u_{i,j} - u_{i,j+1} - u_{i,j-1} = 0. \tag{2.14}$$

Considering Eq. (2.14) over the mesh points given in Fig. 2.3 leads to the following six equations:

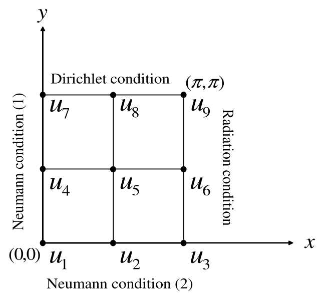
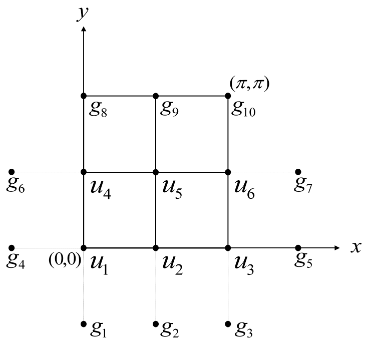


Fig. 2.2 Mesh points on the region



y
(π,π)
g_8 g_9 g_{10}
g_6 u_4 u_5 u_6 g_7
g_4 (0,0) u_1 u_2 u_3 g_5
x
g_1 g_2 g_3

Fig. 2.3 Mesh points on the region satisfying the boundary conditions

$$\begin{cases} -u_2 - g_4 - u_4 - g_1 + (4 - h^2\sigma^2)u_1 = 0, \\ -u_3 - u_1 - u_5 - g_2 + (4 - h^2\sigma^2)u_2 = 0, \\ -g_5 - u_2 - u_6 - g_3 + (4 - h^2\sigma^2)u_3 = 0, \\ -u_5 - g_6 - g_8 - u_1 + (4 - h^2\sigma^2)u_4 = 0, \\ -u_6 - u_4 - g_9 - u_2 + (4 - h^2\sigma^2)u_5 = 0, \\ -g_7 - u_5 - g_{10} - u_3 + (4 - h^2\sigma^2)u_3 = 0, \end{cases} \tag{2.15}$$

where $u_1 := u_{1,1} = u(0,0), u_2 := u_{2,1} = u(h,0), u_3 := u_{3,1} = u(2h,0), u_4 := u_{1,2} = u(0,h),\ u_5 := u_{2,2} = u(h,h),\ u_6 := u_{3,2} = u(2h,h)$ and $h = \pi/2$. Here g_k's for $k = 1, \ldots, 10$ in Fig. 2.3 can be computed by considering the four boundary conditions as follows: first, from Neumann Condition (2) it follows that

$$\frac{u_4 - g_1}{2h} = \frac{u_5 - g_2}{2h} = \frac{u_6 - g_3}{2h} = 0,$$

and thus

$$g_1 = u_4, \quad g_2 = u_5, \quad g_3 = u_6.$$

Second, from Neumann Condition (1), we have

$$u_x(0,0) \approx \frac{u_2 - g_4}{2h} = id \cos\left(\frac{0}{2}\right), \quad u_x(0,h) \approx \frac{u_5 - g_6}{2h} = id \cos\left(\frac{1}{2} \times \frac{h}{2}\right).$$

or equivalently,

$$g_4 = u_2 - 2idh, \quad g_6 = u_5 - 2idh \cos\left(\frac{h}{4}\right),$$

where $d = \sqrt{\sigma^2 - 1/4}$. Third, from the Radiation Condition we obtain

$$\frac{g_5 - u_2}{2h} - idu_3 = \frac{g_7 - u_5}{2h} - idu_6 = 0$$

or equivalently,

$$g_5 = u_2 + 2idhu_3, \quad g_7 = u_5 + 2idhu_6.$$

Finally, it follows from the Dirichlet Condition that g_8, g_9 and g_{10} satisfy

$$g_8 = g_9 = g_{10} = 0.$$

Substituting $g_1, g_2, \ldots, g_{10}$ into the six equations (2.15) yields the following linear systems:

$$\begin{bmatrix} a & -2 & 0 & -2 & 0 & 0 \\ -1 & a & -1 & 0 & -2 & 0 \\ 0 & -2 & a - idh & 0 & 0 & -2 \\ -1 & 0 & 0 & a & -2 & 0 \\ 0 & -1 & 0 & -1 & a & -1 \\ 0 & 0 & -1 & 0 & -2 & a - idh \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} f_1 \\ 0 \\ 0 \\ f_4 \\ 0 \\ 0 \end{bmatrix}, \tag{2.16}$$

where $a = \sqrt{\sigma^2 - 1/4}$, $f_1 = -2idh$, and $f_4 = -2idh \cos(h/4)$. The coefficient matrix of the linear systems (2.16) is not symmetric. By scaling each equation, we obtain symmetric linear systems of the form

$$\begin{bmatrix} a/4 & -1/2 & 0 & -1/2 & 0 & 0 \\ -1/2 & a/2 & -1/2 & 0 & -1 & 0 \\ 0 & -1/2 & (a - idh)/4 & 0 & 0 & -1/2 \\ -1/2 & 0 & 0 & a/2 & -1 & 0 \\ 0 & -1 & 0 & -1 & a & -1 \\ 0 & 0 & -1/2 & 0 & -1 & (a - idh)/2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} f_1/4 \\ 0 \\ 0 \\ f_4/2 \\ 0 \\ 0 \end{bmatrix}. \tag{2.17}$$

The coefficient matrix of the form (2.17) is non-Hermitian but symmetric, i.e., complex symmetric. Krylov subspace methods for solving complex symmetric linear systems are described in Sect. 3.2.

2.1.1.4 Example 3 (Nonsymmetric Matrix)

Consider the following three-dimensional partial differential equation:

$$\begin{array}{ll} [\boldsymbol{a} \cdot (\nabla * \nabla) + \boldsymbol{b} \cdot \nabla + c]u(x, y, z) = g(x, y, z), & \text{in } \Omega, \\ u(x, y, z) = 0, & \text{on } \partial\Omega. \end{array}$$

Here, Ω is the cubic domain, i.e., $(0, 1) \times (0, 1) \times (0, 1)$ and $\partial\Omega$ is the boundary. The symbol "*" denotes element-wise multiplication. Equivalently, the operator $\boldsymbol{a} \cdot (\nabla * \nabla) + \boldsymbol{b} \cdot \nabla + c$ is rewritten as

$$a_1 \frac{\partial^2}{\partial x^2} + a_2 \frac{\partial^2}{\partial y^2} + a_3 \frac{\partial^2}{\partial z^2} + b_1 \frac{\partial}{\partial x} + b_2 \frac{\partial}{\partial y} + b_3 \frac{\partial}{\partial z} + c$$

with $\boldsymbol{a} = [a_1, a_2, a_3]^\top$, $\boldsymbol{b} = [b_1, b_2, b_3]^\top$, and $\boldsymbol{c} = [c_1, c_2, c_3]^\top$.

Using central difference (2.5) for the second derivative of u yields

$$\begin{aligned} a_1 u_{xx}(x, y, z) &\approx \frac{a_1}{h^2}(u(x + h, y, z) - 2u(x, y, z) + u(x - h, y, z)), \\ a_2 u_{yy}(x, y, z) &\approx \frac{a_2}{h^2}(u(x, y + h, z) - 2u(x, y, z) + u(x, y - h, z)), \\ a_3 u_{zz}(x, y, z) &\approx \frac{a_3}{h^2}(u(x, y, z + h) - 2u(x, y, z) + u(x, y, z - h)). \end{aligned}$$

Similarly, using central difference (2.4) for the first derivative of u gives

$$\begin{aligned} b_1 u_x(x, y, z) &\approx \frac{b_1}{2h}(u(x + h, y, z) - u(x - h, y, z)), \\ b_2 u_y(x, y, z) &\approx \frac{b_2}{2h}(u(x, y + h, z) - u(x, y - h, z)), \\ b_3 u_z(x, y, z) &\approx \frac{b_3}{2h}(u(x, y, z + h) - u(x, y, z - h)). \end{aligned}$$

Considering an $(N + 1) \times (N + 1) \times (N + 1)$ grid, together with the central differences above, leads to

$$\begin{aligned} &[\boldsymbol{a} \cdot (\nabla * \nabla) + \boldsymbol{b} \cdot \nabla + c]u(x, y, z) \\ &\approx \frac{a_1}{h^2}(u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}) + \frac{b_1}{2h}(u_{i+1,j,k} - u_{i-1,j,k}) + cu_{i,j,k} \\ &\quad + \frac{a_2}{h^2}(u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}) + \frac{b_2}{2h}(u_{i,j+1,k} - u_{i,j-1,k}) \\ &\quad + \frac{a_3}{h^2}(u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}) + \frac{b_3}{2h}(u_{i,j,k+1} - u_{i,j,k-1}), \end{aligned}$$

where $h = 1/(N + 1)$. Since $u = 0$ on the boundary. Following Sect. 2.1.1.2, we have

$$(I_N \otimes I_N \otimes A + I_N \otimes B \otimes I_N + C \otimes I_N \otimes I_N)\boldsymbol{u} = \boldsymbol{g}, \tag{2.18}$$

where I_N is the N-by-N identity matrix, each element of $\boldsymbol{u}$ and $\boldsymbol{g}$ corresponds to $u(x_i, y_i, z_i)$ and $g(x_i, y_i, z_i)$, and

$$\begin{aligned} A &:= \frac{a_1}{h^2}M_1 + \frac{b_1}{2h}M_2 + cI_N, \\ B &:= \frac{a_2}{h^2}M_1 + \frac{b_2}{2h}M_2, \\ C &:= \frac{a_3}{h^2}M_1 + \frac{b_3}{2h}M_2. \end{aligned}$$

Here, M_1 and M_2 are defined by

$$M_1 := \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}, \quad M_2 := \begin{bmatrix} 0 & 1 & & & \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ & & & -1 & 0 \end{bmatrix}.$$

M_1 and M_2 are N-by-N (tridiagonal Toeplitz) matrices, and blanks in the matrices are zeros.

The size of linear systems (2.18) is $N^3 \times N^3$. If $N = 100$, which is an equispaced grid with $h \approx 0.01$, then the size of the matrix is a million. If $N = 1000$, then the size is now a billion! Therefore, efficient numerical solvers for solving very large (and sparse) linear systems are required for a well approximate solution to the partial differential equation.

The matrix in (2.18) is (real) nonsymmetric if $\boldsymbol{b} \neq \boldsymbol{0}$. Krylov subspace methods for nonsymmetric (non-Hermitian) linear systems are described in Sect. 3.3.

2.1.2 The Finite Element Method

In the previous section, we have seen that a finite difference approximation to (partial) derivatives yields linear recurrence relations, leading to linear systems by using the recurrence relations. The approximate solution corresponds to the values of given grid points, which are the elements of the solution vector for the linear systems. In this section, the *Finite Element Method* (FEM, hereafter) is introduced, and we will see how linear systems are obtained through a simple example. The advantages of the FEM over the finite difference method are (i) square (or rectangular) grid points on the domain are not necessarily needed, and thus it is suitable for not only square or rectangular but other various domains; (ii) there are many mathematical results for the error analysis based on functional analysis.

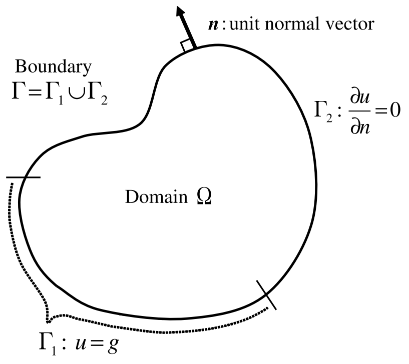

$\boldsymbol{n}$: unit normal vector
Boundary
$\Gamma = \Gamma_1 \cup \Gamma_2$
$\Gamma_2 : \frac{\partial u}{\partial n} = 0$
Domain Ω
$\Gamma_1 : u = g$

Fig. 2.4 The domain and the boundary of the Poisson Eqs. (2.19)–(2.21)

In this section, we consider applying the FEM to the following Poisson equation:

$$-\Delta u(x, y) = f, \quad (x, y) \in \Omega, \tag{2.19}$$

$$\cdot\ u(x, y) = g, \quad (x, y) \in \Gamma_1, \quad \text{Dirichlet Condition,} \tag{2.20}$$

$$\cdot\ \frac{\partial}{\partial n} u(x, y) = 0, \quad (x, y) \in \Gamma_2, \quad \text{Neumann Condition,} \tag{2.21}$$

where $\Delta := \partial^2/\partial x^2 + \partial^2/\partial y^2$. The domain and the boundary with the boundary conditions are illustrated in Fig. 2.4. In the following, the explanations are based on [118, 170].

2.1.3 Weak Form

In this section, an integral equation called the weak form is derived from the Poisson equation.

Consider integrating (2.19) multiplied by an arbitrary function over the domain Ω:

$$-\int_\Omega v \Delta u \, \mathrm{d}\Omega = \int_\Omega v f \, \mathrm{d}\Omega. \tag{2.22}$$

If we can find a function u that satisfies (2.22) for all v, then u is the solution of (2.19). The FEM tries to find the solution u that satisfies (2.22) instead of solving (2.19), which differs from the finite difference methods. Here, function v is referred to as the test function or the weight function.

Now, in order to simplify the mathematical formulas, let v be an arbitrary smooth function but $v = 0$ on Γ_1. Applying the Gauss–Green theorem to the left-hand side of (2.22), the integration by parts yields

$$
\begin{aligned}
\int_\Omega v \Delta u \, \mathrm{d}\Omega &= \int_\Gamma v \frac{\partial u}{\partial n} \, \mathrm{d}\Gamma - \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}\Omega \quad \text{(the Gauss–Green theorem)} \\
&= \int_{\Gamma_1} v \frac{\partial u}{\partial n} \, \mathrm{d}\Gamma_1 + \int_{\Gamma_2} v \frac{\partial u}{\partial n} \, \mathrm{d}\Gamma_2 - \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}\Omega \\
&= - \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}\Omega. \qquad (2.23)
\end{aligned}
$$

Here we used the assumption $v = 0$ on Γ_1 and Neumann condition (2.21) on Γ_2, i.e., $\frac{\partial u}{\partial n} = 0$ on Γ_2.

Substituting (2.23) into (2.22) gives

$$
\int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}\Omega = \int_\Omega vf \, \mathrm{d}\Omega.
$$

Thus all we have to do is to find u such that

$$
\iint_\Omega \left(\frac{\partial v}{\partial x} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \frac{\partial u}{\partial y} \right) \mathrm{d}x\mathrm{d}y = \iint_\Omega vf \, \mathrm{d}x\mathrm{d}y. \qquad (2.24)
$$

The equation does not have the second-order derivative, whereas the original Eq. (2.19) does. In other words, the condition of differentiability of u in (2.24) looks to be weak. Therefore, (2.24) is referred to as the weak form.

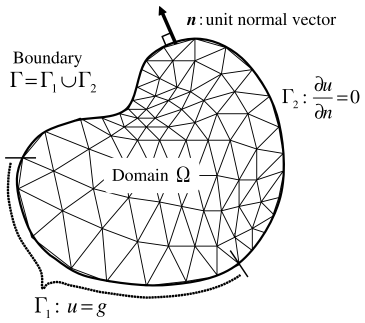
As seen in (2.24), the weak form is the integral over the whole domain. Now consider that the whole domain is divided into finite elements e_i, i.e., $\Omega = \cup_{i=1}^n e_i$ and $e_i \cap e_j = \emptyset$ for $i \neq j$. Then we have

$$
\iint_{e_i} \left(\frac{\partial v}{\partial x} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \frac{\partial u}{\partial y} \right) \mathrm{d}x\mathrm{d}y = \iint_{e_i} vf \, \mathrm{d}x\mathrm{d}y \qquad (2.25)
$$

for $i = 1, 2, \ldots, n$. An example of the domain decomposition by triangle elements is shown in Fig. 2.5. In the next section, we will see how linear systems are obtained from (2.25).

2.1.4 Derivation of Linear Systems

Let finite elements e_i in (2.25) be triangular, see Fig. 2.5. Each node of the triangular element e_i is given the position and a number, and the node numbers of the triangle are given clockwise, see Fig. 2.6. As seen in Fig. 2.6, the position of node i is (x_i, y_i) for $i = 1, 2, 3$.



$\boldsymbol{n}$: unit normal vector
Boundary
$\Gamma = \Gamma_1 \cup \Gamma_2$
$\Gamma_2 : \frac{\partial u}{\partial n} = 0$
Domain Ω
$\Gamma_1 : u = g$

Fig. 2.5 The domain decomposition by triangular finite elements

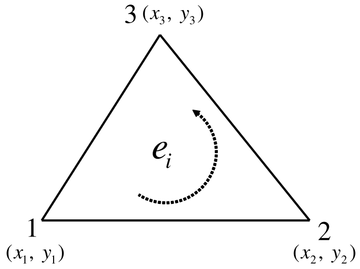


Fig. 2.6 The numbering of triangular finite elements

Now we introduce the following vector-valued function:

$$\boldsymbol{h}(x, y) = \begin{pmatrix} h_1(x, y) \\ h_2(x, y) \\ h_3(x, y) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ x \\ y \end{pmatrix}, \tag{2.26}$$

and let $\widetilde{u}$ be an approximate function to the solution function u over the triangular domain such that $\widetilde{u}$ is a linear function of x, y with $\widetilde{u}(x_i, y_i) = u_i$ for $i = 1, 2, 3$. Then $\widetilde{u}$ is given by

$$\widetilde{u} = \boldsymbol{h}^\top \boldsymbol{u}_e = \begin{pmatrix} 1 & x & y \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-\top} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}. \tag{2.27}$$

Confirming equalities $\widetilde{u}(x_i, y_i) = u_i$ for $1 \le i \le 3$ is left as an exercise for the reader.

Now substituting $\widetilde{u}$ into the left integrand in (2.25) and choosing v as the same form of $\tilde{u}$, i.e., $v = \boldsymbol{h}^\top \boldsymbol{v}_e$ with $\boldsymbol{v}_e := (v_1, v_2, v_3)^\top$ gives

$$\begin{aligned} \frac{\partial v}{\partial x}\frac{\partial \widetilde{u}}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial \widetilde{u}}{\partial y} &= \frac{\partial}{\partial x}\boldsymbol{v}_e^\top \boldsymbol{h} \frac{\partial}{\partial x}\boldsymbol{h}^\top \boldsymbol{u}_e + \frac{\partial}{\partial y}\boldsymbol{v}_e^\top \boldsymbol{h} \frac{\partial}{\partial y}\boldsymbol{h}^\top \boldsymbol{u}_e \\ &= \boldsymbol{v}_e^\top \left(\frac{\partial}{\partial x}\boldsymbol{h} \frac{\partial}{\partial x}\boldsymbol{h}^\top + \frac{\partial}{\partial y}\boldsymbol{h} \frac{\partial}{\partial y}\boldsymbol{h}^\top \right) \boldsymbol{u}_e, \end{aligned}$$

where we used the relation $\boldsymbol{h}^\top \boldsymbol{v}_e = \boldsymbol{v}_e^\top \boldsymbol{h}$. Here, from (2.26), it follows that

$$\boldsymbol{h}_x := \frac{\partial \boldsymbol{h}}{\partial x} = \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{D} \begin{pmatrix} y_2 - y_3 \\ y_3 - y_1 \\ y_1 - y_2 \end{pmatrix} =: \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \tag{2.28}$$

$$\boldsymbol{h}_y := \frac{\partial \boldsymbol{h}}{\partial y} = \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{D} \begin{pmatrix} x_3 - x_2 \\ x_1 - x_3 \\ x_2 - x_1 \end{pmatrix} =: \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}, \tag{2.29}$$

where D is defined by

$$D := \det \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}.$$

D is twice the area of the triangle. Since $\boldsymbol{h}_x$ and $\boldsymbol{h}_y$ do not depend on x, y, the left side of (2.25) is calculated as follows:

$$\begin{aligned} \iint_{e_i} \left(\frac{\partial v}{\partial x}\frac{\partial \tilde{u}}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial \tilde{u}}{\partial y} \right) \mathrm{d}x\mathrm{d}y &= \boldsymbol{v}_e^\top (\boldsymbol{h}_x \boldsymbol{h}_x^\top + \boldsymbol{h}_y \boldsymbol{h}_y^\top) \boldsymbol{u}_e \iint_{e_i} \mathrm{d}x\mathrm{d}y \\ &= \boldsymbol{v}_e^\top \frac{(\boldsymbol{h}_x \boldsymbol{h}_x^\top + \boldsymbol{h}_y \boldsymbol{h}_y^\top) D}{2} \boldsymbol{u}_e \\ &= \boldsymbol{v}_e^\top A_e \boldsymbol{u}_e. \end{aligned}$$

Here, $\iint_{e_i} \mathrm{d}x\mathrm{d}y$ is the area of triangle e_i, i.e., $D/2$. Matrix A_e is a 3×3 symmetric matrix whose (i, j) element, $a_{ij}^{(e)}$, is given by

$$a_{ij}^{(e)} = \frac{D}{2}(b_i b_j + c_i c_j), \tag{2.30}$$

from (2.28) and (2.29). The right-hand side of (2.25) is written as

$$\boldsymbol{f}_e := \begin{pmatrix} f_1^{(e)} \\ f_2^{(e)} \\ f_3^{(e)} \end{pmatrix} = \begin{pmatrix} \iint_{e_i} f_0 h_1 \,\mathrm{d}x\mathrm{d}y \\ \iint_{e_i} f_0 h_2 \,\mathrm{d}x\mathrm{d}y \\ \iint_{e_i} f_0 h_3 \,\mathrm{d}x\mathrm{d}y \end{pmatrix} = \frac{f_0 D}{6} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \tag{2.31}$$

where we used $f(x, y) = f_0$ (constant function) for simplicity. Then, we have

$$\iint_{e_i} v f \,\mathrm{d}x\mathrm{d}y = \boldsymbol{v}_e^\top \boldsymbol{f}_e.$$

On calculating the right-hand side of (2.31), the integrand h_i $(1 \le i \le 3)$ is a triangular pyramid with the height being 1 and the area of the base being $D/2$. Thus the volume of the triangle pyramid is $(D/2) \cdot 1 \cdot 1/3 = D/6$.

From this, the left-hand side and the right-hand side of (2.25) are approximated by

$$\boldsymbol{v}_e^\top A_e \boldsymbol{u}_e, \quad \boldsymbol{v}_e^\top \boldsymbol{f}_e. \tag{2.32}$$

The above process is the derivation of (2.32) regarding a small area e_i. Considering all the linear systems for all e_i's together with boundary conditions leads to large linear systems. By solving the large linear systems, the approximate solution u_i on each node can be obtained. In the next subsection, we will see how large linear systems can be obtained.

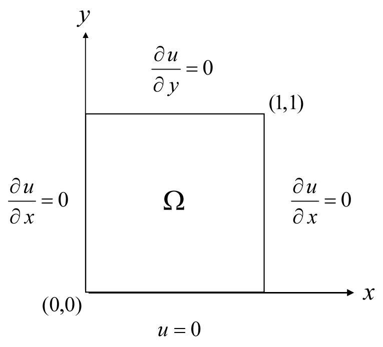
2.1.5 Example

As an example of the previous subsection, we consider Poisson Eq. (2.19) on the $(0, 1) \times (0, 1)$ square domain Ω in Fig. 2.7, and we will see how linear systems are obtained.

We first decompose the square domain in Fig. 2.7 into eight triangles in Fig. 2.8. Next, calculate (2.32) for A_e and $\boldsymbol{f}_e$ in each triangle element in Fig. 2.8, and finally, combine all the information, leading to linear systems.

In what follows, we calculate (2.32) for A_e and $\boldsymbol{f}_e$. We see from Fig. 2.8 that there are two kinds of triangles, i.e., type I for $e_1^{\mathrm{I}}, e_3^{\mathrm{I}}, e_5^{\mathrm{I}}, e_7^{\mathrm{I}}$ and type II for $e_2^{\mathrm{II}}, e_4^{\mathrm{II}}, e_6^{\mathrm{II}}, e_8^{\mathrm{II}}$, leading to two types of linear systems corresponding to (2.8). The details of the two triangles are shown in Fig. 2.9.

For the type I triangle, it follows from (2.28), (2.29), and recalling D being twice the area of the triangle (i.e., $D = h^2$), we have



y
$\frac{\partial u}{\partial y} = 0$
(1,1)
$\frac{\partial u}{\partial x} = 0$
Ω
$\frac{\partial u}{\partial x} = 0$
(0,0)
x
$u = 0$

Fig. 2.7 The (0, 1) × (0, 1) square domain

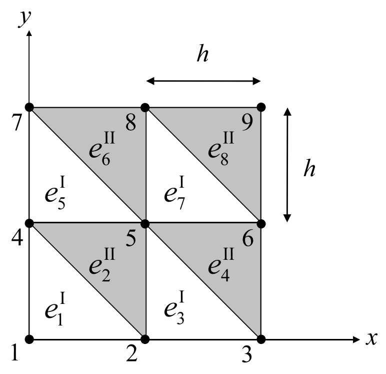


Fig. 2.8 Triangle mesh

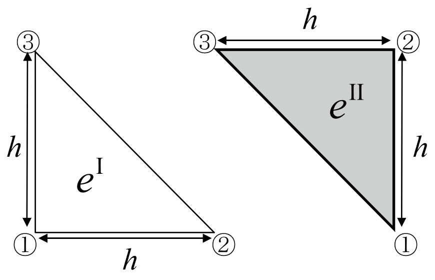


Fig. 2.9 Two types of triangles

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix} -h \\ h \\ 0 \end{pmatrix}, \quad \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix} -h \\ 0 \\ h \end{pmatrix}.$$

Then, from (2.30) and (2.31), we obtain

$$A_{e^{\mathrm{I}}} = \frac{1}{2} \begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{f}_{e^{\mathrm{I}}} = \frac{f_0 h^2}{6} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \tag{2.33}$$

which correspond to (2.32).

Similarly, for the type II triangle, we have

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix} 0 \\ h \\ -h \end{pmatrix}, \quad \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix} -h \\ h \\ 0 \end{pmatrix},$$

and thus

$$A_{e^{\mathrm{II}}} = \frac{1}{2} \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}, \quad \boldsymbol{f}_{e^{\mathrm{II}}} = \frac{f_0 h^2}{6} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{2.34}$$

Table 2.1 Relation between local node numbers and global node numbers.

Element	e_1^{I}	e_2^{II}	e_3^{I}	e_4^{II}	e_5^{I}	e_6^{II}	e_7^{I}	e_8^{II}
Node number of triangle element	Whole node numbers							
1	1	2	2	3	4	5	5	6
2	2	5	3	6	5	8	6	9
3	4	4	5	5	7	7	8	8

In the following, we explain how to combine all the information to obtain linear systems. In order to combine them, it is required to relate global node numbers 1–9 in Fig. 2.8 with local node numbers 1, 2, and 3 in Fig. 2.9, and the resulting table is shown in Table 2.1. Finally, linear systems are derived as follows. We first consider triangle element e_1^{I}. From (2.33) and Table 2.1, we have

$$A_{e_1^{\mathrm{I}}} = \frac{1}{2}\begin{array}{cc}\begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} & \begin{array}{c}1\\2\\4\end{array} \\ \begin{array}{ccc}1 & \quad 2 & \quad 4\end{array} & \end{array}, \quad \boldsymbol{f}_{e^{\mathrm{I}}} = \frac{f_0 h^2}{6}\begin{pmatrix}1\\1\\1\end{pmatrix}\begin{array}{c}1\\2\\4\end{array}$$

and by using the information we construct the following matrix and right-hand side:

$$\frac{1}{2}\begin{array}{cc}\begin{pmatrix}
\mathbf{2} & \mathbf{-1} & 0 & \mathbf{-1} & 0 & 0 & 0 & 0 & 0 \\
\mathbf{-1} & \mathbf{1} & 0 & \mathbf{0} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{-1} & \mathbf{0} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} & \begin{array}{c}\mathbf{1}\\\mathbf{2}\\3\\\mathbf{4}\\5\\6\\7\\8\\9\end{array} \\
\begin{array}{ccccccccc}\mathbf{1} & \mathbf{2} & 3 & \mathbf{4} & 5 & 6 & 7 & 8 & 9\end{array} & \end{array}, \quad \frac{f_0 h^2}{6}\begin{pmatrix}\mathbf{1}\\\mathbf{1}\\0\\\mathbf{1}\\0\\0\\0\\0\\0\end{pmatrix}\begin{array}{c}\mathbf{1}\\\mathbf{2}\\3\\\mathbf{4}\\5\\6\\7\\8\\9\end{array}$$

for the linear systems. We next consider triangle element e_2^{II}. From (2.34) and Table 2.1, we have

$$A_{e_2^{\mathrm{II}}} = \frac{1}{2}\begin{array}{cc}\begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} & \begin{array}{c}2\\5\\4\end{array} \\ \begin{array}{ccc}2 & \quad 5 & \quad 4\end{array} & \end{array}, \quad \boldsymbol{f}_{e^{\mathrm{II}}} = \frac{f_0 h^2}{6}\begin{pmatrix}1\\1\\1\end{pmatrix}\begin{array}{c}2\\5\\4\end{array}$$

and adding the information to the aforementioned matrix and the right-hand side yields

$$\frac{1}{2}\begin{array}{cc}\left(\begin{array}{ccccccccc} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & \mathbf{2} & 0 & \mathbf{0} & \mathbf{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & \mathbf{0} & 0 & \mathbf{2} & \mathbf{-1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{-1} & 0 & \mathbf{-1} & \mathbf{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) & \begin{array}{c} 1 \\ \mathbf{2} \\ 3 \\ \mathbf{4} \\ \mathbf{5} \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \\ \begin{array}{ccccccccc} 1 & \mathbf{2} & 3 & \mathbf{4} & \mathbf{5} & 6 & 7 & 8 & 9 \end{array} & \end{array}, \qquad \frac{f_0 h^2}{6}\left(\begin{array}{c} 1 \\ \mathbf{2} \\ 0 \\ \mathbf{2} \\ \mathbf{1} \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right)\begin{array}{c} 1 \\ \mathbf{2} \\ 3 \\ \mathbf{4} \\ \mathbf{5} \\ 6 \\ 7 \\ 8 \\ 9 \end{array}$$

and similarly for the remaining elements $e_3^{\mathrm{I}}, e_4^{\mathrm{II}}, \ldots, e_8^{\mathrm{II}}$, we finally have

$$\frac{1}{2}\begin{pmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 0 \\ 0 & -2 & 0 & -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & 0 & -1 & 0 & -2 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{pmatrix} = \frac{f_0 h^2}{6}\begin{pmatrix} 1 \\ 3 \\ 2 \\ 3 \\ 6 \\ 3 \\ 2 \\ 3 \\ 1 \end{pmatrix}.$$

The linear systems do not include the information of Dirichlet condition ($u(x, 0) = 0$ for $x \in [0, 1]$). It follows from Figs. 2.7 and 2.8 that the following condition is required for satisfying the Dirichlet condition:

$$u_1 = u_2 = u_3 = 0.$$

Thus, removing the first three rows from the linear systems and substituting $u_1 = u_2 = u_3 = 0$ into the resulting linear systems yields:

$$\frac{1}{2}\begin{pmatrix} 4 & -2 & 0 & -1 & 0 & 0 \\ -2 & 8 & -2 & 0 & -2 & 0 \\ 0 & -2 & 4 & 0 & 0 & -1 \\ -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{pmatrix} = \frac{f_0 h^2}{6}\begin{pmatrix} 3 \\ 6 \\ 3 \\ 2 \\ 3 \\ 1 \end{pmatrix}. \tag{2.35}$$

By solving the linear systems we have $u_4, u_5, \ldots, u_9$.

Finally, we compare the approximate solution with the exact solution. The exact solution of the Poisson equation is $u(x, y) = f_0 y - \frac{1}{2} f_0 y^2$. When $f_0 = 1$, the values of u_4, u_5, u_6 for the exact solution are 0.375 and u_7, u_8, u_9 are 0.5. Solving linear systems (2.35) with $f_0 = 1$ and $h = \frac{1}{2}$ yields $u_4 \approx 0.381$, $u_5 \approx 0.375$, $u_6 \approx 0.369$ and $u_7 \approx 0.524$, $u_8 \approx 0.500$, $u_9 \approx 0.476$.

If a more accurate approximate solution is required, the number of nodes should be increased. From the above derivation, we see that the size of the linear systems is almost the same as the number of the whole nodes.

2.2 Computational Physics

2.2.1 Large-Scale Electronic Structure Calculation

Molecular dynamics simulations in large scale systems can be efficiently computed by using the one-body density matrix ρ, which is based on the fact that any physical quantity $\langle X \rangle$ can be obtained by

$$\langle X \rangle = \mathrm{Tr}\,(\rho X) = \iint \rho(\boldsymbol{r}, \boldsymbol{r}') X(\boldsymbol{r}', \boldsymbol{r})\, \mathrm{d}\boldsymbol{r}\mathrm{d}\boldsymbol{r}'$$

and the corresponding discretization form is written as

$$\sum_{i,j} \rho_{ij} X_{ji}.$$

It is important to note here that if X is a short-range operator, we only need the short-range behavior of the density matrix. As an extreme example, if X is a diagonal matrix (i.e., off-diagonal elements are zeros), then we only need the diagonal element of the density matrix because $\sum_{i,j} \rho_{ij} X_{ji} = \sum_i \rho_{ii} X_{ii}$, leading to very efficient computation.

In what follows, we explain the density matrix ρ in detail. It is known that the (i, j) element of the density matrix ρ is given by

$$\rho_{ij} = -\frac{1}{\pi} \int_{-\infty}^{\infty} \mathrm{Im}\, G_{ij}(\epsilon) f\left(\frac{\epsilon - \mu}{K_B T}\right) \mathrm{d}\epsilon, \tag{2.36}$$

where $f(x)$ is the Fermi distribution function, μ is a chemical potential that may be determined so that the total number of electrons equals the sum of the diagonal elements of the density matrix. Here, $G_{ij}(\epsilon)$ is the (i, j) element of the following matrix:

$$G(\epsilon) = [(\epsilon + i\delta)I - H]^{-1}, \tag{2.37}$$

where i is the unit imaginary number, δ is an (infinitely) small positive number ($\delta \to 0_+$), and I and H are the identity matrix and a Hamiltonian matrix. The matrix G is referred to as *Green's function*. The (i, j) element of the Hamiltonian matrix H may be given by the following real space integration:

$$H_{ij} = \int f_i(\boldsymbol{r}) \left(\frac{-\hbar}{2m} \Delta + V(\boldsymbol{r}) \right) f_j(\boldsymbol{r}) \, \mathrm{d}\boldsymbol{r},$$

where m and $\hbar$ are the mass of the electron and the (reduced) Planck constant, Δ is the Laplacian. $f_j(\boldsymbol{r})$'s are prescribed functions that are linearly independent. $V(\boldsymbol{r})$ is a prescribed function regarding the interactions between electrons and nuclei. For the details, see, e.g., [184, 190] and the references therein.

We now describe how linear systems arise from the above computation. Let $\boldsymbol{e}_i$ be the ith unit vector. Then from (2.37), the (i, j) element of $G(\epsilon)$ is calculated by

$$G_{ij}(\epsilon) = \boldsymbol{e}_i^\top [(\epsilon + i\delta)I - H]^{-1} \boldsymbol{e}_j.$$

Therefore, letting $\boldsymbol{x}_j = [(\epsilon + i\delta)I - H]^{-1}\boldsymbol{e}_j$ yields

$$[(\epsilon + i\delta)I - H]\boldsymbol{x}_j = \boldsymbol{e}_j,$$

which is a linear system.

In computational physics, the Hamiltonian matrix H is Hermitian or real symmetric. If H is real symmetric, the coefficient matrix $(\epsilon + i\delta)I - H$ is complex symmetric. Furthermore, for computing the integration in (2.36), a suitable numerical quadrature formula is used, which requires $G_{ij}(\epsilon_k)$ for $k = 1, 2, \ldots, m$. Thus, for computing the (i, j) element of the density matrix $\rho_{i,j}$, we need to solve linear systems of the form

$$(A + \sigma_k I)\boldsymbol{x}_j^{(k)} = \boldsymbol{b}, \quad k = 1, 2, \ldots, m,$$

where $A := i\delta I - H$, $\boldsymbol{b} := \boldsymbol{e}_j$ and $\sigma_k := \epsilon_k$. These are known as shifted linear systems, which are efficiently solved by Krylov subspace methods by utilizing the shift-invariance of Krylov subspace. Krylov subspace methods for shifted linear systems are described in Chap. 4.

2.2.2 Lattice Quantum Chromodynamics

Quantum chromodynamics (QCD) is the theory of the strong interaction between quarks and gluons, and lattice QCD is a non-perturbative approach to solving QCD, where the theory is formulated on a lattice in space and time.

It is known that the most time-consuming part of lattice QCD is solving linear systems

$$Ax = \boldsymbol{b},$$

which are often written as $A\psi = \phi$ and the solution is used for obtaining quark propagators. For the details of the quark propagators including a summary of lattice QCD, see, e.g., [69]. In the following, we describe the structure of the Wilson fermion matrix A.

The Wilson fermion matrix is written as $A = I - \kappa D$, where I is the identity matrix, and κ is a real nonnegative parameter, referred to as the hopping parameter. Matrix D is called the hopping matrix. We now explain the structure of the hopping matrix.

Let us consider a four-dimensional hypercubic lattice which can be regarded as an equispaced grid of the hypercube. Each grid point can be written as a vector x such that

$$\boldsymbol{x} \in \Omega := \{(x_1, x_2, x_3, x_4) \,:\, x_1, x_2, x_3, x_4 \in \{1, 2, \ldots, N\}\}.$$

For simplicity, we assume that x_1, x_2, x_3, x_4 run from 1 to the same number N. Then it is easy to see that the number of elements of Ω is N^4.

Using $\boldsymbol{x}, \boldsymbol{y} \in \Omega$, the hopping matrix D is written by block matrix from

$$D(\boldsymbol{x}; \boldsymbol{y}) = \sum_{\mu=1}^{4} \left\{ \left[(I_4 - \gamma_\mu) \otimes U_\mu(\boldsymbol{x})\right] \delta_{\boldsymbol{x},\boldsymbol{y}-\boldsymbol{e}_\mu} + \left[(I_4 + \gamma_\mu) \otimes U_\mu^{\mathrm{H}}(\boldsymbol{x} - \boldsymbol{e}_\mu)\right] \delta_{\boldsymbol{x},\boldsymbol{y}+\boldsymbol{e}_\mu} \right\}. \tag{2.38}$$

Here, $\delta_{\boldsymbol{x},\boldsymbol{y}}$ is the Kronecker delta, i.e., $\delta_{\boldsymbol{x},\boldsymbol{y}} = 1$ if $\boldsymbol{x} = \boldsymbol{y}$ and $\delta_{\boldsymbol{x},\boldsymbol{y}} = 0$ if $\boldsymbol{x} \neq \boldsymbol{y}$. $D(\boldsymbol{x}; \boldsymbol{y})$ is a small matrix of the size 12×12 and the hopping matrix D is given as follows:

$$D = \begin{bmatrix} D_{1,1} & D_{1,2} & \cdots & D_{1,N^4} \\ D_{2,1} & D_{2,2} & \cdots & D_{2,N^4} \\ \vdots & \vdots & \ddots & \vdots \\ D_{N^4,1} & D_{N^4,2} & \cdots & D_{N^4,N^4} \end{bmatrix}$$

$$= \begin{bmatrix} D(1,1,1,1;1,1,1,1) & D(1,1,1,1;1,1,1,2) & \cdots & D(1,1,1,1;N,N,N,N) \\ D(1,1,1,2;1,1,1,1) & D(1,1,1,2;1,1,1,2) & \cdots & D(1,1,1,2;N,N,N,N) \\ \vdots & \vdots & \ddots & \vdots \\ D(N,N,N,N;1,1,1,1) & D(N,N,N,N;1,1,1,2) & \cdots & D(N,N,N,N;N,N,N,N) \end{bmatrix}.$$

The size of the hopping matrix D is thus $12N^4 \times 12N^4$. If $N = 64$, then the number of unknowns for the linear systems is about 200 million! The size grows much more rapidly than that arising from the discretization of three-dimensional partial differential equations in Sect. 2.1.1.4.

The details of (2.38) are explained as follows: I_4 is the 4-by-4 identity matrix, $\delta_{\boldsymbol{x},\boldsymbol{y}}$ is a function ($\delta : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}$), where $\delta_{\boldsymbol{x},\boldsymbol{y}} = 0$ if $\boldsymbol{x} \neq \boldsymbol{y}$, and $\delta_{\boldsymbol{x},\boldsymbol{y}} = 1$ if $\boldsymbol{x} = \boldsymbol{y}$. The symbol $\boldsymbol{e}_\mu$ is the transpose of the μth unit vector, or equivalently the μth row of the 4×4 identity matrix I_4. For each μ, the symbol γ_μ is a matrix given by

$$\gamma_1 = \sigma_y \otimes \sigma_x, \quad \gamma_2 = \sigma_y \otimes \sigma_y, \quad \gamma_3 = \sigma_y \otimes \sigma_z, \quad \gamma_4 = \sigma_z \otimes I_2,$$

where I_2 is the 2-by-2 identity matrix, and σ_x, σ_y, and σ_z are the Pauli matrices, i.e.,

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Finally, for each μ and $\boldsymbol{x}$, the symbol $U_\mu(\boldsymbol{x}) (= U_\mu([x_1, x_2, x_3, x_4])$ for $\mu = 1, 2, 3, 4$ belongs to SU(3), i.e., a 3-by-3 unitary matrix whose determinant is one. $U_\mu(\boldsymbol{x})$ corresponds to the background gauge field. $U_\mu(\boldsymbol{x})$ is called "cold" if $U_\mu(\boldsymbol{x}) = I_3$ and "hot" if the $U_\mu(\boldsymbol{x}) \in \mathrm{SU}(3)$ is randomly chosen.

Below are some examples for $D(\boldsymbol{x}, \boldsymbol{y})$ when $N = 2$:

$$\begin{aligned}
D_{1,1} &= D(1,1,1,1;1,1,1,1) = O,\\
D_{1,2} &= D(1,1,1,1;1,1,1,2) = (I_4 - \gamma_4) \otimes U_4([1,1,1,1]),\\
D_{1,3} &= D(1,1,1,1;1,1,2,1) = (I_4 - \gamma_3) \otimes U_3([1,1,1,1]),\\
D_{1,4} &= D(1,1,1,1;1,1,2,2) = O,\\
D_{1,5} &= D(1,1,1,1;1,2,1,1) = (I_4 - \gamma_2) \otimes U_2([1,1,1,1]),\\
D_{1,6} &= D(1,1,1,1;1,2,1,2) = O,\\
D_{1,7} &= D(1,1,1,1;1,2,2,1) = O,\\
D_{1,8} &= D(1,1,1,1;1,2,2,2) = O,\\
D_{1,9} &= D(1,1,1,1;2,1,1,1) = (I_4 - \gamma_1) \otimes U_1([1,1,1,1]),\\
&\vdots\\
D_{2,1} &= D(1,1,1,2;1,1,1,1) = (I_4 + \gamma_4) \otimes U_4^{\mathrm{H}}([1,1,1,1]),\\
D_{2,2} &= D(1,1,1,2;1,1,1,2) = O,\\
D_{2,3} &= D(1,1,1,2;1,1,2,1) = O,\\
D_{2,4} &= D(1,1,1,2;1,1,2,2) = (I_4 - \gamma_3) \otimes U_3([1,1,1,2]),\\
&\vdots\\
D_{16,16} &= D([2,2,2,2;2,2,2,2]) = O.
\end{aligned}$$

Note that $D_{2,1} = (I_4 + \gamma_4) \otimes U_4^{\mathrm{H}}([1,1,1,2] - \boldsymbol{e}_4) = (I_4 + \gamma_4) \otimes U_4^{\mathrm{H}}([1,1,1,1])$.

Matrix D is neither symmetric nor Hermitian, but it has a hidden symmetry as described next. Let

$$\gamma_5 = \sigma_x \otimes I_2.$$

Then, the hopping matrix D has the following property:

$$\Gamma_5 D = D^{\mathrm{H}} \Gamma_5, \tag{2.39}$$

where $\Gamma_5 = I_{N^4} \otimes \gamma_5 \otimes I_3$. This property is known as γ_5-symmetric (or γ_5-Hermitian). From Theorem 2.1-(5), matrix γ_5 is real symmetric and matrix Γ_5 is also real symmetric, and thus we have $\Gamma_5 = \Gamma_5^{\mathrm{H}}$. Then, we can rewrite (2.39) as

$$\Gamma_5 D = (\Gamma_5 D)^{\mathrm{H}}.$$

This means that $\Gamma_5 D$ is Hermitian. From Corollary 2.1, we have $\Gamma_5^{-1} = I_{N^4}^{-1} \otimes \gamma_5^{-1} \otimes I_3^{-1} = I_{N^4} \otimes \gamma_5 \otimes I_3 = \Gamma_5$. Thus we have another representation of (2.39):

$$\Gamma_5 D \Gamma_5 = D^{\mathrm{H}}. \tag{2.40}$$

Now, let us confirm (2.40) for $N = 2$. Let $P = \gamma_5 \otimes I_3$. Then the (i, j) block element of D is

$$\begin{aligned}
(\Gamma_5 D \Gamma_5)_{i,j} &= \left(\begin{bmatrix} P & & \\ & \ddots & \\ & & P \end{bmatrix} \begin{bmatrix} D_{1,1} & \cdots & D_{1,16} \\ \vdots & \ddots & \vdots \\ D_{16,1} & \cdots & D_{16,16} \end{bmatrix} \begin{bmatrix} P & & \\ & \ddots & \\ & & P \end{bmatrix} \right)_{i,j} \\
&= \left(\begin{bmatrix} PD_{1,1}P & \cdots & PD_{1,16}P \\ \vdots & \ddots & \vdots \\ PD_{16,1}P & \cdots & PD_{16,16}P \end{bmatrix} \right)_{i,j} \\
&= PD_{i,j}P.
\end{aligned}$$

As an example, consider $(i, j) = (1, 2)$. Then, from Theorem 2.1-(5) and (6), it follows that

$$\begin{aligned}
(\Gamma_5 D \Gamma_5)_{1,2} &= PD_{1,2}P \\
&= (\gamma_5 \otimes I_3) \left[(I_4 - \gamma_4) \otimes U_4([1, 1, 1, 1]) \right] (\gamma_5 \otimes I_3) \\
&= \gamma_5 (I_4 - \gamma_4) \gamma_5 \otimes U_4([1, 1, 1, 1]) \\
&= (I_4 - \gamma_5 \gamma_4 \gamma_5) \otimes U_4([1, 1, 1, 1]) \\
&= (I_4 + \gamma_4) \otimes U_4([1, 1, 1, 1]) \\
&= \left[(I_4 + \gamma_4)^{\mathrm{H}} \otimes U_4^{\mathrm{H}}([1, 1, 1, 1]) \right]^{\mathrm{H}} \\
&= \left[(I_4 + \gamma_4) \otimes U_4^{\mathrm{H}}([1, 1, 1, 1]) \right]^{\mathrm{H}} \\
&= D_{2,1}^{\mathrm{H}}.
\end{aligned}$$

Similarly, we can confirm that $(\Gamma_5 D \Gamma_5)_{i,j} = D^{\mathrm{H}}_{j,i}$ for the other i, j's. Thus we see that (2.40) holds true for the case $N = 2$.

In lattice QCD, there is a need to compute some small eigenvalues in magnitude. In this case, the shift-and-invert Lanczos (Arnoldi) method may be used. The method requires solving linear systems at each iteration. See [12] for algorithms to compute eigenvalue problems. A thick-restart Lanczos type method for eigenvalue problems arising in lattice QCD was proposed in [108].

2.3 Machine Learning

Linear systems arise in the field of machine learning. In this section, through two examples: least-squares regression and least-squares classification, we will see the importance of linear systems.

2.3.1 Least-squares Regression

In this subsection, we consider one of the simplest supervised learning: least-squares learning. Given training data set $(\boldsymbol{x}_i, y_i)$ $(i = 1, \dots, n)$ and a function $f_{\boldsymbol{\theta}}(\boldsymbol{x})$. Then the least-squares learning learns a parameter $\boldsymbol{\theta} \in \mathbb{R}^m$ such that the following error function is minimized:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2. \tag{2.41}$$

If $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is given by the linear combination of functions $f_1(\boldsymbol{x}), \dots, f_m(\boldsymbol{x})$ of the form

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_1 f_1(\boldsymbol{x}) + \theta_2 f_2(\boldsymbol{x}) + \theta_m f_m(\boldsymbol{x}), \tag{2.42}$$

then the error function (2.41) is rewritten as[1]

$$J(\boldsymbol{\theta}) = \|\boldsymbol{y} - M\boldsymbol{\theta}\|_2^2, \tag{2.43}$$

where $\boldsymbol{y} = [y_1, y_2, \dots, y_n]^\top$, $\boldsymbol{\theta} = [\theta_1, \theta_2, \dots, \theta_m]^\top$, and

[1] In this section, we explicitly describe the 2-norm as $\|\cdot\|_2$. In machine learning, 1-norm $\|\cdot\|_1$ is also used, especially for obtaining a sparse solution of the least-squares problems of the form $\|\boldsymbol{y} - M\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1$.

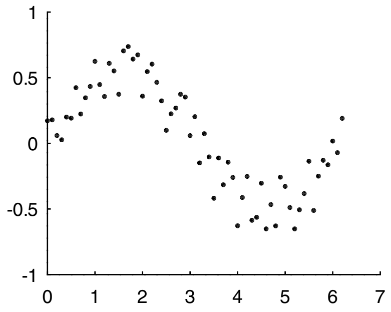


Fig. 2.10 The plot of the given data set

$$M = \begin{bmatrix} f_1(\boldsymbol{x}_1) \; f_2(\boldsymbol{x}_1) \; \dots f_m(\boldsymbol{x}_1) \\ f_1(\boldsymbol{x}_2) \; f_2(\boldsymbol{x}_2) \; \dots f_m(\boldsymbol{x}_2) \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ f_1(\boldsymbol{x}_n) \; f_2(\boldsymbol{x}_n) \; \dots f_m(\boldsymbol{x}_n) \end{bmatrix}.$$

For simplicity, column vectors of M are assumed to be linearly independent. For minimizing the error function (2.43), the following decomposition is useful:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \|\boldsymbol{y} - M\boldsymbol{\theta}\|_2^2 = (\boldsymbol{y} - M\boldsymbol{\theta})^\top(\boldsymbol{y} - M\boldsymbol{\theta}) \\ &= \boldsymbol{v}_{\boldsymbol{\theta}}^\top(M^\top M)^{-1}\boldsymbol{v}_{\boldsymbol{\theta}} - (M^\top\boldsymbol{y})^\top(M^\top M)^{-1}(M^\top\boldsymbol{y}) + \boldsymbol{y}^\top\boldsymbol{y}. \end{aligned} \tag{2.44}$$

Here $\boldsymbol{v}_{\boldsymbol{\theta}} = M^\top\boldsymbol{y} - M^\top M\boldsymbol{\theta}$. Since the columns of M are linearly independent, $M^\top M$ is a symmetric positive definite matrix, i.e., all the eigenvalues are positive, or equivalently $\boldsymbol{x}^\top M^\top M\boldsymbol{x} > 0$ for all nonzero vectors $\boldsymbol{x}$. From this, $J(\boldsymbol{\theta})$ is minimized if $\boldsymbol{v}_{\boldsymbol{\theta}} = \boldsymbol{0}$, i.e., $M^\top\boldsymbol{y} - M^\top M\boldsymbol{\theta} = \boldsymbol{0}$. Thus we need to solve the following linear systems:

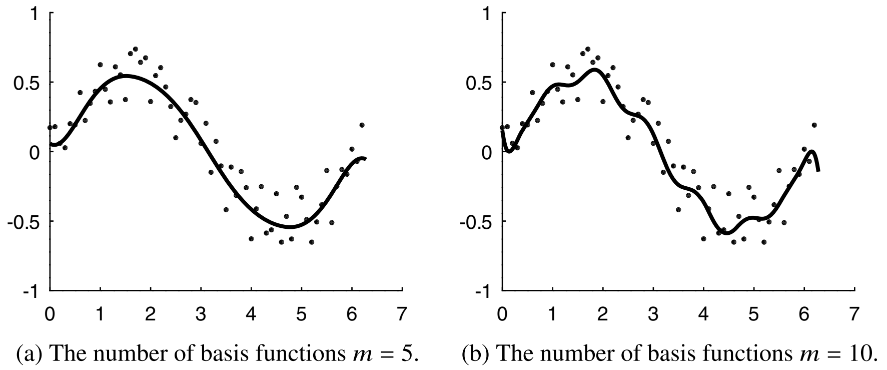
$$M^\top M\boldsymbol{\theta} = M^\top\boldsymbol{y}. \tag{2.45}$$

As a numerical example, consider $y = (1/2)\sin(x) + \epsilon$, where ϵ is a noise. We generated a data set (training data set) with $(x_1, x_2, \dots, x_{63}) = (0, 0.1, 0.2, \dots, 6.2)$ and $(y_1, y_2, \dots, y_{63}) = (y(0), y(0.1), y(0.2), \dots, y(6.2))$. The visualization of the data is shown in Fig. 2.10.

Next, we consider (2.42). For the basis functions $f_1(x), f_2(x), \dots, f_m(x)$, we use

$$f_k(x) = \begin{cases} \sin\frac{1}{2}kx & (k : \text{even}), \\ \cos\frac{1}{2}kx & (k : \text{odd}). \end{cases}$$

The unknown vector $\boldsymbol{\theta}$ of the model function (2.42) is given by solving linear systems (2.45), and the plots of $f_{\boldsymbol{\theta}}(x)$ with $m = 5$ and $m = 10$ are shown in Fig. 2.11.



(a) The number of basis functions $m = 5$. (b) The number of basis functions $m = 10$.

Fig. 2.11 The plots of $f_{\boldsymbol{\theta}}(x)$ in (2.42) with $m = 5$ and $m = 10$

From Fig. 2.11a, the function $f_{\boldsymbol{\theta}}(x)$ with $m = 5$ is robust against the noise. On the other hand, the function with $m = 10$ seems to try to fit the data with the given noise. This phenomenon is known as *overfitting*.

In what follows, one remedy for the overfitting is introduced, which also requires solving (parameterized) linear systems. We now consider the following constrained minimization problem of the form

$$\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - M\boldsymbol{\theta}\|_2^2 \text{ subject to } \|\boldsymbol{\theta}\|_2^2 \le R. \tag{2.46}$$

If R is infinite, the minimization problem is equivalent to (2.43).

In order to solve (2.46), the dual problem is useful. Let $f(\boldsymbol{\theta}) = \|\boldsymbol{y} - M\boldsymbol{\theta}\|_2^2$ and $g(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 - R$, then (2.46) can be rewritten as

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \text{ subject to } g(\boldsymbol{\theta}) \le 0. \tag{2.47}$$

Here we introduce the Lagrange function with $\lambda \ge 0$:

$$L(\boldsymbol{\theta}, \lambda) := f(\boldsymbol{\theta}) + \lambda g(\boldsymbol{\theta}).$$

Since $\lambda \ge 0$ and $g(\boldsymbol{\theta}) \le 0$, we obtain

$$\max_{\lambda \ge 0} L(\boldsymbol{\theta}, \lambda) = \begin{cases} f(\boldsymbol{\theta}) & (\text{if } g(\boldsymbol{\theta}) \le 0), \\ +\infty & (\text{otherwise}). \end{cases}$$

Thus (2.47) is equivalent to

$$\min_{\boldsymbol{\theta}} \max_{\lambda \ge 0} L(\boldsymbol{\theta}, \lambda).$$

We now consider the following dual problem:

$$\max_{\lambda \geq 0} \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda). \tag{2.48}$$

If f and g are convex, and certain conditions are satisfied, we have

$$\max_{\lambda \geq 0} \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda) = \min_{\boldsymbol{\theta}} \max_{\lambda \geq 0} L(\boldsymbol{\theta}, \lambda),$$

which is called the strong duality. This is related to the Lagrangian duality theory; see [26] for the details.

The optimization problem (2.48) reads

$$\max_{\lambda \geq 0} \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda) \Leftrightarrow \max_{\lambda \geq 0} \min_{\boldsymbol{\theta}} \|\boldsymbol{y} - M\boldsymbol{\theta}\|_2^2 + \lambda(\|\boldsymbol{\theta}\|_2^2 - R). \tag{2.49}$$

Similar to (2.44), the error function can be decomposed as

$$\begin{aligned} &\|\boldsymbol{y} - M\boldsymbol{\theta}\|_2^2 + \lambda(\|\boldsymbol{\theta}\|_2^2 - R) \\ &\quad = \boldsymbol{w}_{\boldsymbol{\theta}}^\top (M^\top M + \lambda I)^{-1} \boldsymbol{w}_{\boldsymbol{\theta}} - (M^\top \boldsymbol{y})^\top (M^\top M + \lambda I)^{-1} M^\top \boldsymbol{y} + \boldsymbol{y}^\top \boldsymbol{y} - \lambda R, \end{aligned}$$

where $\boldsymbol{w}_{\boldsymbol{\theta}} = M^\top \boldsymbol{y} - (M^\top M + \lambda I)\boldsymbol{\theta}$. Since $\lambda \geq 0$, matrix $M^\top M + \lambda I$ is symmetric positive definite. Thus $\boldsymbol{\theta}$ can be obtained by solving the following linear systems:

$$(M^\top M + \lambda I)\boldsymbol{\theta} = M^\top \boldsymbol{y}. \tag{2.50}$$

Let $\boldsymbol{\theta}(\lambda)$ be the solution of the linear systems (2.50). Then (2.48) is equivalent to

$$\max_{\lambda \geq 0} L(\boldsymbol{\theta}(\lambda), \lambda).$$

For approximately solving the above maximization problem, one may set $0 \leq \lambda_1 < \lambda_2 < \cdots < \lambda_m$ and then compute

$$\max\{L(\boldsymbol{\theta}(\lambda_1), \lambda_1), L(\boldsymbol{\theta}(\lambda_2), \lambda_2), \ldots, L(\boldsymbol{\theta}(\lambda_m), \lambda_m)\}.$$

In this case, we need $\boldsymbol{\theta}(\lambda_1), \boldsymbol{\theta}(\lambda_2), \ldots, \boldsymbol{\theta}(\lambda_m)$, i.e.,

$$(M^\top M + \lambda_k I)\boldsymbol{\theta}(\lambda_k) = M^\top \boldsymbol{y} \quad k = 1, 2, \ldots, m. \tag{2.51}$$

Equations (2.51) are known as *shifted linear systems*. Krylov subspace methods for solving shifted linear systems are described in Chap. 4.

Here, we show below a numerical example whose data is the same as in Fig. 2.10. Solving linear systems (2.50) with a given λ yields the unknown vector $\boldsymbol{\theta}$ of the model function (2.42). The plot of $f_{\boldsymbol{\theta}}(x)$ with $m = 10$ and $\lambda = 1.0$ is shown in Fig. 2.12b. For reference and convenience, Fig. 2.11b is shown again in Fig. 2.12a.

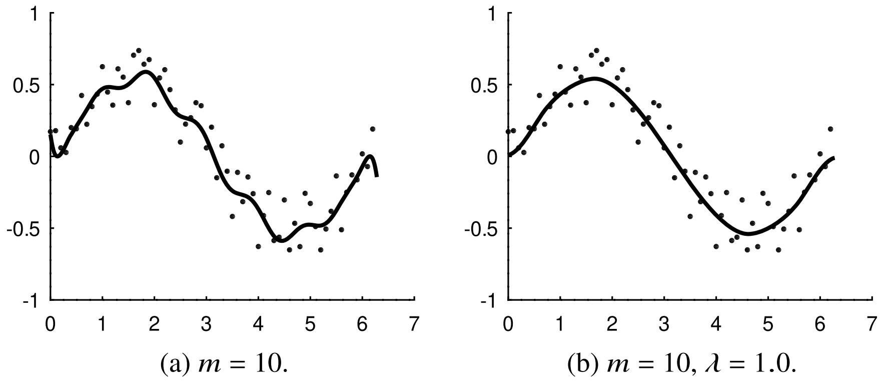


Fig. 2.12 The plots of $f_{\boldsymbol{\theta}}(x)$ in (2.42) with $m = 10$ and $m = 10$ with $\lambda = 1.0$

In Fig. 2.12b, we see the result is now robust against the noise and thus seems to avoid the overfitting that is seen in Fig. 2.12a.

2.3.2 Least-squares Classification

In this subsection, least-squares probabilistic classification [178] (see also [148] for the summary) is described.

Let $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} = \{1, 2, \ldots, c\}$, where c is the number of classes. The problem of the classification is to classify $\boldsymbol{x} \in \mathcal{X}$ into classes $y \in \mathcal{Y}$, based on a given dataset (training dataset) $(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ for $i = 1, 2, \ldots, n$. Here $(\boldsymbol{x}_i, y_i)$ means that the data $\boldsymbol{x}_i \in \mathcal{X}$ belongs to the class $y_i \in \mathcal{Y}$.

The probabilistic pattern recognition is to estimate the class-posterior probability $p(y|\boldsymbol{x})$ from the training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Once $p(y|\boldsymbol{x})$ is obtained, new data $\boldsymbol{x}$ can be classified by

$$\hat{y} = \mathop{\arg\max}_{y \in \{1,2,\ldots,c\}} p(y|\boldsymbol{x}).$$

Let $p(\boldsymbol{x})$ be the marginal density of $\boldsymbol{x}$ with $p(\boldsymbol{x}) > 0$ for all $\boldsymbol{x}$. Then the class-posterior probability $p(y|\boldsymbol{x})$ is written as

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, y)}{p(\boldsymbol{x})}, \tag{2.52}$$

since $p(\boldsymbol{x}, y) = p(y|\boldsymbol{x})p(\boldsymbol{x})$ and the assumption $p(\boldsymbol{x}) > 0$.

The least-squares probabilistic classification uses the following linear model $q(y|\boldsymbol{x}; \boldsymbol{\theta}^{(y)})$ for the class-posterior probability $p(y|\boldsymbol{x})$:

$$q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)}) = \boldsymbol{\theta}^{(y)\top}\boldsymbol{f}(\boldsymbol{x}) = \theta_1^{(y)}f_1(\boldsymbol{x}) + \theta_2^{(y)}f_2(\boldsymbol{x}) + \cdots + \theta_b^{(y)}f_b(\boldsymbol{x}), \tag{2.53}$$

where $\boldsymbol{\theta}^{(y)} = [\theta_1^{(y)}, \theta_2^{(y)}, \ldots, \theta_b^{(y)}]^\top$, and $f_i(\boldsymbol{x})$ $(i = 1, \ldots, b)$ are the given basis functions, e.g.,

$$f_i(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x}_i) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2}{2h^2}\right). \tag{2.54}$$

Here $K(\boldsymbol{x}, \boldsymbol{x}_i)$ is known as the *Gaussian kernel* and h is a scalar parameter that is set by users.

Parameter $\boldsymbol{\theta}^{(y)}$ is determined by approximately minimizing the following least squares error:

$$J_y(\boldsymbol{\theta}^{(y)}) = \frac{1}{2}\int \left(q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)}) - p(y|\boldsymbol{x})\right)^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

$J_y(\boldsymbol{\theta}^{(y)})$ can be rewritten as

$$\begin{aligned} J_y(\boldsymbol{\theta}^{(y)}) &= \frac{1}{2}\int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)})^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} - \int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)}) p(y|\boldsymbol{x}) p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + c \\ &= \frac{1}{2}\int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)})^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} - \int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)}) p(\boldsymbol{x}, y)\,\mathrm{d}\boldsymbol{x} + c \\ &= \frac{1}{2}\int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)})^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} - \int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)}) p(y) p(\boldsymbol{x}|y)\,\mathrm{d}\boldsymbol{x} + c, \end{aligned} \tag{2.55}$$

where $c = (1/2)\int p(y|\boldsymbol{x})^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, and (2.52) was used in the second equation.

By using the average of the training data $(\boldsymbol{x}_1, y_1)$, $(\boldsymbol{x}_2, y_2)$, ..., $(\boldsymbol{x}_n, y_n)$, we have the following approximation:

$$\int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)})^2 p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \frac{1}{n}\sum_{i=1}^{n} q(y|\boldsymbol{x}_i;\boldsymbol{\theta}^{(y)})^2,$$

$$\int q(y|\boldsymbol{x};\boldsymbol{\theta}^{(y)}) p(y) p(\boldsymbol{x}|y)\,\mathrm{d}\boldsymbol{x} \approx \frac{1}{n_y}\sum_{i:y_i=y} q(y|\boldsymbol{x}_i;\boldsymbol{\theta}^{(y)}) p(y) \approx \frac{1}{n}\sum_{i:y_i=y} q(y|\boldsymbol{x}_i;\boldsymbol{\theta}^{(y)}).$$

In the last approximation, we used $p(y) \approx n_y/n$, where n_y is the number of training data that belong to the class $y \in \mathcal{Y}$. Note that the symbol $\sum_{i:y_i=y}$ means the sum over i such that $y_i = y$ for the given class $y \in \mathcal{Y}$.

Using this approximation to (2.55) together with regularization term $(\lambda/2n)$ $\|\boldsymbol{\theta}^{(y)}\|_2^2$, see also (2.49), we have the following new minimization problem:

$$\hat{J}_y(\boldsymbol{\theta}^{(y)}) = \frac{1}{2n}\sum_{i=1}^{n} q(y|\boldsymbol{x}_i;\boldsymbol{\theta}^{(y)})^2 - \frac{1}{n}\sum_{i:y_i=y} q(y|\boldsymbol{x}_i;\boldsymbol{\theta}^{(y)}) + \frac{\lambda}{2n}\|\boldsymbol{\theta}^{(y)}\|_2^2, \tag{2.56}$$

where the constant c was omitted because it does not affect the minimization problem.

Let $(M)_{i,j} = f_i(\boldsymbol{x}_j)$. Then from (2.53), approximate error function (2.56) can be rewritten as

$$\hat{J}_y(\boldsymbol{\theta}^{(y)}) = \frac{1}{2n}\boldsymbol{\theta}^{(y)\top} M^\top M \boldsymbol{\theta}^{(y)} - \frac{1}{n}\boldsymbol{\theta}^{(y)\top} M^\top \boldsymbol{p}^{(y)} + \frac{\lambda}{2n}\boldsymbol{\theta}^{(y)\top}\boldsymbol{\theta}^{(y)}, \tag{2.57}$$

where $\boldsymbol{p}^{(y)}$ is a vector whose ith element $p_i^{(y)}$ is defined by

$$p_i^{(y)} = \begin{cases} 1 & (\text{if } y_i = y), \\ 0 & (\text{if } y_i \neq y). \end{cases}$$

Similar to the previous subsection, (2.57) can be decomposed as

$$\hat{J}_y(\boldsymbol{\theta}^{(y)}) = \frac{1}{2n}\left(M^\top \boldsymbol{p}^{(y)} - G\boldsymbol{\theta}^{(y)}\right)^\top G^{-1}\left(M^\top \boldsymbol{p}^{(y)} - G\boldsymbol{\theta}^{(y)}\right) - \frac{1}{2n}\boldsymbol{p}^{(y)\top} M G^{-1} M^\top \boldsymbol{p}^{(y)},$$

where $G = M^\top M + \lambda I$. If columns of M are linearly independent, then G is symmetric positive definite for all $\lambda \geq 0$. Thus $\hat{J}_y(\boldsymbol{\theta}^{(y)})$ is minimized when $M^\top \boldsymbol{p}^{(y)} - G\boldsymbol{\theta}^{(y)} = \boldsymbol{0}$, which leads to

$$(M^\top M + \lambda I)\boldsymbol{\theta}^{(y)} = M^\top \boldsymbol{p}^{(y)}. \tag{2.58}$$

After obtaining the solution of $\boldsymbol{\theta}^{(y)}$, we have (2.53). Then, the following normalization gives the approximated class probability:

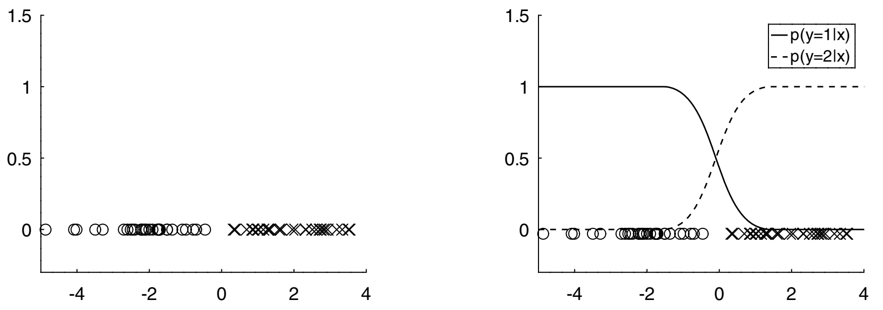
$$\hat{p}(y|\boldsymbol{x}) = \frac{\max(0, q(y|\boldsymbol{x}; \boldsymbol{\theta}^{(y)}))}{\sum_{\tilde{y}=1}^{c} \max(0, q(\tilde{y}|\boldsymbol{x}; \boldsymbol{\theta}^{(\tilde{y})}))}.$$

In what follows, a simple numerical example is illustrated to see how the least-squares probabilistic classification works. We use the training data as given in Fig. 2.13a, where there are two classes: $\circ$ denotes class $y = 1$ and $\times$ denotes class $y = 2$. From (2.53) and (2.54), the following model is used:

$$q(y|\boldsymbol{x}; \boldsymbol{\theta}^{(y)}) = \sum_{i:y_i=y} \theta_i^{(y)} K(\boldsymbol{x}, \boldsymbol{x}_i).$$

This means that we will obtain two function $q(y|\boldsymbol{x}; \boldsymbol{\theta}^{(y)})$ for $y = 1$ and $y = 2$, after solving two linear systems (2.58) for $y = 1$ and $y = 2$.

The result of the least-squares probabilistic classification is given in Fig. 2.13, where the number of samples $n = 60$, and the two parameters λ in (2.56) and h in (2.54) are $\lambda = 0.1$ and $h = 1$.



(a) Training samples of two classes. (b) The class-posterior probabilities.

Fig. 2.13 The result of the least-squares probabilistic classification

Fig. 2.13a shows the distribution of two classes of samples as mentioned before, and Fig. 2.13b is the corresponding result. From Fig. 2.13b, we see that the probability of the point $x = 0$ that belongs to class 1 ($y = 1$) is about 50%, i.e., $p(y = 1|x = 0) \approx 0.5$.

2.4 Matrix Equations

Consider the following matrix equation:

$$AX + XB = C, \tag{2.59}$$

where A and B are m-by-m and n-by-n matrices. The matrix equation is known as the *Sylvester equation*, which arises in the field of control theory. The Sylvester equation also arises in a derivation of Newton's method for computing the matrix square root, see Sect. 5.3.1.

The Sylvester equation can be transformed into linear systems of the form $M\boldsymbol{x} = \boldsymbol{c}$, and in what follows, we will see the connection between the Sylvester equation in (2.59) and linear systems $M\boldsymbol{x} = \boldsymbol{c}$.

Let G be an n-by-m matrix whose column vectors are written as $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_m$. Then, matrix G is written as $G = [\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_m]$. We now define *vec operator*. The vec operator is a map from $\mathbb{C}^{n\times m}$ to $\mathbb{C}^{nm}$, and vec(G) is written as

$$\mathrm{vec}(G) = \begin{bmatrix} \boldsymbol{g}_1 \\ \vdots \\ \boldsymbol{g}_m \end{bmatrix}. \tag{2.60}$$

Some properties of the vec operator are listed next.

Theorem 2.3 *Let $X \in \mathbb{C}^{m\times n}$, $A_i \in \mathbb{C}^{m\times m}$, $B_i \in \mathbb{C}^{n\times n}$, and P_{mn} be a permutation matrix given in Theorem 2.2. Then,*

(1) $\mathrm{vec}(A_1XB_1 + \cdots + A_kXB_k) = (B_1^\top \otimes A_1 + \cdots + B_k^\top \otimes A_k)\mathrm{vec}(X)$,
(2) $\mathrm{vec}(X^\top) = P_{mn}\mathrm{vec}(X)$.

Proof We first give a proof of (2). From the definition of P_{mn} in Theorem 2.2,

$$P_{mn}\mathrm{vec}(X) = \begin{bmatrix} (E_{11})^\top & \dots & (E_{1n})^\top \\ \vdots & \ddots & \vdots \\ (E_{m1})^\top & \dots & (E_{mn})^\top \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix},$$

and thus the kth block element of $P_{mn}\mathrm{vec}(X)$ is given by

$$(P_{mn}\mathrm{vec}(X))_k = \sum_{j=1}^n E_{kj}^\top \boldsymbol{x}_j = \sum_{j=1}^n E_{jk}\boldsymbol{x}_j = \sum_{j=1}^n \boldsymbol{e}_j\boldsymbol{e}_k^\top \boldsymbol{x}_j = \sum_{j=1}^n \boldsymbol{e}_j x_{kj} = \begin{bmatrix} x_{k1} \\ \vdots \\ x_{kn} \end{bmatrix},$$

which is the kth column of $X^\top$. Therefore,

$$P_{mn}\mathrm{vec}(X) = \begin{bmatrix} (P_{mn}\mathrm{vec}(X))_1 \\ \vdots \\ (P_{mn}\mathrm{vec}(X))_n \end{bmatrix} = \mathrm{vec}(X^\top).$$

Next, we give a proof of (1). Since $\mathrm{vec}(X+Y) = \mathrm{vec}(X) + \mathrm{vec}(Y)$, it is sufficient to prove $\mathrm{vec}(AXB) = (B^\top \otimes A)\mathrm{vec}(X)$. Let $M = XB$ and $M = [\boldsymbol{m}_1, \dots, \boldsymbol{m}_n]$, then $AM = [A\boldsymbol{m}_1, \dots, A\boldsymbol{m}_n]$. Thus

$$\mathrm{vec}(AXB) = \mathrm{vec}(AM) = \begin{bmatrix} A\boldsymbol{m}_1 \\ \vdots \\ A\boldsymbol{m}_n \end{bmatrix} = (I \otimes A)\begin{bmatrix} \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{m}_n \end{bmatrix} = (I_n \otimes A)\mathrm{vec}(M), \quad (2.61)$$

and the vec(M) is calculated as

$$\begin{aligned}\mathrm{vec}(XB) &= \mathrm{vec}((B^\top X^\top)^\top) = P_{nm}\mathrm{vec}(B^\top X^\top) = P_{nm}(I_m \otimes B^\top)\mathrm{vec}(X^\top) \\ &= P_{nm}(I_m \otimes B^\top)P_{mn}\mathrm{vec}(X) = P_{mn}^\top(I_m \otimes B^\top)P_{mn}\mathrm{vec}(X) \\ &= (B^\top \otimes I_m)\mathrm{vec}(X).\end{aligned}$$

The last equation holds from (2.13), and this result together with (2.61) yields

$$\mathrm{vec}(AXB) = (I_n \otimes A)\mathrm{vec}(XB) = (I_n \otimes A)(B^\top \otimes I_m)\mathrm{vec}(X) = (B^\top \otimes A)\mathrm{vec}(X).$$

The last equation follows from Theorem 2.1-(6), which concludes the proof. □

We are now ready to describe a relation between the Sylvester equation and linear systems. Applying Theorem 2.3-(1) to Sylvester Eq. (2.59) yields

$$\mathrm{vec}(AX + XB) = \mathrm{vec}(C) \Leftrightarrow \underbrace{[(I_n \otimes A) + (B^\top \otimes I_m)]}_{M}\underbrace{\mathrm{vec}(X)}_{\boldsymbol{x}} = \underbrace{\mathrm{vec}(C)}_{\boldsymbol{c}},$$

where I_m is the m-by-m identity matrix.

Here we consider a slightly modified matrix equation of the form

$$AX + X^\top B = C.$$

The matrix is referred to as the *T-congruence Sylvester equation*. Applying the vec operator and using Theorem 2.3-(2) yields

$$\begin{aligned}
AX + X^\top B = C &\Leftrightarrow \mathrm{vec}(AX + X^\top B) = \mathrm{vec}(C)\\
&\Leftrightarrow \mathrm{vec}(AX) + \mathrm{vec}(X^\top B) = \mathrm{vec}(C)\\
&\Leftrightarrow \mathrm{vec}(I_n \otimes A)\mathrm{vec}(X) + (B^\top \otimes I_m)\mathrm{vec}(X^\top) = \mathrm{vec}(C)\\
&\Leftrightarrow (I_n \otimes A)\mathrm{vec}(X) + (B^\top \otimes I_m)P_{mn}\mathrm{vec}(X) = \mathrm{vec}(C)\\
&\Leftrightarrow \underbrace{[(I_n \otimes A) + (B^\top \otimes I_m)P_{mn}]}_{\tilde{M}}\underbrace{\mathrm{vec}(X)}_{\boldsymbol{x}} = \underbrace{\mathrm{vec}(C)}_{\boldsymbol{c}}.
\end{aligned}$$

Therefore, the T-congruence Sylvester equation is also equivalent to linear systems of the form $\tilde{M}\boldsymbol{x} = \boldsymbol{c}$.

2.5 Optimization

Consider unconstrained optimization problems of the form

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}), \tag{2.62}$$

where $f(\boldsymbol{x})$ is a smooth function. Unless function $f(\boldsymbol{x})$ has a special structure, e.g., a convex function, it is in general difficult to find the solution of the minimization problems. In order to find the solution, as is known in standard calculus, the critical points, $\boldsymbol{x}_\mathrm{c}$ such that $\nabla f(\boldsymbol{x}_\mathrm{c}) = \mathbf{0}$, are usually important.

In this section, Newton's method is introduced over a Euclidean space and a Riemannian manifold (Grassmann manifold) to obtain the critical points, and we will see how linear systems arise in each iteration step of Newton's method. As a preliminary, tensor notations are introduced in the next subsection. We will see that tensor notations enable us to describe the Taylor series systematically and concisely.

2.5.1 Tensor Notations

A matrix denoted by A is a two-dimensional array that has (i, j) elements, which are written as $(A)_{ij}$ or a_{ij}. A *tensor* denoted by $\mathcal{A}$ is a multidimensional array that has, for a three-dimensional array, (i, j, k) elements, which are written as $(\mathcal{A})_{ijk}$ or a_{ijk}.

More generally, an Nth-order tensor (or an N-way tensor) $\mathcal{A}$ is an N-dimensional array whose $i_1, i_2, \ldots, i_N$ elements are written as $(\mathcal{A})_{i_1,i_2,\ldots,i_N}$ or $a_{i_1,i_2,\ldots,i_N}$. If all the elements $a_{i_1,i_2,\ldots,i_N} \in \mathbb{R}$ with $i_1, i_2, \ldots, i_N$ running from $i_1 = i_2 = \cdots = i_N = 1$ to $i_1 = I_1, i_2 = I_2, \ldots, i_N = I_N$, then $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$.

Given two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a scalar value $\alpha \in \mathbb{R}$, tensor addition and scalar multiplication are defined by

$$(\mathcal{X} + \mathcal{Y})_{i_1,i_2,\ldots,i_N} = x_{i_1,i_2,\ldots,i_N} + y_{i_1,i_2,\ldots,i_N}, \quad (\alpha\mathcal{X})_{i_1,i_2,\ldots,i_N} = \alpha x_{i_1,i_2,\ldots,i_N}.$$

A tensor–matrix multiplication, or *n-mode product*, is defined next. Let $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and $U \in \mathbb{R}^{J \times I_n}$. The n-mode product of $\mathcal{X}$ and U is defined as

$$(\mathcal{X} \times_n U)_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \sum_{k=1}^{I_n} x_{i_1,i_2,\ldots,i_{n-1},k,i_{n+1},\ldots,i_N} u_{j,k}.$$

For a third-order tensor,

$$(\mathcal{X} \times_1 U)_{j,i_2,i_3} = \sum_{k=1}^{I_1} x_{k,i_2,i_3} u_{j,k},$$

$$(\mathcal{X} \times_2 U)_{i_1,j,i_3} = \sum_{k=1}^{I_2} x_{i_1,k,i_3} u_{j,k},$$

$$(\mathcal{X} \times_3 U)_{i_1,i_2,j} = \sum_{k=1}^{I_3} x_{i_1,i_2,k} u_{j,k}.$$

From the definition, it is easy to see

$$(\mathcal{X} \times_1 A) \times_2 B = (\mathcal{X} \times_2 B) \times_1 A. \tag{2.63}$$

Thus, one may write

$$\mathcal{X} \times_1 A \times_2 B,$$

and more generally $\mathcal{X} \times_1 A_1 \times_2 A_2 \times_3 \cdots \times_N A_N$. If $\mathcal{X}$ is a matrix (rewritten as X, instead of $\mathcal{X}$), (2.63) is equivalent to

$$(AX)B^\top = A(XB^\top).$$

Thus, in standard linear algebra, the parentheses are omitted, i.e., $AXB^\top$. The singular value decomposition of matrix X is $U\Sigma V^{\mathrm{H}}$, where U and V are unitary matrices and Σ is a diagonal matrix whose diagonal elements are $(\Sigma)_{i,i} = \sigma_i \geq 0$. Then using tensor notations, the singular value decomposition of X can be rewritten as

$$X = U\Sigma V^\top = \Sigma \times_1 U \times_2 V.$$

A more general product is a *contracted product* $\langle \cdot, \cdot \rangle$, which is the following map:

$$\begin{aligned}\langle \cdot, \cdot \rangle_{1,\ldots,m;1,\ldots,m} &: \mathbb{R}^{I_1\times\cdots\times I_m\times J_1\times\cdots\times J_N} \times \mathbb{R}^{I_1\times\cdots\times I_m\times K_1\times\cdots\times K_p} \\ &\to \mathbb{R}^{J_1\times\cdots\times J_N\times K_1\times\cdots\times K_p}, \\ \langle \mathcal{X}, \mathcal{Y} \rangle_{1,\ldots,m;1,\ldots,m} &:= \sum_{i_1=1}^{I_1}\cdots\sum_{i_m=1}^{I_m} x_{i_1,\ldots,i_m,j_1,\ldots,j_N} y_{i_1,\ldots,i_m,k_1,\ldots,k_p}.\end{aligned}$$

More generally, some examples of contracted products of two tensors are given next. For $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times I_3}$ and $\mathcal{Y} \in \mathbb{R}^{J_1\times J_2\times J_3\times J_4}$, below is an example of contracted products of the two tensors.

$$\langle \mathcal{X}, \mathcal{Y} \rangle_{1,3:2,3} = \sum_{k=1}^{I_1}\sum_{l=1}^{I_3} x_{k,i_2,l} \times y_{j_1,k,l,j_4},$$

where we assumed $I_1 = J_2$ and $I_3 = J_3$. Another example is as follows:

$$\langle \mathcal{X}, \mathcal{Y} \rangle_{1,2:2,4} = \sum_{k=1}^{I_1}\sum_{l=1}^{I_2} x_{k,l,i_3} \times y_{j_1,k,j_3,l},$$

where we assumed $I_1 = J_2$ and $I_2 = J_4$.

2.5.2 Newton's Method on Euclidean Space

For a smooth function $f(\boldsymbol{x})$ with $\boldsymbol{x} \in \mathbb{R}^n$, the mode-$n$ product of a tensor and a vector (a matrix having only one column) is useful for describing the Taylor series of $f(\boldsymbol{x}+\boldsymbol{h})$ at $\boldsymbol{x}$:

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \frac{1}{1!}[f^{(1)}(\boldsymbol{x})] \times_1 \boldsymbol{h} \tag{2.64}$$

$$+ \frac{1}{2!}[f^{(2)}(\boldsymbol{x})] \times_1 \boldsymbol{h} \times_2 \boldsymbol{h} + \frac{1}{3!}[f^{(3)}(\boldsymbol{x})] \times_1 \boldsymbol{h} \times_2 \boldsymbol{h} \times_3 \boldsymbol{h} + \cdots, \tag{2.65}$$

where

$$
\begin{aligned}
[f^{(1)}(\boldsymbol{x})]_{i_1} &= \frac{\partial}{\partial x_{i_1}} f(\boldsymbol{x}), \quad [f^{(1)}(\boldsymbol{x})] \in \mathbb{R}^n, \\
[f^{(2)}(\boldsymbol{x})]_{i_1 i_2} &= \frac{\partial^2}{\partial x_{i_1} \partial x_{i_2}} f(\boldsymbol{x}), \quad [f^{(2)}(\boldsymbol{x})] \in \mathbb{R}^{n \times n}, \\
[f^{(3)}(\boldsymbol{x})]_{i_1 i_2 i_3} &= \frac{\partial^3}{\partial x_{i_1} \partial x_{i_2} \partial x_{i_3}} f(\boldsymbol{x}), \quad [f^{(3)}(\boldsymbol{x})] \in \mathbb{R}^{n \times n \times n}, \\
&\vdots
\end{aligned}
$$

for $i_1, i_2, i_3, \ldots \in \{1, 2, \ldots, n\}$. The vector $f^{(1)}(\boldsymbol{x})$ is the gradient of $f(\boldsymbol{x})$, i.e., $f^{(1)}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$. The matrix $f^{(2)}(\boldsymbol{x})$ is called a *Hessian matrix*, which is denoted by $H(\boldsymbol{x})$. With this notation, the Taylor series (2.65) can be rewritten as

$$
f(\boldsymbol{x} + \boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^\top \nabla f(\boldsymbol{x}) + \frac{1}{2} \boldsymbol{h}^\top H(\boldsymbol{x}) \boldsymbol{h} + \frac{1}{3!} [f^{(3)}(\boldsymbol{x})] \times_1 \boldsymbol{h} \times_2 \boldsymbol{h} \times_3 \boldsymbol{h} + \cdots . \tag{2.66}
$$

We now consider finding a local minimizer $\boldsymbol{x}^*$ that locally minimizes $f(\boldsymbol{x})$. Let $\boldsymbol{x}_n$ be an approximate minimizer of $f(\boldsymbol{x})$ (i.e., $\boldsymbol{x}_n \approx \boldsymbol{x}^*$). Then, the original problem is transformed into finding a correction vector $\boldsymbol{h}$ so that $f(\boldsymbol{x}_k + \boldsymbol{h})$ is minimized.

Newton's method finds a correction vector $\boldsymbol{h}$ such that $\boldsymbol{x}_k + \boldsymbol{h}$ minimizes not $f(\boldsymbol{x}_k + \boldsymbol{h})$ but the quadratic approximation of $f(\boldsymbol{x}_k + \boldsymbol{h})$, i.e.,

$$
\min_{\boldsymbol{h} \in \mathbb{R}^n} q(\boldsymbol{h}), \quad q(\boldsymbol{h}) := f(\boldsymbol{x}_k) + \boldsymbol{h}^\top \nabla f(\boldsymbol{x}_k) + \frac{1}{2} \boldsymbol{h}^\top H(\boldsymbol{x}_k) \boldsymbol{h}.
$$

If $H(\boldsymbol{x}_k)$ is positive definite, the minimizer of $q(\boldsymbol{h})$ is unique and is given by the solution of $\nabla q(\boldsymbol{h}) = \boldsymbol{0}$. Since $\nabla q(\boldsymbol{h}) = \nabla f(\boldsymbol{x}_k) + H(\boldsymbol{x}_k)\boldsymbol{h}$, the correction vector $\boldsymbol{h}$ at the kth iteration step, denoted by $\boldsymbol{h}_k$, can be obtained by solving linear systems of the form

$$
H(\boldsymbol{x}_k) \boldsymbol{h}_k = -\nabla f(\boldsymbol{x}_k). \tag{2.67}
$$

After solving (2.67), we obtain a new approximate minimizer $\boldsymbol{x}_{k+1} (= \boldsymbol{x}_k + \boldsymbol{h}_k)$. At each iteration k, we need to solve the linear systems (2.67).

2.5.3 Newton's Method on Riemannian Manifold

Similar to the previous section, linear systems also arise in minimization problems over Riemannian manifolds when we consider Newton's method over Riemannian manifolds. As an example, we consider an optimization problem over the Grassmann manifold, which is an example of Riemannian manifolds.

We consider the following minimization problems

$$\min_{X \in \mathbb{R}^{n \times p}} f(X), \tag{2.68}$$

where f is assumed to be a smooth function. Using vec operator (2.60), the function in (2.68) is equivalent to $f(\tilde{\boldsymbol{x}})$, where $\tilde{\boldsymbol{x}} = \mathrm{vec}(X)$. Thus the minimization problem (2.68) is essentially the same as (2.62).

If X satisfies the condition $X^\top X = I_p$, then the minimization problem (2.68) can be written as

$$\min_{X \in \mathrm{St}(p,n)} f(X), \tag{2.69}$$

where $\mathrm{St}(p, n) = \{X \in \mathbb{R}^{n\times p} : X^\top X = I_p\}$ $(p \le n)$ and $\mathrm{St}(p, n)$ is referred to as the *Stiefel manifold*. The condition $X^\top X = I_p$ means that all the column vectors $\boldsymbol{x}_i$ $(i = 1, \dots, p)$ of X are orthonormal, i.e., $\boldsymbol{x}_i^\top \boldsymbol{x}_j = 0$ for $i \neq j$ and $\boldsymbol{x}_i^\top \boldsymbol{x}_j = 1$ for $i = j$.

We now further assume that the function f has the following property:

$$f(XQ) = f(X) \tag{2.70}$$

for any orthogonal matrix $Q \in \mathbb{R}^{p\times p}$. This property leads to difficulty in solving the minimization problem. To be specific, let X_k be an approximate minimizer of $f(X)$ and let X_{k+1} be the next iterate given by $X_k Q_k$ for some orthogonal matrix Q_k. Then the iterate X_{k+1} is not an improved approximate solution because the value of the objective function does not change. In order to avoid such a situation, one promising approach is that for a given $X_k \in \mathrm{St}(p, n)$ we regard the set $[X_k] := \{X_k Q : Q^\top Q = I_p\}$ as one element. Then the next iterate $[X_{k+1}](\neq [X_k])$ is meaningful because X_{k+1} can avoid the situation $f(X_k) = f(X_{k+1})$ caused by $X_{k+1} = X_k Q_k$.

In general, the set $\mathrm{Grass}(p, n) := \{[X] : X \in \mathrm{St}(p, n)\}$ is referred to as the *Grassmann manifold*. The approach mentioned above corresponds to considering the following minimization problem:

$$\min_{[Y] \in \mathrm{Grass}(p,n)} f(Y), \tag{2.71}$$

instead of considering (2.69) with (2.70). In numerical computation, the following approximate solutions are produced: $X_0 \in [Y_0], X_1 \in [Y_1], \dots$, i.e., representatives of the equivalence class $[Y_k]$ for $k = 0, 1, \dots$.

The minimization problem (2.71) can be regarded as an unconstrained minimization problem over the Grassmann manifold. Similar to (2.67), Newton's method over the Grassmann manifold requires us to solve the following linear systems (*Newton-Grassmann equation*):

$$Y_{k\perp}^{\top}\partial^2 f(Y_k)[Y_{k\perp}C_k] - C_k Y_k^{\top}\partial f(Y_k) = \underbrace{-Y_{k\perp}^{\top}\partial f(Y_k)}_{(n-p)\times p}, \tag{2.72}$$

where $Y_k \in \mathrm{St}(p, n)$, $Y_{k\perp} \in \mathrm{St}(n-p, n)$ whose column vectors are orthogonal to the column vectors of Y_k, and the unknown matrix to be solved is $C_k \in \mathbb{R}^{(n-p)\times p}$. Here the (i, j) elements of $\partial f(Y) \in \mathbb{R}^{n\times p}$ ($Y \in \mathbb{R}^{n\times p}$) and $\partial^2 f(Y)[Z] \in \mathbb{R}^{n\times p}$ ($Z \in \mathbb{R}^{n\times p}$) are given by

$$(\partial f(Y))_{ij} = \frac{\partial}{\partial y_{ij}} f(Y),$$

$$(\partial^2 f(Y)[Z])_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{p} \frac{\partial^2}{\partial y_{ij}\partial y_{kl}} f(Y) \times z_{kl}.$$

We now use a fourth-order tensor $\mathcal{G} \in \mathbb{R}^{n\times p\times n\times p}$ whose (i, j, k, l) element is defined by

$$(\mathcal{G})_{ijkl} = \frac{\partial^2}{\partial y_{ij}\partial y_{kl}} f(Y).$$

Then, $\partial^2 f(Y_k)[Z]$ can be regarded as the following contracted product of $\mathcal{G}$ and Z:

$$\partial^2 f(Y_k)[Y_{k\perp}C_k] = \langle \mathcal{G}, Z\rangle_{3,4;1,2}\,.$$

For the contracted product, see Sect. 2.5.1.

After obtaining C_k by solving (2.72), one may think the next iterate is $Y_{k+1} = Y_{k\perp}C_k$. However, in general, $Y_{k\perp}C_k$ does not belong to the Grassmann manifold. Therefore $Y_{k\perp}C_k$ needs to be mapped to the Grassmann manifold, and the geometrically suitable map is referred to as retraction. The next iterate Y_{k+1} is the matrix over the Grassmann manifold that is obtained from $Y_{k\perp}C_k$ by using one of the retractions. For the details of optimization over matrix manifolds, see [3, 56, 155].

2.5.3.1 Example

As an example, we consider minimizing the following Rayleigh quotient on the Grassman manifold:

$$\min_{[Y]\in \mathrm{Grass}(p,n)} f(Y), \quad f(Y) := \frac{1}{2}\mathrm{tr}(Y^{\top}AY),$$

where $Y \in \mathbb{R}^{n\times p}$ and $A \in \mathbb{R}^{n\times n}$. Let $Q \in \mathbb{R}^{p\times p}$ be an orthogonal matrix. Then, $f(YQ) = \mathrm{tr}((YQ)^{\top}AYQ)/2 = \mathrm{tr}(Q^{\top}Y^{\top}AYQ)/2 = \mathrm{tr}(Y^{\top}AYQQ^{\top})/2 = \mathrm{tr}(Y^{\top}AY)/2 = f(Y)$. Thus we consider $\{YQ \in \mathbb{R}^{n\times p} : Q^{\top}Q = I_p\}$ as one element (i.e., equivalence class), which is denoted by $[Y]$. In what follows, we calculate $\partial f(Y)$ and

$\partial^2 f(Y)[Z]$. The (i,j) element of $\partial f(Y)$ can be calculated as follows:

$$\begin{aligned}
(\partial f(Y))_{ij} &= \frac{\partial}{\partial y_{ij}} f(Y) = \frac{\partial}{\partial y_{ij}} \frac{1}{2}\mathrm{tr}(Y^\top AY) = \frac{1}{2}\frac{\partial}{\partial y_{ij}} \sum_{k,l,m} y_{kl}^\top a_{lm} y_{mk} \\
&= \frac{1}{2}\frac{\partial}{\partial y_{ij}} \sum_{k,l,m} y_{lk} a_{lm} y_{mk} = \frac{1}{2}\sum_{k,l,m} \frac{\partial y_{lk}}{\partial y_{ij}} a_{lm} y_{mk} + \frac{1}{2}\sum_{k,l,m} y_{lk} a_{lm} \frac{\partial y_{mk}}{\partial y_{ij}} \\
&= \frac{1}{2}\sum_{m} \frac{\partial y_{ij}}{\partial y_{ij}} a_{im} y_{mj} + \frac{1}{2}\sum_{l} y_{lj} a_{li} \frac{\partial y_{ij}}{\partial y_{ij}} = \frac{1}{2}\sum_{m} a_{im} y_{mj} + \frac{1}{2}\sum_{l} y_{lj} a_{li} \\
&= \frac{1}{2}\sum_{m} a_{im} y_{mj} + \frac{1}{2}\sum_{l} a_{il}^\top y_{lj} \\
&= \frac{1}{2}(AY)_{ij} + \frac{1}{2}(A^\top Y)_{ij}.
\end{aligned}$$

Here we used the symbol "$^\top$" for scalar value $y_{kl}^\top$, which means y_{lk}. The notation is convenient if one calculates the (i,j) element of $A^\top B$. From the definition of matrix–matrix multiplications, we have $(AB)_{ij} = \sum_k a_{ik} b_{kj}$. Using the notation, for the (i,j) element of $A^\top B$, we have $(A^\top B)_{ij} = \sum_k a_{ik}^\top b_{kj} = \sum_k a_{ki} b_{kj}$.

Since $A = A^\top$, it follows that

$$\partial f(Y) = AY. \tag{2.73}$$

Next, the (i,j) element of $\partial^2 f(Y)[Z]$ is given by

$$\begin{aligned}
(\partial^2 f(Y)[Z])_{ij} &= \sum_{k,l} \frac{\partial^2}{\partial y_{ij}\partial y_{kl}} f(Y) \times z_{kl} = \sum_{k,l} \frac{\partial}{\partial y_{ij}} \left(\frac{\partial}{\partial y_{kl}} f(Y) \right) \times z_{kl} \\
&= \sum_{k,l} \frac{\partial}{\partial y_{ij}} \left(\frac{1}{2}\sum_m a_{km} y_{ml} + \frac{1}{2}\sum_m a_{km}^\top y_{ml} \right) \times z_{kl} \\
&= \sum_{k,l} \left(\frac{1}{2}\sum_m a_{km} \frac{\partial y_{ml}}{\partial y_{ij}} + \frac{1}{2}\sum_m a_{km}^\top \frac{\partial y_{ml}}{\partial y_{ij}} \right) \times z_{kl} \\
&= \sum_{k} \left(\frac{1}{2} a_{ki} \frac{\partial y_{ij}}{\partial y_{ij}} + \frac{1}{2} a_{ki}^\top \frac{\partial y_{ij}}{\partial y_{ij}} \right) \times z_{kj} \\
&= \sum_{k} \left(\frac{1}{2} a_{ki} z_{kj} + \frac{1}{2} a_{ki}^\top z_{kj} \right) \\
&= \sum_{k} \left(\frac{1}{2} a_{ik}^\top z_{kj} + \frac{1}{2} a_{ik} z_{kj} \right) \\
&= \left(\frac{1}{2}(A^\top Z)_{ij} + \frac{1}{2}(AZ)_{ij} \right).
\end{aligned}$$

Since $A = A^\top$, we obtain

$$\partial^2 \bar{f}(Y)[Z] = AZ. \tag{2.74}$$

Substituting (2.73) and (2.74) into Newton–Grassmann Eq. (2.72) yields

$$(Y_{k\perp}^\top A Y_{k\perp}) C_k - C_k (Y_k^\top A Y_k) = -Y_{k\perp}^\top A Y_k, \tag{2.75}$$

where C is an unknown matrix to be solved. This is the Sylvester equation as described in (2.59).

After obtaining C_k by solving (2.75), the next iterate Y_{k+1} is, for example, given by $Y_{k+1} = Q_k$, where Q_k is the (thin) QR factorization of $Y_{k\perp} C_k$.

Chapter 3
Classification and Theory of Krylov Subspace Methods

Abstract Krylov subspace methods are roughly classified into three groups: ones for Hermitian linear systems, for complex symmetric linear systems, and for non-Hermitian linear systems. Non-Hermitian linear systems include complex symmetric linear systems since a complex symmetric matrix is non-Hermitian and symmetric. Krylov subspace methods for complex symmetric linear systems use the symmetry of the coefficient matrix, leading to more efficient Krylov subspace methods than ones for non-Hermitian linear systems. This chapter also presents preconditioning techniques to boost the speed of convergence of Krylov subspace methods.

3.1 Hermitian Linear Systems

In this section, we give derivations of the Conjugate Gradient (CG) method, the Conjugate Residual (CR) method, and the Minimal Residual (MINRES) method. These methods are used for the case where matrix A is Hermitian (or real symmetric). The CG method is usually used for the case where A is a Hermitian positive definite matrix, and the CR method and the MINRES method are used for a Hermitian (indefinite) matrix. In exact precision arithmetic, the CR method and the MINRES method produce the same approximate solutions. We will see that these methods find the best approximate solutions at each iteration step. To be specific, the CG method for a Hermitian positive definite matrix finds the best approximate solution such that a weighted norm of the error is minimized. On the other hand, the CR method and the MINRES method find the best approximate solution such that the 2-norm of the residual is minimized.

3.1.1 The Conjugate Gradient (CG) Method

The CG method [95] was proposed by Hestenes (1906–1991) and Stiefel (1909–1978) in 1952, and is the best-known Krylov subspace method. In exact precision arithmetic, as well as other Krylov subspace methods the CG method produces the

T. Sogabe, *Krylov Subspace Methods for Linear Systems*,
Springer Series in Computational Mathematics 60,
https://doi.org/10.1007/978-981-19-8532-4_3

exact solution within finite iteration steps. This means that the CG method has two features: a direct method as in Sect. 1.3 and an iterative method as in Sect. 1.6. For the chronological history of the development of the CG method, see an excellent review by Golub and O'Leary [80].

In what follows, the CG method is derived from the Lanczos process (Algorithm 1.11). From (1.42), the Lanczos process in matrix form is given by

$$AR_n = R_{n+1}T_{n+1,n}, \tag{3.1}$$

where

$$\begin{aligned}
R_n &:= [\boldsymbol{r}_0, \boldsymbol{r}_1, \dots, \boldsymbol{r}_{n-1}], \\
T_{n+1,n} &= \begin{bmatrix} t_{0,0} & t_{0,1} & & \\ -\alpha_0^{-1} & t_{1,1} & \ddots & \\ & -\alpha_1^{-1} & \ddots & t_{n-2,n-1} \\ & & \ddots & t_{n-1,n-1} \\ & & & -\alpha_{n-1}^{-1} \end{bmatrix}, \\
t_{k-1,k} &:= \frac{(\boldsymbol{r}_{k-1}, A\boldsymbol{r}_k)}{(\boldsymbol{r}_{k-1}, \boldsymbol{r}_{k-1})}, \quad (k = 1, 2, \dots, n-1), \\
t_{k,k} &:= \frac{(\boldsymbol{r}_k, A\boldsymbol{r}_k)}{(\boldsymbol{r}_k, \boldsymbol{r}_k)}, \quad (k = 0, 1, \dots, n-1).
\end{aligned}$$

Here, the scaling parameters (α_k's) have not been determined yet. Let $\boldsymbol{x}_k$ be the kth approximate solution of $A\boldsymbol{x} = \boldsymbol{b}$ such that $\boldsymbol{x}_k$'s satisfy the following equation:

$$R_n = [\boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{b} - A\boldsymbol{x}_1, \dots, \boldsymbol{b} - A\boldsymbol{x}_{n-1}] = \boldsymbol{b}\mathbf{1}_n^\top - AX_n,$$

where $\mathbf{1}_n := [1, 1, \dots, 1]^\top \in \mathbb{R}^n$ and $X_n := [\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_{n-1}]$. Notice that R_n is the matrix given in (3.1). Then, it follows that

$$\begin{aligned}
AR_n &= R_{n+1}T_{n+1,n} = (\boldsymbol{b}\mathbf{1}_{n+1}^\top - AX_{n+1})T_{n+1,n} \\
\Leftrightarrow R_n &= A^{-1}(\boldsymbol{b}\mathbf{1}_{n+1}^\top - AX_{n+1})T_{n+1,n} = \boldsymbol{x}\mathbf{1}_{n+1}^\top T_{n+1,n} - X_{n+1}T_{n+1,n}.
\end{aligned} \tag{3.2}$$

Now we use the condition $\mathbf{1}_{n+1}^\top T_{n+1,n} = [0, 0, \dots, 0]$ to determine scaling parameters α_k, i.e., $\alpha_0^{-1} = t_{0,0}$ and $\alpha_{k-1}^{-1} = t_{k-1,k-1} + t_{k-2,k-1}$ for $k = 2, 3, \dots, n$. Then, the condition and (3.2) yield

$$R_n = -X_{n+1}T_{n+1,n}, \tag{3.3}$$

where the matrix $T_{n+1,n}$ in (3.1) is rewritten as

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

and the matrix $T_{n+1,n}$ has the following LDU decomposition:

$$T_{n+1,n} = \underbrace{\begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \alpha_0^{-1} & & & \\ & \alpha_1^{-1} & & \\ & & \ddots & \\ & & & \alpha_{n-1}^{-1} \end{bmatrix}}_{D^{-1}} \underbrace{\begin{bmatrix} 1 & \alpha_0 t_{0,1} & & \\ & 1 & \ddots & \\ & & \ddots & \alpha_{n-1} t_{n-1,n} \\ & & & 1 \end{bmatrix}}_{U}, \quad (3.4)$$

where L is the $n+1$-by-n lower bidiagonal matrix, D^{-1} is the n-by-n diagonal matrix, and U is the n-by-n upper bidiagonal matrix. Thus, we obtain

$$R_n = -X_{n+1} L D^{-1} U. \quad (3.5)$$

We now define $P_n (= [\boldsymbol{p}_0, \boldsymbol{p}_1, \dots, \boldsymbol{p}_{n-1}]) := R_n U^{-1}$. Then, (3.5) can be rewritten as

$$P_n D = X_{n+1} (-L), \quad (3.6)$$

which is equivalent to

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \quad k = 0, 1, \dots, n-1. \quad (3.7)$$

The relation between $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ and (3.7) yields

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k, \quad k = 0, 1, \dots, n-1. \quad (3.8)$$

The definition $P_n = R_n U^{-1}$ leads to $P_n U = R_n$, which is equivalent to

$$\boldsymbol{p}_0 = \boldsymbol{r}_0, \quad (3.9)$$

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_{k-1} \boldsymbol{p}_{k-1}, \quad k = 1, 2, \dots, n-1, \quad (3.10)$$

where $\beta_{k-1} := -\alpha_{k-1} t_{k-1,k}$.

As described above, computing α_k requires $A\boldsymbol{r}_k$ because $\alpha_k = (t_{k,k} + t_{k-1,k})^{-1}$, which means two matrix–vector multiplications are required per iteration step, i.e., $A\boldsymbol{r}_k$ and $A\boldsymbol{p}_k$ if we use the iterates (3.7)–(3.10). In what follows, we will see that $A\boldsymbol{r}_k$ is not required. To see this, first we derive a computational formula for α_k.

From (3.1), (3.4), and the definition of P_n, it follows that

$$AP_n = R_{n+1}LD^{-1}.$$

This leads to

$$P_n^{\mathrm{H}}AP_nD = U^{-\mathrm{H}}R_n^{\mathrm{H}}R_{n+1}L. \tag{3.11}$$

Since $R_n = [\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n-1}]$ in (3.1) is generated by the Lanczos process, we have $\boldsymbol{r}_i^{\mathrm{H}}\boldsymbol{r}_j = 0$ for $i \neq j$. In matrix form, this means (i, j) elements of $R_n^{\mathrm{H}}R_{n+1}$ is zero for $i \neq j$. This fact yields that the $(k+1, k+1)$ element of $U^{-\mathrm{H}}R_n^{\mathrm{H}}R_{n+1}L$ in (3.11) is $\boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k$ because L and $U^{-\mathrm{H}}$ are the lower triangular matrices with all the diagonals being one. Thus from the $(k+1, k+1)$ element in (3.11), we have $\boldsymbol{p}_k^{\mathrm{H}}A\boldsymbol{p}_k\alpha_k = \boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k$, or

$$\alpha_k = \frac{\boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k}{\boldsymbol{p}_k^{\mathrm{H}}A\boldsymbol{p}_k}. \tag{3.12}$$

Since A is Hermitian, α_k is a real number for all k, which will be later used for deriving the computational formula for β_k.

Next, we give a computational formula for β_k. From (3.1), it follows that

$$R_n^{\mathrm{H}}AR_n = R_n^{\mathrm{H}}R_{n+1}T_{n+1,n}, \tag{3.13}$$

The $(k+1, k)$ element of $R_n^{\mathrm{H}}AR_n$ is $\boldsymbol{r}_k^{\mathrm{H}}A\boldsymbol{r}_{k-1}$. As seen in (3.11), $R_n^{\mathrm{H}}R_{n+1}$ is a matrix whose (k, k) elements are $\boldsymbol{r}_{k-1}^{\mathrm{H}}\boldsymbol{r}_{k-1}$ for $k = 1, 2 \ldots, n$ and the other elements are zero. From this, it is seen that the $(k+1, k)$ element of $R_n^{\mathrm{H}}R_{n+1}T_{n+1,n}$ is $\boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k \times (-\alpha_{k-1})^{-1}$. Thus we obtain $\boldsymbol{r}_k^{\mathrm{H}}A\boldsymbol{r}_{k-1} = \boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k \times (-\alpha_{k-1})^{-1}$. Because α_{k-1} is a real number as mentioned before, $\boldsymbol{r}_k^{\mathrm{H}}A\boldsymbol{r}_{k-1}$ is also a real number. Thus $\boldsymbol{r}_k^{\mathrm{H}}A\boldsymbol{r}_{k-1} = (\boldsymbol{r}_k^{\mathrm{H}}A\boldsymbol{r}_{k-1})^{\mathrm{H}} = \boldsymbol{r}_{k-1}^{\mathrm{H}}A^{\mathrm{H}}\boldsymbol{r}_k = \boldsymbol{r}_{k-1}^{\mathrm{H}}A\boldsymbol{r}_k$. From this fact it follows that $\boldsymbol{r}_{k-1}^{\mathrm{H}}A\boldsymbol{r}_k = \boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k \times (-\alpha_{k-1})^{-1}$. Recalling $\beta_{k-1} = -\alpha_{k-1}t_{k-1,k}$ yields

$$\beta_{k-1} = -\alpha_{k-1}t_{k-1,k} = \frac{\boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k}{\boldsymbol{r}_{k-1}^{\mathrm{H}}A\boldsymbol{r}_k}\frac{\boldsymbol{r}_{k-1}^{\mathrm{H}}A\boldsymbol{r}_k}{\boldsymbol{r}_{k-1}^{\mathrm{H}}\boldsymbol{r}_{k-1}} = \frac{\boldsymbol{r}_k^{\mathrm{H}}\boldsymbol{r}_k}{\boldsymbol{r}_{k-1}^{\mathrm{H}}\boldsymbol{r}_{k-1}}. \tag{3.14}$$

From (3.7)–(3.10), (3.12), and (3.14), the algorithm of the CG method is obtained, which is listed in Algorithm 3.1.

Properties of the CG method are described in Proposition 3.1.

Proposition 3.1 *Let $\boldsymbol{p}_n, \boldsymbol{r}_n$ be the vectors in the CG method, then:*

1. $(\boldsymbol{r}_i, \boldsymbol{r}_j) = 0$ *for* $i \neq j$,
2. $(\boldsymbol{p}_i, A\boldsymbol{p}_j) = 0$ *for* $i \neq j$.

Proof The first property readily follows from the fact that (3.1) is obtained from the Lanczos process. For the second property, it follows from (3.11) that we have

Algorithm 3.1 The CG method

```
Input: $x_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \dots,$ until convergence do
2:     $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $\alpha_n = \frac{(\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{p}_n, A\boldsymbol{p}_n)}$
4:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
5:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
6:     $\beta_n = \frac{(\boldsymbol{r}_{n+1}, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n, \boldsymbol{r}_n)}$
7: end for
```

$$P_n^{\mathrm{H}} A P_n = U^{-\mathrm{H}} R_n^{\mathrm{H}} R_{n+1} L D^{-1}.$$

$P_n^{\mathrm{H}} A P_n$ is Hermitian because A is Hermitian. Thus, $U^{-\mathrm{H}} R_n^{\mathrm{H}} R_{n+1} L D^{-1}$ is Hermitian. Since the (i, j) element $R_n^{\mathrm{H}} R_{n+1}$ is zero for $i \neq j$, matrix $U^{-\mathrm{H}} R_n^{\mathrm{H}} R_{n+1} L D^{-1}$ is a lower triangular matrix. This means that $P_n^{\mathrm{H}} A P_n$ is a Hermitian and lower triangular matrix. Thus $P_n^{\mathrm{H}} A P_n$ is a diagonal matrix, which is equivalent to the second property. □

From Proposition 3.1, we have the following properties of the CG method:

Corollary 3.1 *Let $\boldsymbol{p}_n$ and $\boldsymbol{r}_n$ be the vectors in the CG method, then:*

1. $\boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$,
2. $A\boldsymbol{p}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$.

Proof From (3.8) and (3.9), we have $\boldsymbol{r}_1 = \boldsymbol{r}_0 - \alpha_0 A\boldsymbol{r}_0$. This means that $\mathrm{span}\{\boldsymbol{r}_0, \boldsymbol{r}_1\} = \mathcal{K}_2(A, \boldsymbol{r}_0)$. Similarly, $\mathrm{span}\{\boldsymbol{r}_0, \boldsymbol{r}_1, \dots, \boldsymbol{r}_{n-1}\} = \mathcal{K}_n(A, \boldsymbol{r}_0)$. From the first property of Proposition 3.1, it follows that $\boldsymbol{r}_n \perp \boldsymbol{r}_0, \boldsymbol{r}_1, \dots, \boldsymbol{r}_{n-1}$. Thus $\boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$. Next, from (3.8) and the first property of Proposition 3.1, it follows that $A\boldsymbol{p}_n = (\boldsymbol{r}_n - \boldsymbol{r}_{n+1})/\alpha_n \perp \boldsymbol{r}_0, \boldsymbol{r}_1, \dots, \boldsymbol{r}_{n-1}$, and thus $A\boldsymbol{p}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$. □

From the proof of Corollary 3.1, it is easy to see that

$$\boldsymbol{r}_n \in \mathcal{K}_{n+1}(A, \boldsymbol{r}_0), \quad \boldsymbol{p}_n \in \mathcal{K}_{n+1}(A, \boldsymbol{r}_0), \quad \boldsymbol{x}_n - \boldsymbol{x}_0 \in \mathcal{K}_n(A, \boldsymbol{r}_0). \tag{3.15}$$

If the coefficient matrix A is a Hermitian (or real symmetric) positive definite matrix, then the CG method produces the optimal approximate solution in terms of A-norm of the error as described below.

Theorem 3.1 *Let $A\boldsymbol{x} = \boldsymbol{b}$ be Hermitian positive definite linear systems. Then the CG method produces approximate solutions such that A-norm of the error is minimized:*

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)} \|\boldsymbol{e}_n\|_A,$$

where $\|\boldsymbol{e}_n\|_A = (\boldsymbol{e}_n^{\mathrm{H}} A \boldsymbol{e}_n)^{1/2}$ and $\boldsymbol{e}_n = \boldsymbol{x} - \boldsymbol{x}_n$ is the error of the nth approximate solution of the CG method.

Proof Since A is Hermitian positive definite, the Cholesky decomposition exists, i.e., $A = LL^{\mathrm{H}}$. (The Cholesky decomposition for a real symmetric positive definite matrix is described in Sect. 1.4.1.) Recall that the approximate solutions of the CG method satisfy

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + \boldsymbol{z}_n, \quad \boldsymbol{z}_n \in \mathcal{K}_n(A, \boldsymbol{r}_0),$$

where $\boldsymbol{z}_n$ is determined by $\boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$, see Corollary 3.1. Let V_n be a matrix whose column vectors correspond to orthonormalized basis vectors of $\mathcal{K}_n(A, \boldsymbol{r}_0)$. Then,

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n\boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathbb{C}^n, \tag{3.16}$$

where $\boldsymbol{y}_n$ is determined by $V_n^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0}$. From (3.16) and $\boldsymbol{e}_n = \boldsymbol{x} - \boldsymbol{x}_n$, it follows that

$$\boldsymbol{e}_n = \boldsymbol{e}_0 - V_n\boldsymbol{y}_n,$$

where $\boldsymbol{y}_n$ is determined by $V_n^{\mathrm{H}}LL^{\mathrm{H}}\boldsymbol{e}_n = \boldsymbol{0}$ since $V_n^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0}$ and $\boldsymbol{r}_n = \boldsymbol{b} - A\boldsymbol{x}_n = A\boldsymbol{x} - A\boldsymbol{x}_n = A\boldsymbol{e}_n = LL^{\mathrm{H}}\boldsymbol{e}_n$. Here we used the Cholesky decomposition $A = LL^{\mathrm{H}}$. Equivalently, we have

$$L^{\mathrm{H}}\boldsymbol{e}_n = L^{\mathrm{H}}\boldsymbol{e}_0 - L^{\mathrm{H}}V_n\boldsymbol{y}_n,$$

where $\boldsymbol{y}_n$ is determined by $(L^{\mathrm{H}}V_n)^{\mathrm{H}}L^{\mathrm{H}}\boldsymbol{e}_n = \boldsymbol{0}$. This means that the CG method finds approximate solutions such that

$$\min_{\boldsymbol{y}_n \in \mathbb{C}^n} \|L^{\mathrm{H}}\boldsymbol{e}_n\|, \tag{3.17}$$

because $(L^{\mathrm{H}}V_n)^{\mathrm{H}}L^{\mathrm{H}}\boldsymbol{e}_n = \boldsymbol{0}$ is equivalent to the normal equation $(L^{\mathrm{H}}V_n)^{\mathrm{H}}(L^{\mathrm{H}}V_n)\boldsymbol{y}_n = (L^{\mathrm{H}}V_n)^{\mathrm{H}}L^{\mathrm{H}}\boldsymbol{e}_0$. The minimization (3.17) is equivalent to

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)} \|L^{\mathrm{H}}\boldsymbol{e}_n\|.$$

Since $\|L^{\mathrm{H}}\boldsymbol{e}_n\|^2 = (L^{\mathrm{H}}\boldsymbol{e}_n)^{\mathrm{H}}(L^{\mathrm{H}}\boldsymbol{e}_n) = \boldsymbol{e}_n^{\mathrm{H}}LL^{\mathrm{H}}\boldsymbol{e}_n = \boldsymbol{e}_n^{\mathrm{H}}A\boldsymbol{e}_n$, we finally have

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)} \|\boldsymbol{e}_n\|_A,$$

which concludes the proof. □

Theorem 3.1 shows an optimality of the CG method. On the other hand, we cannot see how fast the CG method converges. To see the convergence rate later, the matrix polynomial representation of the CG method is described. The residual vector $\boldsymbol{r}_n$ and the auxiliary vector $\boldsymbol{p}_n$ of the CG method can be expressed by using two polynomials R_n and P_n,

$$\boldsymbol{r}_n = R_n(A)\boldsymbol{r}_0, \quad \boldsymbol{p}_n = P_n(A)\boldsymbol{r}_0, \tag{3.18}$$

where $R_n(\lambda)$ denotes Lanczos polynomials, which satisfy the following three-term recurrence relations:

$$R_0(\lambda) = 1, \tag{3.19}$$

$$R_1(\lambda) = (1 - \alpha_0\lambda)R_0(\lambda), \tag{3.20}$$

$$\begin{aligned} R_n(\lambda) = &\left(1 + \frac{\beta_{n-2}}{\alpha_{n-2}}\alpha_{n-1} - \alpha_{n-1}\lambda\right) R_{n-1}(\lambda) \\ &- \frac{\beta_{n-2}}{\alpha_{n-2}}\alpha_{n-1}R_{n-2}(\lambda), \quad n = 2, 3, \dots, \end{aligned} \tag{3.21}$$

where $R_n(0) = 1$. R_n and P_n satisfy the following two-term recurrences:

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

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

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

The rate of the convergence of the CG method is described in Theorem 3.2.

Theorem 3.2 *The A-norm of the error of the CG method satisfies*

$$\|\boldsymbol{e}_n\|_A \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n \|\boldsymbol{e}_0\|_A, \tag{3.25}$$

where κ is the condition number of A.

Proof Since the residual vector of the CG method can be described as $\boldsymbol{r}_n = R(A)\boldsymbol{r}_0$ and $\boldsymbol{r}_n = A\boldsymbol{e}_n$, we obtain $\boldsymbol{e}_n = R_n(A)\boldsymbol{e}_0$. From (3.25) it follows that

$$\|\boldsymbol{e}_n\|_A = \|R_n(A)\boldsymbol{e}_0\|_A = \min_{p_n \in \mathcal{P}_n} \|p_n(A)\boldsymbol{e}_0\|_A,$$

where $\mathcal{P}_n$ is the set of polynomials of degree n with $\mathcal{P}_0 = 1$.

Using the diagonalization of Hermitian positive definite matrix A, we have $A = PDP^{\mathrm{H}}$ with unitary matrix P and diagonal matrix D whose diagonal elements are eigenvalues of A. Let $A^{1/2} = PD^{1/2}P^{\mathrm{H}}$, where $D^{1/2}$ is a diagonal matrix whose diagonal elements are square roots of eigenvalues of A. Note that all the eigenvalues of A are positive and $A^{1/2}$ is Hermitian, i.e., $(A^{1/2})^{\mathrm{H}} = A^{1/2}$. We see that $A^{1/2}A^{1/2} = PD^{1/2}P^{\mathrm{H}}PD^{1/2}P^{\mathrm{H}} = PD^{1/2}D^{1/2}P^{\mathrm{H}} = PDP^{\mathrm{H}} = A$. Then we have

$$\begin{aligned}
\|\boldsymbol{e}_n\|_A &= \min_{p_n \in \mathcal{P}_n} \|p_n(A)\boldsymbol{e}_0\|_A, \\
&= \min_{p_n \in \mathcal{P}_n} \sqrt{(p_n(A)\boldsymbol{e}_0)^{\mathrm{H}} A (p_n(A)\boldsymbol{e}_0)} \\
&= \min_{p_n \in \mathcal{P}_n} \sqrt{(p_n(A)\boldsymbol{e}_0)^{\mathrm{H}} (A^{1/2})^{\mathrm{H}} A^{1/2} (p_n(A)\boldsymbol{e}_0)} \\
&= \min_{p_n \in \mathcal{P}_n} \sqrt{(A^{1/2} p_n(A)\boldsymbol{e}_0)^{\mathrm{H}} (A^{1/2} p_n(A)\boldsymbol{e}_0)} \\
&= \min_{p_n \in \mathcal{P}_n} \|A^{1/2} p_n(A)\boldsymbol{e}_0\|.
\end{aligned}$$

Using $A = PDP^{\mathrm{H}}$ and $A^{1/2} = PD^{1/2}P^{\mathrm{H}}$ yields

$$\begin{aligned}
\|\boldsymbol{e}_n\|_A &= \min_{p_n \in \mathcal{P}_n} \|A^{1/2} p_n(A)\boldsymbol{e}_0\| = \min_{p_n \in \mathcal{P}_n} \|PD^{1/2}P^{\mathrm{H}} p_n(PDP^{\mathrm{H}})\boldsymbol{e}_0\| \\
&= \min_{p_n \in \mathcal{P}_n} \|PD^{1/2}P^{\mathrm{H}} P p_n(D) P^{\mathrm{H}} \boldsymbol{e}_0\| = \min_{p_n \in \mathcal{P}_n} \|PD^{1/2} p_n(D) P^{\mathrm{H}} \boldsymbol{e}_0\| \\
&= \min_{p_n \in \mathcal{P}_n} \|P p_n(D) D^{1/2} P^{\mathrm{H}} \boldsymbol{e}_0\| = \min_{p_n \in \mathcal{P}_n} \|P p_n(D) P^{\mathrm{H}} P D^{1/2} P^{\mathrm{H}} \boldsymbol{e}_0\| \\
&= \min_{p_n \in \mathcal{P}_n} \|P p_n(D) P^{\mathrm{H}} A^{1/2} \boldsymbol{e}_0\| \\
&\le \min_{p_n \in \mathcal{P}_n} \|P p_n(D) P^{\mathrm{H}}\| \|A^{1/2} \boldsymbol{e}_0\| \\
&= \min_{p_n \in \mathcal{P}_n} \|p_n(D)\| \|\boldsymbol{e}_0\|_A \\
&= \min_{p_n \in \mathcal{P}_n} \max_{i=1,2,\ldots,N} |p_n(\lambda_i)| \times \|\boldsymbol{e}_0\|_A.
\end{aligned}$$

Thus, we have

$$\frac{\|\boldsymbol{e}_n\|_A}{\|\boldsymbol{e}_0\|_A} \le \min_{p_n \in \mathcal{P}_n} \max_{i=1,2,\ldots,N} |p_n(\lambda_i)|.$$

Let a be the smallest eigenvalue and b be the largest eigenvalue of A. Then all the eigenvalues λ_i belong to the closed interval $[a, b]$. Thus we have the following inequalities:

$$\frac{\|\boldsymbol{e}_n\|_A}{\|\boldsymbol{e}_0\|_A} \le \min_{p_n \in \mathcal{P}_n} \max_{i=1,2,\ldots,N} |p_n(\lambda_i)| \le \min_{p_n \in \mathcal{P}_n} \max_{\lambda \in [a,b]} |p_n(\lambda)|.$$

The solution of the min-max problem can be rewritten using Chebyshev polynomials as follows:

$$\min_{p_n \in \mathcal{P}_n} \max_{\lambda \in [a,b]} |p_n(\lambda)| = \max_{\lambda \in [a,b]} \frac{|T_n(\frac{2\lambda - b - a}{b-a})|}{T_n(\frac{-b-a}{b-a})},$$

where

$$T_n(x) = \frac{1}{2}\left[\left(x + \sqrt{x^2-1}\right)^n + \left(x - \sqrt{x^2-1}\right)^n\right] \quad \text{for } |x| \geq 1. \qquad (3.26)$$

From the property of Chebyshev polynomials, $|T_n(\frac{2\lambda-b-a}{b-a})| \leq 1$, from which we have

$$\min_{p_n\in\mathcal{P}_n}\max_{\lambda\in[a,b]}|p_n(\lambda)| \leq \frac{1}{|T_n(\frac{-b-a}{b-a})|} = \frac{1}{|T_n(\frac{-\kappa-1}{\kappa-1})|},$$

where $\kappa = b/a$ is the condition number of A. Using (3.26) yields

$$\min_{p_n\in\mathcal{P}_n}\max_{\lambda\in[a,b]}|p_n(\lambda)| \leq \frac{1}{|T_n(\frac{-\kappa-1}{\kappa-1})|} = \frac{2}{\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^n} \leq \frac{2}{\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^n}$$
$$= 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n,$$

which concludes the proof. □

Theorem 3.2 indicates that the speed of convergence of the CG method depends on the condition number of the coefficient matrix A, i.e., the smaller the condition number is, the faster the CG method converges. It is therefore natural to consider the following equivalent linear systems:

$$A\boldsymbol{x} = \boldsymbol{b} \Leftrightarrow \tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}},$$

so that the condition number of $\tilde{A}$ is much smaller than that of A, and then apply the CG method not to $A\boldsymbol{x} = \boldsymbol{b}$ but to the transformed linear systems $\tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$. The resulting algorithm is called the preconditioned (PCG) method. The technique to construct $\tilde{A}$ is called *preconditioning*, which will be discussed in Sect. 3.5.

We now describe the algorithm of the preconditioned CG method. Let $A \approx K = LL^{\mathrm{H}}$. Then $\tilde{A} = L^{-1}AL^{-\mathrm{H}}$ is expected to be close to the identity matrix whose condition number is one. We now consider applying the CG method to the following transformed linear system

$$\tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}},$$

where $\tilde{A} = L^{-1}AL^{-\mathrm{H}}$, $\tilde{\boldsymbol{x}} = L^{\mathrm{H}}\boldsymbol{x}$, and $\tilde{\boldsymbol{b}} = L^{-1}\boldsymbol{b}$. Then, the CG method for $\tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$ is expected to converge much faster than the CG method for $A\boldsymbol{x} = \boldsymbol{b}$. From Algorithm 3.1, the iterates of the CG method for $\tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$ are written as

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

Using the following rewrites:

$$\tilde{\boldsymbol{x}} \Rightarrow L^{\mathrm{H}}\boldsymbol{x}, \quad \tilde{\boldsymbol{p}} \Rightarrow L^{\mathrm{H}}\boldsymbol{p}, \quad \tilde{\boldsymbol{r}} \Rightarrow L^{-1}\boldsymbol{r},$$

we have

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

Since $K^{-1} = L^{-\mathrm{H}}L^{-1}$, we have the PCG method that is described in Algorithm 3.2.

Algorithm 3.2 The preconditioned CG method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \ldots,$ until convergence do
2:     $\boldsymbol{p}_n = K^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $\alpha_n = \frac{(K^{-1}\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{p}_n, A\boldsymbol{p}_n)}$
4:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$
5:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
6:     $\beta_n = \frac{(K^{-1}\boldsymbol{r}_{n+1}, \boldsymbol{r}_{n+1})}{(K^{-1}\boldsymbol{r}_n, \boldsymbol{r}_n)}$
7: end for
```

Some notes on the CG method are listed next.

1. From Theorem 3.1, in exact precision arithmetic, the CG method finds the exact solution within N iteration steps.
2. We can still use the CG method even if the coefficient matrix is not positive definite. In this case, the optimality in Theorem 3.1 does not hold anymore,

and the CG method may suffer from breakdown, i.e., zero division, because $(\boldsymbol{p}_n, A\boldsymbol{p}_n) > 0$ for any $\boldsymbol{p}_n \neq \boldsymbol{0}$ does not hold anymore, see line 3 of Algorithm 3.1. In practice, breakdown (zero division) does not occur, but near breakdown may occur, i.e., $(\boldsymbol{p}_n, A\boldsymbol{p}_n) \approx 0$, leading to numerical instability.
3. In exact precision arithmetic, the CG method for Hermitian indefinite matrices also finds the exact solution within N iteration steps if breakdown does not occur.

In the next two subsections, the CR method and the MINRES method for Hermitian linear systems are described.

3.1.2 The Conjugate Residual (CR) Method

The Conjugate Residual (CR) method [177] was proposed by in 1955 Stiefel, who is one of the authors of the CG method. The feature of the CR method is that it has optimality in terms of the residual norm, which will be described later.

It is possible to derive the CR method like the derivation of the CG method, but here we give a concise derivation of the CR method using the algorithm of the CG method. A drawback of the derivation here is that A is assumed to be Hermitian positive definite, whereas the CR method can be used for Hermitian indefinite linear systems. On the other hand, we will see the derivation is much simpler than that of the CG method. A derivation of the CR method similar to that of the CG method can be found in Sect. 3.2.2 by restricting the derivation of the COCR method for complex symmetric linear systems to that for real symmetric ones.

We now derive the CR method from the CG method. If A is Hermitian positive definite, then there exists a square root of A: the Cholesky factorization of A, see Sect. 1.4.1, is obtained as LL^{H}, and L can be decomposed as $U\Sigma V^{\mathrm{H}}$ by using singular value decomposition.[1] Then, we have

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

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

$$A\tilde{\boldsymbol{x}} = A^{\frac{1}{2}}\boldsymbol{b}, \quad \tilde{\boldsymbol{x}} = A^{\frac{1}{2}}\boldsymbol{x}. \tag{3.27}$$

It follows from the algorithm of the CG method that

[1] U and V are unitary matrices, and Σ is a diagonal matrix whose diagonal elements are nonnegative.

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

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 (3.27) 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 the CR method (Algorithm 3.3).

Algorithm 3.3 The CR method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N, \beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \dots,$ until convergence do
2:     $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1}$
4:     $\alpha_n = \frac{(\boldsymbol{r}_n, A\boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}$
5:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$
6:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
7:     $\beta_n = \frac{(\boldsymbol{r}_{n+1}, A\boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n, A\boldsymbol{r}_n)}$
8: end for
```

From line 7 of Algorithm 3.3, the CR method may suffer from breakdown if Hermitian matrix A is indefinite, i.e., $(\boldsymbol{r}_n, A\boldsymbol{r}_n) = 0$.

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

$$(\boldsymbol{r}_i, A\boldsymbol{r}_j) = 0 \quad \text{for } i \neq j, \tag{3.28}$$

$$(A\boldsymbol{p}_i, A\boldsymbol{p}_j) = 0 \quad \text{for } i \neq j. \tag{3.29}$$

Comparing the two properties ((3.28) and (3.29)) and those of the CG method, the CR method finds approximate solutions such that

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + \boldsymbol{z}_n, \quad \boldsymbol{z}_n \in \mathcal{K}_n(A, \boldsymbol{r}_0),$$

where $\boldsymbol{z}_n$ is determined by $\boldsymbol{r}_n \perp A\mathcal{K}_n(A, \boldsymbol{r}_0)(:= \mathcal{K}_n(A, A\boldsymbol{r}_0))$, or equivalently

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

where $\boldsymbol{y}_n$ is determined by $(AV_n)^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0}$. Following the proof of Theorem 3.1, we have

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A,\boldsymbol{r}_0)} \|\boldsymbol{e}_n\|_{A^2}.$$

Since $\|\boldsymbol{e}_n\|_{A^2}^2 = \boldsymbol{e}_n^{\mathrm{H}} A^2 \boldsymbol{e}_n = (A\boldsymbol{e}_n)^{\mathrm{H}} A\boldsymbol{e}_n = \boldsymbol{r}_n^{\mathrm{H}}\boldsymbol{r}_n = \|\boldsymbol{r}_n\|^2$, the CR method finds approximate solutions such that

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A,\boldsymbol{r}_0)} \|\boldsymbol{r}_n\|.$$

Therefore, the CR method produces the optimal approximate solution in terms of the residual norm.

Similar to the derivation of the preconditioned CG method, the preconditioned CR method can be derived from applying the CR method to the transformed linear systems $\tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$. The resulting algorithm is described in Algorithm 3.4.

Algorithm 3.4 The preconditioned CR method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \ldots,$ until convergence do
2:     $\boldsymbol{p}_n = K^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $(A\boldsymbol{p}_n = AK^{-1}\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1})$
4:     $\alpha_n = \frac{(K^{-1}\boldsymbol{r}_n, AK^{-1}\boldsymbol{r}_n)}{(A\boldsymbol{p}_n, K^{-1}A\boldsymbol{p}_n)}$
5:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$
6:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
7:     $(K^{-1}\boldsymbol{r}_{n+1} = K^{-1}\boldsymbol{r}_n - \alpha_n K^{-1}A\boldsymbol{p}_n)$
8:     $\beta_n = \frac{(K^{-1}\boldsymbol{r}_{n+1}, AK^{-1}\boldsymbol{r}_{n+1})}{(K^{-1}\boldsymbol{r}_n, AK^{-1}\boldsymbol{r}_n)}$
9: end for
```

Note that lines 3 and 7 in Algorithm 3.4 are added for reducing the number of matrix–vector multiplications of the form $A\boldsymbol{v}$ and solving linear systems of the form $K\boldsymbol{v} = \boldsymbol{w}$. In fact, at each iteration, it seems that we need to compute

$$A\boldsymbol{p}_n, \quad K^{-1}\boldsymbol{r}_{n+1}, \quad A(K^{-1}\boldsymbol{r}_n), \quad K^{-1}(A\boldsymbol{p}_n).$$

But from lines 3 and 7, we do not need to compute $A\boldsymbol{p}_n$ and $K^{-1}\boldsymbol{r}_{n+1}$, except $K^{-1}\boldsymbol{r}_0$ at the first iteration step. Thus, essentially we only need to compute the following form:

$$A\boldsymbol{v}, \quad K^{-1}\boldsymbol{w}$$

at each iteration step. Here, $\boldsymbol{v} := K^{-1}\boldsymbol{r}_{n+1}$ is obtained by line 7 and $\boldsymbol{w} := A\boldsymbol{p}_n$ is obtained by line 3.

3.1.3 The Minimal Residual (MINRES) Method

The Minimal Residual (MINRES) method [143] was proposed by Paige and Saunders in 1975. The MINRES method generates $\boldsymbol{x}_n$ that minimizes $\|\boldsymbol{b} - A\boldsymbol{x}_n\|$ over the affine space $\boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)$. Therefore, in exact precision arithmetic, the MINRES method and the CR method produce the same approximate solutions. The minimization can be achieved by using the Lanczos process.

Here, we give a derivation process of the MINRES method. Let V_n be the orthonormal basis of $\mathcal{K}_n(A, \boldsymbol{r}_0)$. The MINRES method finds $\boldsymbol{x}_n$ over the affine space $\boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)$, i.e.,

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n \boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathbb{C}^n. \tag{3.30}$$

The corresponding residual vector is given by

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

From the matrix form of the Lanczos process in (1.42), 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 $\beta := \|\boldsymbol{r}_0\|$. The 2-norm of the residual vector is written as

$$\begin{aligned}
\|\boldsymbol{r}_n\| &= \|V_{n+1}(\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n)\| \\
&= \sqrt{[V_{n+1}(\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n)]^{\mathrm{H}} V_{n+1}(\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n)} \\
&= \sqrt{(\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n)^{\mathrm{H}} V_{n+1}^{\mathrm{H}} V_{n+1}(\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n)} \\
&= \sqrt{(\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n)^{\mathrm{H}} (\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n)} \\
&= \|\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}_n\|.
\end{aligned}$$

The MINRES method finds approximate solutions such that the 2-norm of the residual vector is minimized, i.e., $\boldsymbol{y}_n$ is chosen such that

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y} \in \mathbb{C}^n} \|\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}\|. \tag{3.31}$$

By solving the least-squares problem in (3.31) using a *Givens rotation* that is one of the unitary matrices, the MINRES method is obtained.

In the following, we describe how Givens rotations work for solving (3.31). For simplicity, we consider the case $n = 4$ for solving it. $\beta \boldsymbol{e}_1 - T_{5,4} \boldsymbol{y}$ is written as

$$\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y} = \begin{bmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} t_{11} & t_{12} & & \\ t_{21} & t_{22} & t_{23} & \\ & t_{32} & t_{33} & t_{34} \\ & & t_{43} & t_{44} \\ & & & t_{54} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix},$$

where $t_{ij} = t_{ji},\ 1 \le i,j \le 4$. Let G_1 be a matrix of the Givens rotation, i.e.,

$$G_1 = \begin{bmatrix} c_1 & s_1 & & & \\ -\bar{s}_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \quad \text{with} \quad c_1 = \frac{|t_{11}|}{\sqrt{|t_{11}|^2 + |t_{21}|^2}}, \quad \bar{s}_1 = \frac{t_{21}}{t_{11}} c_1,$$

and s_1 is obtained by the conjugate of $\bar{s}_1$. There is one exception: set $s_1 = \bar{s}_1 = 1$ if $t_{11} = 0$. 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

$$G_1(\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}) = \begin{bmatrix} g_1^{(1)} \\ g_2^{(1)} \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} t_{11}^{(1)} & t_{12}^{(1)} & t_{13}^{(1)} & \\ & t_{22}^{(1)} & t_{23}^{(1)} & \\ & t_{32} & t_{33} & t_{34} \\ & & t_{43} & t_{44} \\ & & & t_{54} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix},$$

where $g_2^{(1)} = -\bar{s}_1 \beta$. Next, let G_2 be a unitary matrix defined by

$$G_2 = \begin{bmatrix} 1 & & & & \\ & c_2 & s_2 & & \\ & -\bar{s}_2 & c_2 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \quad \text{with} \quad c_2 = \frac{|t_{22}^{(1)}|}{\sqrt{|t_{22}^{(1)}|^2 + |t_{32}|^2}}, \quad \bar{s}_2 = \frac{t_{32}}{t_{22}^{(1)}} c_2.$$

There is one exception: set $s_2 = \bar{s}_2 = 1$ if $t_{22}^{(1)} = 0$. 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

$$G_2 G_1(\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}) = \begin{bmatrix} g_1^{(1)} \\ g_2^{(2)} \\ g_3^{(2)} \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} t_{11}^{(1)} & t_{12}^{(1)} & t_{13}^{(1)} & \\ & t_{22}^{(2)} & t_{23}^{(2)} & t_{24}^{(2)} \\ & & t_{33}^{(2)} & t_{34}^{(2)} \\ & & t_{43} & t_{44} \\ & & & t_{54} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix},$$

where $g_3^{(2)} = (-1)^2 \bar{s}_2 \bar{s}_1 \beta$. Similarly, using G_3 and G_4, we finally obtain

$$
\begin{aligned}
Q_4(\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}) &= \begin{bmatrix} g_1^{(1)} \\ g_2^{(2)} \\ g_3^{(3)} \\ g_4^{(4)} \\ g_5^{(5)} \end{bmatrix} - \begin{bmatrix} t_{11}^{(1)} & t_{12}^{(1)} & t_{13}^{(1)} & \\ & t_{22}^{(2)} & t_{23}^{(2)} & t_{24}^{(2)} \\ & & t_{33}^{(3)} & t_{34}^{(3)} \\ & & & t_{44}^{(4)} \\ & & & \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{g}_4 \\ g_5^{(5)} \end{bmatrix} - \begin{bmatrix} R_4 \\ \boldsymbol{0}^\top \end{bmatrix} \boldsymbol{y}_4,
\end{aligned}
$$

where $Q_4 = G_4 G_3 G_2 G_1$ and $g_5^{(5)} = (-1)^4 \bar{s}_4 \bar{s}_3 \bar{s}_2 \bar{s}_1 \beta$. Hence, we obtain $\boldsymbol{y}_4$ by solving the following equation:

$$
R_4 \boldsymbol{y}_4 = \boldsymbol{g}_4 \tag{3.32}
$$

using backward substitution. It is interesting to see that we can obtain the value of $\min_{\boldsymbol{y} \in \mathbb{C}^4} \|\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}\|$ without solving the equation $R_4 \boldsymbol{y}_4 = \boldsymbol{g}_4$, since the value satisfies

$$
\begin{aligned}
\|\boldsymbol{r}_4\| &= \min_{\boldsymbol{y} \in \mathbb{C}^4} \|\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y}\| = \min_{\boldsymbol{y} \in \mathbb{C}^4} \|Q_4(\beta \boldsymbol{e}_1 - T_{5,4}\boldsymbol{y})\| \\
&= \left\| \begin{bmatrix} \boldsymbol{g}_4 \\ g_5^{(5)} \end{bmatrix} - \begin{bmatrix} R_4 \\ \boldsymbol{0}^\top \end{bmatrix} \boldsymbol{y}_4 \right\| = \left\| \begin{bmatrix} \boldsymbol{0} \\ g_5^{(5)} \end{bmatrix} \right\| \\
&= |g_5^{(5)}|.
\end{aligned}
$$

Thus, $|g_5^{(5)}|$ corresponds to the residual norm of $\boldsymbol{r}_4$, and this can be used as a stopping criterion. To obtain the approximate solution, it follows from (3.30) and (3.32) that

$$
\boldsymbol{x}_4 = \boldsymbol{x}_0 + V_4 R_4^{-1} \boldsymbol{g}_4. \tag{3.33}
$$

Here, we introduce a matrix $P_4 := V_4 R_4^{-1}$ with columns $[\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4]$. Then, from $[\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4] R_4 = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \boldsymbol{v}_4]$, we have the following recurrences:

$$
\boldsymbol{p}_1 = \boldsymbol{v}_1 / t_{11}^{(1)}, \tag{3.34}
$$

$$
\boldsymbol{p}_2 = (\boldsymbol{v}_2 - t_{12}^{(1)} \boldsymbol{p}_1) / t_{22}^{(2)}, \tag{3.35}
$$

$$
\boldsymbol{p}_3 = (\boldsymbol{v}_3 - t_{13}^{(1)} \boldsymbol{p}_1 - t_{23}^{(2)} \boldsymbol{p}_2) / t_{33}^{(3)}, \tag{3.36}
$$

$$
\boldsymbol{p}_4 = (\boldsymbol{v}_4 - t_{24}^{(2)} \boldsymbol{p}_2 - t_{34}^{(3)} \boldsymbol{p}_3) / t_{44}^{(4)}. \tag{3.37}
$$

From (3.33), $\boldsymbol{x}_4 = \boldsymbol{x}_0 + P_4 \boldsymbol{g}_4 = \boldsymbol{x}_0 + \sum_{i=1}^4 g_i \boldsymbol{p}_i$ and thus

$$
\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + g_i \boldsymbol{p}_i, \quad i = 1, \ldots, 4, \tag{3.38}
$$

where g_i is the ith element of $\boldsymbol{g}_4$. The approximate solutions $\boldsymbol{x}_1, \dots, \boldsymbol{x}_4$ of the MINRES method are obtained by (3.38) with the recurrences (3.34)–(3.37).

The derivation for the case $n = 4$ can be easily generalized to the nth iteration step, leading to Algorithm 3.5. From Algorithm 3.5, the MINRES method never suffers from breakdown since $\beta_n \neq 0$ unless $\tilde{\boldsymbol{v}}_{n+1} = \boldsymbol{0}$.

Algorithm 3.5 The MINRES method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_0 = 0$, $\boldsymbol{v}_0 = \boldsymbol{0}$, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \dots, 0)^\top$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$
2: for $n = 1, 2, \dots$ do
3:     (Lanczos process)
4:     $\alpha_n = (\boldsymbol{v}_n, A\boldsymbol{v}_n)$
5:     $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n \boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$
6:     $\beta_n = (\tilde{\boldsymbol{v}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$
7:     $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\beta_n$
8:     $t_{n-1,n} = \beta_{n-1}$,  $t_{n,n} = \alpha_n$
9:     $t_{n+1,n} = \beta_n$
10:    (Givens rotations)
11:    for $i = \max\{1, n-2\}, \dots, n-1$ do
12:        $\begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{bmatrix} \begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix}$
13:    end for
14:    $c_n = \frac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}}$
15:    $\bar{s}_n = \frac{t_{n+1,n}}{t_{n,n}} c_n$
16:    $t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n}$
17:    $t_{n+1,n} = 0$
18:    $\begin{bmatrix} g_n \\ g_{n+1} \end{bmatrix} = \begin{bmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{bmatrix} \begin{bmatrix} g_n \\ 0 \end{bmatrix}$
19:    (Update $\boldsymbol{x}_n$)
20:    $\boldsymbol{p}_n = (\boldsymbol{v}_n - t_{n-2,n}\boldsymbol{p}_{n-2}$
21:            $- t_{n-1,n}\boldsymbol{p}_{n-1})/t_{n,n}$
22:    $\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + g_n \boldsymbol{p}_n$
23:    (Check convergence)
24:    if $|g_{n+1}|/\|\boldsymbol{b}\| \le \epsilon$, then stop
25: end for
```

As mentioned above, the MINRES method never suffers from breakdown. Paige and Saunders also proposed the SYMMLQ method in [143], which is a variant of the CG method, to avoid breakdown for Hermitian indefinite linear systems. A brief explanation is given next.

Recall that the approximate solution x_n of the CG method is given by

$$\begin{aligned} \boldsymbol{x}_n &= \boldsymbol{x}_0 + V_n \boldsymbol{y}_n, \\ \boldsymbol{r}_n &= \boldsymbol{r}_0 - V_{n+1} T_{n+1,n} \boldsymbol{y}_n = V_{n+1}(\beta \boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}_n), \end{aligned}$$

where $\boldsymbol{y}_n$ is determined by $\boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$ (see Corollary 3.1), i.e., $V^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0}$. This yields

$$T_n \boldsymbol{y}_n = \beta \boldsymbol{e}_1, \tag{3.39}$$

where T_n is the tridiagonal matrix obtained by removing the last row of $T_{n+1,n}$. The CG method is obtained by the LDL^{H} decomposition of T_n, i.e.,

$$T_n = LDL^{\mathrm{H}},$$

where L is a unit lower bidiagonal matrix and D is a diagonal matrix. If A is Hermitian positive definite, T_n is also Hermitian positive definite. Thus this decomposition never suffers from breakdown. On the other hand, when A is indefinite, LDL^{H} of T_n may not be obtained due to breakdown.

In order to circumvent the problem, Paige and Saunders proposed to use the *LQ factorization* of T_n, i.e.,

$$T_n = L_n Q_n, \quad Q_n^{\mathrm{H}} Q_n = I,$$

where L_n is a lower triangular matrix and the decomposition is obtained by Givens rotations. Thus the decomposition is free from breakdown. The resulting algorithm is known as the SYMMLQ method. When the coefficient matrix A is Hermitian positive definite, the SYMMLQ method gives the same result as the CG method in exact precision arithmetic, and for indefinite cases, the SYMMLQ method succeeded in avoiding breakdown when solving (3.39).

For a brief historical note including the relation between the SYMMLQ method and (less-known) Fridman's method of 1962, see Sect. 2.4 in [160].

Another important remedy is "composite step" technique by Bank and Chan [16] and similar techniques (hyperbolic pairs) in [58, 126]. The key idea of the composite step is decomposing T_n in (3.39) into

$$T_n = \tilde{L}\tilde{D}\tilde{L}^{\mathrm{H}},$$

where $\tilde{L}$ is a unit lower block bidiagonal matrix and $\tilde{D}$ is a block diagonal matrix whose block is of size 1×1 or 2×2, i.e.,

$$\tilde{D} = \begin{bmatrix} D_1 & & \\ & \ddots & \\ & & D_m \end{bmatrix},$$

where D_i is a scalar or a 2×2 matrix. This decomposition is also free from breakdown, see, e.g., [16, Theorem 2.2]. An algorithmic explanation of the composite step technique (the composite step BiCG method) is described in Sect. 3.3.2.

3.2 Complex Symmetric Linear Systems

In this section, we consider complex symmetric linear systems of the form

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

where A is *complex symmetric*, i.e., $A = A^{\top}$ and $A \neq A^{\mathrm{H}}$. Note that complex symmetric matrix A is not Hermitian. Below is an example of a complex symmetric matrix:

$$A = \begin{bmatrix} 1+i & 2+i \\ 2+i & 3 \end{bmatrix}.$$

For applications of complex symmetric linear systems, see the Helmholtz equation in Sect. 2.1.1.3 and a large-scale electronic structure calculation in Sect. 2.2.1.

This section describes the COCG method, the COCR method, and the QMR_SYM method for solving complex symmetric linear systems. These methods reduce to the CG method, the CR method, and the MINRES method if these methods are applied to real symmetric linear systems. Other Krylov subspace methods and applications for complex symmetric linear systems, see, e.g., [1, 30, 37, 40, 83, 84, 124, 141] and the references therein.

3.2.1 *The Conjugate Orthogonal Conjugate Gradient (COCG) Method*

The COCG method [197] is the best-known method for solving complex symmetric linear systems. The algorithm can be formally derived by the CG method with the change of the dot product $(\boldsymbol{a}, \boldsymbol{b})$ into $(\bar{\boldsymbol{a}}, \boldsymbol{b})$. Here $\bar{\boldsymbol{a}}$ denotes the complex conjugate of $\boldsymbol{a}$. With this rewrite, the algorithm of the COCG method is described in Algorithm 3.6.

Algorithm 3.6 The COCG method (note: $(\bar{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \ldots,$ until convergence do
2:     $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $\alpha_n = \frac{(\bar{\boldsymbol{r}}_n, \boldsymbol{r}_n)}{(\bar{\boldsymbol{p}}_n, A\boldsymbol{p}_n)}$
4:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
5:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
6:     $\beta_n = \frac{(\bar{\boldsymbol{r}}_{n+1}, \boldsymbol{r}_{n+1})}{(\bar{\boldsymbol{r}}_n, \boldsymbol{r}_n)}$
7: end for
```

The residual vector $\boldsymbol{r}_n$ of the COCG method satisfies the following relation:

$$\boldsymbol{r}_n \perp \overline{\mathcal{K}_n(A, \boldsymbol{r}_0)},$$

where $\overline{\mathcal{K}_n(A, \boldsymbol{r}_0)} := \mathcal{K}_n(\bar{A}, \bar{\boldsymbol{r}}_0)$. From Algorithm 3.6, it is easy to see that if matrix A is a real symmetric matrix, then the COCG method is the same as the CG method. The preconditioned COCG method is described in Algorithm 3.7.

Algorithm 3.7 The preconditioned COCG method (note: $(\overline{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{p}_{-1} = \boldsymbol{0}$, $\beta_{-1} = 0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \dots,$ until convergence do
2:     $\boldsymbol{p}_n = K^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $\alpha_n = \frac{(\overline{K}^{-1}\overline{\boldsymbol{r}}_n, \boldsymbol{r}_n)}{(\overline{\boldsymbol{p}}_n, A\boldsymbol{p}_n)}$
4:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
5:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
6:     $\beta_n = \frac{(\overline{K}^{-1}\overline{\boldsymbol{r}}_{n+1}, \boldsymbol{r}_{n+1})}{(\overline{K}^{-1}\overline{\boldsymbol{r}}_n, \boldsymbol{r}_n)}$
7: end for
```

3.2.2 *The Conjugate Orthogonal Conjugate Residual (COCR) Method*

The COCR method [171] is also a Krylov subspace method for solving complex symmetric linear systems. The algorithm can be formally derived by the CR method with the change of the dot product $(\boldsymbol{a}, \boldsymbol{b})$ into $(\overline{\boldsymbol{a}}, \boldsymbol{b})$. The COCR method produces approximate solutions such that the residual vector satisfies

$$\boldsymbol{r}_n \perp \overline{A\mathcal{K}_n(A, \boldsymbol{r}_0)},$$

where $\overline{A\mathcal{K}_n(A, \boldsymbol{r}_0)} := \overline{A}\mathcal{K}_n(\overline{A}, \overline{\boldsymbol{r}}_0)$. From this, if matrix A is real symmetric, the COCR method is identical to the CR method. This indicates that if A is complex symmetric and is close to the real matrix, then it can be expected that the residual 2-norm tends to decrease almost monotonically or tends to show smooth convergence behavior.

In what follows, the COCR method is derived. To begin with, the conjugate A-orthogonalization process is introduced in Algorithm 3.8.

Algorithm 3.8 The conjugate A-orthogonalization process without normalization (note: $(\overline{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: $\boldsymbol{v}_0 = \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Input: $t_{0,0} = (\overline{A}\overline{\boldsymbol{v}}_0, A\boldsymbol{v}_0)/(\overline{\boldsymbol{v}}_0, A\boldsymbol{v}_0)$
Input: $\boldsymbol{v}_1 = -\alpha_0(A\boldsymbol{v}_0 - t_{0,0}\boldsymbol{v}_0)$
1: $t_{0,0} = (\overline{A}\overline{\boldsymbol{v}}_1 A\boldsymbol{v}_1)/(\overline{\boldsymbol{v}}_1, A\boldsymbol{v}_1)$
2: $\boldsymbol{v}_1 = -\alpha_0(A\boldsymbol{v}_1 - t_{0,0}\boldsymbol{v}_1)$
3: for $n = 1, 2, \dots, N-1$ do
4:     $t_{n-1,n} = (\overline{A}\overline{\boldsymbol{v}}_{n-1}, A\boldsymbol{v}_n)/(\overline{\boldsymbol{v}}_{n-1}, A\boldsymbol{v}_{n-1})$
5:     $t_{n,n} = (\overline{A}\overline{\boldsymbol{v}}_n, A\boldsymbol{v}_n)/(\overline{\boldsymbol{v}}_n, A\boldsymbol{v}_n)$
6:     $\boldsymbol{v}_{n+1} = -\alpha_n(A\boldsymbol{v}_n - t_{n,n}\boldsymbol{v}_n - t_{n-1,n}\boldsymbol{v}_{n-1})$
7: end for
```

Similar to the Lanczos process (Algorithm 1.11), the vectors $\boldsymbol{v}_i$'s of Algorithm 3.8 satisfy the following relations:

$$\begin{aligned} &\mathrm{span}\{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n\} = \mathcal{K}_n(A, \boldsymbol{r}_0), \\ &(\overline{\boldsymbol{v}_i}, A\boldsymbol{v}_j) = 0 \quad \text{for } i \neq j. \end{aligned} \tag{3.40}$$

In what follows, the COCR method is derived from Algorithm 3.8. Let $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$ be the initial residual vector for a given initial guess $\boldsymbol{x}_0$. Then, Algorithm 3.8 can be expressed in the following matrix form:

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

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

$$AR_n = R_{n+1}T_{n+1,n}. \tag{3.41}$$

Since the matrix R_n is generated by Algorithm 3.8, it follows from A-orthogonal property (3.40) that we have $R_n^\top AR_n = D_n$, where D_n is an n-by-n diagonal matrix. The scalar parameters $\alpha_0, \dots, \alpha_{n-1}$ still remain unknown. We show that if we determine the scalar parameters such that approximate solutions $\boldsymbol{x}_0, \dots, \boldsymbol{x}_{n-1}$ can be extracted from the information of $\boldsymbol{r}_0, \dots, \boldsymbol{r}_{n-1}$, then we obtain the algorithm of the COCR method.

Let $X_n := [\boldsymbol{x}_0, \dots, \boldsymbol{x}_{n-1}]$ and $\mathbf{1}$ be $[1, \dots, 1]^\top$. Then $R_n := [\boldsymbol{r}_0, \boldsymbol{r}_1, \dots, \boldsymbol{r}_{n-1}]$ and X_n are related as follows:

$$R_n = [\boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{b} - A\boldsymbol{x}_1, \dots, \boldsymbol{b} - A\boldsymbol{x}_{n-1}] = \boldsymbol{b}\mathbf{1}_n^\top - AX_n. \tag{3.42}$$

Substituting (3.42) into (3.41) yields

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

Then,

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

where $\boldsymbol{x}$ is the exact solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$. If the condition of the form

$$\boldsymbol{x}\mathbf{1}_{n+1}^\top T_{n+1,n} = O_n \tag{3.43}$$

holds, then $R_n = -X_{n+1}T_{n+1,n}$. The relation $R_n = -X_{n+1}T_{n+1,n}$ implies that the approximate solution vectors can be obtained by residual vectors. Conversely, if the relationship does not hold, the approximate solution vectors cannot be obtained by residual vectors.

Since unknown parameters $\alpha_0, \dots, \alpha_{n-1}$ in $T_{n+1,n}$ have not yet been determined, we determine the parameters so that (3.43) holds. 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

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

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

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

From (3.41) and the above factorization, we obtain

$$AR_n = R_{n+1}T_{n+1,n} = R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}B_n^{(U)}. \tag{3.44}$$

Here, $P_n := R_n(B_n^{(U)})^{-1}$. From (3.44), it follows that

$$AP_n = AR_n(B_n^{(U)})^{-1} = R_{n+1}B_{n+1,n}^{(L)}\Omega_n^{-1}. \tag{3.45}$$

The above matrix forms are equivalent to the following recurrences:

$$\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_{k-1}A\boldsymbol{p}_{k-1}, \quad 1 \le k \le n. \tag{3.46}$$

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

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_{k-1}\boldsymbol{p}_{k-1}, \quad 1 \le k \le n, \tag{3.47}$$

where $\beta_{k-1} := -\alpha_{k-1} t_{k-1,k}$. We now give the computational formulas of the approximate solutions. From (3.45),

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

Thus, we obtain

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

and it is equivalent to

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_{k-1}\boldsymbol{p}_{k-1}, \quad 1 \le k \le n. \tag{3.48}$$

Now, we give more practical computational formulas of α_k and β_k. From (3.41), it follows that

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

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

$$\beta_k = -\alpha_k t_{k,k+1} = \frac{(\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_{k+1})}{(\overline{A}\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_k)} \times \frac{(\overline{A}\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k+1})}{(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)} = \frac{(\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_{k+1})}{(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)}. \tag{3.49}$$

Here, we used the relation $(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k+1}) = (\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_k)$ because

$$(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k+1}) = (\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_{k+1})^\top = (\boldsymbol{r}_k^\top A\boldsymbol{r}_{k+1})^\top = (\bar{\boldsymbol{r}}_{k+1}, A^\top \boldsymbol{r}_k) = (\bar{\boldsymbol{r}}_{k+1}, A\boldsymbol{r}_k). \tag{3.50}$$

From (3.45) and recalling $P_n = R_n (B_n^{(U)})^{-1}$, it follows that

$$(AP_n)^\top AP_n = (AP_n)^\top R_{n+1} B^{(L)}_{n+1,n} \Omega_n^{-1} = (B_n^{(U)})^{-\top} R_n^\top AR_{n+1} B^{(L)}_{n+1,n} \Omega_n^{-1}.$$

$(B_n^{(U)})^{-\top}$ is a lower triangular matrix, and A-orthogonal property (3.40) indicates that the off-diagonal elements of $R_n^\top AR_{n+1}$ are zeros. From this, $(AP_n)^\top AP_n$ is a lower triangular matrix of the form

$$(AP_n)^\top AP_n = \begin{bmatrix} d_0 & & & \\ * & \ddots & & \\ \vdots & \ddots & \ddots & \\ * & \cdots & * & d_{n-1} \end{bmatrix},$$

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

$$\alpha_k = \frac{(\bar{\boldsymbol{r}}_k, A\boldsymbol{r}_k)}{(\overline{A\boldsymbol{p}}_k, A\boldsymbol{p}_k)}. \tag{3.51}$$

Moreover, since $(AP_n)^\top AP_n$ is complex symmetric, we have

$$(AP_n)^\top AP_n = \begin{bmatrix} d_0 & & & \\ * & \ddots & & \\ \vdots & \ddots & \ddots & \\ * & \cdots & * & d_{n-1} \end{bmatrix} = \begin{bmatrix} d_0 & * & \cdots & * \\ & \ddots & \ddots & \vdots \\ & & \ddots & * \\ & & & d_{n-1} \end{bmatrix},$$

which means $(AP_n)^\top AP_n = \mathrm{diag}(d_0, \ldots, d_{n-1})$. Thus $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are conjugate A^2-orthogonal, i.e.,

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

From (3.46)–(3.51), we obtain the algorithm of the COCR method, which is listed in Algorithm 3.9.

Algorithm 3.9 The COCR method (note: $(\bar{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \ldots,$ until convergence do
2:     $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $A\boldsymbol{p}_n = A\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1}$
4:     $\alpha_n = \frac{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}{(\overline{A\boldsymbol{p}}_n, A\boldsymbol{p}_n)}$
5:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$
6:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
7:     $\beta_n = \frac{(\bar{\boldsymbol{r}}_{n+1}, A\boldsymbol{r}_{n+1})}{(\bar{\boldsymbol{r}}_n, A\boldsymbol{r}_n)}$
8: end for
```

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

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

where $A \approx K = K_1K_1^\top$. Then, we have the preconditioned COCR method which is described in Algorithm 3.10.

Algorithm 3.10 The preconditioned COCR method (note: $(\bar{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{p}_{-1} = \boldsymbol{0}$, $\beta_{-1} = 0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \dots,$ until convergence do
2:     $\boldsymbol{p}_n = K^{-1}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $(A\boldsymbol{p}_n = AK^{-1}\boldsymbol{r}_n + \beta_{n-1}A\boldsymbol{p}_{n-1})$
4:     $\alpha_n = \frac{(\overline{K}^{-1}\bar{\boldsymbol{r}}_n, AK^{-1}\boldsymbol{r}_n)}{(\bar{A}\bar{\boldsymbol{p}}_n, K^{-1}A\boldsymbol{p}_n)}$
5:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$
6:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
7:     $(K^{-1}\boldsymbol{r}_{n+1} = K^{-1}\boldsymbol{r}_n - \alpha_n K^{-1}A\boldsymbol{p}_n)$
8:     $\beta_n = \frac{(\overline{K}^{-1}\bar{\boldsymbol{r}}_{n+1}, AK^{-1}\boldsymbol{r}_{n+1})}{(\overline{K}^{-1}\bar{\boldsymbol{r}}_n, AK^{-1}\boldsymbol{r}_n)}$
9: end for
```

Note that lines 3 and 7 in Algorithm 3.10 are added for reducing the number of matrix–vector multiplications of the form $A\boldsymbol{v}$ and solving linear systems of the form $K\boldsymbol{v} = \boldsymbol{w}$, which is the same technique as described in the preconditioned CR method in Sect. 3.1.2.

3.2.3 The Quasi-Minimal Residual (QMR_SYM) Method

The QMR_SYM method [61] that is named in van der Vorst's book [196, p.112] is also a well-known Krylov subspace method for solving complex symmetric linear systems, which can be derived from the complex symmetric Lanczos process described in Sect. 1.9.3.

This subsection describes the derivation process of the QMR_SYM method. Let V_n be an N-by-n matrix whose column vectors are produced by the complex symmetric Lanczos process (Algorithm 1.10). Then, the QMR_SYM method finds an approximate solution from

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n\boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathbb{C}^n. \tag{3.52}$$

Here, $\boldsymbol{y}_n$ is to be determined by a condition as described later. Note that $\boldsymbol{x}_n$ lies in the affine space $\boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)$. The corresponding residual vector is

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

From the matrix form of the complex symmetric Lanczos process (1.41), 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 $\beta := \|\boldsymbol{r}_0\|$. The above derivation process is similar to that of the MINRES method in Sect. 3.1.3. The difference is that if we choose $\boldsymbol{y}_n$ such that the norm of the residual is minimized as well as the MINRES method, then it may lead to a large amount of computational costs since

$$\|\boldsymbol{r}_n\| = \|V_{n+1}(\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}_n)\| \neq \|(\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}_n)\|, \tag{3.53}$$

due to the fact that $V_{n+1}^{\mathrm{H}}V_{n+1} \neq I$. This means that we need to minimize $\|V_{n+1}(\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}_n)\|$ if we try to obtain a minimal residual approximate solution. As a result, the computational costs and memory requirements grow as the number of iterations increases.

The QMR_SYM method, therefore, is designed not by the minimal residual approach but by a quasi-minimal residual approach, i.e., choosing $\boldsymbol{y}_n$ such that

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y}\in\mathbb{C}^n} \|\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\|. \tag{3.54}$$

The resulting vector $\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}_n$ is referred to as a quasi-residual vector. Here, we note that if V_n satisfies the relation $V_n^{\mathrm{H}}V_n = I_n$, then the above choice leads to the minimization of the residual norm. Since A is complex symmetric, in general 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 [61], which has the form

$$A = B + i\sigma I, \quad B = B^{\top} \in \mathbb{R}^{N\times N},\ \sigma \in \mathbb{R}.$$

To achieve the minimization (3.54), Givens rotations described in Sect. 3.1.3 play an important role. Multiplying $\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}$ by Givens rotations $Q_n = G_n \cdots G_1$ such that $Q_n^{\mathrm{H}}Q_n = I_n$ and

$$Q_n T_{n+1,n} = \begin{bmatrix} R_n \\ \boldsymbol{0}^{\top} \end{bmatrix},$$

we have

$$\min_{\boldsymbol{y}\in\mathbb{C}^n} \|\beta\boldsymbol{e}_1 - T_{n+1,n}\boldsymbol{y}\| = \min_{\boldsymbol{y}\in\mathbb{C}^n} \left\| \begin{bmatrix} \boldsymbol{g}_n \\ g_{n+1} \end{bmatrix} - \begin{bmatrix} R_n \\ \boldsymbol{0}^{\top} \end{bmatrix} \boldsymbol{y} \right\|, \tag{3.55}$$

where

$$\begin{bmatrix} \boldsymbol{g}_n \\ g_{n+1} \end{bmatrix} = \beta Q_n \boldsymbol{e}_1$$

and R_n is an upper triangular matrix. Thus, we have $\boldsymbol{y}_n = R_n^{-1}\boldsymbol{g}_n$ as the solution of (3.55), from which, together with (3.52), we obtain the approximate solution $\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n R_n^{-1}\boldsymbol{g}_n$. Similar to the MINRES method, introducing $P_n := V_n R_n^{-1}$ gives a three-term recurrence relation. Then, we have a more practical formula of $\boldsymbol{x}_n$:

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

where g_i and $\boldsymbol{p}_i$ are the ith element of $\boldsymbol{g}_n$ and the ith column of P_n respectively. The complete algorithm of the QMR_SYM method is described in Algorithm 3.11.

Unlike the MINRES method, the residual norm $\|\boldsymbol{r}_n\|$ is not obtained from (3.55) due to the relation (3.53). Thus the problem is how to compute $\|\boldsymbol{r}_n\|$ for evaluating the quality of approximate solutions. One simple solution is to compute $\|\boldsymbol{b} - A\boldsymbol{x}_n\|$, which requires an additional matrix–vector multiplication per iteration. On the other hand, it is possible to circumvent the computation of the matrix–vector multiplication. The technique is given next. Similar to the MINRES method, the approximate solutions of the QMR_SYM method are given by

$$\begin{aligned}\boldsymbol{p}_n &= \frac{1}{t_{n,n}}(\boldsymbol{v}_n - t_{n-2,n}\boldsymbol{p}_{n-2} - t_{n-1,n}\boldsymbol{p}_{n-1}),\\ \boldsymbol{x}_n &= \boldsymbol{x}_{n-1} + g_n \boldsymbol{p}_n.\end{aligned}$$

Since $\boldsymbol{r}_n = \boldsymbol{b} - A\boldsymbol{x}_n$, we have

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - g_n A\boldsymbol{p}_n.$$

Here, $A\boldsymbol{p}_n$ can be updated by

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

Since $A\boldsymbol{v}_n$ is computed by the complex symmetric Lanczos process, we can compute the residual 2-norm $\|\boldsymbol{r}_n\|$ without the direct computation of $\|\boldsymbol{b} - A\boldsymbol{x}_n\|$, leading to a cost-efficient evaluation of $\|\boldsymbol{r}_n\|$ to check the convergence. These recurrences are added to lines 24–26 in Algorithm 3.11.

3.3 Non-Hermitian Linear Systems

Let us recall that the CG method and the CR method (or the MINRES method) have the following two favorable properties:

1. The residual vectors satisfy Ritz-Galerkin approach (1.24) or minimal residual approach (1.26).
2. The residual vectors are generated by short-term recurrences.

As described in Sect. 3.1, the first property determines the optimality of approximate solutions, and the second property plays an important role in keeping the computational costs and memory requirements constant at each iteration step. Recalling that the Ritz–Galerkin approach (1.24) is for the CG method, and the minimal residual approach (1.26) is for the CR method and the MINRES method.

Algorithm 3.11 The QMR_SYM method (note: $(\bar{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: $x_0 \in \mathbb{C}^N$, $\beta_0 = 0$, $v_0 = p_{-1} = p_0 = 0$, $r_0 = b - Ax_0$
Output: $x_n$
1: $g = (\|r_0\|, 0, \dots, 0)^\top$, $v_1 = r_0/\|r_0\|$
2: for $n = 1, 2, \dots$ do
3:    (Complex symmetric Lanczos process)
4:    $\alpha_n = (\bar{v}_n, Av_n)$
5:    $\tilde{v}_{n+1} = Av_n - \alpha_n v_n - \beta_{n-1} v_{n-1}$
6:    $\beta_n = (\bar{\tilde{v}}_{n+1}, \tilde{v}_{n+1})^{1/2}$
7:    $v_{n+1} = \tilde{v}_{n+1}/\beta_n$
8:    $t_{n-1,n} = \beta_{n-1}$,  $t_{n,n} = \alpha_n$
9:    $t_{n+1,n} = \beta_n$
10:   (Givens rotations)
11:   for $i = \max\{1, n-2\}, \dots, n-1$ do
12:      $\begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{bmatrix} \begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix}$
13:   end for
14:   $c_n = \frac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}}$
15:   $\bar{s}_n = \frac{t_{n+1,n}}{t_{n,n}} c_n$
16:   $t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n}$
17:   $t_{n+1,n} = 0$
18:   $\begin{bmatrix} g_n \\ g_{n+1} \end{bmatrix} = \begin{bmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{bmatrix} \begin{bmatrix} g_n \\ 0 \end{bmatrix}$
19:   (Update $x_n$)
20:   $p_n = (v_n - t_{n-2,n} p_{n-2}$
21:            $- t_{n-1,n} p_{n-1})/t_{n,n}$
22:   $x_n = x_{n-1} + g_n p_n$
23:   (Check convergence)
24:   $Ap_n = (Av_n - t_{n-2,n} Ap_{n-2}$
25:            $- t_{n-1,n} Ap_{n-1})/t_{n,n}$
26:   $r_n = r_{n-1} - g_n Ap_n$
27:   if $\|r_n\|/\|b\| \le \epsilon$, then stop
28: end for
```

Here, a question arises: are there Krylov subspace methods satisfying both the properties for non-Hermitian linear systems? The answer is negative, which was proved in [57] and known as the *Faber–Manteuffel theorem*. More concretely, if matrix A has the form

$$A = e^{i\phi}(icI + G) \quad (c \ge 0,\ 0 \le \phi \le 2\pi,\ B^{-1}G^{\mathrm{H}}B = G),$$

then algorithms satisfying the both properties can be constructed. For the details, see [57] and Greenbaum's book [81, Chap. 6]. See also [121] and the references therein.

For solving non-Hermitian linear systems, one may consider the following transformed linear systems:

$$A^{\mathrm{H}}A\boldsymbol{x} = A^{\mathrm{H}}\boldsymbol{b}. \tag{3.56}$$

If A is nonsingular, then $A^{\mathrm{H}}A$ is Hermitian positive definite. Thus the CG method can be applied to the transformed linear systems. However, the condition number of $A^{\mathrm{H}}A$ may be twice as large as the condition number of the original matrix A. Hence, from Theorem 3.2, the speed of convergence may be slow. Furthermore, if the condition number is very large, then it follows from the error analysis (1.3), the accuracy of an obtained approximate solution may be very low. Note that the error analysis (1.3) does not depend on numerical algorithms for solving linear systems. This means that the accuracy of an approximate solution computed by any numerical algorithm may become low.

On the other hand, if the condition number $A^{\mathrm{H}}A$ is not so large, there are cases where applying the CG method to (3.56) is a method of choice. In fact, if A is a unitary matrix, then $A^{\mathrm{H}}A = I$, which indicates that the CG method gives the solution within only one iteration. Thus, we can guess that this approach is particularly useful for the case where A is close to a unitary matrix. Well-known algorithms in this approach are the CGNE method and the CGNR method, which will be described in Sect. 3.4.1.

In what follows, Krylov subspace methods for non-Hermitian linear systems are derived. We will see that the BiCG method, the BiCR method, and the QMR method can be regarded as extensions of the CG method, the CR method, and the MINRES method respectively, and these are based on the bi-Lanczos process (or a bi-Lanczos-like process), and the GMRES method can be regarded as an extension of the MINRES method, and the GMRES method is based on the Arnoldi process. We will also see how the BiCG method has been improved via product-type Krylov subspace methods and the IDR(s) method.

3.3.1 The Bi-Conjugate Gradient (BiCG) Method

Similar to the derivation process of the CG method in Sect. 3.1.1, the BiCG method [58, 119] can be derived by the bi-Lanczos process in Section 1.9.2.

In this subsection, we give a concise way of the derivation to see that the BiCG method for non-Hermitian linear systems can be derived from the preconditioned COCG method (Algorithm 3.7). First, we consider the following $2N \times 2N$ complex symmetric linear system:

$$\begin{bmatrix} O & A \\ A^{\top} & O \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{x}}^{*} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \bar{\boldsymbol{b}}^{*} \end{bmatrix}, \text{ or } \tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}. \tag{3.57}$$

We can see from the system (3.57) that it is mathematically equivalent to

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

If we apply the algorithm of COCG with the preconditioner

$$\tilde{M} = \begin{bmatrix} O & I \\ I & O \end{bmatrix}, \quad I : \text{identity matrix}, \tag{3.58}$$

to (3.57), then the resulting algorithm at the nth iteration step can be written as

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

Substituting $\tilde{M}^{-1}(=\tilde{M})$ of (3.58) and the vectors

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

into the previous recurrences, and using the following results:

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

we readily obtain the algorithm of BiCG for solving non-Hermitian linear systems described in Algorithm 3.12. If matrix A is real non-symmetric, the aforementioned derivation process corresponds to the derivation in van der Vorst's book [196, Chap. 7], where the BiCG method for real non-symmetric linear systems is derived from the CG method.

It can be seen from Algorithm 3.12 that the choice $\boldsymbol{r}_0^* = \boldsymbol{r}_0$ reduces to the CG method if the coefficient matrix A is Hermitian.

Similar to the CG iterates as shown in Proposition 3.1, the BiCG iterates have the following relations:

Proposition 3.2 *Let $\boldsymbol{p}_n, \boldsymbol{r}_n$ be the vectors in the BiCG method. Then,*

1. $(\boldsymbol{r}_i^*, \boldsymbol{r}_j) = 0$ *for $i \neq j$,*
2. $(\boldsymbol{p}_i^*, A\boldsymbol{p}_j) = 0$ *for $i \neq j$.*

Algorithm 3.12 The BiCG method

```
Input: $x_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, p_{-1} = p^*_{-1} = 0, r_0 = b - Ax_0$
Input: Choose $r^*_0 \in \mathbb{C}^N$, e.g., $r^*_0 = r_0$
Output: $x_n$
1: for $n = 0, 1, 2, \dots,$ until convergence do
2:     $p_n = r_n + \beta_{n-1} p_{n-1}$,   $p^*_n = r^*_n + \overline{\beta}_{n-1} p^*_{n-1}$
3:     $\alpha_n = \frac{(r^*_n, r_n)}{(p^*_n, Ap_n)}$
4:     $x_{n+1} = x_n + \alpha_n p_n$
5:     $r_{n+1} = r_n - \alpha_n A p_n$,   $r^*_{n+1} = r^*_n - \overline{\alpha}_n A^{\mathrm{H}} p^*_n$
6:     $\beta_n = \frac{(r^*_{n+1}, r_{n+1})}{(r^*_n, r_n)}$
7: end for
```

From Proposition 3.2, the following properties hold true:

Corollary 3.2 *Let $\boldsymbol{p}_n, \boldsymbol{r}_n$ be the vectors in the BiCG method. Then,*

1. $\boldsymbol{r}_n \perp \mathcal{K}_n(A^{\mathrm{H}}, \boldsymbol{r}^*_0)$,
2. $A\boldsymbol{p}_n \perp \mathcal{K}_n(A^{\mathrm{H}}, \boldsymbol{r}^*_0)$.

In what follows, we give the other computational formulas for α_n and β_n of the BiCG method, which will be used to derive the product-type Krylov subspace methods in Section 3.3.8.

Since $\boldsymbol{r}^*_n \in \mathcal{K}_{n+1}(A^{\mathrm{H}}, \boldsymbol{r}^*_0)$, the vector $\boldsymbol{r}^*_n$ can be written as

$$\begin{aligned} \boldsymbol{r}^*_n &= c_n (A^{\mathrm{H}})^n \boldsymbol{r}^*_0 + \underbrace{c_{n-1}(A^{\mathrm{H}})^{n-1}\boldsymbol{r}^*_0 + \cdots + c_0 \boldsymbol{r}^*_0}_{z} \\ &= c_n (A^{\mathrm{H}})^n \boldsymbol{r}^*_0 + \boldsymbol{z}, \quad \boldsymbol{z} \in \mathcal{K}_n(A^{\mathrm{H}}, \boldsymbol{r}^*_0). \end{aligned}$$

From the above equation and Corollary 3.2, it follows that

$$\begin{aligned} (\boldsymbol{r}^*_n, \boldsymbol{r}_n) &= (c_n (A^{\mathrm{H}})^n \boldsymbol{r}^*_0 + \boldsymbol{z}, \boldsymbol{r}_n) = c_n((A^{\mathrm{H}})^n \boldsymbol{r}^*_0, \boldsymbol{r}_n) + (\boldsymbol{z}, \boldsymbol{r}_n) \\ &= c_n((A^{\mathrm{H}})^n \boldsymbol{r}^*_0, \boldsymbol{r}_n) \\ (\boldsymbol{r}^*_n, A\boldsymbol{p}_n) &= (c_n (A^{\mathrm{H}})^n \boldsymbol{r}^*_0 + \boldsymbol{z}, A\boldsymbol{p}_n) = c_n((A^{\mathrm{H}})^n \boldsymbol{r}^*_0, A\boldsymbol{p}_n) + (\boldsymbol{z}, A\boldsymbol{p}_n) \\ &= c_n((A^{\mathrm{H}})^n \boldsymbol{r}^*_0, A\boldsymbol{p}_n). \end{aligned}$$

Thus, α_n of the BiCG method (Algorithm 3.12) can be rewritten as

$$\alpha_n = \frac{(\boldsymbol{r}^*_n, \boldsymbol{r}_n)}{(\boldsymbol{p}^*_n, A\boldsymbol{p}_n)} = \frac{((A^{\mathrm{H}})^n \boldsymbol{r}^*_0, \boldsymbol{r}_n)}{((A^{\mathrm{H}})^n \boldsymbol{r}^*_0, A\boldsymbol{p}_n)}. \tag{3.59}$$

Similarly, we give the other computational formulas for β_n. From Corollary 3.2, two vectors $(A^{\mathrm{H}})^n \boldsymbol{r}^*_0 \in \mathcal{K}_{n+1}(A^{\mathrm{H}}, \boldsymbol{r}^*_0)$ and $A\boldsymbol{p}_{n+1}$ are orthogonal, and thus

$$\begin{aligned}
0 &= ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{p}_{n+1}) \\
&= ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{r}_{n+1} + \beta_n A\boldsymbol{p}_n) \\
&= ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{r}_{n+1}) + \beta_n ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{p}_n) \\
&= ((A^{\mathrm{H}})^{n+1} \boldsymbol{r}_0^*, \boldsymbol{r}_{n+1}) + \beta_n ((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{p}_n),
\end{aligned}$$

which immediately leads to

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

Using (3.59) together with the above result yields

$$\beta_n = -\alpha_n \frac{((A^{\mathrm{H}})^{n+1} \boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{((A^{\mathrm{H}})^n \boldsymbol{r}_0^*, \boldsymbol{r}_n)}. \tag{3.60}$$

3.3.2 The Composite Step Bi-Conjugate Gradient (CSBiCG) Method

From the algorithm of the BiCG method (Algorithm 3.12), there are two kinds of possible (near) breakdown:

1. $\rho_n := (\boldsymbol{r}_n^*, \boldsymbol{r}_n) = 0$ for $\boldsymbol{r}_n \neq \boldsymbol{0}$ (Lanczos breakdown),
2. $\sigma_n := (\boldsymbol{p}_n^*, A\boldsymbol{p}_n) = 0$ (pivot breakdown).

The Lanczos breakdown can be circumvented by the look-ahead Lanczos process in [145, 189]. Bank and Chan focused on the pivot breakdown and proposed the Composite Step Bi-Conjugate Gradient (CSBiCG) method to cure the pivot breakdown [16, 17] under the assumption that the Lanczos breakdown does not occur. Below is a brief explanation of the algorithm that is based on [17].

If $\sigma_n = 0$, then $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - (\rho_n/\sigma_n)A\boldsymbol{p}_n$ and $\boldsymbol{r}_{n+1}^* = \boldsymbol{r}_n^* - (\overline{\rho}_n/\overline{\sigma}_n)A^{\mathrm{H}}\boldsymbol{p}_n^*$ cannot be obtained. Consider auxiliary vectors:

$$\begin{aligned}
\boldsymbol{z}_{n+1} &= \sigma_n \boldsymbol{r}_n - \rho_n A\boldsymbol{p}_n, \\
\boldsymbol{z}_{n+1}^* &= \overline{\sigma}_n \boldsymbol{r}_n - \overline{\rho}_n A^{\mathrm{H}}\boldsymbol{p}_n^*.
\end{aligned}$$

Then, $\boldsymbol{z}_{n+1}$ and $\boldsymbol{z}_{n+1}^*$ exist even if $\sigma_n = 0$, and $\boldsymbol{z}_{n+1}$ and $\boldsymbol{z}_{n+1}^*$ belong to $\mathcal{K}_{n+2}(A, \boldsymbol{r}_0)$ and $\mathcal{K}_{n+2}(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$ respectively. The composite step technique then considers $\boldsymbol{r}_{n+2}$ and $\boldsymbol{r}_{n+2}^*$ as follows:

$$\begin{aligned}
\boldsymbol{r}_{n+2} &= \boldsymbol{r}_n - c_1 A\boldsymbol{p}_n - c_2 A\boldsymbol{z}_{n+1}, \\
\boldsymbol{r}_{n+2}^* &= \boldsymbol{r}_n^* - c_1^* A^{\mathrm{H}}\boldsymbol{p}_n^* - c_2^* A^{\mathrm{H}}\boldsymbol{z}_{n+1}^*,
\end{aligned}$$

where c_1, c_2, c_1^*, c_2^* are determined so that

$$\begin{aligned} \boldsymbol{r}_{n+2} &\perp \operatorname{span}\{\boldsymbol{p}_n^*, \boldsymbol{z}_{n+1}^*\}, \\ \boldsymbol{r}_{n+2}^* &\perp \operatorname{span}\{\boldsymbol{p}_n, \boldsymbol{z}_{n+1}\}, \end{aligned}$$

i.e., $(\boldsymbol{r}_{n+2}, \boldsymbol{p}_n^*) = (\boldsymbol{r}_{n+2}, \boldsymbol{z}_{n+1}^*) = (\boldsymbol{r}_{n+2}^*, \boldsymbol{p}_n) = (\boldsymbol{r}_{n+2}^*, \boldsymbol{z}_{n+1}) = 0$. Similarly, let

$$\begin{aligned} \boldsymbol{p}_{n+2} &= \boldsymbol{r}_{n+2} + d_1\boldsymbol{p}_n + d_2\boldsymbol{z}_{n+1}, \\ \boldsymbol{p}_{n+2}^* &= \boldsymbol{r}_{n+2}^* + d_1^*\boldsymbol{p}_n^* + d_2^*\boldsymbol{z}_{n+1}^*, \end{aligned}$$

and d_1, d_2, d_1^*, d_2^* are determined so that

$$\begin{aligned} A\boldsymbol{p}_{n+2} &\perp \operatorname{span}\{\boldsymbol{p}_n^*, \boldsymbol{z}_{n+1}^*\}, \\ A^{\mathrm{H}}\boldsymbol{p}_{n+2}^* &\perp \operatorname{span}\{\boldsymbol{p}_n, \boldsymbol{z}_{n+1}\}. \end{aligned}$$

Then, it is shown in [17, Theorem 4.4] that $(\boldsymbol{r}_i^*, \boldsymbol{r}_j) = (\boldsymbol{p}_i^*, A\boldsymbol{p}_j) = 0$ for $i \neq j$, which means that the composite step residual vector $\boldsymbol{r}_{n+2}$ and $A\boldsymbol{p}_{n+2}$ satisfies

$$\boldsymbol{r}_{n+2} \perp \mathcal{K}_{n+2}(A^{\mathrm{H}}, \boldsymbol{r}_0^*), \quad A\boldsymbol{p}_{n+2} \perp \mathcal{K}_{n+2}(A^{\mathrm{H}}, \boldsymbol{r}_0^*).$$

which are the same properties that the BiCG method has, see Corollary 3.2. The resulting algorithm is described in Algorithm 3.13.

From Algorithm 3.13, the number of matrix–vector multiplications is the same as that of the BiCG method. The composite step techniques are used for other Krylov subspace methods [35, 113].

At each step in Algorithm 3.13, one needs to choose a 1×1 step (BiCG step) or a 2×2 step (composite step). In [17], the following choice is proposed:

```
1: **if** $\|\boldsymbol{z}_{n+1}\| \leq \|\boldsymbol{r}_n\|\,|\sigma_n|$ **then**
2:    $1 \times 1$ step
3: **else**
4:    $\nu_{n+2} = \|\delta_n \boldsymbol{r}_n - \rho_n^3 \zeta_{n+1} \boldsymbol{q}_n - \theta_{n+1}\rho_n^2 \boldsymbol{y}_{n+1}\|$
5:    **if** $\nu_{n+2}|\sigma_n| < \|\boldsymbol{z}_{n+1}\|\,|\delta_n|$ **then**
6:       $2 \times 2$ step
7:    **else**
8:       $1 \times 1$ step
9:    **end if**
10: **end if**
```

This choice leads to not only avoiding (near) pivot breakdowns but also some smoothing of the convergence history in residual norms.

Algorithm 3.13 The Composite Step BiCG method

```
Input: $x_0 \in \mathbb{C}^N, r_0 = b - Ax_0$
Input: Choose $r_0^* \in \mathbb{C}^N$, e.g., $r_0^* = r_0$
Output: $x_n$
1: $p_0 = r_0, \quad p_0^* = r_0^*$
2: $q_0 = Ap_0, \quad q_0^* = A^{\mathrm{H}} p_0^*$
3: $\rho_0 = (r_0^*, r_0)$
4: $n = 0$
5: while until convergence do
6:    $\sigma_n = (p_n^*, q_n)$
7:    $z_{n+1} = \sigma_n r_n - \rho_n q_n, \quad z_{n+1}^* = \overline{\sigma}_n r_n^* - \overline{\rho}_n q_n^*$
8:    $y_{n+1} = Az_{n+1}, \quad y_{n+1}^* = A^{\mathrm{H}} z_{n+1}^*$
9:    $\theta_{n+1} = (z_{n+1}^*, z_{n+1})$
10:   $\zeta_{n+1} = (z_{n+1}^*, y_{n+1})$
11:   if $1 \times 1$ step then
12:      $\alpha = \rho_n / \sigma_n$
13:      $\rho_{n+1} = \theta_{n+1} / \sigma_n^2$
14:      $\beta_{n+1} = \rho_{n+1} / \rho_n$
15:      $x_{n+1} = x_n + \alpha_n p_n$
16:      $r_{n+1} = r_n - \alpha_n q_n, \quad r_{n+1}^* = r_n^* - \overline{\alpha}_n q_n^*$
17:      $p_{n+1} = z_{n+1}/\sigma_n + \beta_{n+1} p_n, \quad p_{n+1}^* = z_{n+1}^*/\overline{\sigma}_n + \overline{\beta}_{n+1} p_n^*$
18:      $q_{n+1} = y_{n+1}/\sigma_n + \beta_{n+1} q_n, \quad q_{n+1}^* = y_{n+1}^*/\overline{\sigma}_n + \overline{\beta}_{n+1} q_n^*$
19:      $n = n + 1$
20:   end if
21:   if $2 \times 2$ step then
22:      $\delta_n = \sigma_n \zeta_{n+1} \rho_n^2 - \theta_{n+1}^2$
23:      $\alpha_n = \zeta_{n+1} \rho_n^3 / \delta_n, \quad \alpha_{n+1} = \theta_{n+1} \rho_n^2 / \delta_n$
24:      $x_{n+2} = x_n + \alpha_n p_n + \alpha_{n+1} z_{n+1}$
25:      $r_{n+2} = r_n - \alpha_n q_n - \alpha_{n+1} y_{n+1}, \quad r_{n+2}^* = r_n^* - \overline{\alpha}_n q_n^* - \overline{\alpha}_{n+1} y_{n+1}$
26:      $\rho_{n+2} = (r_{n+2}^*, r_{n+2})$
27:      $\beta_{n+1} = \rho_{n+2} / \rho_n, \quad \beta_{n+2} = \rho_{n+2} \sigma_n / \theta_{n+1}$
28:      $p_{n+2} = r_{n+2} + \beta_{n+1} p_n + \beta_{n+2} z_{n+1}, \quad p_{n+2}^* = r_{n+2}^* + \overline{\beta}_{n+1} p_n^* + \overline{\beta}_{n+2} z_{n+1}^*$
29:      $q_{n+2} = Ap_{n+2}, \quad q_{n+2}^* = A^{\mathrm{H}} p_{n+2}^*$
30:      $n = n + 2$
31:   end if
32: end while
```

The following theorem shows the best approximation result of the (composite step) BiCG method:

Theorem 3.3 ([16]) *Consider applying the (composite step) BiCG method to real nonsymmetric linear systems with initial guess $\boldsymbol{x}_0 = \boldsymbol{0}$ and $\boldsymbol{r}_0^* = \boldsymbol{r}_0$. Suppose that for all $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^N$, we have*

$$|\boldsymbol{w}^\top A \boldsymbol{v}| \le \Gamma \|\boldsymbol{v}\| \|\boldsymbol{w}\|,$$

where Γ is a constant independent of $\boldsymbol{v}$ and $\boldsymbol{w}$. Further, suppose that for those steps in the (composite step) BiCG method in which we compute an approximation $\boldsymbol{x}_n$, we have

$$\inf_{\substack{\boldsymbol{v}\in\mathcal{K}_n(A,\boldsymbol{r}_0)\\ \|\boldsymbol{v}\|=1}} \sup_{\substack{\boldsymbol{w}\in\mathcal{K}_n(A^\top,\boldsymbol{r}_0^*)\\ \|\boldsymbol{w}\|\le 1}} \boldsymbol{w}^\top A\boldsymbol{v} \ge \delta_n \ge \delta > 0.$$

Then

$$\|\boldsymbol{x}-\boldsymbol{x}_n\| \le (1+\Gamma/\delta) \inf_{\boldsymbol{v}\in\mathcal{K}_n(A,\boldsymbol{r}_0)} \|\boldsymbol{x}-\boldsymbol{v}\|.$$

Theorem 3.3 is a simplified result of Theorem 4.1 in [16]. From Theorem 3.3, we see that the (composite step) BiCG method produces an approximate solution close to the best approximate solution in error norm when Γ/δ is close to 0.

3.3.3 The Bi-Conjugate Residual (BiCR) Method

The BiCR method [169] is an extension of the CR method to non-Hermitian linear systems.[2] In this subsection, we show that the BiCR method can be derived from the preconditioned COCR method in Algorithm 3.10. When we apply the algorithm of the COCR method with the preconditioner (3.58) to (3.57), the resulting algorithm at the *n*th iteration step can be written as

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

Substituting $\tilde{M}^{-1}(=\tilde{M})$ of (3.58) and the vectors

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

into the previous recurrences, and using the following results:

[2] The method was mentioned in the unpublished paper [38] by Chronopulous and Ma in 1989, and Dr. Chronopulous emailed the author about this fact on Apr. 6th, 2020.

$$\begin{aligned}(\overline{\tilde{M}^{-1}\tilde{r}_n^{\mathrm{COCR}}}, \tilde{A}\tilde{M}^{-1}\tilde{r}_n^{\mathrm{COCR}}) &= \left[\bar{r}_n^{*\top} r_n^\top\right]\begin{bmatrix} Ar_n \\ A^\top \bar{r}_n^* \end{bmatrix} \\ &= \bar{r}_n^{*\top} Ar_n + r_n^\top A^\top \bar{r}_n^* \\ &= r_n^{*\mathrm{H}} Ar_n + (r_n^\top A^\top \bar{r}_n^*)^\top \\ &= (r_n^*, Ar_n) + r_n^{*\mathrm{H}} Ar_n \\ &= 2(r_n^*, Ar_n)\end{aligned}$$

and

$$\begin{aligned}(\overline{\tilde{A}\tilde{p}_n^{\mathrm{COCR}}}, \tilde{M}^{-1}\tilde{A}\tilde{p}_n^{\mathrm{COCR}}) &= \left[(Ap_n)^\top (A^\top \bar{p}_n^*)^\top\right]\begin{bmatrix} A^\top \bar{p}_n^* \\ Ap_n \end{bmatrix} \\ &= (Ap_n)^\top A^\top \bar{p}_n^* + (A^\top \bar{p}_n^*)^\top Ap_n \\ &= (p_n^\top A^\top A^\top \bar{p}_n^*)^\top + (A^{\mathrm{H}} p_n^*)^{\mathrm{H}} Ap_n \\ &= (A^{\mathrm{H}} p_n^*)^{\mathrm{H}} Ap_n + (A^{\mathrm{H}} p_n^*, Ap_n) \\ &= 2(A^{\mathrm{H}} p_n^*, Ap_n),\end{aligned}$$

we readily obtain the algorithm of the BiCR method for solving non-Hermitian linear systems.

Algorithm 3.14 The BiCR method

```
Input: $x_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, p_{-1} = p_{-1}^* = 0, r_0 = b - Ax_0$
Input: Choose $r_0^* \in \mathbb{C}^N$, e.g., $r_0^* = r_0$
Output: $x_n$
1: for $n = 0, 1, 2, \dots,$ until convergence do
2:     $p_n = r_n + \beta_{n-1} p_{n-1}$,   $p_n^* = r_n^* + \overline{\beta}_{n-1} p_{n-1}^*$
3:     $Ap_n = Ar_n + \beta_{n-1} Ap_{n-1}$
4:     $\alpha_n = \frac{(r_n^*, Ar_n)}{(A^{\mathrm{H}} p_n^*, Ap_n)}$
5:     $x_{n+1} = x_n + \alpha_n p_n$
6:     $r_{n+1} = r_n - \alpha_n Ap_n$,   $r_{n+1}^* = r_n^* - \overline{\alpha}_n A^{\mathrm{H}} p_n^*$
7:     $\beta_n = \frac{(r_{n+1}^*, Ar_{n+1})}{(r_n^*, Ar_n)}$
8: end for
```

Similar to Proposition 3.2, BiCR iterates hold the following properties:

Proposition 3.3 *Let p_n, r_n be the vectors in the BiCR method. Then,*

1. $(r_i^*, Ar_j) = 0$ *for $i \neq j$,*
2. $(A^{\mathrm{H}} p_i^*, Ap_j) = 0$ *for $i \neq j$.*

If the coefficient matrix A is Hermitian or real symmetric, the choice $r_0^* = r_0$ reduces to the CR method that generates optimal approximate solutions in terms of residual 2-norms. Thus, if A is close to Hermitian or real symmetric, it is expected that the BiCR method has smooth convergence behavior and generates near minimal residual solutions.

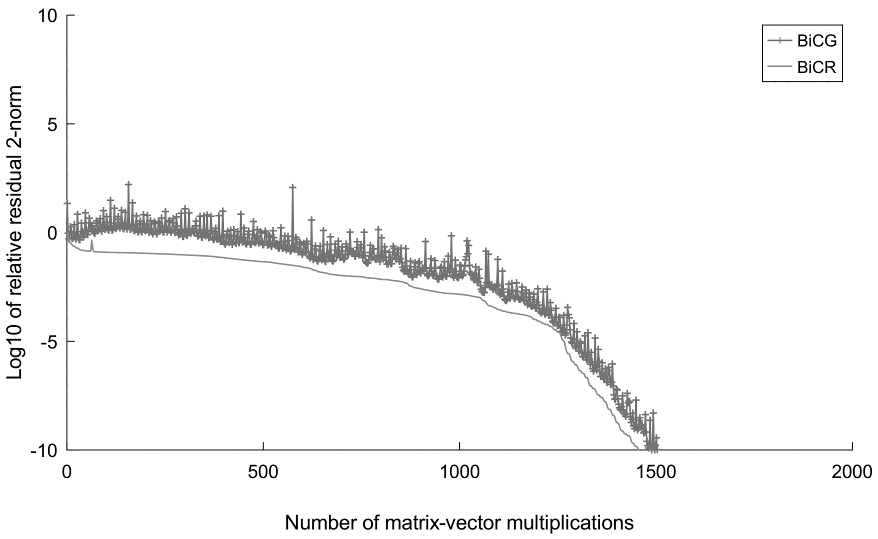


Fig. 3.1 Convergence histories of BiCG and BiCR for $b_1 = b_2 = b_3 = 0.1$

To see this, we consider nonsymmetric linear systems (2.18) with parameters $N = 20$, $a_1 = a_2 = a_3 = 1$, two different numbers $b_1 = b_2 = b_3 = 0.1$ (near symmetric matrix) and $b_1 = b_2 = b_3 = 10$ (far symmetric matrix), and $c = 700$. The horizontal axis is the number of matrix–vector multiplications. The stopping criterion used for the BiCG method and the BiCR method is $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \le 10^{-10}$.

The results for $b_1 = b_2 = b_3 = 0.1$ (near symmetric matrix) are shown in Fig. 3.1. We see from Fig. 3.1 that the BiCR method shows much smoother convergence behavior than the BiCG method.

The results for $b_1 = b_2 = b_3 = 0.1$ (far symmetric matrix) are shown in Fig. 3.2. From Fig. 3.2, the BiCG method and the BiCR method have similar convergence behavior, and the BiCR method does not show smooth convergence behavior.

3.3.4 The Quasi-Minimal Residual (QMR) Method

In this subsection, we describe the idea of the Quasi-Minimal Residual (QMR) method [66]. The derivation process of the QMR method is almost the same as that of the QMR_SYM method in Sect. 3.2.3. The main difference is the use not of the complex symmetric Lanczos process but the bi-Lanczos process. Let V_n be the bi-orthogonal basis of $\mathcal{K}_n(A, \boldsymbol{r}_0)$ via the bi-Lanczos process given in Algorithm 1.9. Then, the QMR method finds an approximate solution $\boldsymbol{x}_n$ such that $\boldsymbol{x}_n$ lies in the affine space $\boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)$, i.e.,

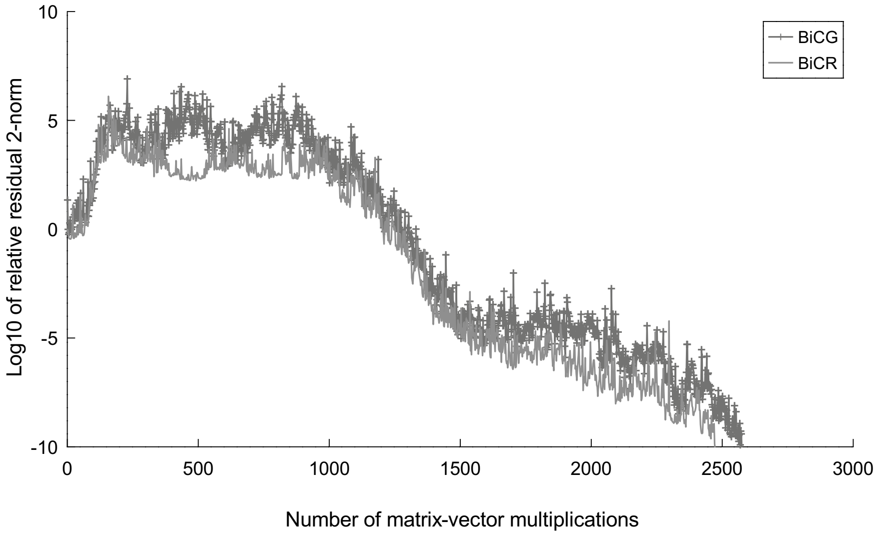


Fig. 3.2 Convergence histories of BiCG and BiCR for $b_1 = b_2 = b_3 = 10$

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

The corresponding residual vector is given by

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

From the matrix form of the bi-Lanczos process (1.40), 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 $\beta := \|\boldsymbol{r}_0\|$. The above derivation process is very similar to that of the MINRES method; however, in this case, if we choose $\boldsymbol{y}_n$ such that the norm of the residual is minimized, then it may lead to a large amount of computational costs and memory requirement as described in Sect. 3.2.3. Similar to the QMR_SYM method, the QMR method chooses $\boldsymbol{y}_n$ such that

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y} \in \mathbb{C}^n} \|\beta \boldsymbol{e}_1 - T_{n+1,n} \boldsymbol{y}\|. \tag{3.61}$$

The minimization problem can be solved efficiently by Givens rotations. Following the solution of (3.31) and Sect. 3.2.3, the QMR method is obtained and the algorithm is described in Algorithm 3.15.

Finally, one should note that Algorithm 3.15, as well as the BiCG method and the BiCR method, has a possibility of breakdown (zero division). In finite precision arithmetic, such breakdown is very rare; however, near breakdown may occur, and

Algorithm 3.15 The QMR method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_0 = 0$, $\gamma_0 = 0$, $\boldsymbol{v}_0 = \boldsymbol{0}$, $\boldsymbol{w}_0 = \boldsymbol{0}$, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \dots, 0)^\top$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$
2: for $n = 1, 2, \dots$ do
3:    (Bi-Lanczos process)
4:    $\alpha_n = (\boldsymbol{w}_n, A\boldsymbol{v}_n)$,
5:    $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n\boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$
6:    $\tilde{\boldsymbol{w}}_{n+1} = A^{\mathrm{H}}\boldsymbol{w}_n - \overline{\alpha}_n\boldsymbol{w}_n - \gamma_{n-1}\boldsymbol{w}_{n-1}$
7:    $\gamma_n = \|\tilde{\boldsymbol{v}}_{n+1}\|_2$
8:    $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\gamma_n$
9:    $\beta_n = (\tilde{\boldsymbol{w}}_{n+1}, \boldsymbol{v}_{n+1})$
10:   $\boldsymbol{w}_{n+1} = \tilde{\boldsymbol{w}}_{n+1}/\overline{\beta}_n$
11:   (Givens rotations)
12:   for $i = \max\{1, n-2\}, \dots, n-1$ do
13:      $\begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -\overline{s}_i & c_i \end{bmatrix} \begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix}$
14:   end for
15:   $c_n = \frac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}}$
16:   $\overline{s}_n = \frac{t_{n+1,n}}{t_{n,n}} c_n$
17:   $t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n}$
18:   $t_{n+1,n} = 0$
19:   $\begin{bmatrix} g_n \\ g_{n+1} \end{bmatrix} = \begin{bmatrix} c_n & s_n \\ -\overline{s}_n & c_n \end{bmatrix} \begin{bmatrix} g_n \\ 0 \end{bmatrix}$
20:   (Update $\boldsymbol{x}_n$)
21:   $\boldsymbol{p}_n = (\boldsymbol{v}_n - t_{n-2,n}\boldsymbol{p}_{n-2}$
22:          $- t_{n-1,n}\boldsymbol{p}_{n-1})/t_{n,n}$
23:   $\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + g_n\boldsymbol{p}_n$
24:   (Check convergence)
25:   $A\boldsymbol{p}_n = (A\boldsymbol{v}_n - t_{n-2,n}A\boldsymbol{p}_{n-2}$
26:          $- t_{n-1,n}A\boldsymbol{p}_{n-1})/t_{n,n}$
27:   $\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - g_n A\boldsymbol{p}_n$
28:   if $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \le \epsilon$, then stop
29: end for
```

this causes numerical instability. Hence, to avoid the (near) breakdown problem, the QMR method in [66] uses the look-ahead Lanczos process, which was proposed by Taylor [189] and Parlett et al. [145].

3.3.5 *The Generalized Minimal Residual (GMRES) Method*

The derivation process of the GMRES method [152] is closely related to the MINRES method in Sect. 3.1.3. The GMRES method generates $\boldsymbol{x}_n$ that minimizes $\|\boldsymbol{b} - A\boldsymbol{x}_n\|$ over the affine space $\boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)$. This can efficiently be achieved by using the Arnoldi process in Section 1.9.1. Now, we give the derivation process of the GMRES method.

Let V_n be an N-by-n matrix whose columns are the orthonormal basis vectors of $\mathcal{K}_n(A, \boldsymbol{r}_0)$ by the Arnoldi process. Then, since $\boldsymbol{x}_n$ lies in the affine space $\boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)$, we have

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

The corresponding residual vector is written as

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

From the matrix representation of the Arnoldi process in (1.38) it follows that

$$\boldsymbol{r}_n = V_{n+1}(\beta \boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}_n), \tag{3.62}$$

where $\beta := \|\boldsymbol{r}_0\|$. Since $V_{n+1}^{\mathrm{H}} V_{n+1} = I_{n+1}$, i.e., the identity matrix, the 2-norm of the residual vector is given by

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

Hence, the 2-norm of the residual vector can be minimized by choosing $\boldsymbol{y}_n$ as follows:

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y}\in\mathbb{C}^n} \|\beta \boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}\|. \tag{3.63}$$

The vector $\boldsymbol{y}_n$ can be obtained by using Givens rotations. Following the solution of (3.31), the GMRES method is obtained. Now, we describe the algorithm of GMRES in Algorithm 3.16, which is based on the Arnoldi process of modified Gram–Schmidt type (Algorithm 1.8).

Algorithm 3.16 The GMRES method of modified Gram-Schmidt type

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: $\boldsymbol{g} = (\|\boldsymbol{r}_0\|, 0, \dots, 0)^\top$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$
2: for $n = 1, 2, \dots$ do
3:    (Arnoldi process)
4:    $\boldsymbol{t} = A\boldsymbol{v}_n$
5:    for $i = 1, 2, \dots, n$ do
6:       $h_{i,n} = (\boldsymbol{v}_i, \boldsymbol{t})$
7:       $\boldsymbol{t} = \boldsymbol{t} - h_{i,n}\boldsymbol{v}_i$
8:    end for
9:    $h_{n+1,n} = \|\boldsymbol{t}\|$
10:   $\boldsymbol{v}_{n+1} = \boldsymbol{t}/h_{n+1,n}$
11:   (Givens rotations)
12:   for $i = 1, 2, \dots, n-1$ do
13:      $\begin{bmatrix} h_{i,n} \\ h_{i+1,n} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{bmatrix} \begin{bmatrix} h_{i,n} \\ h_{i+1,n} \end{bmatrix}$
14:   end for
15:   $c_n = \frac{|h_{n,n}|}{\sqrt{|h_{n,n}|^2 + |h_{n+1,n}|^2}}$
16:   $\bar{s}_n = \frac{h_{n+1,n}}{h_{n,n}} c_n$
17:   $h_{n,n} = c_n h_{n,n} + s_n h_{n+1,n}$
18:   $h_{n+1,n} = 0$
19:   $\begin{bmatrix} g_n \\ g_{n+1} \end{bmatrix} = \begin{bmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{bmatrix} \begin{bmatrix} g_n \\ 0 \end{bmatrix}$
20:   (Check convergence)
21:   if $|g_{n+1}|/\|\boldsymbol{b}\| \le \epsilon$, then
22:      $\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n H_n^{-1}\boldsymbol{g}$
23:   end if
24: end for
```

Note that computing $\boldsymbol{c} := H_n^{-1}\boldsymbol{g}$ in line 22 of Algorithm 3.16 corresponds to solving linear systems of the form $H_n\boldsymbol{c} = \boldsymbol{g}$. After Givens rotations, H_n becomes not a Hessenberg but an upper triangular matrix. Thus the linear systems can easily be solved by the back substitution as described in (1.6).

In what follows, the convergence analysis of the GMRES method is described.

Proposition 3.4 *Let A be an N-by-N diagonalizable matrix, i.e., $A = X\Lambda X^{-1}$ with $\Lambda = diag\{\lambda_1, \dots, \lambda_N\}$, and let $\boldsymbol{r}_n$ be the m-step GMRES residual vector. Then*

$$\|\boldsymbol{r}_n\| \le \kappa(X)\epsilon_n\|\boldsymbol{r}_0\|,$$

where $\kappa(X) = \|X\| \|X^{-1}\|$ *and*

$$\epsilon^{(n)} = \min_{p \in \mathbb{P}_n, p(0)=1} \max_{i=1,\dots,N} |p(\lambda_i)|.$$

Here, $\mathbb{P}_m$ *denotes the set of all polynomials of degree at most m.*

Proof The GMRES method finds an approximate solution $\boldsymbol{x}_n$ such that

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A, \boldsymbol{r}_0)} \|\boldsymbol{r}_n\|.$$

Since $\boldsymbol{r}_n \in \mathcal{K}_{n+1}(A, \boldsymbol{r}_0)$, the GMRES residual vector can be written in the matrix polynomial form $\boldsymbol{r}_n = (I + c_1 A + c_2 A^2 + \cdots + c_n A^n)\boldsymbol{r}_0$, so that c_i's are determined by minimizing $\|\boldsymbol{r}_n\|$, or equivalently $\boldsymbol{r}_n = p_n^{(\text{opt})}(A)\boldsymbol{r}_0$ with $p_n^{(\text{opt})}(0) = 1$, and $p_n^{(\text{opt})}$ is determined by minimizing $\|\boldsymbol{r}_n\|$, i.e.,

$$\min_{p \in \mathbb{P}_n, p(0)=1} \|p_n(A)\boldsymbol{r}_0\|.$$

Therefore, for any $p \in \mathbb{P}_n$ and $p(0) = 1$, the residual 2-norm $\|\boldsymbol{r}_n\|$ can be bounded by

$$\begin{aligned}
\|\boldsymbol{r}_n\| &= \|p_n^{(\text{opt})}(A)\boldsymbol{r}_0\| \\
&\leq \|p_n(A)\boldsymbol{r}_0\| \\
&= \|X p_n(\Lambda) X^{-1} \boldsymbol{r}_0\| \\
&\leq \|X\| \|X^{-1}\| \|p_n(\Lambda)\| \|\boldsymbol{r}_0\| \\
&= \kappa(X) \max_{i=1,\dots,N} |p_n(\lambda_i)| \|\boldsymbol{r}_0\|
\end{aligned}$$

Since the above inequality holds for any $p \in \mathbb{P}_n$ and $p(0) = 1$, we have

$$\begin{aligned}
\|\boldsymbol{r}_n\| &\leq \kappa(X) \left(\min_{p \in \mathbb{P}_n, p(0)=1} \max_{i=1,\dots,N} |p(\lambda_i)| \right) \|\boldsymbol{r}_0\| \\
&= \kappa(X) \epsilon_n \|\boldsymbol{r}_0\|,
\end{aligned}$$

which concludes the proof. □

From Proposition 3.4, if the condition number of X is small, the value of ϵ_n depends highly on the speed of convergence. Further, if the condition number of X is small and eigenvalues are well clustered, the GMRES method shows good convergence behavior. If matrix A is symmetric, then the condition number of X is 1. Therefore the convergence of the GMRES method (or the MINRES method) depends only on the distribution of eigenvalues. By using an upper bound of ϵ_n, the convergence behavior can be estimated without using all the eigenvalues. For the details, see an excellent book by Saad [151, pp. 206–207].

Nice features of the GMRES method are that the GMRES method never suffers from breakdown, and the residual 2-norm decreases monotonically and has the optimality regarding the residual 2-norm, unlike Krylov subspace methods based on the BiCG method and the BiCR method. On the other hand, a drawback of the GMRES method is that the computational costs and memory requirement of the GMRES method increase linearly with the number of iterations. For example, if the number of iterations is 10000, then an $N \times 10000$ matrix needs to be stored and the size of the least–squares problem in (3.63) is 10001×10000. To overcome the drawback, a restarted version of the GMRES method is proposed, which is described in Algorithm 3.17.

The idea of the restarted GMRES method denoted by GMRES(m) is that we run the GMRES method until the prescribed maximum iteration number m is reached, and then we restart the GMRES method using the initial guess as the approximate solution produced by the previous GMRES method at the mth iteration step.

Algorithm 3.17 The GMRES(m) method

```
Input: Initial guess $x_0 \in \mathbb{C}^N$ and restart number $m$
Output: $x_m$
1: Run $m$ iterations of the GMRES method (Algorithm 3.16) with $x_0$ and produce $x_m$.
2: while convergence do
3:     Set $x_0 = x_m$.
4:     Run $m$ iterations of the GMRES method (Algorithm 3.16) with $x_0$ and produce $x_m$.
5: end while
```

Consider the restart frequency m of the GMRES(m) method as fixed. This means that we apply the m-step GMRES method to the linear systems repeatedly until convergence. On the other hand, an unfixed restart frequency means that we run the GMRES(m_1) and then run the GMRES(m_2) method, the GMRES(m_3) method, and so on, until convergence. If $m_1 = m_2 = \cdots = m$ then this corresponds to the fixed restart frequency.

Efficient ways for determining m_k for $k = 1, 2, \dots$ are studied in [117] and by some authors, e.g., [14, 137, 176, 211].

Another important development of the GMRES method is *augmentation*. There are two approaches to the augmentation: one is to use previous approximate solutions, e.g., the LGMRES method [15]; another is to use approximate eigenvectors corresponding to small eigenvalues in magnitude, e.g., the GMRES-E method [134]. The basic ideas of the LGMRES method and the GMRES-E method are described next.

The group of m iteration steps (e.g., lines 3 and 4 in Algorithm 3.17) is called a cycle, and the approximate solution of the LGMRES method after the jth cycle is denoted by $\boldsymbol{x}_m^{(j)}$. Let $\boldsymbol{z}^{(k)} := \boldsymbol{x}_m^{(j-1)} - \boldsymbol{x}_m^{(j-2)}$. Then for a given l, the LGMRES method finds an approximate solution in the following way:

$$\boldsymbol{x}_m^{(j)} = \boldsymbol{x}_m^{(j-1)} + q_{m-1}^{(j)}(A)\boldsymbol{r}_m^{(j-1)} + \sum_{k=j-l}^{j-1} \alpha_{j,k}(A)\boldsymbol{z}^{(k)},$$

where $q_{m-1}^{(j)}(A)$ is a polynomial of degree $m-1$, and $\alpha_{j,k}(A)$'s are polynomials of degree k. These polynomials are determined by minimizing the residual 2-norm of $\boldsymbol{x}_m^{(j)}$. For the related algorithms, see, e.g., [13, 103, 104, 106].

The approximate solution of the GMRES-E method after the jth cycle is denoted by $\boldsymbol{x}_m^{(j)}$, and $\boldsymbol{p}_k^{(j)}$ is an approximate eigenvector (Harmonic Ritz vectors) that corresponds to the kth smallest approximate eigenvalue (Harmonic Ritz value) in magnitude. Then, for a given p the GMRES-E method finds an approximate solution in the following way:

$$\boldsymbol{x}_m^{(j)} = \boldsymbol{x}_m^{(j-1)} + q_{m-1}^{(j)}(A)\boldsymbol{r}_m^{(j-1)} + \sum_{k=1}^{p} \beta_{j,k}(A)\boldsymbol{p}_k^{(j-1)},$$

where $q_{m-1}^{(j)}(A)$ is a polynomial of degree $m-1$, and $\beta_{j,k}(A)$'s are polynomials of degree k. These polynomials are determined by minimizing the residual 2-norm of $\boldsymbol{x}_m^{(j)}$. For the related algorithms, see, e.g., [11, 135]. By combining the LGMRES method and the GMRES-E method, the LGMRES-E method using a switching controller is proposed in [31]. Roughly speaking, the switching rule is to choose the LGMRES method while good convergence appears and to choose the GMRES-E method when slow convergence is found.

The augmentation is not only for the GMRES method. In fact, frameworks of augmentation and deflation for Krylov subspace methods including the BiCG method, the BiCR method, and the other important Krylov space methods are developed in [74, 89, 90].

Finally, we provide the result of a numerical example for the GMRES method and the GMRES(m) method. Consider nonsymmetric linear systems (2.18) with parameters $N = 20$, $a_1 = a_2 = a_3 = 1$, $b_1 = b_2 = b_3 = 1$, and $c = 1$. The result is shown in Fig. 3.3, where the stopping criterion used for the GMRES method and the GMRES(m) method is $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq 10^{-10}$, and the horizontal axis is the number of matrix–vector multiplications.

We see from Fig. 3.3 that the residual 2-norms of the GMRES method, the GMRES(10) method, and the GMRES(20) method decrease monotonically, which are theoretically guaranteed. In terms of the number of matrix–vector multiplications, the GMRES method performs better than the restarted GMRES method (GMRES(10), GMRES(20)). As for the GMRES method, the required number of matrix–vector multiplications is the smallest of all, since it finds the best approximate solution over the Krylov subspace. On the other hand, as mentioned before,

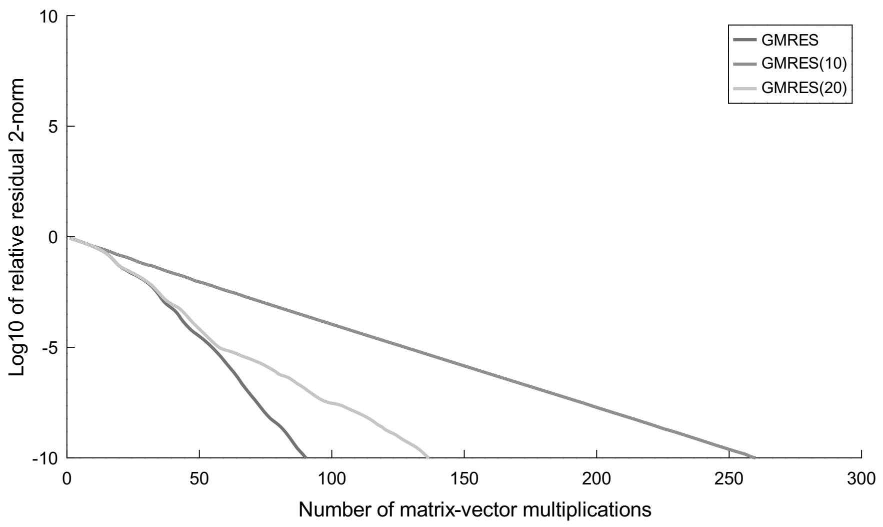


Fig. 3.3 Convergence histories of GMRES, GMRES(10), and GMRES(20)

the computational cost of the GMRES method grows linearly as the number of iterations increases. This means that matrix–vector multiplications may not become the most time-consuming parts when the required number of iterations is very large. In terms of CPU time, the GMRES method required 0.39 sec., GMRES(10) 0.31 sec., GMRES(20) 0.26 sec.[3] Therefore, with respect to CPU time, the GMRES(20) method was the best of all, which balanced the increasing computational costs and the required number of iterations.

3.3.6 The Generalized Conjugate Residual (GCR) Method

The Generalized Conjugate Residual (GCR) method [54] is another Krylov solver based on the Arnoldi process. In exact precision arithmetic, the GCR method produces the same approximate solution as the GMRES method. The algorithm is obtained by the following Arnoldi process with the weight $A^{\mathrm{H}}A$ and $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$:

[3] Codes: GNU Octave version 6.2.0, OS: macOS Moterey version 12.2.1, CPU: Apple M1 pro.

```
set $\boldsymbol{v}_1 = \dfrac{\boldsymbol{r}_0}{\|\boldsymbol{r}_0\|_{A^{\mathrm{H}}A}}$,
for $n = 1, 2, \dots,$ do:
  $h_{i,n} = (\boldsymbol{v}_i, A\boldsymbol{v}_n)_{A^{\mathrm{H}}A}, \quad i = 1, 2, \dots, n,$
  $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \sum_{i=1}^{n} h_{i,n}\boldsymbol{v}_i,$     (3.64)
  $h_{n+1,n} = \|\tilde{\boldsymbol{v}}_{n+1}\|_{A^{\mathrm{H}}A},$
  $\boldsymbol{v}_{n+1} = \dfrac{\tilde{\boldsymbol{v}}_{n+1}}{h_{n+1,n}}.$     (3.65)
end
```

The process generates $A^{\mathrm{H}}A$-orthonormal vectors, i.e.,

$$(\boldsymbol{v}_i, A^{\mathrm{H}}A\boldsymbol{v}_j) = (A\boldsymbol{v}_i, A\boldsymbol{v}_j) = \delta_{i,j}.$$

Let V_n be a matrix whose columns are $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n$. Then, the GCR method generates

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n\boldsymbol{y}_n$$

such that the residual 2-norm $\|\boldsymbol{r}_n\|$ is minimized. The corresponding residual vector can be written as

$$\boldsymbol{r}_n = \boldsymbol{r}_0 - AV_n\boldsymbol{y}_n = \boldsymbol{r}_0 - \sum_{i=1}^{n} y_i A\boldsymbol{v}_i. \tag{3.66}$$

From the above relation, we have

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - y_n A\boldsymbol{v}_n. \tag{3.67}$$

Unknown parameters $\boldsymbol{y}_i$ ($i = 1, 2, \dots, n$) are determined by minimizing the residual 2-norm, i.e., from (3.66) this can be achieved by

$$y_i = (A\boldsymbol{v}_i, \boldsymbol{r}_n), \quad \text{for } i = 1, 2, \dots, n.$$

Since $A\boldsymbol{v}_i$ is an orthonormal vector, we have

$$y_i = (A\boldsymbol{v}_i, \boldsymbol{r}_0), \quad \text{for } i = 1, 2, \dots, n.$$

Here, we define the vector $\boldsymbol{p}_n := -y_n\tilde{\boldsymbol{v}}_{n+1}$. Then, it follows from (3.64) that

$$\boldsymbol{p}_n = -y_n A\boldsymbol{v}_n - \sum_{i=1}^{n} (\boldsymbol{v}_i, -y_n A\boldsymbol{v}_n)_{A^{\mathrm{H}}A} \boldsymbol{v}_i. \tag{3.68}$$

From (3.67) we have $-y_n A\boldsymbol{v}_n = \boldsymbol{r}_n - \boldsymbol{r}_{n-1}$. Substituting this relation into (3.68) yields

$$\begin{aligned}\boldsymbol{p}_n &= \boldsymbol{r}_n - \boldsymbol{r}_{n-1} - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_n - \boldsymbol{r}_{n-1})_{A^{\mathrm{H}}A} \boldsymbol{v}_i \\ &= \boldsymbol{r}_n - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_n)_{A^{\mathrm{H}}A} \boldsymbol{v}_i - \left(\boldsymbol{r}_{n-1} - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_{n-1})_{A^{\mathrm{H}}A} \boldsymbol{v}_i \right).\end{aligned} \tag{3.69}$$

Since $\boldsymbol{r}_{n-1}$ lies in $\mathcal{K}_n(A, \boldsymbol{r}_0)$, it can be expanded as $\boldsymbol{r}_{n-1} = \sum_{i=1}^{n} c_i \boldsymbol{v}_i$. Then,

$$(\boldsymbol{v}_j, \boldsymbol{r}_{n-1})_{A^{\mathrm{H}}A} = \sum_{i=1}^{n} c_i (\boldsymbol{v}_j, \boldsymbol{v}_i) = c_i$$

because $(\boldsymbol{v}_j, \boldsymbol{v}_i) = \delta_{j,i}$. Thus we have

$$\boldsymbol{r}_{n-1} = \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_{n-1})_{A^{\mathrm{H}}A} \boldsymbol{v}_i. \tag{3.70}$$

From (3.69) and (3.70), the vector $\boldsymbol{p}_n$ can be written as

$$\boldsymbol{p}_n = \boldsymbol{r}_n - \sum_{i=1}^{n} (\boldsymbol{v}_i, \boldsymbol{r}_n)_{A^{\mathrm{H}}A} \boldsymbol{v}_i. \tag{3.71}$$

On the other hand, $\boldsymbol{p}_n$ and $\boldsymbol{v}_{n+1}$ are related by

$$\boldsymbol{p}_n = -\frac{y_n}{|y_n|} \|A\boldsymbol{p}_n\| \boldsymbol{v}_{n+1},$$

because it follows from the definition of $\boldsymbol{p}_n$ and (3.65) that

$$\begin{aligned}\boldsymbol{p}_n &= -y_n \tilde{\boldsymbol{v}}_{n+1} \\ &= -y_n \|\tilde{\boldsymbol{v}}_{n+1}\|_{A^{\mathrm{H}}A} \boldsymbol{v}_{n+1} \\ &= -y_n \sqrt{(A\tilde{\boldsymbol{v}}_{n+1}, A\tilde{\boldsymbol{v}}_{n+1})}\ \boldsymbol{v}_{n+1} \\ &= -y_n \sqrt{(-\frac{1}{y_n} A\boldsymbol{p}_n, -\frac{1}{y_n} A\boldsymbol{p}_n)}\ \boldsymbol{v}_{n+1} \\ &= -\frac{y_n}{|y_n|} \sqrt{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}\ \boldsymbol{v}_{n+1}.\end{aligned}$$

Hence, using $\boldsymbol{v}_n = -(|y_{n-1}|/y_{n-1}) \times \boldsymbol{p}_{n-1}/\sqrt{(A\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})}$, we have the following relations:

$$y_n A\boldsymbol{v}_n = (A\boldsymbol{v}_n, \boldsymbol{r}_0)A\boldsymbol{v}_n = \frac{(A\boldsymbol{p}_{n-1}, \boldsymbol{r}_0)}{(A\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})}A\boldsymbol{p}_{n-1},$$
$$(\boldsymbol{v}_i, \boldsymbol{r}_n)_{A^{\mathrm{H}}A}\boldsymbol{v}_i = \frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_n)}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}\boldsymbol{p}_{i-1}.$$

Therefore, substituting the above results into (3.67) and (3.71), we have

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

Now, from the definition of $\boldsymbol{p}_n$, we have $\boldsymbol{p}_{n-1} = c\boldsymbol{v}_n$ for a constant c. Thus $\boldsymbol{p}_{n-1}$ is $A^{\mathrm{H}}A$-orthogonal to $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{n-1}$. From (3.66), 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

$$\alpha_{n-1} = \frac{(A\boldsymbol{p}_{n-1}, \boldsymbol{r}_{n-1})}{(A\boldsymbol{p}_{n-1}, A\boldsymbol{p}_{n-1})}, \tag{3.72}$$
$$\beta_{n-1,i} = -\frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_n)}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}. \tag{3.73}$$

Then, we have

$$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \alpha_{n-1}A\boldsymbol{p}_{n-1}, \tag{3.74}$$
$$\boldsymbol{p}_n = \boldsymbol{r}_n + \sum_{i=1}^{n} \beta_{n-1,i}\boldsymbol{p}_{i-1}. \tag{3.75}$$

From the recurrence relation of the residual vector, we obtain

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \alpha_{n-1}\boldsymbol{p}_{n-1}. \tag{3.76}$$

To reduce the number of matrix–vector multiplications, use the following recurrences:

$$A\boldsymbol{p}_n = A\boldsymbol{r}_n + \sum_{i=1}^{n} \beta_{n-1,i} A\boldsymbol{p}_{i-1}. \tag{3.77}$$

From (3.72)–(3.77), the algorithm of the GCR method is obtained in Algorithm 3.18.

Algorithm 3.18 The GCR method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{p}_0 = \boldsymbol{r}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, 2, \dots,$ until convergence do
2:     $\alpha_n = \frac{(A\boldsymbol{p}_n, \boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}$
3:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
4:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
5:     $\beta_{n,i} = -\frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_{n+1})}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}, \quad 1 \le i \le n+1$
6:     $\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i} \boldsymbol{p}_{i-1}$
7:     $(A\boldsymbol{p}_{n+1} = A\boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i} A\boldsymbol{p}_{i-1})$
8: end for
```

The properties of the GCR method are given next.

Proposition 3.5 *The GCR iterates hold the following properties:*

(G1) $(A\boldsymbol{p}_i, A\boldsymbol{p}_j) = 0, \quad i \neq j,$
(G2) $(\boldsymbol{r}_i, A\boldsymbol{p}_j) = 0, \quad i > j,$
(G3) $(\boldsymbol{r}_i, A\boldsymbol{p}_i) = (\boldsymbol{r}_i, A\boldsymbol{r}_i),$
(G4) $(\boldsymbol{r}_i, A\boldsymbol{r}_j) = 0, \quad i > j,$
(G5) $(\boldsymbol{r}_i, A\boldsymbol{p}_i) = (\boldsymbol{r}_0, A\boldsymbol{p}_i), \quad i \geq j.$

For the proof, see [54, Theorem 3.1].

From the above properties, we see that the previous derivation is based on the property (G1). In exact precision arithmetic, the GCR method, as well as the GMRES method, converges within at most N iterations. However, the GCR method has the same shortcoming as the GMRES method in that the computational work and the required memory increase with the number of iterations. Hence, a restarted version of GCR which is referred to as the GCR(k) method was proposed in [54]. The algorithm is described in Algorithm 3.19.

Another alternative is to keep 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 the Orthomin method proposed by Vinsome [201]. Later the Orthomin method was referred to as the Orthomin(k) method in [54]. The algorithm of the Orthomin(k) method is described in Algorithm 3.20.

Algorithm 3.19 The GCR(k) method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{p}_0 = \boldsymbol{r}_0$
Output: $\boldsymbol{x}_{k+1}$
 1: for $n = 0, 1, \dots, k$ until convergence do
 2:     $\alpha_n = \frac{(A\boldsymbol{p}_n, \boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}$
 3:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
 4:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
 5:     $\beta_{n,i} = -\frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_{n+1})}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}, \quad 1 \le i \le n+1$
 6:     $\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i}\boldsymbol{p}_{i-1}$
 7:     $(A\boldsymbol{p}_{n+1} = A\boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i}A\boldsymbol{p}_{i-1})$
 8: end for
 9: $\boldsymbol{x}_0 = \boldsymbol{x}_{k+1}$
10: repeat
```

Algorithm 3.20 The Orthomin(k) method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{p}_0 = \boldsymbol{r}_0$
Output: $\boldsymbol{x}_n$
 1: for $n = 0, 1, \dots,$ until $\|\boldsymbol{r}_n\| \le \epsilon \|\boldsymbol{b}\|$ do
 2:     $\alpha_n = \frac{(A\boldsymbol{p}_n, \boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}$
 3:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
 4:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
 5:     $\beta_{n,i} = -\frac{(A\boldsymbol{p}_{i-1}, A\boldsymbol{r}_{n+1})}{(A\boldsymbol{p}_{i-1}, A\boldsymbol{p}_{i-1})}, \quad n-k+2 \le i \le n+1$
 6:     $\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{i=1}^{\min\{k,n+1\}} \beta_{n,i}\boldsymbol{p}_{i-1}$
 7:     $(A\boldsymbol{p}_{n+1} = A\boldsymbol{r}_{n+1} + \sum_{i=1}^{n+1} \beta_{n,i}A\boldsymbol{p}_{i-1})$
 8: end for
```

The convergence analysis of the GCR method, the GCR(k) method, and the Orthomin(k) method is given in Proposition 3.6.

Proposition 3.6 *Let M be the symmetric part of $A \in \mathbb{R}^{N\times N}$ and let R be the skew-symmetric part of A, i.e., $M = (A + A^\top)/2, R = (A - A)^\top/2$. If M is positive definite, then $\{\boldsymbol{r}_n\}$ that are the sequence of residuals generated by the GCR method, the GCR(k) method, or the Orthomin(k) method satisfy*

$$\|\boldsymbol{r}_n\| \le \left(1 - \frac{\lambda_{\min}(M)^2}{\lambda_{\max}(A^\top A)}\right)^{\frac{n}{2}} \|\boldsymbol{r}_0\|$$

and

$$\|\boldsymbol{r}_n\| \le \left(1 - \frac{\lambda_{\min}(M)^2}{\lambda_{\min}(M)\lambda_{\max}(M) + \rho(R)^2}\right)^{\frac{n}{2}} \|\boldsymbol{r}_0\|,$$

where $\lambda_{\min}(M)$ and $\lambda_{\max}(M)$ are the minimum and maximum eigenvalues of the symmetric positive definite matrix M, and $\rho(R)$ is the spectral radius of R.

See (1.19) for the spectral radius. For the proof, see [54, Theorem 4.4].

We can see from Proposition 3.6 that if A is close to I, then the three methods converge fast since $\lambda_{\min}(M) \approx \lambda_{\max}(M) \approx 1$ and $R \approx O$.

3.3.7 The Full Orthogonalization Method (FOM)

The FOM [149] is a generalization of the CG method. As well as the CG method, the FOM finds approximate solutions such that

$$\begin{aligned} \boldsymbol{x}_n &= \boldsymbol{x}_0 + \boldsymbol{z}_n, \quad \boldsymbol{z}_n \in \mathcal{K}_n(A, \boldsymbol{r}_0), \\ \boldsymbol{r}_n &\perp \mathcal{K}_n(A, \boldsymbol{r}_0). \end{aligned}$$

The FOM uses the Arnoldi method in Sect. 1.9.1 to produce the orthonormal basis vectors of $\mathcal{K}_n(A, \boldsymbol{r}_0)$. Thus the residual vector $\boldsymbol{r}_n$ has the same form as (3.62), i.e.,

$$\boldsymbol{r}_n = V_{n+1}(\beta \boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}_n). \tag{3.78}$$

The difference between the GMRES method and the FOM is that the GMRES method finds $\boldsymbol{y}_n$ such that the 2-norm of the residual vector $\|\boldsymbol{r}_n\|$ is minimized, and the FOM finds $\boldsymbol{y}_n$ such that the residual vector is orthogonal to the n dimensional Krylov subspace $\boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$. Let $\boldsymbol{v}_i$ be the ith column of V_{n+1}. Then

$$\begin{aligned} V_n^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0} &\Leftrightarrow \boldsymbol{v}_i^{\mathrm{H}}\boldsymbol{r}_n = 0 \text{ for } i = 1, 2, \dots, n \\ &\Leftrightarrow \boldsymbol{r}_n^{\mathrm{H}}(c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_n\boldsymbol{v}_n) = 0 \text{ for all } c_i \in \mathbb{C} \\ &\Leftrightarrow \boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0). \end{aligned}$$

Note that the statement $\boldsymbol{v}_i^{\mathrm{H}}\boldsymbol{r}_n = 0 \Leftarrow \boldsymbol{r}_n^{\mathrm{H}}(c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_n\boldsymbol{v}_n) = 0$ can be shown by setting $c_i = 1$ and $c_j = 0$ for $i \neq j$. From this, finding $\boldsymbol{y}_n$ such that $\boldsymbol{r}_n \perp \mathcal{K}_n(A, \boldsymbol{r}_0)$ is equivalent to determining $\boldsymbol{y}_n$ such that $V_n^{\mathrm{H}}\boldsymbol{r}_n = \boldsymbol{0}$. Then, from (3.78) it follows that

$$\boldsymbol{0} = V_n^{\mathrm{H}}\boldsymbol{r}_n = V_n^{\mathrm{H}}V_{n+1}(\beta\boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}_n) = \beta\boldsymbol{e}_1 - H_{n,n}\boldsymbol{y}_n.$$

Thus $\boldsymbol{y}_n$ can be obtained by solving the following linear systems:

$$H_{n,n}\boldsymbol{y}_n = \beta\boldsymbol{e}_1.$$

Since $H_{n,n}$ is a Hessenberg matrix, the linear systems can be solved by the LU decomposition with the computational cost of $O(n^2)$. After the advent of the GMRES method, the FOM became less attractive because, unlike the GMRES method, the FOM may suffer from breakdown due to using the LU decomposition without pivoting. On the other hand, the FOM is still attractive to use for solving shifted linear systems in Chap. 4, when considering the restart.

3.3.8 Product-Type Krylov Subspace Methods

As seen in Sect. 3.3.1, the BiCG method requires A^{H} for solving $A\boldsymbol{x} = \boldsymbol{b}$. P. Sonneveld found that squaring the BiCG polynomials leads to an algorithm without using the information of A^{H}, and the resulting algorithm is known as the CGS method [173]. Though the convergence depends on linear systems, the CGS method is much faster than the BiCG method when the BiCG method shows smooth convergence behavior. However, the CGS method sometimes has much irregular convergence and may suffer from loss of accuracy of the approximate solution or may diverge. H. A. van der Vorst observed that squaring the BiCG polynomials can be replaced by the multiplication of degree-one minimum residual polynomials and the BiCG polynomial, leading to the BiCGSTAB method [195], which tends to have smoother convergence behavior than the CGS method. M. H. Gutknecht proposed the multiplication of degree-two minimum residual polynomials and the BiCG polynomials, which is known as the BiCGSTAB2 method [87]. G. L. G. Sleijpen and D. R. Fokkema proposed the multiplication of degree-ℓ minimal residual polynomials and the BiCG polynomials, which is known as the BiCGSTAB(ℓ) method [162]. S.-L. Zhang constructed a framework that includes the CGS method, the BiCGSTAB method, and the BiCGSTAB2 method. In the framework, the GPBiCG method was proposed in [212]. These methods are referred to as product-type methods (based on the Bi-CG method) first named by S.-L. Zhang.

In this section, Zhang's framework is explained, based on Zhang's book in [72]. As seen in (3.18), the residual vector of the CG method can be written as the multiplication of Lanczos polynomials and the initial residual vector $\boldsymbol{r}_0$. Similarly, the residual vector of the BiCG method can also be written as follows:

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

where

$$R_0(\lambda) = 1, \quad P_0(\lambda) = 1, \tag{3.79}$$

$$R_n(\lambda) = R_{n-1}(\lambda) - \alpha_{n-1}\lambda P_{n-1}(\lambda), \tag{3.80}$$

$$P_n(\lambda) = R_n(\lambda) + \beta_{n-1}P_{n-1}(\lambda) \quad n = 1, 2, \ldots \tag{3.81}$$

The differences between the (matrix) polynomial representation of the CG method and that of the BiCG method are α_i and β_i of the Lanczos polynomials. The nth residual vector of product-type methods based on the BiCG method is defined by the product of n degree polynomials and the nth residual vector of the BiCG method as follows:

$$\boldsymbol{r}_n = H_n(A)\boldsymbol{r}_n^{\mathrm{BiCG}} = H_n(A)R_n(A)\boldsymbol{r}_0. \tag{3.82}$$

3.3.8.1 Restructuring of Residual Polynomials

S.-L. Zhang proposed H_n in (3.82) as the following three-term recurrence relations:

$$H_0(\lambda) := 1, \tag{3.83}$$
$$H_1(\lambda) := (1 - \zeta_0\lambda)H_0(\lambda), \tag{3.84}$$
$$H_n(\lambda) := (1 + \eta_{n-1} - \zeta_{n-1}\lambda)H_{n-1}(\lambda) - \eta_{n-1}H_{n-2}(\lambda), \quad n = 2, 3, \dots \tag{3.85}$$

We see that H_n is similar to the Lanczos polynomials (3.19)–(3.21).

In the following, H_n is rewritten as the form of coupled two-term recurrence relation. We introduce G_{n-1} as

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

and we rewrite (3.85) as

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

Then $H_n(\lambda)$ in (3.83)–(3.85) can be rewritten as

$$H_0(\lambda) = 1, \quad G_0(\lambda) = \zeta_0, \tag{3.86}$$
$$H_n(\lambda) = H_{n-1}(\lambda) - \lambda G_{n-1}(\lambda), \tag{3.87}$$
$$G_n(\lambda) = \zeta_n H_n(\lambda) + \eta_n G_{n-1}(\lambda), \quad n = 1, 2, \dots \tag{3.88}$$

3.3.8.2 Recurrence Formulas for the Iterates of the Product-Type Methods

Since recurrence relations of R_n, P_n, H_n, G_n are already given in (3.79)–(3.81) and in (3.86)–(3.88), the residual vector $\boldsymbol{r}_{n+1} = H_{n+1}(A)R_{n+1}(A)\boldsymbol{r}_0$ can be computed by the previous residual vector $\boldsymbol{r}_n = H_n(A)R_n(A)\boldsymbol{r}_0$ using the recurrence relations. In fact, $H_{n+1}R_{n+1}$ can be expanded as follows:

$$H_{n+1}R_{n+1} = H_nR_{n+1} - \eta_n\lambda G_{n-1}R_{n+1} - \zeta_n\lambda H_nR_{n+1} \tag{3.89}$$
$$= H_nR_n - \alpha_n\lambda H_nP_n - \lambda G_nR_{n+1}, \tag{3.90}$$
$$H_nR_{n+1} = H_nR_n - \alpha_n\lambda H_nP_n, \tag{3.91}$$
$$\lambda G_nR_{n+2} = H_nR_{n+1} - H_{n+1}R_{n+1} - \alpha_{n+1}\lambda H_nP_{n+1} + \alpha_{n+1}\lambda H_{n+1}P_{n+1}, \tag{3.92}$$
$$H_{n+1}P_{n+1} = H_{n+1}R_{n+1} + \beta_nH_nP_n - \beta_n\lambda G_nP_n, \tag{3.93}$$
$$\lambda H_nP_{n+1} = \lambda H_nR_{n+1} + \beta_n\lambda H_nP_n, \tag{3.94}$$
$$\lambda G_nP_n = \zeta_n\lambda H_nP_n + \eta_n(H_{n-1}R_n - H_nR_n + \beta_{n-1}\lambda G_{n-1}P_{n-1}), \tag{3.95}$$
$$G_nR_{n+1} = \zeta_nH_nR_n + \eta_nG_{n-1}R_n - \alpha_n\lambda G_nP_n. \tag{3.96}$$

Let us define new auxiliary vectors as

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

Then, from (3.89)–(3.96) we have the following recurrence formulas:

$$\begin{aligned}
\boldsymbol{r}_{n+1} &= \boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n A\boldsymbol{t}_n && (3.97)\\
&= \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n - A\boldsymbol{z}_n, && (3.98)\\
\boldsymbol{t}_n &= \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n, && (3.99)\\
\boldsymbol{y}_{n+1} &= \boldsymbol{t}_n - \boldsymbol{r}_{n+1} - \alpha_{n+1}\boldsymbol{w}_n + \alpha_{n+1}A\boldsymbol{p}_{n+1}, && (3.100)\\
\boldsymbol{p}_{n+1} &= \boldsymbol{r}_{n+1} + \beta_n(\boldsymbol{p}_n - \boldsymbol{u}_n), && (3.101)\\
\boldsymbol{w}_n &= A\boldsymbol{t}_n + \beta_n A\boldsymbol{p}_n, && (3.102)\\
\boldsymbol{u}_n &= \zeta_n A\boldsymbol{p}_n + \eta_n(\boldsymbol{t}_{n-1} - \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{u}_{n-1}), && (3.103)\\
\boldsymbol{z}_n &= \zeta_n \boldsymbol{r}_n + \eta_n \boldsymbol{z}_{n-1} - \alpha_n \boldsymbol{u}_n. && (3.104)
\end{aligned}$$

Two recurrence relations, (3.97) and (3.98), are described for the residual vector and the approximate solution. The first recurrence relation (3.97) will be used to determine the two parameters ζ_n and η_n, and the second recurrence relation (3.98) can be used to obtain the approximate solution as follows:

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

3.3.8.3 Computations for α_n and 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

$$\begin{aligned}
(\boldsymbol{r}_0^*, \boldsymbol{r}_n) &= (H_n(A^{\text{H}})\boldsymbol{r}_0^*, \boldsymbol{r}_n^{\text{BiCG}}) = \left((-1)^n \prod_{i=0}^{n-1} \zeta_i\right) ((A^{\text{H}})^n \boldsymbol{r}_0^*, \boldsymbol{r}_n^{\text{BiCG}}),\\
(\boldsymbol{r}_0^*, A\boldsymbol{p}_n) &= (H_n(A^{\text{H}})\boldsymbol{r}_0^*, A\boldsymbol{p}_n^{\text{BiCG}}) = \left((-1)^n \prod_{i=0}^{n-1} \zeta_i\right) ((A^{\text{H}})^n \boldsymbol{r}_0^*, A\boldsymbol{p}_n^{\text{BiCG}}).
\end{aligned}$$

Thus, from (3.59) and (3.60) it follows that

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

Table 3.1 Choices of ζ_n and η_n for the product-type methods

CGS	$\zeta_n = \alpha_n, \quad \eta_n = \dfrac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n$
BiCGSTAB	$\zeta_n = \underset{\zeta_n \in \mathbb{C}}{\arg\min} \|\boldsymbol{r}_{n+1}\|, \quad \eta_n = 0$
GPBiCG	$\{\zeta_n, \eta_n\} = \underset{\zeta_n, \eta_n \in \mathbb{C}}{\arg\min} \|\boldsymbol{r}_{n+1}\|$
BiCGSTAB2	BiCGSTAB parameters at even iterations
	GPBiCG parameters at odd iterations

3.3.8.4 Implementation Details

Krylov subspace methods, such as the CGS method, the BiCGSTAB method, the BiCGSTAB2 method, and the GPBiCG method, can be derived from Zhang's framework by determining the two parameters ζ_n and η_n. The relations among these Krylov subspace methods and the parameters are listed in Table 3.1.

3.3.8.5 The Choice for the CGS Method

Setting $\zeta_n = \alpha_n$ and $\eta_n = (\beta_{n-1}/\alpha_{n-1})\alpha_n$ in recurrence relations (3.87) and (3.88) yields the CGS method, i.e., the polynomials H_n and G_n become the Lanczos polynomials:

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

This fact leads to the relation $\boldsymbol{p}_n - \boldsymbol{u}_n = \boldsymbol{z}_n/\alpha_n$, since from the definitions of auxiliary vectors we have

$$\begin{aligned} \boldsymbol{p}_n - \boldsymbol{u}_n &= H_n(A)P_n(A)\boldsymbol{r}_0 - AG_n(A)P_n(A)\boldsymbol{r}_0 \\ &= R_n(A)P_n(A)\boldsymbol{r}_0 - \alpha_n A P_n(A)P_n(A)\boldsymbol{r}_0 \\ &= (R_n(A) - \alpha_n A P_n(A))P_n(A)\boldsymbol{r}_0 \\ &= R_{n+1}(A)P_n(A)\boldsymbol{r}_0 \\ &= P_n(A)R_{n+1}(A)\boldsymbol{r}_0 \\ &= P_n(A)\boldsymbol{r}_{n+1}^{\text{BiCG}} \\ &= \frac{1}{\alpha_n} \times G_n(A)\boldsymbol{r}_{n+1}^{\text{BiCG}} \\ &= \frac{1}{\alpha_n} \times \boldsymbol{z}_n. \end{aligned}$$

Notice that $\boldsymbol{t}_{n-1} - \boldsymbol{r}_n = A\boldsymbol{z}_{n-1}$ from (3.98) and (3.99). We use the recurrence formula (3.98) to update $\boldsymbol{r}_{n+1}$, and then the auxiliary vectors $\boldsymbol{t}_n, \boldsymbol{y}_n$, and $\boldsymbol{w}_n$ are not required in the recurrence formulas (3.97)–(3.104). Now we introduce the following auxiliary vectors:

$$\tilde{\boldsymbol{u}}_n := A^{-1}\boldsymbol{u}_n/\alpha_n, \quad \tilde{\boldsymbol{z}}_n := \boldsymbol{z}_n/\alpha_n,$$

instead of $\boldsymbol{u}_n$ and $\boldsymbol{z}_n$. By using the relation $(\boldsymbol{r}_0^*, A\boldsymbol{p}_n) = (\boldsymbol{r}_0^*, \boldsymbol{u}_n/\alpha_n)$, the CGS method can be obtained and the algorithm is described in Algorithm 3.21. Here $\tilde{\boldsymbol{u}}_n$ and $\tilde{\boldsymbol{z}}_n$ were rewritten as $\boldsymbol{u}_n$ and $\boldsymbol{z}_n$.

Algorithm 3.21 The CGS method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0$, $\boldsymbol{u}_{-1} = \boldsymbol{z}_{-1} = \boldsymbol{0}$, $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Input: Choose $\boldsymbol{r}_0^* \in \mathbb{C}^N$, e.g., $\boldsymbol{r}_0^* = \boldsymbol{r}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, \ldots$ do
2:    $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{z}_{n-1}$
3:    $\boldsymbol{u}_n = \boldsymbol{p}_n + \beta_{n-1}(\boldsymbol{z}_{n-1} + \beta_{n-1}\boldsymbol{u}_{n-1})$
4:    $\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{u}_n)}$
5:    $\boldsymbol{z}_n = \boldsymbol{p}_n - \alpha_n A\boldsymbol{u}_n$
6:    $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n(\boldsymbol{p}_n + \boldsymbol{z}_n)$
7:    $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A(\boldsymbol{p}_n + \boldsymbol{z}_n)$
8:    $\beta_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}$
9: end for
```

The residual vector of the CGS method is written as $\boldsymbol{r}_{n+1} = R_{n+1}(A)R_{n+1}(A)\boldsymbol{r}_0$. The CGS method is short for the (Bi)CG squared method, which comes from the term $R_{n+1}(A)R_{n+1}(A)$. From this, the CGS method is expected to converge as twice as fast as the BiCG method. On the other hand, when the BiCG method shows irregular convergence behavior as is often the case, the CGS method gives much irregular convergence behavior, leading to much less accurate approximate solutions or no convergence.

3.3.8.6 The Choice for the BiCGSTAB Method

If we choose $\eta_n = 0$ and ζ_n such that the 2-norm of the residual vector $\|\boldsymbol{r}_{n+1}\| = \|\boldsymbol{t}_n - \zeta_n A\boldsymbol{t}_n\|$ is minimized, then the BiCGSTAB method is obtained. To achieve the minimization, we choose ζ_n such that the following property holds:

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

or equivalently it follows from (1.34) that

$$\zeta_n = \frac{\boldsymbol{t}_n^{\mathrm{H}} A \boldsymbol{t}_n}{(A\boldsymbol{t}_n)^{\mathrm{H}}(A\boldsymbol{t}_n)}.$$

Since $\eta_n = 0$, we obtain

$$\boldsymbol{u}_n = \zeta_n A \boldsymbol{p}_n, \quad \boldsymbol{z}_n = \zeta_n \boldsymbol{t}_n.$$

The algorithm of the BiCGSTAB method is described in Algorithm 3.22.

The BiCGSTAB method is short for the BiCG stabilized method. Below is an explanation of the meaning of the stabilization. Since $\eta_n = 0$, the recurrence relations in (3.87) and (3.88) reduce to

$$H_n(\lambda) = H_{n-1}(\lambda) - \zeta_{n-1}\lambda H_{n-1}(\lambda) = (1 - \zeta_{n-1}\lambda)H_{n-1}(\lambda).$$

This means that the residual vector of the BiCGSTAB method is given by

$$\boldsymbol{r}_{n+1} = H_{n+1}(A)R_{n+1}(A)\boldsymbol{r}_0 = (I - \zeta_n A)\boldsymbol{r}_n^{\mathrm{BiCG}}.$$

This implies that since ζ_n is chosen such that $\|\boldsymbol{r}_{n+1}\|$ is minimized, the residual vector of the BiCG method is expected to be stabilized, i.e., the BiCGSTAB method is expected to have smoother convergence behavior than the BiCG method.

Algorithm 3.22 The BiCGSTAB method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Input: Choose $\boldsymbol{r}_0^* \in \mathbb{C}^N$, e.g., $\boldsymbol{r}_0^* = \boldsymbol{r}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, \dots$ do
2:     $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \zeta_{n-1}A\boldsymbol{p}_{n-1})$
3:     $\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)}$
4:     $\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
5:     $\zeta_n = \frac{(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)}$
6:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n + \zeta_n\boldsymbol{t}_n$
7:     $\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \zeta_n A\boldsymbol{t}_n$
8:     $\beta_n = \frac{\alpha_n}{\zeta_n} \times \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}$
9: end for
```

3.3.8.7 The Choice for the GPBiCG Method

For the GPBiCG method, two parameters η_n and ζ_n are chosen by minimizing the 2-norm of the residual vector $\|\boldsymbol{r}_{n+1}\| = \|\boldsymbol{t}_n - \eta_n\boldsymbol{y}_n - \zeta_n A\boldsymbol{t}_n\|$. The following orthogonal condition enables us to achieve the minimization:

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

or equivalently it follows from (1.31) and (1.32) with setting $\boldsymbol{d} = \boldsymbol{t}_n, M = [\boldsymbol{y}_n, A\boldsymbol{t}_n]$, $\boldsymbol{x} = [\eta_n, \zeta_n]^\top$ that we have

$$\zeta_n = \frac{(\boldsymbol{y}_n, \boldsymbol{y}_n)(A\boldsymbol{t}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, \boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)},$$
$$\eta_n = \frac{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)}.$$

The algorithm of the GPBiCG method is described in Algorithm 3.23. Note the GPBiCG method uses two parameters ζ_n and η_n for the minimization of the residual 2-norm, and the BiCGSTAB method uses one parameter ζ_n for the minimization of the residual 2-norm and $\eta_n = 0$. From this, the GPBiCG method is expected to give smoother convergence behavior than the BiCGSTAB method.

Algorithm 3.23 The GPBiCG method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N, \beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{t}_{-1} = \boldsymbol{u}_{-1} = \boldsymbol{w}_{-1} = \boldsymbol{z}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Input: Choose $\boldsymbol{r}_0^* \in \mathbb{C}^N$, e.g., $\boldsymbol{r}_0^* = \boldsymbol{r}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, \ldots$ do
2:     $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \boldsymbol{u}_{n-1})$
3:     $\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)}$
4:     $\boldsymbol{y}_n = \boldsymbol{t}_{n-1} - \boldsymbol{r}_n - \alpha_n \boldsymbol{w}_{n-1} + \alpha_n A\boldsymbol{p}_n$
5:     $\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
6:     $\zeta_n = \frac{(\boldsymbol{y}_n, \boldsymbol{y}_n)(A\boldsymbol{t}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, \boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)}$
7:     $\eta_n = \frac{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, A\boldsymbol{t}_n)(A\boldsymbol{t}_n, \boldsymbol{y}_n)}$
8:     if $n = 0$ then
9:         $\zeta_n = \frac{(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)}, \ \eta_n = 0$
10:    end if
11:    $\boldsymbol{u}_n = \zeta_n A\boldsymbol{p}_n + \eta_n(\boldsymbol{t}_{n-1} - \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{u}_{n-1})$
12:    $\boldsymbol{z}_n = \zeta_n \boldsymbol{r}_n + \eta_n \boldsymbol{z}_{n-1} - \alpha_n \boldsymbol{u}_n$
13:    $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n + \boldsymbol{z}_n$
14:    $\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n A\boldsymbol{t}_n$
15:    $\beta_n = \frac{\alpha_n}{\zeta_n} \times \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}$
16:    $\boldsymbol{w}_n = A\boldsymbol{t}_n + \beta_n A\boldsymbol{p}_n$
17: end for
```

From Zhang's framework, the GPBiCG(m, ℓ) method was proposed by Fujino [71]. A unified generalization of the GPBiCG method and the BiCGSTAB(ℓ) method was proposed by K. Aihara [4], which is referred to as the GPBiCGstab(L) method.

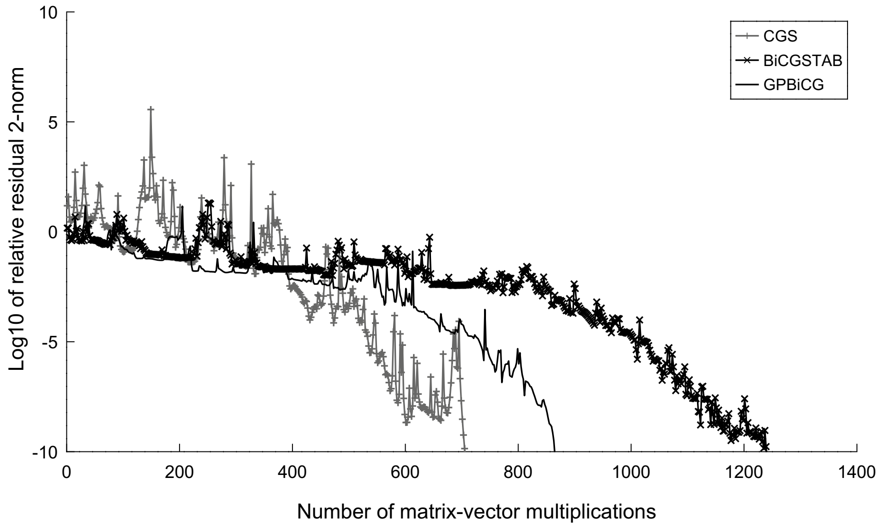


Fig. 3.4 Convergence histories of CGS, BiCGSTAB, and GPBiCG for $N = 20$, $c = 300$

3.3.8.8 Numerical Examples

In this subsection, some illustrative numerical examples are shown. We consider nonsymmetric linear systems (2.18) with parameters $N = 20$, $a_1 = a_2 = a_3 = 1$, $b_1 = b_2 = b_3 = 1$, and two different parameters $c = 300$ and $c = 900$. The stopping criterion used for all the Krylov subspace methods is $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq 10^{-10}$.

The convergence histories of the CGS method, the BiCGSTAB method, and the GPBiCG method are shown in Fig. 3.4 for $c = 300$. The horizontal axis is not the number of iterations but the number of matrix–vector multiplications. (The CGS method, the BiCGSTAB method, and the GPBiCG method require two matrix–vector multiplications per iteration.)

From Fig. 3.4, the GPBiCG method converges faster than BiCGSTAB method, and the CGS method is the best in terms of the number of matrix–vector multiplications, or equivalently the number of iterations. On the other hand, the CGS method shows irregular convergence behavior, which may lead to a loss of accuracy. The GPBiCG method and the BiCGSTAB method show relatively smooth convergence behavior, which may be caused by local minimization of the residual 2-norms. In terms of accuracy, the log10 of the true relative residual 2-norms for the CGS method, the BiCGSTAB method, and the GPBiCG method are –9.79, –10.50, –10.17 respectively. From which, all the methods produced sufficiently accurate approximate solutions.

Next, the convergence histories for $c = 900$ are shown in Fig. 3.5. We see from Fig. 3.5 that the CGS method shows much irregular convergence behavior and the slowest convergence, and the GPBiCG method is the best in terms of the number of matrix–vector multiplications. In terms of accuracy, the log10 of the true relative

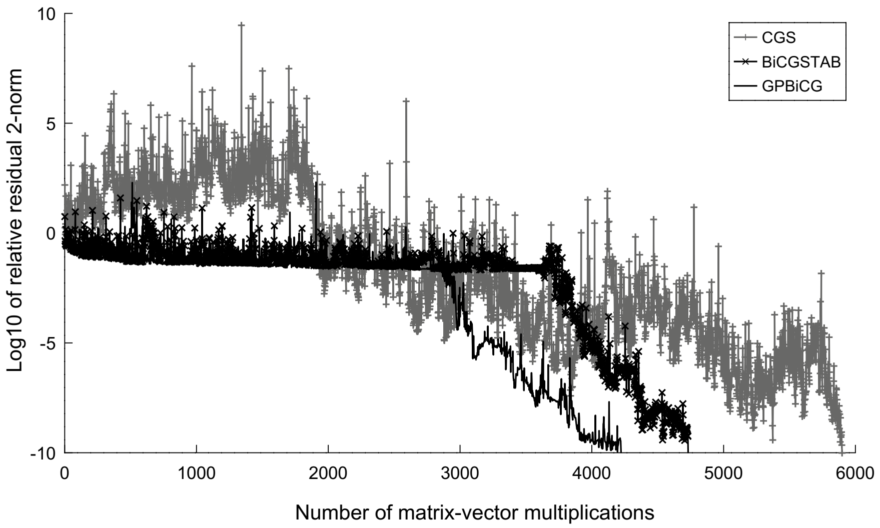


Fig. 3.5 Convergence histories of CGS, BiCGSTAB, and GPBiCG for $N = 20$, $c = 900$

residual 2-norms for the CGS method, the BiCGSTAB method, and the GPBiCG method are –6.37, –10.03, –9.79 respectively. This means that the CGS method suffers from a loss of accuracy of about four digits. This loss of accuracy is known as a *residual gap*. Therefore, it is strongly recommended to check the true relative residual 2-norm, i.e., $\|\boldsymbol{b} - A\boldsymbol{x}_n\|/\|\boldsymbol{b}\|$ (not $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\|$!), after satisfying the stopping criterion.

3.3.8.9 Other Related Methods

The QMR method in Sect. 3.3.4 requires the information of A^{H}, as well as the BiCG method. Similar to the idea of the CGS method, R. W. Freund proposed a squared variant of the QMR method referred to as the TFQMR (transpose-free QMR) method [63], which tends to give smooth convergence behavior. Based on the BiCGSTAB method and the idea of the QMR method, the QMRCGSTAB method was proposed in [34].

The BiCR method in Sect. 3.3.3 may give a smoother convergence behavior than the BiCG method when the coefficient matrix A is close to Hermitian, since the BiCR method with the choice $\boldsymbol{r}_0^* = \boldsymbol{r}_0$ becomes the CR method when A is Hermitian and the residual 2-norm of the CR method monotonically decreases. Thus it is natural to replace the product-type Krylov subspace methods based on the BiCG method with those based on the BiCR method. In fact, the product-type methods based on the BiCR method can be constructed, and the resulting algorithms (CRS, BiCRSTAB, GPBiCR) are derived in [165]. K. Abe and G. L. G. Sleijpen independently proposed product-type methods based on the BiCR method, see [2].

Y.-F. Jing et al. proposed an approach similar to the BiCR method called the BiCOR method [114], and the product-type variants were proposed in [32].

3.3.9 Induced Dimension Reduction (IDR(s)) Method

The Induced Dimension Reduction (IDR) method was developed by Wesseling and Sonneveld around 1979 [205]. The IDR method is mathematically equivalent to the BiCGSTAB method, and the BiCGSTAB method can be regarded as a stabilized variant of the IDR method. It is written in the acknowledgements in [195] that Sonneveld suggested that van der Vorst reconsiders Sonneveld's IDR.

Though the IDR method was not given further attention, about 30 years later, Sonneveld and van Gijzen proposed an innovative algorithm called the IDR(s) method [174]. In this section, the theory and the derivation of the IDR(s) method are described. The derivation in this section looks to be redundant, but it makes it easier to understand the principle of the IDR(s) method.

Theorem 3.4 (The IDR theorem [174] [205, p. 550]) *Let A be any matrix in $\mathbb{C}^{N\times N}$, let $\boldsymbol{v}_0$ be any nonzero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the complete Krylov space, i.e., $\mathcal{G}_0 := \mathcal{K}_N(A, \boldsymbol{v}_0)$. Let $\mathcal{S}$ denote any (proper) subspace of $\mathbb{C}^N$ such that $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace,*[4] *and define the sequences $\mathcal{G}_j$, $j = 1, 2, \dots$ as*

$$\mathcal{G}_j = (I - \omega_j A)(\mathcal{G}_{j-1} \cap \mathcal{S}), \tag{3.105}$$

where the ω_j's are nonzero scalars. Then:

(1) $\mathcal{G}_j \subset \mathcal{G}_{j-1}$ *for all* $j > 0$.
(2) $\mathcal{G}_j = \{\mathbf{0}\}$ *for some* $j \le N$.

Proof The proof is based on [174, Theorem 2.1].

The first statement is shown by induction. First, we show $\mathcal{G}_1 \subset \mathcal{G}_0$. From $\mathcal{G}_0 = \mathcal{K}_N(A, \boldsymbol{v}_0)$, it follows that $(I - \omega_1 A)\mathcal{G}_0 \subset \mathcal{G}_0$, because for a given $\boldsymbol{v} \in (I - \omega_1 A)\mathcal{G}_0$ we have

$$\begin{aligned}
\boldsymbol{v} &= (I - \omega_1 A)(c_0\boldsymbol{v}_0 + c_1 A\boldsymbol{v}_0 + \dots + c_{N-1}A^{N-1}\boldsymbol{v}_0) \\
&= c_0(I - \omega_1 A)\boldsymbol{v}_0 + c_1(I - \omega_1 A)A\boldsymbol{v}_0 + \dots + c_{N-1}(I - \omega_1 A)A^{N-1}\boldsymbol{v}_0 \\
&= d_0\boldsymbol{v}_0 + d_1 A\boldsymbol{v}_0 + \dots + d_{N-1}A^{N-1}\boldsymbol{v}_0 + d_N A^N \boldsymbol{v}_0 \\
&= \tilde{d}_0\boldsymbol{v}_0 + \tilde{d}_1 A\boldsymbol{v}_0 + \dots + \tilde{d}_{N-1}A^{N-1}\boldsymbol{v}_0 \\
&\in \mathcal{G}_0.
\end{aligned}$$

[4] This means that $\mathcal{S} \cap \mathcal{G}_0$ does not contain any eigenvector of A, and the trivial invariant subspace is $\{\mathbf{0}\}$.

Thus we have $\mathcal{G}_1 \subset \mathcal{G}_0$ because

$$\mathcal{G}_1 = (I - \omega_1 A)(\mathcal{G}_0 \cap \mathcal{S}) \subset (I - \omega_1 A)\mathcal{G}_0 \subset \mathcal{G}_0.$$

Next we show $\mathcal{G}_{j+1} \subset \mathcal{G}_j$ (i.e., $\boldsymbol{x} \in \mathcal{G}_{j+1} \Rightarrow \boldsymbol{x} \in \mathcal{G}_j$) for all $j > 0$ using the assumption $\mathcal{G}_j \subset \mathcal{G}_{j-1}$. Let $\boldsymbol{x} \in \mathcal{G}_{j+1}$. Then $\boldsymbol{x}$ can be written as $\boldsymbol{x} = (I - \omega_{j+1}A)\boldsymbol{y}$ for some $\boldsymbol{y} \in \mathcal{G}_j \cap \mathcal{S}$. From the assumption $\mathcal{G}_j \subset \mathcal{G}_{j-1}$, it follows that $\boldsymbol{y} \in \mathcal{G}_j \cap \mathcal{S} \subset \mathcal{G}_{j-1} \cap \mathcal{S}$, and thus $(I - \omega_j A)\boldsymbol{y} \in \mathcal{G}_j$. Since $\boldsymbol{y} \in \mathcal{G}_j$ and $\boldsymbol{y} - \omega_j A\boldsymbol{y} \in \mathcal{G}_j$, we have $A\boldsymbol{y} \in \mathcal{G}_j$, and thus $(I - \omega_{j+1}A)\boldsymbol{y} = \boldsymbol{x} \in \mathcal{G}_j$.

The second statement is shown next. The property $\mathcal{G}_{j+1} \subset \mathcal{G}_j$ means that $\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j)$ or $\dim(\mathcal{G}_{j+1}) = \dim(\mathcal{G}_j)$.

If $\mathcal{G}_j \cap \mathcal{S} \neq \mathcal{G}_j$, then $\dim(\mathcal{G}_j \cap \mathcal{S}) < \dim(\mathcal{G}_j)$ and thus we have $\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j)$ because $\dim(\mathcal{G}_{j+1}) = \dim((I - \omega_j A)(\mathcal{G}_j \cap \mathcal{S})) \leq \dim(\mathcal{G}_j \cap \mathcal{S}) < \dim(\mathcal{G}_j)$.

On the other hand, if $\mathcal{G}_j \cap \mathcal{S} = \mathcal{G}_j$, then $\mathcal{G}_j \subset \mathcal{S}$, and we have $\mathcal{G}_{j+1} = (I - \omega_{j+1}A)(\mathcal{G}_j \cap \mathcal{S}) = (I - \omega_{j+1}A)\mathcal{G}_j \subset \mathcal{G}_j$. (The last inclusion follows from the first statement $\mathcal{G}_{j+1} \subset \mathcal{G}_j$.) This implies $A\mathcal{G}_j \subset \mathcal{G}_j$, which means that $\mathcal{G}_j$ is an invariant subspace of A. Recalling $\mathcal{G}_j \subset \mathcal{S}$ and $\mathcal{G}_j \subset \mathcal{G}_0$ leads to $\mathcal{G}_j \subset \mathcal{G}_0 \cap \mathcal{S}$. From the assumption that $\mathcal{G}_0 \cap \mathcal{S}$ do not share a nontrivial invariant subspace of A, the subspace $\mathcal{G}_j$ is the trivial invariant subspace of A, i.e., $\mathcal{G}_j = \{\mathbf{0}\}$.

Therefore either $\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j)$ or $\dim(\mathcal{G}_j) = 0$ occurs, which indicates that $\dim(\mathcal{G}_k) = 0$, i.e., $\mathcal{G}_k = \{\mathbf{0}\}$ for some $k \leq N$. □

Corollary 3.3 *Let $\mathcal{S}$ be a subspace of $\mathbb{C}^N$ with* $\dim(\mathcal{S}) = N - s$ *for a given* $s \geq 1$. *If $\mathcal{G}_{i-1} + \mathcal{S} = \mathbb{C}^N$ and $(I - \omega_j A)$ is nonsingular, then*

$$\dim(\mathcal{G}_j) = \dim(\mathcal{G}_{j-1}) - s. \tag{3.106}$$

Proof Let W_1, W_2 be subspaces of $\mathbb{C}^N$ (or a general vector space V). Then it is well known in linear algebra that $\dim(W_1) + \dim(W_2) = \dim(W_1 + W_2) + \dim(W_1 \cap W_2)$. Let $W_1 = \mathcal{G}_{i-1}$, $W_2 = \mathcal{S}$. Then, from the assumption $\mathcal{G}_{i-1} + \mathcal{S} = \mathbb{C}^N$, we have $\dim(\mathcal{G}_{i-1} + \mathcal{S}) = N$. Thus $\dim(\mathcal{G}_i) = \dim((I - \omega_j A)(\mathcal{G}_{i-1} \cap \mathcal{S})) = \dim(\mathcal{G}_{i-1} \cap \mathcal{S}) = \dim(\mathcal{G}_{i-1}) + \dim(\mathcal{S}) - N = \dim(\mathcal{G}_{i-1}) - s$. □

Remark 3.1 From Corollary 3.3, if $\mathcal{G}_{i-1} + \mathcal{S} = \mathbb{C}^N$ for $i = 1, 2, \ldots, n$, then $\dim(\mathcal{G}_n) = \dim(\mathcal{G}_0) - ns$. Thus, if N/s is an integer and $\dim(\mathcal{G}_0) = N$, we have $\dim(\mathcal{G}_{N/s}) = 0$, i.e., $\mathcal{G}_{N/s} = \{\mathbf{0}\}$. This remark will be cited in Remark 3.2 for the maximum required number of matrix–vector multiplications of the IDR(s) method.

3.3.9.1 The Derivation of the IDR(s) Method

The IDR(s) method with a given number $s \geq 1$ generates residual vectors $\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots$ such that:

1. $\boldsymbol{r}_{j(s+1)}, \boldsymbol{r}_{j(s+1)+1}, \ldots, \boldsymbol{r}_{j(s+1)+s} \in \mathcal{G}_j, \quad j = 0, 1, 2, \ldots;$
2. The subspace $\mathcal{S}$ in (3.105) is chosen so that $\dim \mathcal{S} = N - s$.

For simplicity, we consider the case $s = 2$, i.e., the IDR(2) method. In this case, the IDR(2) method produces the following residual vectors:

$$\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathcal{G}_0, \quad \boldsymbol{r}_3, \boldsymbol{r}_4, \boldsymbol{r}_5 \in \mathcal{G}_1, \quad \boldsymbol{r}_6, \boldsymbol{r}_7, \boldsymbol{r}_8 \in \mathcal{G}_2, \ldots$$

Here we explain how to produce $\boldsymbol{r}_3, \boldsymbol{r}_4, \boldsymbol{r}_5 \in \mathcal{G}_1$ from the given residual vectors $\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathcal{G}_0$. Since $s = 2$, it follows from Remark 3.1 that we need to choose $\mathcal{S}$ such that $\dim(\mathcal{S}) = N - 2$. The standard choice is

$$\mathcal{S} = \operatorname{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}^{\perp} \tag{3.107}$$

with arbitrary linearly independent vectors $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, where the symbol $\perp$ is the orthogonal complement.

Now, let $S = [\boldsymbol{s}_1, \boldsymbol{s}_2] \in \mathbb{C}^{N\times 2}$ with linearly independent vectors $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$. Then for a vector $\boldsymbol{v} \in \mathbb{C}^N$, the following fact holds true:

$$\boldsymbol{v} \perp \mathcal{S} \Leftrightarrow S^{\mathrm{H}}\boldsymbol{v} = \boldsymbol{0}. \tag{3.108}$$

In what follows, we use the following definitions:

$$\Delta\boldsymbol{r}_k := \boldsymbol{r}_{k+1} - \boldsymbol{r}_k, \quad dR_k := [\Delta\boldsymbol{r}_k, \Delta\boldsymbol{r}_{k+1}], \quad \boldsymbol{c}_k = \left(S^{\mathrm{H}} dR_k\right)^{-1} S^{\mathrm{H}}\boldsymbol{r}_{k+2}. \tag{3.109}$$

The following vector $\boldsymbol{v}_0$ lies in $\mathcal{G}_0 \cap \mathcal{S}$:

$$\boldsymbol{v}_0 = \boldsymbol{r}_2 - dR_0\boldsymbol{c}_0 \in \mathcal{G}_0 \cap \mathcal{S}. \tag{3.110}$$

In fact, $\boldsymbol{v}_0 \in \mathcal{G}_0$ because $\boldsymbol{v}_0 \in \operatorname{span}\{\boldsymbol{r}_2, \Delta\boldsymbol{r}_0, \Delta\boldsymbol{r}_1\}$ with $\boldsymbol{r}_2, \Delta\boldsymbol{r}_0, \Delta\boldsymbol{r}_1 \in \mathcal{G}_0$ from the assumption $\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathcal{G}_0$, and $\boldsymbol{v}_0 \in \mathcal{S}$ because it is easy to see $S^{\mathrm{H}}\boldsymbol{v}_0 = \boldsymbol{0}$ and the relation (3.108).

From (3.105), we can produce $\boldsymbol{r}_3 \in \mathcal{G}_1$ by

$$\boldsymbol{r}_3 = (I - \omega_0 A)\boldsymbol{v}_0 \in \mathcal{G}_1, \tag{3.111}$$

where ω_0 is usually chosen so that $\|\boldsymbol{r}_3\|$ is minimized, i.e., $\omega_0 = (A\boldsymbol{v}_0, \boldsymbol{v}_0)/(A\boldsymbol{v}_0, A\boldsymbol{v}_0)$.

In order to produce $\boldsymbol{r}_4 \in \mathcal{G}_1$, we need a vector $\boldsymbol{v}_1$ such that $\boldsymbol{v}_1 \in \mathcal{G}_0 \cap \mathcal{S}$. To this end, $\boldsymbol{v}_1$ is constructed by using $\boldsymbol{r}_3$ and dR_1 as follows:

$$\boldsymbol{v}_1 = \boldsymbol{r}_3 - dR_1\boldsymbol{c}_1 \in \mathcal{G}_0 \cap \mathcal{S}.$$

From a similar discussion to that above, we see $\boldsymbol{v}_1 \in \mathcal{G}_0$ and $\boldsymbol{v}_1 \in \mathcal{S}$. Thus, we have

$$\boldsymbol{r}_4 = (I - \omega_0 A)\boldsymbol{v}_1 \in \mathcal{G}_1.$$

Similarly,

$$\boldsymbol{v}_2 = \boldsymbol{r}_4 - dR_2\boldsymbol{c}_2 \in \mathcal{G}_0 \cap \mathcal{S}, \quad \boldsymbol{r}_5 = (I - \omega_0 A)\boldsymbol{v}_2 \in \mathcal{G}_1.$$

Now, we obtained $\boldsymbol{r}_3, \boldsymbol{r}_4, \boldsymbol{r}_5 \in \mathcal{G}_1$ from the given residual vectors $\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathcal{G}_0$. Similarly, the following algorithm produces $\boldsymbol{r}_6, \boldsymbol{r}_7, \boldsymbol{r}_8 \in \mathcal{G}_2$ from $\boldsymbol{r}_3, \boldsymbol{r}_4, \boldsymbol{r}_5 \in \mathcal{G}_1$:

$$\begin{aligned}
\boldsymbol{v}_3 &= \boldsymbol{r}_5 - dR_3\boldsymbol{c}_3 \in \mathcal{G}_1 \cap \mathcal{S}, \quad \boldsymbol{r}_6 = (I - \omega_1 A)\boldsymbol{v}_3 \in \mathcal{G}_2,\\
\boldsymbol{v}_4 &= \boldsymbol{r}_6 - dR_4\boldsymbol{c}_4 \in \mathcal{G}_1 \cap \mathcal{S}, \quad \boldsymbol{r}_7 = (I - \omega_1 A)\boldsymbol{v}_4 \in \mathcal{G}_2,\\
\boldsymbol{v}_5 &= \boldsymbol{r}_7 - dR_5\boldsymbol{c}_5 \in \mathcal{G}_1 \cap \mathcal{S}, \quad \boldsymbol{r}_8 = (I - \omega_1 A)\boldsymbol{v}_5 \in \mathcal{G}_2,
\end{aligned}$$

where ω_1 is usually chosen so that $\|\boldsymbol{r}_6\|$ is minimized, i.e., $\omega_1 = (A\boldsymbol{v}_3, \boldsymbol{v}_3)/(A\boldsymbol{v}_3, A\boldsymbol{v}_3)$.

Now, we describe how to extract approximate solutions $\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5$ from $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2$. In what follows, we use the following definition:

$$\Delta\boldsymbol{x}_k := \boldsymbol{x}_{k+1} - \boldsymbol{x}_k, \quad dX_k := [\Delta\boldsymbol{x}_k, \Delta\boldsymbol{x}_{k+1}].$$

Then from (3.110) and (3.111) $\boldsymbol{x}_3$ is obtained as follows:

$$\begin{aligned}
\boldsymbol{r}_3 &= (I - \omega_0 A)\boldsymbol{v}_0 = \boldsymbol{r}_2 - dR_0\boldsymbol{c}_0 - \omega_0 A\boldsymbol{v}_0\\
&\Leftrightarrow \boldsymbol{b} - A\boldsymbol{x}_3 = \boldsymbol{b} - A\boldsymbol{x}_2 - dR_0\boldsymbol{c}_0 - \omega_0 A\boldsymbol{v}_0\\
&\Leftrightarrow \boldsymbol{x}_3 = \boldsymbol{x}_2 + A^{-1}dR_0\boldsymbol{c}_0 + \omega_0\boldsymbol{v}_0\\
&\Leftrightarrow \boldsymbol{x}_3 = \boldsymbol{x}_2 - dX_0\boldsymbol{c}_0 + \omega_0\boldsymbol{v}_0.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\boldsymbol{x}_4 &= \boldsymbol{x}_3 - dX_1\boldsymbol{c}_1 + \omega_0\boldsymbol{v}_1,\\
\boldsymbol{x}_5 &= \boldsymbol{x}_4 - dX_2\boldsymbol{c}_2 + \omega_0\boldsymbol{v}_2.
\end{aligned}$$

$\boldsymbol{x}_6, \boldsymbol{x}_7, \boldsymbol{x}_8$ can be obtained as follows:

$$\begin{aligned}
\boldsymbol{x}_6 &= \boldsymbol{x}_5 - dX_3\boldsymbol{c}_3 + \omega_1\boldsymbol{v}_3,\\
\boldsymbol{x}_7 &= \boldsymbol{x}_7 - dX_4\boldsymbol{c}_4 + \omega_1\boldsymbol{v}_4,\\
\boldsymbol{x}_8 &= \boldsymbol{x}_8 - dX_5\boldsymbol{c}_5 + \omega_1\boldsymbol{v}_5.
\end{aligned}$$

Finally, we describe how to determine initial residual vectors $\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathcal{G}_0$ and $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2$. These vectors are given as follows: we compute

$$\begin{aligned}
\boldsymbol{r}_0 &= \boldsymbol{b} - A\boldsymbol{x}_0,\\
\boldsymbol{r}_1 &= (I - c_0 A)\boldsymbol{r}_0, \quad \boldsymbol{x}_1 = \boldsymbol{x}_0 + c_0\boldsymbol{r}_0,\\
\boldsymbol{r}_2 &= (I - c_1 A)\boldsymbol{r}_1, \quad \boldsymbol{x}_2 = \boldsymbol{x}_1 + c_1\boldsymbol{r}_1,
\end{aligned}$$

where $\boldsymbol{x}_0$ is an arbitrary vector, and c_0 and c_1 are chosen so that $\|\boldsymbol{r}_1\|$ and $\|\boldsymbol{r}_2\|$ are minimized.

From the above discussion, the IDR(s) method with $s = 2$ can be easily generalized to the IDR(s) method with a general number $s \geq 1$, and the algorithm is described in Algorithm 3.24. The symbol "$\Delta X(:, i)$" means that the ith column of matrix ΔX.

Algorithm 3.24 The IDR(s) method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $P \in \mathbb{C}^{N \times s}$, $s \geq 1$
Output: $\boldsymbol{x}_i$
 1: $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
    (The other initial residuals $\boldsymbol{r}_1, \dots, \boldsymbol{r}_s \in \mathcal{G}_0$)
 2: for $i = 1$ to $s$ do
 3:     $\boldsymbol{v} = A\boldsymbol{r}_{i-1}$, $\omega = \frac{\boldsymbol{v}^{\mathrm{H}}\boldsymbol{r}_{i-1}}{\boldsymbol{v}^{\mathrm{H}}\boldsymbol{v}}$
 4:     $\Delta X(:, i) = \omega \boldsymbol{r}_{i-1}$, $\Delta R(:, i) = -\omega \boldsymbol{v}$
 5:     $\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \Delta X(:, i)$; $\boldsymbol{r}_i = \boldsymbol{r}_{i-1} + \Delta R(:, i)$
 6: end for
 7: $j = 1, i = s$
 8: $M = P^{\mathrm{H}}\Delta R$, $\boldsymbol{h} = P^{\mathrm{H}}\boldsymbol{r}_i$
 9: while $\frac{\|\boldsymbol{r}_i\|}{\|\boldsymbol{b}\|} > \epsilon$ do
10:     for $k = 0$ to $s$ do
11:         Solve $\boldsymbol{c}$ from $M\boldsymbol{c} = \boldsymbol{h}$
12:         $\boldsymbol{q} = -\Delta R\boldsymbol{c}$
13:         $\boldsymbol{v} = \boldsymbol{r}_i + \boldsymbol{q}$
14:         if $k = 0$ then
15:             $\boldsymbol{t} = A\boldsymbol{v}$, $\omega = \frac{\boldsymbol{t}^{\mathrm{H}}\boldsymbol{v}}{\boldsymbol{t}^{\mathrm{H}}\boldsymbol{t}}$
16:             $\Delta R(:, j) = \boldsymbol{q} - \omega \boldsymbol{t}$
17:             $\Delta X(:, j) = -\Delta X\boldsymbol{c} + \omega \boldsymbol{v}$
18:         else
19:             $\Delta X(:, j) = -\Delta X\boldsymbol{c} + \omega \boldsymbol{v}$
20:             $\Delta R(:, j) = -A\Delta X(:, j)$
21:         end if
            (Update approximate solutions $x_i$)
22:         $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i + \Delta R(:, j)$
23:         $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \Delta X(:, j)$
24:         $\Delta \boldsymbol{m} = P^{\mathrm{H}}\Delta R(:, j)$
25:         $M(:, j) = \Delta \boldsymbol{m}$
26:         $\boldsymbol{h} = \boldsymbol{h} + \Delta \boldsymbol{m}$
27:         $i = i + 1, j = j + 1$
28:         $j = (j - 1)\%s + 1$    (%: modulo operation, i.e. $a\%n = r$, where $a = mn + r$.)
29:     end for
30: end while
```

Remark 3.2 In order to produce residual vectors in $\mathcal{G}_j$ from residual vectors in $\mathcal{G}_{j-1}$, the IDR(s) method requires $s + 1$ matrix–vector multiplications. From Remark 3.1, the residual vector in $\mathcal{G}_{N/s}$ is zero. Thus, in exact precision arithmetic, the number of matrix–vector multiplications for the IDR(s) method is at most $(s + 1) \times N/s = N + N/s$ to obtain the exact solution. Note that the number of matrix–vector multiplications for the product-type Krylov subspace methods in Sect. 3.3.8 is at most $2N$.

From the following remark, the IDR(s) method can be regarded as an extension of the BiCGSTAB method.

Remark 3.3 The IDR(1) method and the BiCGSTAB method give the same residual vectors at the even steps.

Remark 3.4 In terms of orthogonality, the IDR(s) method produces residual vectors that belong to the following so-called Sonneveld subspace:

$$\left\{\Omega_j(A)\boldsymbol{v} \,:\, \boldsymbol{v} \perp \mathcal{B}_j(A^{\mathrm{H}}, S)\right\}. \tag{3.112}$$

Here $\Omega_j(A) := \Pi_{i=1}^{j}(I - \omega_i A)$, and $\mathcal{B}_j(A^{\mathrm{H}}, S)$ is a subspace of $\mathbb{C}^N$ generated by using A and $S := [\boldsymbol{s}_1, \ldots, \boldsymbol{s}_s]$, which is defined by

$$\mathcal{B}_j(A^{\mathrm{H}}, S) := \mathcal{K}_j(A^{\mathrm{H}}, \boldsymbol{s}_1) + \cdots + \mathcal{K}_j(A^{\mathrm{H}}, \boldsymbol{s}_s).$$

Here the symbol "+" means the sum of subspaces.[5] $\mathcal{B}_j(A^{\mathrm{H}}, R_0^*)$ will be redefined in Definition 3.1 for block Krylov subspace methods. For the details of the view in (3.112), see [161, 163].

We give a rough explanation of Remark 3.4 using the IDR(s) method with $s = 2$ for the two cases $j = 1, 2$ in (3.112). First we consider the case $j = 1$. From (3.105) $\mathcal{G}_1$ is generated by

$$\mathcal{G}_1 = (I - \omega_1 A)(\mathcal{G}_0 \cap \mathcal{S}),$$

where $\mathcal{S} = \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}^{\perp}$, see also (3.107). Then, $\boldsymbol{r} \in \mathcal{G}_1$ can be written as

$$\boldsymbol{r} = (I - \omega_1 A)\boldsymbol{v}, \quad \boldsymbol{v} \in \mathcal{G}_0, \quad \boldsymbol{v} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}.$$

Note that $\boldsymbol{v} \in \mathcal{S}$ means $\boldsymbol{v} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}$. Since $\mathcal{B}_1(A^{\mathrm{H}}, S) = \mathcal{K}_1(A^{\mathrm{H}}, \boldsymbol{s}_1) + \mathcal{K}_1(A^{\mathrm{H}}, \boldsymbol{s}_2) = \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}$, we have

$$\boldsymbol{r} \in \{(I - \omega_1 A)\boldsymbol{v} \,:\, \boldsymbol{v} \perp \mathcal{B}_1(A^{\mathrm{H}}, S)\} = \{\Omega_1(A)\boldsymbol{v} \,:\, \boldsymbol{v} \perp \mathcal{B}_1(A^{\mathrm{H}}, S)\},$$

which corresponds to the case $j = 1$ in (3.112).

Next, we consider the case $j = 2$. $\mathcal{G}_2$ is generated by

$$\mathcal{G}_2 = (I - \omega_2 A)(\mathcal{G}_1 \cap \mathcal{S}).$$

Then a residual vector $\tilde{\boldsymbol{r}} \in \mathcal{G}_2$ can be written as

$$\tilde{\boldsymbol{r}} = (I - \omega_2 A)\tilde{\boldsymbol{v}}, \quad \tilde{\boldsymbol{v}} \in \mathcal{G}_1, \quad \tilde{\boldsymbol{v}} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}. \tag{3.113}$$

[5] Given two subspaces W_1 and W_2, the sum of W_1 and W_2 is defined by $W_1 + W_2 := \{\boldsymbol{w}_1 + \boldsymbol{w}_2 \,:\, \boldsymbol{w}_1 \in W_1, \boldsymbol{w}_2 \in W_2\}$.

Since $\tilde{\boldsymbol{v}} \in \mathcal{G}_1 = (I - \omega_1 A)(\mathcal{G}_0 \cap \mathcal{S})$, the vector $\tilde{\boldsymbol{v}}$ can be written as

$$\tilde{\boldsymbol{v}} = (I - \omega_1 A)\boldsymbol{v}, \quad \boldsymbol{v} \in \mathcal{G}_0, \quad \boldsymbol{v} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}. \tag{3.114}$$

From (3.113) and (3.114), we have

$$\tilde{\boldsymbol{r}} = (I - \omega_2 A)(I - \omega_1 A)\boldsymbol{v}, \tag{3.115}$$

where

$$\tilde{\boldsymbol{v}} = (I - \omega_1 A)\boldsymbol{v} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}, \quad \boldsymbol{v} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}.$$

Note that

$$\begin{aligned}(I - \omega_1 A)\boldsymbol{v} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\} &\Leftrightarrow ((I - \omega_1 A)\boldsymbol{v})^{\mathrm{H}}\boldsymbol{s}_i = 0 \quad (i = 1, 2)\\ &\Leftrightarrow \boldsymbol{v}^{\mathrm{H}}(I - \omega_1 A)^{\mathrm{H}}\boldsymbol{s}_i = 0 \quad (i = 1, 2)\\ &\Leftrightarrow \boldsymbol{v} \perp \mathrm{span}\{(I - \omega_1 A)^{\mathrm{H}}\boldsymbol{s}_1, (I - \omega_1 A)^{\mathrm{H}}\boldsymbol{s}_2\}.\end{aligned}$$

$\boldsymbol{v} \perp \mathrm{span}\{(I - \omega_1 A)^{\mathrm{H}}\boldsymbol{s}_1, (I - \omega_1 A)^{\mathrm{H}}\boldsymbol{s}_2\}$ and $\boldsymbol{v} \perp \mathrm{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2\}$ imply

$$\boldsymbol{v} \perp \boldsymbol{s}_1, \quad \boldsymbol{v} \perp \boldsymbol{s}_2, \quad \boldsymbol{v} \perp A^{\mathrm{H}}\boldsymbol{s}_1, \quad \boldsymbol{v} \perp A^{\mathrm{H}}\boldsymbol{s}_1.$$

Thus,

$$\begin{aligned}\boldsymbol{v} \perp\ &\mathrm{span}\{\boldsymbol{s}_1, A^{\mathrm{H}}\boldsymbol{s}_1, \boldsymbol{s}_2, A^{\mathrm{H}}\boldsymbol{s}_2\}.\\ &= \mathrm{span}\{\boldsymbol{s}_1, A^{\mathrm{H}}\boldsymbol{s}_1\} + \mathrm{span}\{\boldsymbol{s}_2, A^{\mathrm{H}}\boldsymbol{s}_2\}\\ &= \mathcal{K}_2(A^{\mathrm{H}}, \boldsymbol{s}_1) + \mathcal{K}_2(A^{\mathrm{H}}, \boldsymbol{s}_2)\\ &= \mathcal{B}_2(A^{\mathrm{H}}, S).\end{aligned}$$

Then, from (3.115), together with the above result, we have

$$\tilde{\boldsymbol{r}} = (I - \omega_2 A)(I - \omega_1 A)\boldsymbol{v}, \quad \boldsymbol{v} \perp \mathcal{B}_2(A^{\mathrm{H}}, S), \tag{3.116}$$

from which we obtain

$$\tilde{\boldsymbol{r}} \in \{\Omega_2(A)\boldsymbol{v} \,:\, \boldsymbol{v} \perp \mathcal{B}_2(A^{\mathrm{H}}, S)\}. \tag{3.117}$$

This corresponds to the case $j = 2$ in (3.112).

In what follows, some references related to the IDR(s) method are described. An elegant explanation of the IDR(s) method is found in [88], where a visualization of the IDR theorem in [88, Fig. 3.1] is useful for intuitively understanding the IDR theorem (Theorem 3.4). A more stable and accurate variant of the IDR(s) method is proposed by the same authors in [199], and the IDR(s) method with the quasi-minimal residual strategy is proposed in [47]. Variants of the IDR(s) method with partial orthonormalization is proposed in [209]. Extensions of the IDR(s) methods using the idea of the BiCGSTAB(ℓ) method are known as the IDRstab method [164] and the GBiCGSTAB(s, L) method [185]. Variants of the IDRstab method are found in [5–7].

3.3.9.2 Numerical Example

In this subsection, typical convergence behavior of the IDR(s) method is shown by one numerical experiment. We consider nonsymmetric linear systems (2.18) with parameters $N = 20$, $a_1 = a_2 = a_3 = 1$, $b_1 = b_2 = b_3 = 1$, and $c = 300$. The stopping criterion is $\|\boldsymbol{r}_n\|/\|\boldsymbol{b}\| \leq 10^{-10}$.

The convergence histories of the IDR(s) method ($s = 1, 2, 4$) and the BiCGSTAB method are shown in Fig. 3.6. The horizontal axis is the number of matrix–vector multiplications.

From Fig. 3.6, we see that as the number of s in the IDR(s) method increases, the required number of matrix–vector multiplications decreases. The IDR(4) method and the IDR(2) method converge faster than the BiCGSTAB method. In terms of accuracy, the true relative residual 2-norms for the BiCGSTAB method, the IDR(1) method, the IDR(2) method, and the IDR(4) method are –10.50, –10.04, –10.2, –8.76.

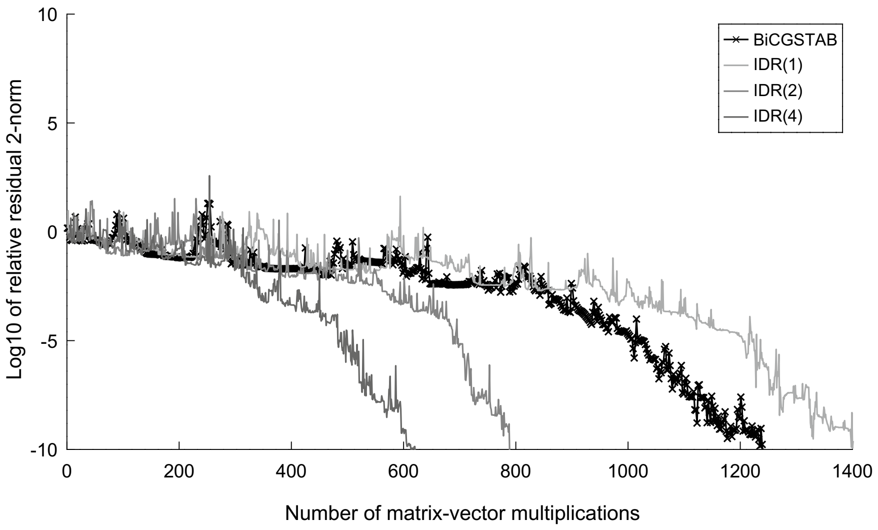


Fig. 3.6 Convergence histories of BiCGSTAB, IDR(1), IDR(2), and IDR(4)

3.3.10 Block Induced Dimension Reduction (Block IDR(s)) Method

The block IDR(s) method [46] is an extension of the IDR(s) method to solve the block linear systems of the form

$$AX = B, \tag{3.118}$$

where $A \in \mathbb{C}^{N\times N}$, $B \in \mathbb{C}^{N\times m}$, and m is usually much less than N.

Prior to the theory of the block IDR(s) method, the framework of block Krylov subspace methods is briefly described after the following definitions:

Definition 3.1 (Block Krylov subspace) Let $V \in \mathbb{C}^{N\times m}$. Then $\mathcal{K}_n(A, V) \subset \mathbb{C}^{N\times m}$ defined by

$$\begin{aligned}\mathcal{K}_n(A, V) := &\left\{VC_0 + AVC_1 + \cdots + A^{n-1}VC_{n-1} \ : \ C_0, C_1, \ldots, C_{n-1} \in \mathbb{C}^{m\times m}\right\}\\ &(= \text{block span}\{V, AV, \ldots, A^{n-1}V\})\end{aligned}$$

is called the *block Krylov subspace* with respect to A and V.

Definition 3.2 (Sum of Krylov subspaces) Let $V = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m] \in \mathbb{C}^{N\times m}$. Then $\mathcal{B}_n(A, V) \subset \mathbb{C}^N$ is defined by

$$\begin{aligned}\mathcal{B}_n(A, V) &:= \mathcal{K}_n(A, \boldsymbol{v}_1) + \mathcal{K}_n(A, \boldsymbol{v}_2) + \cdots + \mathcal{K}_n(A, \boldsymbol{v}_m)\\ &= \left\{V\boldsymbol{c}_0 + AV\boldsymbol{c}_1 + \cdots + A^{n-1}V\boldsymbol{c}_{n-1} \ : \ \boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{n-1} \in \mathbb{C}^m\right\}.\end{aligned}$$

Definition 3.3 (Grade of block Krylov subspace) The smallest number n such that

$$\dim(\mathcal{B}_n(A, V)) = \dim(\mathcal{B}_{n+1}(A, V))$$

is called *the block grade of A with respect to V*, which is denoted by $\nu(V, A)$.

Remark 3.5 Definitions 3.1 and 3.3 are extensions of Definitions 1.2 and 1.3. In fact, if $m = 1$, these definitions are identical to Definitions 1.2 and 1.3.

We now describe the framework of the block Krylov subspace methods. Let $X_0 \in \mathbf{C}^{N\times s}$ be an initial guess, and let $R_0 := B - AX_0$ be the corresponding initial residual matrix. Then, block Krylov subspace methods find the nth approximate solution X_n as follows:

$$X_n = X_0 + Z_n, \quad Z_n \in \mathcal{K}_n(A, R_0).$$

The corresponding residual matrix is given by

$$R_n = B - AX_n = R_0 - AZ_n \in \mathcal{K}_{n+1}(A, R_0).$$

Determining Z_n yields various block Krylov subspace methods, e.g.,

- The block CG method: $R_n \perp \mathcal{B}_n(A, R_0)$.
- The block COCG method: $R_n \perp \overline{\mathcal{B}_n(A, R_0)}$.
- The block BiCG method: $R_n \perp \mathcal{B}_n(A^{\mathrm{H}}, R_0^*)$.

Here, R_0^* is an initial shadow residual matrix,[6] and the symbol $R_n \perp \mathcal{B}_n(A, V)$ means that all the column vectors of R_n are orthogonal to $\mathcal{B}_n(A, V)$.

Here, we give a brief history of the block Krylov subspace methods. In 1980, O'Leary proposed the block BiCG method (and the block CG method) in [142]. In 1990, Vital proposed the block GMRES method in [202]. Block Krylov subspace methods based on the QMR method were proposed in 1997 [65, 157]. In 2003, Guennouni et al. proposed the block BiCGSTAB method [86]. In 2011, Du et al. proposed the block IDR(s) method [46]. For other block Krylov subspace methods, see [153, 181, 182, 210].

Now we turn our attention to deriving the block IDR(s) method. The block IDR(s) method is based on the following theorem (see [46]), which is a slight modification of Theorem 3.4.

Theorem 3.5 *Let A be any matrix in $\mathbb{C}^{N\times N}$, let $\boldsymbol{v}_0$ be any nonzero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0 = \mathcal{B}_{\nu(R_0,A)}(A, R_0)$. Let $\mathcal{S}$ denote any (proper) subspace of $\mathbb{C}^N$ such that $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace,[7] and define the sequences $\mathcal{G}_j$, $j = 1, 2, \dots$ as*

$$\mathcal{G}_j = (I - \omega_j A)(\mathcal{G}_{j-1} \cap \mathcal{S}), \tag{3.119}$$

where the ω_j's are nonzero scalars. Then:

(1) $\mathcal{G}_j \subset \mathcal{G}_{j-1}$ *for all* $j > 0$.
(2) $\mathcal{G}_j = \{\mathbf{0}\}$ *for some* $j \le N$.

Proof The only difference between Theorems 3.4 and 3.5 is the definition of $\mathcal{G}_0$, i.e., $\mathcal{G}_0 = \mathcal{K}_N(A, \boldsymbol{v}_0)$ or $\mathcal{G}_0 = \mathcal{B}_{\nu(R_0,A)}(A, R_0)$. In the proof of Theorem 3.4-(1), $\mathcal{G}_0 = \mathcal{K}_N(A, \boldsymbol{v}_0)$ is used only for proving $\mathcal{G}_1 \subset \mathcal{G}_0$. Thus, all we have to do is to prove $\mathcal{G}_1 \subset \mathcal{G}_0$. From $\mathcal{G}_0 = \mathcal{B}_{\nu(R_0,A)}(A, R_0)$, we have $(I - \omega_1 A)\mathcal{G}_0 \subset \mathcal{G}_0$, because for a given $\boldsymbol{v} \in (I - \omega_1 A)\mathcal{G}_0 = (I - \omega_1 A)\mathcal{B}_{\nu(R_0,A)}(A, R_0)$ we have

$$\begin{aligned}
\boldsymbol{v} &= (I - \omega_1 A)(R_0\boldsymbol{c}_0 + AR_0\boldsymbol{c}_1 + \cdots + A^{\nu(R_0,A)-1}R_0\boldsymbol{c}_{\nu(R_0,A)-1}) \\
&= (I - \omega_1 A)R_0\boldsymbol{c}_0 + (I - \omega_1 A)AR_0\boldsymbol{c}_1 + \cdots + (I - \omega_1 A)A^{\nu(R_0,A)-1}R_0\boldsymbol{c}_{\nu(R_0,A)-1} \\
&= R_0\boldsymbol{d}_0 + AR_0\boldsymbol{d}_1 + \cdots + A^{\nu(R_0,A)-1}R_0\boldsymbol{d}_{\nu(R_0,A)-1} + A^{\nu(R_0,A)}R_0\boldsymbol{d}_{\nu(R_0,A)} \\
&= R_0\tilde{\boldsymbol{d}}_0 + AR_0\tilde{\boldsymbol{d}}_1 + \cdots + A^{\nu(R_0,A)-1}R_0\tilde{\boldsymbol{d}}_{\nu(R_0,A)-1} \\
&\in \mathcal{G}_0.
\end{aligned}$$

[6] In practice, R_0^* is usually set to $R_0^* = R_0$ or a random matrix.

[7] This means that $\mathcal{S} \cap \mathcal{G}_0$ does not contain any eigenvector of A, and the trivial invariant subspace is $\{\mathbf{0}\}$.

Thus $\mathcal{G}_1 = (I - \omega_1 A)(\mathcal{G}_0 \cap \mathcal{S}) \subset (I - \omega_1 A)\mathcal{G}_0 \subset \mathcal{G}_0$.

Theorem 3.5-(2) is shown by following the proof of Theorem 3.4-(2) and using $\dim \mathcal{G}_0 \le N$. □

Similar to Corollary 3.3, we have the following fact:

Corollary 3.4 *Let $\mathcal{S}$ be a subspace of $\mathbb{C}^N$ with $\dim(\mathcal{S}) = N - sm$ for a given $s, m \ge 1$. If $\mathcal{G}_{i-1} + \mathcal{S} = \mathbb{C}^N$ and $(I - \omega_j A)$ is nonsingular, then*

$$\dim(\mathcal{G}_j) = \dim(\mathcal{G}_{j-1}) - sm.$$

The difference between Corollaries 3.3, 3.4 is the parameter m. The parameter m corresponds to the size of the number of right-hand sides of $AX = B$, where B is an N-by-m matrix.

In what follows, the block IDR(s) method is explained. The block IDR(s) method generates residual matrices $R_k = B - AX_k$ for $k = 0, 1, \dots$ such that:

1. all the column vectors of $R_{j(s+1)}, R_{j(s+1)+1}, \dots, R_{j(s+1)+s}$ belong to $\mathcal{G}_j$ in (3.119) for $j = 0, 1, 2, \dots$;
2. the subspace $\mathcal{S}$ in (3.119) is chosen so that $\dim \mathcal{S} = N - sm$.

Let $S = [\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_{sm}] \in \mathbb{C}^{N \times sm}$. Then, a standard choice of the subspace $\mathcal{S}$ is as follows:

$$\mathcal{S} = \text{span}\{S\}^{\perp} = \text{span}\{\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_{sm}\}^{\perp},$$

where all the column vectors of $S \in \mathbb{C}^{N \times sm}$ are linearly independent, and thus it is easy to see that $\dim \mathcal{S} = N - sm$.

We now briefly explain the block IDR(s) method with $s = 2$. For $j = 0, 1, 2$, the block IDR(2) method produces the following residual matrices:

$$R_0, R_1, R_2, \quad R_3, R_4, R_5, \quad R_6, R_7, R_8,$$

where all the column vectors R_0, R_1, R_2 belong to $\mathcal{G}_0$, all the column vectors R_3, R_4, R_5 belong to $\mathcal{G}_1$, and all the column vectors R_3, R_4, R_5 belong to $\mathcal{G}_2$.

In what follows, we describe how to obtain R_3, R_4, R_5 from R_0, R_1, R_2 under the assumption that all the column vectors of R_0, R_1, R_2 belong to $\mathcal{G}_0$. Similar to the IDR(s) method, the following definitions are useful for the block IDR(s) method:

$$\Delta R_k := R_{k+1} - R_k, \quad dR_k^{\blacksquare} := [\Delta R_k, \Delta R_{k+1}], \quad C_k = \left(S^{\mathrm{H}} dR_k^{\blacksquare}\right)^{-1} S^{\mathrm{H}} R_{k+2}. \tag{3.120}$$

All the column vectors of the following matrix V_0 lie in $\mathcal{G}_0 \cap \mathcal{S}$:

$$V_0 = R_2 - dR_0^{\blacksquare} C_0, \quad \text{Im}(V_0) \subset \mathcal{G}_0 \cap \mathcal{S}$$

because all the column vectors of $R_2 \in \mathbb{C}^{N\times m}$ and $dR_0 \in \mathbb{C}^{N\times 2m}$ belong to $\mathcal{G}_0$ from the assumption that all the column vectors of R_0, R_1, R_2 belong to $\mathcal{G}_0$, and all the column vectors of V_0 also belong to $\mathcal{S}$ from the relation $S^{\mathrm{H}}V_0 = O$, where O is the $2m \times m$ zero matrix. Here $\mathrm{Im}(V_0)$ is the image of V_0, i.e., the subspace spanned by all the column vectors of V_0.

From (3.119), the residual matrix R_3 whose column vectors belong to $\mathcal{G}_1$ can be produced by

$$R_3 = (I - \omega_0 A)V_0, \quad \mathrm{Im}(R_3) \subset \mathcal{G}_1, \tag{3.121}$$

where ω_0 is usually chosen so that $\|R_3\|_{\mathrm{F}}$ is minimized,[8] i.e.,

$$\omega_0 = \frac{\mathrm{Tr}(T_0^{\mathrm{H}}V_0)}{\mathrm{Tr}(T_0^{\mathrm{H}}T_0)}, \quad T_0 := AV_0.$$

The Frobenius norm minimization follows from the following fact:

Proposition 3.7 *Let $V, W \in \mathbb{C}^{m\times n}$. Then the following minimization problem:*

$$\min_{\omega\in\mathbb{C}} \|V - \omega W\|_{\mathrm{F}}$$

can be solved by

$$\omega = \frac{\mathrm{Tr}(W^{\mathrm{H}}V)}{\mathrm{Tr}(W^{\mathrm{H}}W)}.$$

Proof Let $V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$ and $W = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n]$, then using

$$\boldsymbol{v} := \mathrm{vec}(V) = \begin{bmatrix} \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{v}_n \end{bmatrix}, \quad \boldsymbol{w} := \mathrm{vec}(W) = \begin{bmatrix} \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_n \end{bmatrix}$$

yields $\|V - \omega W\|_F = \|\boldsymbol{v} - \omega\boldsymbol{w}\|$. Thus from (1.34), the minimizer is

$$\omega = \frac{\boldsymbol{w}^{\mathrm{H}}\boldsymbol{v}}{\boldsymbol{w}^{\mathrm{H}}\boldsymbol{w}} = \frac{\mathrm{Tr}(W^{\mathrm{H}}V)}{\mathrm{Tr}(W^{\mathrm{H}}W)},$$

which concludes the proof. □

Similar to the derivation of $\boldsymbol{r}_3$ and $\boldsymbol{r}_4$ in the IDR(s) method, $\boldsymbol{R}_3$ and $\boldsymbol{R}_4$ are obtained as follows:

[8] For $R = (r_{i,j}) \in \mathbb{C}^{N\times m}$, the symbol $\|R\|_{\mathrm{F}} := (\sum_{i=1}^{N}\sum_{j=1}^{m} \bar{r}_{i,j}r_{i,j})^{1/2}$ is called the *Frobenius norm* of R, and for a square matrix $A = (a_{i,j}) \in \mathbb{C}^{N\times N}$, the symbol $\mathrm{Tr}(A) := \sum_{i=1}^{N} a_{i,i}$ is called the *trace* of A.

$$\begin{aligned}
V_1 &= R_3 - dR_1^{\blacksquare} C_1, \quad \mathrm{Im}(V_1) \subset \mathcal{G}_0 \cap \mathcal{S}, \\
R_4 &= (I - \omega_0 A) V_1, \quad \mathrm{Im}(R_4) \subset \mathcal{G}_1, \\
V_2 &= R_4 - dR_2^{\blacksquare} C_2, \quad \mathrm{Im}(V_2) \subset \mathcal{G}_0 \cap \mathcal{S}, \\
R_5 &= (I - \omega_0 A) V_2, \quad \mathrm{Im}(R_5) \subset \mathcal{G}_1.
\end{aligned}$$

From the above calculations, we have residual matrices R_k $(k = 0, \dots, 5)$ such that

$$\begin{aligned}
&\mathrm{Im}(R_k) \subset \mathcal{G}_0 \quad \text{for } k = 0, 1, 2, \\
&\mathrm{Im}(R_k) \subset \mathcal{G}_1 \quad \text{for } k = 3, 4, 5.
\end{aligned}$$

Next, we describe how to obtain approximate solution matrices X_3, X_4, X_5 from X_0, X_1, X_2. Let

$$\Delta X_k := X_{k+1} - X_k, \quad dX_k^{\blacksquare} := [\Delta X_k, \Delta X_{k+1}].$$

Then X_3 is obtained as follows:

$$\begin{aligned}
R_3 = (I - \omega_0 A) V_0 &= R_2 - dR_0^{\blacksquare} C_0 - \omega_0 A V_0 \\
\Leftrightarrow B - AX_3 &= B - AX_2 - dR_0^{\blacksquare} C_0 - \omega_0 A V_0 \\
\Leftrightarrow X_3 &= X_2 + A^{-1} dR_0^{\blacksquare} C_0 + \omega_0 V_0 \\
\Leftrightarrow X_3 &= X_2 - dX_0^{\blacksquare} C_0 + \omega_0 V_0.
\end{aligned}$$

Similarly,

$$\begin{aligned}
X_4 &= X_3 - dX_1^{\blacksquare} C_1 + \omega_0 V_1, \\
X_5 &= X_4 - dX_2^{\blacksquare} C_2 + \omega_0 V_2.
\end{aligned}$$

We have described how to obtain R_k and X_k for $k = 3, 4, 5$ from R_k and X_k for $k = 0, 1, 2$ in the block IDR(s) method with $s = 2$. Following the above derivation, we have the block IDR(s) method in Algorithm 3.25. Further, the preconditioned version of the block IDR(s) method is shown in Algorithm 3.26. Note that the symbol "$B(:, \ell)$" represents the ℓth column of matrix B, and the symbol "$\Delta X(:, im + 1 : (i + 1)m)$" represents the $(im + 1)$th,$(im + 2)$th,..., $(i + 1)m$th columns of matrix ΔX.

Remark 3.6 From Corollary 3.4, the dimension reduction is usually sm, i.e., dim $(\mathcal{G}_j) = \dim(\mathcal{G}_{j-1}) - sm$ and the block IDR(s) method requires $m(s + 1)$ matrix–vector multiplications. Thus, in exact precision arithmetic, the total number of matrix–vector multiplications for the block IDR(s) method is at most $m(s + 1) \times N/sm = N + N/s$ to obtain the exact solution matrix, which does not depend on the block size m. Note that if the IDR(s) method is applied to $AX = B$ (i.e., $A\boldsymbol{x}_1 = \boldsymbol{b}_1, A\boldsymbol{x}_2 = \boldsymbol{b}_2, \dots, A\boldsymbol{x}_m = \boldsymbol{b}_m$), then the required number of matrix–vector multiplications of the IDR(s) method is at most $m(N + N/s)$, which is m times larger than that of the block IDR(s) method.

From the following remark, the block IDR(s) method can be regarded as an extension of the block BiCGSTAB method.

Remark 3.7 The block IDR(1) method and the block BiCGSTAB method give the same residual vectors at the even steps.

Algorithm 3.25 The block IDR(s) method

```
Input: $X_0 \in \mathbb{C}^{N\times s}$, $R_0 = B - AX_0$, $P \in \mathbb{C}^{n\times sm}$
Output: $X_i$
 1: for $i = 0$ to $s - 1$ do
 2:    $V = AR_i$, $\omega = \frac{\mathrm{Tr}(V^{\mathrm{H}}R_i)}{\mathrm{Tr}(V^{\mathrm{H}}V)}$
 3:    $\Delta X(:, im+1 : (i+1)m) = \omega R_i$, $\Delta R(:, im+1 : (i+1)m) = -\omega V$
 4:    $X_{i+1} = X_i + \Delta X(:, im+1 : (i+1)m)$, $R_{i+1} = R_i + \Delta R(:, im+1 : (i+1)m)$
 5: end for
 6: $j = 1, i = s$
 7: $M = P^{\mathrm{H}}\Delta R, H = P^{\mathrm{H}}R_i$
 8: while $\max_{\ell\in\{1,2,\ldots,m\}} \frac{\|R_i(:,\ell)\|_{\mathrm{F}}}{\|B(:,\ell)\|_{\mathrm{F}}} > \epsilon$ do
 9:    for $k = 0$ to $s$ do
10:       Solve $C$ from $MC = H$
11:       $Q = -\Delta RC$,
12:       $V = R_i + Q$
13:       if $k = 0$ then
14:          $T = AV$, $\omega = \frac{\mathrm{Tr}(T^{\mathrm{H}}V)}{\mathrm{Tr}(T^{\mathrm{H}}T)}$
15:          $\Delta R(:, (j-1)m+1 : jm) = Q - \omega T$
16:          $\Delta X(:, (j-1)m+1 : jm) = -\Delta XC + \omega V$
17:       else
18:          $\Delta X(:, (j-1)m+1 : jm) = -\Delta XC + \omega V$
19:          $\Delta R(:, (j-1)m+1 : jm) = -A\Delta X(:, (j-1)m+1 : jm)$
20:       end if
21:       $R_{i+1} = R_i + \Delta R(:, (j-1)m+1 : jm)$
22:       $X_{i+1} = X_i + \Delta X(:, (j-1)m+1 : jm)$
23:       $\Delta M = P^{\mathrm{H}}\Delta R(:, (j-1)m+1 : jm)$
24:       $M(:, (j-1)m+1 : jm) = \Delta M$
25:       $H = H + \Delta M$
26:       $i = i+1, j = j+1$
27:       $j = (j-1)\%s + 1$   (%: modulo operation, i.e. $a\%n = r$, where $a = mn + r$.)
28:    end for
29: end while
```

Algorithm 3.26 The preconditioned block IDR(s) method

```
Input: $X_0 \in \mathbb{C}^{N\times s}$, $R_0 = B - AX_0$, $P \in \mathbb{C}^{n\times sm}$
Input: $X_0 = KX_0$ ($K$: preconditioning matrix)
Output: $X_i$
 1: for $i = 0$ to $s - 1$ do
 2:   $W = K^{-1}R_i$
 3:   $V = AW, \omega = \frac{\mathrm{Tr}((K_1^{-1}V)^{\mathrm{H}}K_1^{-1}R_i)}{\mathrm{Tr}((K_1^{-1}V)^{\mathrm{H}}K_1^{-1}V)}$
 4:   $\Delta X(:, im+1 : (i+1)m) = \omega R_i$, $\Delta R(:, im+1 : (i+1)m) = -\omega V$
 5:   $X_{i+1} = X_i + \Delta X(:, im+1 : (i+1)m)$, $R_{i+1} = R_i + \Delta R(:, im+1 : (i+1)m)$
 6: end for
 7: $j = 1, i = s$
 8: $M = P^{\mathrm{H}}\Delta R, H = P^{\mathrm{H}}R_i$
 9: while $\max_{\ell\in\{1,2,\ldots,m\}} \frac{\|R_i(:,\ell)\|_{\mathrm{F}}}{\|B(:,\ell)\|_{\mathrm{F}}} > \epsilon$ do
10:   for $k = 0$ to $s$ do
11:     Solve $C$ from $MC = H$
12:     $Q = -\Delta RC$
13:     $V = R_i + Q$
14:     if $k = 0$ then
15:       $T = AK^{-1}V, \omega = \frac{\mathrm{Tr}((K_1^{-1}T)^{\mathrm{H}}K_1^{-1}V)}{\mathrm{Tr}((K_1^{-1}T)^{\mathrm{H}}K_1^{-1}T)}$
16:       $\Delta R(:, (j-1)m+1 : jm) = Q - \omega T$
17:       $\Delta X(:, (j-1)m+1 : jm) = -\Delta XC + \omega V$
18:     else
19:       $\Delta X(:, (j-1)m+1 : jm) = -\Delta XC + \omega V$
20:       $\Delta R(:, (j-1)m+1 : jm) = -AK^{-1}\Delta X(:, (j-1)m+1 : jm)$
21:     end if
22:     $R_{i+1} = R_i + \Delta R(:, (j-1)m+1 : jm)$
23:     $X_{i+1} = X_i + \Delta X(:, (j-1)m+1 : jm)$
24:     $\Delta M = P^{\mathrm{H}}\Delta R(:, (j-1)m+1 : jm)$
25:     $M(:, (j-1)m+1 : jm) = \Delta M$
26:     $H = H + \Delta M$
27:     $i = i+1, j = j+1$
28:     $j = (j-1)\%s + 1$   (%: modulo operation, i.e. $a\%n = r$, where $a = mn + r$.)
29:   end for
30: end while
31: $X_i = K^{-1}X_i$
```

3.4 Other Krylov Subspace Methods

This section describes the Krylov subspace methods based on normal equations and augmented linear systems: the CGNE method, the CGNR method, and the LSQR method.

3.4.1 Krylov Subspace Methods for Normal Equations

As described in the introduction in Sect. 3.3, we can apply the CG method to normal equation (3.56) because $A^{\mathrm{H}}A$ is Hermitian positive definite. In this subsection, the details are described. Instead of non-Hermitian linear systems $A\boldsymbol{x} = \boldsymbol{b}$, we consider the following normal equations:

$$A^{\mathrm{H}}A\boldsymbol{x} = A^{\mathrm{H}}\boldsymbol{b} \tag{3.122}$$

or

$$AA^{\mathrm{H}}\boldsymbol{y} = \boldsymbol{b},\ \boldsymbol{x} = A^{\mathrm{H}}\boldsymbol{y}. \tag{3.123}$$

As mentioned before, it is natural to solve the normal equations by the CG method, since $A^{\mathrm{H}}A$ and AA^{H} are Hermitian positive definite. If we apply the CG method to (3.122), then we have the CGNR method (or the CGLS method) [95] given in Algorithm 3.27.

Algorithm 3.27 The CGNR method

```
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, \boldsymbol{p}_{-1} = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Output: $\boldsymbol{x}_n$
1: for $n = 0, 1, \ldots,$ until convergence do
2:     $\boldsymbol{p}_n = A^{\mathrm{H}}\boldsymbol{r}_n + \beta_{n-1}\boldsymbol{p}_{n-1}$
3:     $\alpha_n = \frac{(A^{\mathrm{H}}\boldsymbol{r}_n, A^{\mathrm{H}}\boldsymbol{r}_n)}{(A\boldsymbol{p}_n, A\boldsymbol{p}_n)}$
4:     $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$
5:     $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n A\boldsymbol{p}_n$
6:     $\beta_n = \frac{(A^{\mathrm{H}}\boldsymbol{r}_{n+1}, A^{\mathrm{H}}\boldsymbol{r}_{n+1})}{(A^{\mathrm{H}}\boldsymbol{r}_n, A^{\mathrm{H}}\boldsymbol{r}_n)}$
7: end for
```

It readily follows from the algorithm of the CGNR method and Theorem 3.1 that the CGNR method minimizes the $A^{\mathrm{H}}A$-norm of the error, or the 2-norm of the residual vector:

$$\min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A^{\mathrm{H}}A, A^{\mathrm{H}}\boldsymbol{r}_0)} \|\boldsymbol{x} - \boldsymbol{x}_n\|_{A^{\mathrm{H}}A} = \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A^{\mathrm{H}}A, A^{\mathrm{H}}\boldsymbol{r}_0)} \|\boldsymbol{r}_n\|,$$

where $\boldsymbol{r}_n := \boldsymbol{b} - A\boldsymbol{x}_n$.

On the other hand, if we apply the CG method to (3.123), then we have the CGNE method [41] (or Craig's method) given in Algorithm 3.28.

It readily follows from the algorithm of the CGNE method and (3.16) that the CGNE method minimizes the AA^{H}-norm of the error on the linear systems $AA^{\mathrm{H}}\boldsymbol{y} = \boldsymbol{b}$:

$$\min_{\boldsymbol{y}_n \in \boldsymbol{y}_0 + \mathcal{K}_n(AA^{\mathrm{H}}, \boldsymbol{b} - AA^{\mathrm{H}}\boldsymbol{y}_0)} \|\boldsymbol{y} - \boldsymbol{y}_n\|_{AA^{\mathrm{H}}}.$$

Algorithm 3.28 The CGNE method

```
Input: $x_0 \in \mathbb{C}^N$, $\beta_{-1} = 0, p_{-1} = 0, r_0 = b - Ax_0$
Output: $x_n$
1: for $n = 0, 1, \ldots,$ until convergence do
2:     $p_n = A^{\mathrm{H}} r_n + \beta_{n-1} p_{n-1}$
3:     $\alpha_n = \frac{(r_n, r_n)}{(p_n, p_n)}$
4:     $x_{n+1} = x_n + \alpha_n p_n$
5:     $r_{n+1} = r_n - \alpha_n A p_n$
6:     $\beta_n = \frac{(r_{n+1}, r_{n+1})}{(r_n, r_n)}$
7: end for
```

It follows from the above fact and $\boldsymbol{y}_n = A^{-\mathrm{H}} \boldsymbol{x}_n$ that the CGNE method generates $\boldsymbol{x}_n$ such that

$$\begin{aligned}
&\min_{\boldsymbol{y}_n \in \boldsymbol{y}_0 + \mathcal{K}_n(AA^{\mathrm{H}}, \boldsymbol{b} - AA^{\mathrm{H}} \boldsymbol{y}_0)} \|\boldsymbol{y} - \boldsymbol{y}_n\|_{AA^{\mathrm{H}}} \\
&= \min_{A^{-\mathrm{H}} \boldsymbol{x}_n \in A^{-\mathrm{H}} \boldsymbol{x}_0 + \mathcal{K}_n(AA^{\mathrm{H}}, \boldsymbol{b} - A\boldsymbol{x}_0)} \|A^{-\mathrm{H}} \boldsymbol{x} - A^{-\mathrm{H}} \boldsymbol{x}_n\|_{AA^{\mathrm{H}}} \\
&= \min_{A^{-\mathrm{H}} \boldsymbol{x}_n \in A^{-\mathrm{H}} \boldsymbol{x}_0 + \mathcal{K}_n(AA^{\mathrm{H}}, \boldsymbol{b} - A\boldsymbol{x}_0)} \|\boldsymbol{x} - \boldsymbol{x}_n\| \\
&= \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + A^{\mathrm{H}} \mathcal{K}_n(AA^{\mathrm{H}}, \boldsymbol{b} - A\boldsymbol{x}_0)} \|\boldsymbol{x} - \boldsymbol{x}_n\| \\
&= \min_{\boldsymbol{x}_n \in \boldsymbol{x}_0 + \mathcal{K}_n(A^{\mathrm{H}} A, A^{\mathrm{H}} \boldsymbol{r}_0)} \|\boldsymbol{x} - \boldsymbol{x}_n\|,
\end{aligned}$$

where $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$. Thus, the CGNE method minimizes the 2-norm of the error on the linear systems $A\boldsymbol{x} = \boldsymbol{b}$.

The convergence of the CGNR method and the CGNE method performs very well in some special cases [64, 138]. However, since the condition number of $A^{\mathrm{H}}A$ (or AA^{H}) is twice that of A, these methods may show slow convergence from Theorem 3.2.

If the coefficient matrix is ill-conditioned, the LSQR method [144] is often preferred. This algorithm is based on the Golub–Kahan bidiagonalization process [78] and the process is obtained by applying the Lanczos process to the following augmented linear systems of the form

$$\begin{bmatrix} I & A \\ A^{\mathrm{H}} & O \end{bmatrix} \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix}.$$

We write it as $\tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$. The process begins with a unit vector:

$$\boldsymbol{w}_1 := \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{0} \end{bmatrix} = \frac{1}{\|\boldsymbol{b}\|} \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix}.$$

Let $h_{0,1} := \|\boldsymbol{b}\|$. Then we have $h_{01}\boldsymbol{u}_1 = \boldsymbol{b}$. For the first step, applying the Lanczos process (Algorithm 1.11) to $\tilde{A}$ yields

$$\tilde{\boldsymbol{w}}_2 = \tilde{A}\boldsymbol{w}_1 - \alpha_1 \boldsymbol{w}_1 = \begin{bmatrix} \boldsymbol{0} \\ A^{\mathrm{H}}\boldsymbol{u}_1 \end{bmatrix},$$
$$\boldsymbol{w}_2 = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{v}_1 \end{bmatrix} = \frac{1}{\|A^{\mathrm{H}}\boldsymbol{u}_1\|} \begin{bmatrix} \boldsymbol{0} \\ A^{\mathrm{H}}\boldsymbol{u}_1 \end{bmatrix}.$$

Using $h_{1,1} := \|A^{\mathrm{H}}\boldsymbol{u}_1\|$ gives

$$h_{1,1}\boldsymbol{v}_1 = A^{\mathrm{H}}\boldsymbol{u}_1. \tag{3.124}$$

For the second step, we have

$$\tilde{\boldsymbol{w}}_3 = \tilde{A}\boldsymbol{w}_2 - \alpha_2 \boldsymbol{w}_2 - \beta_1 \boldsymbol{w}_1 = \begin{bmatrix} A\boldsymbol{v}_1 - \beta_1 \boldsymbol{u}_1 \\ \boldsymbol{0} \end{bmatrix},$$
$$\boldsymbol{w}_3 = \begin{bmatrix} \boldsymbol{u}_2 \\ \boldsymbol{0} \end{bmatrix} = \frac{1}{\|A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1\|} \begin{bmatrix} A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1 \\ \boldsymbol{0} \end{bmatrix}.$$

Using $h_{2,1} := \|A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1\|$, the equation on $\boldsymbol{w}_3$ is rewritten as

$$h_{2,1}\boldsymbol{u}_2 = A\boldsymbol{v}_1 - h_{1,1}\boldsymbol{u}_1. \tag{3.125}$$

For the third step, it follows that

$$\tilde{\boldsymbol{w}}_4 = \tilde{A}\boldsymbol{w}_3 - \alpha_3 \boldsymbol{w}_3 - \beta_2 \boldsymbol{w}_2 = \begin{bmatrix} \boldsymbol{0} \\ A^{\mathrm{H}}\boldsymbol{u}_2 - \beta_2 \boldsymbol{v}_1 \end{bmatrix},$$
$$\boldsymbol{w}_4 = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{v}_2 \end{bmatrix} = \frac{1}{\|A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1\|} \begin{bmatrix} \boldsymbol{0} \\ A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1 \end{bmatrix}.$$

The equation on $\boldsymbol{w}_4$ is rewritten as

$$h_{2,2}\boldsymbol{v}_2 = A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1 \tag{3.126}$$

by using $h_{2,2} := \|A^{\mathrm{H}}\boldsymbol{u}_2 - h_{2,1}\boldsymbol{v}_1\|$. Hence, from (3.124)–(3.126), we have the following matrix form:

$$A\boldsymbol{v}_1 = [\boldsymbol{u}_1, \boldsymbol{u}_2] \begin{bmatrix} h_{1,1} \\ h_{2,1} \end{bmatrix} \Leftrightarrow AV_1 = U_2 B_{2,1},$$
$$A^{\mathrm{H}}[\boldsymbol{u}_1, \boldsymbol{u}_2] = [\boldsymbol{v}_1, \boldsymbol{v}_2] \begin{bmatrix} h_{1,1} & h_{2,1} \\ 0 & h_{2,2} \end{bmatrix} \Leftrightarrow A^{\mathrm{H}}U_2 = V_1 B_{2,1}^{\top} + h_{2,2}\boldsymbol{v}_2 \boldsymbol{e}_2^{\top},$$

where $U_2 = [\boldsymbol{u}_1, \boldsymbol{u}_2]$, $V_1 = \boldsymbol{v}_1$, and $B_{2,1}^\top = [h_{1,1}, h_{2,1}]$. The above recurrences correspond to the first few steps of the Golub–Kahan bidiagonalization process [78]. The algorithm of the Golub–Kahan bidiagonalization process is described in Algorithm 3.29.

Algorithm 3.29 The Golub–Kahan bidiagonalization process

```
Input: $A \in \mathbb{C}^{N\times N}, \boldsymbol{b} \in \mathbb{C}^N$
Output: $\boldsymbol{u}_i, \boldsymbol{v}_i$ $(i = 1, 2, \dots)$
1: $h_{0,1} = \|\boldsymbol{b}\|, \boldsymbol{u}_1 = \boldsymbol{b}/h_{0,1}$
2: $h_{1,1} = \|A^{\mathrm{H}}\boldsymbol{u}_1\|, \boldsymbol{v}_1 = A^{\mathrm{H}}\boldsymbol{u}_1/h_{1,1}$
3: for $n = 1, 2, \dots$ do
4:     $\tilde{\boldsymbol{u}}_{n+1} = A\boldsymbol{v}_n - h_{n,n}\boldsymbol{u}_n$
5:     $h_{n+1,n} = \|\tilde{\boldsymbol{u}}_{n+1}\|$
6:     $\boldsymbol{u}_{n+1} = \tilde{\boldsymbol{u}}_{n+1}/h_{n+1,n}$
7:     $\tilde{\boldsymbol{v}}_{n+1} = A^{\mathrm{H}}\boldsymbol{u}_{n+1} - h_{n+1,n}\boldsymbol{v}_n$
8:     $h_{n+1,n+1} = \|\tilde{\boldsymbol{v}}_{n+1}\|$
9:     $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/h_{n+1,n+1}$
10: end for
```

The Golub–Kahan bidiagonalization process can be written in matrix form

$$\begin{aligned} AV_n &= U_{n+1}B_{n+1,n}, \\ A^{\mathrm{H}}U_{n+1} &= V_nB_{n+1,n}^\top + h_{n+1,n+1}\boldsymbol{v}_{n+1}\boldsymbol{e}_{n+1}^\top, \end{aligned} \tag{3.127}$$

where

$$U_n := [\boldsymbol{u}_1, \dots, \boldsymbol{u}_n],\ V_n := [\boldsymbol{v}_1, \dots, \boldsymbol{v}_n],\ B_{n+1,n} := \begin{bmatrix} h_{1,1} & & \\ h_{2,1} & \ddots & \\ & \ddots & h_{n,n} \\ & & h_{n+1,n} \end{bmatrix}.$$

It is clear from the bidiagonalization process that U_n and V_n satisfy the following properties:

$$U_n^{\mathrm{H}}U_n = V_n^{\mathrm{H}}V_n = I_n. \tag{3.128}$$

We are now ready for the derivation of the LSQR method. 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

$$\boldsymbol{r}_n = \boldsymbol{b} - AV_n\boldsymbol{y}_n. \tag{3.129}$$

Substituting (3.127) into (3.129) leads to

$$\begin{aligned} \boldsymbol{r}_n &= \boldsymbol{b} - U_{n+1} B_{n+1,n} \boldsymbol{y}_n \\ &= U_{n+1}(h_{0,1}\boldsymbol{e}_1 - B_{n+1,n}\boldsymbol{y}_n). \end{aligned}$$

From the property (3.128) it follows that

$$\min_{\boldsymbol{y}_n \in \mathbb{C}^n} \|\boldsymbol{r}_n\| = \min_{\boldsymbol{y}_n \in \mathbb{C}^n} \|h_{0,1}\boldsymbol{e}_1 - B_{n+1,n}\boldsymbol{y}_n\|. \tag{3.130}$$

Hence, similar to the MINRES method in Sect. 3.1.3, the solution of $\boldsymbol{y}_n$ in (3.130) can be obtained by using Givens rotations, and following the derivation of the MINRES method, we have the LSQR algorithm described in Algorithm 3.30.

Algorithm 3.30 The LSQR method

```
Input: $A \in \mathbb{C}^{N\times N}, \boldsymbol{b} \in \mathbb{C}^N$
Output: $\boldsymbol{x}_n$
1: $\boldsymbol{x}_0 = \boldsymbol{0}$, $h_{0,1} = \|\boldsymbol{b}\|$, $\boldsymbol{u}_1 = \boldsymbol{b}/h_{0,1}$
2: $h_{1,1} = \|A^{\mathrm{H}}\boldsymbol{u}_1\|$, $\boldsymbol{v}_1 = A^{\mathrm{H}}\boldsymbol{u}_1/h_{1,1}$
3: for $n = 1, 2, \dots$ do
4:     (G.-K. bidiagonalization process)
5:     $\tilde{\boldsymbol{u}}_{n+1} = A\boldsymbol{v}_n - h_{n,n}\boldsymbol{u}_n$
6:     $h_{n+1,n} = \|\tilde{\boldsymbol{u}}_{n+1}\|$
7:     $\boldsymbol{u}_{n+1} = \tilde{\boldsymbol{u}}_{n+1}/h_{n+1,n}$
8:     $\tilde{\boldsymbol{v}}_{n+1} = A^{\mathrm{H}}\boldsymbol{u}_{n+1} - h_{n+1,n}\boldsymbol{v}_n$
9:     $h_{n+1,n+1} = \|\tilde{\boldsymbol{v}}_{n+1}\|$
10:    $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/h_{n+1,n+1}$
11:    (Givens rotations)
12:    for $i = \max\{1, n-1\}, \dots, n-1$ do
13:        $\begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -\bar{s}_i & c_i \end{bmatrix} \begin{bmatrix} t_{i,n} \\ t_{i+1,n} \end{bmatrix}$
14:    end for
15:    $c_n = \frac{|t_{n,n}|}{\sqrt{|t_{n,n}|^2 + |t_{n+1,n}|^2}}$
16:    $\bar{s}_n = \frac{t_{n+1,n}}{t_{n,n}} c_n$
17:    $t_{n,n} = c_n t_{n,n} + s_n t_{n+1,n}$
18:    $t_{n+1,n} = 0$
19:    $\begin{bmatrix} g_n \\ g_{n+1} \end{bmatrix} = \begin{bmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{bmatrix} \begin{bmatrix} g_n \\ 0 \end{bmatrix}$
20:    (Update $\boldsymbol{x}_n$)
21:    $\boldsymbol{p}_n = (\boldsymbol{v}_n - t_{n-1,n}\boldsymbol{p}_{n-1})/t_{n,n}$
22:    $\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + g_n\boldsymbol{p}_n$
23:    (Check convergence)
24:    if $|g_{n+1}|/\|\boldsymbol{b}\| \le \epsilon$, then stop.
25: end for
```

3.5 Preconditioning Techniques

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

$$K^{-1}A\boldsymbol{x} = K^{-1}\boldsymbol{b} \tag{3.131}$$

has the same solution as that of the original one. Hence, we can obtain the solution of the original linear system by applying Krylov subspace methods to the transformed linear system. The matrix K is called a *preconditioner* and a favorable preconditioner satisfies the following properties:

- $K^{-1}A \approx I$.
- $K^{-1}\boldsymbol{z}$ is readily obtained for any vector $\boldsymbol{z}$.

If A is Hermitian positive definite (h.p.d.), the transformed system (3.131) is not useful in practice because the coefficient matrix $K^{-1}A$ is not h.p.d. any longer. To preserve the h.p.d. structure of the coefficient matrix, we split the preconditioner into $K = K_1K_1^{\mathrm{H}}$ and then transform the linear system into

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

The above coefficient matrix $K_1^{-1}AK_1^{-\mathrm{H}}$ is h.p.d. Hence, Krylov subspace methods for h.p.d. linear systems, e.g., the CG method, can be applied to it. If the coefficient matrix A is non-Hermitian, then the corresponding preconditioning is

$$K_1^{-1}AK_2^{-1}(K_2\boldsymbol{x}) = K_1^{-1}\boldsymbol{b}, \tag{3.132}$$

where $K = K_1K_2$. The above form is called *left and right preconditioning*. When $K_1 = I$, we have

$$AK^{-1}(K\boldsymbol{x}) = \boldsymbol{b}. \tag{3.133}$$

This preconditioning is referred to as *right preconditioning*, and when $K_2 = I$ we have (3.131), which is referred to as *left preconditioning*.

The right preconditioning is often used for the GMRES method and the GCR method. In this case, these methods find approximate solutions such that the 2-norm of the residuals for the original linear system $A\boldsymbol{x} = \boldsymbol{b}$ is minimized. On the other hand, if the left preconditioning is used, then these methods find approximate solutions such that the 2-norm of the residuals for the transformed linear system $K^{-1}A\boldsymbol{x} = K^{-1}\boldsymbol{b}$ is minimized.

For the overviews of preconditioning techniques, see, e.g., [20, 146]. See also preconditioned algorithms of bi-Lanczos-type Krylov subspace methods [109]. Preconditioned algorithms of Arnoldi-type Krylov subspace methods, the GMRES method for singular linear systems, have been studied in [49, 94, 136, 206].

3.5.1 Incomplete Matrix Decomposition Preconditioners

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

One of the simplest incomplete factorizations is the *D-ILU preconditioner* that was proposed by Pommerell (see, e.g., [147, pp.77–78]). 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 $\mathrm{diag}(K) = D_A$, i.e., $(K)_{ii} = (D_A)_{ii}$ for all i. Hence, only the computation of D is required.

The algorithm of the D-ILU preconditioner is described in Algorithm 3.31, where $N_0(A)$ is the index set of nonzero elements of matrix A, i.e.,

$$N_0(A) := \{(i, j) \ : \ a_{i,j} \neq 0\}.$$

After running Algorithm 3.31, we obtain $d_1, d_2, \dots, d_N$. Then setting

$$D = \begin{bmatrix} d_1^{-1} & & \\ & \ddots & \\ & & d_N^{-1} \end{bmatrix}$$

yields the D-ILU preconditioner of the form $K = (D + L_A)D^{-1}(D + U_A)$.

Algorithm 3.31 The D-ILU decomposition

```
Input: $A \in \mathbb{C}^{N \times N}$, $N_0(A)$ (index set of nonzeros of $A$)
Output: $D^{-1} = \mathrm{diag}(d_1, d_2, \dots, d_N)$
1: for $i = 1, 2, \dots, N$ do
2:     $d_i = a_{i,i}$
3: end for
4: for $i = 1, 2, \dots, N$ do
5:     $d_i = 1/d_i$
6:     for $j = i + 1, i + 2, \dots, N$ do
7:         if $(i, j) \in N_0(A)$ and $(j, i) \in N_0(A)$, then $d_j = d_j - a_{j,i} d_i a_{i,j}$
8:     end for
9: end for
```

The implementation of the preconditioned Krylov subspace methods is explained next. D-ILU preconditioned Krylov subspace methods need to compute the multiplication of K^{-1} and a vector $\boldsymbol{v}$, i.e., $w = K^{-1}\boldsymbol{v} = (D + U_A)^{-1}D(D + L_A)^{-1}\boldsymbol{v}$, and the vector w is computed as follows:

1. solve $(D + L_A)\boldsymbol{y} = \boldsymbol{v}$ by forward substitution;
2. compute $\boldsymbol{z} = D\boldsymbol{y}$;
3. solve $(D + U_A)\boldsymbol{w} = \boldsymbol{z}$ by back substitution.

Note that since the D-ILU preconditioner explicitly contains the off-diagonal parts of the original matrix, Eisenstat's trick [53] (see also Section 3.5.4) can be used to give a more efficient implementation of the D-ILU preconditioned Krylov subspace methods.

One of the most well-known and powerful incomplete LU decompositions is given by Meijerink and van der Vorst [127]. This factorization is referred to as ILU(0), and the explanation is given next.

The *ILU(0) preconditioner* is based on the LU decomposition of A as described in Algorithm 1.1. Algorithm 3.32 is a slightly modified version of Algorithm 1.1 for ease of presentation of the ILU(0) preconditioner.

The input in Algorithm 3.32 is A as described below.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & A_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & A_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix},$$

and the output is $a_{i,j}$, where

$$A = LU,$$

with

$$L = \begin{bmatrix} 1 & & & \\ a_{2,1} & \ddots & & \\ \vdots & \ddots & \ddots & \\ a_{N,1} & \cdots & a_{N,N-1} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ & \ddots & \ddots & \vdots \\ & & \ddots & a_{N-1,N} \\ & & & a_{N,N} \end{bmatrix}.$$

Notice that the input $a_{i,j}$ is the original matrix and the output $a_{i,j}$ is overwritten for the matrices L and U.

Algorithm 3.32 The LU decomposition (a slightly modified version of Algorithm 1.1)

```
1: for $i = 2, 3, \ldots, N$ do
2:   for $k = 1, 2, \ldots, i-1$ do
3:     $a_{i,k} = a_{i,k}/a_{k,k}$
4:     for $j = k+1, k+2, \ldots, N$ do
5:       $a_{i,j} = a_{i,j} - a_{i,k} \times a_{k,j}$
6:     end for
7:   end for
8: end for
```

The ILU(0) preconditioner is the LU decomposition in Algorithm 3.32 without computing (i, j) element if $(i, j) \notin N_0(A)$. The algorithm of the ILU(0) preconditioner is described in Algorithm 3.33.

From Algorithm 3.33, the number of nonzeros in the factorized matrix is usually the same as that in the original matrix. Hence, the upper and lower matrices of the

Algorithm 3.33 The ILU(0) preconditioner

```
1: for $i = 2, 3, \ldots, N$ do
2:   for $k = 1, 2, \ldots, i-1$ do
3:     $\boxed{\text{if } (i,k) \in N_0(A), \text{ then}}$ $a_{i,k} = a_{i,k}/a_{k,k}$
4:     for $j = k+1, k+2, \ldots, N$ do
5:       $\boxed{\text{if } (i,j) \in N_0(A), \text{ then}}$ $a_{i,j} = a_{i,j} - a_{i,k} \times a_{k,j}$
6:     end for
7:   end for
8: end for
```

ILU(0) preconditioner are sparse matrices when A is a sparse matrix. This fact is important in terms of both memory and computational costs.

As an extension of the ILU(0) preconditioner, the *ILU*(p) *preconditioner* is also proposed in the same paper [127], where p denotes the level of fill-in. The definition of the level of fill-in is given as follows: let the initial level of fill-in in position (i, j) for $a_{i,j} \neq 0$ be zero, and let the initial level of fill-in in position (i, j) for $a_{i,j} = 0$ be ∞. Then the level of fill-in in position (i, j) is defined by

$$\text{Level}_{ij} := \min_{1 \le k \le \min\{i,j\}} \{\text{Level}_{ik} + \text{Level}_{kj} + 1\}. \tag{3.134}$$

For example, consider

$$A = \begin{bmatrix} * & * & 0 & 0 & 0 & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 & * & 0 & 0 \\ 0 & * & * & * & 0 & 0 & 0 & * & 0 \\ 0 & 0 & * & * & * & 0 & 0 & 0 & * \\ 0 & 0 & 0 & * & * & * & 0 & 0 & 0 \\ * & 0 & 0 & 0 & * & * & * & 0 & 0 \\ 0 & * & 0 & 0 & 0 & * & * & * & 0 \\ 0 & 0 & * & 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & 0 & 0 & 0 & * & * \end{bmatrix}, \tag{3.135}$$

where the symbol "*" is nonzero. Since the nonzero elements of A have level zero and the zeros in A have level ∞, we have the following matrix whose elements are levels of fill-in:

$$\begin{bmatrix} 0 & 0 & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ 0 & 0 & 0 & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & 0 & 0 & 0 & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & 0 & 0 & 0 & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & 0 & 0 & 0 & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & 0 & 0 & 0 & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty & 0 & 0 & 0 & \infty \\ \infty & \infty & 0 & \infty & \infty & \infty & 0 & 0 & 0 \\ \infty & \infty & \infty & 0 & \infty & \infty & \infty & 0 & 0 \end{bmatrix}, \tag{3.136}$$

Applying (3.134) to (3.136), we have the following matrix whose level of the (i, j) position is 1:

$$\begin{bmatrix} 0 & 0 & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ 0 & 0 & 0 & \infty & \infty & 1 & 0 & \infty & \infty \\ \infty & 0 & 0 & 0 & \infty & \infty & 1 & 0 & \infty \\ \infty & \infty & 0 & 0 & 0 & \infty & \infty & 1 & 0 \\ \infty & \infty & \infty & 0 & 0 & 0 & \infty & \infty & 1 \\ 0 & 1 & \infty & \infty & 0 & 0 & 0 & \infty & \infty \\ \infty & 0 & 1 & \infty & \infty & 0 & 0 & 0 & \infty \\ \infty & \infty & 0 & 1 & \infty & \infty & 0 & 0 & 0 \\ \infty & \infty & \infty & 0 & 1 & \infty & \infty & 0 & 0 \end{bmatrix},$$

Applying (3.134) to the above matrix, we can determine the (i, j) element with $\mathrm{Level}_{ij} = 2$.

$$\begin{bmatrix} 0 & 0 & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ 0 & 0 & 0 & \infty & \infty & 1 & 0 & \infty & \infty \\ \infty & 0 & 0 & 0 & \infty & 2 & 1 & 0 & \infty \\ \infty & \infty & 0 & 0 & 0 & \infty & 2 & 1 & 0 \\ \infty & \infty & \infty & 0 & 0 & 0 & \infty & 2 & 1 \\ 0 & 1 & 2 & \infty & 0 & 0 & 0 & \infty & 2 \\ \infty & 0 & 1 & 2 & \infty & 0 & 0 & 0 & \infty \\ \infty & \infty & 0 & 1 & 2 & \infty & 0 & 0 & 0 \\ \infty & \infty & \infty & 0 & 1 & 2 & \infty & 0 & 0 \end{bmatrix},$$

Applying (3.134) to the above matrix, all the levels of fill-in are determined and the matrix is given by

$$\begin{bmatrix} 0 & 0 & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ 0 & 0 & 0 & \infty & \infty & 1 & 0 & \infty & \infty \\ \infty & 0 & 0 & 0 & \infty & 2 & 1 & 0 & \infty \\ \infty & \infty & 0 & 0 & 0 & 3 & 2 & 1 & 0 \\ \infty & \infty & \infty & 0 & 0 & 0 & 3 & 2 & 1 \\ 0 & 1 & 2 & 3 & 0 & 0 & 0 & 3 & 2 \\ \infty & 0 & 1 & 2 & 3 & 0 & 0 & 0 & 3 \\ \infty & \infty & 0 & 1 & 2 & 3 & 0 & 0 & 0 \\ \infty & \infty & \infty & 0 & 1 & 2 & 3 & 0 & 0 \end{bmatrix}, \tag{3.137}$$

Let $N_p(A)$ be the index set that corresponds to $\text{Level}_{ij} \le p$, i.e.,

$$N_p(A) := \{(i,j) \ : \ \text{Level}_{ij} \le p\}.$$

For example, from (3.137), $N_p(A)$ of (3.135) with $p = 1, 2, 3$ is given as follows:

$$N_1(A) = N_0(A) \cup \{(2,6), (3,7), (4,8), (5,9), (6,2), (7,3), (8,4), (9,5)\},$$
$$N_2(A) = N_1(A) \cup \{(3,6), (4,7), (5,8), (6,9), (6,3), (7,4), (8,5), (9,6)\},$$
$$N_3(A) = N_2(A) \cup \{(4,6), (5,7), (6,8), (7,9), (6,4), (7,5), (8,6), (9,7)\}.$$

The ILU(p) preconditioner is the LU decomposition in Algorithm 3.32 without computing the (i,j) element if $(i,j) \notin N_p(A)$. Now, the algorithm of the ILU(p) preconditioner is given in Algorithm 3.34.

Algorithm 3.34 The ILU(p) preconditioner

```
1: for $i = 2, 3, \ldots, N$ do
2:   for $k = 1, 2, \ldots, i-1$ do
3:     $\boxed{\text{if } (i,k) \in N_p(A), \text{ then}}$ $a_{i,k} = a_{i,k}/a_{k,k}$
4:     for $j = k+1, k+2, \ldots, N$ do
5:       $\boxed{\text{if } (i,j) \in N_p(A), \text{ then}}$ $a_{i,j} = a_{i,j} - a_{i,k} \times a_{k,j}$
6:     end for
7:   end for
8: end for
```

Another successful variant of ILU is an ILU with a threshold approach proposed by Saad [150]. This factorization is referred to as ILUT(p,τ), where p is used for saving memory and τ is a criterion for dropping elements, i.e., forcing the small elements to be zero in magnitude.

3.5.2 Approximate Inverse Preconditioners

In the previous preconditioners, their performance depends 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 [82] attempt to minimize the following Frobenius norm:

$$\min_K \|I - AK\|_{\mathrm{F}},$$

where $\|A\|_F = \sqrt{\sum_{i,j} a_{i,j}^2}$. Since the above minimization can be written as

$$\min_K \left\| \left[\boldsymbol{e}_1 - A\boldsymbol{k}_1, \boldsymbol{e}_2 - A\boldsymbol{k}_2, \dots, \boldsymbol{e}_N - A\boldsymbol{k}_N \right] \right\|_{\mathrm{F}},$$

we have N independent least-squares problems

$$\min_{\boldsymbol{k}_i} \|\boldsymbol{e}_i - A\boldsymbol{k}_i\|, \quad i = 1, \dots, N.$$

Hence, the construction of this preconditioner is very suitable for parallel computing.

Another outstanding idea for approximate inverses was given by Benzi and Tůma [24]. This idea is finding nonsingular matrices V and W such that

$$W^{\mathrm{H}}AV = D. \tag{3.138}$$

Then, it follows from $(W^{\mathrm{H}}AV)^{-1} = V^{-1}A^{-1}W^{-\mathrm{H}} = D^{-1}$ that the inverse of the matrix A is given as

$$A^{-1} = VD^{-1}W^{\mathrm{H}}.$$

Since the equation (3.138) implies $(\boldsymbol{w}_i, A\boldsymbol{v}_j) = 0$ for $i \neq j$, V and W are computed by an A-biorthogonalization process. The algorithm, referred to as the *Sparse Approximate Inverse (AINV) preconditioner*, is given in Algorithm 3.35. In the algorithm, $\boldsymbol{a}_i$ and $\boldsymbol{c}_i$ denote the ith column of A and A^{H} respectively. From Algorithm 3.35, we see that the algorithm uses a MGS-style A-biorthogonalization process and is used for non-Hermitian matrices. For Hermitian and normal matrices, the corresponding approximate inverse preconditioners have been developed, see [21, 22].

3.5.3 Matrix Polynomial Preconditioners

We have described two types of preconditioners. One of them is approximating the coefficient matrix $A \approx M$, and $M^{-1}\boldsymbol{v}$ can easily be computed. The other one is approximating the inverse of the coefficient matrix $A^{-1} \approx M$. 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 a *polynomial preconditioner*.

Algorithm 3.35 The AINV preconditioner

```
Input: $\boldsymbol{w}_i^{(0)} = \boldsymbol{v}_i^{(0)} = \boldsymbol{e}_i,\ 1 \le i \le N$
1: for $i = 1, \dots, N$ do
2:     for $j = i, \dots, N$ do
3:         $p_j^{(i-1)} = \boldsymbol{a}_i^{\mathrm{H}} \boldsymbol{v}_j^{(i-1)},\quad q_j^{(i-1)} = \boldsymbol{c}_i^{\mathrm{H}} \boldsymbol{w}_j^{(i-1)}$
4:     end for
5:     if $i = N$, then exit.
6:     for $j = i+1, \dots, N$ do
7:         $\boldsymbol{v}_j^{(i)} = \boldsymbol{v}_j^{(i-1)} - \left(\frac{p_j^{(i-1)}}{p_i^{(i-1)}}\right)\boldsymbol{v}_i^{(i-1)},\quad \boldsymbol{w}_j^{(i)} = \boldsymbol{w}_j^{(i-1)} - \left(\frac{q_j^{(i-1)}}{q_i^{(i-1)}}\right)\boldsymbol{w}_i^{(i-1)}$
8:             $v_{k,j}^{(i)} = 0$ if $|\boldsymbol{v}_{k,j}^{(i)}| <$ Tol, $1 \le k \le N,\quad w_{k,j}^{(i)} = 0$ if $|\boldsymbol{w}_{k,j}^{(i)}| <$ Tol, $1 \le k \le N$
9:     end for
10: end for
11: $\boldsymbol{z}_i = \boldsymbol{z}_i^{(i-1)},\ \boldsymbol{w}_i = \boldsymbol{w}_i^{(i-1)},\ p_i = p_i^{(i-1)},\ 1 \le i \le N$
12: $V = [\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_N],\ V = [\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_N],\ D = \mathrm{diag}(p_1, p_2, \dots, p_N)$
```

Among the most well-known polynomial preconditioners are the *Neumann expansion* and *Euler expansion*. Let A be of the form $I - B$ with $\rho(B) < 1$. Then, we can write the inverse matrix of A as

$$
\begin{aligned}
A^{-1} &= \sum_{i=0}^{\infty} B^i \quad \text{(Neumann expansion)} \\
&= \prod_{i=0}^{\infty} (I + B^{2^i}). \quad \text{(Euler expansion)}
\end{aligned}
$$

Hence, from the above expansions, we can readily obtain and moreover control an approximate inverse of A by using the low order terms of the Neumann (or Euler) expansion. This approach was studied in [50], and see also [116, 198].

3.5.4 Preconditioners Based on Stationary Iterative Methods

The idea of stationary iterative methods in Sect. 1.6 can also be used for constructing efficient preconditioners for Krylov subspace methods. In this section, Jacobi, Gauss–Seidel, and SOR-type preconditioners are described.

The *Jacobi preconditioner* (or the *diagonal preconditioner*) is the simplest preconditioner and is usually more effective than solving the original linear systems $A\boldsymbol{x} = \boldsymbol{b}$ by unpreconditioned Krylov subspace methods. The Jacobi preconditioner is constructed by using the diagonal entries of the coefficient matrix as follows:

$$K_J := \mathrm{diag}(a_{1,1}, a_{2,2}, \dots, a_{n,n}).$$

If the matrix A is a diagonal matrix, then $K_J^{-1}A = I$. This means Jacobi preconditioned Krylov subspace methods obtain the solution within only one iteration step. In general, there is an approach to choosing diagonal matrix D as a preconditioner so that the condition number of $D^{-1}A$ is as small as possible. The analysis of the condition numbers of $D^{-1}A$ is found in [59, 194].

The *symmetric Gauss–Seidel (SGS) preconditioner* uses more information of A than the Jacobi preconditioner. The SGS preconditioner is named after the Gauss–Seidel method, which is one of the stationary iterative methods in Sect. 1.6. The preconditioner is defined as

$$K_{SGS} := (D - L)D^{-1}(D - U),$$

where $A := D - L - U$. If all the diagonal elements of A are scaled to one, then we have a simpler preconditioner:

$$K_{SGS} = (I - L)(I - U).$$

In this case, it is known that there is an efficient implementation as follows: when using the preconditioner K_{SGS}, we have the transformed linear systems of the form

$$(I - L)^{-1}A(I - U)^{-1}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \quad \tilde{\boldsymbol{x}} = (I - U)\boldsymbol{x}, \quad \tilde{\boldsymbol{b}} = (I - L)^{-1}\boldsymbol{b}.$$

Hence, applying Krylov subspace methods to the transformed linear systems, 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 at each iteration step, this usually leads to about double computational cost per iteration. However, recalling $A = I - L - U$, it follows that

$$\begin{aligned}(I-L)^{-1}A(I-U)^{-1} &= (I-L)^{-1}(I-L-U)(I-U)^{-1}\\ &= (I-L)^{-1}[(I-L)+(I-U-I)](I-U)^{-1}\\ &= (I-L)^{-1}[(I-L)(I-U)^{-1}+I-(I-U)^{-1}]\\ &= (I-U)^{-1}+(I-L)^{-1}[I-(I-U)^{-1}].\end{aligned}$$

Thus, we obtain

$$\begin{aligned}(I-L)^{-1}A(I-U)^{-1}\boldsymbol{z} &= (I-U)^{-1}\boldsymbol{z}+(I-L)^{-1}[I-(I-U)^{-1}]\boldsymbol{z}\\ &= \boldsymbol{t}+(I-L)^{-1}(\boldsymbol{z}-\boldsymbol{t}),\end{aligned}$$

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 [53].

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}. *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

$$\begin{aligned}\tilde{A}\boldsymbol{z} &= (\tilde{D}-L)^{-1}A(\tilde{D}-U)^{-1}\boldsymbol{z}\\ &= (\tilde{D}-L)^{-1}[(\tilde{D}-L)+(D-2\tilde{D})+(\tilde{D}-U)](\tilde{D}-U)^{-1}\boldsymbol{z}\\ &= (\tilde{D}-U)^{-1}\boldsymbol{z}+(\tilde{D}-L)^{-1}(D-2\tilde{D})(\tilde{D}-U)^{-1}\boldsymbol{z}+(\tilde{D}-L)^{-1}\boldsymbol{z}\\ &= \boldsymbol{t}+(\tilde{D}-L)^{-1}[(D-2\tilde{D})\boldsymbol{t}+\boldsymbol{z}],\end{aligned}$$

where $\boldsymbol{t} = (\tilde{D}-U)^{-1}\boldsymbol{z}$.

3.5.5 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 improve the performance of preconditioners, e.g., apply Krylov subspace methods to the following systems with a reordering matrix P:

$$K^{-1}PAP^{\top}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}},$$

where $\tilde{x} = P\boldsymbol{x}$, $\tilde{\boldsymbol{b}} = K^{-1}P\boldsymbol{b}$. From this idea, we can use many reordering techniques such as Cuthill–Mckee (CM) [42], Reverse Cuthill–Mckee (RCM) [75], Nested Dissection (ND) [76], and Minimum Degree (MD) [77]. For these algorithms, see also [51, 123, 151]. CM and RCM decrease the bandwidth of a matrix, MD reduces the number of fill-ins of a matrix, and ND generates an approximate block diagonal matrix and is suitable for parallel computing.

For symmetric definite matrices, Duff and Meurant gave important results on the effects of reorderings that reordering techniques for direct solvers did not enhance the quality of preconditioners [52].

For nonsymmetric matrices, 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 [151, p.334]. On the other hand, Benzi et al. [23] studied the effects of reorderings on ILU-type preconditioners and obtained the result that RCM dramatically enhanced the preconditioner in a case where the coefficient matrix is highly nonsymmetric. Similar results of the effects on approximate inverse preconditioners are also discussed in [25].

Chapter 4
Applications to Shifted Linear Systems

Abstract As seen in Sect. 2.2 (computational physics) and Sect. 2.3 (machine learning), one needs to solve a set of linear systems of the form

$$(A + \sigma_k I)\boldsymbol{x}^{(k)} = \boldsymbol{b}, \quad k = 1, 2, \ldots, m,$$

where σ_k is a scalar, which are called *shifted linear systems*. When using the LU decomposition in Sect. 1.3.1, m times LU decompositions are required. Similarly, when using stationary iterative methods in Sect. 1.6, solving m linear systems, i.e., $(L + D + \sigma_k I)\boldsymbol{z} = \boldsymbol{v}$ for all k, is required. To solve these linear systems, one can apply a suitable Krylov subspace method to each linear system. On the other hand, if the initial residual vector with respect to the ith linear system and the initial residual vector with respect to the jth linear system are collinear, i.e., the angle between the two initial residual vectors is 0 or π, the generated Krylov subspaces become the same, which is referred to as the shift-invariance property of Krylov subspaces. This means that we can share the information of only one Krylov subspace to solve all the shifted linear systems, leading to efficient computations of shifted linear systems. In this chapter, Krylov subspace methods using the shift-invariance property are derived from the Krylov subspace methods in Chap. 3.

4.1 Shifted Linear Systems

The Krylov subspace methods in Chap. 3 are attractive for solving the following linear systems:

$$A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}, \quad k = 1, 2, \ldots, m, \tag{4.1}$$

where $A^{(k)}$ is a nonsingular matrix of the form

$$A^{(k)} := A + \sigma_k I, \tag{4.2}$$

T. Sogabe, *Krylov Subspace Methods for Linear Systems*,
Springer Series in Computational Mathematics 60,
https://doi.org/10.1007/978-981-19-8532-4_4

with I being the identity matrix and $\sigma_k \in \mathbb{C}$. The linear systems (4.1) are referred to as *shifted linear systems*. For simplicity, we also use the following notation:

$$(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}.$$

Here $\alpha^\sigma, \beta^\sigma, \ldots$ and $\boldsymbol{a}^\sigma, \boldsymbol{b}^\sigma, \ldots$ denote scalars and vectors related to the shifted linear system $(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}$. Note that α^σ is not the α to the power of σ.

If a Krylov subspace method with the initial guess $\boldsymbol{x}_0 = \boldsymbol{0}$ is applied to (4.1), then from (1.22) the Krylov subspace method finds the approximate solutions of (4.1) over the following subspaces:

$$\boldsymbol{x}_n^{(k)} \in \mathcal{K}_n(A^{(k)}, \boldsymbol{b}), \quad k = 1, 2, \ldots, m. \tag{4.3}$$

From the definition of Krylov subspace (1.20), the following *shift invariance property* holds true:

$$\mathcal{K}_n(A, \boldsymbol{b}) = \mathcal{K}_n(A^{(k)}, \boldsymbol{b}). \tag{4.4}$$

This can be easily shown by induction and the definition of a Krylov subspace. As an example, for the case $n = 3$ it follows that

$$\begin{aligned}
\mathcal{K}_3(A + \sigma I, \boldsymbol{b}) &= \{c_1\boldsymbol{b} + c_2(A + \sigma I)\boldsymbol{b} + c_3(A + \sigma I)^2\boldsymbol{b} \, : \, c_1, c_2, c_3 \in \mathbb{C}\} \\
&= \{(c_1 + c_2\sigma + c_3\sigma^2)\boldsymbol{b} + (c_2 + 2c_3\sigma)A\boldsymbol{b} + c_3A^2\boldsymbol{b} \, : \, c_1, c_2, c_3 \in \mathbb{C}\} \\
&= \{d_1\boldsymbol{b} + d_2A\boldsymbol{b} + d_3A^2\boldsymbol{b} \, : \, d_1, d_2, d_3 \in \mathbb{C}\} \\
&= \mathcal{K}_3(A, \boldsymbol{b}).
\end{aligned}$$

In the third equation, the relation between c_i and d_i is given by

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 1 & \sigma & \sigma^2 \\ 0 & 1 & 2\sigma \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}.$$

Since the 3×3 matrix is nonsingular, for any d_1, d_2, d_3 there exist c_1, c_2, c_3. Thus the third equation holds true. From this example, we see that the shift invariance property can be slightly extended as follows:

$$\mathcal{K}_n(A, \boldsymbol{v}) = \mathcal{K}_n(A^{(k)}, \boldsymbol{w}), \quad \boldsymbol{v} = c\boldsymbol{w}.$$

If $\boldsymbol{v} = c\boldsymbol{w}$ holds for some c, the two vectors are referred to as *collinear*.

Now from the shift invariance property (4.4), it follows that (4.3) can be rewritten as

$$\boldsymbol{x}_n^{(k)} \in \mathcal{K}_n(A, \boldsymbol{b}), \quad k = 1, 2, \ldots, m.$$

This means that only one Krylov subspace, $\mathcal{K}_n(A, \boldsymbol{b})$, is required for obtaining the approximate solutions $\boldsymbol{x}_n^{(1)}, \boldsymbol{x}_n^{(2)}, \dots, \boldsymbol{x}_n^{(m)}$. Note that if the shift-invariance property does not hold, m different Krylov subspaces are required.

In the following sections, we will see how to utilize the shift-invariance property and how this property is incorporated into Krylov subspace methods.

4.2 Shifted Hermitian Linear Systems

Throughout this section, the coefficient matrix $A^{(k)}$ in (4.1) is assumed to be Hermitian,[1] i.e., $A^{(k)} = (A^{(k)})^{\mathrm{H}}$.

4.2.1 The Shifted CG Method

In this subsection, the algorithm of the CG method utilizing the shift-invariance property (4.4) is described. The CG method applied to the shifted linear systems was considered in [207, 208], where all the Lanczos vectors have to be stored. In what follows, the memory-efficient algorithm based on [67, 70] is described.

It follows from Corollary 3.1, (3.15), and (4.4) that applying the CG method (Algorithm 3.1) with $\boldsymbol{x}_0 = \boldsymbol{x}_0^{(k)} = \boldsymbol{0}$ to $A\boldsymbol{x} = \boldsymbol{b}$ and $A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}$ produces residual vectors $\boldsymbol{r}_n$ and $\boldsymbol{r}_n^{(k)} := \boldsymbol{b} - A^{(k)}\boldsymbol{x}_n^{(k)}$ with

$$\boldsymbol{r}_n,\ \boldsymbol{r}_n^{(k)} \in \mathcal{K}_{n+1}(A, \boldsymbol{b}) \perp \mathcal{K}_n(A, \boldsymbol{b}), \quad k = 1, 2, \dots, m.$$

This means that $\boldsymbol{r}_n, \boldsymbol{r}_n^{(k)} \in \mathcal{K}_{n+1}(A, \boldsymbol{b}) \cap \mathcal{K}_n(A, \boldsymbol{b})^{\perp}$, where $\mathcal{K}_n(A, \boldsymbol{b})^{\perp}$ is the orthogonal complement of $\mathcal{K}_n(A, \boldsymbol{b})$. Since $\mathcal{K}_n(A, \boldsymbol{b}) \subset \mathcal{K}_{n+1}(A, \boldsymbol{b})$,

$$\dim\left(\mathcal{K}_{n+1}(A, \boldsymbol{b}) \cap \mathcal{K}_n(A, \boldsymbol{b})^{\perp}\right) = 1. \tag{4.5}$$

Therefore $\boldsymbol{r}_n$ and all $\boldsymbol{r}_n^{(k)}$'s belong to the same one-dimensional subspace of $\mathbb{C}^N$, i.e., there exists scalar values $\pi_n^{(k)} \in \mathbb{C}$ such that

$$\boldsymbol{r}_n = \pi_n^{(k)} \boldsymbol{r}_n^{(k)}, \quad k = 1, 2, \dots, m. \tag{4.6}$$

The relation (4.6) implies that if $\pi_n^{(k)}$ is known, all the residuals $\boldsymbol{r}_n^{(k)}$ can be produced by the residual $\boldsymbol{r}_n$. In what follows, $\pi_n^{(k)}$ is determined.

From (3.18), the residual vector $\boldsymbol{r}_{n+1}$ can be written as

$$\boldsymbol{r}_{n+1} = \left(1 + \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n - \alpha_n A\right)\boldsymbol{r}_n - \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n \boldsymbol{r}_{n-1}. \tag{4.7}$$

[1] Since $A^{(k)}$ is Hermitian, it follows that $\sigma \in \mathbb{R}$.

Similarly, the residual vector $\boldsymbol{r}_{n+1}^{(k)}$ can also be written as

$$\boldsymbol{r}_{n+1}^{(k)} = \left\{1 + \frac{\beta_{n-1}^{(k)}}{\alpha_{n-1}^{(k)}}\alpha_n^{(k)} - \alpha_n^{(k)}(A + \sigma_k I)\right\}\boldsymbol{r}_n^{(k)} - \frac{\beta_{n-1}^{(k)}}{\alpha_{n-1}^{(k)}}\alpha_n^{(k)}\boldsymbol{r}_{n-1}^{(k)}. \tag{4.8}$$

Substituting (4.6) into (4.8) yields

$$\boldsymbol{r}_{n+1} = \left\{1 + \frac{\beta_{n-1}^{(k)}}{\alpha_{n-1}^{(k)}}\alpha_n^{(k)} - \alpha_n^{(k)}(A + \sigma_k I)\right\}\frac{\pi_{n+1}^{(k)}}{\pi_n^{(k)}}\boldsymbol{r}_n - \frac{\beta_{n-1}^{(k)}\alpha_n^{(k)}\pi_{n+1}^{(k)}}{\alpha_{n-1}^{(k)}\pi_{n-1}^{(k)}}\boldsymbol{r}_{n-1}. \tag{4.9}$$

Comparing the coefficients of $A\boldsymbol{r}_n$, $\boldsymbol{r}_n$, and $\boldsymbol{r}_{n-1}$ in (4.7) and (4.9) yields

$$\alpha_n^{(k)} = \frac{\pi_n^{(k)}}{\pi_{n+1}^{(k)}}\alpha_n, \tag{4.10}$$

$$\left(1 + \frac{\beta_{n-1}^{(k)}}{\alpha_{n-1}^{(k)}}\alpha_n^{(k)} - \alpha_n^{(k)}\sigma_k\right)\frac{\pi_{n+1}^{(k)}}{\pi_n^{(k)}} = 1 + \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n, \tag{4.11}$$

$$\frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n = \frac{\beta_{n-1}^{(k)}\alpha_n^{(k)}\pi_{n+1}^{(k)}}{\alpha_{n-1}^{(k)}\pi_{n-1}^{(k)}}. \tag{4.12}$$

Substituting (4.10) into (4.12) gives

$$\beta_{n-1}^{(k)} = \left(\frac{\pi_{n-1}^{(k)}}{\pi_n^{(k)}}\right)^2 \beta_{n-1}. \tag{4.13}$$

Substituting (4.10) and (4.13) into (4.11) leads to

$$\pi_{n+1}^{(k)} = \left(1 + \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n + \alpha_n\sigma_k\right)\pi_n^{(k)} - \frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n\pi_{n-1}^{(k)}. \tag{4.14}$$

Since $\boldsymbol{r}_0 = \boldsymbol{r}_0^{(k)}$, it follows that $\pi_0^{(k)} = 1$. From (3.21) and (4.14), we have the following relation:

$$\pi_{n+1}^{(k)} = R_{n+1}(-\sigma_k). \tag{4.15}$$

In what follows, the computational formula of the approximate solution vector $\boldsymbol{x}_{n+1}^{(k)}$ for (4.1) is derived from (4.8). Let $\boldsymbol{p}_{n-1}^{(k)} := (A + \sigma_k I)^{-1}(\boldsymbol{r}_{n-1}^{(k)} - \boldsymbol{r}_n^{(k)})/\alpha_{n-1}^{(k)}$, then from (4.8) we have

$$\boldsymbol{p}_n^{(k)} = \boldsymbol{r}_n^{(k)} + \beta_{n-1}^{(k)}\boldsymbol{p}_{n-1}^{(k)}, \tag{4.16}$$

$$\boldsymbol{r}_{n+1}^{(k)} = \boldsymbol{r}_n^{(k)} - \alpha_n^{(k)}(A + \sigma_k I)\boldsymbol{p}_n^{(k)}. \tag{4.17}$$

Substituting $\boldsymbol{r}_n^{(k)} = \boldsymbol{b} - (A + \sigma_k I)\boldsymbol{x}_n^{(k)}$ into (4.17) yields

$$\boldsymbol{x}_{n+1}^{(k)} = \boldsymbol{x}_n^{(k)} + \alpha_n^{(k)} \boldsymbol{p}_n^{(k)}. \tag{4.18}$$

From (4.6) the search direction $\boldsymbol{p}_n^{(k)}$ is updated by

$$\boldsymbol{p}_n^{(k)} = \frac{1}{\pi_n^{(k)}} \boldsymbol{r}_n + \beta_{n-1}^{(k)} \boldsymbol{p}_{n-1}^{(k)}. \tag{4.19}$$

It is worth mentioning that approximate solutions are computed without using the recurrence (4.17). Thus, matrix–vector multiplications $(A + \sigma_k I)\boldsymbol{p}_n^{(k)}$ for all k's are not required, which is much cost-efficient.

Using (4.6), (4.10)–(4.15), (4.18), and (4.19), we obtain the shifted CG method described in Algorithm 4.1. In Algorithm 4.1, the CG method is applied to $A^{(s)}\boldsymbol{x} = \boldsymbol{b}$ for a given $s \in \{1, 2, \dots, m\}$ that is referred to as a *seed system*, and approximate solutions $\boldsymbol{x}_n^{(k)}$ for the other shifted linear systems $A^{(k)}\boldsymbol{x} = \boldsymbol{b}$ are computed by using the residual vector $\boldsymbol{r}_n$ produced by the CG method.

Algorithm 4.1 has the following properties:

1. the multiplication of $A^{(s)}$ and a vector is required, but the multiplications of $A^{(k)}$ for the other k and a vector are not required;
2. the approximate solution $\boldsymbol{x}_n^{(k)}$ of the shifted CG method is the same as the nth approximate solution of the CG method (with $\boldsymbol{x}_0 = \boldsymbol{0}$) applied to $A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}$.

It is obvious from Algorithm 4.1 that the first property holds. The second property is also obvious from the derivation process of the shifted CG method.

In Algorithm 4.1, the relation $\|\boldsymbol{r}_n^{(k)}\| = \|\boldsymbol{r}_n\|/|\pi_n^{(k)}|$ can be used for monitoring the convergence of the approximate solution $\boldsymbol{x}_n^{(k)}$. Finally, notice that for computing $A^{(s)}\boldsymbol{v}$, we compute $A\boldsymbol{v} + \sigma_s \boldsymbol{v}$.

4.2.2 The Shifted CR Method

The shifted CR method can be very similarly derived from the CR method in Sect. 3.1.2 and the derivation process given in Sect. 4.2.1. The algorithm of the shifted CR method is described in Algorithm 4.2.

The main differences between the shifted CG method and the shifted CR method are the computational formulas for $\alpha_n^{(s)}$ and $\beta_n^{(s)}$. The shifted CR method (Algorithm 4.2) has the following properties:

1. the multiplication of $A^{(s)}$ and a vector is required, but the multiplications of $A^{(k)}$ for the other k and a vector are not required, which is the same property as that of the shifted CG method;
2. Unlike the shifted CG method, the approximate solution $\boldsymbol{x}_n^{(k)}$ of the shifted CR method is not the same as the nth approximate solution of the CR method (with $\boldsymbol{x}_0 = \boldsymbol{0}$) applied to $A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}$.

Algorithm 4.1 The shifted CG method

```
Input: Choose a seed system $s \in S = \{1, 2, \ldots, m\}$ and set $\boldsymbol{r}_0^{(s)} = \boldsymbol{b},\ \beta_{-1}^{(s)} = 0$
Input: $A$ and $\sigma_s$ for $k = 1, 2, \ldots, m$
Input: $\boldsymbol{x}_0^{(k)} = \boldsymbol{p}_{-1}^{(k)} = \boldsymbol{0}, \pi_0^{(s,k)} = \pi_{-1}^{(s,k)} = 1$ for $k = 1, 2, \ldots, m$
Output: $\boldsymbol{x}_n^{(k)}$ for $k = 1, 2, \ldots, m$
1: for $n = 0, 1, \ldots,$ until convergence do
2:    $\boldsymbol{p}_n^{(s)} = \boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(s)} \boldsymbol{p}_{n-1}^{(s)}$
3:    $\alpha_n^{(s)} = \frac{(\boldsymbol{r}_n^{(s)}, \boldsymbol{r}_n^{(s)})}{(\boldsymbol{p}_n^{(s)}, A^{(s)} \boldsymbol{p}_n^{(s)})}$
4:    $\boldsymbol{x}_{n+1}^{(s)} = \boldsymbol{x}_n^{(s)} + \alpha_n^{(s)} \boldsymbol{p}_n^{(s)}$
5:    {Begin shifted system}
6:    for $k (\neq s) = 1, 2, \ldots, m$ do
7:       if $\|\boldsymbol{r}_n^{(k)}\| (= \|\boldsymbol{r}_n^{(s)}\| / |\pi_n^{(k)}|) > \epsilon_2 \|\boldsymbol{b}\|$ then
8:          $\pi_{n+1}^{(s,k)} = R_{n+1}^{(s)}(\sigma_s - \sigma_k)$    $\{\leftarrow$ see (4.15)$\}$
9:                $= \left[1 + \frac{\beta_{n-1}^{(s)}}{\alpha_{n-1}^{(s)}} \alpha_n^{(s)} - \alpha_n^{(s)} (\sigma_s - \sigma_k)\right] \pi_n^{(s,k)} - \frac{\beta_{n-1}^{(s)}}{\alpha_{n-1}^{(s)}} \alpha_n^{(s)} \pi_{n-1}^{(s,k)}$
10:         $\beta_{n-1}^{(k)} = \left(\frac{\pi_{n-1}^{(s,k)}}{\pi_n^{(s,k)}}\right)^2 \beta_{n-1}^{(s)}$
11:         $\alpha_n^{(k)} = \frac{\pi_n^{(s,k)}}{\pi_{n+1}^{(s,k)}} \alpha_n^{(s)}$
12:         $\boldsymbol{p}_n^{(k)} = \frac{1}{\pi_n^{(s,k)}} \boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(k)} \boldsymbol{p}_{n-1}^{(k)}$
13:         $\boldsymbol{x}_{n+1}^{(k)} = \boldsymbol{x}_n^{(k)} + \alpha_n^{(k)} \boldsymbol{p}_n^{(k)}$
14:      end if
15:   end for
16:   {End shifted system}
17:   $\boldsymbol{r}_{n+1}^{(s)} = \boldsymbol{r}_n^{(s)} - \alpha_n^{(s)} A^{(s)} \boldsymbol{p}_n^{(s)}$
18:   $\beta_n^{(s)} = \frac{(\boldsymbol{r}_{n+1}^{(s)}, \boldsymbol{r}_{n+1}^{(s)})}{(\boldsymbol{r}_n^{(s)}, \boldsymbol{r}_n^{(s)})}$
19: end for
```

Algorithm 4.2 The shifted CR method

```
Input: Choose a seed system $s \in S = \{1, 2, \ldots, m\}$ and set $\boldsymbol{r}_0^{(s)} = \boldsymbol{b},\ \beta_{-1}^{(s)} = 0$
Input: $\boldsymbol{x}_0^{(k)} = \boldsymbol{p}_{-1}^{(k)} = \boldsymbol{0}, \pi_0^{(s,k)} = \pi_{-1}^{(s,k)} = 1$ for $k = 1, 2, \ldots, m$
Output: $\boldsymbol{x}_n^{(k)}$ for $k = 1, 2, \ldots, m$
1: for $n = 0, 1, \ldots,$ until convergence do
2:    $\boldsymbol{p}_n^{(s)} = \boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(s)} \boldsymbol{p}_{n-1}^{(s)}$
3:    $(A^{(s)} \boldsymbol{p}_n^{(s)} = A^{(s)} \boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(s)} A^{(s)} \boldsymbol{p}_{n-1}^{(s)})$
4:    $\alpha_n^{(s)} = \frac{(\boldsymbol{r}_n^{(s)}, A^{(s)} \boldsymbol{r}_n^{(s)})}{(A^{(s)} \boldsymbol{p}_n^{(s)}, A^{(s)} \boldsymbol{p}_n^{(s)})}$
5:    $\boldsymbol{x}_{n+1}^{(s)} = \boldsymbol{x}_n^{(s)} + \alpha_n^{(s)} \boldsymbol{p}_n^{(s)}$
6:    {Begin shifted system}
7:    Run lines 6-15 of the shifted CG method.
8:    {End shifted system}
9:    $\boldsymbol{r}_{n+1}^{(s)} = \boldsymbol{r}_n^{(s)} - \alpha_n^{(s)} A^{(s)} \boldsymbol{p}_n^{(s)}$
10:   $\beta_n^{(s)} = \frac{(\boldsymbol{r}_{n+1}^{(s)}, A^{(s)} \boldsymbol{r}_{n+1}^{(s)})}{(\boldsymbol{r}_n^{(s)}, A^{(s)} \boldsymbol{r}_n^{(s)})}$
11: end for
```

The second property comes from the fact that the nth residual vectors of the CR method for $A\boldsymbol{x} = \boldsymbol{b}$ and $A^{(k)}\boldsymbol{x} = \boldsymbol{b}$ do not belong to the same one-dimensional subspace because the residual vectors belong to

$$\boldsymbol{r}_n^{\mathrm{CR}} \in \mathcal{K}_{n+1}(A, \boldsymbol{b}) \cap A\mathcal{K}_n(A, \boldsymbol{b})^{\perp},$$
$$\boldsymbol{r}_n^{(k)\mathrm{CR}} \in \mathcal{K}_{n+1}(A^{(k)}, \boldsymbol{b}) \cap A^{(k)}\mathcal{K}_n(A^{(k)}, \boldsymbol{b})^{\perp}.$$

Notice that $\mathcal{K}_{n+1}(A, \boldsymbol{b}) = \mathcal{K}_{n+1}(A^{(k)}, \boldsymbol{b})$ holds true from the shift invariance property (4.4), but in general $A\mathcal{K}_n(A, \boldsymbol{b}) \neq A^{(k)}\mathcal{K}_n(A^{(k)}, \boldsymbol{b})$. Thus,

$$\mathcal{K}_{n+1}(A, \boldsymbol{b}) \cap A\mathcal{K}_n(A, \boldsymbol{b})^{\perp} \neq \mathcal{K}_{n+1}(A^{(k)}, \boldsymbol{b}) \cap A^{(k)}\mathcal{K}_n(A^{(k)}, \boldsymbol{b})^{\perp}.$$

This means that $\boldsymbol{r}_n^{\mathrm{CR}}$ and $\boldsymbol{r}_n^{(k)\mathrm{CR}}$ are not produced by the same one-dimensional subspace, which are not collinear.

4.2.3 The Shifted MINRES Method

The shifted MINRES method is based on the MINRES method in Sect. 3.1.3, which achieves the minimization of residual 2-norms:

$$\min_{\boldsymbol{x}_n^{(k)} \in \mathcal{K}_n(A^{(k)}, \boldsymbol{b}) (= \mathcal{K}_n(A, \boldsymbol{b}))} \left\| \boldsymbol{b} - A^{(k)}\boldsymbol{x}_n^{(k)} \right\|. \tag{4.20}$$

In what follows, we will see how the above minimization problems can efficiently be solved, and note that the idea below corresponds to the special case of the shifted GMRES method [43] that will be described in Sect. 4.4.3.

From the Lanczos process in Sect. 1.9.4, $\boldsymbol{x}_n^{(k)} \in \mathcal{K}_n(A, \boldsymbol{b})$ can be rewritten as

$$\boldsymbol{x}_n^{(k)} = V_n \boldsymbol{y}_n^{(k)}, \quad k = 1, 2, \ldots, m, \tag{4.21}$$

where $\boldsymbol{y}_n^{(k)} \in \mathbb{C}^n$. The corresponding residual vectors for (4.1) are $\boldsymbol{r}_n^{(k)} := \boldsymbol{b} - A^{(k)}\boldsymbol{x}_n^{(k)} = \boldsymbol{b} - (A + \sigma_k I)\boldsymbol{x}_n^{(k)}$. From (4.21) and the matrix form of the Lanczos process (1.42), it follows that

$$\begin{aligned} \boldsymbol{r}_n^{(k)} &= \boldsymbol{b} - (A + \sigma_k I) V_n \boldsymbol{y}_n^{(k)} \\ &= V_{n+1} g_1 \boldsymbol{e}_1 - (A V_n + \sigma_k V_n) \boldsymbol{y}_n^{(k)} \\ &= V_{n+1}\left(g_1 \boldsymbol{e}_1 - T_{n+1,n}^{(k)} \boldsymbol{y}_n^{(k)}\right), \quad T_{n+1,n}^{(k)} := T_{n+1,n} + \sigma_k \begin{bmatrix} I_n \\ \boldsymbol{0}^{\top} \end{bmatrix}. \end{aligned} \tag{4.22}$$

Here, $\boldsymbol{e}_1=(1,0,\dots,0)^\top$ is the first unit vector and $g_1=(\boldsymbol{b},\boldsymbol{b})^{1/2}$. From (4.20) and (4.22), we have

$$\begin{aligned}\min_{\boldsymbol{x}_n\in\mathcal{K}_n(A,\boldsymbol{b})}\left\|\boldsymbol{b}-A^{(k)}\boldsymbol{x}_n\right\| &= \min_{\boldsymbol{y}_n^{(k)}\in\mathbb{C}^n}\left\|V_{n+1}\left(g_1\boldsymbol{e}_1-T_{n+1,n}^{(k)}\boldsymbol{y}_n^{(k)}\right)\right\| \\ &= \min_{\boldsymbol{y}_n^{(k)}\in\mathbb{C}^n}\left\|g_1\boldsymbol{e}_1-T_{n+1,n}^{(k)}\boldsymbol{y}_n^{(k)}\right\|. \qquad (4.23)\end{aligned}$$

The minimization problem (4.23) can be solved by Givens rotations that are described in Sect. 3.1.3. The algorithm of the shifted MINRES method is shown in Algorithm 4.3.

Algorithm 4.3 The shifted MINRES method

```
Input: $\boldsymbol{x}_0^{(k)}=\boldsymbol{p}_{-1}^{(k)}=\boldsymbol{p}_0^{(k)}=\boldsymbol{0}$, $\boldsymbol{v}_1=\boldsymbol{b}/(\boldsymbol{b},\boldsymbol{b})^{1/2}$, $g_1^{(k)}=(\boldsymbol{b},\boldsymbol{b})^{1/2}$, $\beta_{-1}=0$
Output: $\boldsymbol{x}_n^{(k)}$ for $l=1,2,\dots,m$
1: for $n=1,2,\dots$ do
2:   {The Lanczos process}
3:   $\alpha_n=(\boldsymbol{v}_n,A\boldsymbol{v}_n)$
4:   $\tilde{\boldsymbol{v}}_{n+1}=A\boldsymbol{v}_n-\alpha_n\boldsymbol{v}_n-\beta_{n-1}\boldsymbol{v}_{n-1}$
5:   $\beta_n=(\tilde{\boldsymbol{v}}_{n+1},\tilde{\boldsymbol{v}}_{n+1})^{1/2}$
6:   $\boldsymbol{v}_{n+1}=\tilde{\boldsymbol{v}}_{n+1}/\beta_n$
7:   $t_{n-1,n}^{(k)}=\beta_{n-1},\ t_{n,n}^{(k)}=\alpha_n+\sigma_\ell,\ t_{n+1,n}^{(k)}=\beta_n$
8:   {Solve least-squares problems by Givens rotations}
9:   for $k=1,2,\dots,m$ do
10:    if $|g_{n+1}^{(k)}|/\|\boldsymbol{b}\|>\epsilon$ then
11:      for $i=\max\{1,n-2\},\dots,n-1$ do
12:        $\begin{bmatrix}t_{i,n}^{(k)}\\ t_{i+1,n}^{(k)}\end{bmatrix}=\begin{bmatrix}c_i^{(k)} & s_i^{(k)}\\ -\overline{s}_i^{(k)} & c_i^{(k)}\end{bmatrix}\begin{bmatrix}t_{i,n}^{(k)}\\ t_{i+1,n}^{(k)}\end{bmatrix}$
13:      end for
14:      $c_n^{(k)}=\frac{|t_{n,n}^{(k)}|}{\sqrt{|t_{n,n}^{(k)}|^2+|t_{n+1,n}^{(k)}|^2}}$
15:      $\overline{s}_n^{(k)}=\frac{t_{n+1,n}^{(k)}}{t_{n,n}^{(k)}}c_n^{(k)}$
16:      $t_{n,n}^{(k)}=c_n^{(k)}t_{n,n}^{(k)}+s_n^{(k)}t_{n+1,n}^{(k)}$
17:      $t_{n+1,n}^{(k)}=0$
18:      $\begin{bmatrix}g_n^{(k)}\\ g_{n+1}^{(k)}\end{bmatrix}=\begin{bmatrix}c_n^{(k)} & s_n^{(k)}\\ -\overline{s}_n^{(k)} & c_n^{(k)}\end{bmatrix}\begin{bmatrix}g_n^{(k)}\\ 0\end{bmatrix}$
19:      {Update approximate solutions $\boldsymbol{x}_n^{(k)}$}
20:      $\boldsymbol{p}_n^{(k)}=\boldsymbol{v}_n-(t_{n-2,n}^{(k)}/t_{n-2,n-2}^{(k)})\boldsymbol{p}_{n-2}^{(k)}-(t_{n-1,n}^{(k)}/t_{n-1,n-1}^{(k)})\boldsymbol{p}_{n-1}^{(k)}$
21:      $\boldsymbol{x}_n^{(k)}=\boldsymbol{x}_{n-1}^{(k)}+(g_n^{(k)}/t_{n,n}^{(k)})\boldsymbol{p}_n^{(k)}$
22:    end if
23:  end for
24:  if $|g_{n+1}^{(k)}|/\|\boldsymbol{b}\|\le\epsilon$ for all $\ell$, then exit.
25: end for
```

From Algorithm 4.3 and the derivation process, we see that:

1. the multiplication of $A^{(s)}$ and a vector is required, but the multiplications of $A^{(k)}$ for the other k and a vector are not required, which is the same property as that of the shifted CG method and the shifted CR method;
2. the approximate solution $\boldsymbol{x}_n^{(k)}$ of the shifted MINRES method is the same as the nth approximate solution of the MINRES method (with $\boldsymbol{x}_0 = \boldsymbol{0}$) applied to $A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}$.

4.3 Shifted Complex Symmetric Linear Systems

Throughout this section, the coefficient matrix $A^{(k)}$ in (4.1) is assumed to be complex symmetric, i.e., $A^{(k)} = (A^{(k)})^\top \neq (A^{(k)})^\mathrm{H}$.

4.3.1 *The Shifted COCG Method*

The derivation of the shifted COCG method [184] is the same as in (4.6)–(4.19). The only differences between the shifted CG method and the shifted COCG methods are α_n and β_n. The algorithm of the shifted COCG method is described in Algorithm 4.4.

Algorithm 4.4 The shifted COCG method (note: $(\overline{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: Choose a seed system $s \in S = \{1, 2, \dots, m\}$ and set $\boldsymbol{r}_0^{(s)} = \boldsymbol{b},\ \beta_{-1}^{(s)} = 0$
Input: $\boldsymbol{x}_0^{(k)} = \boldsymbol{p}_{-1}^{(k)} = \boldsymbol{0}, \pi_0^{(s,k)} = \pi_{-1}^{(s,k)} = 1$ for $k = 1, 2, \dots, m$
Output: $\boldsymbol{x}_n^{(k)}$ for $k = 1, 2, \dots, m$
1: for $n = 0, 1, \dots,$ until convergence do
2:     $\boldsymbol{p}_n^{(s)} = \boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(s)} \boldsymbol{p}_{n-1}^{(s)}$
3:     $\alpha_n^{(s)} = \frac{(\overline{\boldsymbol{r}}_n^{(s)}, \boldsymbol{r}_n^{(s)})}{(\overline{\boldsymbol{p}}_n^{(s)}, A^{(s)} \boldsymbol{p}_n^{(s)})}$
4:     $\boldsymbol{x}_{n+1}^{(s)} = \boldsymbol{x}_n^{(s)} + \alpha_n^{(s)} \boldsymbol{p}_n^{(s)}$
5:     {Begin shifted system}
6:     Run lines 6–15 of the shifted CG method.
7:     {End shifted system}
8:     $\boldsymbol{r}_{n+1}^{(s)} = \boldsymbol{r}_n^{(s)} - \alpha_n^{(s)} A^{(s)} \boldsymbol{p}_n^{(s)}$
9:     $\beta_n^{(s)} = \frac{(\overline{\boldsymbol{r}}_{n+1}^{(s)}, \boldsymbol{r}_{n+1}^{(s)})}{(\overline{\boldsymbol{r}}_n^{(s)}, \boldsymbol{r}_n^{(s)})}$
10: end for
```

Algorithm 4.4 has the following properties:

1. the multiplication of $A^{(s)}$ and a vector is required, but the multiplications of $A^{(k)}$ for the other k and a vector are not required;
2. the approximate solution $\boldsymbol{x}_n^{(k)}$ of the shifted COCG method is the same as the nth approximate solution of the COCG method (with $\boldsymbol{x}_0 = \boldsymbol{0}$) applied to $A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}$.

When matrix $A^{(k)}$ is real symmetric for all k, the shifted COCG method is equivalent to the shifted CG method.

In [190], the shifted COCG method is generalized to solving the following linear systems:

$$(A + \sigma_k B)\boldsymbol{x}^{(k)} = \boldsymbol{b}, \tag{4.24}$$

which are referred to as *generalized shifted linear systems*. Here A and B are real symmetric and σ_k's are complex numbers, and thus the coefficient matrix $A + \sigma_k B$ is complex symmetric. It is easy to see that if B is the identity matrix, then the generalized shifted linear systems (4.24) reduce to the shifted linear systems.

We note that the shifted COCG method and the other shifted Krylov subspace methods cannot be applied to (4.24) because the shift invariance properties do not hold any longer, i.e., $\mathcal{K}_n(A + \sigma_i B, \boldsymbol{b}) \neq \mathcal{K}_n(A + \sigma_j B, \boldsymbol{b})$ for $i \neq j$. One remedy is to consider the following shifted linear systems:

$$(B^{-1}A + \sigma_k I)\boldsymbol{x}^{(k)} = B^{-1}\boldsymbol{b},$$

where B is assumed to be nonsingular. However, $B^{-1}A + \sigma_k I$ is non-Hermitian, and thus one may use shifted Krylov subspace methods for non-Hermitian linear systems. On the other hand, without using such general solvers, it is shown that an algorithm similar to the shifted COCG method is constructed, which is referred to as the generalized shifted COCG method [190]. If $B = I$, the generalized shifted COCG method reduces to the shifted COCG method.

As pointed out in Sect. 2.2.1, if the Hamiltonian matrix is real symmetric, then the coefficient matrices $(\epsilon + i\delta)I - H$ of the shifted linear systems are complex symmetric. Thus the shifted COCG method is a method of choice, and applications to computational physics are found in, e.g., [73, 110, 111]. An open-source library of the shifted COCG method and related solvers is developed in [100].

The generalized shifted linear systems also arise in the following linear systems:

$$(\sigma^2 A + \sigma B + C)\boldsymbol{x} = \boldsymbol{b}, \tag{4.25}$$

where A, B, C are complex symmetric. Parameterized linear systems of this type are considered in [158, 159], together with their application to structural dynamics. The interesting observation in [158, §3] is that (4.25) can be transformed into the following linear systems:

$$\left(\begin{bmatrix} B & C \\ C^\top & O \end{bmatrix} + \sigma \begin{bmatrix} A & O \\ O & -C^\top \end{bmatrix}\right)\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix} \tag{4.26}$$

if C is nonsingular. Since A, B, C are complex symmetric, the coefficient matrices of (4.26) are complex symmetric generalized shifted linear systems. Thus, the generalized shifted COCG method can also be used to solve (4.25).

4.3.1.1 Seed Switching Technique

We can see from Algorithm 4.4 (see also line 8 of Algorithm 4.1) that if $|\pi_n^{(s,k)}| = |R_n^{(s)}(\sigma_s - \sigma_k)| \geq 1$, then $\|\boldsymbol{r}_n^{(k)}\| \leq \|\boldsymbol{r}_n^{(s)}\|$. Hence, if we could find a seed system such that $|R_n^{(s)}(\sigma_s - \sigma_k)| \geq 1$, then all the shifted systems could be solved. However, it is extremely difficult to find such a system in advance except for some special cases discussed in [67, Corollary 1]. The seed switching technique [166] will avoid such a problem, and consists of the following steps:

(S1) Choose a seed system, and then start Algorithm 4.4.
(S2) If the seed system was solved at the nth iteration, then find a new one that remained unsolved.
(S3) Start Algorithm 4.4 from the $(n + 1)$th iteration using the new seed system.

In (S2), one of the criteria for choosing the new seed system $\tilde{s}$ may be

$$\tilde{s} = \arg\max_{i \in I}\{\|\boldsymbol{r}_n^{(i)}\|\},$$

where I denotes an index set of the unsolved linear systems. In (S3), we need two steps to switch the old seed system to the new one. First, compute

$$\pi_{n+1}^{(s,\tilde{s})} = R_{n+1}^{(s)}(\sigma_s - \sigma_{\tilde{s}}), \quad \beta_n^{(\tilde{s})} = \left(\pi_n^{(s,\tilde{s})}/\pi_{n+1}^{(s,\tilde{s})}\right)^2 \beta_n$$

to obtain $\boldsymbol{r}_{n+1}^{(\tilde{s})}$ and $\beta_n^{(\tilde{s})}\boldsymbol{p}_n^{(\tilde{s})}$. Since it follows from $\boldsymbol{r}_{n+1}^{(\tilde{s})} + \beta_n^{(\tilde{s})}\boldsymbol{p}_n^{(\tilde{s})}$ that we obtain $\boldsymbol{p}_{n+1}^{(\tilde{s})}$, one can start the COCG method solving the system $(A + \sigma_{\tilde{s}} I)\boldsymbol{x}^{(\tilde{s})} = \boldsymbol{b}$ from the $(n + 1)$th iteration step. Second, to solve the remaining linear systems by using the new seed $\tilde{s}$, the parameters $\alpha_{n+1}^{(i)}$ and $\beta_n^{(i)}$ must be generated from the new seed. We see that they can readily be generated by the following polynomial:

$$\pi_{n+1}^{(\tilde{s},i)} = R_{n+1}^{(\tilde{s})}(\sigma_{\tilde{s}} - \sigma_i) \quad \text{for all} \quad i \in I.$$

To obtain the above polynomial, one needs to compute

$$\alpha_i^{(\tilde{s})} = \left(\pi_i^{(s,\tilde{s})}/\pi_{i+1}^{(s,\tilde{s})}\right) \alpha_i, \quad \beta_j^{(\tilde{s})} = \left(\pi_j^{(s,\tilde{s})}/\pi_{j+1}^{(s,\tilde{s})}\right)^2 \beta_j$$

for $i = 0, \ldots, n, \ \ j = 0, \ldots, n - 1$. Hence, the switching strategy requires only scalar operations, and moreover we can see that if breakdown does not occur, iterating the process from (S2) to (S3) enables us to keep solving the systems without losing the dimension of the Krylov subspace that has been generated until the last switching.

The seed switching technique enables us to be free from the problem of the choice of the seed system, and can also be applied to the shifted CG method in Sect. 4.2.1 and the shifted BiCG method in Sect. 4.4.1, since the residual vectors of the shifted CG (BiCG) method for shifted systems are true CG (BiCG) residuals.

4.3.2 The Shifted COCR Method

The derivation of the shifted COCR method [172] is also the same as in (4.6)–(4.19). The only differences between the shifted CR method and the shifted COCR methods are α_n and β_n. The algorithm of the shifted COCR method is described in Algorithm 4.5.

Algorithm 4.5 has the following properties:

1. the multiplication of $A^{(s)}$ and a vector is required, but the multiplications of $A^{(k)}$ for the other k and a vector are not required, which is the same property as that of the shifted COCG method;
2. unlike the shifted COCG method, $\boldsymbol{x}_n^{(k)}$ of the shifted COCR method is not the same as the nth approximate solution of the COCR method (with $\boldsymbol{x}_0 = \boldsymbol{0}$) applied to $A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}$.

Algorithm 4.5 The shifted COCR method (note: $(\overline{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: Choose a seed system $s \in S = \{1, 2, \dots, m\}$ and set $\boldsymbol{r}_0^{(s)} = \boldsymbol{b},\ \beta_{-1}^{(s)} = 0$
Input: $\boldsymbol{x}_0^{(k)} = \boldsymbol{p}_{-1}^{(k)} = \boldsymbol{0},\ \pi_0^{(s,k)} = \pi_{-1}^{(s,k)} = 1$ for $k = 1, 2, \dots, m$
Output: $\boldsymbol{x}_n^{(k)}$ for $k = 1, 2, \dots, m$
1: for $n = 0, 1, \dots$, until convergence do
2:    $\boldsymbol{p}_n^{(s)} = \boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(s)} \boldsymbol{p}_{n-1}^{(s)}$
3:    $(A^{(s)}\boldsymbol{p}_n^{(s)} = A^{(s)}\boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(s)} A^{(s)}\boldsymbol{p}_{n-1}^{(s)})$
4:    $\alpha_n^{(s)} = \frac{(\overline{\boldsymbol{r}}_n^{(s)}, A^{(s)}\boldsymbol{r}_n^{(s)})}{(\overline{A}(\sigma_s)\overline{\boldsymbol{p}}_n^{(s)}, A^{(s)}\boldsymbol{p}_n^{(s)})}$
5:    $\boldsymbol{x}_{n+1}^{(s)} = \boldsymbol{x}_n^{(s)} + \alpha_n^{(s)} \boldsymbol{p}_n^{(s)}$
6:    {Begin shifted system}
7:    Run lines 6–15 of the shifted CG method.
8:    {End shifted system}
9:    $\boldsymbol{r}_{n+1}^{(s)} = \boldsymbol{r}_n^{(s)} - \alpha_n^{(s)} A^{(s)} \boldsymbol{p}_n^{(s)}$
10:   $\beta_n^{(s)} = \frac{(\overline{\boldsymbol{r}}_{n+1}^{(s)}, A^{(s)}\boldsymbol{r}_{n+1}^{(s)})}{(\overline{\boldsymbol{r}}_n^{(s)}, A^{(s)}\boldsymbol{r}_n^{(s)})}$
11: end for
```

When matrix $A^{(k)}$ is real symmetric for all k, the shifted COCR method is equivalent to the shifted CR method.

4.3.3 The Shifted QMR_SYM Method

The shifted QMR method for solving non-Hermitian linear systems was proposed in [62]. As is known in [61], the shifted QMR_SYM method is a simplification of the shifted QMR method, and the derivation of the shifted QMR_SYM is similar to the shifted MINRES method.

From the complex symmetric Lanczos process in Sect. 1.9.3, $\boldsymbol{x}_n \in \mathcal{K}_n(A, \boldsymbol{b})$ can be rewritten as

$$\boldsymbol{x}_n^{(k)} = V_n \boldsymbol{y}_n^{(k)}, \quad k = 1, 2, \dots, m, \tag{4.27}$$

where $\boldsymbol{y}_n^{(k)} \in \mathbb{C}^n$. The corresponding residual vectors for (4.1) are $\boldsymbol{r}_n^{(k)} := \boldsymbol{b} - (A + \sigma_k I)\boldsymbol{x}_n^{(k)}$. From (4.27) and the matrix form of the complex symmetric Lanczos process (1.41), it follows that

$$\begin{aligned} \boldsymbol{r}_n^{(k)} &= \boldsymbol{b} - (A + \sigma_k I) V_n \boldsymbol{y}_n^{(k)} \\ &= V_{n+1} g_1 \boldsymbol{e}_1 - (A V_n + \sigma_k V_n) \boldsymbol{y}_n^{(k)} \\ &= V_{n+1}\left(g_1 \boldsymbol{e}_1 - T_{n+1,n}^{(k)} \boldsymbol{y}_n^{(k)}\right), \quad T_{n+1,n}^{(k)} := T_{n+1,n} + \sigma_k \begin{bmatrix} I_n \\ \boldsymbol{0}^\top \end{bmatrix}. \end{aligned} \tag{4.28}$$

Here, $\boldsymbol{e}_1$=$(1, 0, \dots, 0)^\top$ is the first unit vector and $g_1 = (\bar{\boldsymbol{b}}, \boldsymbol{b})^{1/2}$. As well as the QMR_SYM method in Sect. 3.2.3, $\boldsymbol{y}_n^{(k)}$ is determined by solving the following minimization problem.

$$\min_{\boldsymbol{y}_n^{(k)} \in \mathbb{C}^n} \left\| g_1 \boldsymbol{e}_1 - T_{n+1,n}^{(k)} \boldsymbol{y}_n^{(k)} \right\|. \tag{4.29}$$

The minimization problem (4.29) can be solved by Givens rotations that are described in Sect. 3.1.3. The algorithm of the shifted QMR_SYM method is shown in Algorithm 4.6.

Algorithm 4.6 holds the following properties:

1. the multiplication of $A^{(s)}$ and a vector is required, but the multiplications of $A^{(k)}$ for the other k and a vector are not required, which is the same property as those of the shifted CG method and the shifted CR method;
2. the approximate solution $\boldsymbol{x}_n^{(k)}$ of the shifted QMR_SYM method is the same as the nth approximate solution of the QMR_SYM method (with $\boldsymbol{x}_0 = \boldsymbol{0}$) applied to $A^{(k)}\boldsymbol{x}^{(k)} = \boldsymbol{b}$.

Algorithm 4.6 The shifted QMR_SYM method (note: $(\bar{\boldsymbol{a}}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}$)

```
Input: $\boldsymbol{x}_0^{(k)} = \boldsymbol{p}_{-1}^{(k)} = \boldsymbol{p}_0^{(k)} = \boldsymbol{0}$, $\boldsymbol{v}_1 = \boldsymbol{b}/(\bar{\boldsymbol{b}}, \boldsymbol{b})^{1/2}$, $g_1^{(k)} = (\bar{\boldsymbol{b}}, \boldsymbol{b})^{1/2}$, $\beta_{-1} = 0$
Output: $\boldsymbol{x}_n^{(k)}$ for $l = 1, 2, \dots, m$
1: for $n = 1, 2, \dots$ do
2:   (The complex symmetric Lanczos process)
3:   $\alpha_n = (\bar{\boldsymbol{v}}_n, A\boldsymbol{v}_n)$
4:   $\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n \boldsymbol{v}_n - \beta_{n-1}\boldsymbol{v}_{n-1}$
5:   $\beta_n = (\bar{\tilde{\boldsymbol{v}}}_{n+1}, \tilde{\boldsymbol{v}}_{n+1})^{1/2}$
6:   $\boldsymbol{v}_{n+1} = \tilde{\boldsymbol{v}}_{n+1}/\beta_n$
7:   $t_{n-1,n}^{(\ell)} = \beta_{n-1}$, $t_{n,n}^{(\ell)} = \alpha_n + \sigma_\ell$, $t_{n+1,n}^{(\ell)} = \beta_n$
8:   (Solve least-squares problems by Givens rotations)
9:   Run lines 9–24 of the shifted MINRES method.
10: end for
```

In [60], the following shifted linear systems are considered:

$$(A + \sigma_k I)\boldsymbol{x}^{(k)} = \boldsymbol{b}, \tag{4.30}$$

where A is Hermitian and σ_k's are complex numbers. Thus the coefficient matrix $(A + \sigma_k I)$ is neither Hermitian nor complex symmetric. In this case, one may use the shifted QMR method for non-Hermitian shifted linear systems. On the other hand, there is an algorithm in [60] that is more efficient than the shifted QMR method so that the multiplication of $(A + \sigma_k I)^{\mathrm{H}}$ and a vector is not required, i.e., the number of required matrix–vector multiplications is only one per each iteration step. The key idea is to generate a Krylov subspace of $\mathcal{K}_n(A, \boldsymbol{b})$ by the Lanczos process, and the basis vectors are used to solve (4.30). Note that if we generate $\mathcal{K}_n(A + \sigma I, \boldsymbol{b})$ with σ being a complex number instead of $\mathcal{K}_n(A, \boldsymbol{b})$, then we need the bi-Lanczos process whose computational cost is about twice as large as that of the Lanczos process when the matrix–vector multiplication is the most time-consuming part.

In [168], the shifted QMR_SYM method is generalized to solving (complex symmetric) generalized shifted linear systems (4.24). Similar to the generalized shifted COCG method as mentioned in Sect. 4.3.1, the generalized shifted QMR_SYM method [168] reduces to the shifted QMR_SYM method when $B = I$. Furthermore, based on the shifted weighted QMR_SYM method in [167] for complex symmetric shifted linear systems, the corresponding algorithm for solving complex symmetric generalized shifted linear systems is proposed in [168].

4.4 Shifted Non-Hermitian Linear Systems

Throughout this section, the coefficient matrix $A^{(k)}$ in (4.1) is assumed to be non-Hermitian, i.e., $A^{(k)} \neq (A^{(k)})^{\mathrm{H}}$.

4.4.1 The Shifted BiCG Method

As seen in Sect. 4.2.1, the ith residual vector $\boldsymbol{r}_i$ of the CG method for the seed system $A\boldsymbol{x} = \boldsymbol{b}$ and the ith residual vector $\boldsymbol{r}_i^{\sigma}$ for the shifted system $(A + \sigma I)\boldsymbol{x}^{\sigma} = \boldsymbol{b}$ are collinear. Theorem 4.1 is a generalized result of the condition that two residual vectors of a Krylov subspace method are collinear.

Theorem 4.1 ([67]) *Let $W_1 \subseteq W_2 \subseteq \cdots \subseteq W_k$ be a sequence of subspaces of $\mathbb{C}^N$ such that $\dim(W_i) = i$ and $W_i \cap \mathcal{K}_{i+1}(A, \boldsymbol{b})^{\perp} = \{\boldsymbol{0}\}$ for $i = 1, 2, \ldots, k$. Let $\boldsymbol{x}_i \in \mathcal{K}_i(A, \boldsymbol{b})^{\perp}$ be an approximate solution of $A\boldsymbol{x} = \boldsymbol{b}$ defined via the following Petrov–Galerkin condition of the residual $\boldsymbol{r}_i = \boldsymbol{b} - A\boldsymbol{x}_i = p_i(A)\boldsymbol{b}$:*

$$\boldsymbol{r}_i \perp W_i \quad \textit{for } i = 1, 2, \ldots, k,$$

where $p_i(A) = \sum_{k=0}^{i} c_k A^i$.

Similarly, let $\boldsymbol{x}_i^\sigma \in \mathcal{K}_i(A+\sigma I, \boldsymbol{b}) = \mathcal{K}_i(A, \boldsymbol{b})$ *be the approximation to the solution of* $(A+\sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}$ *with the residual* $\boldsymbol{r}^\sigma = \boldsymbol{b} - (A+\sigma I)\boldsymbol{x}_i^\sigma = p_i^\sigma(A+\sigma I)\boldsymbol{b}$*, again satisfying*

$$\boldsymbol{r}_i^\sigma \perp W_i \quad \textit{for } i = 1, 2, \dots, k.$$

Then $\boldsymbol{r}_i$ *and* $\boldsymbol{r}_i^\sigma$ *are collinear, i.e., there exists* $\pi_i^\sigma \in \mathbb{C}$ *such that* $\boldsymbol{r}_i = \pi_i^\sigma \boldsymbol{r}_i^\sigma$*.*

Proof Let U_1, U_2 be subspaces of $\mathbb{C}^N$. Then from standard linear algebra, it follows that $(U_1 \cap U_2)^\perp = U_1^\perp + U_2^\perp$. From assumption $W_i \cap \mathcal{K}_{i+1}(A, \boldsymbol{b})^\perp = \{\boldsymbol{0}\}$, we have

$$(W_i \cap \mathcal{K}_{i+1}(A, \boldsymbol{b})^\perp)^\perp = \{\boldsymbol{0}\}^\perp \Leftrightarrow W_i^\perp + \mathcal{K}_{i+1}(A, \boldsymbol{b}) = \mathbb{C}^N. \tag{4.31}$$

$\dim W_i^\perp = N - i$ and $\mathcal{K}_{i+1}(A, \boldsymbol{b}) = i + 1$, and thus $\dim W_i^\perp + \dim \mathcal{K}_{i+1}(A, \boldsymbol{b}) = N + 1$. On the other hand, from (4.31), $\dim(W_i^\perp + \mathcal{K}_{i+1}(A, \boldsymbol{b})) = \dim \mathbb{C}^N = N$. Thus

$$\dim(W_i^\perp \cap \mathcal{K}_{i+1}(A, \boldsymbol{b})) = 1.$$

Since $\boldsymbol{r}_i, \boldsymbol{r}_i^\sigma \in \mathcal{K}_{i+1}(A, \boldsymbol{b}) \cap W_i^\perp$, two residual vectors $\boldsymbol{r}_i, \boldsymbol{r}_i^\sigma$ lie in the same one-dimensional subspace, which means that $\boldsymbol{r}_i, \boldsymbol{r}_i^\sigma$ are collinear. □

Theorem 4.1 indicates that using the relation $W_n = \mathcal{K}_n(A^{\mathrm{H}}, \boldsymbol{r}_0^*)$, the BiCG residual vector $\boldsymbol{r}_n$ from $A\boldsymbol{x} = \boldsymbol{b}$ and the BiCG residual vector $\boldsymbol{r}_n^{(k)}$ from $(A + \sigma_k I)\boldsymbol{x}^{(k)} = \boldsymbol{b}$ are collinear, i.e.,

$$\boldsymbol{r}_n = \pi_n^{(k)} \boldsymbol{r}_n^{(k)}.$$

Thus, following the derivation of the shifted CG method in Sect. 4.2.1, the algorithm of the shifted BiCG method is obtained, which is listed in Algorithm 4.7.

Similarly, the BiCR method in Sect. 3.3.3 is developed to solve shifted linear systems, which is referred to as the shifted BiCR method in [85].

4.4.2 The Shifted BiCGSTAB Method

The shifted BiCGSTAB method and the shifted BiCGSTAB(ℓ) method are proposed in [67]. Since the derivation of the shifted BiCGSTAB(ℓ) method is somewhat complicated, the derivation of the shifted BiCGSTAB method is described, which is based on the explanation in [140].

For the derivation of the shifted BiCGSTAB method, we consider the seed system $A\boldsymbol{x} = \boldsymbol{b}$ and the shifted system $(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}$.

First of all, let us recall the BiCGSTAB method in Sect. 3.3.8. Let $\boldsymbol{r}_{n+1}$ be the residual vector of the BiCGSTAB method. Then $\boldsymbol{r}_{n+1}$ and the other iterates $\boldsymbol{t}_n$, $\boldsymbol{p}_n$, $\boldsymbol{x}_n$ are described by

Algorithm 4.7 The shifted BiCG method

```
Input: Choose a seed system $s \in S = \{1, 2, \dots, m\}$ and set $\boldsymbol{r}_0^{(s)} = \boldsymbol{b}$, $\beta_{-1}^{(s)} = 0$
Input: $A$ and $\sigma_s$ for $k = 1, 2, \dots, m$
Input: $\boldsymbol{x}_0^{(k)} = \boldsymbol{p}_{-1}^{(k)} = \boldsymbol{p}_{-1}^{*(k)} = \boldsymbol{0}$, $\pi_0^{(s,k)} = \pi_{-1}^{(s,k)} = 1$ for $k = 1, 2, \dots, m$
Input: Choose $\boldsymbol{r}_0^* \in \mathbb{C}^N$, e.g., $\boldsymbol{r}_0^* = \boldsymbol{r}_0$
Output: $\boldsymbol{x}_n^{(k)}$ for $k = 1, 2, \dots, m$
1: for $n = 0, 1, \dots$, until convergence do
2:     $\boldsymbol{p}_n^{(s)} = \boldsymbol{r}_n^{(s)} + \beta_{n-1}^{(s)} \boldsymbol{p}_{n-1}^{(s)}$,   $\boldsymbol{p}_n^{*(s)} = \boldsymbol{r}_n^{*(s)} + \overline{\beta}_{n-1}^{(s)} \boldsymbol{p}_{n-1}^{*(s)}$
3:     $\alpha_n^{(s)} = \frac{(\boldsymbol{r}_n^{*(s)}, \boldsymbol{r}_n^{(s)})}{(\boldsymbol{p}_n^{*(s)}, A^{(s)} \boldsymbol{p}_n^{(s)})}$
4:     $\boldsymbol{x}_{n+1}^{(s)} = \boldsymbol{x}_n^{(s)} + \alpha_n^{(s)} \boldsymbol{p}_n^{(s)}$
5:     {Begin shifted system}
6:     Run lines 6–15 of the shifted CG method.
7:     {End shifted system}
8:     $\boldsymbol{r}_{n+1}^{(s)} = \boldsymbol{r}_n^{(s)} - \alpha_n^{(s)} A^{(s)} \boldsymbol{p}_n^{(s)}$,   $\boldsymbol{r}_{n+1}^{*(s)} = \boldsymbol{r}_n^{*(s)} - \overline{\alpha}_n A^{(s)\mathrm{H}} \boldsymbol{p}_n^{*(s)}$
9:     $\beta_n^{(s)} = \frac{(\boldsymbol{r}_{n+1}^{*(s)}, \boldsymbol{r}_{n+1}^{(s)})}{(\boldsymbol{r}_n^{*(s)}, \boldsymbol{r}_n^{(s)})}$
10: end for
```

$$\begin{aligned}
\boldsymbol{p}_n &:= Q_n(A)\boldsymbol{p}_n^{\mathrm{BiCG}} = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \zeta_{n-1} A \boldsymbol{p}_{n-1}) \\
&= \boldsymbol{r}_n + \beta_{n-1}\left[\boldsymbol{p}_{n-1} + \frac{\zeta_{n-1}}{\alpha_{n-1}}(\boldsymbol{t}_{n-1} - \boldsymbol{r}_{n-1})\right], \\
\boldsymbol{t}_n &:= Q_n(A)\boldsymbol{r}_{n+1}^{\mathrm{BiCG}} = \boldsymbol{r}_n - \alpha_n A \boldsymbol{p}_n, \\
\boldsymbol{r}_{n+1} &:= Q_{n+1}(A)\boldsymbol{r}_{n+1}^{\mathrm{BiCG}} = \boldsymbol{t}_n - \zeta_n A \boldsymbol{t}_n, \\
\boldsymbol{x}_{n+1} &:= \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n + \zeta_n \boldsymbol{t}_n,
\end{aligned}$$

where $\boldsymbol{r}_{n+1}^{\mathrm{BiCG}}$ and $\boldsymbol{p}_n^{\mathrm{BiCG}}$ are the BiCG residual vector at $n+1$ iteration step and the BiCG search direction at n iteration step respectively, and the polynomial $Q_{n+1}(z)$ is defined by

$$Q_0(\lambda) := 1, \tag{4.32}$$

$$Q_{n+1}(\lambda) := (1 - \zeta_n \lambda) Q_n(\lambda), \quad n = 0, 1, \dots \tag{4.33}$$

As for the BiCGSTAB method, ζ_n is determined so that $\|\boldsymbol{r}_{n+1}\|$ is minimized.

Recall that the residual vector of the shifted BiCG method in Sect. 4.4.1 for $(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}$ is written as

$$\boldsymbol{r}_{n+1}^{\sigma,\mathrm{BiCG}} = \xi_{n+1}^\sigma \boldsymbol{r}_{n+1}^{\mathrm{BiCG}}, \quad \xi_{n+1}^\sigma \in \mathbb{C}.$$

Here, from (4.14), the scalar ξ_{n+1}^σ is defined by $\xi_{n+1}^\sigma := (\pi_{n+1}^\sigma)^{-1}$ for all n, and thus we have

$$\begin{aligned}
\xi_{n+1}^{\sigma} &= \frac{1}{\left(1+\frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n+\alpha_n\sigma\right)\pi_n^{\sigma}-\frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n\pi_{n-1}^{\sigma}} \\
&= \frac{1}{\left(1+\frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n+\alpha_n\sigma\right)(\xi_n^{\sigma})^{-1}-\frac{\beta_{n-1}}{\alpha_{n-1}}\alpha_n(\xi_{n-1}^{\sigma})^{-1}} \\
&= \frac{\xi_n^{\sigma}\xi_{n-1}^{\sigma}\alpha_{n-1}}{(1+\alpha_n\sigma)\,\xi_{n-1}^{\sigma}\alpha_{n-1}+\alpha_n\beta_{n-1}(\xi_{n-1}^{\sigma}-\xi_n^{\sigma})},
\end{aligned} \tag{4.34}$$

where $\xi_{-1}^{\sigma}=\xi_0^{\sigma}=1$.

We are now ready to describe the shifted BiCGSTAB method. The residual vector of the shifted BiCGSTAB method is defined by

$$\boldsymbol{r}_{n+1}^{\sigma,\mathrm{STAB}} := Q_{n+1}^{\sigma}(A)\boldsymbol{r}_{n+1}^{\sigma,\mathrm{BiCG}},$$

where $Q_{n+1}^{\sigma}(\lambda)$ are recursively defined as

$$Q_{n+1}^{\sigma}(\lambda) := \left[1-\zeta_n^{\sigma}(\lambda+\sigma)\right]Q_n^{\sigma}(\lambda), \quad n=0,1,\ldots \tag{4.35}$$

with $Q_0^{\sigma}(\lambda):=1$. Here, ζ_n^{σ} is determined so that the shifted BiCGSTAB residual and the BiCGSTAB residual are colinear, i.e.,

$$\boldsymbol{r}_{n+1}^{\sigma,\mathrm{STAB}} = c_n\boldsymbol{r}_{n+1}^{\mathrm{STAB}}, \quad c_n\in\mathbb{C}.$$

To this end, ζ_n^{σ} is determined so that

$$Q_{n+1}^{\sigma}(\lambda) = \tau_{n+1}^{\sigma}Q_{n+1}(\lambda), \quad \tau_{n+1}^{\sigma}\in\mathbb{C}. \tag{4.36}$$

Then, the residual vector of the shifted BiCGSTAB method is defined and written as

$$\begin{aligned}
\boldsymbol{r}_{n+1}^{\sigma,\mathrm{STAB}} :&= Q_{n+1}^{\sigma}(A)\boldsymbol{r}_{n+1}^{\sigma,\mathrm{BiCG}} \\
&= \tau_{n+1}^{\sigma}Q_{n+1}(A)\xi_{n+1}^{\sigma}\boldsymbol{r}_{n+1}^{\mathrm{BiCG}} \\
&= \tau_{n+1}^{\sigma}\xi_{n+1}^{\sigma}Q_{n+1}(A)\boldsymbol{r}_{n+1}^{\mathrm{BiCG}} \\
&= \tau_{n+1}^{\sigma}\xi_{n+1}^{\sigma}\boldsymbol{r}_{n+1}^{\mathrm{STAB}}.
\end{aligned} \tag{4.37}$$

In what follows, the parameters τ_{n+1}^{σ}, ξ_{n+1}^{σ} in (4.37) are determined. It follows from (4.33), (4.35), and (4.36) that

$$\underbrace{\tau_{n+1}^{\sigma}(1-\zeta_n\lambda)Q_n(\lambda)}_{Q_{n+1}^{\sigma}(\lambda)} = \left[1-\zeta_n^{\sigma}(\lambda+\sigma)\right]\underbrace{\tau_n^{\sigma}Q_n(\lambda)}_{Q_n^{\sigma}(\lambda)}. \tag{4.38}$$

Comparing the coefficients on both sides of (4.38) yields

$$\zeta_n \tau_{n+1}^\sigma = \zeta_n^\sigma \tau_n^\sigma, \quad \tau_{n+1}^\sigma = (1 - \zeta_n^\sigma \sigma)\tau_n^\sigma,$$

from which, we obtain

$$\tau_{n+1}^\sigma = \frac{\tau_n^\sigma}{1 + \zeta_n \sigma}, \quad \zeta_n^\sigma = \frac{\zeta_n}{1 + \zeta_n \sigma}, \tag{4.39}$$

where $\tau_0^\sigma = 1$ from the relation (4.36) and $Q_0^\sigma(\lambda) = Q_0(\lambda) = 1$. The shifted BiCGSTAB iterates for $(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}$ are now given by

$$\begin{aligned}
\boldsymbol{p}_n^\sigma &:= Q_n^\sigma(A)\boldsymbol{p}_n^{\sigma,\mathrm{BiCG}} = \boldsymbol{r}_n^\sigma + \beta_{n-1}^\sigma \left[\boldsymbol{p}_{n-1}^\sigma + \frac{\zeta_{n-1}^\sigma}{\alpha_{n-1}^\sigma}(\boldsymbol{t}_{n-1}^\sigma - \boldsymbol{r}_{n-1}^\sigma) \right] \\
&= \tau_n^\sigma \xi_n^\sigma \boldsymbol{r}_n + \beta_{n-1}^\sigma \left[\boldsymbol{p}_{n-1}^\sigma + \frac{\zeta_{n-1}^\sigma}{\alpha_{n-1}^\sigma} \tau_{n-1}^\sigma (\xi_n^\sigma \boldsymbol{t}_{n-1}^\sigma - \xi_{n-1}^\sigma \boldsymbol{r}_{n-1}) \right],
\end{aligned} \tag{4.40}$$

$$\boldsymbol{t}_n^\sigma := Q_n^\sigma(A)\boldsymbol{r}_{n+1}^{\sigma,\mathrm{BiCG}} = \tau_n^\sigma \xi_{n+1}^\sigma Q_n \boldsymbol{r}_{n+1}^{\mathrm{BiCG}} = \tau_n^\sigma \xi_{n+1}^\sigma \boldsymbol{t}_n, \tag{4.41}$$

$$\boldsymbol{r}_{n+1}^\sigma := Q_{n+1}^\sigma(A)\boldsymbol{r}_{n+1}^{\sigma,\mathrm{BiCG}} = \tau_{n+1}^\sigma \xi_{n+1}^\sigma Q_{n+1} \boldsymbol{r}_{n+1}^{\mathrm{BiCG}} = \tau_{n+1}^\sigma \xi_{n+1}^\sigma \boldsymbol{r}_{n+1}^{\mathrm{STAB}}, \tag{4.42}$$

$$\boldsymbol{x}_{n+1}^\sigma := \boldsymbol{x}_n^\sigma + \alpha_n^\sigma \boldsymbol{p}_n^\sigma + \zeta_n^\sigma \boldsymbol{t}_n^\sigma. \tag{4.43}$$

From (4.34), (4.39), (4.40)–(4.43), the shifted BiCGSTAB method is obtained, and the algorithm is written in Algorithm 4.8.

4.4.3 The Shifted GMRES Method

In this subsection, the shifted GMRES method [43] is derived. To this end, we consider applying the GMRES method to the shifted linear systems $(A + \sigma I)\boldsymbol{x} = \boldsymbol{b}$. For simplicity, the initial guess is set to $\boldsymbol{x}_0 = \boldsymbol{0}$. Then the GMRES method finds an approximate solution over the following Krylov subspace:

$$\boldsymbol{x}_n^\sigma \in \mathcal{K}_n(A + \sigma I, \boldsymbol{b}). \tag{4.44}$$

It follows from the shift-invariance property (4.4) that we have $\mathcal{K}_n(A + \sigma I, \boldsymbol{b}) = \mathcal{K}_n(A, \boldsymbol{b})$. Thus when using the Arnoldi process, the orthonormal basis vectors of $\mathcal{K}_n(A + \sigma I, \boldsymbol{b})$ are equivalent to those of $\mathcal{K}_n(A, \boldsymbol{b})$, i.e., V_n in Sect. 1.9.1. Then, (4.44) and the corresponding residual are written as

$$\boldsymbol{x}_n^\sigma = V_n \boldsymbol{y}_n, \quad \boldsymbol{y}_n \in \mathbb{C}^n$$

Algorithm 4.8 The shifted BiCGSTAB method

```
Input: $\boldsymbol{x}_0 = \mathbf{0}$, $\boldsymbol{p}_{-1} = \mathbf{0}$, $\boldsymbol{r}_0 = \boldsymbol{b}$, $\beta_{-1} = 0$
Input: Choose $\boldsymbol{r}_0^* \in \mathbb{C}^N$, e.g., $\boldsymbol{r}_0^* = \boldsymbol{r}_0$
Input: $\boldsymbol{x}_0^{(k)} = \mathbf{0}$, $\boldsymbol{p}_0^{(k)} = \boldsymbol{b}$, $\beta_{-1}^{(k)} = 0$, $\alpha_{-1}^{(k)} = \xi_{-1}^{(k)} = \xi_0^{(k)} = \tau_0^{(k)} = 1$ for $k = 1, 2, \dots, m$
Output: $\boldsymbol{x}_n^{(k)}$
 1: for $n = 0, 1, \dots$ do
 2:    $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \zeta_{n-1} A \boldsymbol{p}_{n-1})$
 3:    $\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, A\boldsymbol{p}_n)}$
 4:    $\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n A \boldsymbol{p}_n$
 5:    $\zeta_n = \frac{(A\boldsymbol{t}_n, \boldsymbol{t}_n)}{(A\boldsymbol{t}_n, A\boldsymbol{t}_n)}$
 6:    $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n + \zeta_n \boldsymbol{t}_n$
 7:    $\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \zeta_n A \boldsymbol{t}_n$
 8:    $\beta_n = \frac{\alpha_n}{\zeta_n} \times \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}$
 9:    {Begin shifted system}
10:    for $k = 1, 2, \dots, m$ do
11:       $\xi_{n+1}^{(k)} = \frac{\xi_n^{(k)} \xi_{n-1}^{(k)} \alpha_{n-1}}{(1+\alpha_n \sigma)\xi_{n-1}^{(k)} \alpha_{n-1} + \alpha_n \beta_{n-1} \left(\xi_{n-1}^{(k)} - \xi_n^{(k)}\right)}$
12:       $\alpha_n^{(k)} = \frac{\xi_{n+1}^{(k)}}{\xi_n^{(k)}} \alpha_k$
13:       $\zeta_n^{(k)} = \frac{\zeta_n}{1+\zeta_n \sigma}$
14:       $\boldsymbol{x}_{n+1}^{(k)} = \boldsymbol{x}_n^{(k)} + \alpha_n^{(k)} \boldsymbol{p}_n^{(k)} + \zeta_n^{(k)} \boldsymbol{t}_n^{(k)}$
15:       $\tau_{n+1}^{(k)} = \frac{\tau_n^{(k)}}{1+\zeta_n \sigma}$
16:       $\beta_n^{(k)} = \left(\frac{\xi_{n+1}^{(k)}}{\xi_n^{(k)}}\right)^2 \beta_k$
17:       $\boldsymbol{p}_{n+1}^{(k)} = \tau_{n+1}^{(k)} \xi_{n+1}^{(k)} \boldsymbol{r}_{n+1} + \beta_n^{(k)} \left[\boldsymbol{p}_n^{(k)} + \frac{\zeta_n^{(k)}}{\alpha_n^{(k)}} \tau_n^{(k)} \left(\xi_{n+1}^{(k)} \boldsymbol{t}_n^{(k)} - \xi_n^{(k)} \boldsymbol{r}_n\right)\right]$
18:    end for
19:    {End shifted system}
20: end for
```

and

$$
\begin{aligned}
\boldsymbol{r}_n^{\sigma} &= \boldsymbol{b} - (A + \sigma I)\boldsymbol{x}_n^{\sigma} \\
&= \|\boldsymbol{b}\| V_{n+1} \boldsymbol{e}_1 - A V_n \boldsymbol{y}_n - \sigma V_n \boldsymbol{y}_n \\
&= \|\boldsymbol{b}\| V_{n+1} \boldsymbol{e}_1 - V_{n+1} H_{n+1,n} \boldsymbol{y}_n - \sigma V_{n+1} \begin{bmatrix} \boldsymbol{y}_n \\ 0 \end{bmatrix} \\
&= \|\boldsymbol{b}\| V_{n+1} \boldsymbol{e}_1 - V_{n+1} H_{n+1,n} \boldsymbol{y}_n - \sigma V_{n+1} \begin{bmatrix} I_n \\ \mathbf{0}^{\top} \end{bmatrix} \boldsymbol{y}_n \\
&= V_{n+1} \left[\|\boldsymbol{b}\| \boldsymbol{e}_1 - \left(H_{n+1,n} + \begin{bmatrix} \sigma I_n \\ \mathbf{0}^{\top} \end{bmatrix} \right) \boldsymbol{y}_n \right] \\
&= V_{n+1} \left(\|\boldsymbol{b}\| \boldsymbol{e}_1 - H_{n+1,n}^{\sigma} \boldsymbol{y}_n \right),
\end{aligned}
$$

where I_n is the $n \times n$ identity matrix and $H^{\sigma}_{n+1,n}$ is a so-called shifted Hessenberg matrix whose elements are $(H^{\sigma}_{n+1,n})_{i,j} = (H_{n+1,n})_{i,j}$ for $i \neq j$ and $(H^{\sigma}_{n+1,n})_{i,j} = (H_{n+1,n})_{i,j} + \sigma$ for $i = j$. Then similar to (3.63), the residual norm can be minimized by solving

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

The above least-squares problems can be efficiently solved by Givens rotations that are described in Sect. 3.1.3. Using $\boldsymbol{y}^{\sigma}_n$, the approximate solution is now given by

$$\boldsymbol{x}^{\sigma}_n = V_n \boldsymbol{y}^{\sigma}_n.$$

If the coefficient matrix A is Hermitian and $\sigma \in \mathbb{R}$, then the shifted GMRES method reduces to the shifted MINRES method in Sect. 4.2.3. The algorithm of the shifted GMRES method is written in Algorithm 4.9.

Algorithm 4.9 The shifted GMRES method

```
Input: $\sigma_k$ $(k = 1, 2, \dots, m)$
Output: $\boldsymbol{x}^{(k)}_n$ $(k = 1, 2, \dots, m)$
1: $\boldsymbol{g} = (\|\boldsymbol{b}\|, 0, \dots, 0)^\top$, $\boldsymbol{v}_1 = \boldsymbol{b}/\|\boldsymbol{b}\|$
2: for $n = 1, 2, \dots$ do
3:    (Arnoldi process)
4:    $\boldsymbol{t} = A\boldsymbol{v}_n$
5:    for $i = 1, 2, \dots, n$ do
6:       $h_{i,n} = (\boldsymbol{v}_i, \boldsymbol{t})$
7:       $\boldsymbol{t} = \boldsymbol{t} - h_{i,n}\boldsymbol{v}_i$
8:    end for
9:    $h_{n+1,n} = \|\boldsymbol{t}\|$
10:   $\boldsymbol{v}_{n+1} = \boldsymbol{t}/h_{n+1,n}$
11:   {Begin shifted system}
12:   $h^{(k)}_{i,n} = h_{i,n}$ for $i \neq n$
13:   $h^{(k)}_{n,n} = h_{n,n} + \sigma_k$
14:   for $k = 1, 2, \dots, m$ do
15:      {Givens rotations}
16:      for $i = 1, 2, \dots, n-1$ do
17:         $\begin{bmatrix} h^{(k)}_{i,n} \\ h^{(k)}_{i+1,n} \end{bmatrix} = \begin{bmatrix} c^{(k)}_i & s^{(k)}_i \\ -\bar{s}^{(k)}_i & c^{(k)}_i \end{bmatrix} \begin{bmatrix} h^{(k)}_{i,n} \\ h^{(k)}_{i+1,n} \end{bmatrix}$
18:      end for
19:      $c^{(k)}_n = \frac{|h^{(k)}_{n,n}|}{\sqrt{|h^{(k)}_{n,n}|^2 + |h^{(k)}_{n+1,n}|^2}}$
20:      $\bar{s}_n = \frac{h^{(k)}_{n+1,n}}{h^{(k)}_{n,n}} c_n$
21:      $h^{(k)}_{n,n} = c^{(k)}_n h^{(k)}_{n,n} + s^{(k)}_n h^{(k)}_{n+1,n}$
22:      $h^{(k)}_{n+1,n} = 0$
23:      $\begin{bmatrix} g^{(k)}_n \\ g^{(k)}_{n+1} \end{bmatrix} = \begin{bmatrix} c^{(k)}_n & s^{(k)}_n \\ -\bar{s}^{(k)}_n & c^{(k)}_n \end{bmatrix} \begin{bmatrix} g^{(k)}_n \\ 0 \end{bmatrix}$
24:      (Check convergence)
25:      if $|g^{(k)}_{n+1}|/\|\boldsymbol{b}\| \le \epsilon$, then
26:         $\boldsymbol{x}^{(k)}_n = V_n \left(H^{(k)}_n\right)^{-1} \boldsymbol{g}^{(k)}$
27:      end if
28:   end for
29:   {End shifted system}
30: end for
```

The differences between the GMRES method and the shifted GMRES method are the initial guess $\boldsymbol{x}_0 = \boldsymbol{0}$ and line 13 in Algorithm 4.9 which corresponds to producing $H^{\sigma}_{n+1,n}$. Note that after Givens rotations H_n becomes the upper triangular matrix, and thus $\boldsymbol{c}^{(k)} := (H^{(k)}_n)^{-1}\boldsymbol{g}$ in line 26 can easily be obtained by solving $H^{(k)}_n \boldsymbol{c}^{(k)} = \boldsymbol{g}$.

As mentioned in Sect. 3.3.5, the GMRES method is not practical when the number of iterations is large, due to growing memory requirement and computational costs.

Instead, the restarted GMRES method is useful in practice, and was described in Algorithm 3.17. For the same reason, the shifted GMRES method is not practical when the number of iterations is large. It is therefore natural to consider the restarted version for the shifted linear systems. However, if we use the idea of Algorithm 3.17, then we face a difficulty in that after the restart the initial residual vectors $\boldsymbol{r}_0^{(i)}$ and $\boldsymbol{r}_0^{(j)}$ are, in general, not colinear any longer, and thus we cannot share the basis vectors of Krylov subspaces for solving shifted linear systems.

Frommer and Gräsner [68] nicely circumvented the difficulty. In what follows, the idea and the corresponding algorithm are described. Let $\boldsymbol{r}_n$ be the residual vector of the GMRES method for $A\boldsymbol{x} = \boldsymbol{b}$. Then they consider forcing the residual vector $\boldsymbol{r}_n^\sigma$ for the shifted system $(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}$ to become collinear with $\boldsymbol{r}_n$, i.e., find the approximate solution $\boldsymbol{x}^\sigma = \boldsymbol{x}_0^\sigma + V_n \boldsymbol{y}_n^\sigma$ such that

$$\boldsymbol{r}_n^\sigma = \beta_n \boldsymbol{r}_n.$$

$\boldsymbol{r}_n$ is written as

$$\boldsymbol{r}_n = \boldsymbol{b} - A\boldsymbol{x}_n = V_{n+1}\boldsymbol{z}_{n+1},$$

where $\boldsymbol{z}_{n+1} := \|\boldsymbol{r}_0\|\boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}_n$. We now have the following equation:

$$\begin{aligned}
\boldsymbol{r}_n^\sigma = \beta_n \boldsymbol{r}_n &\Leftrightarrow \boldsymbol{b} - A^\sigma \boldsymbol{x}_n^\sigma = \beta_n V_{n+1}\boldsymbol{z}_{n+1} \\
&\Leftrightarrow \boldsymbol{b} - A^\sigma(\boldsymbol{x}_0^\sigma + V_n \boldsymbol{y}_n^\sigma) = \beta_n V_{n+1}\boldsymbol{z}_{n+1} \\
&\Leftrightarrow \beta_0 \boldsymbol{r}_0 - A^\sigma V_n \boldsymbol{y}_n^\sigma = \beta_n V_{n+1}\boldsymbol{z}_{n+1} \\
&\Leftrightarrow \beta_0 \boldsymbol{r}_0 - V_{n+1} H_{n+1,n}^\sigma \boldsymbol{y}_n^\sigma = \beta_n V_{n+1}\boldsymbol{z}_{n+1} \\
&\Leftrightarrow \beta_0 \boldsymbol{r}_0 = V_{n+1}(\beta_n \boldsymbol{z}_{n+1} + H_{n+1,n}^\sigma \boldsymbol{y}_n^\sigma) \\
&\Leftrightarrow V_{n+1}(H_{n+1,n}^\sigma \boldsymbol{y}_n^\sigma + \beta_n \boldsymbol{z}_{n+1}) = \beta_0 \boldsymbol{r}_0 \\
&\Leftrightarrow V_{n+1}(H_{n+1,n}^\sigma \boldsymbol{y}_n^\sigma + \beta_n \boldsymbol{z}_{n+1}) = \beta_0 \|\boldsymbol{r}_0\| V_{n+1}\boldsymbol{e}_1 \\
&\Leftrightarrow H_{n+1,n}^\sigma \boldsymbol{y}_n^\sigma + \beta_n \boldsymbol{z}_{n+1} = \beta_0 \|\boldsymbol{r}_0\| \boldsymbol{e}_1,
\end{aligned}$$

from which β_n and $\boldsymbol{y}_n^\sigma$ are determined by solving the following $(n+1) \times (n+1)$ linear systems:

$$\begin{bmatrix} H_{n+1,n}^\sigma & \boldsymbol{z}_{n+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}_n^\sigma \\ \beta_n \end{bmatrix} = \beta_0 \|\boldsymbol{r}_0\| \boldsymbol{e}_1. \tag{4.45}$$

After the restart, the new initial residual vectors $\boldsymbol{r}_0 (= \boldsymbol{r}_n)$, $\boldsymbol{r}_0^\sigma (= \boldsymbol{r}_n^\sigma)$ become collinear. Thus we can use Krylov subspace $\mathcal{K}_n(A, \boldsymbol{b})$ for $A\boldsymbol{x} = \boldsymbol{b}$ to solve shifted linear systems $(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}^\sigma$.

The algorithm of the restarted shifted GMRES method is given in Algorithm 4.10.

Note that from the relation $\|\boldsymbol{r}_n^{(k)}\| = \|\beta_n^{(k)}\boldsymbol{r}_n\|$ one can check the convergence of $\|\boldsymbol{r}_n^{(k)}\|$ of Algorithm 4.10 by monitoring $|\beta_n^{(k)}| \times \|\boldsymbol{r}_n\|$. For further developments of the (restarted) shifted GMRES method, see [55, 105, 115, 175], and the references therein.

The full orthogonalization method (FOM) [149] is an extension of the CG method to solving non-Hermitian linear systems. As well as the CG method, the residual vector $\boldsymbol{r}_n^{\mathrm{FOM}}$ of the FOM satisfies

$$\boldsymbol{r}_n^{\mathrm{FOM}}(\in K_{n+1}(A,\boldsymbol{b})) \perp \mathcal{K}_n(A,\boldsymbol{b}).$$

Algorithm 4.10 The restarted shifted GMRES method

```
Input: $\sigma_k$ $(k = 1, 2, \dots, m)$
Input: $\boldsymbol{x}_0 \in \mathbb{C}^N$
Input: Set $\boldsymbol{x}_0^{(k)} \in \mathbb{C}^N$ such that $\boldsymbol{r}_0^{(k)} = \beta_0^{(k)}\boldsymbol{r}_0$, e.g., $\boldsymbol{x}_0 = \boldsymbol{x}_0^{(k)} = \boldsymbol{0}$ for all $k$.
Output: $\boldsymbol{x}_n^{(k)}$ $(k = 1, 2, \dots, m)$
 1: $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\beta = \|\boldsymbol{r}_0\|$
 2: Run the Arnoldi process in Algorithm 1.8 with $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$.
 3: Compute $\boldsymbol{y}_n$ such that $\|\beta\boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}_n\|$ is minimized.
 4: $\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n\boldsymbol{y}_n$, $\boldsymbol{z}_{n+1} = \beta\boldsymbol{e}_1 - H_{n+1,n}\boldsymbol{y}_n$
 5: {Shifted systems}
 6: for $k = 1, 2, \dots, m$ do
 7:     $H_{n+1,n}^{(k)} = H_{n+1,n} + \begin{bmatrix} \sigma_k I_n \\ \boldsymbol{0}^\top \end{bmatrix}$
 8:     Solve $\begin{bmatrix} H_{n+1,n}^{(k)} & \boldsymbol{z}_{n+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{y}_n^{(k)} \\ \beta_n^{(k)} \end{bmatrix} = \beta_0^{(k)}\beta\boldsymbol{e}_1$.
 9:     $\boldsymbol{x}_n^{(k)} = \boldsymbol{x}_0^{(k)} + V_m\boldsymbol{y}_m^{(k)}$
10: end for
11: if not convergence then
12:     Set $\boldsymbol{x}_0 = \boldsymbol{x}_n$, $\boldsymbol{x}_0^{(k)} = \boldsymbol{x}_n^{(k)}$, $\beta_0^{(k)} = \beta_n^{(k)}$, and go to line 1.
13: end if
```

When the FOM is applied to shifted linear systems, we have

$$\boldsymbol{r}_n^{\sigma,\mathrm{FOM}}(\in K_{n+1}(A+\sigma I,\boldsymbol{b})) \perp \mathcal{K}_n(A+\sigma I,\boldsymbol{b}).$$

Then from the shift invariance property $\mathcal{K}_n(A+\sigma I,\boldsymbol{b}) = \mathcal{K}_n(A,\boldsymbol{b})$, it follows that

$$\boldsymbol{r}_n^{\mathrm{FOM}}, \boldsymbol{r}_n^{\sigma,\mathrm{FOM}} \in K_{n+1}(A,\boldsymbol{b}) \cap K_n(A,\boldsymbol{b})^{\perp}.$$

From (4.5), the two residual vectors $\boldsymbol{r}_n^{\mathrm{FOM}}, \boldsymbol{r}_n^{\sigma,\mathrm{FOM}}$ belong to a one-dimensional subspace. Thus the residual vectors applied to $A\boldsymbol{x} = \boldsymbol{b}$ and $(A+\sigma I)\boldsymbol{x}^{\sigma} = \boldsymbol{b}$ are collinear. This implies that we do not need to consider the trick (4.45) for keeping collinearity when considering the restart, since the residual vectors are collinear at each iteration step. For the details of the restarted shifted FOM, see [158]. Further development of the restarted shifted FOM, see [55, 112], and the references therein.

4.4.4 *The Shifted IDR(s) Method*

The IDR(s) method in Sect. 3.3.9 is extended to solving shifted linear systems and the algorithm is referred to as the shifted IDR(s) method [48]. Prior to the derivation of the shifted IDR(s) method, a variant of the IDR theorem (Theorem 3.4) is described next.

Corollary 4.1 ([48]) *Let* $\boldsymbol{r}_0^{(i)}$ $(i = 1, \dots, m)$ *be collinear to each other,* $\mathcal{G}_0^{(i)} = \mathcal{K}_n(A + \sigma_i I, \boldsymbol{r}_0^{(i)})$*,* $\mathcal{S}$ *be a subspace of* $\mathbb{R}^n$*, and define sequences of subspaces* $\mathcal{G}_j^{(i)}$ *as*

$$\mathcal{G}_j^{(i)} = \left[I - \omega_j^{(i)}(A + \sigma_i I)\right](\mathcal{G}_{j-1}^{(i)} \cap \mathcal{S}), \quad \omega_j^{(i)} \neq 0, \quad j = 1, 2, \dots,$$

then it holds that $\mathcal{G}_j^{(1)} = \mathcal{G}_j^{(2)} = \cdots = \mathcal{G}_j^{(m)}$*.*

Corollary 4.1 implies that if we consider solving the following linear system (seed system) and shifted linear system:

$$A\boldsymbol{x} = \boldsymbol{b}, \quad (A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b},$$

then the collinear approach, as described in the shifted BiCGSTAB method, $\boldsymbol{r}_i = \pi_i^\sigma \boldsymbol{r}_i^\sigma$ is promising, since $\boldsymbol{r}_i = \boldsymbol{0}$ and $\pi_i^\sigma \neq 0$ lead to $\boldsymbol{r}_i^\sigma = \boldsymbol{0}$. Here $\boldsymbol{r}_i$ is the ith residual vector of the original linear system (seed system) $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{r}_i^\sigma$ is the ith residual vector of the shifted linear system $(A + \sigma I)\boldsymbol{x}^\sigma = \boldsymbol{b}$. We now describe how to compute $\boldsymbol{r}_{k+1}^\sigma$ from the information $\boldsymbol{r}_{k+1}$ when $\boldsymbol{r}_i = \pi_i^\sigma \boldsymbol{r}_i^\sigma$ $(\pi_i^\sigma \in \mathbb{C}, i = 0, 1, \dots, k)$. From the initial step of the IDR(s) method in Algorithm 3.24, the initial residual vectors of the seed system are given as

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \omega_k A\boldsymbol{r}_k, \quad (k = 0, \dots, s-1). \tag{4.46}$$

Similar to (4.46), the initial residual vectors of the shifted system are as follows:

$$\boldsymbol{r}_{k+1}^\sigma = \boldsymbol{r}_k^\sigma - \omega_k^\sigma (A + \sigma I)\boldsymbol{r}_k^\sigma = (1 - \sigma\omega_k^\sigma)\boldsymbol{r}_k^\sigma - \omega_k^\sigma A\boldsymbol{r}_k^\sigma, \tag{4.47}$$

where the parameters $\omega_k^\sigma \in \mathbb{C}$ are unknown.

Substituting the relations $\boldsymbol{r}_i = \pi_i^\sigma \boldsymbol{r}_i^\sigma$ $(i = 0, \dots, k)$ into (4.46) yields

$$\boldsymbol{r}_{k+1}^\sigma = \frac{\pi_k^\sigma}{\pi_{k+1}^\sigma}\boldsymbol{r}_k^\sigma - \omega_k \frac{\pi_k^\sigma}{\pi_{k+1}^\sigma} A\boldsymbol{r}_k^\sigma. \tag{4.48}$$

From the relations $\boldsymbol{r}_i = \pi_i^\sigma \boldsymbol{r}_i^\sigma$, (4.46) and residual polynomial $R_i(0) = 1$, $\boldsymbol{r}_i = R_i(A)\boldsymbol{r}_0$, it follows that

$$\begin{cases} \pi_k^\sigma = R_k(-\sigma), \\ \pi_{k+1}^\sigma = \pi_k^\sigma + \omega_k \sigma \pi_k^\sigma. \end{cases}$$

Thus, parameter π_{k+1}^σ in (4.48) can be obtained when π_k^σ is computed.

Comparing the coefficients of (4.47) and (4.48), we obtain $\omega_k^\sigma = \omega_k \pi_k^\sigma / \pi_{k+1}^\sigma$. The corresponding approximate solution of the shifted system is written as

$$\boldsymbol{x}_{k+1}^\sigma = \boldsymbol{x}_k^\sigma + \omega_k^\sigma \boldsymbol{r}_k^\sigma. \tag{4.49}$$

Next, we describe the derivation process of $\boldsymbol{r}_{k+1}^\sigma$ by using $\boldsymbol{r}_{k+1}$ in the main step of the IDR(s) method in Algorithm 3.24. From (3.110) and (3.111), $\boldsymbol{v}_k$ and $\boldsymbol{r}_{k+1}$ of the IDR(s) method are written as

$$\boldsymbol{v}_k = \boldsymbol{r}_k - \sum_{l=1}^{s} \gamma_l \Delta \boldsymbol{r}_{k-l} = (1-\gamma_1)\boldsymbol{r}_k + \sum_{l=1}^{s-1} (\gamma_l - \gamma_{l+1}) v r_{k-l} + \gamma_s \boldsymbol{r}_{k-s}, \tag{4.50}$$

$$\boldsymbol{r}_{k+1} = (I - \omega_j A)\boldsymbol{v}_k. \tag{4.51}$$

It follows from (4.50) and (4.51) that we obtain

$$\boldsymbol{r}_{k+1} = (I - \omega_j A)\left((1-\gamma_1)\boldsymbol{r}_k + \sum_{l=1}^{s-1} (\gamma_l - \gamma_{l+1}) \boldsymbol{r}_{k-l} + \gamma_s \boldsymbol{r}_{k-s} \right). \tag{4.52}$$

Residual vectors of the shifted system in the same form of (4.52) are defined as follows:

$$\begin{aligned} \boldsymbol{r}_{k+1}^\sigma &= \left(I - \omega_j^\sigma (A + \sigma I)\right) \left((1-\gamma_1^\sigma)\boldsymbol{r}_k^\sigma + \sum_{l=1}^{s-1} (\gamma_l^\sigma - \gamma_{l+1}^\sigma) \boldsymbol{r}_{k-l}^\sigma + \gamma_s^\sigma \boldsymbol{r}_{k-s}^\sigma \right) \\ &= \left(I - \frac{\omega_j^\sigma}{1 - \sigma\omega_j^\sigma} A \right) (1 - \sigma \omega_j^\sigma) \left((1-\gamma_1^\sigma)\boldsymbol{r}_k^\sigma + \sum_{l=1}^{s-1} (\gamma_l^\sigma - \gamma_{l+1}^\sigma) \boldsymbol{r}_{k-l}^\sigma + \gamma_s^\sigma \boldsymbol{r}_{k-s}^\sigma \right), \end{aligned} \tag{4.53}$$

where parameters $\omega_j^\sigma, \gamma_1^\sigma, \ldots, \gamma_s^\sigma \in \mathbb{C}$ are unknown. Substituting $\boldsymbol{r}_i = \pi_i^\sigma \boldsymbol{r}_i^\sigma$ ($i = 0, 1, \ldots, k+1$) into (4.52) yields

$$\boldsymbol{r}_{k+1}^\sigma = (I - \omega_j A)\left((1-\gamma_1)\frac{\pi_k^\sigma}{\pi_{k+1}^\sigma}\boldsymbol{r}_k^\sigma + \sum_{l=1}^{s-1} (\gamma_l - \gamma_{l+1}) \frac{\pi_{k-l}^\sigma}{\pi_{k+1}^\sigma} \boldsymbol{r}_{k-l}^\sigma + \gamma_s \frac{\pi_{k-s}^\sigma}{\pi_{k+1}^\sigma} \boldsymbol{r}_{k-s}^\sigma \right). \tag{4.54}$$

It follows from (4.52), $\boldsymbol{r}_i = \pi_i^\sigma \boldsymbol{r}_i^\sigma \, (i = 0, 1, \ldots, k+1)$, and $R_i(0) = 1$ that we obtain π_{k+1}^σ as follows:

$$\pi_{k+1}^\sigma = (1 + \omega_j \sigma) \left((1 - \gamma_1)\pi_k^\sigma + \sum_{l=1}^{s-1} (\gamma_l - \gamma_{l+1})\pi_{k-l}^\sigma + \gamma_s \pi_{k-s}^\sigma \right). \tag{4.55}$$

Comparing the corresponding coefficients of (4.53) and (4.54) gives

$$\begin{cases} \frac{\omega_j^\sigma}{1-\sigma\omega_j^\sigma} = \omega_j, \\ 1 - \gamma_1^\sigma = \frac{1-\gamma_1}{1-\sigma\omega_j^\sigma} \times \frac{\pi_k^\sigma}{\pi_{k+1}^\sigma}, \\ \gamma_l^\sigma - \gamma_{l+1}^\sigma = \frac{\gamma_l - \gamma_{l+1}}{1-\sigma\omega_j^\sigma} \times \frac{\pi_{k-l}^\sigma}{\pi_{k+1}^\sigma}, \\ \gamma_s^\sigma = \frac{\gamma_s}{1-\sigma\omega_j^\sigma} \times \frac{\pi_{k-s}^\sigma}{\pi_{k+1}^\sigma} \end{cases} \tag{4.56}$$

for $l = 1, 2, \ldots, s-1$. From (4.56), parameters $\omega_k^\sigma, \gamma_1^\sigma, \ldots, \gamma_s^\sigma$ are determined as

$$\begin{cases} \omega_j^\sigma = \frac{\omega_j}{1+\sigma\omega_j}, \\ \gamma_1^\sigma = 1 - (1-\gamma_1)(1+\sigma\omega_j)\frac{\pi_k^\sigma}{\pi_{k+1}^\sigma}, \\ \gamma_{l+1}^\sigma = \gamma_l^\sigma - (\gamma_l - \gamma_{l+1})(1+\sigma\omega_j)\frac{\pi_{k-l}^\sigma}{\pi_{k+1}^\sigma} \end{cases} \tag{4.57}$$

for $l = 1, 2, \ldots, s-1$.

Using the relation $\boldsymbol{r}_{k+1}^\sigma = \boldsymbol{b} - (A + \sigma I)\boldsymbol{x}_{k+1}^\sigma$, the approximate solutions $\boldsymbol{x}_{k+1}^\sigma$ can be derived from (4.53) as

$$\boldsymbol{x}_{k+1}^\sigma = \boldsymbol{x}_k^\sigma + \omega_j^\sigma \boldsymbol{v}_k^\sigma - \sum_{l=1}^{s} \gamma_l^\sigma \Delta \boldsymbol{x}_{k-l}^\sigma, \tag{4.58}$$

where $\boldsymbol{v}_k^\sigma = \boldsymbol{r}_k^\sigma - \sum_{l=1}^s \gamma_l^\sigma \Delta \boldsymbol{r}_{k-l}^\sigma$. For practical computation, one can express $\boldsymbol{v}_k^\sigma$ by the residual vectors $\boldsymbol{r}_{k-s}, \ldots, \boldsymbol{r}_k$ of the seed system.

All the steps above give the shifted IDR(s) method as described in Algorithm 4.11.

Algorithm 4.11 The shifted IDR(s) method (Seed system: $(A+\sigma_f I)\boldsymbol{x}=\boldsymbol{b}$)

```
Input: $\boldsymbol{x}_0^{(i)}=\boldsymbol{0}, \boldsymbol{r}_0=\boldsymbol{b}, P\in\mathbb{C}^{n\times s}, \pi_0^{(f,i)}=1, \sigma_i$ for $i=1,\dots,m$
Output: $\boldsymbol{x}^{(i)}$ for $i=1,\dots,m$
1: for $k=0,1,\dots,s-1$ do
2:   $\boldsymbol{v}=(A+\sigma_f I)\boldsymbol{r}_k, \omega=\boldsymbol{v}^{\mathrm{H}}\boldsymbol{r}_k/\boldsymbol{v}^{\mathrm{H}}\boldsymbol{v}$
3:   $\Delta X(:,k+1)=\omega\boldsymbol{r}_k, \Delta R(:,k+1)=-\omega\boldsymbol{v}$
4:   $\boldsymbol{x}_{k+1}^{(f)}=\boldsymbol{x}_k^{(f)}+\Delta X(:,k+1), \boldsymbol{r}_{k+1}=\boldsymbol{r}_k+\Delta R(:,k+1)$
5:   (Iteration for shifted system)
6:   for $i(\neq f)=1,2,\dots,m$ do
7:     if $\|\boldsymbol{r}_k^{(i)}\|>\varepsilon\|\boldsymbol{b}\|$ then
8:       $\pi_{k+1}^{(f,i)}=\pi_k^{(f,i)}+\omega(\sigma_i-\sigma_f)\pi_k^{(f,i)}, \omega^{(f,i)}=\frac{\omega}{1+\omega(\sigma_i-\sigma_f)}$
9:       $\boldsymbol{x}_{k+1}^{(i)}=\boldsymbol{x}_k^{(i)}+\frac{\omega^{(f,i)}}{\pi_k^{(f,i)}}\boldsymbol{r}_k$
10:    end if
11:  end for
12: end for
13: $j=1, k=s, M=P^{\mathrm{H}}\Delta R, \boldsymbol{h}=P^{\mathrm{H}}\boldsymbol{r}_k$
14: while stopping criterion is not satisfied do
15:   for $l=0,1,\dots,s$ do
16:     Solve $c$ from $M\boldsymbol{c}=\boldsymbol{h}$
17:     $\boldsymbol{q}=-\Delta R\boldsymbol{c}, \boldsymbol{v}=\boldsymbol{r}_k+\boldsymbol{q}$
18:     if $l=0$ then
19:       $\boldsymbol{t}=(A+\sigma_f I)\boldsymbol{v}, \omega=\boldsymbol{t}^{\mathrm{H}}\boldsymbol{v}/\boldsymbol{t}^{\mathrm{H}}\boldsymbol{t}$
20:       $\Delta R(:,j)=\boldsymbol{q}-\omega\boldsymbol{t}, \Delta X(:,j)=-\Delta X\boldsymbol{c}+\omega\boldsymbol{v}$
21:     else
22:       $\Delta X(:,j)=-\Delta X\boldsymbol{c}+\omega\boldsymbol{v}, \Delta R(:,j)=-(A+\sigma_f I)\Delta X(:,j)$
23:     end if
24:     $\boldsymbol{r}_{k+1}=\boldsymbol{r}_k+\Delta R(:,j), \boldsymbol{x}_{k+1}^{(f)}=\boldsymbol{x}_k^{(f)}+\Delta X(:,j)$
25:     $\delta\boldsymbol{m}=P^{\mathrm{H}}\Delta R(:,j), M(:,j)=\delta\boldsymbol{m}, \boldsymbol{h}=\boldsymbol{h}+\delta\boldsymbol{m}$
26:     (Iteration for shifted system)
27:     $\gamma_1=\boldsymbol{c}_{j-1}, \gamma_2=\boldsymbol{c}_{j-2},\dots,\gamma_{j-1}=\boldsymbol{c}_1$
28:     $\gamma_j=\boldsymbol{c}_s, \gamma_{j+2}=\boldsymbol{c}_{s-1},\dots,\gamma_s=\boldsymbol{c}_j$
29:     for $i(\neq f)=1,2,\dots,m$ do
30:       if $\|\boldsymbol{r}_k^{(i)}\|>\varepsilon\|\boldsymbol{b}\|$ then
31:         $\alpha_i=1+\omega(\sigma_i-\sigma_f)$
32:         $\pi_{k+1}^{(f,i)}=\alpha_i\left((1-\gamma_1)\pi_k^{(f,i)}+\sum_{g=1}^{s-1}(\gamma_g-\gamma_{g+1})\pi_{k-g}^{(f,i)}+\gamma_s\pi_{k-s}^{(f,i)}\right)$
33:         $\gamma_1^{(f,i)}=1-\alpha_i(1-\gamma_1)\pi_k^{(f,i)}/\pi_{k+1}^{(f,i)}$
34:         $\gamma_{g+1}^{(f,i)}=\gamma_g^{(f,i)}-(\gamma_g-\gamma_{g+1})\alpha_i\pi_{k-g}^{(f,i)}/\pi_{k+1}^{(f,i)}\ (g=1,\dots,s-1)$
35:         $\boldsymbol{x}_{k+1}^{(i)}=\boldsymbol{x}_k^{(i)}+\omega\boldsymbol{v}/\pi_{k+1}^{(f,i)}-\sum_{g=1}^{s}\gamma_g^{(f,i)}(\boldsymbol{x}_{k+1-g}^{(i)}-\boldsymbol{x}_{k-g}^{(i)})$
36:       end if
37:     end for
38:     $k=k+1, j=j+1$
39:     $j=(j-1)\%s+1$   (%: modulo operation, i.e. $a\%n=r$, where $a=mn+r$.)
40:   end for
41: end while
```

Chapter 5
Applications to Matrix Functions

The square root of a positive number received attention in the ancient world. Indeed, an approximation to $\sqrt{2}$ is found in the Yale Babylonian Collection YBC 7289 clay tablet, which was created between 1800 BC and 1600 BC.

On the other hand, the notion of matrix functions such as matrix square root is relatively new: the notion of the square root of a matrix was found by Cayley in 1858 [33], and a definition of matrix functions was given by Sylvester in 1883 [180].

Nowadays, matrix functions arise in many scientific fields such as particle physics, quantum information, and control theory, thus efficient numerical algorithms have been developed by many researchers.

In this chapter, the definition of matrix functions is described, and then numerical algorithms of matrix functions are described such as matrix square root, matrix pth root, matrix exponential, matrix logarithm, and matrix trigonometric function. We will see that Krylov subspace methods or shifted Krylov subspace methods can be useful for computing specific elements of the large matrix functions. The best-known book on matrix functions is Higham's book [97]. In what follows, the explanations of matrix functions are based on [36, 97, 154], and the size of matrix A is n-by-n.

5.1 Jordan Canonical Form

Among some equivalent definitions of matrix functions, we adopt the definition using Jordan canonical form. In this section, Jordan canonical form is explained.

Definition 5.1 (Jordan block) The following $m \times m$ square matrix is referred to as a Jordan block:

T. Sogabe, *Krylov Subspace Methods for Linear Systems*,
Springer Series in Computational Mathematics 60,
https://doi.org/10.1007/978-981-19-8532-4_5

$$J_k = \begin{bmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{bmatrix} \in \mathbb{C}^{m\times m},$$

where $\lambda_k \in \mathbb{C}$. Since J_k is the upper triangular matrix, all the eigenvalues of J_k are the same as the diagonal elements, i.e., λ_k.

Definition 5.2 (Jordan matrix) The direct sum of Jordan blocks $J_1 \in \mathbb{C}^{m_1\times m_1}$, $J_2 \in \mathbb{C}^{m_2\times m_2}, \dots, J_p \in \mathbb{C}^{m_p\times m_p}$ is given by

$$J = \mathrm{diag}(J_1, J_2, \dots, J_p) = \begin{bmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_p \end{bmatrix} \in \mathbb{C}^{n\times n}. \tag{5.1}$$

The matrix is referred to as a *Jordan matrix*. Here, $n = m_1 + m_2 + \cdots + m_p$, and the symbol $\mathrm{diag}(J_1, J_2, \dots, J_p)$ denotes a matrix whose diagonal blocks are $J_1, J_2, \dots, J_p$.

Example 1 Let $p = 3$, $m_1 = 2$, $m_2 = 1$, $m_3 = 3$, $\lambda_1 = 1$, $\lambda_2 = \lambda_3 = 2$. Then $n = m_1 + m_2 + m_3 = 6$, and we have the following 6×6 Jordan matrix:

$$J = \mathrm{diag}(J_1, J_2, J_3) = \left[\begin{array}{cc|c|ccc} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{array}\right].$$

Theorem 5.1 (Jordan canonical form) *A square matrix $A \in \mathbb{C}^{n\times n}$ is similar to a Jordan matrix, i.e., for any square matrix A, there exists a nonsingular matrix Z such that*

$$Z^{-1}AZ = J.$$

The form $A = ZJZ^{-1}$ is referred to as the Jordan canonical form of A.

Example 2 The following matrix is similar to the Jordan matrix J in Example 1.

$$A = \begin{bmatrix} -1 & 1 & -1 & -2 & -2 & -1 \\ 1 & 2 & 2 & 1 & 2 & 1 \\ -3 & -3 & -4 & -3 & -5 & -3 \\ 4 & 0 & 3 & 5 & 4 & 2 \\ 4 & 4 & 7 & 4 & 8 & 4 \\ -3 & -2 & -5 & -3 & -5 & -1 \end{bmatrix}.$$

Indeed, we see that $Z^{-1}AZ = J$, where

$$Z = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 1 & -1 \\ 0 & -1 & 2 & 0 & -2 & 1 \\ -1 & 0 & 0 & 2 & 0 & -1 \\ 0 & 1 & -2 & 0 & 3 & -2 \\ 0 & -1 & 1 & -1 & -2 & 4 \end{bmatrix}, \quad Z^{-1} = \begin{bmatrix} 3 & 1 & 1 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 & 1 & 1 \\ 1 & 2 & 3 & 1 & 2 & 1 \\ 2 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Among several definitions of matrix functions, a definition using Jordan canonical form in Theorem 5.1 is adopted in the next section.

5.2 Definition and Properties of Matrix Functions

The set of all the eigenvalues of matrix A is called the *spectrum* of A. The definition of a *function on the spectrum of a matrix* is given as follows.

Definition 5.3 (function on the spectrum of a matrix) Let $\lambda_1, \lambda_2, \dots, \lambda_s$ be distinct eigenvalues of matrix A, and let n_i be the size of a Jordan block with respect to eigenvalue λ_i of A. If there exist the following values:

$$f(\lambda_i), \frac{\mathrm{d}}{\mathrm{d}x} f(\lambda_i), \dots, \frac{\mathrm{d}^{n_i-1}}{\mathrm{d}x^{n_i-1}} f(\lambda_i) \quad (i = 1, 2, \dots, s),$$

then $f(x)$ is called *a function on the spectrum of matrix A*.

In what follows, we use the symbol $f^{(i)}(x)$ instead of $\frac{\mathrm{d}^i}{\mathrm{d}x^i} f(x)$.

Example 3 Let A be matrix J in Example 1. Then the number of distinct eigenvalues is $s = 2$ since $\lambda_1 = \lambda_2 = 1$ and $\lambda_3 = 2$. The maximum sizes of Jordan blocks corresponding to $\lambda_1 = \lambda_2 = 1$ and $\lambda_3 = 2$ are $n_1 = 2$ and $n_2 = 3$. The spectrum of matrix A is $\{\lambda_1, \lambda_3\}$. We now give an example of a function on the spectrum of matrix A. Let $f(x) = x^{-1}$. Then $f(x)$ is a function on the spectrum of matrix A, because $f^{(1)}(x) = -x^{-2}$, $f^{(2)}(x) = 2x^{-3}$ and there exist the values of the functions on the spectrum as follows: $f(\lambda_1) = 1$, $f^{(1)}(\lambda_1) = -1$, $f(\lambda_3) = 1/2$, $f^{(1)}(\lambda_3) = -1/4$, and $f^{(2)}(\lambda_3) = 1/4$.

A matrix function of A can be defined by using Jordan canonical form in Theorem 5.1 and a function on the spectrum of a matrix in Definition 5.3 as follows:

Definition 5.4 (Matrix function) Let $f(x)$ be a function on the spectrum of matrix $A \in \mathbb{C}^{n\times n}$ and $A = ZJZ^{-1}$ be Jordan canonical form of A. Then, matrix function $f(A) = f(ZJZ^{-1})$ is defined as follows:

$$f(A) = Z\text{diag}(f(J_1), f(J_2), \ldots, f(J_p))Z^{-1}.$$

Here $f(J_k)$ is the following $m_k \times m_k$ matrix:

$$f(J_k) = \begin{bmatrix} f(\lambda_k) & \frac{f^{(1)}(\lambda_k)}{1!} & \cdots & \frac{f^{(m_k-1)}(\lambda_k)}{(m_k-1)!} \\ & f(\lambda_k) & \ddots & \vdots \\ & & \ddots & \frac{f^{(1)}(\lambda_k)}{1!} \\ & & & f(\lambda_k) \end{bmatrix} \in \mathbb{C}^{m_k\times m_k},$$

and $m_k \times m_k$ is the size of Jordan block corresponding to eigenvalue λ_k.

It follows from Definition 5.4 that when λ_k is an eigenvalue of matrix A, the eigenvalue of matrix function $f(A)$ is $f(\lambda_k)$.

In particular, when matrix A is diagonalizable, the Jordan blocks are diagonal matrices D_k for $k = 1, 2, \ldots, p$. Then it follows from $f(A) = Z\text{diag}(D_1, D_2, \ldots, D_p)Z^{-1}$ that $f(A)$ and A have the same eigenvectors.

Example 4 Let A be matrix J in Example 1 and let $f(x) = x^{-1}$. Jordan canonical form of A is given by $A = ZJZ^{-1}$, where Z is the identity matrix. Then $f(A) = f(ZJZ^{-1}) = f(J)$ and thus from $\lambda_1 = \lambda_2 = 1, \lambda_3 = 2$ and Definition 5.4, we have

$$f(A) = \left[\begin{array}{cc|c|ccc} f(\lambda_1) & \frac{f^{(1)}(\lambda_1)}{1!} & 0 & 0 & 0 & 0 \\ 0 & f(\lambda_1) & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & f(\lambda_2) & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & f(\lambda_3) & \frac{f^{(1)}(\lambda_3)}{1!} & \frac{f^{(2)}(\lambda_3)}{2!} \\ 0 & 0 & 0 & 0 & f(\lambda_3) & \frac{f^{(1)}(\lambda_3)}{1!} \\ 0 & 0 & 0 & 0 & 0 & f(\lambda_3) \end{array}\right]$$

$$= \left[\begin{array}{cc|c|ccc} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 2^{-1} & -4^{-1} & 8^{-1} \\ 0 & 0 & 0 & 0 & 2^{-1} & -4^{-1} \\ 0 & 0 & 0 & 0 & 0 & 2^{-1} \end{array}\right].$$

It is easy to see that $f(A)$ is the inverse of matrix A, i.e., A^{-1} was derived from the definition of the matrix function $f(A)$.

Example 5 We consider other examples of matrix functions of A. Let

$$A = \begin{bmatrix} -1 & 6 & -9 \\ 4 & -4 & 12 \\ 3 & -5 & 11 \end{bmatrix}.$$

Then the Jordan canonical form is given by

$$A = ZJZ^{-1},$$

where

$$Z^{-1}AZ = J = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

and

$$Z = \begin{bmatrix} 3 & 0 & -1 \\ 0 & 2 & 1 \\ -1 & 1 & 1 \end{bmatrix}, \quad Z^{-1} = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 2 & -3 \\ 2 & -3 & 6 \end{bmatrix}.$$

Using these matrices, $A^{1/2}$ corresponding to $f(x) = x^{1/2}$ is given by

$$A^{1/2} = Z \begin{bmatrix} \sqrt{2} & \frac{1}{2\sqrt{2}} & -\frac{1}{16\sqrt{2}} \\ 0 & \sqrt{2} & \frac{1}{2\sqrt{2}} \\ 0 & 0 & \sqrt{2} \end{bmatrix} Z^{-1}.$$

It is easy to see that $A^{1/2}A^{1/2} = A$.

The last example is a matrix exponential function of A. The matrix exponential function e^A corresponding to $f(x) = \mathrm{e}^x$ is given by

$$\mathrm{e}^A = Z \begin{bmatrix} \mathrm{e}^2 & \mathrm{e}^2 & \frac{\mathrm{e}^2}{2} \\ 0 & \mathrm{e}^2 & \mathrm{e}^2 \\ 0 & 0 & \mathrm{e}^2 \end{bmatrix} Z^{-1}.$$

Notice that computing Jordan canonical form is numerically unstable. These examples, therefore, are not for practical computations.

We enumerate some properties of matrix functions. For details see, e.g., [97].

Theorem 5.2 (Properties of matrix functions) *Let $f(x)$ be a function on the spectrum of matrix A. Then:*

(1) $Af(A) = f(A)A$.
(2) $XA = AX \Rightarrow Xf(A) = f(A)X$.

(3) $f(A^\top) = f(A)^\top$.
(4) $f(XAX^{-1}) = Xf(A)X^{-1}$.
(5) *If* λ_k *is an eivenvalue of* A*, then* $f(\lambda_k)$ *is an eigenvalue of* $f(A)$*.*

Let $f(x)$ *and* $g(x)$ *be functions on the spectrum of matrix* A*. Then:*

(6) *If* $(f+g)(x) := f(x) + g(x)$*, then* $(f+g)(A) = f(A) + g(A)$*.*
(7) *If* $(fg)(x) := f(x)g(x)$*, then* $(fg)(A) = f(A)g(A)$*.*

Using Definition 5.3, it follows that

(8) $f(A) = g(A)$ *if and only if* $f(\lambda_i) = g(\lambda_i), \dots, f^{(n_i-1)}(\lambda_i) = g^{(n_i-1)}(\lambda_i)$ $(i = 1, 2, \dots, s)$.

From Definition 5.4, we can define $A^{1/n}$, e^A, $\sin A$, $\cos A$, $\log A$ that correspond to elementary functions $x^{1/n}$, e^x, $\sin x$, $\cos x$, $\log x$. However, since numerically computing the Jordan canonical form is quite unstable, it is not recommended to compute matrix functions via the Jordan canonical form. Therefore, many researchers have devised numerically stable algorithms for matrix functions.

In what follows, some numerical algorithms are described for computing matrix functions. We will see that the notion of a Krylov subspace is useful for computing some of the matrix functions.

5.3 Matrix Square Root and Matrix *p*th Root

5.3.1 Matrix Square Root

Square roots of real number $a > 0$ are $a^{1/2}$ and $-a^{1/2}$, and the principal square root is $a^{1/2}$. The notion of the principal square root is extended to complex numbers as follows: let z be a complex number, and for $z = re^{i\theta}$ $(r > 0, -\pi < \theta < \pi)$, we define $z^{1/2} = r^{1/2}e^{i\theta/2}$. This is called the principal square root of z.

Now we consider the case of matrix square roots and define the principal square root of A. X is called a square root of A if $X^2 = A$. The square roots of a matrix differ from square roots of complex or real numbers in that there may be infinitely many square roots. For example, let

$$S_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad S_2 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad S(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix}.$$

Then $S_1^2 = S_2^2 = S(\theta)^2 = I$ for any θ. These are square roots of the identity matrix. Thus this is an example for which there are infinitely many square roots.

While there are infinitely many square roots, there is a unique square root whose eigenvalues lie in the right half-plane of the complex plane. The unique square root is called the principal matrix square root. In the above example, S_1 is the principal matrix square root of the identity matrix I.

The next theorem presents the notion of the principal matrix square root.

Theorem 5.3 (principal matrix square root) *Let A be a complex square matrix whose eigenvalues ($\lambda_k \neq 0$) do not lie in the negative real axis of the complex plane. Then there exists a unique square root of A such that all the eigenvalues of the square root lie in the right half-plane of the complex plane.*

In Theorem 5.3, the unique matrix square root is referred to as *the principal matrix square root* of A, and denoted by $A^{1/2}$.

In what follows, numerical algorithms, a direct method, and an iterative method are described for the (principal) matrix square root.

First, a well-known direct method based on the Schur decomposition is described. The Schur decomposition is to decompose matrix A as QTQ^{H}, where T is an upper triangular matrix, Q is a unitary matrix. If we can compute $U := T^{1/2}$, then $(QUQ^{\mathrm{H}})^2 = QU^2Q^{\mathrm{H}} = QTQ^{\mathrm{H}} = A$. Thus the (principal) matrix square root is given by $A^{1/2} = QUQ^{\mathrm{H}}$. Since U is also an upper tridiagonal matrix, U can be obtained by sequentially solving $U^2 = T$. The resulting algorithm is shown in Algorithm 5.1. Here (i, j) element of matrix T (and U) is denoted by t_{ij} (and u_{ij}).

Algorithm 5.1 (Direct method) The Schur method for $X = A^{1/2}$

```
1: Compute the Schur decomposition of $A$, i.e., $A = QTQ^{\mathrm{H}}$.
2: for $i = 1, 2, \ldots, n$ do
3:     $u_{ii} = t_{ii}^{1/2}$
4: end for
5: for $j = 2, 3, \ldots, n$ do
6:     for $i = j-1, j-2, \ldots, 1$ do
7:         $u_{ij} = \frac{1}{u_{ii}+u_{jj}}(t_{ij} - \sum_{k=i+1}^{j-1} u_{ik}u_{kj})$
8:     end for
9: end for
10: $X = QUQ^{\mathrm{H}}$
```

Assume that matrix A satisfies the condition in Theorem 5.3. Then from line 3 in Algorithm 5.1, we see that $u_{ii} = t_{ii}^{1/2} (i = 1, 2, \ldots, n)$ and u_{ii} is the principal square root of complex number t_{ii}. From the assumption of matrix A, all the u_{ii}'s lie in the right half-plane of the complex plane. Thus $u_{ii} + u_{jj} \neq 0$, which is the denominator in line 7 in Algorithm 5.1. Thus Algorithm 5.1 never suffers from breakdown.

Next, some iterative methods are described. The iterative methods can be regarded as Newton's method (Newton–Raphson method) for $X^2 = A$. We now give a well-known derivation of Newton's method for matrix square roots.

Let X_k be an approximate solution of $A^{1/2}$ and let E be the corresponding error matrix such that $X_k + E = A^{1/2}$. If we can obtain E so that $(X_k + E)^2 = (A^{1/2})^2 \Leftrightarrow X_k^2 + EX_k + X_kE + E^2 = A$, then we have $X_k + E = A^{1/2}$. But it may be more difficult to solve the original problem since there are additional terms $EX_k + X_kE$. On the other hand, from the assumption that $X_k \approx A^{1/2}$, we can expect that $E \approx O$ (zero matrix). Thus instead of considering the correction equation above, we consider obtaining E_k from $X_k^2 + E_kX_k + X_kE_k = A$, which is a Sylvester equation in (2.59). Then we have the following Newton's method:

```
1. Set an initial guess $X_0$,
2. For $k = 1, 2, \ldots$, until convergence,
3.     Solve $E_k X_k + X_k E_k = A - X_k^2$ to obtain $E_k$,
4.     $X_{k+1} = X_k + E_k$.
5. End
```

If we choose an initial guess X_0 such that $AX_0 = X_0 A$, then the Sylvester equation is simplified. By the choice, it can be shown that $E_k X_k = X_k E_k$. Then from $E_k = (1/2)X_k^{-1}(A - X_k^2) = (1/2)(X_k^{-1}A - X_k)$, it follows that $X_k + E_k = (1/2)(X_k + X_k^{-1}A)$. We now describe Newton's method for the matrix square root in Algorithm 5.2.

For the details of the above derivation and a derivation based on Fréchet derivative, see [96].

Algorithm 5.2 (Iterative method) Newton's method for $X = A^{1/2}$

```
1: Choose $X_0$ such that $AX_0 = X_0 A$, e.g., $X_0 = I$.
2: for $k = 0, 1, \ldots$, until convergence do
3:     $X_{k+1} = \frac{1}{2}(X_k + X_k^{-1}A)$
4: end for
```

Let A be a matrix satisfying the condition in Theorem 5.3. If all the eigenvalues of $A^{1/2}X_0$ lie in the right-half plane of the complex plane, then X_k quadratically converges to the matrix square root $A^{1/2}$ [97, p.140].

In Algorithm 5.2, there is a restriction for the choice of an initial guess, i.e., $AX_0 = X_0 A$. The usual choice is $X_0 = A$ or $X_0 = I$. Since matrix polynomial $c_0 I + c_1 A + c_2 A^2 + \cdots + c_n A^n$ and A commute, it is natural to choose $X_0 (= c_0 I + c_1 A + c_2 A^2 + \cdots + c_n A^n)$ such that the norm of the residual $\|X_0^2 - A\|$ is minimized. An approximate minimization using a Gröbner basis and a low degree matrix polynomial is proposed in [132].

A numerically stable variant of Algorithm 5.2 is the incremental Newton (IN) iteration [101] that is described in Algorithm 5.3.

Algorithm 5.3 (Iterative method) The IN method for $X = A^{1/2}$

```
1: $X_0 = A$; $E_0 = \frac{1}{2}(I - A)$
2: for $k = 0, 1, \ldots$, until convergence do
3:     $X_{k+1} = X_k + E_k$
4:     $E_{k+1} = -\frac{1}{2}E_k X_{k+1}^{-1} E_k$
5: end for
```

A feature of Algorithm 5.3 is that matrix A does not appear in the main iteration, i.e., line 3. Though Algorithm 5.3 is more numerically stable than Algorithm 5.2, additional matrix–matrix multiplications are required. The computational cost of Algorithm 5.3 is about 7/4 times higher than that of Algorithm 5.2 per iteration step.

5.3.2 Matrix pth Root

In this section, direct and iterative methods are described for solving matrix *p*th roots.

A matrix X such that $X^p = A$ is called a matrix *p*th root. As described in Sect. 5.3.1, there may be infinitely many matrix square roots. Among them, there is the principal matrix square root that is characterized in Theorem 5.3. Similarly, the notion of the principal matrix *p*th root is given as follows:

Theorem 5.4 (principal matrix *p*th root) *Let A be a complex square matrix whose eigenvalues ($\lambda_k \neq 0$) do not lie in the negative real axis of the complex plane. Then there exists a unique pth root of A such that all the eigenvalues of the pth root lie in sector area* $\{z \in \mathbb{C} : -\pi/p < \arg(z) < \pi/p\}$.

The unique matrix *p*th root is called the *principal matrix pth root* of A and is denoted by $A^{1/p}$. If $p = 2$, then Theorem 5.4 is equivalent to Theorem 5.3.

In what follows, direct methods for $A^{1/p}$ are described. The Schur method is a direct method based on the Schur decomposition. After the Schur decomposition of matrix A ($A = QTQ^{\mathrm{H}}$), it follows that $A^{1/p} = QT^{1/p}Q^{\mathrm{H}}$. For obtaining $T^{1/p}$, let $U := T^{1/p}$. Since U is an upper tridiagonal matrix, U^p is also an upper tridiagonal matrix. Then, all we have to do is to solve $U^p = T$. The resulting algorithm is summarized in Algorithm 5.4.

Algorithm 5.4 (Direct method) Schur method for $X = A^{1/p}$

```
1: Compute Schur decomposition of $A = QTQ^{\mathrm{H}}$.
2: for $j = 1, 2, \dots, n$ do
3:     $u_{jj} = t_{jj}^{1/p}, v_{jj}^{(1)} = 1, v_{jj}^{(k+2)} = u_{jj}^{k+1}$ $(k = 0, 1, \dots, p-2)$
4:     for $i = j-1, j-2, \dots, 1$ do
5:         for $k = 0, 1, \dots, p-2$ do
6:             $w_{k+2} = \sum_{\ell=i+1}^{j-1} u_{i\ell} v_{\ell j}^{(k+2)}$
7:         end for
8:         $u_{ij} = (t_{ij} - \sum_{k=0}^{p-2} v_{ii}^{(p-k-1)} w_{k+2}) / (\sum_{k=0}^{p-1} v_{ii}^{(p-k)} v_{jj}^{(k+1)})$
9:         for $k = 0, 1, \dots, p-2$ do
10:            $v_{ij}^{(k+2)} = \sum_{\ell=0}^{k} v_{ii}^{(k-\ell+1)} u_{ij} v_{jj}^{(\ell+1)} + \sum_{\ell=0}^{k-1} v_{ii}^{(k-\ell)} w_{\ell+2}$
11:        end for
12:    end for
13: end for
14: $X = QUQ^{\mathrm{H}}$
```

As for iterative methods, Algorithms 5.5 and 5.6 are known as extensions of Algorithms 5.2 and 5.3, respectively.

Algorithm 5.5 (Iterative method) Newton's method for $A^{1/p}$

```
1: Set $p$, and choose $X_0$ such that $AX_0 = X_0A$, e.g., $X_0 = A$.
2: for $k = 0, 1, \dots,$ until convergence do
3:     $X_{k+1} = \frac{1}{p}\left[(p-1)X_k + X_k^{1-p}A\right]$
4: end for
```

The computational cost of Algorithm 5.6 is higher than Algorithm 5.5. On the other hand, it is known in [102] that Algorithm 5.6 has numerical stability. For other useful variants of Newton's method, see, e.g., [102, Eqs. (3.6), (3.9)]. A cost-efficient variant of Algorithm 5.6 having numerical stability is found in [187].

Algorithm 5.6 (Iterative method) The incremental Newton's method for $A^{1/p}$

```
1: Set $p$, $X_0 = I$, $E_0 = \frac{1}{p}(A - I)$.
2: for $k = 0, 1, \dots,$ until convergence do
3:     $X_{k+1} = X_k + E_k$, $F_k = X_k X_{k+1}^{-1}$
4:     $E_{k+1} = -\frac{1}{p}E_k\left[X_{k+1}^{-1}I + 2X_{k+1}^{-1}F_k + \dots + (p-1)X_{k+1}^{-1}F_k^{p-2}\right]E_k$
5: end for
```

5.4 Matrix Exponential Function

As seen in Definition 5.4, the *matrix exponential function* is defined by the Jordan canonical form. On the other hand, the following equivalent definition is useful for computing the matrix function:

$$\mathrm{e}^A := I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots . \tag{5.2}$$

The series converges for any square matrix $A \in \mathbb{C}^{n\times n}$. In this section, numerical algorithms for matrix exponential functions are described.

Fundamental properties of matrix functions are summarized in Theorem 5.5. These properties are easily proved from the definition in (5.2)

Theorem 5.5 (Properties of matrix exponential functions) *For $A, B \in \mathbb{C}^{n\times n}$:*

(1) $\mathrm{e}^O = I$, *where O is the zero matrix;*
(2) $\mathrm{e}^A\mathrm{e}^B = \mathrm{e}^{A+B}$ *if $AB = BA$;*
(3) $\mathrm{e}^A\mathrm{e}^{-A} = I, \quad (\mathrm{e}^A)^{-1} = \mathrm{e}^{-A}$;
(4) $\mathrm{e}^{XAX^{-1}} = X\mathrm{e}^AX^{-1}$, *where X is a nonsingular matrix;*
(5) $\mathrm{e}^{A^{\mathrm{H}}} = (\mathrm{e}^A)^{\mathrm{H}}$.

From (5) in Theorem 5.5, matrix exponential function e^A is Hermitian if A is Hermitian. From (3) and (5) in Theorem 5.5, matrix exponential function e^A is unitary, i.e., $(\mathrm{e}^A)^{\mathrm{H}}\mathrm{e}^A = \mathrm{e}^{-A}\mathrm{e}^A = I$ if A is skew-Hermitian ($A^{\mathrm{H}} = -A$).

From 2) in Theorem 5.5 with $A = aC$ and $B = bC$ for scalar values a and b, we have $\mathrm{e}^{aC}e^{bC} = \mathrm{e}^{(a+b)C}$. Here, if $a = 1$ and $b = -1$, then we have $\mathrm{e}^{C}\mathrm{e}^{-C} = \mathrm{e}^{0C} = \mathrm{e}^{O} = I$ that corresponds to 3) in Theorem 5.5.

5.4.1 Numerical Algorithms for Matrix Exponential Functions

One of the simplest ways to approximately compute matrix exponential functions is to truncate the series in (5.2), i.e., $I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots + \frac{1}{m!}A^m$. If all the eigenvalues of matrix A are close to zero, then the truncation error may be small for small n. The above approach is, however, inefficient for a matrix A with no such distribution of the eigenvalues of A.

A better and simple way to compute matrix exponential functions is to consider $\mathrm{e}^{A/s}$ instead of e^{A}. We see that $(\mathrm{e}^{A/s})^s = \mathrm{e}^{A}$ for $s \geq 1$, and the all eigenvalues of A/s are s times smaller than the eigenvalues of A. Then truncating the following series:

$$\mathrm{e}^{A/s} = I + \frac{1}{1!s}A + \frac{1}{2!s^2}A^2 + \frac{1}{3!s^3}A^3 + \cdots \tag{5.3}$$

yields

$$F_{r,s} = I + \frac{1}{1!s}A + \frac{1}{2!s^2}A^2 + \frac{1}{3!s^3}A^3 + \cdots + \frac{1}{r!s^r}A^r, \tag{5.4}$$

which is an approximation to $(F_{r,s})^s \approx \mathrm{e}^{A}$. Then, the following property holds:[1]

Theorem 5.6 ([179]) *Let $A \in \mathbb{C}^{n\times n}$ and $F_{r,s}$ be the truncation given in (5.4). Then*

$$\|\mathrm{e}^{A} - (F_{r,s})^s\| \leq \frac{\|A\|^{r+1}\mathrm{e}^{\|A\|}}{s^r(r+1)!}, \tag{5.5}$$

where $\|\cdot\|$ is any matrix norm that is submultiplicative.[2]

From the right-hand side of Theorem 5.6, The truncation error can be estimated by computing $\|A\|$ or its upper bound. For example, we consider the case $s = 2^m$. Assume that $\|A\| = 10$. Then if we use $r = 8$, $m = 9$, $s = 2^m$, the right-hand side of (5.5) is about 1.29×10^{-14}, which means that it is possible to determine s and r such that the truncation error is less than a given tolerance, if $\|A\|$ is estimated. When $\|A\|$ is a Frobenius norm, then it is easy to compute the norm by the definition of a Frobenius norm. When $\|A\|$ is a matrix 2-norm, the maximum singular value of A is needed, which can also be easily estimated by the Golub–Kahan bidiagonalization process (Algorithm 3.29).

[1] In [179], Theorem 5.6 is stated on Banach algebra, which is a more general result.

[2] See (Nm4) in Section 1.1.2 for the term "submultiplicative".

A more cost-efficient method is as follows: instead of computing (5.4), we use the Padé approximant to $\mathrm{e}^{A/s}$ and compute sth power of the approximant. The Padé approximant $R_{p,q}(A)$ to matrix exponential function e^{A} is given below:

$$R_{p,q}(A) = [D_{p,q}(A)]^{-1} N_{p,q}(A), \tag{5.6}$$

where

$$N_{p,q}(A) = \sum_{j=0}^{p} n_j A^j, \quad n_j = \frac{(p+q-j)!}{(p+q)!} \times \frac{p!}{j!(p-j)!},$$
$$D_{p,q}(A) = \sum_{j=0}^{q} d_j (-A)^j, \quad d_j = \frac{(p+q-j)!}{(p+q)!} \times \frac{q!}{j!(q-j)!}.$$

When approximating the matrix exponential, it is recommended to use the diagonal Padé approximant, i.e., $p = q$. From $\mathrm{e}^{A} = [\mathrm{e}^{A/2^m}]^{2^m} \approx [R_{q,q}(A/2^m)]^{2^m}$, we need to choose two parameters q and m. If $\|A\|/2^m \le 1/2$, then

$$[R_{q,q}(A/2^m)]^{2^m} = \mathrm{e}^{A+E}, \quad \frac{\|E\|}{\|A\|} \le 8 \left[\frac{\|A\|}{2^m} \right]^{2q} \frac{(q!)^2}{(2q)!(2q+1)!}.$$

For the details, see [133, p.12]. The inequality leads to optimum parameters q and m such that $q+m$ is minimized under the condition that the truncation error holds $\|E\|/\|A\| \le \epsilon$ for a given tolerance ϵ. The optimum parameters for a tolerance $\epsilon = 10^{-15}$ with respect to $\|A\|$ are given in Table 5.1. For related studies, see [8, 98].

If Schur decomposition $A = QTQ^{\mathrm{H}}$ is computed, then it follows from (5.2) that $\mathrm{e}^{A} = e^{QTQ^{\mathrm{H}}} = Q\mathrm{e}^{T}Q^{\mathrm{H}}$. Thus all we have to do is to compute e^{T} using (diagonal) Padé approximant.

In particular, when A is Hermitian, T becomes a diagonal matrix. Let $\lambda_1, \lambda_2, \dots, \lambda_n$ be the diagonal elements of T. Then the matrix exponential function can be computed by

$$\mathrm{e}^{A} = Q \begin{bmatrix} e^{\lambda_1} & & \\ & \ddots & \\ & & e^{\lambda_n} \end{bmatrix} Q^{\mathrm{H}}. \tag{5.7}$$

Further, if A is diagonalizable, i.e., $A = XDX^{-1}$ for a nonsingular matrix X and a diagonal matrix D, then we have $\mathrm{e}^{A} = X\mathrm{e}^{D}X^{-1}$, where e^{D} is the same form as in (5.7). On the other hand, if the condition number X is high, this approach can be numerically unstable.

Table 5.1 Optimum parameters q and m in $R_{q,q}(A/2^m)$ satisfying the error tolerance $\epsilon = 10^{-15}$.

$\|A\|$	10^{-3}	10^{-2}	10^{-1}	1	10	10^2	10^3	10^4	10^5	10^6
q	3	3	4	6	6	6	6	6	6	6
m	0	0	0	1	5	8	11	15	18	21

5.4.2 Multiplication of a Matrix Exponential Function and a Vector

We consider the following linear ordinary differential equation:

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = A\boldsymbol{x}(t) + \boldsymbol{f}(t), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{5.8}$$

where $A \in \mathbb{C}^{n\times n}$, $\boldsymbol{x}, \boldsymbol{f} \in \mathbb{C}^n$, $t \in \mathbb{R}$. The solution can be given by using matrix exponential functions as follows:

$$\boldsymbol{x}(t) = \mathrm{e}^{tA}\boldsymbol{x}_0 + \int_0^t \mathrm{e}^{(t-\tau)A}\boldsymbol{f}(\tau)\,\mathrm{d}\tau. \tag{5.9}$$

We see that in (5.9) there are multiplications of matrix exponential functions and vectors. In particular, if $\boldsymbol{f}(t) = \boldsymbol{0}$, then the solution is written as $\boldsymbol{x}(t) = \mathrm{e}^{tA}\boldsymbol{x}_0$. This explicit form of the solution means that we know the solution at an arbitrary time t if the multiplications are obtained.

If matrix exponential functions have already been computed, then all we have to do is to compute the matrix–vector multiplication whose computational cost is of order n^2, where n is a matrix of size $n \times n$. On the other hand, computing the cost of a matrix exponential function may be of order n^3 even if matrix A is sparse. Thus computing matrix exponential functions is inefficient, when it comes to computing the solution of (5.8). Note that the solution only requires the result of the multiplication of a matrix exponential functions and a vector. In such a case, an approximation to the multiplication using Krylov subspace is useful, which is described below.

Since matrix exponential function e^{tA} is defined by the series (5.2), it is natural to use matrix polynomials $p_m(A)$ of degree m to approximate e^{tA}. The multiplication of a matrix polynomial $p_m(A)$ and a vector belongs to Krylov subspace $\mathcal{K}_m(A, \boldsymbol{x}_0)$. Thus the optimal approximation is given by solving the following least-squares problem:

$$\min_{\boldsymbol{x}_m \in \mathcal{K}_m(A,\boldsymbol{x}_0)} \|\mathrm{e}^{tA}\boldsymbol{x}_0 - \boldsymbol{x}_m\|. \tag{5.10}$$

Let V_m be a matrix whose columns are orthonormalized basis vectors of $\mathcal{K}_m(A, \boldsymbol{x}_0)$. Then, any vector $\boldsymbol{x} \in \mathcal{K}_m(A, \boldsymbol{x}_0)$ can be written as $\boldsymbol{x} = V_m\boldsymbol{y}$, where $\boldsymbol{y} \in \mathbb{C}^m$. Thus (5.10) is equivalent to

$$\min_{\boldsymbol{y}\in\mathbb{C}^m} \|\mathrm{e}^{tA}\boldsymbol{x}_0 - V_m\boldsymbol{y}\|.$$

The solution is $\boldsymbol{y} = V_m^{\mathrm{H}}\mathrm{e}^{tA}\boldsymbol{x}_0$, and thus the solution of (5.10) is given by

$$(\boldsymbol{x}(t) \approx)\ \boldsymbol{x}_m = V_m V_m^{\mathrm{H}}\mathrm{e}^{tA}\boldsymbol{x}_0. \tag{5.11}$$

In what follows, we consider using the Arnoldi process in Section 1.9.1 to compute basis vectors of $\mathcal{K}_m(A, \boldsymbol{x}_0)$ and give an approximate solution of (5.11). From the Arnoldi process in Section 1.9.1, it follows from $\boldsymbol{x}_0 = \|\boldsymbol{x}_0\| V_m \boldsymbol{e}_1$ that we obtain $\boldsymbol{x}_m = \|\boldsymbol{x}_0\| V_m (V_m^{\mathrm{H}} \mathrm{e}^{tA} V_m) \boldsymbol{e}_1$, where $\boldsymbol{e}_1 = [1, 0, \dots, 0]^\top$. Furthermore, using the relation $V_m^{\mathrm{H}} A V_m = H_m$ (Hessenberg matrix) in (1.39), we consider an approximation $V_m^{\mathrm{H}} \mathrm{e}^{tA} V_m \approx \mathrm{e}^{t V_m^{\mathrm{H}} A V_m} = \mathrm{e}^{t H_m}$. Then we have

$$\boldsymbol{x}_m = \|\boldsymbol{x}_0\| V_m (V_m^{\mathrm{H}} \mathrm{e}^{tA} V_m) \boldsymbol{e}_1 \approx \|\boldsymbol{x}_0\| V_m \mathrm{e}^{t H_m} \boldsymbol{e}_1. \tag{5.12}$$

$\mathrm{e}^{t H_m} \boldsymbol{e}_1$ is the multiplication of an $m \times m$ small matrix exponential function and a vector, whose computational costs are of $\mathrm{O}(m^2)$. This comes from the fact that the multiplication of the Padé approximant (to $\mathrm{e}^{t H_m}$) and a vector requires $H_m^{-1} \boldsymbol{v}$, i.e., solving linear systems $H_m \boldsymbol{z} = \boldsymbol{v}$, whose computational cost is of $\mathrm{O}(m^2)$.

If matrix A is Hermitian, a suitable choice of m for satisfying a given error tolerance is provided by using Theorem 5.7.

Theorem 5.7 ([99], Theorem 2) *Let A be Hermitian and all the eigenvalues lie in the interval $[-4\rho, 0]$. Then the error of (5.12) is given by*

$$\|\mathrm{e}^{tA} \boldsymbol{x}_0 - \|\boldsymbol{x}_0\|_2 V_m \mathrm{e}^{t H_m} \boldsymbol{e}_1\| \le \begin{cases} c \cdot \mathrm{e}^{\frac{-m^2}{5\rho t}} & (\sqrt{4\rho t} \le m \le 2\rho t), \\ c \cdot \frac{1}{\rho t \mathrm{e}^{\rho t}} \cdot \left(\frac{\mathrm{e}\rho t}{m}\right)^m & (2\rho t \le m), \end{cases}$$

where $c = 10\|\boldsymbol{x}_0\|$.

In practice, first estimate the minimum eigenvalue $\lambda_{\min}$ of A and set $\rho = -\lambda_{\min}/4$. Then, the dimension of Krylov subspace m is chosen so that the upper bound error in Theorem 5.7 is less than the given tolerance, e.g., 10^{-12}. The usefulness of the error bound is shown in [139], together with devising a practical computation avoiding the loss of orthogonality regarding the basis vectors of Krylov subspaces and an efficient method of parallel computation.

5.5 Matrix Trigonometric Functions

Here we consider matrix trigonometric functions: matrix sine function $\sin(A)$ and matrix cosine function $\cos(A)$. $\sin(x)$ and $\cos(x)$ can be expanded as

$$\sin(x) = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \cdots,$$
$$\cos(x) = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \cdots.$$

$\sin(A)$ and $\cos(A)$ are defined for any $A \in \mathbb{C}^{n \times n}$ by using the above expansions as described below:

$$\sin(A) = A - \frac{1}{3!}A^3 + \frac{1}{5!}A^5 - \frac{1}{7!}A^7 + \cdots,$$
$$\cos(A) = I - \frac{1}{2!}A^2 + \frac{1}{4!}A^4 - \frac{1}{6!}A^6 + \cdots.$$

Below are the fundamental properties of $\sin(A)$ and $\cos(A)$.

Theorem 5.8 (Some properties of $\sin(A)$ and $\cos(A)$) *The following facts hold true for $A, B \in \mathbb{C}^{n\times n}$:*

(1) $e^{iA} = \cos(A) + i\sin(A)$;
(2) $\sin(A) = (e^{iA} - e^{-iA})/2i$;
(3) $\cos(A) = (e^{iA} + e^{-iA})/2$;
(4) $\sin(-A) = -\sin(A)$, $\cos(-A) = \cos(A)$;
(5) $\sin(A)^2 + \cos(A)^2 = I$;
(6) $\sin(A \pm B) = \sin(A)\cos(B) \pm \cos(A)\sin(B)$ *if* $AB = BA$;
(7) $\cos(A \pm B) = \cos(A)\cos(B) \mp \sin(A)\sin(B)$ *if* $AB = BA$.

(2) and (3) of Theorem 5.8 can be derived from (1) of Theorem 5.8. From (2) and (3), $\sin(A)$ and $\cos(A)$ can be obtained by matrix exponential functions. If A is a real matrix, it follows from 1) of Theorem 5.8 that $\cos(A)$ corresponds to the real part of e^{iA}, and $\sin(A)$ corresponds to the imaginary part of e^{iA}. From this, Algorithm 5.7 is obtained for computing $\sin(A)$ and $\cos(A)$ as described in Algorithm 5.7.

Algorithm 5.7 Computation of $S = \sin(A)$ and $C = \cos(A)$

```
1: Compute $X = e^{iA}$.
2: if $A$ is a real matrix then
3:     $S$ = Im(X) (imaginary part of $A$), $C$ = Re(X) (real part of $A$)
4: end if
5: if $A$ is a complex matrix then
6:     $S = \frac{1}{2i}(X - X^{-1})$, $C = \frac{1}{2}(X + X^{-1})$
7: end if
```

5.6 Matrix Logarithm

A matrix X satisfying $e^X = A$ for $A \in \mathbb{C}^{n\times n}$ is referred to as a *matrix logarithm*. Recall that the notion of the principal matrix square root is described in Theorem 5.3. Similarly, the notion of the principal matrix logarithm is described in Theorem 5.9.

Theorem 5.9 (Principal matrix logarithm) *Let A be a complex square matrix whose eigenvalues ($\lambda_k \neq 0$) do not lie in the negative real axis of the complex plane. Then there exists a unique matrix logarithm of A such that all the eigenvalues of the square root lie in the strip $\{z \in \mathbb{C} : -\pi < \mathrm{Im}(z) < \pi\}$.*

The unique matrix in Theorem 5.9 is referred to as *the principal matrix logarithm* of A, denoted by $\log(A)$. Below are some properties of $\log(A)$.

Theorem 5.10 (Some properties of $\log(A)$**)** *Let A be a complex square matrix whose eigenvalues ($\lambda_k \neq 0$) do not lie in the negative real axis of the complex plane. Then, for* $-1 \leq \alpha \leq 1$ *we have* $\log(A^\alpha) = \alpha \log(A)$. *In particular,* $\log(A^{-1}) = -\log(A)$, $\log(A^{1/2}) = (1/2)\log(A)$.

For the proof of Theorem 5.10, see the proof of [97, Theorem 11.2].

In what follows, some numerical algorithms for computing $\log(A)$ are described. When matrix A satisfies $\rho(A - I) < 1$[3], $\log(A)$ can be expanded as follows:

$$\log(A) = \log(I + (A - I)) = (A - I) - \frac{1}{2}(A - I)^2 + \frac{1}{3}(A - I)^3 - \frac{1}{4}(A - I)^4 + \cdots , \tag{5.13}$$

which is the Neumann series, see also Section 3.5.3. If A does not satisfy $\rho(A - I) < 1$, we cannot use (5.13) for approximately computing $\log(A)$. Even in this case, the expansion can be useful after the following modifications: it follows from Theorem 5.10 that for a natural number k we have

$$\log(A) = k \log(A^{1/k}),$$

and $A^{1/k}$ gets closer to the identity matrix I as k gets larger. Thus all we have to do is to find k from (5.13) such that the following series is convergent:

$$\begin{aligned} \log(A^{1/k}) &= \log(I + (A^{1/k} - I)) \\ &= (A^{1/k} - I) - \frac{1}{2}(A^{1/k} - I)^2 + \frac{1}{3}(A^{1/k} - I)^3 + \cdots . \end{aligned} \tag{5.14}$$

In practice, set $k = 2^m$, and for the right-hand side of (5.14) we use the Padé approximant to $\log(1 + x)$. Below are some examples of the diagonal Padé approximant to $\log(I + X)$:

$$\begin{aligned} R_{1,1}(X) &= (2I + X)^{-1}(2X), \\ R_{2,2}(X) &= (6I + 6X + X^2)^{-1}(6X + 3X^2), \\ R_{3,3}(X) &= (60I + 90X + 36X^2 + 3X^3)^{-1}(60X + 60X^2 + 11X^3). \end{aligned}$$

The algorithm is listed in Algorithm 5.8. For the matrix 2^mth root, see Sect. 5.3.2. For an improvement of Algorithm 5.8, see [9].

Algorithm 5.8 Computation of $\log(A)$

```
1: Set a natural number $m$ such that $\rho(A^{1/2^m} - I) < 1$.
2: $X = A^{1/2^m} - I$
3: $Y = R_{q,q}(X)$ (the diagonal Padé aproximant)
4: $\log(A) \approx 2^m Y$
```

[3] $\rho(X) = |\lambda_{\max}|$ is the spectral radius of X, see Section 1.6.4.

Numerical algorithms based on Newton's method are listed in Algorithms 5.9 and 5.10. Under a certain condition, Algorithm 5.9 shows a locally quadratic convergence to $\log(A)$, and Algorithm 5.10 shows a locally cubic convergence to $\log(A)$.

Algorithm 5.9 Iterative method 1 for $X = \log(A)$

```
1: Choose $X_0$ such that $AX_0 = X_0A$, e.g., $X_0 = A$.
2: for $k = 0, 1, \dots,$ until convergence do
3:     $X_{k+1} = X_k - I + \mathrm{e}^{-X_k}A$
4: end for
```

Algorithm 5.10 Iterative method 2 for $X = \log(A)$

```
1: Choose $X_0$ such that $AX_0 = X_0A$, e.g., $X_0 = A$.
2: for $k = 0, 1, \dots,$ until convergence do
3:     $X_{k+1} = X_k + \frac{1}{2}\left(\mathrm{e}^{-X_k}A - A^{-1}\mathrm{e}^{X_k}\right)$
4: end for
```

Let $Y = \mathrm{e}^{-X_k}A$ in Algorithm 5.10. Then from 3) of Theorem 5.5 we have $Y^{-1} = A^{-1}\mathrm{e}^{X_k}$, and thus we obtain $X_{k+1} = X_k + (Y - Y^{-1})/2$. This implies that the computational cost of Algorithm 5.10 is about the sum of computational costs of Algorithm 5.9 and Y^{-1}. Since the cost of Y^{-1} is relatively much smaller than that of the matrix exponential function, the cost of Algorithm 5.9 is nearly equal to that of Algorithm 5.10. From this, Algorithm 5.10 will be faster than Algorithm 5.9 since Algorithm 5.10 shows a cubic convergence.

When matrix A is Hermitian positive definite, the Schur decomposition of A corresponds to the eigen-decomposition $A = QDQ^{\mathrm{H}}$, where Q is a unitary matrix and D is a diagonal matrix whose diagonal elements are eigenvalues λ_i of A, and thus $\mathrm{e}^X = A \Leftrightarrow \mathrm{e}^X = QDQ^{\mathrm{H}} \Leftrightarrow Q^{\mathrm{H}}\mathrm{e}^X Q = D \Leftrightarrow \mathrm{e}^{Q^{\mathrm{H}}XQ} = D$. Let $M = Q^{\mathrm{H}}XQ$. Then M satisfying $\mathrm{e}^M = D$ can be written as $M = \mathrm{diag}(\log(\lambda_1), \dots, \log(\lambda_n))$. From the definition of M, we have $X (= \log(A)) = QMQ^{\mathrm{H}}$, leading to the following computation of $\log(A)$:

$$\log(A) = Q \begin{bmatrix} \log(\lambda_1) & & \\ & \ddots & \\ & & \log(\lambda_n) \end{bmatrix} Q^{\mathrm{H}}.$$

In general, if matrix A satisfies the assumption in Theorem 5.9 and diagonalizable (i.e., $A = VDV^{-1}$), then it may be possible to compute $\log(A)$ by

$$\log(A) = V \begin{bmatrix} \log(\lambda_1) & & \\ & \ddots & \\ & & \log(\lambda_n) \end{bmatrix} V^{-1}.$$

However, if V is ill-conditioned, then this approach is not recommended due to numerical instability of the computation of V^{-1}.

If matrix A is large and sparse, and if we need specific elements of $\log(A)$, computing the following integration (e.g., [97, Theorem 11.1]) may be a method of choice:

$$\log(A) = (A - I) \int_0^1 [t(A - I) + I]^{-1} \, \mathrm{d}t. \tag{5.15}$$

Applying the jth unit vector $\boldsymbol{e}_j$ to (5.15) from the right yields

$$\log(A)\boldsymbol{e}_j = (A - I) \int_0^1 [t(A - I) + I]^{-1} \boldsymbol{e}_j \, \mathrm{d}t. \tag{5.16}$$

This means that computing the right-hand side of (5.16) yields the ith column vector of $\log(A)$. Now, let $\boldsymbol{x}^{(t)} = [t(A - I) + I]^{-1} \boldsymbol{e}_j$. Then we have

$$[t(A - I) + I]\boldsymbol{x}^{(t)} = \boldsymbol{e}_j,$$

which are (continuous) shifted linear systems for t. For the numerical quadrature, if t is discretized as $t_1, t_2, \ldots, t_m$, then we need to solve

$$[t_i(A - I) + I]\boldsymbol{x}^{(i)} = \boldsymbol{e}_j \quad \text{for } i = 1, 2, \ldots, m. \tag{5.17}$$

For $t_i \neq 0$, the equations can be rewritten as

$$\left[A + (t_i^{-1} - 1)I\right]\tilde{\boldsymbol{x}}^{(i)} = \boldsymbol{e}_j \quad \text{for } i = 1, 2, \ldots, m, \tag{5.18}$$

where $\tilde{\boldsymbol{x}}^{(i)} = t_i\boldsymbol{x}^{(i)}$. Therefore, computing specific elements of the matrix logarithm via a quadrature formula is an important application for shifted Krylov subspace methods in Chap. 4.

Among many numerical quadratures, the double exponential (DE) formula [183] is regarded as one of the most successful methods, especially if the integrand has endpoint (near) singularities. For the developments of the DE formula, see, e.g., [192].

Using the DE formula for computing matrix functions was first considered in [188]. In what follows, the DE formula for the matrix logarithm is described. Applying variable transformation $u = 2t - 1$ to (5.15) yields

$$\log(A) = (A - I) \int_{-1}^1 [(1 + u)(A - I) + 2I]^{-1} \, \mathrm{d}u. \tag{5.19}$$

Then, using the DE transformation $u = \tanh(\sinh(x))$ gives

$$\log(A) = (A - I) \int_{-\infty}^{\infty} F_{\mathrm{DE}}(x)\, \mathrm{d}x, \tag{5.20}$$

where

$$F_{\mathrm{DE}}(x) := \cosh(x)\mathrm{sech}^2(\sinh(x))\, [(1 + \tanh(\sinh(x)))(A - I) + 2I]^{-1}. \tag{5.21}$$

Note that the matrix $(1 + \tanh(\sinh(x)))(A - I) + 2I$ in F_{DE} is nonsingular for any $x \in (-\infty, \infty)$.

An algorithm for computing $\log(A)$ the DE formula is given in Algorithm 5.11. For the theoretical details of Algorithm 5.11, see [188].

From line 16 of Algorithm 5.11, vector $(A - I)T\boldsymbol{e}_j$ is the jth column of $\log(A)$, and from lines 3 and 15 we need to solve the following equations:

Algorithm 5.11 Computation of $\log(A)$ based on the DE formula

```
1: Input: $A \in \mathbb{R}^{n\times n}$, $m \in \mathbb{N}$, $\epsilon > 0$ a tolerance for the interval truncation error
2: Output: $X \approx \log(A)$
3: Set $F_{\mathrm{DE}}(x) = \cosh(x)\mathrm{sech}^2(\sinh(x))\, [(1 + \tanh(\sinh(x)))(A - I) + 2I]^{-1}$.
4: Compute $\|A - I\|$, $\|A^{-1}\|$, and $\rho(A)$.
5: $\theta = |\log(\rho(A))|$
6: $\epsilon_{\max} = \frac{3}{\theta}\frac{\|A - I\|\|A^{-1}\|}{1 + \|A^{-1}\|}$
7: if $\epsilon \geq \epsilon_{\max}$ then
8:     $\epsilon \leftarrow \epsilon_{\max}/2$
9: end if
10: $a = \min\left\{\frac{\theta\epsilon}{3\|A - I\|}, \frac{1}{2\|A - I\|}\right\}$
11: $b = \max\left\{1 - \frac{\theta\epsilon}{3\|A - I\|\|A^{-1}\|}, \frac{2\|A^{-1}\|}{2\|A^{-1}\| + 1}\right\}$
12: $l = \mathrm{arsinh}(\mathrm{artanh}(2a - 1))$
13: $r = \mathrm{arsinh}(\mathrm{artanh}(2b - 1))$
14: $h = (r - l)/(m - 1)$
15: $T = \frac{h}{2}(F_{\mathrm{DE}}(l) + F_{\mathrm{DE}}(r)) + h\sum_{i=1}^{m-2} F_{\mathrm{DE}}(l + ih)$
16: $X = (A - I)T$
```

$$[(1 + \tanh(\sinh(l + ih)))(A - I) + 2I]\boldsymbol{x} = \boldsymbol{e}_j \quad \text{for } i = 0, 1, \ldots, m - 1,$$

which can also be rewritten as shifted linear systems. For the rewrite, see (5.17) and (5.18). Thus, if matrix A is large and sparse, (shifted) Krylov subspace methods are attractive to use.

5.7 Matrix Fractional Power

Matrix fractional power can be defined by using $\log(A)$ as follows:

$$A^\alpha = \exp(\alpha \log(A)), \tag{5.22}$$

where $0 < \alpha < 1$. The condition $0 < \alpha < 1$ looks to be too restrictive, but it is satisfactory in practice. In fact, if one wants to compute $A^{2.3}$, then $A^{2.3}$ can be decomposed by $A^{2.3} = A^2 A^{0.3}$. Therefore, the problem is how to compute $A^{0.3}$ that corresponds to (5.22) with $\alpha = 0.3$.

Computing A^α via the definition in (5.22) requires two matrix functions: a matrix logarithm and a matrix exponential function. On the other hand, these matrix functions do not appear in the following integral form:

$$A^\alpha = \frac{\sin(\alpha\pi)}{\alpha\pi} A \int_0^\infty (t^{1/\alpha} I + A)^{-1}\,\mathrm{d}t \qquad (0 < \alpha < 1). \tag{5.23}$$

Application of the DE formula to (5.23) is considered in [186] as described next:

$$A^\alpha = \int_{-\infty}^{\infty} F_{\mathrm{DE}}(x)\,\mathrm{d}x, \tag{5.24}$$

where

$$F_{\mathrm{DE}}(x) = t'(x) F(t(x)), \qquad F(t) = \frac{\sin(\alpha\pi)}{\alpha\pi} A (t^{1/\alpha} I + A)^{-1}. \tag{5.25}$$

The algorithm for computing A^α by the DE formula is described in Algorithm 5.12. For the theoretical details of Algorithm 5.12, see [186].

Algorithm 5.12 m-point DE formula for computing A^α

```
1: **Input** $A \in \mathbb{R}^{n\times n}$, $\alpha \in (0,1)$, $\epsilon > 0$, $m$
2: $l, r = \texttt{GetInterval}(A, \alpha, \epsilon)$
3: Set $\tilde{F}_{\mathrm{DE}}(x) := \exp(\alpha\pi \sinh(x)/2)\cosh(x)\left[\exp(\pi\sinh(x)/2)I + A\right]^{-1}$.
4: $h = (r - l)/(m - 1)$
5: $T = h[\tilde{F}_{\mathrm{DE}}(l) + \tilde{F}_{\mathrm{DE}}(r)]/2 + h\sum_{k=1}^{m-2}\tilde{F}_{\mathrm{DE}}(l + kh)$
6: **Output** $\sin(\alpha\pi)AT/2 \approx A^\alpha$
7:
8: **function** $\texttt{GetInterval}(A, \alpha, \epsilon)$
9:     Compute $\|A\|$, $\|A^{-1}\|$.
10:    $a_1 = [\alpha\pi(1+\alpha)\epsilon]/[4\sin(\alpha\pi)(1+2\alpha)]$,   $a_2 = (2\|A^{-1}\|)^{-\alpha}$
11:    $a = \min\{a_1, a_2\}$
12:    $b_1 = [\pi(1-\alpha)(2-\alpha)\epsilon]^{\alpha/(\alpha-1)}/[4\sin(\alpha\pi)(3-2\alpha)\|A\|]^{\alpha/(\alpha-1)}$,   $b_2 = (2\|A\|)^\alpha$
13:    $b = \max\{b_1, b_2\}$
14:    $l = \mathrm{asinh}(2\log(a)/\alpha\pi)$,   $r = \mathrm{asinh}(2\log(b)/\alpha\pi)$
15:    **return** $l, r$
16: **end function**
```

From Algorithm 5.12, the jth column of computed A^α is $(\sin(\alpha\pi)AT/2)\boldsymbol{e}_j$. Therefore, from line 5 the $\tilde{F}_{\mathrm{DE}}(l+ih)\boldsymbol{e}_j$'s must be computed using the following equations:

$$\mathrm{e}^{\alpha\pi\,\sinh(l+ih)/2}\cosh(l+ih)\left(\mathrm{e}^{\pi\,\sinh(l+ih)/2}I+A\right)\boldsymbol{x}^{(i)}=\boldsymbol{e}_j \quad \text{for } i=0,1,\ldots,m-1.$$

Similar to (5.17) and (5.18), the equations can be rewritten as shifted linear systems. Thus, if matrix A is large and sparse, using (shifted) Krylov subspace methods will be a method of choice.

Software

The emphasis of this book is on algorithm design, and the detailed history of the Krylov subspace methods is omitted. For those who would like to know the history in detail, the book by Gérard Meurant and Jurjen Duintjer Tebbens [129] is highly recommended, and provides a detailed history with more than a thousand references and Matlab/Octave functions of Krylov subspace methods that come in handy for many users.

For the convenience of possible users of Krylov subspace methods, some available software packages are listed below.

- Fortran 90 (CCGPACK 2.0 by Piotr J. Flatau)
  https://code.google.com/archive/p/conjugate-gradient-lib/
  $\rightarrow$ downloads $\rightarrow$ ccgpak2_0.zip
  User manual https://arxiv.org/abs/1208.4869
  COCR, CSYM, BiCGSTAB(ℓ), GPBiCG(m, ℓ), BiCOR and others are available.

- Fortran 90 (Kω)
  Kω: an open-source library for the shifted Krylov subspace methods
  https://www.pasums.issp.u-tokyo.ac.jp/komega/en/
  Shifted CG, Shifted COCG, Shifted BiCG are available.

- GNU Octave (version 6.4.0)
  https://octave.org/doc/v6.4.0/Specialized-Solvers.html#Specialized-Solvers
  CG, CR, BiCG, QMR, CGS, BiCGSTAB, GMRES, and others are available.

- Julia (IterativeSolvers.jl)
  https://iterativesolvers.julialinearalgebra.org/dev/
  CG, MINRES, GMRES, IDR(s), BiCGSTAB(ℓ) are available.

- Python (SciPy 1.8.0)
  https://docs.scipy.org/doc/scipy/reference/sparse.linalg.html
  CG, MINRES, BiCG, QMR, CGS, BiCGSTAB, GMRES, and others are available.

T. Sogabe, *Krylov Subspace Methods for Linear Systems*, Springer Series in Computational Mathematics 60, https://doi.org/10.1007/978-981-19-8532-4

References

1. Abe, K., Fujino, S.: Converting BiCR method for linear equations with complex symmetric matrices. App. Math. Comput. **321**, 564–576 (2018)
2. Abe, K., Sleijpen, G.L.G.: BiCR variants of the hybrid BiCG methods for solving linear systems with nonsymmetric matrices. J. Comput. Appl. Math. **234**, 985–994 (2010)
3. Absil, P.-A., Mahony, R., Sepulchre, R.: Optimization Algorithms on Matrix Manifolds. Princeton University Press, Princeton (2008)
4. Aihara, K.: GPBi-CGstab(L): A Lanczos-type product method unifying Bi-CGstab(L) and GPBi-CG. Numer. Linear Algebra Appl. **27**, e2298 (2020)
5. Aihara, K., Abe, K., Ishiwata, E.: An alternative implementation of the IDRstab method saving vector updates. JSIAM Lett. **3**, 69–72 (2011)
6. Aihara, K., Abe, K., Ishiwata, E.: A quasi-minimal residual variant of IDRstab using the residual smoothing technique. Appl. Math. Comput. **236**, 67–77 (2014)
7. Aihara, K., Abe, K., Ishiwata, E.: A variant of IDRstab with reliable update strategies for solving sparse linear systems. J. Comput. Appl. Math. **259**, 244–258 (2014)
8. Al-Mohy, A.H., Higham, N.J.: A new scaling and squaring algorithm for the matrix exponential. SIAM J. Matrix Anal. Appl. **31**, 970–989 (2009)
9. Al-Mohy, A.H., Higham, N.J.: Improved inverse scaling and squaring algorithms for the matrix logarithm. SIAM J. Sci. Comput. **34**, C153–C169 (2012)
10. Axelsson, O.: Iterative Solution Methods. Cambridge University Press, Cambridge (1994)
11. Baglama, J., Reichel, L.: Augmented GMRES-type methods. Numer. Linear Algebra Appl. **14**, 337–350 (2007)
12. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H.A. (eds.): Templates for the Solution of Algebraic Eigenvalue Problems. SIAM, Philadelphia (2000)
13. Baker, A.H., Dennis, J.M., Jessup, E.R.: On improving linear solver performance: a block variant of GMRES. SIAM J. Sci. Comput. **27**, 1608–1626 (2006)
14. Baker, A.H., Jessup, E.R., Kolev, T.V.: A simple strategy for varying the restart parameter in GMRES(m). J. Comput. Appl. Math. **230**, 751–761 (2009)
15. Baker, A.H., Jessup, E.R., Manteuffel, T.: A technique for accelerating the convergence of restarted GMRES. SIAM J. Matrix Anal. Appl. **26**, 962–984 (2005)
16. Bank, R.E., Chan, T.F.: An analysis of the composite step biconjugate gradient method. Numer. Math. **66**, 295–319 (1993)

17. Bank, R.E., Chan, T.F.: A composite step bi-conjugate gradient algorithm for nonsymmetric linear systems. Numer. Algorithms **7**, 1–16 (1994)
18. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, Philadelphia (1994)
19. Bayliss, A., Goldstein, C., Turkel, E.: An iterative method for the Helmholtz equation. J. Comput. Phys. **49**, 443–457 (1983)
20. Benzi, M.: Preconditioning techniques for large linear systems: a survey. J. Comput. Phys. **182**, 418–477 (2002)
21. Benzi, M., Cullum, J.K., Tůma, M.: Robust approximate inverse preconditioning for the conjugate gradient method. SIAM J. Sci. Comput. **22**, 1318–1332 (2000)
22. Benzi, M., Meyer, C.D., Tůma, M.: A sparse approximate inverse preconditioner for the conjugate gradient method. SIAM J. Sci. Comput. **17**, 1135–1149 (1996)
23. Benzi, M., Szyld, D.B., van Duin, A.: Orderings for incomplete factorization preconditioning of nonsymmetric problems. SIAM J. Sci. Comput. **20**, 1652–1670 (1999)
24. Benzi, M., Tůma, M.: A sparse approximate inverse preconditioner for nonsymmetric linear systems. SIAM J. Sci. Comput. **19**, 968–994 (1998)
25. Benzi, M., Tůma, M.: A robust preconditioner with low memory requirements for large sparse least squares problems. SIAM J. Sci. Comput. **25**, 499–512 (2003)
26. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
27. Bramble, J.H.: Multigrid Methods. Longman Scientific and Technical, Harlow (1993)
28. Briggs, W.L., Henson, V.E., McCormick, S.F.: A Multigrid Tutorial, 2nd edn. SIAM, Philadelphia (2000)
29. Bruaset, A.M.: A Survey of Preconditioned Iterative Methods. Routledge, Boca Raton (1995)
30. Bunse-Gerstner, A., Stöver, R.: On a conjugate gradient-type method for solving complex symmetric linear systems. Linear Algebra Appl. **287**, 105–123 (1999)
31. Cabral, J.C., Schaerer, C.E., Bhaya, A.: Improving GMRES(m) using an adaptive switching controller. Numer. Linear Algebra Appl. **27**, e2305 (2020)
32. Carpentieri, B., Jing, Y.-F., Huang, T.-Z.: The BiCOR and CORS iterative algorithms for solving nonsymmetric linear systems. SIAM J. Sci. Comput. **33**, 3020–3036 (2011)
33. Cayley, A.: A memoir on the theory of matrices. Philos. Trans. R. Soc. **148**, 17–37 (1858)
34. Chan, T.F., Gallopoulos, E., Simoncini, V., Szeto, T., Tong, C.H.: A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems. SIAM J. Sci. Comput. **15**, 338–347 (1994)
35. Chan, T.F., Szeto, T.: Composite step product methods for solving nonsymmetric linear systems. SIAM J. Sci. Comput. **17**, 1491–1508 (1996)
36. Chiba, K.: Gyouretsu no Kansu to Jordan Hyoujun Kei (in Japanese). Scientist Press, Tokyo (2010)
37. Choi, S.-C.: Minimal residual methods for complex symmetric, skew symmetric, and skew Hermitian systems. Tech. rep., Comput. Inst., Univ. Chicago, Chicago, IL, USA (2013)
38. Chronopoulos, A.T., Ma, S.: On squaring Krylov subspace iterative methods for nonsymmetric linear systems. Tech. Rep. TR 89-67 (1989)
39. Cipra, B.A.: The best of the 20th century: Editors name top 10 algorithms. SIAM News **33**, 1–2 (2000)
40. Clemens, M., Weiland, T., Rienen, U.V.: Comparison of Krylov-type methods for complex linear systems applied to high-voltage problems. IEEE Trans. Magn. **34**, 3335–3338 (1998)
41. Craig, E.J.: The N-step iteration procedures. J. Math. Phys. **34**, 64–73 (1955)
42. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th National Conference, pp. 157–172. Association for Computing Machinery, New York, NY, USA (1969)
43. Datta, B.N., Saad, Y.: Arnoldi methods for large Sylvester-like observer matrix equations, and an associated algorithm for partial spectrum assignment. Linear Algebra Appl. **154–156**, 225–244 (1991)

44. Davis, T.A.: Direct Methods for Sparse Linear Systems. SIAM, Philadelphia (2006)
45. Demmel, J.W.: Applied Numerical Linear Algebra. SIAM, Philadelphia (1997)
46. Du, L., Sogabe, T., Yu, B., Yamamoto, Y., Zhang, S.-L.: A block IDR(s) method for nonsymmetric linear systems with multiple right-hand sides. J. Comput. Appl. Math. **235**, 4095–4106 (2011)
47. Du, L., Sogabe, T., Zhang, S.-L.: A variant of the IDR(s) method with the quasi-minimal residual strategy. J. Comput. Appl. Math. **236**, 621–630 (2011)
48. Du, L., Sogabe, T., Zhang, S.-L.: IDR(s) for solving shifted nonsymmetric linear systems. J. Comput. Appl. Math. **274**, 35–43 (2015)
49. Du, Y.-S., Hayami, K., Zheng, N., Morikuni, K., Yin, J.-F.: Kaczmarz-type inner-iteration preconditioned flexible GMRES methods for consistent linear systems. SIAM J. Sci. Comput. **43**, S345–S366 (2021)
50. Dubois, P.F., Greenbaum, A., Rodrigue, G.H.: Approximating the inverse of a matrix for use in iterative algorithms on vector processors. Computing **22**, 257–268 (1979)
51. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct Methods for Sparse Matrices. Oxford University Press, Oxford (1986)
52. Duff, I.S., Meurant, G.A.: The effect of ordering on preconditioned conjugate gradients. BIT Numer. Math. **29**, 635–657 (1989)
53. Eisenstat, S.C.: Efficient implementation of a class of preconditioned conjugate gradient methods. SIAM J. Sci. Stat. Comput. **2**, 1–4 (1981)
54. Eisenstat, S.C., Elman, H.C., Schultz, M.H.: Variational iterative methods for nonsymmetric systems of linear equations. SIAM J. Numer. Anal. **20**, 345–357 (1983)
55. Elbouyahyaoui, L., Heyouni, M., Tajaddini, A., Saberi-Movahed, F.: On restarted and deflated block FOM and GMRES methods for sequences of shifted linear systems. Numer. Algorithms **87**, 1257–1299 (2021)
56. Eldén, L., Savas, B.: A Newton-Grassmann method for computing the best multilinear rank-(r_1, r_2, r_3) approximation of a tensor. SIAM J. Matrix Anal. Appl. **31**, 248–271 (2009)
57. Faber, V., Manteuffel, T.: Necessary and sufficient conditions for the existence of a conjugate gradient method. SIAM J. Numer. Anal. **21**, 352–362 (1984)
58. Fletcher, R.: Conjugate gradient methods for indefinite systems. In: Watson, G.A. (ed.) Numerical Analysis, pp. 73–89. Springer, Berlin, Heidelberg (1976)
59. Forsythe, G.E., Straus, E.G.: On best conditioned matrices. Proc. Amer. Math. Soc. **6**, 340–345 (1955)
60. Freund, R.W.: On conjugate gradient type methods and polynomial preconditioners for a class of complex non-Hermitian matrices. Numer. Math. **57**, 285–312 (1990)
61. Freund, R.W.: Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices. SIAM J. Sci. Stat. Comput. **13**, 425–448 (1992)
62. Freund, R.W.: Solution of shifted linear systems by quasi-minimal residual iterations. In: Reichel, L., Ruttan, A., Varga, R.S. (eds.) Numerical Linear Algebra and Scientific Computation, pp. 101–122. De Gruyter (1993)
63. Freund, R.W.: A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. SIAM J. Sci. Comput. **14**, 470–482 (1993)
64. Freund, R.W., Golub, G.H., Nachtigal, N.M.: Iterative solution of linear systems. Acta Numer. **1**, 57–100 (1992)
65. Freund, R.W., Malhotra, M.: A block QMR algorithm for non-Hermitian linear systems with multiple right-hand sides. Linear Algebra Appl. **254**, 119–157 (1997)
66. Freund, R.W., Nachtigal, N.M.: QMR: a quasi-minimal residual method for non-Hermitian linear systems. Numer. Math. **60**, 315–339 (1991)
67. Frommer, A.: BiCGStab(ℓ) for families of shifted linear systems. Computing **70**, 87–109 (2003)
68. Frommer, A., Glässner, U.: Restarted GMRES for shifted linear systems. SIAM J. Sci. Comput. **19**, 15–26 (1998)

69. Frommer, A., Lippert, T., Medeke, B., Schilling, K. (eds.): Numerical challenges in lattice quantum chromodynamics: joint interdisciplinary workshop of John von Neumann institute for computing, Jülich, and Institute of Applied Computer Science, Wuppertal University, August 1999. Springer-Verlag, Berlin Heidelberg (2000)
70. Frommer, A., Maass, P.: Fast CG-based methods for Tikhonov-Phillips regularization. SIAM J. Sci. Comput. **20**, 1831–1850 (1999)
71. Fujino, S.: GPBiCG(m, l): A hybrid of BiCGSTAB and GPBiCG methods with efficiency and robustness. Appl. Numer. Math. **41**, 107–117 (2002)
72. Fujino, S., Zhang, S.-L.: Hanpukuhou no Suri (in Japanese). Asakura Shoten, Tokyo (1996)
73. Fujiwara, T., Hoshi, T., Yamamoto, S., Sogabe, T., Zhang, S.-L.: Novel algorithm of large-scale simultaneous linear equations. J. Phys.: Condens. Matter **22**, 074206 (2010)
74. Gaul, A., Gutknecht, M.H., Liesen, J., Nabben, R.: A framework for deflated and augmented Krylov subspace methods. SIAM J. Matrix Anal. Appl. **34**, 495–518 (2013)
75. George, A.: Computer implementation of the finite element method. Tech. Rep. STAN-CS-208, Stanford University, Stanford, CA, USA (1971)
76. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**, 345–363 (1973)
77. George, A., Liu, J.W.: The evolution of the minimum degree ordering algorithm. SIAM Rev. **31**, 1–19 (1989)
78. Golub, G., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. J. Soc. Ind. Appl. Math. Ser. B Numer. Anal. **2**, 205–224 (1965)
79. Golub, G.H., Loan, C.F.V.: Matrix Computations, 4th edn. Johns Hopkins University Press, Baltimore (2013)
80. Golub, G.H., O'Leary, D.P.: Some history of the conjugate gradient and Lanczos algorithms: 1948–1976. SIAM Rev. **31**, 50–102 (1989)
81. Greenbaum, A.: Iterative Methods for Solving Linear Systems. SIAM, Philadelphia (1997)
82. Grote, M.J., Huckle, T.: Parallel preconditioning with sparse approximate inverses. SIAM J. Sci. Comput. **18**, 838–853 (1997)
83. Gu, X.-M., Clemens, M., Huang, T.-Z., Li, L.: The SCBiCG class of algorithms for complex symmetric linear systems with applications in several electromagnetic model problems. Comput. Phys. Commun. **191**, 52–64 (2015)
84. Gu, X.-M., Huang, T.-Z., Li, L., Li, H., Sogabe, T., Clemens, M.: Quasi-minimal residual variants of the COCG and COCR methods for complex symmetric linear systems in electromagnetic simulations. IEEE Trans. Microw. Theory Tech. **62**, 2859–2867 (2014)
85. Gu, X.-M., Huang, T.-Z., Meng, J., Sogabe, T., Li, H.B., Li, L.: BiCR-type methods for families of shifted linear systems. Comput. Math. Appl. **68**, 746–758 (2014)
86. Guennouni, A.E., Jbilou, K., Sadok, H.: A block version of BiCGSTAB for linear systems with multiple right-hand sides. Electron. Trans. Numer. Anal. **16**, 129–142 (2003)
87. Gutknecht, M.H.: Variants of BiCGSTAB for matrices with complex spectrum. SIAM J. Sci. Comput. **14**, 1020–1033 (1993)
88. Gutknecht, M.H.: IDR explained. Electron. Trans. Numer. Anal. **36**, 126–148 (2009)
89. Gutknecht, M.H.: Spectral deflation in Krylov solvers: a theory of coordinate space based methods. Electron. Trans. Numer. Anal. **39**, 156–185 (2012)
90. Gutknecht, M.H.: Deflated and augmented Krylov subspace methods: a framework for deflated BiCG and related solvers. SIAM J. Matrix Anal. Appl. **35**, 1444–1466 (2014)
91. Hackbusch, W.: Multi-Grid Methods and Applications. Springer-Verlag, Berlin Heidelberg (1985)
92. Hackbusch, W.: Iterative Solution of Large Sparse Systems of Equations, Applied Mathematical Sciences, vol. 95. Springer-Verlag, New York (1994)
93. Hadjidimos, A.: Successive overrelaxation (SOR) and related methods. J. Comput. Appl. Math. **123**, 177–199 (2000)
94. Hayami, K., Yin, J.-F., Ito, T.: GMRES methods for least squares problems. SIAM J. Matrix Anal. Appl. **31**, 2400–2430 (2010)

95. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Natl. Bur. Stand. **49**, 409–436 (1952)
96. Higham, N.J.: Newton's method for the matrix square root. Math. Comp. **46**, 537–549 (1986)
97. Higham, N.J.: Functions of Matrices. SIAM, Philadelphia (2008)
98. Higham, N.J.: The scaling and squaring method for the matrix exponential revisited. SIAM Rev. **51**, 747–764 (2009)
99. Hochbruck, M., Lubich, C.: On Krylov subspace approximations to the matrix exponential operator. SIAM J. Numer. Anal. **34**, 1911–1925 (1997)
100. Hoshi, T., Kawamura, M., Yoshimi, K., Motoyama, Y., Misawa, T., Yamaji, Y., Todo, S., Kawashima, N., Sogabe, T.: Kω - Open-source library for the shifted Krylov subspace method of the form $(zI - H)\,\boldsymbol{x}{=}\boldsymbol{b}$. Comput. Phys. Commun. **258**, 107536 (2021)
101. Iannazzo, B.: A note on computing the matrix square root. Calcolo **40**, 273–283 (2003)
102. Iannazzo, B.: On the Newton method for the matrix pth root. SIAM J. Matrix Anal. Appl. **28**, 503–523 (2006)
103. Imakura, A., Li, R.-C., Zhang, S.-L.: Locally optimal and heavy ball GMRES methods. Japan J. Indust. Appl. Math. **33**, 471–499 (2016)
104. Imakura, A., Sogabe, T., Zhang, S.-L.: An efficient variant of the GMRES(m) method based on the error equations. E. Asian J. App. Math. **2**, 19–32 (2012)
105. Imakura, A., Sogabe, T., Zhang, S.-L.: An efficient variant of the restarted shifted GMRES method for solving shifted linear systems. J. Math. Res. Appl. **33**, 127–141 (2013)
106. Imakura, A., Sogabe, T., Zhang, S.-L.: A look-back-type restart for the restarted Krylov subspace methods for solving non-Hermitian linear systems. Japan J. Indust. Appl. Math. **35**, 835–859 (2018)
107. Ipsen, I.C.F.: Numerical Matrix Analysis: Linear Systems and Least Squares. SIAM, Philadelphia (2009)
108. Ishikawa, K.-I., Sogabe, T.: A thick-restart Lanczos type method for Hermitian J-symmetric eigenvalue problems. Japan J. Indust. Appl. Math. **38**, 233–256 (2021)
109. Itoh, S.: Improvement of preconditioned bi-Lanczos-type algorithms with residual norm minimization for the stable solution of systems of linear equations. Japan J. Indust. Appl. Math. **39**, 17–74 (2022)
110. Iwase, S., Hoshi, T., Ono, T.: Numerical solver for first-principles transport calculation based on real-space finite-difference method. Phys. Rev. E **91**, 063305 (2015)
111. Jin, S., Bulgac, A., Roche, K., Wlazłowski, G.: Coordinate-space solver for superfluid many-fermion systems with the shifted conjugate-orthogonal conjugate-gradient method. Phys. Rev. C **95**, 044302 (2017)
112. Jing, Y., Huang, T.: Restarted weighted full orthogonalization method for shifted linear systems. Comput. Math. Appl. **57**, 1583–1591 (2009)
113. Jing, Y., Huang, T., Carpentieri, B., Duan, Y.: Exploiting the composite step strategy to the biconjugate A-orthogonal residual method for non-Hermitian linear systems. J. Appl. Math. **2013**, 16pp. (2013)
114. Jing, Y., Huang, T.-Z., Zhang, Y., Li, L., Cheng, G.-H., Ren, Z.-G., Duan, Y., Sogabe, T., Carpentieri, B.: Lanczos-type variants of the COCR method for complex nonsymmetric linear systems. J. Comput. Phys. **228**, 6376–6394 (2009)
115. Jing, Y., Yuan, P., Huang, T.: A simpler GMRES and its adaptive variant for shifted linear systems. Numer. Linear Algebra Appl. **24**, e2076 (2017)
116. Johnson, O.G., Micchelli, C.A., Paul, G.: Polynomial preconditioners for conjugate gradient calculations. SIAM J. Numer. Anal. **20**, 362–376 (1983)
117. Joubert, W.: On the convergence behavior of the restarted GMRES algorithm for solving nonsymmetric linear systems. Numer. Linear Algebra Appl. **1**, 427–447 (1994)
118. Kikuchi, F.: Yugen Yosoho Gaisetsu (in Japanese). Saiensu-Sha, Tokyo (1980)
119. Lanczos, C.: Solution of systems of linear equations by minimized iterations. J. Res. Natl. Bur. Stand. **49**, 33 (1952)
120. Lee, D., Hoshi, T., Sogabe, T., Miyatake, Y., Zhang, S.-L.: Solution of the k-th eigenvalue problem in large-scale electronic structure calculations. J. Comput. Phys. **371**, 618–632 (2018)

121. Liesen, J., Strakoš, Z.: On optimal short recurrences for generating orthogonal Krylov subspace bases. SIAM Rev. **50**, 485–503 (2008)
122. Liesen, J., Strakoš, Z.: Krylov Subspace Methods: Principles and Analysis. Oxford University Press, Oxford (2012)
123. Liu, J.W.H.: Modification of the minimum-degree algorithm by multiple elimination. ACM Trans. Math. Softw. **11**, 141–153 (1985)
124. Liu, L., Sekiya, K., Ogino, M., Masui, K.: A COMINRES-QLP method for solving complex symmetric linear systems. Electr. Eng. Jpn. **214**, e23325 (2021)
125. Lottes, J.: Towards Robust Algebraic Multigrid Methods for Nonsymmetric Problems. Springer, Cham (2017)
126. Luenberger, D.: Hyperbolic pairs in the method of conjugate gradients. SIAM J. Appl. Math. **6**, 1263–1267 (1969)
127. Meijerink, J.A., van der Vorst, H.A.: An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. Math. Comp. **31**, 148–162 (1977)
128. Meng, G.-Y.: A practical asymptotical optimal SOR method. Appl. Math. Comput. **242**, 707–715 (2014)
129. Meurant, G., Tebbens, J.D.: Krylov Methods for Nonsymmetric Linear Systems: From Theory to Computations, Springer Series in Computational Mathematics, vol. 57. Springer Nature, Switzerland (2020)
130. Miyatake, Y., Sogabe, T., Zhang, S.-L.: On the equivalence between SOR-type methods for linear systems and the discrete gradient methods for gradient systems. J. Comput. Appl. Math. **342**, 58–69 (2018)
131. Miyatake, Y., Sogabe, T., Zhang, S.-L.: Adaptive SOR methods based on the Wolfe conditions. Numer. Algorithms **84**, 117–132 (2020)
132. Mizuno, S., Moriizumi, Y., Usuda, T.S., Sogabe, T.: An initial guess of Newton's method for the matrix square root based on a sphere constrained optimization problem. JSIAM Lett. **8**, 17–20 (2016)
133. Moler, C., Van Loan, C.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM Rev. **45**, 3–49 (2003)
134. Morgan, R.B.: A restarted GMRES method augmented with eigenvectors. SIAM J. Matrix Anal. Appl. **16**, 1154–1171 (1995)
135. Morgan, R.B.: GMRES with deflated restarting. SIAM J. Sci. Comput. **24**, 20–37 (2002)
136. Morikuni, K., Hayami, K.: Inner-iteration Krylov subspace methods for least squares problems. SIAM J. Matrix Anal. Appl. **34**, 1–22 (2013)
137. Moriya, K., Nodera, T.: The DEFLATED-GMRES(m, k) method with switching the restart frequency dynamically. Numer. Linear Algebra Appl. **7**, 569–584 (2000)
138. Nachtigal, N.M., Reddy, S.C., Trefethen, L.N.: How fast are nonsymmetric matrix iterations? SIAM J. Matrix Anal. Appl. **13**, 778–795 (1992)
139. Noritake, S., Imakura, A., Yamamoto, Y., Zhang, S.-L.: A large-grained parallel solver for linear simultaneous ordinary differential equations based on matrix exponential and its evaluation (in Japanese). Trans. JSIAM **19**, 293–312 (2009)
140. Ogasawara, M., Tadano, H., Sakurai, T., Itoh, S.: A Krylov subspace method for shifted linear systems and its application to eigenvalue problems (in Japanese). Trans. JSIAM **14**, 193–205 (2004)
141. Ogino, M., Takei, A., Sugimoto, S., Yoshimura, S.: A numerical study of iterative substructuring method for finite element analysis of high frequency electromagnetic fields. Comput. Math. Appl. **72**, 2020–2027 (2016)
142. O'Leary, D.P.: The block conjugate gradient algorithm and related methods. Linear Algebra Appl. **29**, 293–322 (1980)
143. Paige, C.C., Saunders, M.A.: Solution of sparse indefinite systems of linear equations. SIAM J. Numer. Anal. **12**, 617–629 (1975)
144. Paige, C.C., Saunders, M.A.: LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**, 43–71 (1982)

145. Parlett, B.N., Taylor, D.R., Liu, Z.A.: A look-ahead Lanczos algorithm for unsymmetric matrices. Math. Comp. **44**, 105–124 (1985)
146. Pearson, J.W., Pestana, J.: Preconditioners for Krylov subspace methods: An overview. GAMM-Mitteilungen **43**, e202000015 (2020)
147. Pommerell, C.: Solution of Large Unsymmetric Systems of Linear Equations. Ph.D. thesis, ETH Zürich, Switzerland (1992)
148. Quinn, J.A., Sugiyama, M.: A least-squares approach to anomaly detection in static and sequential data. Pattern Recognit. Lett. **40**, 36–40 (2014)
149. Saad, Y.: Krylov subspace methods for solving large unsymmetric linear systems. Math. Comp. **37**, 105–126 (1981)
150. Saad, Y.: ILUT: A dual threshold incomplete LU factorization. Numer. Linear Algebra Appl. **1**, 387–402 (1994)
151. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
152. Saad, Y., Schultz, M.H.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **7**, 856–869 (1986)
153. Saito, S., Tadano, H., Imakura, A.: Development of the block BiCGSTAB(ℓ) method for solving linear systems with multiple right hand sides. JSIAM Lett. **6**, 65–68 (2014)
154. Sakurai, T., Matsuo, T., Katagiri, T. (eds.): Numerical Linear Algebra: Theory and HPC (in Japanese). Kyoritsu Shuppan, Tokyo (2018)
155. Sato, H., Aihara, K.: Riemannian Newton's method on the Grassmann manifold exploiting the quotient structure (in Japanese). Trans. JSIAM **28**, 205–241 (2018)
156. Sattler, K.D. (ed.): 21st Century Nanoscience—A Handbook: Nanophysics Source Book (1st ed.). CRS Press, Boca Raton (2020)
157. Simoncini, V.: A stabilized QMR version of block BiCG. SIAM J. Matrix Anal. Appl. **18**, 419–434 (1997)
158. Simoncini, V.: Restarted full orthogonalization method for shifted linear systems. BIT Numer. Math. **43**, 459–466 (2003)
159. Simoncini, V., Perotti, F.: On the numerical solution of $(\lambda^2 A + \lambda B + C)\boldsymbol{x} = \boldsymbol{b}$ and application to structural dynamics. SIAM J. Sci. Comput. **23**, 1875–1897 (2002)
160. Simoncini, V., Szyld, D.B.: Recent computational developments in Krylov subspace methods for linear systems. Numer. Linear Algebra Appl. **14**, 1–59 (2007)
161. Simoncini, V., Szyld, D.B.: Interpreting IDR as a Petrov-Galerkin method. SIAM J. Sci. Comput. **32**, 1898–1912 (2010)
162. Sleijpen, G.L.G., Fokkema, D.R.: BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. Electron. Trans. Numer. Anal. **1**, 11–32 (1993)
163. Sleijpen, G.L.G., Sonneveld, P., van Gijzen, M.B.: Bi-CGSTAB as an induced dimension reduction method. Appl. Numer. Math. **60**, 1100–1114 (2010)
164. Sleijpen, G.L.G., van Gijzen, M.B.: Exploiting BiCGstab(ℓ) strategies to induce dimension reduction. SIAM J. Sci. Comput. **32**, 2687–2709 (2010)
165. Sogabe, T.: Extensions of the Conjugate Residual Method. Ph.D. thesis, The University of Tokyo, Tokyo, Japan (2006)
166. Sogabe, T., Hoshi, T., Zhang, S.-L., Fujiwara, T.: A numerical method for calculating the Green's function arising from electronic structure theory. In: Kaneda, Y., Kawamura, H., Sasai, M. (eds.) Frontiers of Computational Science, pp. 189–195. Springer, Berlin, Heidelberg (2007)
167. Sogabe, T., Hoshi, T., Zhang, S.-L., Fujiwara, T.: On a weighted quasi-residual minimization strategy for solving complex symmetric shifted linear systems. Electron. Trans. Numer. Anal. **31**, 126–140 (2008)
168. Sogabe, T., Hoshi, T., Zhang, S.-L., Fujiwara, T.: Solution of generalized shifted linear systems with complex symmetric matrices. J. Comput. Phys. **231**, 5669–5684 (2012)
169. Sogabe, T., Sugihara, M., Zhang, S.-L.: An extension of the conjugate residual method to nonsymmetric linear systems. J. Comput. Appl. Math. **226**, 103–113 (2009)
170. Sogabe, T., Yamamoto, Y.: Basic Mathematical Algorithm for Computational Science (in Japanese). Zhang, S.-L. (ed.), Kaneda, Y., (ed. in chief), Sasai, M (ed. in chief), Kyoritsu Shuppan, Tokyo (2019)

171. Sogabe, T., Zhang, S.-L.: A COCR method for solving complex symmetric linear systems. J. Comput. Appl. Math. **199**, 297–303 (2007)
172. Sogabe, T., Zhang, S.-L.: An extension of the COCR method to solving shifted linear systems with complex symmetric matrices. E. Asian J. Appl. Math. **1**, 97–107 (2011)
173. Sonneveld, P.: CGS, a fast Lanczos-type solver for nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **10**, 36–52 (1989)
174. Sonneveld, P., van Gijzen, M.B.: IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. SIAM J. Sci. Comput. **31**, 1035–1062 (2008)
175. Soodhalter, K.M., Szyld, D.B., Xue, F.: Krylov subspace recycling for sequences of shifted linear systems. Appl. Numer. Math. **81**, 105–118 (2014)
176. Sosonkina, M., Watson, L.T., Kapania, R.K., Walker, H.F.: A new adaptive GMRES algorithm for achieving high accuracy. Numer. Linear Algebra Appl. **5**, 275–297 (1998)
177. Stiefel, E.: Relaxationsmethoden bester strategie zur lösung linearer gleichungssysteme. Comment. Math. Helv. **29**, 157–179 (1955)
178. Sugiyama, M.: Superfast-trainable multi-class probabilistic classifier by least-squares posterior fitting. IEICE Trans. Inform. Syst. **E93-D**, 2690–2701 (2010)
179. Suzuki, M.: Generalized Trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems. Communs Math. Phys. **51**, 183–190 (1976)
180. Sylvester, J.J.: On the equation to the secular inequalities in the planetary theory. Philos. Mag. **16**, 267–269 (1883)
181. Tadano, H.: Development of the Block BiCGGR2 method for linear systems with multiple right-hand sides. Japan J. Indust. Appl. Math. **36**, 563–577 (2019)
182. Tadano, H., Sakurai, T., Kuramashi, Y.: Block BiCGGR: A new block Krylov subspace method for computing high accuracy solutions. JSIAM Lett. **1**, 44–47 (2009)
183. Takahasi, H., Mori, M.: Double exponential formulas for numerical integration. Publ. Res. Inst. Math. Sci. **9**, 721–741 (1973)
184. Takayama, R., Hoshi, T., Sogabe, T., Zhang, S.-L., Fujiwara, T.: Linear algebraic calculation of the Green's function for large-scale electronic structure theory. Phys. Rev. B **73**, 165108 (2006)
185. Tanio, M., Sugihara, M.: GBi-CGSTAB(s, L): IDR(s) with higher-order stabilization polynomials. J. Comput. Appl. Math. **235**, 765–784 (2010)
186. Tatsuoka, F., Sogabe, T., Miyatake, Y., Kemmochi, T., Zhang, S.-L.: Computing the matrix fractional power with the double exponential formula. Electron. Trans. Numer. Anal. **54**, 558–580 (2021)
187. Tatsuoka, F., Sogabe, T., Miyatake, Y., Zhang, S.-L.: A cost-efficient variant of the incremental newton iteration for the matrix pth root. J. Math. Res. Appl. **37**, 97–106 (2017)
188. Tatsuoka, F., Sogabe, T., Miyatake, Y., Zhang, S.-L.: Algorithms for the computation of the matrix logarithm based on the double exponential formula. J. Comput. Appl. Math. **373**, 112396 (2020)
189. Taylor, D.: Analysis of the Look Ahead Lanczos Algorithm. Ph.D. thesis, University of California, Berkley (1982)
190. Teng, H., Fujiwara, T., Hoshi, T., Sogabe, T., Zhang, S.-L., Yamamoto, S.: Efficient and accurate linear algebraic methods for large-scale electronic structure calculations with nonorthogonal atomic orbitals. Phys. Rev. B **83**, 165103 (2011)
191. Trefethen, L.N., Bau, D.: Numerical Linear Algebra: Twenty-Fifth. Anniversary SIAM, Philadelphia (2022)
192. Trefethen, L.N., Weideman, J.A.C.: The exponentially convergent trapezoidal rule. SIAM Rev. **56**, 385–458 (2014)
193. Trottenberg, U., Oosterlee, C.W., Schüller, A.: Multigrid. Academic Press, NewYork (2001)
194. van der Sluis, A.: Condition numbers and equilibration of matrices. Numer. Math. **14**, 14–23 (1969)
195. van der Vorst, H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **13**, 631–644 (1992)

196. van der Vorst, H.A.: Iterative Krylov Methods for Large Linear Systems. Cambridge University Press, Cambridge (2003)
197. van der Vorst, H.A., Melissen, J.B.M.: A Petrov-Galerkin type method for solving $\boldsymbol{Ax}$=$\boldsymbol{b}$, where A is symmetric complex. IEEE Trans. Magn. **26**, 706–708 (1990)
198. van Gijzen, M.B.: A polynomial preconditioner for the GMRES algorithm. J. Comput. Appl. Math. **59**, 91–107 (1995)
199. van Gijzen, M.B., Sonneveld, P.: Algorithm 913: An elegant IDR(s) variant that efficiently exploits biorthogonality properties. ACM Trans. Math. Softw. **38**, 5:1–5:19 (2011)
200. Varga, R.S.: Matrix Iterative Analysis, 2nd edn. Springer-Verlag, Berlin Heidelberg (2000)
201. Vinsome, P.K.W.: Orthomin, an iterative method for solving sparse sets of simultaneous linear equations. In: Fourth Symposium on Reservoir Simulation, pp. 149–159. Society of Petroleum Engineers of AIME (1976)
202. Vital, B.: Etude de Quelques Méthodes de Résolution de Problèmes Linéaires de Grande Taille sur Multiprocesseur. Ph.D. thesis, Univ. de Rennes I, Rennes, France (1990)
203. Weiss, R.: Parameter-Free Iterative Linear Solvers, 1st edn. Akademie Verlag, Berlin (1996)
204. Wesseling, P.: An Introduction to Multigrid Methods. Wiley, Chichester (1991)
205. Wesseling, P., Sonneveld, P.: Numerical experiments with a multiple grid and a preconditioned Lanczos type method. In: Rautmann, R. (ed.) Approximation Methods for Navier-Stokes Problems, pp. 543–562. Springer, Berlin, Heidelberg (1980)
206. Yin, J.-F., Hayami, K.: Preconditioned GMRES methods with incomplete Givens orthogonalization method for large sparse least-squares problems. J. Comput. Appl. Math. **226**, 177–186 (2009)
207. Young, D.M.: The search for high level parallelism for the iterative solution of large sparse linear systems. In: Carey, G.F. (ed.) Parallel Supercomputing: Methods, Algorithms and Applications, pp. 89–105. Wiley, Chichester, UK (1989)
208. Young, D.M., Vona, B.R.: On the use of rational iterative methods for solving large sparse linear systems. Appl. Numer. Math. **10**, 261–278 (1992)
209. Zemke, J.-P.M.: Variants of IDR with partial orthonormalization. Electron. Trans. Numer. Anal. **46**, 245–272 (2017)
210. Zhang, J.H., Zhao, J.: A novel class of block methods based on the block $AA^\top$-Lanczos bi-orthogonalization process for matrix equations. Int. J. Comput. Math. **90**, 341–359 (2013)
211. Zhang, L., Nodera, T.: A new adaptive restart for GMRES(m) method. ANZIAM J. **46**, C409–C425 (2004)
212. Zhang, S.-L.: GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems. SIAM J. Sci. Comput. **18**, 537–551 (1997)

Index

T. Sogabe, *Krylov Subspace Methods for Linear Systems*,
Springer Series in Computational Mathematics 60,
https://doi.org/10.1007/978-981-19-8532-4