A LOOK-AHEAD VARIANT OF THE LANCZOS ALGORITHM AND ITS APPLICATION TO THE QUASI-MINIMAL RESIDUAL METHOD FOR NON-HERMITIAN LINEAR SYSTEMS

by

Noël Maurice Nachtigal

B.S. Mathematics and Physics, Univ. of South Carolina (1987)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology
August 1991

©Massachusetts Institute of Technology 1991
All Rights Reserved

Signature of Author ____________________________________________

Department of Mathematics
August 16, 1991

Certified by ________________________________________________

Lloyd N. Trefethen
Associate Professor of Applied Mathematics
Thesis Supervisor

Accepted by ________________________________________________

Alar Toomre
Chairman, Applied Mathematics Committee

Accepted by ________________________________________________

Sigurdur Helgason
Chairman, Departmental Graduate Committee

ARCHIVES
M A S S A C H U S E T T S I N S T I T U T E O F T E C H N O L O G Y

MAR 1 1 1992
LIBRARIES
A Look-Ahead Variant of the Lanczos Algorithm 
and its Application to the Quasi-Minimal 
Residual Method for Non-Hermitian Linear Systems 

by 

Noël Maurice Nachtigal 

Submitted to the Department of Mathematics 
on August 16, 1991, in partial fulfillment of the 
requirements for the degree of 
Doctor of Philosophy 

Abstract 

The Lanczos algorithm can be used both for eigenvalue problems and to solve linear systems. However, when applied to non-Hermitian matrices, the classical Lanczos algorithm is susceptible to breakdowns and potential instabilities. In addition, the biconjugate gradient (BCG) algorithm, which is the natural generalization of the conjugate gradient algorithm to non-Hermitian linear systems, has a second source of breakdowns, independent of the Lanczos breakdowns. In this thesis, we present two new results. We propose an implementation of a look-ahead variant of the Lanczos algorithm which overcomes the breakdowns by skipping over those steps where a breakdown or a near-breakdown would occur. The new algorithm can handle look-ahead steps of any length and requires the same number of matrix-vector products and inner products per step as the classical Lanczos algorithm without look-ahead. Based on the proposed look-ahead Lanczos algorithm, we then present a novel BCG-like approach, the quasi-minimal residual (QMR) method, which avoids the second source of breakdowns in the BCG algorithm. We present details of the new method and discuss some of its properties. In particular, we discuss the relationship between QMR and BCG, showing how one can recover the BCG iterates, when they exist, from the QMR iterates. We also present convergence results for QMR, showing the connection between QMR and the generalized minimal residual (GMRES) algorithm, the optimal method in this class of methods. Finally, we give some numerical examples, both for eigenvalue computations and for non-Hermitian linear systems. 

Thesis Supervisor: Lloyd N. Trefethen 
Title: Associate Professor of Applied Mathematics
Contributions of this thesis

The main contributions of this thesis are a new variant of the Lanczos algorithm with look-ahead, capable of handling blocks of arbitrary size, and its combination with the quasi-minimal residual method to obtain a new algorithm for general non-Hermitian matrices. The new variant of the Lanczos algorithm is the result of joint work with Roland Freund and Martin Gutknecht, while the new quasi-minimal residual algorithm is the result of joint work with Roland Freund. All the results discussed in this thesis have been presented elsewhere, and here they are merely summarized in a coherent way. A very detailed presentation of the results, together with details of an actual implementation and listings of the corresponding FORTRAN codes, can be found in the RIACS technical reports by Freund, Gutknecht, and Nachtigal [21] and by Freund and Nachtigal [23, 24]. Shorter versions of these reports are to appear in the SIAM Journal on Scientific and Statistical Computing and Numerische Mathematik, respectively, and are also available as ETH technical reports [22, 25]. The basis for most of the work presented in this thesis can be found in the work of Martin Gutknecht and Roland Freund. An extensive treatment of the theory behind the Lanczos algorithm, its exact breakdowns and its connections with orthogonal polynomials and Padé tables can be found in two reports by Martin Gutknecht [34, 35] submitted to the SIAM Journal on Matrix Analysis and Applications. Here we extend this work to handle the case of near-breakdowns and discuss an implementation of the resulting look-ahead Lanczos algorithm. The quasi-minimal residual approach was first proposed by Freund [20] for the case of complex symmetric matrices. It is extended here to the case of general non-Hermitian matrices.
Acknowledgments

If I were to properly acknowledge all the professors and fellow students who have influenced my life over the two decades of study, I would fill several pages and probably still leave someone unmentioned. I will instead just mention a few, who epitomize the selflessness, dedication, and personal touch that have characterized so many.

No amount of thanks would be enough to express how grateful I am to the two people who have guided me during my graduate studies, Nick Trefethen and Roland Freund. Nick has patiently started me down the path of mathematical research, always letting me try yet another problem when the current one wasn’t appealing enough. As an advisor and a friend, he always supported me, and his office door was always open, whether for a hint when I was struggling with research, or for a friendly ear and some advice when I was struggling with life. Roland, whom I had the exceptional luck to meet and work with as a summer student at RIACS, has selflessly shared with me his vast knowledge and endless source of ideas. He has put up with my constant pestering and stream of questions, as witnessed by some 900+ mail messages accumulated in 10 months.

Most of the writing for this thesis was done during visits to ETH in Zürich and Université Paris VI in Paris. Thanks are due to Martin Gutknecht and Yvon Maday, who have made the visits possible and have extended their warm hospitality, as well as to Markus Hegland at ETH and the entire group at Paris VI, who made the stays so memorable. Thanks are also due to Martin Gutknecht and Marlis Hochbruck for reading drafts of the thesis and making valuable suggestions.

Finally, and foremost, I wish to thank my family. I can never hope to repay the many sacrifices they have made for me over the years. I can only hope that they will do me the honor of accepting this thesis as part of their success. Next to their influence and contribution, all others fade, and it is to them that the thesis is lovingly dedicated.
Table of Contents

1. Introduction ......................................................... 9
2. Background
   2.1. General Information ........................................ 13
   2.2. Computing Residual Polynomials ............................ 16
   2.3. The Arnoldi Process ......................................... 20
   2.4. The Lanczos Process ......................................... 24
   2.5. Orthogonal Polynomials ..................................... 27
3. The Lanczos Algorithm with Look-Ahead
   3.1. The Classical Algorithm .................................... 31
   3.2. Connection with Polynomials and Moment Matrices ....... 33
   3.3. Basic Idea and Notation of Look-Ahead Lanczos .......... 38
   3.4. The Parlett, Taylor, and Liu Approach .................... 41
   3.5. The Mismatch Theorem ...................................... 44
   3.6. The Freund, Gutknecht, and Nachtigal Approach ........... 48
   3.7. The Near-Breakdown Case ................................... 52
   3.8. Building Blocks ............................................ 57
   3.9. Implementation Details .................................... 63
   3.10. Numerical Examples (Eigenvalues) ......................... 68
4. The Quasi-Minimal Residual Algorithm
   4.1. The Basic QMR Approach ................................... 71
   4.2. A QMR Algorithm ........................................... 78
   4.3. Connection with GMRES .................................... 81
   4.4. Residual Norm Upper Bound ................................ 85
   4.5. Residual Vector Updating .................................. 86
   4.6. Connection with BCG ....................................... 87
   4.7. Numerical Example (Linear Systems) ....................... 93
5. Conclusions ......................................................... 101
6. Appendix: Notation List ........................................... 103
7. References ......................................................... 109
Chapter 1. Introduction

One of the fundamental building blocks of numerical computing is the ability to solve linear systems $Ax = b$. These systems arise very often in scientific computations, for example, from the discretization by finite differences or by finite elements of partial differential equations, or as intermediate steps in computing the solution of nonlinear problems.

One approach that can be used to solve linear systems consists of direct methods, wherein one computes a factorization of the matrix $A$ and then uses the factors to compute $x = A^{-1}b$. Here $A$ is assumed to be nonsingular. The classic examples of direct methods are the LU and QR factorizations. While these methods are relatively well understood and robust algorithms are available to implement them, they have the drawback of requiring large amounts of work and storage. If $A$ is an $N \times N$ matrix, then in general the LU or QR factorization requires $\mathcal{O}(N^3)$ work and $\mathcal{O}(N^2)$ storage. This remains true even if the matrix $A$ is sparse, with only $\mathcal{O}(N)$ nonzero elements in it. Since many of the matrices that arise, for example, from the discretization of partial differential equations are sparse, work and storage become important issues in practice. Furthermore, these systems are typically so large as to make the storage requirements of direct methods prohibitive. This is especially true when the problems modeled are 3-dimensional rather than 2-dimensional. Thus, using direct methods on sparse matrices is usually not an attractive option.

An alternative to direct methods is iterative methods. In these, one computes a sequence of approximate solutions to the linear system, starting from an initial guess and repeatedly applying some algorithm to generate subsequent approximations from previous ones. The first examples of iterative methods are from the nineteenth century, the Jacobi $[41]$ and Gauss-Seidel iterations $[26, 72]$. The Gauss-Seidel iteration introduced the idea of relaxation $[74]$, extended to overrelaxation in the successive overrelaxation (SOR) iteration $[19, 84, 85]$. Another method, introduced at the beginning of the twentieth century, was Richardson’s method $[62]$, which nicely shows the relationship between iterative methods, polynomials in the matrix, and the spectrum of the matrix. More recent iterative methods are the conjugate gradient (CG) algorithm of Hestenes and Stiefel $[37]$, the biconjugate gradient (BCG) algorithm $[47, 18]$, the Chebyshev iteration $[28, 30, 31, 51, 52]$, GMRES $[71]$, and many others. Typically, such methods require the computation of a few (one or two) matrix-vector multiplications at each step, which is at worst

9
an $O(N^2)$ operation, but can be an $O(N)$ operation for sparse matrices. The hope is that the method converges quickly, thus saving work when compared to a direct method. Furthermore, these methods can further reduce work and storage requirements by taking full advantage of sparsity or of other structure that leads to matrix-free algorithms, where the matrix $A$ is never built explicitly.

An important issue in iterative methods is the symmetry of the matrix. For general Hermitian positive definite matrices, the conjugate gradient algorithm is a very powerful method, especially when preconditioned. However, it cannot be used directly on non-Hermitian matrices. There are two variants used for solving non-Hermitian systems, CGNR [37] (robustly implemented in the LSQR algorithm of Paige and Saunders [58]) and CGNE [9, 36], both based on symmetrizing the system by implicitly forming and solving the normal equations. However, the condition number of the iteration matrix is then the square of the condition number of the original non-Hermitian matrix: as a result, convergence often becomes unacceptably slow. Finally, from among the many methods proposed for non-Hermitian systems, some of the more robust methods, such as ORTHOMIN [82] or the generalized minimal residual algorithm (GMRES) [71], typically require work and storage that increase linearly with the iteration number. This has led to a concerted effort to find a robust iterative method suitable for non-Hermitian systems, with work and storage requirements comparable to those of the classic CG algorithm. Significant interest has focused on the biconjugate gradient algorithm (BCG) [47, 18], a natural extension of the CG algorithm to non-Hermitian systems, and several variants, such as conjugate gradients squared (CGS) [73], or more recently, Bi-CGSTAB [81], have been proposed. An attractive feature of these methods is their use of a three-term recurrence, which makes implementation cheap in terms of both work and storage. However, while these methods are known to work well in many cases, they are susceptible to breakdowns and instabilities, making them unsuitable for use as black-box solvers.

BCG has two independent sources of breakdowns, one coming from a breakdown of the underlying Lanczos method, and another coming from the Galerkin condition used to generate the iterates. The work presented here has been directed mainly at curing or avoiding these breakdowns. First, inspired by the recent work by Gutknecht [34, 35], we propose a new implementation of the nonsymmetric Lanczos algorithm, using look-ahead to handle the possible breakdowns. This cures one of the breakdowns in BCG, leaving only the Galerkin breakdown. We then combine the look-ahead Lanczos algorithm with the quasi-minimal residual (QMR) method
proposed by Freund [20] to obtain a robust algorithm for non-Hermitian linear systems. The resulting QMR algorithm is closely related to BCG, but unlike BCG, does not have any inherent breakdown sources, and hence achieves smoother convergence curves and good numerical properties. It is possible to derive convergence results for QMR, showing its connection to GMRES. One can also derive results showing the connection between QMR and BCG and how one may use QMR to stably recover the BCG iterates when they exist.

The outline of the thesis is as follows. In Chapter 2, we discuss the background for the Arnoldi and Lanczos methods, with their application to eigenvalue problems and linear systems. We focus on the orthogonality conditions imposed in each method and their effect on the properties of the corresponding linear solvers. In this context, we briefly discuss the FOM, GMRES, BCG, and QMR methods. We also review the connection between the Arnoldi and Lanczos methods and orthogonal polynomials, focusing on how the use of different inner products leads to long recurrences for Arnoldi and short recurrences for Lanczos. In Chapter 3, we discuss the Lanczos algorithm in some detail, with particular emphasis on its breakdowns. We review the work of Parlett, Taylor, and Liu [61], who were the first to present a variant of the Lanczos algorithm that uses look-ahead to avoid the breakdowns, and who discussed details of an implementation for the case of blocks of size at most 2. We then present our variant of the algorithm with look-ahead, and we discuss details of its implementation. Our implementation has the advantage that it extends easily to blocks of size greater than 2, while performing the same number of inner products per step as the classical Lanczos algorithm. As one will see, our approach differs from the approach of Parlett, Taylor, and Liu even for the case of blocks of size 1 or 2. We also present some numerical examples obtained by applying the Lanczos algorithm to the computation of eigenvalues of matrices. In Chapter 4, we combine the Lanczos algorithm with the QMR algorithm to obtain a new algorithm for non-Hermitian systems. We discuss the QMR approach and its implementation, as well as some of its properties. In particular, we present the connections with BCG and GMRES. We also present some numerical examples with the new algorithm applied to non-Hermitian linear systems. Finally, in Chapter 5, we make some concluding remarks. A summary of the notation used in this thesis can be found in the Appendix.
Chapter 2. Background

Section 2.1. General Information

We are interested in solving the linear system

\[ Ax = b \]  \hspace{1cm} (2.1)

with a polynomial based iterative method. Here \( A \) is an arbitrary \( N \times N \) (possibly complex) nonsingular matrix, and \( b \in \mathbb{C}^N \) is a given right-hand side vector. We are particularly interested in the case when \( A \) is non-Hermitian, but the methods we are going to discuss also apply (usually in a simplified form) to Hermitian matrices.

Given an initial guess \( x_0 \in \mathbb{C}^N \) for the solution \( A^{-1}b \), let \( r_0 = b - Ax_0 \) denote the initial residual. A polynomial based iterative method (hereafter referred to simply as an iterative method) is one where we can write the solution \( x_n \) at step \( n \) as

\[ x_n = x_0 + q_n(A)r_0, \quad n \geq 1, \]  \hspace{1cm} (2.2)

or, expanding \( r_0 \) and grouping like terms,

\[ x_n = (I - Aq_n(A))x_0 + q_n(A)b = p_n(A)x_0 + q_n(A)b. \]  \hspace{1cm} (2.3)

Here \( I \) is the \( N \times N \) identity matrix, \( q_n(z) \) is a polynomial of degree at most \( n - 1 \) called the iteration polynomial, and \( p_n(z) = 1 - zq_n(z) \) is a polynomial of degree at most \( n \) called the residual polynomial, constrained by \( p_n(0) = 1 \). The residual \( r_n \) at step \( n \) is then given by

\[ r_n = b - Ax_n = r_0 - Aq_n(A)r_0 = (I - Aq_n(A))r_0 = p_n(A)r_0, \]  \hspace{1cm} (2.4)

while the error \( e_n \) at step \( n \) is given by

\[ e_n = A^{-1}b - x_n = A^{-1}r_n = A^{-1}p_n(A)r_0 = A^{-1}p_n(A)Ae_0 = p_n(A)e_0. \]  \hspace{1cm} (2.5)

As (2.3–2.5) show, the idea behind iterative methods is to choose \( q_n(z) \) such that, roughly speaking, \( q_n(A) \approx A^{-1} \), or equivalently to choose \( p_n(z) \) such that \( p_n(A) \approx 0 \), with \( p_n(0) = I \).

From (2.4) and (2.5) we also see that

\[ \frac{\|r_n\|_2}{\|r_0\|_2} \leq \|p_n(A)\|_2 \quad \text{and} \quad \frac{\|e_n\|_2}{\|e_0\|_2} \leq \|p_n(A)\|_2, \]  \hspace{1cm} (2.6)
where $\|\cdot\|_2$ denotes the 2-norm of a vector or a matrix. Suppose now that $A$ is diagonalizable. Then there exists a matrix $U = [u_1 \cdots u_N]$ with the eigenvectors as columns and a diagonal matrix $D = \text{diag}(\lambda_1, \ldots, \lambda_N)$ with the eigenvalues on the diagonal, such that $A = UDU^{-1}$. Let us define, for a polynomial $p(z)$ and a compact set $S \subset \mathbb{C}$,

$$\|p\|_S = \max_{z \in S} |p(z)|,$$

and let us denote by $\kappa(M)$ the 2-norm condition number of a matrix $M$,

$$\kappa(M) = \|M\|_2 \|M^{-1}\|_2.$$

From (2.4) we then have

$$\|r_n\|_2 = \|p_n(A)r_0\|_2 = \|UP_n(D)U^{-1}r_0\|_2 \leq \|U\|_2 \|U^{-1}\|_2 \|p_n(D)\|_2 \|r_0\|_2,$$

so that

$$\frac{\|r_n\|_2}{\|r_0\|_2} \leq \kappa(U)\|p_n\|_\Lambda, \quad (2.7)$$

where $\Lambda = \{\lambda_1, \ldots, \lambda_N\}$ denotes the spectrum of $A$. Similarly, we obtain from (2.5)

$$\frac{\|e_n\|_2}{\|e_0\|_2} \leq \kappa(U)\|p_n\|_\Lambda.$$

Since $p_n(0) = 1$, $\|p_n\|_\Lambda$ can be related to the distribution of eigenvalues with respect to the origin. However, also important is $\kappa(U)$, which is a measure of the normality of the matrix $A$. We recall that $A$ is normal if $A^HA = AA^H$, or equivalently, if it has a complete set of orthogonal eigenvectors. For normal matrices $\kappa(U) = 1$ (when the eigenvectors are scaled to be unit vectors), hence in this case convergence of an iterative method is governed exclusively by the distribution of eigenvalues. For non-normal matrices with large $\kappa(U)$, convergence may be governed more by the pseudospectra of the matrix, as Nachtigal, Reichel and Trefethen showed in [55].

The $\epsilon$-pseudospectrum $\Lambda_\epsilon(A)$ is defined [80] by

$$\Lambda_\epsilon(A) = \left\{ z \in \mathbb{C} : \|(zI - A)^{-1}\|_2 \geq \epsilon^{-1}, \epsilon \geq 0 \right\},$$

where $(zI - A)^{-1}$ is called the resolvent of $A$. An equivalent definition is

$$\Lambda_\epsilon(A) = \left\{ z \in \mathbb{C} : z \text{ is an eigenvalue of } A + \Delta, \Delta \in \mathbb{C}^{N \times N}, \|\Delta\|_2 \leq \epsilon \right\}.$$
From the pseudospectrum $\Lambda_\varepsilon(A)$, a bound for $\|p(A)\|_{\Lambda_\varepsilon(A)}$ can be obtained using Cauchy's integral for matrices. Given an $\varepsilon$-pseudospectrum $\Lambda_\varepsilon(A)$, its boundary $\partial \Lambda_\varepsilon$, and letting $L(\partial \Lambda_\varepsilon)$ denote the length of the boundary, we have

$$
\|p(A)\|_{\Lambda_\varepsilon(A)} = \left\| \frac{1}{2\pi i} \int_{\partial \Lambda_\varepsilon} p(z)(zI - A)^{-1} \, dz \right\|_2 \leq \frac{1}{2\pi} \frac{L(\partial \Lambda_\varepsilon)}{\varepsilon} \max_{z \in \Lambda_\varepsilon(A)} |p(z)|.
$$

With this, the corresponding bound for (2.7) is given by

$$
\frac{\|r_n\|_2}{\|r_0\|_2} \leq \frac{1}{2\pi} \frac{L(\partial \Lambda_\varepsilon)}{\varepsilon} \max_{z \in \Lambda_\varepsilon(A)} |p(z)|.
$$

When $\kappa(U)$ is large, this bound may be tighter than the bound in (2.7). In particular, when $A$ is not diagonalizable, this bound still holds, whereas (2.7) is vacuous (when $A$ is not diagonalizable, $\kappa(U)$ can be taken to be infinite).
Section 2.2. Computing Residual Polynomials

Let us turn now to the issue of finding a polynomial \( p_n(z) \) suitable for the iteration. One approach is an explicit one: given an \( \epsilon \)-pseudospectrum \( \Lambda_\epsilon(A) \), construct \( p_n \) such that \( \| p_n \|_{\Lambda_\epsilon(A)} \) is small, and then apply this polynomial until convergence. However, as \( \Lambda_\epsilon(A) \) is required beforehand and is usually not known in advance, this approach leads to hybrid methods, which have two phases. In a first phase, the pseudospectrum, or some region enclosing it, or some parameters depending on it are computed, and then in a second phase a suitable residual polynomial \( p_n(z) \) is computed and applied. Examples of methods from this class are numerous, such as the Richardson iteration [62], successive overrelaxation [19], the Chebyshev iteration [28, 30, 31, 51, 52], the semiterative methods of Eiermann, Niethammer, and Varga [14], and many others. Another approach is an implicit one, which bypasses the explicit computation of the pseudospectrum, leading to parameter-free methods. In this class we find the Jacobi [41] and Gauss-Seidel [26, 72] iterations, CG [37] (for Hermitian positive definite matrices) and its non-Hermitian variants CGNR [37] and CGNE [9, 36]. We also find Arnoldi-based methods such as ORTHOMIN [82] and GMRES [71], as well as Lanczos-based methods such as MINRES and SYMMLQ [57] (both for Hermitian indefinite matrices), BCG [47, 18] and QMR [20, 23, 24]. Finally, Nachtigal, Reichel, and Trefethen introduced another type of hybrid in [54]. This hybrid avoids the explicit computation of the pseudospectrum by using GMRES to compute a suitable residual polynomial directly, and then uses a Richardson iteration, which is cheaper than GMRES, to apply the GMRES polynomial until convergence.

We summarize this classification in Table 1 on the next page. As we indicated already, CGNR and CGNE symmetrize the system by implicitly constructing \( A^H A \) or \( A A^H \), respectively, and then apply CG to the resulting Hermitian positive definite matrix. Even though this approach is often ignored in practice, it can be quite effective in some cases [55]. From the remaining methods, two of the more successful ones, GMRES and QMR, use (2.4) directly to obtain a suitable polynomial. It is this approach that we will investigate further.

Let

\[
K_n(v, A) = \text{span}\{v, Av, \ldots, A^{n-1}v\}
\]

denote the \( n \)th Krylov space generated by \( A \) and a starting vector \( v \). We wish to compute \( q_n(A)r_0 \), with \( q_n(z) \) a polynomial of degree at most \( n - 1 \); hence we will build \( K_n(r_0, A) \). This is the reason why these methods are also called Krylov space
• Parameter-free methods
  – Classical methods
    Jacobi [41]
    Gauss-Seidel [26, 72]
  – CG based
    CG [37], CGNR [37], CGNE [9, 36]
  – Arnoldi-based
    ORTHOMIN [82], ORTHORES [86], ORTHODIR [86]
    IOM [66, 68], DIOM [68], FOM [67, 71], GMRES [71]
  – Lanczos-based
    BCG [47, 18], Lanczos/ORTHORES [42, 43, 35]
    Lanczos/ORTHODIR [42, 43, 35], CGS/BIOMINSQ [73, 35]
    BIORESSQ [35], BIODIRSQ [35], Bi-CGSTAB [81]
    MINRES [57], SYMMLQ [57], QMR [20, 23, 24]

• Parameter methods
  – Classical methods
    Richardson [62]
    Chebyshev [28, 30, 31]
    SOR [19, 84, 85], SSOR [2]
  – Hybrids requiring eigenvalue information
    Mantell's Chebyshev method [51, 52]
    Semiiterative methods [14, 13]
    Elman, Saad, and Saylor [15]
    Elman and Streit [16]
    Saad [69]
    etc.
  – Hybrids not requiring eigenvalue information
    Nachtigal, Reichel, and Trefethen [55]

Table 1. Classification of polynomial iterative methods
methods. Let \( v_1 \) be such that

\[
\beta v_1 = r_0
\]

for some scaling factor \( \beta \in \mathbb{R}, \beta \neq 0 \) (typically \( \beta = \|r_0\|_2 \)), and generate \( v_n \) such that

\[
v_n \in K_n(v_1, A) \setminus K_{n-1}(v_1, A), \quad n \geq 2. \tag{2.9}
\]

To do this, we use a recurrence of the form

\[
v_{n+1} = A v_n - \sum_{j=1}^{n} h_{jn} v_j, \quad n \geq 1. \tag{2.10}
\]

For practical purposes, such a recurrence has the advantage that the matrix \( A \) appears only as an operator and not element-wise. Let \( V^{(n)} \) denote the matrix whose columns are \( v_1, \ldots, v_n \). In matrix form, (2.10) is then written as

\[
A V^{(n)} = V^{(n+1)} H^{(n)}_e, \tag{2.11}
\]

where \( H^{(n)} = (h_{ij}), i, j = 1, \ldots, n, \) is the \( n \times n \) matrix of recurrence coefficients, and \( H^{(n)}_e \) is the \( (n + 1) \times n \) extended matrix

\[
H^{(n)}_e = \begin{bmatrix} H^{(n)} & * & \cdots & \cdots & * \\ 0 & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 1 \end{bmatrix}. \tag{2.12}
\]

Both \( H^{(n)} \) and \( H^{(n)}_e \) are unit upper Hessenberg; in particular, this means that their QR decomposition can be easily computed by means of Givens rotations, as we will show in Chapter 4. From (2.2), \( x_n \) is given by

\[
x_n = x_0 + q_n(A)r_0 = x_0 + V^{(n)} z_n, \quad z_n \in \mathbb{C}^n, \tag{2.13}
\]

where the correction to \( x_0 \) is a linear combination of the \( v_j \)'s with coefficients \( z_n \). As an example, if we choose \( \beta = 1 \) and \( h_{jn} = 0 \) for all \( j \) and \( n \), then \( v_n = A^{n-1} r_0 \), and \( z_n \) is the vector of coefficients of \( q_n(z) \). (2.8), (2.11) and (2.13) give

\[
r_n = b - A x_n = r_0 - A V^{(n)} z_n = r_0 - V^{(n+1)} H^{(n)}_e z_n
\]

\[
= V^{(n+1)} \left( \begin{bmatrix} \beta \\ 0 \\ \vdots \\ 0 \end{bmatrix} - H^{(n)}_e z_n \right) = V^{(n+1)} \left( d^{(n+1)} - H^{(n)}_e z_n \right), \tag{2.14}
\]
where
\[ d^{(n+1)} = [\beta \ 0 \ \cdots \ 0]^T \in \mathbb{R}^{n+1}. \] (2.15)

The utility of (2.14) depends largely on the properties of \( V^{(n+1)} \) and of \( H_e^{(n)} \). We will consider two possibilities: requiring the sequence of \( v_j \)'s to be orthogonal to itself, and requiring it to be orthogonal to another sequence of vectors. We will also consider imposing two different conditions on \( r_n \), a Galerkin condition and a least-squares condition, and we will investigate how the requirements on \( V^{(n)} \) affect the conditions on \( r_n \). In this manner, we will obtain four algorithms, FOM, GMRES, BCG, and QMR, of which only QMR is a new algorithm. The four algorithms can be summarized in the following table:

<table>
<thead>
<tr>
<th></th>
<th>Orthogonality</th>
<th>Biorthogonality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galerkin</td>
<td>Arnoldi/FOM</td>
<td>BCG</td>
</tr>
<tr>
<td>Least-Squares</td>
<td>GMRES</td>
<td>QMR</td>
</tr>
</tbody>
</table>
Section 2.3. The Arnoldi Process

The first possibility is to require $V^{(n)}$ to be unitary,

$$v_i^H v_j = \begin{cases} 1 & i = j, \\ 0 & \text{otherwise}. \end{cases} \quad (2.16)$$

As we will see, the advantage of this choice is that $V^{(n)}$ can be eliminated entirely from the formulas and one can obtain an optimal method while reducing the original $N \times N$ problem to an $n \times n$ problem. To enforce (2.16), one sets in (2.10)

$$h_{jn} = \frac{v_i^H A v_n}{v_j^H v_j} = v_j^H A v_n. \quad (2.17)$$

This leads to the Arnoldi process [1, 65, 66]. It was originally used to obtain the eigenvalues of $A$: the $n$ eigenvalues of $H^{(n)}$ turn out to be Ritz values for $A$ [65], and can be used as estimates for $n$ of the eigenvalues of $A$. However, the Arnoldi process can also be used to solve linear systems. To see how this can be done, let us first impose a Galerkin condition, by requiring that $r_n \perp V^{(n)}$. Using (2.12), (2.14), and (2.16), we have

$$\begin{pmatrix} V^{(n)} \end{pmatrix}^H r_n = \begin{pmatrix} V^{(n)} \end{pmatrix}^H \begin{pmatrix} V^{(n+1)} \end{pmatrix} \begin{pmatrix} d^{(n+1)} - H_e^{(n)} z_n \end{pmatrix}$$

$$= \begin{bmatrix} I_n & \vdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \begin{pmatrix} d^{(n+1)} - H_e^{(n)} z_n \end{pmatrix} = d^{(n)} - H^{(n)} z_n = 0,$$

which gives the simple $n \times n$ linear system

$$H^{(n)} z_n = d^{(n)}. \quad (2.18)$$

Here $I_n$ is the $n \times n$ identity matrix. This is Arnoldi’s method for linear systems, also called the full orthogonalization method (FOM) by Saad [67]. It is equivalent to the SYMMLQ algorithm of Paige and Saunders [57] when $A$ is Hermitian, and to the conjugate gradient algorithm [37] when $A$ is Hermitian positive definite. We will indicate later how the system (2.18) is solved. The other condition we consider is a least-squares condition, by requiring that $r_n \perp A V^{(n)}$. Again using (2.14), and the fact that $V^{(n+1)}$ is unitary, we have

$$\min_{x_n} \| b - A x_n \|_2 = \min_{z_n} \| V^{(n+1)} \left( d^{(n+1)} - H_e^{(n)} z_n \right) \|_2$$

$$= \min_{z_n} \| d^{(n+1)} - H_e^{(n)} z_n \|_2, \quad (2.19)$$
which is an \((n + 1) \times n\) least-squares problem. This is where the condition that \(V^{(n)}\) be unitary pays off the most: we have been able to obtain an optimal method while at the same time reducing the size of the problem. This is the generalized minimum residual algorithm (GMRES) of Saad and Schultz [71]. It is equivalent to the MINRES algorithm of Paige and Saunders [57] when \(A\) is Hermitian, and to the conjugate residual algorithm [75] when \(A\) is Hermitian positive definite.

We will indicate only briefly how either (2.18) or (2.19) are solved; Saad and Schultz give full details of a practical implementation in [71]. The key point is that \(H^{(n)}\) and \(H^{(n)}_e\) are upper Hessenberg, hence their QR decomposition can be computed and updated easily through a sequence of Givens rotations. This is a standard procedure, which we will discuss in Chapter 4; for more details, see also Golub and van Loan [29] or Saad and Schultz [71]. One computes a sequence of orthogonal matrices \(Q^{(n)}\) and upper triangular matrices \(R^{(n)}\) and \(R^{(n)}_e\) (differing only in their lower right-hand corner element) such that

\[
Q^{(n-1)}H^{(n)} = R^{(n)} \quad \text{and} \quad Q^{(n)}H^{(n)}_e = \begin{bmatrix} R^{(n)}_e \\ 0 & \cdots & 0 \end{bmatrix}.
\] (2.20)

The derivation is presented in section 4.1. From (2.18) and (2.20), one gets for FOM

\[
\left( R^{(n)} \right) z_{n}^{\text{FOM}} = Q^{(n-1)}d^{(n)},
\]

while from (2.19) and (2.20), one gets for GMRES

\[
\left( R^{(n)}_e \right) z_{n}^{\text{GMRES}} = \left( Q^{(n)}d^{(n+1)} \right)_{1:n},
\]

where the subscript \(1:n\) indicates that only the first \(n\) elements of the vector are taken. It is worth noting here an important difference between the Galerkin approach and the least-squares approach. As seen above, FOM solves the linear system (2.18) to obtain its iterate \(x_{n}^{\text{FOM}}\). If \(H^{(n)}\) is singular, then the system is inconsistent (the proof is given in section 4.4) and the algorithm breaks down. On the other hand, GMRES solves the least-squares problem (2.19) to obtain its iterate \(x_{n}^{\text{GMRES}}\). From (2.12), \(H^{(n)}_e\) always has full column rank, so the least-squares problem always has a unique solution. Hence, GMRES can at worst stagnate, but cannot break down.

The price paid for requiring \(V^{(n)}\) to be unitary is that the recurrences in (2.10) are in general long ones. This can be seen from a result by Faber and Manteuffel [17], who show that in general it is not possible to construct a method that has both short recurrences and an optimality property. They show that — apart from a few
anomalies — the only matrices for which methods with an \( s \)-step recurrence can be constructed are those whose field of values lies on a line segment in the complex plane,

\[
A = e^{i\theta}(aI + T), \quad T = T^H, \quad a \in \mathbb{C}, \theta \in \mathbb{R}.
\] (2.21)

In general, however, \( h_{jn} \) as given by (2.17) is nonzero, and furthermore, it cannot be computed from prior information. This means that at each step \( n \), all \( n \) coefficients \( h_{jn} \) in the \( n \)th column of \( H^{(n)} \) must be computed directly, which requires \( n \) inner products. In addition, all previous vectors \( v_j \) must be stored, hence work and storage requirements increase linearly with \( n \). For large systems, this becomes impractical, and therefore the method is usually restarted after every \( m \) steps. This algorithm is GMRES\((\nu_i)\) [71], and its drawback is that restarting often results in very slow convergence. Another possibility is to stop GMRES after it has computed a suitable polynomial, explicitly obtain this polynomial and then apply it using a cheaper method, such as a Richardson iteration. This algorithm is the hybrid algorithm proposed by Nachtigal, Reichel and Trefethen in [55].

Let us return to the Faber and Manteuffel result. For the Arnoldi case, one can easily show that if \( A \) is in the class (2.21), then the algorithm simplifies considerably: we obtain not only a tridiagonal \( H^{(n)} \), but a rather special tridiagonal matrix. Let us first denote by \( L \) the termination index of the Arnoldi process, i.e., the smallest integer such that

\[
v_{L+1}^H v_{L+1} = 0 \quad \iff \quad v_{L+1} = 0.
\]

Note that \( L \leq N \). For example, consider the case when \( A \) has only simple eigenvalues. Then \( L = N \) if and only if \( v_1 \) has components in all the eigenspaces of \( A \), so that the degree of the minimal polynomial of \( v_1 \) with respect to \( A \) is \( N \). Finally, we also note that \( AV^{(L)} = V^{(L)}H^{(L)} \).

**Proposition.** If \( A \) is in the class (2.21), then \( H^{(n)} \) is also in the class (2.21).

**Proof.**

\[
AV^{(n)} = V^{(n+1)}H^{(n)}_e \Rightarrow
\]

\[
H^{(n)} = \left( V^{(n)} \right)^H A \left( V^{(n)} \right) = \left( V^{(n)} \right)^H e^{i\theta}(aI + T) \left( V^{(n)} \right)
\]

\[
e^{i\theta} \left( aI + \left( V^{(n)} \right)^H T \left( V^{(n)} \right) \right),
\]

which is in the Faber-Manteuffel class (2.21). \( \square \)

The converse is more complicated; in its simple form, it holds only for \( n = N \).
Proposition. If \( L = N \) and \( H^{(N)} \) is in the class (2.21), then \( A \) is also in the class (2.21).

Proof.

\[
AV^{(N)} = V^{(N)}H^{(N)} \Rightarrow
\]

\[
A = \left( V^{(N)} \right) H^{(N)} \left( V^{(N)} \right)^H = \left( V^{(N)} \right) e^{i\theta} (aI + T) \left( V^{(N)} \right)^H
\]

\[
= e^{i\theta} \left( aI + \left( V^{(N)} \right) T \left( V^{(N)} \right)^H \right)
\]

which is in the Faber-Manteuffel class (2.21). \( \square \)

If \( L < N \), one needs to impose additional conditions; \( H^{(L)} \) being in the class (2.21) for \( L < N \) is not sufficient to ensure that \( A \) is also in the class. We illustrate this with a simple example. Suppose \( A \) is a block matrix

\[
A = \begin{bmatrix} T & 0 \\ 0 & N \end{bmatrix}, \quad T = T^H, \; N \neq N^H.
\]

By choosing a starting vector with zero components in the subpart where \( N \) acts, one can hide the nonsymmetry of \( A \), and obtain

\[
H^{(n)} = \left( H^{(n)} \right)^H \quad \text{for all } n \leq L;
\]

but clearly \( A \neq A^H \). We will not elaborate further on this subject; for further details, see Faber and Manteuffel [17] or Huckle's habilitation thesis [40].
Section 2.4. The Lanczos Process

The second possibility concerning the sequence of \( v_j \)'s is to require it to be orthogonal to another sequence of vectors, which we will denote by \( w_j \)'s. Suppose that an arbitrary vector \( w_1 \) is given, and vectors \( w_n \) are generated so that

\[
w_n \in K_n(w_1, A^T) \setminus K_{n-1}(w_1, A^T), \quad n \geq 2. \tag{2.22}\]

We now require biorthogonality between the \( v_j \)'s and the \( w_j \)'s,

\[
w_i^T v_j = \begin{cases} \delta_i = 0 & i = j, \\ 0 & \text{otherwise}. \end{cases} \tag{2.23}\]

As in (2.10), we use a recurrence of the form

\[
w_{n+1} = A^T w_n - \sum_{j=1}^{n} \tilde{h}_{jn} w_j, \quad n \geq 2, \tag{2.24}\]

so that \( A^T \) also appears only as an operator. We have as in (2.11)

\[
A^T W^{(n)} = W^{(n+1)} \tilde{H}^{(n)}, \tag{2.25}\]

where \( W^{(n)} \) is the matrix whose columns are \( w_1, \ldots, w_n \), and \( \tilde{H}^{(n)} \) has the same structure as \( H^{(n)} \) in (2.12). As we will see, the coefficients of \( v_j \) and \( w_j \) are in fact the same, so that \( \tilde{H}^{(n)} = H^{(n)} \). The advantage of this choice of \( V^{(n)} \) and \( W^{(n)} \) is that — in the generic case — \( H^{(n)} \) is tridiagonal, and thus the \( v_j \)'s and the \( w_j \)'s can be computed with short recurrences. On the other hand, as predicted by Faber and Manteuffel, in general we will no longer be able to obtain an optimal method. In addition, we will have to deal with a possible breakdown of the process. Note that the inner product in (2.23) uses the transpose and not the conjugate transpose. We should point out that instead of (2.22) and (2.23), one can equally well require that

\[
w_n \in K_n(w_1, A^H) \setminus K_{n-1}(w_1, A^H), \quad n \geq 2, \tag{2.26}\]

and

\[
w_i^H v_j = \begin{cases} \delta_i = 0 & i = j, \\ 0 & \text{otherwise}. \end{cases} \tag{2.27}\]

The two approaches do not differ significantly. We will use the transpose because this will simplify the formulas somewhat, by not requiring the complex conjugation of the recurrence coefficients. Also, the symmetric case \( A = A^T \), especially for complex \( A \), will be a special case for which the algorithm simplifies. In exchange, the Hermitian case \( A = A^H \) will no longer be a special case.
We will now show that the recurrence coefficients are the same for both the \(v_j\)'s and the \(w_j\)'s, and that one obtains short recurrences. We rewrite (2.10) and (2.24) as

\[
v_{n+1} = Av_n - \alpha_n v_n - \beta_n v_{n-1} - \sum_{j=1}^{n-2} h_{jn} v_j,
\]

\[
w_{n+1} = A^T w_n - \tilde{\alpha}_n w_n - \tilde{\beta}_n w_{n-1} - \sum_{j=1}^{n-2} \tilde{h}_{jn} w_j.
\]

From \(w_n^T v_{n+1} = v_n^T w_{n+1} = 0\) and \(w_{n-1}^T v_{n+1} = v_{n-1}^T w_{n+1} = 0\), one obtains

\[
\alpha_n = \frac{w_n^T Av_n}{\delta_n} = \frac{v_n^T A^T w_n}{\delta_n} = \tilde{\alpha}_n,
\]

\[
\beta_n = \frac{w_{n-1}^T Av_n}{\delta_{n-1}} = \frac{v_{n-1}^T A^T w_n}{\delta_{n-1}} = \tilde{\beta}_n.
\]

In addition, for \(j + 1 < n\),

\[
h_{jn} = \frac{w_j^T Av_n}{\delta_j} = \frac{v_n^T A^T w_n}{\delta_j}
\]

\[
= \left( w_{j+1} + \alpha_j w_j + \beta_j w_{j-1} + \sum_{i=1}^{j-2} h_{ij} w_j \right)^T v_n \frac{1}{\delta_j} = 0,
\]

and similarly

\[
\tilde{h}_{jn} = \frac{v_j^T A^T w_n}{\delta_j} = \frac{(Av_j)^T w_n}{\delta_j}
\]

\[
= \left( v_{j+1} + \alpha_j v_j + \beta_j v_{j-1} + \sum_{i=1}^{j-2} h_{ij} v_j \right)^T w_n \frac{1}{\delta_j} = 0,
\]

as \(w_j^T v_n = v_j^T w_n = 0\) for \(j + 1 < n\). Hence, \(h_{jn} = \tilde{h}_{jn} = 0\) for \(j + 1 < n\), so \(H^{(n)} = \tilde{H}^{(n)}\) is tridiagonal. This is the Lanczos process [46, 83]. The vectors \(v_j\) are called right Lanczos vectors, and the vectors \(w_j\) are called left Lanczos vectors. We will once again denote by \(L \leq N\) the termination index of the Lanczos process. Note the key role played by the recurrence (2.24) that enabled us to expand \(A^T w_j\) in a way that we were not able to do in (2.17). The price paid for the short recurrences is a possible breakdown of the process, in that \(\delta_n\) may be zero, without either \(v_n = 0\) or \(w_n = 0\). A robust implementation of this process requires more care, and will be discussed in detail in Chapter 3.
Once again, the \( n \) eigenvalues of \( H^{(n)} \) are Ritz values for \( A \) [61], and can be used as estimates for \( n \) of the eigenvalues of \( A \). For linear systems, we can again impose a Galerkin condition, by requiring \( r_n \perp W^{(n)} \), where orthogonality is with respect to the inner product used, either (2.23) or (2.27). For (2.23), we have from (2.14)

\[
\begin{align*}
(W^{(n)})^T r_n &= (W^{(n)})^T (V^{(n+1)}) (d^{(n+1)} - H_e^{(n)} z_n) \\
&= \begin{bmatrix} D^{(n)} & 0 \\ 0 & \end{bmatrix} (d^{(n+1)} - H_e^{(n)} z_n) = 0,
\end{align*}
\]

which gives, as in (2.18), the simple \( n \times n \) linear system

\[
H^{(n)} z_n = d^{(n)}.
\]

Here

\[
D^{(n)} = \text{diag}(\delta_1, \ldots, \delta_n)
\]

is assumed nonsingular. This is the biconjugate gradient algorithm (BCG), proposed by Lanczos [47] and later by Fletcher [18]. It is equivalent to the conjugate gradient algorithm [37] when \( A \) is Hermitian positive definite. \( H^{(n)} \) is tridiagonal, so its LU factorization consists of two bidiagonal matrices. One does not have to construct these two factors explicitly; it is possible to invert \( H^{(n)} \) in (2.29) implicitly with little work and storage. However, as before with Arnoldi's method for linear systems, the algorithm will break down if \( H^{(n)} \) is singular. The other condition is again a least-squares condition, requiring that \( r_n \perp AV^{(n)} \). In this case, we have as in (2.19)

\[
\min_{x_n} \| b - Ax_n \|_2 = \min_{z_n} \| V^{(n+1)} (d^{(n+1)} - H_e^{(n)} z_n) \|_2.
\]

However, as this time \( V^{(n+1)} \) is no longer unitary, it is not possible to reduce easily the problem any further. As we will see, the QMR algorithm uses (2.30) as a starting point to obtain a quasi-optimal solution; this will be described in detail in Chapter 4.
Section 2.5. Orthogonal Polynomials

We wish to conclude this chapter with another point of view on the conditions imposed on $V^{(n)}$: the construction of polynomials orthogonal with respect to a formal inner product. We will see that both the Arnoldi process and the Lanczos process can be viewed as constructing a sequence of polynomials orthogonal with respect to an inner product.

Let

$$\mathcal{P}_n = \{ \Psi(\lambda) = \gamma_0 + \gamma_1 \lambda + \cdots + \gamma_n \lambda^n : \gamma_0, \cdots, \gamma_n \in \mathbb{C} \}$$

denote the space of complex polynomials of degree at most $n$. We can then write

$$K_n(v_1, A) = \{ \Psi(A)v_1 : \Psi \in \mathcal{P}_{n-1} \},$$

$$K_n(w_1, A^T) = \{ \Psi(A^T)w_1 : \Psi \in \mathcal{P}_{n-1} \}.$$

From (2.10) we have for both the Arnoldi vectors and for the Lanczos right vectors

$$v_{n+1} = \Psi_n(A)v_1,$$  \hspace{1cm} (2.31)

while from (2.24) we have for the Lanczos left vectors

$$w_{n+1} = \Psi_n(A^T)w_1,$$  \hspace{1cm} (2.32)

for $n < L$. In all cases, $\Psi_n \in \mathcal{P}_n$ is a uniquely defined polynomial; for the Lanczos process, it is also monic ($\gamma_n = 1$).

Let us consider now the inner products used by the two methods. For the Arnoldi process, define the polynomial inner product $\langle \cdot, \cdot \rangle_A$ by

$$\langle \Phi, \Psi \rangle_A = (\Phi(A)v_1)^H (\Psi(A)v_1) = v_1^H \overline{\Phi}(A^H)\Psi(A)v_1,$$  \hspace{1cm} (2.33)

in which case the orthogonality relation (2.16) can be written as

$$\langle \Psi_n, \Psi \rangle_A = 0 \quad \text{for all } \Psi \in \mathcal{P}_{n-1}.$$

Here

$$\overline{\Psi}(\lambda) = \overline{\gamma_0} + \overline{\gamma_1} \lambda + \cdots + \overline{\gamma_n} \lambda^n.$$

For the Lanczos process, define the polynomial inner product $\langle \cdot, \cdot \rangle_L$ by

$$\langle \Phi, \Psi \rangle_L = (\Phi(A^T)w_1)^T (\Psi(A)v_1) = w_1^T \Phi(A)\Psi(A)v_1,$$  \hspace{1cm} (2.34)
in which case the biorthogonality relation (2.23) can be written as

$$\langle \Psi_n, \Psi \rangle_L = 0 \quad \text{for all } \Psi \in \mathcal{P}_{n-1}. $$

Let us also note that for the Lanczos process, in the alternate formulation (2.26), it turns out that the left Lanczos vectors are given by

$$w_{n+1} = \overline{\Psi}(A^H)w_1. $$

Hence, we define here the Lanczos inner product by

$$\langle \Phi, \Psi \rangle_L = (\overline{\Phi}(A^H)w_1)^H (\Psi(A)v_1) = w_1^H \Phi(A)\Psi(A)v_1, \quad (2.35) $$

in which case the orthogonality relation (2.27) can again be written as

$$\langle \Psi_n, \Psi \rangle_L = 0 \quad \text{for all } \Psi \in \mathcal{P}_{n-1}. $$

In all cases, the Arnoldi and Lanczos processes build a sequence of polynomials $\Psi_n$ orthogonal with respect to an inner product, (2.33) for the Arnoldi process, and (2.34) or (2.35) for the Lanczos process. When $A$ is diagonalizable, the expression for the inner product simplifies. Suppose again that $A = UDU^{-1}, U = [u_1 \cdots u_N], D = \text{diag}(\lambda_1, \ldots, \lambda_N).$ Then $A^T = U^{-T}DU^T,$ and $A^H = U^{-H}D^H U^H.$ Let

$$v_1 = U\xi^{(A)} \quad \text{and} \quad w_1 = U^{-T}\xi^{(T)} = U^{-H}\xi^{(H)}$$

be the decompositions of $v_1$ and $w_1$ in the eigenvectors of $A, A^T,$ and $A^H,$ respectively. Here

$$\xi^{(A)} = \begin{bmatrix} \xi_1^{(A)} \\ \vdots \\ \xi_N^{(A)} \end{bmatrix}, \quad \xi^{(T)} = \begin{bmatrix} \xi_1^{(T)} \\ \vdots \\ \xi_N^{(T)} \end{bmatrix}, \quad \text{and} \quad \xi^{(H)} = \begin{bmatrix} \xi_1^{(H)} \\ \vdots \\ \xi_N^{(H)} \end{bmatrix}. $$

Then for the Arnoldi inner product (2.33) we get

$$\langle \Phi, \Psi \rangle_A = \left( U\Phi(D)U^{-1}U\xi^{(A)} \right)^H \left( U\Psi(D)U^{-1}U\xi^{(A)} \right) $$

$$= \sum_{i,j=1}^N \overline{\Phi}(\lambda_i)\Psi(\lambda_j)\xi_i^{(A)}\xi_j^{(A)} u_i^H u_j, \quad (2.36) $$

while for the Lanczos inner product (2.34) we get

28
\[
\langle \Phi, \Psi \rangle_L = \left( U^{-T} \Phi(D) U^T U^{-T} \xi(T) \right)^T \left( U \Psi(D) U^{-1} U \xi(A) \right)
\]
\[
= \sum_{k=1}^{N} \Phi(\lambda_k) \Psi(\lambda_k) \xi_k^{(A)} \xi_k^{(T)}.
\]

We obtain a similar result for the alternate Lanczos inner product (2.31),
\[
\langle \Phi, \Psi \rangle_L = \left( U^{-H} \Phi(D^H) U^H U^{-H} \xi(H) \right)^H \left( U \Psi(D) U^{-1} U \xi(A) \right)
\]
\[
= \sum_{k=1}^{N} \Phi(\lambda_k) \Psi(\lambda_k) \xi_k^{(A)} \xi_k^{(H)}.
\]

The point is that the inner products are discrete polynomial inner products defined on a set of points with respect to the corresponding weights. The main difference between the Arnoldi and Lanczos inner products is the use of \( \Phi \Psi \) versus simply \( \Phi \Psi \), with direct implications in the length of the recurrences used by the two methods.

Let us once again consider the recurrence coefficients, this time in the framework of polynomials. Suppose \( \Phi_n(z) \) is either an Arnoldi or a Lanczos polynomial; we will require it to be monic, ignoring the scaling in the Arnoldi case. Then \( \Phi_0(z) = 1 \), and
\[
\Phi_n(z) = z \Phi_{n-1}(z) - \sum_{j=0}^{n-1} h_{jn} \Phi_j(z), \quad n \geq 1.
\]

If we now require
\[
\langle \Phi_n, \Phi_k \rangle = 0, \quad \text{for } k < n,
\]
with \( \langle \cdot, \cdot \rangle \) either the Arnoldi or one of the Lanczos inner products, then we get
\[
\langle \Phi_n, \Phi_k \rangle = 0 \Rightarrow \langle z \Phi_{n-1}, \Phi_k \rangle - \sum_{j=0}^{n-1} h_{jn} \langle \Phi_j, \Phi_k \rangle = 0
\]
\[
= \langle z \Phi_{n-1}, \Phi_k \rangle - h_{kn} \langle \Phi_k, \Phi_k \rangle = 0
\]
\[
\Rightarrow h_{kn} = \frac{\langle z \Phi_{n-1}, \Phi_k \rangle}{\langle \Phi_k, \Phi_k \rangle} \quad \text{for } k < n.
\]

For the Lanczos inner products, if \( k + 1 < n - 1 \), we have
\[
\langle z \Phi_{n-1}, \Phi_k \rangle = \langle \Phi_{n-1}, z \Phi_k \rangle = \langle \Phi_{n-1}, \Phi_{k+1} + \sum_{j=0}^{k} h_{jk} \Phi_j \rangle
\]
\[
= \langle \Phi_{n-1}, \Phi_{k+1} \rangle + \sum_{j=0}^{k} h_{jk} \langle \Phi_{n-1}, \Phi_j \rangle = 0.
\]
However, for the Arnoldi inner product, we cannot move the $z$ with the $\Phi_k$, because of the conjugation in the inner product. In fact, as Szegö shows in [78], the continuous version of polynomial inner products of the form (2.36) leads to $n$-term recurrences, or rather decompositions, when defined on an arbitrary curve. There are a few special cases, such as straight lines, circles, ellipses, where a fixed term recurrence is possible, but in general the recurrences are long. This means that in our case, $h_{kn}$ is in general nonzero for the Arnoldi inner product, and we recover the long recurrences of the Arnoldi process when applied to an arbitrary matrix. The exceptional case is again matrices that "live" on a line segment, which is the Faber and Manteuffel class of matrices (2.21).

We conclude this chapter with one final note on the inner products used by the Arnoldi and Lanczos processes. As shown above, one of the crucial differences between the two is that Arnoldi naturally leads to long recurrences, while Lanczos naturally leads to short recurrences. There is, however, another very important difference: the Arnoldi inner product is always a true inner product, while the Lanczos inner product is in general only a formal inner product. By this we mean that for Arnoldi, the inner product is positive definite

$$\langle \Psi, \Psi \rangle_A \geq 0 \quad \text{for all } \Psi, \quad \text{and} \quad \langle \Psi, \Psi \rangle_A = 0 \iff \Psi = 0,$$

as long as the degree of $\Psi$ does not exceed the degree of the minimal polynomial of $\nu_1$ with respect to $A$. For Lanczos, the inner product is indefinite, in that there can exist polynomials $\Psi \neq 0$ of any degree such that

$$\langle \Psi, \Psi \rangle_L < 0 \quad \text{or} \quad \langle \Psi, \Psi \rangle_L = 0.$$

As we will see in Chapter 3, the case when $\langle \Psi, \Psi \rangle_L = 0$ with $\Psi \neq 0$ will constitute a breakdown of the algorithm, and we will have to use look-ahead to enable the algorithm to continue beyond such a point.
Chapter 3.
The Lanczos Algorithm with Look-Ahead

Section 3.1. The Classical Algorithm

In this chapter, we will investigate in more detail the Lanczos algorithm. We first recall the classical Lanczos algorithm and some of its connections with orthogonal polynomials and moment matrices. As already mentioned, the algorithm as presented in section 2.4 has a possible breakdown; we will therefore focus our attention on the idea of using look-ahead to cure this problem. After introducing the basic idea and notation of the look-ahead Lanczos process, we will first discuss the variant proposed by Parlett, Taylor, and Liu in [61]. We will then propose our own variant of look-ahead Lanczos, and briefly discuss some aspects related to its implementation. We have already discussed this variant in [21, 22] and a more detailed discussion of an actual implementation can be found there.

The Lanczos algorithm was proposed by Cornelius Lanczos in 1950 [46]. As indicated in Chapter 2, one starts the process with two vectors \( v_1 \) and \( w_1 \) and then generates biorthogonal bases for \( K_n(v_1, A) \) and \( K_n(w_1, A^T) \) using three-term recurrences:

\[
\begin{align*}
  v_{n+1} &= A v_n - \alpha_n v_n - \beta_n v_{n-1}, \\
  w_{n+1} &= A^T w_n - \alpha_n w_n - \beta_n w_{n-1}.
\end{align*}
\]  

(3.1)

The coefficients \( \alpha_n \) and \( \beta_n \) are chosen so as to enforce the biorthogonality condition

\[
w_i^T v_j = \begin{cases} 
\delta_i \neq 0 & i = j, \\
0 & \text{otherwise},
\end{cases}
\]

(3.2)

which gives

\[
\alpha_n = \frac{w_n^T A v_n}{w_n^T v_n} \quad \text{and} \quad \beta_n = \frac{w_{n-1}^T A v_n}{w_{n-1}^T v_{n-1}}.
\]

(3.3)

As was shown in Chapter 2, three-term recurrences are sufficient, as the coefficients in front of all vectors of index \( n - 2 \) and lower vanish automatically as a result of the biorthogonality. In matrix notation, the classical Lanczos algorithm can then be summarized as

\[
A V^{(n)} = V^{(n+1)} H_e^{(n)},
\]

(3.4)

\[
A^T W^{(n)} = W^{(n+1)} H_e^{(n)},
\]

\[
(W^{(n)})^T (V^{(n)}) = D^{(n)} = \text{diag}(\delta_1, \ldots, \delta_n),
\]

(3.5)
where
\[
H_c^{(n)} = \begin{bmatrix}
\alpha_1 & \beta_2 \\
1 & \ddots & & \ddots \\
\ddots & \ddots & \ddots & \ddots \\
\beta_n & 1 & \ddots & 1 \\
1 & \alpha_n & \ddots & 1
\end{bmatrix}
\]
is the famous \((n + 1) \times n\) Lanczos tridiagonal matrix. In practice, for lack of a criterion for choosing \(w_1\), one usually sets \(w_1 = v_1\). We should also note that if one chooses \(w_1 = v_1\) when \(A = A^T\), then the second recurrence in (3.1) and (2.4) coincides with the first.

Let
\[
L_r = L_r(v_1, A) = \dim K_n(v_1, A)
\]
denote the grade of \(v_1\) with respect to \(A\) [83, p. 37], and
\[
L_l = L_l(w_1, A^T) = \dim K_n(w_1, A^T)
\]
denote the grade of \(w_1\) with respect to \(A^T\). Let
\[
L_* = \min(L_l, L_r) \leq N.
\]
As introduced in Chapter 2, let \(L\) denote the termination index of the Lanczos process. \(L\) is the largest integer such that \(v_L \neq 0\) and \(w_L \neq 0\) can be constructed from (3.1); the algorithm terminates at step \(L\) with the nonzero vectors \(v_L\) and \(w_L\). This gives two situations in which the classical Lanczos process can terminate. The first one, referred to as a regular termination, occurs when \(v_{L+1} = 0\) or \(w_{L+1} = 0\). This is the case that matches the termination condition from the Arnoldi process: the Lanczos algorithm has found an invariant subspace of \(C^N\), and \(L = L_* \leq N\). If \(v_{L+1} = 0\), then \(L = L_r\) and the right Lanczos vectors \(v_1, \ldots, v_L\) span an \(A\)-invariant subspace, while if \(w_{L+1} = 0\), then \(L = L_l\) and the left Lanczos vectors \(w_1, \ldots, w_L\) span an \(A^T\)-invariant subspace. The second case, referred to as a serious breakdown by Wilkinson [83, p. 389], occurs when \(w_L^T v_L = 0\) with neither \(v_L = 0\) nor \(w_L = 0\). In this case, \(L < L_*\) and the Lanczos vectors span neither an \(A\)-invariant nor an \(A^T\)-invariant subspace of \(C^N\). We will focus on this serious breakdown and attempt to find a way to cure it so that the algorithm does not terminate before \(L_*\).
Section 3.2. Connection with Polynomials and Moment Matrices

To better understand the serious breakdown, let us consider what happens in terms of polynomials. Using the notation introduced in Chapter 2 in (2.31) and (2.32),

\[ v_{n+1} = \Psi_n(A)v_1, \]
\[ w_{n+1} = \Psi_n(A^T)w_1, \]  

(3.6)

with \( \Psi_n \in \mathcal{P}_n \) a monic polynomial, and the Lanczos inner product \( \langle \cdot, \cdot \rangle_L \) defined by

\[ \langle \Phi, \Psi \rangle_L = (\Phi(A^T)w_1)^T (\Psi(A)v_1) = w_1^T \Phi(A)\Psi(A)v_1, \]

(3.7)

the biorthogonality relation (3.2) can be written as

\[ \langle \Psi_n, \Psi \rangle_L = 0 \quad \text{for all } \Psi \in \mathcal{P}_{n-1}, \]  

(3.8)

\[ \langle \Psi_n, \Psi_n \rangle_L \neq 0. \]  

(3.9)

However, as mentioned in Chapter 2, the inner product \( \langle \cdot, \cdot \rangle_L \) is in general indefinite, in that there can exist polynomials \( \Psi_n \neq 0 \) such that

\[ \langle \Psi_n, \Psi_n \rangle_L < 0 \quad \text{or} \quad \langle \Psi_n, \Psi_n \rangle_L = 0. \]

It is this latter case, when \( \langle \Psi_n, \Psi_n \rangle_L = 0 \) with \( \Psi_n \neq 0 \), that corresponds to a serious breakdown of the classical Lanczos algorithm.

We wish now to relate the breakdown of the Lanczos process to the matrix of moments associated with (3.7). Let

\[ \mu_j = \langle 1, \lambda^j \rangle_L = w_1^T A^j v_1, \quad j \geq 0, \]

denote the \( j \)th Schwarz constant or moment associated with the inner product (3.7), and let

\[ M^{(n)} = \begin{bmatrix} \mu_0 & \cdots & \cdots & \mu_n \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \mu_n & \cdots & \cdots & \mu_{2n} \end{bmatrix} \]

denote the \((n + 1) \times (n + 1)\) Hankel matrix of moments. A polynomial

\[ \Psi_n(\lambda) = \gamma_0 + \gamma_1 \lambda + \cdots + \gamma_n \lambda^n \in \mathcal{P}_n, \quad \Psi_n(\lambda) \neq 0, \]

33
that fulfills (3.8) is called a formally orthogonal polynomial (FOP) of degree \( n \) with respect to the formal inner product (3.7). From (3.8), \( \Psi_n \) is a FOP of degree \( n \) if and only if it fulfills the \( n \) orthogonality conditions

\[
\langle \lambda^j, \Psi_n \rangle_L = w_1^T A^j \Psi_n(A) v_1 = 0, \quad j = 0, \ldots, n - 1.
\] (3.10)

This can be written as the \( n \times (n + 1) \) system

\[
\begin{bmatrix}
\mu_0 & \cdots & \cdots & \mu_{n-1} & \mu_n \\
\vdots & & & \ddots & \vdots \\
\mu_{n-1} & \mu_n & \cdots & \cdots & \mu_{2n-1}
\end{bmatrix}
\begin{bmatrix}
\gamma_0 \\
\vdots \\
\gamma_n
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
\vdots \\
0
\end{bmatrix},
\] (3.11)

or, moving the last column to the right-hand side,

\[
M^{(n-1)}
\begin{bmatrix}
\gamma_0 \\
\vdots \\
\gamma_{n-1}
\end{bmatrix}
= -\gamma_n
\begin{bmatrix}
\mu_n \\
\vdots \\
\mu_{2n-1}
\end{bmatrix}.
\] (3.12)

We identify three cases:

- If \( M^{(n-1)} \) is nonsingular, then the linear system (3.12) always has a unique solution. The polynomial \( \Psi_n \) is a FOP, uniquely determined up to the scalar \( \gamma_n \). It is called regular and it is unique if it is required to be monic. Note also that from (3.12), a regular FOP \( \Psi_n \) has degree exactly \( n \), and furthermore, since (3.10) is empty for \( n = 0 \), FOPs of degree 0 (corresponding to the starting vectors) are always regular.

- If \( M^{(n-1)} \) is singular, but the system (3.12) is consistent, then there exist FOPs \( \Psi_n \) satisfying (3.8), but they are not uniquely determined (not even up to a scalar). In this case, they are called singular.

- If \( M^{(n-1)} \) is singular, and the system (3.12) is inconsistent, then there are no FOPs \( \Psi_n \) of degree \( n \). This case is called deficient by Gutknecht [34], or quasi-orthogonal by Draux [11]. It turns out that by relaxing the orthogonality condition (3.8), one can define so-called deficient FOPs \( \Psi_n \), which obey \( \langle \Psi_n, \Psi \rangle_L = 0 \) but only for \( \Psi \in P_j \) with \( j < n - 1 \); for full details, see [34].

We summarize the above in the following

**Theorem.** A FOP \( \Psi_n \) of degree \( n \) exists if and only if (3.12) has a solution with \( \gamma_n = 1 \). In addition, it is regular if and only if \( M^{(n-1)} \) is nonsingular.
This result, also cited by Gutknecht [34], establishes necessary and sufficient conditions for the existence of a regular FOP $\Psi_n$. However, to check for existence, it is simpler to use the following lemma, rather than check whether $M^{(n-1)}$ is singular or not.

**Lemma.** Let $\Psi_n$ be a regular FOP of degree $n$ with respect to the formal inner product (3.9). Then a regular FOP of degree $n + 1$ exists if and only if

$$\langle \Psi_n, \Psi_n \rangle_L \neq 0.$$ 

**Proof.** $\Psi_n$ is a regular FOP, hence it fulfills (3.10). Adding the equation for $\langle \Psi_{n+1}, \Psi_n \rangle_L$ to (3.11), we get the $n \times n$ system for the coefficients of $\Psi_n$

$$M^{(n)} \begin{bmatrix} \gamma_0 \\ \vdots \\ \vdots \\ \vdots \\ \gamma_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \langle \Psi_n, \Psi_n \rangle_L \end{bmatrix}.$$  \hspace{1cm} (3.13)

"⇒" Suppose that a regular FOP $\Psi_{n+1}$ exists. Then from (3.12), $M^{(n)}$ is nonsingular, which together with $\gamma_n = 1$ and (3.13), means that $\langle \Psi_n, \Psi_n \rangle_L \neq 0$.

"⇐" Suppose that $\langle \Psi_n, \Psi_n \rangle_L \neq 0$. Since $\Psi_n$ is a regular FOP, the solution to (3.13) is unique with $\gamma_n = 1$. Together with $\langle \Psi_n, \Psi_n \rangle_L \neq 0$, this means that $M^{(n)}$ is nonsingular, hence by the previous theorem, $\Psi_{n+1}$ is a regular FOP.

The point is that regular FOPs need not exist for every degree $n$. We wish to stress that this can happen because the Lanczos inner product is in general indefinite. The only exception occurs when $A = A^H$ and we choose $w_1 = \bar{v}_1$ (for the inner product (3.7)) or $w_1 = v_1$ (for the alternate formulation with inner product (2.35)). In this case, the Lanczos inner product becomes a true positive definite inner product. For such an inner product, there always exist unique monic orthogonal polynomials, up to the degree equal to the grade of $v_1$ with respect to $A$. In addition, these polynomials are true orthogonal polynomials with respect to a positive weight whose support is a set of points on the real axis [76]. They have real coefficients and hence (for 3.7)

$$\langle \Psi, \Psi \rangle_L = w_1^T \Psi(A)\Psi(A)v_1 = v_1^H \overline{\Psi(A^H)}\Psi(A)v_1 = \| \Psi(A)v_1 \|^2_2.$$ 

Thus, the Lanczos process cannot break down in this exceptional case. But in general, if $A \neq A^H$ or if we do not choose $w_1$ appropriately, the Lanczos inner product is indefinite and hence the Lanczos process can break down.
There is also another connection between the Lanczos process and moment matrices, also mentioned by Householder in [39] and by Parlett, Taylor, and Liu in [61]. For a given matrix $A$ and vectors $v_1$ and $w_1$, let

$$K^{(n)} = [v_1 \ A v_1 \ \cdots \ A^{n-1} v_1],$$
$$\tilde{K}^{(n)} = [w_1 \ A^T w_1 \ \cdots \ (A^T)^{n-1} w_1],$$

be the matrices of basis vectors for $K_n(v_1, A)$ and $K_n(w_1, A^T)$. It is easy to see that we have

$$M^{(n)} = \left(\tilde{K}^{(n)}\right)^T \left(K^{(n)}\right). \quad (3.14)$$

Furthermore, from (3.6) we see that

$$V^{(n)} = K^{(n)} C^{(n)},$$
$$W^{(n)} = \tilde{K}^{(n)} C^{(n)},$$

where $C^{(n)}$ is the $n \times n$ upper triangular matrix with the coefficients of the Lanczos polynomial $\Psi_{n-1}$ in its $n$th column. Inserting the above into (3.5) gives

$$\left(W^{(n)}\right)^T \left(V^{(n)}\right) = \left(C^{(n)}\right)^T \left(\tilde{K}^{(n)}\right)^T \left(K^{(n)}\right) \left(C^{(n)}\right)$$

$$= \left(C^{(n)}\right)^T \left(M^{(n)}\right) \left(C^{(n)}\right) = D^{(n)},$$

so that

$$M^{(n)} = \left(C^{(n)}\right)^{-T} \left(D^{(n)}\right) \left(C^{(n)}\right)^{-1}. \quad (3.15)$$

Since $\Psi_n$ is monic, the diagonal elements of $C^{(n)}$, and also of $(C^{(n)})^{-1}$, are all 1. Thus, (3.15) shows that the matrix $C^{(n)}$ of coefficients of the Lanczos polynomials makes up (in its inverse form) the $LL^T$ factorization of the moment matrix $M^{(n)}$. This result was also derived by Gragg in [33]. Furthermore, the biorthogonality products $\delta_i = w_i^T v_i$ are exactly the pivots used in the elimination (without pivoting). This approach also gives another derivation for the condition that guarantees no breakdowns in the Lanczos process. Let us once again consider the case when $A = A^H$ and $w_1 = \overline{v_1}$. Since $w_1 = \overline{v_1}$,

$$\tilde{K}^{(n)} = [\overline{v_1} \ A^T \overline{v_1} \ \cdots \ (A^T)^{n-1} \overline{v_1}],$$

so that together with $A = A^H$, we get from (3.14)

$$M^{(n)} = \left(\tilde{K}^{(n)}\right)^T \left(K^{(n)}\right) = \left(K^{(n)}\right)^H \left(K^{(n)}\right) = \left(M^{(n)}\right)^H.$$
Consider now $x^H M^{(n)} x$ for some $x \in \mathbb{C}^{n+1}$. We have

$$x^H M^{(n)} x = x^H \left( K^{(n)} \right)^T \left( K^{(n)} \right) x = x^H \left( K^{(n)} \right)^H \left( K^{(n)} \right) x$$

$$= \left( K^{(n)} x \right)^H \left( K^{(n)} x \right) = \| K^{(n)} x \|_2^2,$$

hence $M^{(n)}$ is Hermitian positive definite. Now Gaussian elimination without pivoting applied to a Hermitian positive definite matrix is guaranteed not to encounter a zero pivot, so the factorization (3.15) and the Lanczos algorithm will not break down when $A = A^H$ and $w_1 = \overline{v_1}$. We have recovered the condition we had before.

Finally, we wish to stress that a breakdown is the result of the choice of $v_1$ and $w_1$, and not of some defect of the matrix $A$. For any matrix, it is possible to generate breakdowns by choosing $v_1$ and $w_1$ appropriately; in particular, for any matrix $A$, the algorithm can be made to break down at the first step by choosing $v_1$ and $w_1$ to be biorthogonal.
Section 3.3. Basic Idea and Notation of Look-Ahead Lanczos

A breakdown at step \( n \) does not necessarily mean that there are no regular FOPs of higher degree. Indeed, for any \( A, v_1, \) and \( w_1, \) there exists a maximal subset of indices \( n_j, \)

\[ \{n_1, n_2, \ldots, n_J\} \subseteq \{1, 2, \ldots, L_*\}, \quad 1 = n_1 < n_2 < \cdots < n_J \leq L_*, \]

such that for each \( j = 1, \ldots, J, \) there exists a monic polynomial \( \Psi_{n_j-1} \) that is a regular formally orthogonal polynomial with respect to (3.7). As noted above, \( n_1 = 1 \) since \( \Psi_0(\lambda) = 1 \) is always a monic regular FOP of degree 0. In addition, for later use, we define \( n_0 = 1. \) We will call the vectors

\[ v_{n_j} = \Psi_{n_j-1}(A)v_1 \quad \text{and} \quad w_{n_j} = \Psi_{n_j-1}(A^T)v_1 \]

regular vectors, as they correspond to regular FOPs. We will then seek to extend the Lanczos algorithm to generate these vectors, skipping whenever necessary over breakdowns to reach the next regular vector. If a breakdown at a step \( n \) is "temporary", and there exists a regular FOP of degree higher than \( n, \) then the breakdown is called curable. If the breakdown is "permanent", in that there exists no regular FOP of higher degree and \( n_J < L_* \) then it is called incurable [79, 61]. In this case, it turns out that the Lanczos algorithm still generates some eigenvalue information, as was shown by Taylor in his Mismatch Theorem [79]. However, for linear systems the only solution is to restart the process with a different choice of starting vectors. We will discuss briefly the Mismatch Theorem in section 3.5, but otherwise we will not concern ourselves with incurable breakdowns, whose likelihood in practice is rather small anyway.

The existence of these breakdowns was recognized from the beginning [46, 83], and recurrence formulas for generating the sequence of regular FOPs were established long ago, though their application to the nonsymmetric Lanczos algorithm was studied only recently. The corresponding recurrences for the regular FOPs were established by Struble [77] for the special case of polynomials orthogonal on the real line with respect to an indefinite inner product where the weight function has changing sign. They are also a result of Magnus’ P-fractions [49, 50]. They were also mentioned by Gragg [32], by Draux [11], by Nuttall and Singh [56], as well as by Kung in his thesis [45] and by Gragg and Lindquist in the context of the partial realization problem [33]. While this body of work dealt only with exact breakdowns, Taylor [79] and Parlett, Taylor, and Liu [61] were the first to propose a
practical extension of the Lanczos algorithm that also dealt with near-breakdowns. They called such an extension a look-ahead Lanczos algorithm, a terminology which we will also adopt. Using the point of view of factoring the moment matrix, they characterized a near-breakdown by the condition \( w_n^T v_n \approx 0 \), with neither \( v_n \approx 0 \) nor \( w_n \approx 0 \). This would correspond to a near-zero pivot in the LU factorization of \( M^{(n)} \). We will review their work in the next section. Recently, there has been a revival of the Lanczos algorithm. In this context, we mention the work of Joubert [43, 44], who considered the Lanczos algorithm and Lanczos-based iterative methods, the work of Boley and his collaborators [4, 5], the recent paper by Parlett [59], and the work of Brezinski and his collaborators on Lanczos-based algorithms and their breakdowns [7, 8]. A complete treatment of the Lanczos method and its connections with orthogonal polynomials and Padé approximation was presented recently by Gutknecht [34, 35]. Our variant of the look-ahead Lanczos algorithm is based on Gutknecht’s work and, as we will see in section 3.6, differs fundamentally from the approach of Parlett, Taylor, and Liu. In particular, we will obtain a different characterization of a breakdown or a near-breakdown.

Before we start the discussion of the Parlett, Taylor, and Liu, and the Freund, Gutknecht, and Nachtigal approaches, it will serve to introduce some terminology and notation we will use in describing both. As indicated, the term look-ahead refers to an extension of the classical nonsymmetric Lanczos algorithm that tries to skip over a breakdown. The assumption underlying both approaches will be that the breakdown is curable and therefore a jump beyond the breakdown is indeed possible. There is, unfortunately, no easy way to check beforehand whether a breakdown is curable or not. A look-ahead algorithm will have to generate — by some means other than the standard Lanczos recurrence — vectors permitting it to span the gap to the next set of regular vectors \( v_{n_j+1} \) and \( w_{n_j+1} \). In analogy to the classical Lanczos algorithm, we will continue to refer to the vectors \( v_n \) and \( w_n \) as right and left Lanczos vectors, regardless of the way they are generated. All the vectors with indices between \( n_j \) and \( n_{j+1} - 1 \) are said to form the \( j \)th block. We will use the index \( j = 1, 2, \ldots \), as a counter for the blocks built by the look-ahead Lanczos algorithm, and we will use \( l = l(n) \) to denote the index of the current block, containing the last computed regular vectors \( v_n \), and \( w_n \). If \( n + 1 = n_{l+1} \), then the \( l \)th block is said to be complete. We denote by \( h_j = n_{j+1} - n_j \) the size of the \( j \)th block. Capital letters with subscript \( j \) will denote matrices containing quantities from or related to the \( j \)th block. For example,

\[
V_j = \begin{bmatrix}
  v_{n_j} & v_{n_j+1} & \cdots & v_{n_{j+1} - 1}
\end{bmatrix}
\]
is the $N \times h_j$ matrix whose columns are the right Lanczos vectors from a complete block $j$. As before, we will continue to use capital letters with superscript $(n)$ for matrices containing quantities from steps 1 through $n$. Thus, the basic form (3.4) of the algorithm remains the same, but now we have, instead of (3.5),

$$
(W(n))^T (V(n)) = D(n).
$$

Here

$$D(n) = \text{diag}(\delta_1, \ldots, \delta_l), \quad \delta_j = W_j^T V_j, \quad j = 1, \ldots, l,
$$

is a block diagonal matrix. The diagonal blocks corresponding to complete blocks will be nonsingular; if the $l$th block is complete, then $D(n)$ itself is also nonsingular. The matrix $H_{\epsilon}^{(n)}$ remains upper Hessenberg, but its exact structure will be dictated by the particular strategy used to generate the blocks in the look-ahead approach.

For reasons of stability, in practice one always computes scaled versions of the right and left Lanczos vectors, rather than the vectors $v_n$ and $w_n$ corresponding to monic polynomials. It has been pointed out by Rutishauser [64] that using the recurrences corresponding to the monic polynomials may lead to overflow or underflow. Furthermore, it has been shown by Taylor [79] that it is not enough to scale both $v_{n+1}$ and $w_{n+1}$ by the same scale factor. Hence, one always computes a pair of intermediate vectors $\tilde{v}_{n+1}$ and $\tilde{w}_{n+1}$, which is then scaled to obtain the final vectors $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$. We will use tilde ($\tilde{}$) to denote quantities related to the intermediate vectors, and hat ($\hat{}$) to denote quantities related to the final scaled vectors. Thus, $\hat{V}(n)$ would denote the matrix with the scaled right Lanczos vectors, and $H_{\epsilon}^{(n)}$ would denote the corresponding matrix of recurrence coefficients. We note that the two matrices of recurrence coefficients are no longer the same, due to the possibly different scalings used for $\hat{V}(n)$ and $\hat{V}(n)$. We will only use the $H_{\epsilon}^{(n)}$ from the recurrence for $\hat{V}(n)$, so we do not introduce any notation for the other matrix of scaled recurrence coefficients. Finally, we will use $\rho_n$ and $\xi_n$ to denote the scale factors,

$$
\tilde{v}_n = \rho_n \hat{v}_n, \quad \text{and} \quad \tilde{w}_n = \xi_n \hat{w}_n.
$$

40
Section 3.4. The Parlett, Taylor, and Liu Approach

We are now ready to review the look-ahead Lanczos algorithm proposed by Parlett, Taylor, and Liu in [61]. We will not present a detailed review of their algorithm, but rather extract the main ideas; the full details can be found in their paper [61]. We should also point out that, in their paper, Parlett, Taylor, and Liu use the Lanczos algorithm with the Hermitian inner product (2.35). We will restate their approach in terms of our formulation with the transpose inner product (2.34).

If the final Lanczos vectors \( \hat{v}_n \) and \( \hat{w}_n \) are required to have inner product 1, so that

\[
\left( \hat{V}^{(n)} \right)^T \left( \hat{V}^{(n)} \right) = I_n,
\]

then the Lanczos recurrences have a slightly different form:

\[
\tilde{v}_{n+1} = \rho_{n+1} \tilde{v}_{n+1} = A \hat{v}_n - \alpha_n \hat{v}_n - \xi_n \hat{v}_{n-1},
\]

\[
\tilde{w}_{n+1} = \xi_{n+1} \tilde{w}_{n+1} = A^T \hat{w}_n - \alpha_n \hat{w}_n - \rho_n \hat{w}_{n-1},
\]

where

\[
\alpha_n = \hat{w}_n^T A \hat{v}_n, \quad \text{and} \quad \tilde{w}_{n+1}^T \tilde{v}_{n+1} = \rho_{n+1} \xi_{n+1} = \theta_{n+1}.
\]

Thus, one first computes \( \tilde{v}_{n+1} \) and \( \tilde{w}_{n+1} \), then their inner product \( \theta_{n+1} \), and finally one chooses one of the scale factors, say \( \rho_{n+1} \), which then determines the other, \( \xi_{n+1} \). Parlett, Taylor, and Liu suggest using

\[
\rho_{n+1} = \sqrt{|\theta_{n+1}|}, \quad \text{and} \quad \xi_{n+1} = \theta_{n+1}/\rho_{n+1}.
\]

The procedure stops when \( \theta_{n+1} = 0 \), either because of a regular termination, or because of a breakdown.

As seen in section 3.2, a breakdown corresponds to a zero pivot in the \( L L^T \) factorization of the moment matrix \( M^{(n)} \). The standard remedy for a zero pivot is partial pivoting, where the pivot row is interchanged with the row below it having the largest entry (in magnitude) in the pivot column. However, in our case, the column of \( M^{(n)} \) is not available; in fact, no entries below the pivot row are available. Hence, pivoting is not a solution. Instead, the remedy used is to consider a more general pivot, a \( 2 \times 2 \) or larger submatrix. This approach was discussed by Parlett and Bunch in [60] and is the basis of the Parlett, Taylor, and Liu approach. In particular, they discuss its application to the case of \( 2 \times 2 \) pivots.

When a breakdown occurs, the algorithm has computed \( \tilde{v}_{n+1}, \tilde{w}_{n+1}, \) and \( \theta_{n+1} \), but due to \( \theta_{n+1} = 0 \), cannot scale the vectors to obtain \( \hat{v}_{n+1} \) and \( \hat{w}_{n+1} \). One then
computes a next pair of vectors, \( \tilde{v}_{n+2} \) and \( \tilde{w}_{n+2} \), using some recurrence other than the Lanczos recurrence, for example,

\[
\tilde{v}_{n+2} = A\tilde{v}_{n+1} - \tilde{\xi}_n \tilde{v}_n, \\
\tilde{w}_{n+2} = A^T \tilde{w}_{n+1} - \tilde{\xi}_n \tilde{w}_n,
\]

where \( \tilde{\xi}_n \) is chosen so that

\[
\left( V^{(n)} \right)^T \tilde{w}_{n+2} = \left( \tilde{W}^{(n)} \right)^T \tilde{v}_{n+2} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}.
\]

One then chooses \( \tilde{v}_{n+1} \) and \( \tilde{w}_{n+2} \) as any two different vectors in the plane of \( \tilde{v}_{n+1} \) and \( \tilde{w}_{n+2} \), other than multiples of \( \tilde{v}_{n+1} \), and one similarly chooses \( \tilde{w}_{n+1} \) and \( \tilde{w}_{n+2} \) from the plane of \( \tilde{w}_{n+1} \) and \( \tilde{w}_{n+2} \). The corresponding polynomials can be chosen to be different, but both of degree \( n + 1 \), which means that the degree of the Lanczos polynomials will jump from \( n - 1 \) to \( n + 1 \) at step \( n \), and then remain \( n + 1 \) at the next step. We remark that these polynomials are deficient FOPs, as they are orthogonal to polynomials of degree up to \( n - 1 \), but not to polynomials of degree \( n \).

To complete the algorithm, one only needs to specify how to choose the vectors \( \tilde{v}_{n+1} \) and \( \tilde{w}_{n+2} \). Letting

\[
M_1 M_2 = \begin{bmatrix} \tilde{w}_{n+1} & \tilde{w}_{n+2} \end{bmatrix}^T \begin{bmatrix} \tilde{v}_{n+1} & \tilde{v}_{n+2} \end{bmatrix}
\]

be a factorization of the \( 2 \times 2 \) diagonal block, we have

\[
I_2 = \begin{bmatrix} \tilde{w}_{n+1} & \tilde{w}_{n+2} \end{bmatrix}^T \begin{bmatrix} \tilde{v}_{n+1} & \tilde{v}_{n+2} \end{bmatrix} = M_1^{-1} \begin{bmatrix} \tilde{w}_{n+1} & \tilde{w}_{n+2} \end{bmatrix}^T \begin{bmatrix} \tilde{v}_{n+1} & \tilde{v}_{n+2} \end{bmatrix} M_2^{-1},
\]

so that we set

\[
\begin{bmatrix} \tilde{v}_{n+1} & \tilde{v}_{n+2} \end{bmatrix} = \begin{bmatrix} \tilde{v}_{n+1} & \tilde{v}_{n+2} \end{bmatrix} M_2^{-1}, \\
\begin{bmatrix} \tilde{w}_{n+1} & \tilde{w}_{n+2} \end{bmatrix} = \begin{bmatrix} \tilde{w}_{n+1} & \tilde{w}_{n+2} \end{bmatrix} M_1^{-T}.
\]

Different factorizations will mean different choices for the vectors \( \tilde{v}_{n+1}, \tilde{v}_{n+2}, \tilde{w}_{n+1}, \) and \( \tilde{w}_{n+2} \). Parlett, Taylor, and Liu review several options and select an LU decomposition with interchange, which corresponds to choosing \( M_1 \) and \( M_2 \) of the form

\[
M_1 = \begin{bmatrix} * & 1 \\ 1 & 0 \end{bmatrix}, \quad \text{and} \quad M_2 = \begin{bmatrix} * & * \\ 0 & * \end{bmatrix},
\]

where * denote possibly nonzero elements.
We still need to specify how to decide when to perform a look-ahead step. The discussion above covers the case of an exact breakdown; in practice, one also needs to cover the case of near-breakdowns, when the pivots in the $LL^T$ factorization above are just small, but not necessarily zero. Parlett, Taylor, and Liu suggest a procedure that also handles near-breakdowns. One first computes

$$\phi_1 = |\cos \angle (\hat{v}_{n+1}, \hat{v}_{n+2})|,$$

$$\phi_2 = \min \{ |\cos \angle (\hat{v}_{n+1}, \hat{w}_{n+1})|, |\cos \angle (\hat{v}_{n+2}, \hat{w}_{n+2})| \}.$$

It turns out that all the quantities involved can be computed at the cost of four extra inner products, but no matrix multiplications. If $\phi_1$ and $\phi_2$ are both small, then the algorithm stops — this would require a pivot larger than $2 \times 2$. Otherwise, given a bias factor, one performs a standard Lanczos step if $\phi_1 \geq \text{(bias factor)} \phi_2$, or a look-ahead step otherwise. Parlett, Taylor, and Liu use a bias factor of 2.

To summarize, the Parlett, Taylor, and Liu approach is based on the view of Lanczos algorithm as a factorization of the moment matrix $M^{(n)}$. They view a breakdown or a near-breakdown as encountering a zero or near-zero pivot in the $LL^T$ factorization without pivoting, and their remedy is based on extending the notion of pivot to submatrices of size $2 \times 2$ or larger, though they only present the implementation of the $2 \times 2$ case. In this case, the Parlett, Taylor, and Liu implementation requires four additional inner products per step, beyond the two normally required by the classical Lanczos algorithm. As we will see in section 3.6, the Freund, Gutknecht, and Nachtigal approach is based on a different view of Lanczos algorithm, and will obtain a different characterization of a breakdown or a near-breakdown. Furthermore, it will not require any additional inner products per step.
Section 3.5. The Mismatch Theorem

In this section, we briefly discuss Taylor’s Mismatch Theorem [79], which addresses
the incurable breakdowns in the Lanczos process. We present a summary of the
discussion about the Mismatch Theorem from section 4 of Gutknecht’s paper [34].
Recall that an incurable breakdown is the case when \( L < L_* \), so that the Lanczos
process terminates before finding any invariant subspaces. In the case of linear
systems, the only recourse is to restart the procedure with different starting vec-
tors. However, in the case of eigenvalue computations, it turns out that one still
obtains some eigenvalue information. Even though the Lanczos process does not
find an invariant subspace, the last regular FOP \( \Psi_{n_j-1} \) is the minimal polynomial
of an invariant subspace of \( A \), and its roots are both the eigenvalues of \( H^{(n_j)} \) and
eigenvalues of \( A \).

**Theorem.** (Mismatch Theorem) *If an incurable breakdown occurs, so that
\( n_j < L_* \), then the regular FOP \( \Psi_{n_j-1} \) is the characteristic polynomial of \( H^{(n_j)} \)
and the minimal polynomial of an invariant subspace of \( A \).*

We will not present the proof of the theorem, we will just discuss some of the
results needed for the proof and we will indicate where the name of the theorem
comes from. All the details missing here can be found in Taylor’s thesis [79] and
Gutknecht’s paper [34].

Consider the power series

\[
F(\lambda) = \sum_{j=0}^{\infty} \frac{\mu_j}{\lambda^{j+1}},
\]

where, as before,

\[
\mu_j = \langle 1, \lambda^j \rangle_L = w_1^T A^j v_1
\]

are the moments associated with the inner product (3.7). The entire discussion in
this section will center around this power series. Since \( \|A\|_2 < \infty \),

\[
|\mu_j| = |w_1^T A^j v_1| \leq \|v_1\|_2 \|w_1\|_2 \|A\|_2^j,
\]

hence the power series (3.16) converges and is analytic at least for \( |\lambda| > \|A\|_2 \),
including \( \infty \). Now

\[
\frac{1}{\lambda} \left( I + \frac{1}{\lambda} A + \frac{1}{\lambda^2} A^2 + \cdots \right) = (\lambda I - A)^{-1}, \quad \text{for } |\lambda| > \|A\|_2,
\]

44
so that

\[ F(\lambda) = \sum_{j=0}^{\infty} \frac{\mu_j}{\lambda^{j+1}} = w_1^T \frac{1}{\lambda} \left( \sum_{j=0}^{\infty} \frac{A^j}{\lambda^j} \right) v_1 \]

\[ = w_1^T (\lambda I - A)^{-1} v_1 \]

is the first moment of the resolvent \((\lambda I - A)^{-1}\). Note that the resolvent exists and is bounded for \(|\lambda| > \|A\|_2\), and vanishes (like \(1/\lambda\)) at \(\infty\). Hence, writing \(F(\lambda)\) as a rational function

\[ F(\lambda) = \frac{q(\lambda)}{p(\lambda)}, \]

we have \(\text{deg}(q) \leq \text{deg}(p) - 1\). We denote by

\[ \nu_F = \text{deg}(p) \]

the order of \(F\).

Suppose now that \(A\) is diagonalizable, so that \(A = UDU^{-1} = UZ^T\), where \(Z = U^{-T} = [z_1 \cdots z_N]\) is the matrix of eigenvectors of \(AT\). For simplicity, suppose that \(A\) has only simple eigenvalues. The arguments below can be extended to the case of an arbitrary \(A\); see [34] for full details. Let

\[ v_1 = \sum_{j=1}^{N} \xi_j u_j = U\xi, \]

\[ w_1 = \sum_{k=1}^{N} \eta_k z_k = Z\eta, \]

be the decompositions of \(v_1\) and \(w_1\) in eigenvectors of \(A\) and \(AT\), respectively. Together with (3.17) and (3.18), we have

\[ F(\lambda) = \frac{q(\lambda)}{p(\lambda)} = w_1^T (\lambda I - A)^{-1} v_1 \]

\[ = \eta^T Z^T (\lambda UU^{-1} - UDU^{-1})^{-1} U\xi \]

\[ = \eta^T U^{-1} U(\lambda I - D)^{-1} U^{-1} U\xi \]

\[ = \sum_{j=1}^{N} \frac{\xi_j \eta_j}{\lambda - \lambda_j}. \quad (3.19) \]

All the eigenvalue information that the Lanczos process has is contained in \(F(\lambda)\). In particular, the roots of \(p(\lambda)\) are eigenvalues of \(A\). The degree of \(p(\lambda)\) is determined by the number of nonzero terms in the sum in (3.19), which is determined by the
number of nonzero products $\eta_j \xi_j$. If the starting vectors $v_1$ and $w_1$ contain enough matching index components, then $\nu_F = \deg(p) = n_j = L_*$, the Lanczos process terminates regularly, and any breakdowns are curable. However, if the number of matching index components is less than $L_*$, then the Lanczos process cannot find an invariant subspace and terminates in less than $L_*$ steps with an incurable breakdown. Nevertheless, the roots of $p(\lambda)$ are still eigenvalues of $A$, so one does recover some eigenvalues information, but no eigenspace information.

We illustrate the discussion above with two simple examples. Suppose $A$ is a $10 \times 10$ matrix with simple eigenvalues $\lambda_1, \ldots, \lambda_{10}$, and let $u_1, \ldots, u_{10}$ and $z_1, \ldots, z_{10}$ be the corresponding right and left eigenvectors.

Suppose first that

$$v_1 = \xi_1 u_1 + \xi_2 u_2 + \xi_3 u_3, \quad \xi_j \neq 0,$$
$$w_1 = \eta_1 z_1 + \eta_2 z_2 + \eta_3 z_3 + \eta_4 z_4, \quad \eta_j \neq 0.$$ 

Then $L_r = 3$, $L_l = 4$, and $L_* = 3$. The sum in (3.19) has three nonzero terms, corresponding to the common indices $\{1, 2, 3\}$, so $\nu_F = \deg(p) = 3 = L_*$. Hence, the Lanczos process will terminate regularly after finding an $A$-invariant subspace. If $\eta_4 = 0$, then the Lanczos process would find both invariant subspaces. Note that we can still have curable breakdowns. For example,

$$w_1^T v_1 = \xi_1 \eta_1 + \xi_2 \eta_2 + \xi_3 \eta_3,$$

so that by choosing the coefficients appropriately, one can obtain $w_1^T v_1 = 0$.

Suppose now that

$$v_1 = \xi_1 u_1 + \xi_2 u_2 + \xi_3 u_3, \quad \xi_j \neq 0,$$
$$w_1 = \eta_2 z_2 + \eta_3 z_3 + \eta_4 z_4, \quad \eta_j \neq 0.$$ 

Then $L_r = L_l = L_* = 3$. However, the sum in (3.19) has only two nonzero terms, corresponding to the common indices $\{2, 3\}$, so $\nu_F = \deg(p) = 2 < L_*$. Hence, the Lanczos process will terminate with an incurable breakdown, without finding any invariant subspaces. Nevertheless, it will find some of the eigenvalues of $A$, as the two roots of $p(\lambda)$ are $\lambda_2$ and $\lambda_3$. The problem here is that too many of the eigencomponents of $v_1$ and $w_1$ are mismatched; this mismatch led to the name of the theorem.

We summarize the termination properties of the look-ahead Lanczos algorithm in the following:
Proposition. There is a termination index $L$ such that the look-ahead Lanczos algorithm will either terminate regularly in step $L$, with $v_{L+1} = 0$ or $w_{L+1} = 0$, or will encounter an incurable breakdown starting at the regular vectors $v_L$ and $w_L$. In either case,

$$
\Lambda(H^{(L)}) \subseteq \Lambda(A).
$$

(3.20)
Section 3.6. The Freund, Gutzkecht, and Nachtigal Approach

We now turn to the look-ahead Lanczos approach proposed by Freund, Gutzkecht, and Nachtigal in [21]. We will summarize the main ideas, but will not cover exhaustively the implementation details. A complete discussion, together with listings of FORTRAN codes, can be found in [21].

Whereas the Parlett, Taylor, and Liu approach is based on the view of Lanczos as an $LL^T$ factorization of the moment matrix $M^{(n)}$, our approach is based on the view of Lanczos as building as sequence of regular formally orthogonal polynomials. The process proceeds smoothly until a breakdown occurs, at which point we will take the view that we need to fill the gap between the last regular FOP and the regular FOP of next higher degree, assuming one exists. In this section, we will discuss the results for the case of exact breakdowns, as presented by Gutzkecht in [34]. In the next section, we extend these results to the near-breakdown case, then in section 3.8 we will discuss the criteria used to decide when to build blocks, and finally in section 3.9 we will present a few implementation details.

Recall from section 3.2 that a breakdown occurs at step $n + 1$ when at step $n = n_i$, the regular FOP $\Psi_{n_i-1}$ has zero length, as reflected in $w_n^Tv_{n_i} = 0$. We assume that there exists a next regular FOP $\Psi_{n_{i+1}-1}$ and we wish to compute it. The approach used is to generate a block of vectors spanning the gap to the next regular FOP. Thus, every block starts with a regular vector and ends before the next regular vector. All vectors in the block, other than the starting regular vector, correspond to singular and deficient FOPs, and will be called inner vectors. In case of a block of size 1, there are no inner vectors, just the regular vectors; this case corresponds to the classical Lanczos algorithm. Otherwise, for blocks of size greater than 1, the inner vectors will be present. It is shown in [34, 11] that the regular FOPs $\Psi_{n_{j-1}-1}$, $\Psi_{n_{j}-1}$, and $\Psi_{n_{j+1}-1}$ satisfy a three-term recurrence of the form

$$\Psi_{n_{j+1}-1}(z) = B_j(z)\Psi_{n_{j}-1}(z) - b_j\Psi_{n_{j-1}-1}(z),$$  \hspace{1cm} (3.21)

where $B_j(z)$ is a monic polynomial of degree $h_j$, and $b_j$ is a scalar. Gutzkecht also shows in [34] how $B_j$ and $b_j$ can be computed. The point for us is that, in order to construct $\Psi_{n_{j+1}-1}$, we need to generate $B_j$ of appropriate degree. If we knew beforehand the size $h_j$ of the block, we could just use a polynomial of exactly that degree. However, as we do not know the size of the gap, we will successively increase the degree of $B_j$ and then test whether we can build a regular FOP with the new polynomial. One can generate the inner vectors using arbitrary recurrences of the
form

$$v_{n+1} = Av_n - \sum_{j=n_l}^{n} h_{jn} v_j,$$

$$w_{n+1} = A^T w_n - \sum_{j=n_l}^{n} \tilde{h}_{jn} w_j. \tag{3.22}$$

It is shown in [33, 11, 34] that, given the last regular vector $v_{n_l}$, the inner vectors $w_j$ in block $l$ satisfy

$$w_j^T v_{n_l} = 0, \quad j = n_l + 1, \ldots, n_{l+1} - 2. \tag{3.23}$$

Thus, the algorithm builds regular vectors until a pair of biorthogonal regular vectors $v_{n_l}$ and $w_{n_l}$ is encountered. One then builds inner vectors $v_{n+1}^{n_l}$ and $w_{n+1}^{n_l}$ from (3.22) until a pair is obtained with $w_{n+1}^{n_l} v_{n_l} \neq 0$, at which point the block closes and the next regular pair $v_{n+1}^{n_l}$ and $w_{n+1}^{n_l}$ can be computed. Gutknecht shows in [34] that the vector form of the recurrence (3.21) for the regular vectors is

$$v_{n_{l+1}} = Av_{n_{l+1}} - V_l \alpha_l - b_l v_{n_l},$$

$$w_{n_{l+1}} = A^T w_{n_{l+1}} - W_l \alpha_l - b_l w_{n_l}. \tag{3.24}$$

This process clearly preserves the matrix form of the look-ahead Lanczos algorithm,

$$AV^{(n)} = V^{(n+1)} H^{(n)}_e,$$

$$A^T W^{(n)} = W^{(n+1)} \tilde{H}^{(n)}_e. \tag{3.25}$$

From (3.22) and (3.24), $H^{(n)}$ is an $n \times n$ block tridiagonal unit upper Hessenberg matrix of the form

$$H^{(n)} = \begin{bmatrix}
\alpha_1 & \beta_2 & & \\
\gamma_2 & \ddots & \ddots & \\
& \ddots & \ddots & \beta_l \\
& & \gamma_l & \alpha_l
\end{bmatrix}, \tag{3.26}$$

where $\alpha_j$ are in general full unit upper Hessenberg matrices of size $h_j \times h_j$, and $\beta_j$ and $\gamma_j$ are rank-1 matrices of size $h_{j-1} \times h_j$ and $h_j \times h_{j-1}$, respectively,

$$\beta_j = \begin{bmatrix}
0 & \cdots & 0 & b_j \\
\vdots & \ddots & \ddots & \ddots \\
\vdots & \ddots & \ddots & \ddots \\
0 & \cdots & \cdots & 0
\end{bmatrix}, \quad \gamma_j = \begin{bmatrix}
0 & \cdots & 0 & 1 \\
\vdots & \ddots & \ddots & \ddots \\
\vdots & \ddots & \ddots & \ddots \\
0 & \cdots & \cdots & 0
\end{bmatrix}. \tag{3.27}$$
$\tilde{H}^{(n)}_t$ has a similar structure. The sizes given are correct only for complete blocks $j$. For the current block $l$, which may be incomplete, $\alpha_l$ has size $\tilde{h}_l \times \tilde{h}_l$, $\beta_l$ has size $h_{l-1} \times \tilde{h}_l$, and $\gamma_l$ has size $\tilde{h}_l \times h_{l-1}$, where $\tilde{h}_l = n + 1 - n_l$.

When the inner vectors are obtained from three-term recurrences, so that (3.22) reduces to

$$v_{n+1} = Av_n - \zeta_n v_n - \eta_n v_{n-1},$$
$$w_{n+1} = A^Tw_n - \zeta_n w_n - \eta_n w_{n-1},$$

then $H^{(n)}_t$ simplifies: $\alpha_j$ become comrade matrices (a term used by Barnett [3]) of size $h_j \times h_j$,

$$\alpha_j = \begin{bmatrix} \zeta_{nj} & \eta_{nj+1} & * \\ 1 & \zeta_{nj+1} & \ddots \\ & \ddots & \ddots & \ddots \\ & & 1 & \eta_{nj+1-2} & * \\ & & & \ddots & \zeta_{nj+1-2} & * \\ & & & & 1 & * \end{bmatrix}.$$  \hspace{1cm} (3.29)

The structure of $H^{(n)}$ was first given (without proof) by Gragg in [32], then later by Draux in [11] and Gragg and Lindquist in [33]. Gutknecht calls a matrix $H^{(n)}$ with the structure (3.27–3.29) a Gragg matrix [34]. The extension we will propose for the near-breakdown case will preserve the structure (3.26–3.27) of $H^{(n)}$, allowing fill-in only in the blocks $\beta_j$.

From (3.23), the inner vectors in the case of exact breakdowns are biorthogonal to all vectors in previous blocks, so that the biorthogonality matrix $D^{(n)}$ has block diagonal structure

$$(W^{(n)})^T V^{(n)} = D^{(n)} = \text{diag}(\delta_1, \ldots, \delta_l), \quad \delta_j = W_j^T V_j.$$  \hspace{1cm} (3.30)

Gutknecht shows in [34] that for a complete block $j$, the diagonal block $\delta_j$ has the form

$$\delta_j = W_j^T V_j = \begin{bmatrix} 0 & \cdots & 0 & d_1 \\ \vdots & \ddots & \ddots & * \\ 0 & \cdots & \cdots & \cdots \\ d_1 & \cdots & \cdots & \cdots \end{bmatrix}, \quad d_1 \neq 0,$$  \hspace{1cm} (3.31)

which corresponds to (3.23), and is clearly nonsingular. He also shows that if $B_j(z) = z^{h_j}$, then $\delta_j$ is Hankel. As the form of $\delta_j$ indicates, the polynomials in the block are as follows:

- the first polynomial is always a regular FOP, orthogonal to all polynomials of lower degree;
the last \( [h_j/2] \) polynomials are deficient FOPs, orthogonal to only some of the polynomials of lower degree;

- any in-between polynomials are singular FOPs, orthogonal to all polynomials of lower degree, but non-unique.

We end this section with a few properties that we will need in the next section and in section 3.9. We will show that

\[
\delta_i^{-1} W_i^T A v_n = \delta_i^{-T} V_i^T A^T w_n, \\
\delta_{i-1}^{-1} W_{i-1}^T A v_n = \delta_{i-1}^{-T} V_{i-1}^T A^T w_n.
\]

(3.32)

Consider first \( \delta_j \). Let \( w_i \) and \( v_k \) be two vectors from block \( j \). From (3.6) and (3.7) and using the fact that polynomials in \( A \) commute, we obtain

\[
w_i^T v_k = w_i^T \Psi_{i-1}(A) \Psi_{k-1}(A) v_1 \\
= w_i^T \Psi_{k-1}(A) \Psi_{i-1}(A) v_1 = w_k^T v_i,
\]

(3.33)

which shows that \( \delta_j \) is symmetric. Let now \( w_i \) and \( v_j \) be two arbitrary Lanczos vectors. Using an argument similar to the one above, and the fact that \( w_i^T A v_j = (w_i^T A v_j)^T \), we get

\[
w_i^T A v_j = w_i^T \Psi_{i-1}(A) A \Psi_{j-1}(A) v_1 = w_i^T \Psi_{j-1}(A) A \Psi_{i-1}(A) v_1 \\
= v_i^T \Psi_{i-1}(A^T) A^T \Psi_{j-1}(A^T) w_1 = v_i^T A^T w_j,
\]

which proves that

\[
W_i^T A v_n = V_i^T A^T w_n, \\
W_{i-1}^T A v_n = V_{i-1}^T A^T w_n.
\]

(3.34)

Combining (3.33) and (3.34), we get (3.32).
Section 3.7. The Near-Breakdown Case

The discussion in the last section was concerned only with exact breakdowns. One would like to extend the look-ahead Lanczos algorithm to handle near-breakdowns as well. Our approach is to modify the goal of the look-ahead Lanczos procedure: rather than aim to compute the complete set of regular FOPs, we will aim to compute only a subset of the set of regular FOPs, namely those regular FOPs that are well-defined in some sense. As we have seen, consecutive regular FOPs are linked by the three-term recurrence (3.21), or in vector form, the recurrence (3.24). However, both recurrences require the complete set of regular FOPs, and some of these may cause numerical problems if computed as regular vectors. As we will see, it is possible to modify the recurrence formulas to compute only a selected subset of the set of regular FOPs, chosen according to criteria that we will discuss in section 3.8. We will sometimes refer to these well-defined FOPs as the computed regular FOPs. We will treat all remaining ill-defined regular FOPs as if they were singular or deficient FOPs, and we will replace them with inner polynomials that in general will be a combination of regular, singular, and deficient polynomials. Rather than introduce new notation, we will continue to use our current notation, with the understanding that from now on the indices $n_j$ refer only to the well-defined regular FOPs and need not cover the complete set of regular FOPs.

The practical look-ahead Lanczos algorithm will build blocks not only when $w_{n_i}^T v_{n_i} = 0$, but possibly at other times as well. For example, a criterion for starting to build a block might be

$$|w_{n_i}^T v_{n_i}| \leq tol,$$

for some tolerance $tol > 0$; the exact criteria for building blocks will be discussed in the next section. The point is that one needs to extend the recurrences (3.22) and (3.24) to handle the case when the inner polynomials are no longer just singular or deficient polynomials. On the other hand, the polynomials computed from the extended version of (3.24) should still be regular FOPs. One would also like to preserve the matrix form (3.25) of the look-ahead Lanczos algorithm, as well as the block diagonal structure (3.30) of the matrix $D^{(n)}$. Finally, one would like to maintain as much as possible the three-term recurrences encountered so far. We will propose an extension of the look-ahead Lanczos algorithm which satisfies all of these requirements, extending the three term recurrences (3.22) and (3.24) to three term recurrences that involve the current block and the last block.

It turns out that it suffices to modify slightly the recurrences (3.22) and (3.24) to obtain the desired recurrences for the computed regular vectors and the inner
vectors. We first show how regular FOPs can be computed with an extended version of (3.24), then we will show how the inner vectors can be computed with an extended version of (3.22). The main results for the sequential version of the look-ahead Lanczos algorithm for near-breakdowns are the following two propositions:

**Proposition.** Suppose that at step \( n \), the Lanczos vectors obey a block biorthogonality relation of the form (3.30). Also, suppose that \( \delta_l \) is nonsingular. Then the vectors \( v_{n+1} \) and \( w_{n+1} \) computed from

\[
\begin{align*}
v_{n+1} &= A v_n - V_l \delta_l^{-1} W_l^T A v_n - V_{l-1} \delta_{l-1}^{-1} W_{l-1}^T A v_n, \\
w_{n+1} &= A^T w_n - W_l \delta_l^{-T} V_l^T A^T w_n - W_{l-1} \delta_{l-1}^{-T} V_{l-1}^T A^T w_n,
\end{align*}
\]  

(3.35)

belong to the set of regular vectors.

**Proof.** To show that vectors obtained from (3.35) correspond to regular FOPs, we need to show that they are biorthogonal to all previous vectors. Clearly, \( v_1 \) and \( w_1 \) are always regular vectors. Consider now \( v_{n+1} \). Multiplying its equation by \( W_i^T \), \( i \leq l \), we have

\[
W_i^T v_{n+1} = W_i^T A v_n - W_i^T V_l \delta_l^{-1} W_l^T A v_n - W_i^T V_{l-1} \delta_{l-1}^{-1} W_{l-1}^T A v_n.
\]

For \( i < l - 1 \), all the terms are zero by the hypothesis of the proposition. For \( i = l - 1 \), the middle term vanishes by the hypothesis, and the other two terms cancel each other. For \( i = l \), the last term vanishes by the hypothesis, and the other two terms cancel each other. In all cases,

\[
W_i^T v_{n+1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}
\]

for \( i \leq l \), so \( v_{n+1} \) is biorthogonal to all previous vectors \( w_j \). A similar argument shows that \( w_{n+1} \) is biorthogonal to all previous vectors \( v_j \). Thus, \( v_{n+1} \) and \( w_{n+1} \) are regular vectors, which means that in fact, the current block \( l \) is complete, and \( n = n_{l+1} - 1 \). \( \Box \)

In the case of exact breakdowns, the inner vectors are automatically biorthogonal to all previous blocks. For near-breakdowns this is not the case, and the biorthogonality must be enforced. We have our second proposition:

**Proposition.** Suppose that at step \( n \), the Lanczos vectors obey a block biorthogonality relation of the form (3.30). Then the vectors \( v_{n+1} \) and \( w_{n+1} \) computed from

\[
\begin{align*}
v_{n+1} &= A v_n - V_l \alpha_l - V_{l-1} \delta_{l-1}^{-1} W_{l-1}^T A v_n, \\
w_{n+1} &= A^T w_n - W_l \tilde{\alpha}_l - W_{l-1} \delta_{l-1}^{-T} V_{l-1}^T A^T w_n,
\end{align*}
\]  

(3.36)

\[53\]
with $\alpha_l$ and $\tilde{\alpha}_l$ some arbitrary inner recurrences, are inner vectors biorthogonal to all vectors in previous blocks.

Proof. By the previous proposition, the first vector in each block is a regular vector, biorthogonal to all previous vectors. Suppose that all other inner vectors from the current block, $v_{n+1}, \ldots, v_n$, are biorthogonal to all vectors in previous blocks. Consider $v_{n+1}$. Multiplying its equation by $W_i^T$, $i \leq l - 1$, we have

$$W_i^T v_{n+1} = W_i^T A v_n - W_i^T V_i \alpha_i - W_i^T V_{l-1} \delta_{l-1}^{-1} W_{l-1}^T A v_n.$$ 

The second term always vanishes by the induction hypothesis. For $i < l - 1$, both remaining terms are zero by the hypothesis of the proposition. For $i = l - 1$, the two remaining terms cancel each other. In all cases,

$$W_i^T v_{n+1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{for } i < l,$n

so $v_{n+1}$ is biorthogonal to all previous blocks. A similar argument shows that $w_{n+1}$ is biorthogonal to all previous blocks. Thus, $v_{n+1}$ and $w_{n+1}$ are inner vectors biorthogonal to all vectors in previous blocks. \[\square\]

The recurrences (3.35) and (3.36) clearly maintain the matrix form (3.25) of the look-ahead Lanczos algorithm. Furthermore, by (3.32), the biorthogonalization coefficients of $V_i$ and $W_i$, and of $V_{l-1}$ and $W_{l-1}$, are in fact the same, so that if $\alpha_l = \tilde{\alpha}_l$ in (3.36), then $\tilde{H}_i^{(n)} = H_i^{(n)}$ in (3.25). The most important point is that $H^{(n)}$ retains the block tridiagonal structure (3.26), which means that all the vectors in the look-ahead Lanczos algorithm are still computed with three-term recurrences, but which may involve blocks of vectors. The blocks $\alpha_j$ and $\gamma_j$ have the same form as in (3.26). The only change is that the blocks $\beta_j$ in (3.27) are now in general full matrices, whose columns are the coefficients $\delta_{l-1}^{-1} W_{l-1}^T A v_n$. They can be shown to still be of rank 1. The biorthogonality matrix $D^{(n)}$ also retains its block diagonal structure (3.30); however, the diagonal blocks $\delta_j$ no longer have the form (3.31), they can now have nonzero elements anywhere.

There are two ways of generating inner vectors that obey (3.36). The first approach is to first apply the inner recurrences $\alpha_l$ and $\tilde{\alpha}_l$, and then immediately biorthogonalize the new vectors against the last block. It is easy to see that this is equivalent to sequentially applying (3.36), hence we will refer to this approach as sequential. There is also another approach for generating the inner vectors, and it differs from (3.36) in the manner in which the inner vectors are biorthogonalized.
One can first build the entire block according to some inner recurrences, and then biorthogonalize it all at once when it is complete. Such an approach has the advantage that all inner products required by the biorthogonalization step are computed at once, which may be more suitable for a parallel architecture. On the other hand, as the block is biorthogonalized only at the end, this approach may involve in the biorthogonalization step more than just the current and the last blocks. We will only discuss the sequential version of the algorithm; for more details on the other approach, see [23].

One option for the inner recurrences $\alpha_l$ and $\tilde{\alpha}_l$ in (3.36) is to orthogonalize in the Euclidean sense the inner vectors against the other vectors in their block. This gives for the inner recurrence coefficients in (3.36)

$$\alpha_l = (V_l^H V_l)^{-1} V_l^H A v_n \quad \text{and} \quad \tilde{\alpha}_l = (W_l^H W_l)^{-1} W_l^H A^T w_n.$$ 

For our version of the sequential algorithm, the look-ahead blocks built are generally small, and as a result, in numerical experiments Euclidean orthogonalization did not lead to improved numerical properties. Thus, in view of the additional inner products required, we do not recommend orthogonalizing inside the blocks. In fact, we will simplify the recurrences in (3.36) by assuming that the inner recurrences are three-term recurrences as in (3.28), so that (3.36) reduces to

$$v_{n+1} = A v_n - \zeta_n v_n - \eta_n v_{n-1} - V_{l-1} \delta_{l-1}^{-1} W_{l-1}^T A v_n,$$
$$w_{n+1} = A^T w_n - \zeta_n w_n - \eta_n w_{n-1} - W_{l-1} \delta_{l-1}^{-1} W_{l-1}^T A v_n,$$

(3.37)

where $\eta_{n_j} = 0$. One may choose the recurrence coefficients $\zeta_n$ and $\eta_n$ so that they remain the same from one block to the next, and change only with respect to their index $n - n_l$ inside the block, or one may choose these coefficients so that they change completely from one block to the next. For instance, one choice for the inner polynomials in (3.37) are suitably scaled and translated Chebyshev polynomials, so that the $\zeta_n$ and $\eta_n$ are generated by the Chebyshev iteration [51]. In this case, the translation parameters could be adjusted from one step to the next using spectral information obtained from previous Lanczos steps. However, in our experience, the algorithm only builds a few small blocks, so the choice of the recurrences in (3.37) is not overly important. We have always used $\zeta_n = 1$ and, for $n \neq n_j$, $\eta_n = 1$.

We can now give a sketch of the sequential version of the look-ahead Lanczos algorithm. As mentioned before, in practice one computes scaled vectors $\tilde{v}_n$ and $\tilde{w}_n$, which we normalize to have unit length, as suggested by Parlett in [59]. As before, tilde ($\tilde{\cdot}$) denotes intermediate quantities.
LAL Algorithm. Sketch of the sequential look-ahead Lanczos algorithm.

(1) Choose \( \hat{v}_1 \) and \( \hat{w}_1 \in \mathbb{C}^N \), with \( \| \hat{v}_1 \|_2 = \| \hat{w}_1 \|_2 = 1 \), and set \( l = 1, n_1 = 1, \hat{V}_1 = \hat{v}_1, \hat{W}_1 = \hat{w}_1, \hat{\delta}_1 = \hat{W}_1^T \hat{V}_1, \hat{V}_0 = \hat{W}_0 = \emptyset, \rho_1 = \xi_1 = 1 \).

For \( n = 1, 2, \ldots \):

(2) Decide whether to construct \( \hat{v}_{n+1} \) and \( \hat{w}_{n+1} \) as regular or inner vectors, and go to step (3) or (4) accordingly.

(3) (Regular step) Compute regular vectors from (3.35)

\[
\begin{align*}
\tilde{v}_{n+1} &= A \hat{v}_n - \hat{V}_l \hat{\delta}_{l-1}^{-1} \hat{W}_l^T A \hat{v}_n - \hat{V}_{l-1} \hat{\delta}_{l-1}^{-1} \hat{W}_{l-1}^T A \hat{v}_n, \\
\tilde{w}_{n+1} &= A^T \hat{w}_n - \hat{W}_l \hat{\delta}_{l-1}^{-T} \hat{V}_l^T A^T \hat{w}_n - \hat{W}_{l-1} \hat{\delta}_{l-1}^{-T} \hat{V}_{l-1}^T A^T \hat{w}_n. 
\end{align*}
\]  

(3.38)

Set \( l = l + 1, n_l = n + 1, \hat{V}_l = \hat{W}_l = \emptyset \), and go to (5).

(4) (Inner step) Compute inner vectors from (3.37)

\[
\begin{align*}
\tilde{v}_{n+1} &= A \hat{\delta}_n - \zeta_n \hat{v}_n - (\eta_n/\rho_n) \hat{v}_{n-1} - \hat{V}_{l-1} \hat{\delta}_{l-1}^{-1} \hat{W}_{l-1}^T A \hat{v}_n, \\
\tilde{w}_{n+1} &= A^T \hat{w}_n - \zeta_n \hat{w}_n - (\eta_n/\xi_n) \hat{w}_{n-1} - \hat{W}_{l-1} \hat{\delta}_{l-1}^{-T} \hat{V}_{l-1}^T A^T \hat{w}_n. 
\end{align*}
\]  

(3.39)

(5) Compute

\[
\rho_{n+1} = \| \tilde{v}_{n+1} \|_2, \quad \text{and} \quad \xi_{n+1} = \| \tilde{w}_{n+1} \|_2.
\]

If \( \rho_{n+1} = 0 \) or \( \xi_{n+1} = 0 \), then stop; otherwise, set

\[
\begin{align*}
\hat{v}_{n+1} &= \tilde{v}_{n+1}/\rho_{n+1}, & \hat{w}_{n+1} &= \tilde{w}_{n+1}/\xi_{n+1}, \\
\hat{V}_l &= [\hat{V}_l \hat{v}_{n+1}], & \hat{W}_l &= [\hat{W}_l \hat{w}_{n+1}], & \hat{\delta}_l &= \hat{W}_l^T \hat{V}_l. 
\end{align*}
\]  

(3.40)

(6) Extract the nonzero elements of the \( n \)th column of \( \hat{H}^{(n)} \) from the recurrence for \( \hat{v}_{n+1} \) in (3.38) or (3.39), scaled appropriately by \( \rho_{n+1} \). Also, set \( \hat{h}_{n+1,n} = \rho_{n+1} \).

Step (1) is the initialization step. For linear systems, \( \hat{v}_1 \) is usually chosen to be \( r_0/\| r_0 \|_2 \), and \( \hat{w}_1 \) is usually chosen to be \( \hat{v}_1 \). Step (2) is the subject of the next section. Steps (3) and (4) are the scaled versions of (3.35) and (3.37), respectively. Step (3) is the standard Lanczos step, computing regular vectors. Step (4) is the inner step, computing inner vectors. Note that the scaled biorthogonality coefficients are no longer the same for \( \hat{V}_l \) and \( \hat{W}_l \), or for \( \hat{V}_{l-1} \) and \( \hat{W}_{l-1} \). Steps (5) and (6) are update steps, which scale the new Lanczos vectors and update the Lanczos blocks and the matrix \( \hat{H}^{(n)}_x \). If \( \hat{v}_{n+1} \) and \( \hat{w}_{n+1} \) are inner vectors, then the current block \( l \) is incomplete and its size increases by 1. If \( \hat{v}_{n+1} \) and \( \hat{w}_{n+1} \) are regular vectors, then the \( l \)th block is complete and a new block, the \( (l+1) \)st, is started with \( \hat{v}_{n+1} \) and \( \hat{w}_{n+1} \) as its first vectors.

56
Section 3.8. Building Blocks

The sketch of the algorithm \textbf{LAL} is complete, except for step (2), which determines whether \( \hat{v}_{n+1} \) and \( \hat{w}_{n+1} \) are built as regular or as inner vectors. In this section, we discuss the criteria used to make this decision. We will aim for three goals. The first goal is to find a criterion or a set of criteria that would enable us to recover the classical Lanczos algorithm in the absence of look-ahead steps. As we will see, given our setup, this is easily accomplished. Secondly, we would like to build blocks that are as small as possible. In terms of algorithm \textbf{LAL}, this means that in step (2) we will close a look-ahead block as soon as we can; other strategies might be possible. Finally, we would like to handle near-breakdowns in a satisfactory manner, especially in the presence of roundoff. We first discuss the criterion suggested by the theory. We will then show that this criterion is not sufficient for finite precision arithmetic, and we will propose additional criteria to remedy its drawbacks.

We should point out that all the results obtained this far, even the ones for the near-breakdown case, have implicitly assumed exact arithmetic. In particular, this assumption was made every time biorthogonality was invoked. In finite precision arithmetic, the biorthogonality of the left and right Lanczos vectors is often lost; eventually, the vectors computed from the three-term recurrences (3.3.5) and (3.3.6) lose biorthogonality with respect to the vectors in blocks further back. As a consequence, the behavior of the algorithm in finite precision arithmetic need not follow the theoretical predictions. For example, the finite termination property \( L \leq N \) is often lost in practice. Another example is that, even though we showed that (3.3.5) computes regular vectors, the proof relies on biorthogonality, and in the presence of roundoff, the vectors computed from (3.3.5) can no longer be guaranteed to be regular vectors.

From (3.3.8), \( \hat{v}_{n+1} \) and \( \hat{w}_{n+1} \) can be built as regular vectors provided that \( \hat{\delta}_l \) is nonsingular. This is the only criterion needed in exact arithmetic. One checks whether \( \hat{\delta}_l \) is singular, for example, by checking its smallest singular value \( \sigma_{\min}(\hat{\delta}_l) \), and performs an inner step if \( \sigma_{\min}(\hat{\delta}_l) = 0 \). It is therefore natural to extend this check to finite precision arithmetic and perform an inner step if \( \sigma_{\min}(\hat{\delta}_l) < \text{tol} \), for some suitably chosen tolerance \( \text{tol} \). For example, Parlett suggests in [59] using \( \text{tol} = \epsilon^{1/3} \) or \( \text{tol} = \epsilon^{1/4} \), where \( \epsilon \) denotes the unit roundoff. In this case, the complete blocks of computed Lanczos vectors would satisfy

\[
\sigma_{\min}(\hat{\delta}_l) \geq \text{tol}, \quad k = 1, \ldots, l-1. \tag{3.41}
\]

It can be shown [61, 59] that enforcing (3.41) would also ensure the linear indepen-
dence of the Lanczos basis vectors. We will show this for \( \hat{V}^{(n)} \); a similar argument applies for \( \hat{W}^{(n)} \). The derivation below is taken from [59]; the proof in [61] is incorrect, as was pointed out by Marlis Hochbruck [38].

From the singular value decomposition of \( \hat{V}^{(n)} \), we know that there exist unit vectors \( u \in \mathbb{C}^N \) and \( z \in \mathbb{C}^n \) such that

\[
\hat{V}^{(n)} z = u \sigma_{\text{min}}(\hat{V}^{(n)}).
\]

Consider now \( \hat{D}^{(n)} z \), for \( n = n_{l+1} - 1 \).

\[
\left\| \hat{D}^{(n)} z \right\|_2 = \sum_{j=1}^l \left\| \delta_j z_j \right\|_2^2 \geq \sum_{j=1}^l \left( \sigma_{\text{min}}(\delta_j) \right)^2 \left\| z_j \right\|_2^2 \\
\geq \left( \min_{1 \leq j \leq l} \left( \sigma_{\text{min}}(\delta_j) \right) \right)^2 \sum_{j=1}^l \left\| z_j \right\|_2^2 \\
= \left( \min_{1 \leq j \leq l} \left( \sigma_{\text{min}}(\delta_j) \right) \right)^2,
\]

where \( z_j \) are the elements of \( z \) corresponding for the \( j \)th Lanczos block. From the above,

\[
\min_{1 \leq j \leq l} \left( \sigma_{\text{min}}(\delta_j) \right) \leq \left\| \hat{D}^{(n)} z \right\|_2 = \left\| \left( \hat{W}^{(n)} \right)^T (\hat{V}^{(n)}) z \right\|_2 \\
\leq \left\| \left( \hat{W}^{(n)} \right)^T \right\|_2 \left\| (\hat{V}^{(n)}) z \right\|_2 = \left\| \left( \hat{W}^{(n)} \right)^T \right\|_2 \left\| u \sigma_{\text{min}}(\hat{V}^{(n)}) \right\|_2 \\
= \sigma_{\text{min}}(\hat{V}^{(n)}) \left\| \left( \hat{W}^{(n)} \right)^T \right\|_2.
\]

Now in the LAL algorithm \( \| w_j \|_2 = 1 \), so

\[
\left\| \left( \hat{W}^{(n)} \right)^T \right\|_2 \leq \sqrt{n},
\]

hence, together with (3.41), we have for the LAL algorithm

\[
\left( \sigma_{\text{min}}(\hat{V}^{(n)}) \right)^{-1} \leq \sqrt{n} \frac{1}{\text{tol}}.
\]

Using a similar argument for \( \hat{W}^{(n)} \), we finally obtain

\[
\sigma_{\text{min}}(\hat{V}^{(n)}) \geq \frac{\text{tol}}{\sqrt{n}}, \quad n = n_j - 1, \quad k = 1, \ldots, \tag{3.42}
\]

\[
\sigma_{\text{min}}(\hat{W}^{(n)}) \geq \frac{\text{tol}}{\sqrt{n}}.
\]

58
which is Theorem 10.1 in Parlett's paper [59]. As \( \sigma_{\min}(\hat{V}^{(n)}) \) and \( \sigma_{\min}(\hat{W}^{(n)}) \) are a measure of the linear independence of the Lanczos vectors, (3.41) would thus ensure linear independence of the Lanczos basis. However, as pointed out by Parlett, Taylor, and Liu, (3.42) relies on biorthogonality for its result, and can fail in finite precision arithmetic when biorthogonality is lost. We illustrate this with an example.

**Example 1.** In Figure 1, we plot \( \sigma_{\min}(\hat{\delta}_l) \) (the dots), \( \min_{1 \leq j \leq n} \left( \sigma_{\min}(\hat{\delta}_j) \right) \) (the solid line), and \( \sqrt{n} \sigma_{\min}(\hat{V}^{(n)}) \) (the dotted line), as functions of the iteration number \( n = 1, 2, \ldots \), for a \( 50 \times 50 \) dense matrix whose elements are taken from a normal distribution with mean 0.0 and variance 1.0. The elements of the Lanczos starting vectors \( v_1 \) and \( w_1 \) were also chosen from this normal distribution, and the vectors were then scaled to have unit length. (3.41) and (3.42) predict that

\[
\sqrt{n} \sigma_{\min}(\hat{V}^{(n)}) \geq \min_{1 \leq j \leq l} \left( \sigma_{\min}(\hat{\delta}_j) \right),
\]

which clearly is not the case. After less than 20 steps, the inequality reverses.

**Figure 1.** Breakdown of (3.42); curves should not cross.
Another way to view the failure of (3.41) is that we are trying to determine when the pivots in the $LL^T$ factorization of the moment matrix $M^{(n)}$ are small (nearly singular, in the case of block pivots). The point is that once biorthogonality is lost due to finite precision arithmetic, $D^{(n)}$ is no longer block diagonal; there is fill-in elsewhere in the matrix. Thus, the pivots are no longer simply the diagonal blocks of $D^{(n)}$, as (3.15) would indicate. In any case, as the example shows, (3.41) alone does not ensure the linear independence of the computed Lanczos vectors. In fact, if the look-ahead strategy is based on (3.41) alone, the algorithm may generate within a block inner vectors that are nearly linearly dependent. If this happens, then the check (3.41) fails in all subsequent iterations and the algorithm never closes the current block: it has generated an artificial breakdown.

In addition, numerical experience suggests yet another problem with (3.41): the behavior of the algorithm is very sensitive to the value used for $tol$, for values reasonably larger than the unit roundoff $\varepsilon$. We also illustrate this with an example.

**Example 2.** We applied the Lanczos algorithm to a nonsymmetric matrix obtained from the discretization of the 3-dimensional partial differential equation

$$Lu = f \quad \text{on} \quad (0,1) \times (0,1) \times (0,1),$$

where

$$Lu = -\frac{\partial}{\partial x} \left( e^{xy} \frac{\partial u}{\partial x} \right) - \frac{\partial}{\partial y} \left( e^{xy} \frac{\partial u}{\partial y} \right) - \frac{\partial}{\partial z} \left( e^{xy} \frac{\partial u}{\partial z} \right)$$

$$+ \beta (x+y+z) \frac{\partial u}{\partial x} + \left( \gamma + \frac{1}{1+x+y+z} \right) u,$$

with Dirichlet boundary conditions $u = 0$. The operator was discretized using centered differences on a uniform $15 \times 15 \times 15$ grid with mesh size $h = 1/16$, with $\beta = 30$ and $\gamma = -250$. This yields a $3375 \times 3375$ sparse nonsymmetric matrix $A$ with 22275 nonzero elements. The right-hand side $f$ was chosen such that

$$u = (1-x)(1-y)(1-z)(1-e^{-x})(1-e^{-y})(1-e^{-z})$$

is the exact solution of the linear system. The Lanczos process was started with $v_1 = w_1 = b$, where $b$ is the discretized right-hand side. The example was run on a Cray-2 at the NASA Ames Research Center; for this machine, $\varepsilon \approx 1.3E-29$. We tested two values of the tolerance $tol$, $\varepsilon^{1/4} \approx 6.0E-08$ and $\varepsilon^{1/3} \approx 2.3E-10$. In Figure 2, we plot $\sigma_{\min} (\delta_t)$ as a function of the iteration number, the dotted line for $\varepsilon^{1/4}$, and the solid line for $\varepsilon^{1/3}$. In the first case, the algorithm builds a block which it never closes, and the singular values become smaller and smaller. Yet if
tol is slightly smaller, as in the second case, the algorithm runs without problems until the end, in this case solving the linear system to the desired accuracy. This indicates that the block built in the first case was not a true breakdown, but rather an artificial incurable breakdown.

![Graph](image)

**Figure 2.** Sensitivity to the singular values criterion.

These phenomena have also been observed by Brezinski [6] in his numerical experiments. It all suggests that, in the presence of roundoff, (3.41) alone is not sufficient to decide whether to build regular or inner vectors. Instead, we propose to relax (3.41) so that it merely ensures that \( \hat{\delta}_l \) is numerically nonsingular, and to add two checks to ensure that the computed Lanczos vectors remain sufficiently linearly independent. Thus, instead of (3.41), our first check will just be

\[
\sigma_{\text{min}}(\hat{\delta}_l) \geq \epsilon, \tag{3.43}
\]

where \( \epsilon \) is the unit roundoff. Experiments have suggested that the look-ahead Lanczos algorithm starts to generate almost linearly dependent Lanczos vectors once a regular vector \( \hat{v}_{n+1} \) is built whose component \( A\hat{v}_n \in K_{n+1}(v_1, A) \) is dominated by the components in the previous Krylov space \( K_n(v_1, A) \), and similarly for \( \hat{w}_{n+1} \).
Thus, we will aim to avoid building such regular vectors, by checking the size of the components in the new Krylov space, compared to the size of the components in the old Krylov space. We will perform an inner step if at least one of the following is true:

\[
\|A\hat{v}_n\|_2 \leq \|\hat{V}_l\hat{\delta}_l^{-1}\hat{W}_l^TA\hat{v}_n\|_2, \\
\|A\hat{w}_n\|_2 \leq \|\hat{V}_{l-1}\hat{\delta}_{l-1}^{-1}\hat{W}_{l-1}^TA\hat{w}_n\|_2, \\
\|A^T\hat{w}_n\|_2 \leq \|\hat{W}_l\hat{\delta}_l^{-T}\hat{V}_l^TA^T\hat{w}_n\|_2, \\
\|A^T\hat{w}_n\|_2 \leq \|\hat{W}_{l-1}\hat{\delta}_{l-1}^{-T}\hat{V}_{l-1}^TA^T\hat{w}_n\|_2.
\]

(3.44)

Our approach to building blocks — or at least the main idea behind it — will be to combine (3.43) and (3.44). In practice, (3.44) has several drawbacks that will lead us to a similar, but cheaper and more flexible set of checks, discussed in the next section. However, the basic approach will remain the same: we will check for numerical singularity of \(\hat{\delta}_l\), and we will attempt to avoid building regular vectors dominated by the components in the old Krylov space \(K_n(v_1,A)\).
Section 3.9. Implementation Details

With the discussion in section 3.8 of the criteria for building blocks, the sketch of the look-ahead Lanczos algorithm LAL is now complete. In this section, we will discuss a few aspects of its implementation. As indicated at the beginning of section 3.8, we wanted an algorithm that would reduce to the standard Lanczos algorithm in the absence of look-ahead steps, that would build look-ahead blocks that are as small as possible, and that would handle near-breakdowns satisfactorily. In addition, from a practical point of view, we will attempt to minimize the number of inner products per step. The classical Lanczos algorithm requires only two inner products per step, if we neglect the additional two inner products needed in practice to scale the Lanczos vectors. We will propose an implementation of the look-ahead Lanczos algorithm that also requires only two inner products per step, regardless of whether a step is an inner step or a regular step.

For a regular step, one needs to compute \( \hat{\delta}_i, \hat{W}_i^T A \hat{\nu}_n, \hat{W}_{i-1}^T A \hat{\nu}_n, \hat{V}_i^T A^T \hat{\omega}_n, \) and \( \hat{V}_{i-1}^T A^T \hat{\omega}_n \) in (3.38). For an inner step, one needs to compute \( \hat{W}_{i-1}^T A \hat{\nu}_n \) and \( \hat{V}_{i-1}^T A^T \hat{\omega}_n \) in (3.39), and to update \( \hat{\delta}_i \) in (3.40). We will show that for a block of size \( h_i \), only 2\( h_i \) inner products are required: 2\( h_i \) – 1 inner products to compute \( \hat{\delta}_i \), and one inner product to compute \( \hat{W}_i^T A \hat{\nu}_n \) and \( \hat{V}_i^T A^T \hat{\omega}_n \) in (3.38). We will obtain the \( \hat{W}_{i-1}^T A \hat{\nu}_n \) and \( \hat{V}_{i-1}^T A^T \hat{\omega}_n \) terms without performing any inner products. To simplify the derivations, we will use the vectors \( v_n \) and \( w_n \) corresponding to the monic Lanczos polynomials. The hat (\( \hat{\cdot} \)) quantities are then obtained simply by scaling appropriately.

Let us first consider \( \delta_i \). We know from (3.33) that \( \delta_i \) is symmetric, hence we only need to compute its upper triangle, including the diagonal. We will now show how to compute \( \delta_i \) with only 2\( h_i \) – 1 inner products. We will see that once the diagonal and the first superdiagonal of \( \delta_i \) have been computed by inner products, the remaining elements of the upper triangle can be computed by recurrences. Let \( v_j \) and \( w_i \) be two vectors from the current block. We assume that the diagonal and the first superdiagonal have been computed already, hence \( i = n_i, \ldots, n_{i+1} - 3 \), and \( j = i + 2, \ldots, n_{i+1} - 1 \). Assuming the inner recurrences (3.36), we have

\[
ww^Tv_j = w^T(Av_{j-1} - \zeta_{j-1}v_{j-1} - \eta_{j-1}v_{j-2})
= (A^Tw_i)^Tv_{j-1} - \zeta_{j-1}w^Tv_{j-1} - \eta_{j-1}w^Tv_{j-2}
= (w_{i+1} + \zeta_iw_i + \eta_iw_{i-1})^Tv_{j-1} - \zeta_{j-1}w^Tv_{j-1} - \eta_{j-1}w^Tv_{j-2}
= w^Tv_{j-1} + (\zeta_i - \zeta_{j-1})w^Tv_{j-1} + \eta_iw^Tv_{j-1} - \eta_{j-1}w^Tv_{j-2}.
\]

As can be seen from the indices, \( w_i^Tv_j \) depends only on elements of \( \delta_i \) from the
previous two columns, and hence can be computed without any inner products. We remark that the recurrences (3.36) are enforced numerically, so computing $w_i^T v_j$ from (3.45) should give the same result — up to roundoff — as computing the inner product directly, and we have indeed noticed this in experiments. The structure of a typical $\delta_l$ for a look-ahead block $l$ is then

$$
\delta_l = \begin{bmatrix}
\bullet & \bullet & + & + & + \\
\bullet & \bullet & + & + \\
\bullet & \bullet & \bullet & \bullet & + \\
\bullet & \bullet & \bullet & \bullet & \bullet \\
\end{bmatrix},
$$

where $\bullet$ denotes elements computed directly by inner products, $+$ denotes elements computed from (3.45), and $\star$ denotes elements obtained by symmetry.

It remains to compute the $W_l^T A v_n$, $W_{l-1}^T A v_n$, $V_l^T A^T w_n$, and $V_{l-1}^T A^T w_n$ terms. We know from (3.34) that

$$
W_l^T A v_n = V_l^T A^T w_n,
$$

$$
W_{l-1}^T A v_n = V_{l-1}^T A^T w_n,
$$

so that the coefficients $\delta_l^{-1} V_l^T A^T w_n$ and $\delta_{l-1}^{-1} V_{l-1}^T A^T w_n$ in (3.38) and (3.39) can be obtained directly from $\delta_l^{-1} V_l^T A \delta_n$ and $\delta_{l-1}^{-1} V_{l-1}^T A \delta_n$, without any additional inner products. We will show how to obtain $W_{l-1}^T A v_n$ with no additional inner products, and how to obtain $W_l^T A v_n$ with only one additional inner product. Consider $w_i^T A v_n$, for $w_i$ a vector from either the current block $l$ or the previous block $l-1$. We have

$$
w_i^T A v_n = (A^T w_i)^T v_n
$$

$$
= (w_{i+1} + \zeta_i w_i + \eta_i w_{i-1})^T v_n
$$

$$
= w_{i+1}^T v_n + \zeta_i w_i^T v_n + \eta_i w_{i-1}^T v_n. \quad (3.46)
$$

Recall that for the previous block, we have

$$
W_{l-1}^T v_n = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}.
$$

If now $w_i$ is the last vector in the previous block, then $i = n_l - 1$, and (3.46) reduces to

$$
w_{n_l-1}^T A v_n = w_{n_l}^T v_n,
$$

which is the $(n - n_l + 1)$st element of the first row of $\delta_l$ and is computed as part of $\delta_l$. If $w_i$ is any other vector from the previous block, then $i < n_l - 1$, and we get

$$
w_i^T A v_n = 0.
$$

64
Thus, $W_{l-1}^T A v_n$ requires no inner products. If $w_i$ is any vector from the current block, other than the last one, then $n_l \leq i \leq n_{l+1} - 1$, and all of the terms needed in (3.46) are already available from $\delta_l$. Finally, if $w_i$ is the last vector in the current block, then $i = n_{l+1} - 1$, and we do not have $w_{n_{l+1}}^T A v_n$; we need to compute this term directly, thus requiring the last inner product. We have

$$W_{l-1}^T A v_n = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ + \end{bmatrix}, \quad \text{and} \quad W_l^T A v_n = \begin{bmatrix} + \\ \vdots \\ + \\ \bullet \end{bmatrix},$$

where $\bullet$ denotes the element computed directly by an inner product, and $+$ denotes elements obtained from $\delta_l$ and (3.46).

Let us now address the practical implementation of the checks (3.43) and (3.44). The check (3.43) requires either computing the singular value decomposition of $\hat{\delta}_l$, or solving the corresponding linear system in (3.38) and checking for numerically zero pivots. As for (3.44), it has two main drawbacks: it is too expensive and too strict. As written, it would require six additional inner products per step. Additionally, one would like to allow a slack factor before the terms on the right-hand sides, since a difference of just a few orders of magnitude could be permissible. However, such an approach introduces the problem of estimating the slack factor. Finally, in practice there is also a problem related to the limited storage available to the user. Theoretically, it is possible to construct examples where the look-ahead Lanczos algorithm will build look-ahead blocks of arbitrary sizes $h_j$. In practice, there is a maximal block size, determined by limits on the storage available. Thus, on one hand we have the drawbacks of the check (3.44), and on the other hand we have at runtime the issue of a maximal block size. To solve these problems, as well as enable the algorithm to proceed as far as possible, we propose the following procedure. First, we will replace the check (3.44), which is expensive, by a cheaper version. Since $\|\hat{\delta}_j\|_2 = \|\hat{w}_j\|_2 = 1$, we have

$$\|V_l \delta_l^{-1} W_l^T A v_n\|_2 = \left\| \sum_{j=n_l}^{n_l} \left( \delta_l^{-1} W_l^T A v_n \right)_j \hat{v}_j \right\|_2$$

$$\leq \sum_{j=n_l}^{n_l} \left\| \left( \delta_l^{-1} W_l^T A v_n \right)_j \hat{v}_j \right\|_2$$

$$= \sum_{j=n_l}^{n_l} \left| \left( \delta_l^{-1} W_l^T A v_n \right)_j \right|,$$
and similarly for the other checks in (3.44). This would replace (3.44) by the checks

\[
\|A\hat{v}_n\|_2 \leq \sum_{j=n_{l_1}}^{n} \left| \left( \delta_{l}^{-1} \hat{W}_{l}^T A\hat{v}_n \right)_j \right|, \\
\|A\hat{v}_n\|_2 \leq \sum_{j=n_{l_1}}^{n_{l-1}} \left| \left( \delta_{l-1}^{-1} \hat{W}_{l-1}^T A\hat{v}_n \right)_j \right|, \\
\|A^T\hat{w}_n\|_2 \leq \sum_{j=n_{l_1}}^{n} \left| \left( \delta_{l}^{-T} \hat{V}_{l}^T A^T \hat{w}_n \right)_j \right|, \\
\|A^T\hat{w}_n\|_2 \leq \sum_{j=n_{l_1}}^{n_{l-1}} \left| \left( \delta_{l-1}^{-T} \hat{V}_{l-1}^T A^T \hat{w}_n \right)_j \right|. 
\] (3.47)

This is already a significant improvement over (3.44), as we are left with only two additional inner products. However, one would still have the problem of estimating a slack factor. Thus, the second step is to replace in (3.47) the terms on the left-hand side with a generic factor \(n(A)\), which would be related to the norm of \(A\) and which would incorporate the slack factor. The final form of the second set of checks used in step (2) of the LAL algorithm is then

\[
n(A) \leq \sum_{j=n_{l_1}}^{n} \left| \left( \delta_{l}^{-1} \hat{W}_{l}^T A\hat{v}_n \right)_j \right|, \\
n(A) \leq \sum_{j=n_{l_1}}^{n_{l-1}} \left| \left( \delta_{l-1}^{-1} \hat{W}_{l-1}^T A\hat{v}_n \right)_j \right|, \\
n(A) \leq \sum_{j=n_{l_1}}^{n} \left| \left( \delta_{l}^{-T} \hat{V}_{l}^T A^T \hat{w}_n \right)_j \right|, \\
n(A) \leq \sum_{j=n_{l_1}}^{n_{l-1}} \left| \left( \delta_{l-1}^{-T} \hat{V}_{l-1}^T A^T \hat{w}_n \right)_j \right|. 
\] (3.48)

We have eliminated all additional inner products. To obtain \(n(A)\), we will use a dynamic procedure that will also enable us to cope with the problem of limited storage. Suppose we are given an initial value for \(n(A)\), either an estimate from the user (for example, \(n(A)\) from a previous run with the same matrix \(A\) under similar conditions), or by setting

\[
n(A) = \max (\|A\hat{v}_1\|_2, \|A^T\hat{w}_1\|_2). 
\] (3.49)

We then update \(n(A)\) dynamically as follows. In each block, whenever an inner step is performed because of (3.48), we keep track of the size of the terms that have
caused one or more of the inequalities in (3.48) to be true. If the block closes, then this information is no longer needed and it is discarded. However, if the algorithm is about to run out of storage, then \( n(A) \) is replaced by the smallest value that has caused an inequality in (3.48) to be true. This value is always larger than the old value of \( n(A) \), hence this procedure monotonically increases \( n(A) \). After updating \( n(A) \), when the block is rebuilt, the new value of \( n(A) \) is guaranteed to pass all the checks in (3.48) at least once, and so the block will close at that point. This will also release the storage that was used by the previous block, ensuring that the algorithm can proceed.

To summarize, we have shown in this section that it is possible to implement the sequential version of the look-ahead Lanczos algorithm with as many inner products per step as the classical Lanczos algorithm. We have also proposed a procedure that implements the checks for step (2) of the algorithm without requiring any additional inner products, if we neglect the inner products that might be required once for (3.49). This procedure also conveniently allowed us to deal with the practical issue of a maximal block size.
Section 3.10. Numerical Examples (Eigenvalues)

We conclude this chapter on the Lanczos algorithm with a few numerical examples. We will use the Freund, Gutknecht, and Nachtigal implementation to compute the Lanczos vectors, and we will then use the matrix $\hat{H}^{(n)}$ of recurrence coefficients to obtain approximations to the eigenvalues of $A$. As noted already, the eigenvalues of $\hat{H}^{(n)}$ are Ritz values for $A$, and can be used as approximations to some of the eigenvalues of $A$. In practice, however, the process is complicated by the problem of spurious and ghost eigenvalues.

Spurious eigenvalues are eigenvalues of $\hat{H}^{(n)}$ that do not correspond to any of the eigenvalues of $A$, while ghost eigenvalues are eigenvalues of $\hat{H}^{(n)}$ that repeatedly approximate an eigenvalue of $A$. As Lanczos pointed out in [46], the Lanczos process does not reveal anything about the multiplicity of the eigenvalues of $A$; it merely finds invariant subspaces. In particular, all of the eigenvalues of $\hat{H}^{(n)}$ should have unit geometric multiplicity [46, 34]. Thus, ghost eigenvalues are just an artifact of finite precision arithmetic, as are spurious eigenvalues. To identify the spurious eigenvalues from among the eigenvalues of $\hat{H}^{(n)}$, we have used the heuristic proposed by Cullum and Willoughby in [10]. The procedure was originally proposed for the tridiagonal matrices generated by the classical Lanczos algorithm, but we have found it to also work satisfactorily for the block tridiagonal matrices generated by the look-ahead Lanczos algorithm. The idea is to compare the eigenvalues of the full block tridiagonal Lanczos matrix $\hat{H}^{(n)}$ with the eigenvalues of the $(n - 1) \times (n - 1)$ submatrix obtained by deleting the first row and column of $\hat{H}^{(n)}$. One first computes all the eigenvalues of $\hat{H}^{(n)}$, and retains as genuine all eigenvalues appearing more than once; this would indicate an eigenvalue of $A$ and its ghosts. From the remaining simple eigenvalues, only the ones that do not appear among the eigenvalues of the submatrix are retained; all the ones that also appear in the spectrum of the submatrix are discarded as spurious. The motivation behind this procedure comes from the Hermitian case, where there are no breakdowns and $\hat{H}^{(n)}$ is always Hermitian tridiagonal. In this case, the eigenvalues of the submatrix should interlace the eigenvalues of the full matrix, and hence any eigenvalues in common must be spurious. In the Hermitian case, one can make a rigorous argument [10], which unfortunately does not carry over to the non-Hermitian case. Nevertheless, the heuristic seems to be effective in practice, though we have not done any extensive testing of the procedure, nor have we addressed some of the issues that come up in combining the procedure with our look-ahead Lanczos variant. For example, one question is whether to delete only the first row and the first column, or to delete a
number of rows and columns equal to the size of the first block. In our examples, the first block was always a $1 \times 1$ block, so we only deleted the first row and column.

**Examples 3 and 4.** Both of these examples are eigenvalue problems taken from the Cullum and Willoughby paper [10]. Consider the 2-D differential operator

$$Lu = -\frac{\partial}{\partial x} \left( e^{-\alpha y} \frac{\partial u}{\partial x} \right) - \frac{\partial}{\partial y} \left( e^{\alpha y} \frac{\partial u}{\partial y} \right)$$

$$+ \beta (x + y) \frac{\partial u}{\partial x} + \beta \frac{\partial}{\partial x} \left( (x + y) u \right)$$

$$+ \gamma (x + y) \frac{\partial u}{\partial y} + \gamma \frac{\partial}{\partial y} \left( (x + y) u \right) + \frac{1}{1 + x + y} u$$

on the unit square $(0,1) \times (0,1)$, with $\beta \in \mathbb{R}$, $\gamma \in \mathbb{R}$. The operator was discretized using centered differences on a uniform $m \times m$ grid with mesh size $h = 1/(m + 1)$, leading to a sparse nonsymmetric matrix of order $N = m^2$.

![Graphs showing the evolution of the Ritz values with increasing steps](image)

**Figure 3.** Ritz values for Example 3.

In Example 3, we use $m = 30$ ($N = 900$), $\beta = 20$, and $\gamma = 0$, and we plot in Figure 3 the Ritz values (denoted by "o") generated by the Lanczos process after $n = 40, 80,$
160, and 320 steps. The eigenvalues of \( \hat{H}^{(n)} \) were obtained using EISPACK. After 40 steps, the complex conjugate pair of Ritz values with maximal real part has converged to eigenvalues of \( A \). After 80 steps, 12 Ritz values (all on the right edge of the spectrum) have converged, while after 160 steps, 30 Ritz values — 24 on the right edge and 6 on the left edge of the spectrum — have converged to eigenvalues of \( A \).

![Graph of Ritz values](image)

**Figure 4.** Ritz values for Example 4.

In Example 4, we use \( m = 63 \) (\( N = 3969 \)), \( \beta = 0 \), and \( \gamma = 250 \), and we plot in Figure 4 the Ritz values generated by the Lanczos process after \( n = 40, 80, 160, \) and 320 steps. In both examples, we again used random starting vectors for \( v_1 \) and \( w_1 \).
Chapter 4.
The Quasi-Minimal Residual Algorithm

Section 4.1. The Basic QMR Approach

With the discussion of the Lanczos algorithm and of the Freund, Gutknecht, and Nachtigal look-ahead variant of the algorithm, we are now ready to discuss the quasi-minimal residual approach for solving linear systems. We will first briefly motivate and give a history of the approach, then recall from the discussion in the previous chapters the elements that are needed for the QMR algorithm. We will present the basic quasi-minimal residual method in some generality, and then discuss some of the details of its implementation and some of its properties when combined with the sequential look-ahead Lanczos variant presented in Chapter 3. We will end our discussion with a few numerical examples of the resulting QMR algorithm applied to non-Hermitian linear systems.

The Lanczos process, when considered alone, can be viewed as a basic tool: it builds basis vectors for two Krylov spaces, and nothing more. However, as discussed already, it does it very efficiently, with little work and storage. One takes advantage of this by combining the Lanczos process with another method that can use the basis vectors. One possibility is to use the matrix $H^{(n)}$ to obtain eigenvalue estimates for $A$, as was done at the end of Chapter 3. Another possibility is to use the Lanczos basis $V^{(n)}$ to solve linear systems. This will be done in this chapter. A key element in the motivation of the QMR approach is the efficiency of the Lanczos process: having obtained the Lanczos basis cheaply, we will not be willing to expend much additional effort to solve the linear system. As we will see, this will mean that we will have to settle for a solution that only has a quasi-optimal property, rather than a true minimization property.

The quasi-minimal residual method was first proposed by Roland Freund for the case of complex symmetric linear systems [20]. In fact, this was also the motivation for choosing the Lanczos inner product (2.34) with the transpose, rather than the inner product (2.35) with the conjugate transpose, when we developed our look-ahead Lanczos algorithm. This way, the algorithm simplifies considerably when $A = A^T$, with work and storage being roughly halved. We should point out that situations where $A$ is complex symmetric are not as rare as it might seem. Problems from electromagnetics or involving the Helmholtz, Schrödinger, and Maxwell’s
equations lead naturally to complex symmetric systems, as the interaction between
two points is symmetric but generally described by complex quantities. In addition,
these problems often have complex boundary conditions. Such problems were the
original motivation for the QMR method, though the algorithm we are going to
describe applies to arbitrary matrices.

Recall from section 2.1 that the idea behind Krylov space methods is to con-
struct at step $n$ an iterate $x_n$ from a shifted Krylov space

$$x_n \in x_0 + K_n(r_0, A), \quad n = 1, 2, \ldots,$$

starting from an initial guess $x_0 \in \mathbb{C}^N$ and the initial residual vector $r_0 = b - Ax_0$.
To build a basis for $K_n(r_0, A)$, we use a look-ahead Lanczos algorithm, started with

$$\beta v_1 = r_0, \quad \beta \in \mathbb{R}, \quad \beta \neq 0,$$

and $w_1 \in \mathbb{C}^N$ arbitrary and nonzero. For now, we do not specify which variant of the
look-ahead algorithm is used, we just assume that the underlying Lanczos algorithm
can build the basis for $K_n(r_0, A)$. Namely, the Lanczos algorithm will construct
the matrix $V^{(n)}$ of basis vectors spanning $K_n(r_0, A)$ and the upper Hessenberg
matrix $H^{(n)}_e$ such that

$$AV^{(n)} = V^{(n+1)}H^{(n)}_e,$$

as in (2.11). For the iterate $x_n$, we have

$$x_n = x_0 + V^{(n)}z_n, \quad z_n \in \mathbb{C}^n,$$

for some coefficient vector $z_n$, so that

$$r_n = b - Ax_n = V^{(n+1)} \left( d^{(n+1)} - H^{(n)}_e z_n \right),$$

with

$$d^{(n+1)} = \begin{bmatrix} \beta & 0 & \cdots & 0 \end{bmatrix}^T \in \mathbb{R}^{n+1}$$

as in (2.15). Let now

$$\Omega^{(n)} = \text{diag} (\omega_1, \ldots, \omega_{n+1}), \quad \omega_j > 0 \text{ for all } j,$$

be an arbitrary $(n + 1) \times (n + 1)$ scaling matrix that we introduce in the expression
for $r_n$ to get

$$r_n = V^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \left( \Omega^{(n)} \right) \left( d^{(n+1)} - H^{(n)}_e z_n \right)$$

$$= V^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \left( \omega_1 d^{(n+1)} - \Omega^{(n)} H^{(n)}_e z_n \right). \quad (4.1)$$
The \( \omega_j \)'s are free parameters that can be used to modify the scaling of the problem. Recall now that the columns of \( V^{(n+1)} \) do not form an orthogonal basis, as was the case for the Arnoldi process. Hence, \( V^{(n+1)} (\Omega^{(n)})^{-1} \) is not unitary regardless of the scaling \( \Omega^{(n)} \), and so imposing a least-squares condition on the residual \( r_n \) leads to a full \( N \times (n+1) \) least-squares problem

\[
\min_{z_n} \|b - Ax_n\|_2 = \min_{z_n} \left\| V^{(n+1)} (\Omega^{(n)})^{-1} \left( \omega_1 d^{(n+1)} - \Omega^{(n)} H_{e}^{(n)} z_n \right) \right\|_2 , 
\]

which is a restatement of (2.30). To solve this problem requires computing the QR decomposition of \( V^{(n+1)} (\Omega^{(n)})^{-1} \), which would demand \( O(Nn^2) \) work and \( O(Nn) \) storage. We would then obtain an algorithm mathematically equivalent to GMRES, but senseless in practice: we have computed \( V^{(n+1)} \) with short recurrences, only to give it all up in order to obtain the optimal \( z_n \). So instead of solving (4.2), we will solve a smaller problem, namely minimizing just the Euclidean norm of the coefficient vector in (4.2). We will obtain \( z_n \) from the minimization problem

\[
\min_{z_n} \left\| \omega_1 d^{(n+1)} - \Omega^{(n)} H_{e}^{(n)} z_n \right\|_2 , 
\]

which is only an \( (n+1) \times n \) least-squares problem. This is the main point of the quasi-minimal residual approach: we have obtained \( V^{(n+1)} \) cheaply, so we are not willing to spend the work and storage required by the full problem (4.2), instead we just solve (4.3). The fact that we still obtain \( z_n \) from a minimization problem will be an invaluable tool for proving some of the properties of the QMR algorithm. In particular, we will be able to describe quantitatively the departure from true optimality caused by dropping the \( V^{(n+1)} (\Omega^{(n)})^{-1} \) terms. We should also mention that the QMR algorithm is equivalent to the MINRES algorithm of Paige and Saunders [57] when \( A = A^H \) (in this case, one sets \( \Omega^{(n)} = I_{n+1} \)).

From (4.3), it is easy to see that the QMR method has the finite termination property: it finds the exact solution \( x = A^{-1} b \) whenever the Lanczos algorithm terminates with an \( A\)-invariant subspace, so that \( L = L_1 = L_* \) and \( v_{L+1} = 0 \). In this case, the matrix \( H_{e}^{(n)} \) is actually a square matrix, as \( h_{L+1,L} = 0 \). Hence, the least-squares problem (4.3) reduces to a linear system with coefficient matrix \( \Omega^{L-1} H^{(L)} \). Since \( A \) is nonsingular and \( \omega_j > 0 \) for all \( j \), \( \Omega^{L-1} H^{(L)} \) is also nonsingular by (3.20), and hence the linear system can be solved exactly, giving \( r_L = 0 \). However, it can also happen that the Lanczos algorithm terminates with an \( A^T\)-invariant subspace \( L = L_r = L_* < L_1 \) or with an incurable breakdown. In either case, the QMR algorithm needs to be restarted with a different choice of starting vectors. As we
will see in section 4.3, the residual $r_L$ corresponding to the last QMR iterate $x_L$ is a good choice for the new starting vector $v_1$.

It turns out that in practice, the finite termination property is of little importance, for two reasons. Firstly, in practice one usually hopes to achieve convergence in $n \ll N$ steps, before the finite termination would occur. Secondly, due to the loss of biorthogonality mentioned in section 3.7, the Lanczos algorithm often does not find invariant subspaces in $L_\ast$ steps. As a result, QMR, as many other methods based on the Lanczos process, in practice loses the finite termination property.

To compute the solution to (4.3), the standard approach takes advantage of the upper Hessenberg form of $\Omega^{(n)}H_e^{(n)}$ and is based on using Givens rotations to compute the QR factorization of $\Omega^{(n)}H_e^{(n)}$. We compute an $(n+1) \times (n+1)$ unitary matrix $Q^{(n)}$ and an $n \times n$ upper triangular matrix $R_e^{(n)}$ such that

$$
\Omega^{(n)}H_e^{(n)} = \left( Q^{(n)} \right)^H \begin{bmatrix} R_e^{(n)} \\ 0 & \ldots & 0 \end{bmatrix}.
$$

(4.4)

Inserting into (4.3) gives

$$
\min_{z_n} \left\| \omega_1 d^{(n+1)} - \Omega^{(n)}H_e^{(n)}z_n \right\|_2
\quad = \min_{z_n} \left\| \left( Q^{(n)} \right)^H \left( \omega_1 Q^{(n)}d^{(n+1)} - \begin{bmatrix} R_e^{(n)} \\ 0 & \ldots & 0 \end{bmatrix}z_n \right) \right\|_2
\quad = \min_{\tilde{t}_n} \left\| \tilde{t}_n - \begin{bmatrix} R_e^{(n)}z_n \end{bmatrix} \right\|_2,
$$

(4.5)

where

$$
\omega_1 Q^{(n)}d^{(n+1)} = \begin{bmatrix} t^{(n)} \\ \tilde{t}_{n+1} \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_n \\ \tilde{\tau}_{n+1} \end{bmatrix} \in \mathbb{C}^{n+1}, \quad \text{with} \quad \tilde{\tau}_1 = \beta \omega_1.
$$

(4.6)

Recall now that the subdiagonal elements of $\tilde{H}_e^{(n)}$ are all nonzero, as long as an invariant subspace has not been found. Thus, as $\omega_j > 0$ for all $j$, $H_e^{(n)}$ and $\Omega^{(n)}H_e^{(n)}$ have full rank, and so $R_e^{(n)}$ is nonsingular. This means that

$$
z_n = \left( R_e^{(n)} \right)^{-1} t^{(n)}
$$

(4.7)

is the unique solution to (4.3) and (4.5) and determines a unique iterate $x_n$ by

$$
x_n = x_0 + V^{(n)} \left( R_e^{(n)} \right)^{-1} t^{(n)}.
$$

(4.8)
The basic structure of the QMR approach is then to first compute $V^{(n+1)}$ and $H_e^{(n)}$ with a look-ahead Lanczos algorithm, then obtain the QR factorization of $\Omega^{(n)}H_e^{(n)}$, and finally obtain $x_n$ from (4.8).

As indicated, the QR decomposition of $\Omega^{(n)}H_e^{(n)}$ is computed with Givens rotations. The approach used for updating the QR decomposition and the iterates is based on a technique first used by Paige and Saunders [57] for their SYMMLQ and MINRES algorithms. One computes

$$Q^{(n)} = G_n \left[ \begin{array}{cc} G_{n-1}^{-1} & 0 \\ 0 & 1 \end{array} \right] \cdots \left[ \begin{array}{cc} G_1 & 0 \\ 0 & I_{n-1} \end{array} \right] = G_n \left[ \begin{array}{cc} Q^{(n-1)} & 0 \\ 0 & 1 \end{array} \right]$$

(4.9)

where for $j = 1, 2, \ldots$,

$$G_j = \left[ \begin{array}{ccc} I_{j-1} & 0 & 0 \\ 0 & c_j & s_j \\ 0 & -s_j & c_j \end{array} \right],$$

with $c_j \in \mathbb{R}$, $s_j \in \mathbb{C}$, and $c_j^2 + |s_j|^2 = 1$. The factorization of $\Omega^{(n)}H_e^{(n)}$ is obtained by updating the factorization of $\Omega^{(n-1)}H_e^{(n-1)}$. At step $n$, we first need to compute $G_n$, which updates $Q^{(n-1)}$ to $Q^{(n)}$. Then, since $G_n$ does not affect the first $n-1$ columns of $R_e^{(n)}$, we just need to compute the next column of $R_e^{(n)}$ and append it to $R_e^{(n-1)}$.

We note here that, letting $\Omega^{(n)} = I_{n+1}$, we have from (4.4) and (4.9)

$$Q^{(n)}H_e^{(n)} = G_n \left[ \begin{array}{cc} Q^{(n-1)} & 0 \\ 0 & 1 \end{array} \right] \left[ \begin{array}{cccc} H^{(n)} & & & \\ & 0 & \cdots & 0 \\ & & 0 & 1 \end{array} \right]$$

$$= G_n \left[ \begin{array}{cc} Q^{(n-1)}H^{(n)} & 0 \\ 0 & \cdots & 0 & 1 \end{array} \right] = \left[ \begin{array}{cc} R_e^{(n)} & 0 \\ 0 & \cdots & 0 \end{array} \right].$$

Hence,

$$Q^{(n-1)}H^{(n)} = R^{(n)}$$

is the QR factorization of $H^{(n)}$, with

$$G_n \left[ \begin{array}{ccc} R^{(n)} \\ 0 & \cdots & 0 \\ 1 \end{array} \right] = \left[ \begin{array}{ccc} R_e^{(n)} \\ 0 & \cdots & 0 \end{array} \right],$$

so that, as noted in (2.20), $R^{(n)}$ and $R_e^{(n)}$ differ only in their lower right-hand corner element.

To compute $G_n$, one first premultiplies the last column of $\Omega^{(n)}H_e^{(n)}$ by the previous Givens rotations, obtaining

$$\tilde{u} = \left[ \begin{array}{cccc} G_{n-1} & 0 \\ 0 & 1 \end{array} \right] \cdots \left[ \begin{array}{cccc} G_1 & 0 \\ 0 & I_{n-1} \end{array} \right] \left( \Omega^{(n)}H_e^{(n)} \right)_{:,n}$$

$$= \left[ u_1 \quad \cdots \quad u_{n-1} \quad \tilde{u}_n \quad \tilde{u}_{n+1} \right]^T \in \mathbb{C}^{n+1}.$$
Here

\[ \mathbf{u}_{n+1} = \left( \Omega^{(n)} \mathbf{H}_e^{(n)} \right)_{n+1, n}, \]

and \( \mathbf{u} \) matches the last column of \( R_e^{(n)} \) in all but the last two elements. We now have to multiply it by a suitably chosen Givens rotation \( G_n \) that would modify the last two elements and would zero out \( \mathbf{u}_{n+1} \). That is, we want \( c_n \) and \( s_n \) such that

\[
\begin{bmatrix}
  c_n & s_n \\
  -\bar{s}_n & c_n
\end{bmatrix}
\begin{bmatrix}
  \mathbf{u}_n \\
  \bar{\mathbf{u}}_{n+1}
\end{bmatrix} =
\begin{bmatrix}
  c_n \mathbf{u}_n + s_n \bar{\mathbf{u}}_{n+1} \\
  0
\end{bmatrix} =
\begin{bmatrix}
  \mathbf{u}_n \\
  0
\end{bmatrix}.
\]

To achieve this, set

\[
c_n = \frac{|\mathbf{u}_n|}{\left( \bar{\mathbf{u}}_{n+1}^2 + |\mathbf{u}_n|^2 \right)^{1/2}}, \quad \bar{s}_n = c_n \frac{\mathbf{u}_{n+1}}{\mathbf{u}_n}, \quad \text{if } \mathbf{u}_n \neq 0,
\]

\[
c_n = 0, \quad \bar{s}_n = 1, \quad \text{if } \mathbf{u}_n = 0.
\]

With this, the vector

\[
(G_n \mathbf{u})_{1:n} = [u_1 \quad \cdots \quad u_n]^T \in \mathbb{C}^n
\]

is the last column of \( R_e^{(n)} \), and the update of the QR factorization of \( \Omega^{(n)} \mathbf{H}_e^{(n)} \) is complete. To update \( t^{(n-1)} \) to \( t^{(n)} \) and compute \( \mathbf{\tau}_{n+1} \) in (4.6), we set

\[
\begin{bmatrix}
  t^{(n)} \\
  \mathbf{\tau}_{n+1}
\end{bmatrix} = G_n
\begin{bmatrix}
  t^{(n-1)} \\
  \mathbf{\tau}_n \\
  0
\end{bmatrix},
\]

which due to the form of \( G_n \) leaves \( t^{(n-1)} \) unchanged and gives

\[
\mathbf{\tau}_n = c_n \mathbf{\tau}_n \quad \text{and} \quad \mathbf{\tau}_{n+1} = -\bar{s}_n \mathbf{\tau}_n. \tag{4.11}
\]

Finally, with \( t^{(n)} \) computed, we can update \( x_n \). Let us define in (4.8) vectors \( p_j \) by

\[
\mathbf{P}^{(n)} = [p_1 \quad \cdots \quad p_n] = \mathbf{V}^{(n)} \left( R_e^{(n)} \right)^{-1},
\]

so that together with (4.6), (4.8), and (4.11), we get the update formula

\[
x_n = x_{n-1} + \mathbf{\tau}_n p_n. \tag{4.12}
\]
To see how we can obtain \( p_n \), suppose we partition \( P^{(n)} \) into blocks corresponding to the blocks built by the look-ahead Lanczos algorithm, so that

\[
P^{(n)} = [P_1 \quad \cdots \quad P_l],
\]

where \( P_j \) contains the vectors \( p_{nj} \) through \( p_{nj+1-1} \). Then from the definition of \( P^{(n)} \), we get

\[
V^{(n)} = P^{(n)} P^{(n),}_e,
\]

so that the last block \( P_l \) can be obtained from the last block \( V_l \) and the previous blocks \( P_j \). The problem here is that, depending on the nonzero structure of \( R^{(n),}_e \), the expression for \( P_l \) may involve several of the previous blocks \( P_j \). As the structure of \( R^{(n),}_e \) is closely linked to the structure of \( H^{(n),}_e \), here it becomes crucial to use a look-ahead Lanczos algorithm that keeps \( H^{(n),}_e \) as close to block tridiagonal as possible.

To summarize, we have discussed the basic QMR approach, assuming only an underlying look-ahead Lanczos algorithm. Next we investigate how the basic approach can be combined with the LAL algorithm and how to take advantage of the features of that particular implementation of the look-ahead Lanczos algorithm.
Section 4.2. A QMR Algorithm

The basic QMR approach and most of its properties remain the same regardless of the variant of the look-ahead Lanczos algorithm used to build the basis for $K_n(r_0, A)$. However, some of its implementation details change, especially as a function of the structure of $H^{(n)}$.

We will first present a sketch of a QMR algorithm based on LAL, and we will then remark on how this choice of the Lanczos algorithm affects the various steps of the QMR approach. As before, hat (') denotes quantities corresponding to the scaled vectors $\hat{v}_n$ and $\hat{w}_n$.

**QMR Algorithm. Sketch of the QMR algorithm based on LAL.**

1. Choose $x_0 \in \mathbb{C}^N$ and set $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, $\hat{v}_1 = r_0/\beta$. Choose $\hat{w}_1 \in \mathbb{C}^N$ with $\|\hat{w}_1\|_2 = 1$, and choose $\omega_1 > 0$. Carry out the rest of step (1) of algorithm LAL.

For $n = 1, 2, \ldots$:

2. Carry out the $n$th step of algorithm LAL. This updates $\hat{V}^{(n)}$ to $\hat{V}^{(n+1)}$ and $\hat{H}^{(n)}_\epsilon$ to $\hat{H}^{(n)}_\epsilon$.

3. Choose $\omega_{n+1} > 0$ and update the QR factorization of $\Omega^{(n)} \hat{H}^{(n)}_\epsilon$. Update $t^{(n)}$ and compute $\hat{r}_{n+1}$.

4. Update $x_n$.

5. If $x_n$ has not converged, go to step (2).

Step (1) is the initialization step. Step (2) is carried out by the LAL algorithm, which generates $\hat{v}_{n+1}$ and the next column of $\hat{H}^{(n)}_\epsilon$. For step (3), we first choose $\omega_{n+1}$; we will discuss this choice shortly. Once $\omega_{n+1}$ has been chosen, we now have the last column of $\Omega^{(n)} \hat{H}^{(n)}_\epsilon$, and we can update the QR factorization of the matrix. The main point here is that the LAL algorithm generates $\hat{H}^{(n)}_\epsilon$ of block tridiagonal structure, so its last column has the form

$$
\left(\Omega^{(n)} \hat{H}^{(n)}_\epsilon\right)_{:,n} = \begin{bmatrix}
0 \\
\vdots \\
0 \\
\omega_{i_n} \hat{h}_{i_n,n} \\
\vdots \\
\omega_n \hat{h}_{n,n} \\
\omega_{n+1} \rho_{n+1}
\end{bmatrix} \in \mathbb{C}^{n+1},
$$

where the first nonzero element does not occur before row $i_n$. Here, $i_n = n_{l-1}$ if LAL performed an inner step, and thus $l$ is unchanged, or $i_n = n_{l-2}$ if LAL
performed a regular step, and thus \( l \) has been updated. As defined in section 3.3, \( n_0 = 1 \) by convention. Since the column vector is quite sparse, only the Givens rotations \( G_{i_n} \) through \( G_{n-1} \) have to be applied in (4.10), the rotations \( G_1 \) through \( G_{i_n-1} \) are not needed and can be discarded.

Another important effect of the block tridiagonal structure of \( \tilde{H}_e^{(n)} \) is that the recurrences used to update \( P_l \) from (4.13) are three-term recurrences. When \( G_{i_n} \) is applied to the last column of \( \Omega^{(n)} \tilde{H}_e^{(n)} \), it might introduce a nonzero element in row \( i_n - 1 \), but that is the only fill-in that results in \( R_e^{(n)} \). Hence, \( R_e^{(n)} \) has the structure

\[
R_e^{(n)} = \begin{bmatrix}
\delta_1 & \epsilon_2 & \theta_3 \\
\vdots & \ddots & \ddots \\
\vdots & \ddots & \ddots & \theta_l \\
\epsilon_l & \cdots & \cdots & \cdots & \delta_l
\end{bmatrix},
\]

where \( \delta_j \) and \( \epsilon_j \) have the same size as the corresponding blocks \( \alpha_j \) and \( \beta_j \) in (3.26). Moreover, the LAL algorithm produces matrices \( \tilde{H}_e^{(n)} \) that have full column rank, as \( \tilde{h}_{j+1,j} = \rho_{j+1} > 0 \). Hence, \( R_e^{(n)} \) is nonsingular, and so \( \delta_j \) are nonsingular upper triangular matrices. Finally, as pointed out above, \( \theta_j \) has nonzero elements only in its last row. From this structure of \( R_e^{(n)} \), we see that (4.13) leads to

\[
\hat{V}_j = P_j \delta_j + P_{j-1} \epsilon_j + P_{j-2} \theta_j.
\]

so that

\[
P_j = \left( \hat{V}_j - P_{j-1} \epsilon_j - P_{j-2} \theta_j \right) \delta_j^{-1},
\]

which gives a short recurrence for \( P_j \). From \( P_j \), we extract \( p_n \) and then update \( x_n \) according to (4.12).

We now turn to the choice of \( \omega_j \). As the LAL algorithm generates \( \tilde{V}^{(n)} \) with unit length column vectors, we choose \( \omega_j = 1 \) for all \( j \). This means that the basis vectors in the representation (4.1) have unit length, which is a natural requirement. Other choices of \( \omega_j \) might be possible, which is why we formulated the QMR algorithm with \( \Omega^{(n)} \) in it.

To summarize, we have shown how the QMR method combines with the LAL algorithm. We noted that the full QMR approach of section 4.1 simplifies considerably. The QR factorization of \( \Omega^{(n)} \tilde{H}_e^{(n)} \) involves only a few of the most recent Givens rotations, and computing \( p_n \) — which is needed for \( x_n \) — is done with short recurrences. We again stress that both of these simplifications depended on the block tridiagonal structure of \( \tilde{H}_e^{(n)} \).
In the next few sections, we turn our attention to those properties of the basic QMR approach that are independent of the implementation details of the underlying Lanczos algorithm. For convenience and clarity, we will treat these properties over several sections.
Section 4.3. Connection with GMRES

We start with the most important of the results about the QMR approach, namely how far is the quasi-optimal residual approach from the optimal approach, GMRES. We will present two results, one relating the norms of the two residual vectors, $r_n^{\text{QMR}}$ and $r_n^{\text{GMRES}}$, and the other relating the two approximation problems.

**Theorem.**

\[
\|r_n^{\text{QMR}}\|_2 \leq \kappa(\Omega^{(n)}) \kappa(V^{(n+1)}) \|r_n^{\text{GMRES}}\|_2.
\]

**Proof.** Let $\mathcal{R}^{(n)} \subseteq \mathbb{C}^N$ be the set

\[
\mathcal{R}^{(n)} = \left\{ r_n : r_n = V^{(n+1)} \left( \Omega^{(n)} \right)^{-1} y_n, y_n = \omega_1 \sigma_1^{(n+1)} - \Omega^{(n)} H_c^{(n)} z_n, z_n \in \mathbb{C}^n \right\},
\]

and let $U \Sigma V^H$ be the singular value decomposition of $V^{(n+1)}$, with $U \in \mathbb{C}^{N \times N}$ and $V \in \mathbb{C}^{(n+1) \times (n+1)}$ unitary matrices, and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_{n+1}) \in \mathbb{R}^{N \times (n+1)}$. Suppose that the $\sigma_j$'s are ordered $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{n+1}$. Assuming linear independence of the Lanczos basis vectors, which is ensured by the look-ahead Lanczos algorithm, $\sigma_j > 0$. Let then $\Sigma^+ = \text{diag}(\sigma_1^{-1}, \ldots, \sigma_{n+1}^{-1}) \in \mathbb{R}^{(n+1) \times N}$. Together with $\omega_j > 0$, this gives for all $r_n \in \mathcal{R}^{(n)}$

\[
y_n = \Omega^{(n)} V \Sigma^+ U^H r_n \Rightarrow \|y_n\|_2 \leq \max_i (\omega_i) \frac{1}{\sigma_{n+1}} \|r_n\|_2.
\]

Note the since $V^{(n+1)}$ has full rank, $y_n$ is uniquely determined from each $r_n \in \mathcal{R}^{(n)}$ by the above. Now $r_n^{\text{QMR}}$ is the unique $r_n \in \mathcal{R}^{(n)}$ having minimal $\|y_n\|_2$, while $r_n^{\text{GMRES}}$ is the unique $r_n \in \mathcal{R}^{(n)}$ having minimal $\|r_n\|_2$. So

\[
\|r_n^{\text{QMR}}\|_2 = \left\| V^{(n+1)} \left( \Omega^{(n)} \right)^{-1} y_n^{\text{QMR}} \right\|_2 \leq \sigma_1 \max_j \left( \frac{1}{\omega_j} \right) \|y_n^{\text{QMR}}\|_2
\]

\[
\leq \sigma_1 \max_j \left( \frac{1}{\omega_j} \right) \max_i (\omega_i) \frac{1}{\sigma_{n+1}} \|r_n^{\text{GMRES}}\|_2
\]

\[
= \kappa(\Omega^{(n)}) \kappa(V^{(n+1)}) \|r_n^{\text{GMRES}}\|_2.
\]

For the algorithm QMR based on the algorithm LAL, we can bound $\kappa(V^{(n+1)})$ to get:

**Corollary.** If the look-ahead Lanczos algorithm underlying the QMR algorithm generates $\hat{V}^{(n+1)}$ with

\[
\|\hat{v}_j\|_2 = 1 \quad \text{for all } j
\]

81
and

\[ \min_{1 \leq j \leq l} \left( \sigma_{\text{min}}(\hat{\delta}_j) \right) \geq tol \]

for some tolerance tol, then

\[ \| r_n^{QMR} \|_2 \leq \kappa(\Omega^{(n)}) \frac{n + 1}{tol} \| r_n^{GMRES} \|_2. \]

**Proof.** From (3.42), under the assumption \( \| \hat{v}_j \|_2 = 1 \) for all \( j \), we have

\[ \sigma_{\text{min}}(\hat{V}^{(n+1)}) \geq \frac{tol}{\sqrt{n + 1}}, \quad \text{and} \quad \sigma_{\text{max}}(\hat{V}^{(n+1)}) \leq \sqrt{n + 1}. \]

Inserting into the theorem above, we immediately obtain the corollary. \( \Box \)

Note that for the above bounds, the optimal choice of \( \Omega^{(n)} \) is \( \Omega^{(n)} = I_{n+1} \). Also, note that these bounds imply that in exact arithmetic, the GMRES and QMR algorithms started with the same initial guess will converge after the same number of iterations.

The second result links directly the approximation problems solved by QMR and GMRES. Recall that \( L \) denotes the termination index of the Lanczos algorithm and that \( \mathcal{P}_n \) denotes the space of complex polynomials of degree at most \( n \). Let

\[ \mathcal{R}_n = \{ p_n \in \mathcal{P}_n : p_n(0) = 1 \} \]

de note the space of complex residual polynomials of degree at most \( n \). With this, we have our second convergence result for QMR:

**Theorem.** Suppose that the \( L \times L \) matrix \( H^{(L)} \) generated by the look-ahead Lanczos algorithm is diagonalizable,

\[ H^{(L)} = XD_HX^{-1}, \]

and let

\[ H = \left( \Omega^{(L-1)} \right) H^{(L)} \left( \Omega^{(L-1)} \right)^{-1}. \]

\[ (4.14) \]

Then for \( n = 1, \ldots, L - 1 \),

\[ \| r_n^{QMR} \|_2 \leq \| r_0 \|_2 \| V^{(n+1)} \|_2 \kappa(X) \max_{1 \leq j \leq n+1} \left( \frac{\omega_1}{\omega_j} \right) \varepsilon^{(n)}, \]

where

\[ \varepsilon^{(n)} = \max_{p_n \in \mathcal{R}_n} \min_{\lambda \in \Lambda(A)} | p_n(\lambda) |. \]

(4.15)
Proof. We start from (4.1):

\[ r_n = V^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \left( \omega_1 d^{(n+1)} - \Omega^{(n)} H_e^{(n)} z_n \right), \]

which gives

\[ \| r_n \|_2 \leq \| V^{(n+1)} \|_2 \left\| \left( \Omega^{(n)} \right)^{-1} \right\|_2 \left\| \omega_1 d^{(n+1)} - \Omega^{(n)} H_e^{(n)} z_n \right\|_2. \]

Let

\[ \vartheta_n = \min_z \left\| e_1^{(n+1)} - \Omega^{(n)} H_e^{(n)} z \right\|_2, \quad z \in \mathbb{C}^n, \tag{4.16} \]

where

\[ e_1^{(n+1)} = [1 \ 0 \ \cdots \ 0]^T \in \mathbb{R}^{n+1}. \]

Now if \( z_n \) is given by (4.3), then from (4.5)

\[ \left\| \omega_1 d^{(n+1)} - \Omega^{(n)} H_e^{(n)} z_n \right\|_2 = |\tilde{r}_{n+1}| = \omega_1 |\beta| \vartheta_n, \]

so that

\[ \| r_n \|_2 \leq |\beta| \left\| V^{(n+1)} \right\|_{2 \max 1 \leq j \leq n+1} \left( \frac{\omega_1}{\omega_j} \right) \vartheta_n. \]

Let now \( 1 \leq n \leq L - 1 \) be arbitrary, but fixed. By

\[ H^{(L)} = \begin{bmatrix} H_e^{(n)} & * \\ 0 & * \end{bmatrix} \]

and (4.14), we have

\[ H \begin{bmatrix} z \\ 0 \end{bmatrix} = \begin{bmatrix} \left( \Omega^{(n)} \right) H_e^{(n)} \left( \Omega^{(n-1)} \right)^{-1} z \\ 0 \end{bmatrix}, \quad \text{for all } z \in \mathbb{C}^n. \tag{4.17} \]

\( H^{(n)} \) is an upper Hessenberg matrix with nonzero subdiagonal elements. As \( \omega_j > 0 \), \( H \) is also an upper Hessenberg matrix with nonzero subdiagonal elements, hence

\[ \left\{ \begin{bmatrix} z \\ 0 \end{bmatrix} \right\} : z \in \mathbb{C}^n = \left\{ p(H)e_1^{(L)} : p \in \mathcal{P}_{n-1} \right\}. \tag{4.18} \]

From (4.16–4.18) we have

\[ \vartheta_n = \min_z \left\| e_1^{(L)} - H \begin{bmatrix} z \\ 0 \end{bmatrix} \right\|_2 = \min_{p_n \in \mathcal{P}} \left\| p_n(H)e_1^{(L)} \right\|_2. \]

\( H^{(L)} \) is assumed diagonalizable, so by (4.17) \( H \) is also diagonalizable, hence

\[ \vartheta_n \leq \kappa(X) \min_{p_n \in \mathcal{P}} \max_{\lambda \in \Lambda(H)} |p_n(\lambda)|. \]

83
From (4.17) and (3.20), \( \Lambda(H) = \Lambda(H^{(L)}) \subseteq \Lambda(A) \), and we immediately obtain the desired inequality. 

Once again, for the look-ahead Lanczos algorithm LAL, \( \|V^{(n+1)}\|_2 \leq \sqrt{n+1} \), so the bound simplifies in this case. The standard bounds for GMRES [71] have a similar form. Suppose that \( A \) is diagonalizable, so that \( A = UDU^{-1} \). Then the GMRES bounds are

\[
\|r_n^{\text{GMRES}}\|_2 \leq \|r_0\|_2 \kappa(U)\varepsilon^{(n)}, \quad n = 1, \ldots,
\]

where \( \varepsilon^{(n)} \) is the same as in (4.16). Thus, the bounds for QMR and GMRES involve the same approximation problem. Finally, note that the results above imply that if the Lanczos algorithm terminates before finding an \( A \)-invariant subspace, so that the QMR algorithm needs to be restarted, then the last residual \( r_L \) is a good choice for the new starting vector \( v_1 \).
Section 4.4. Residual Norm Upper Bound

In practice, the convergence check at step (5) of the algorithm QMR often involves \( \|r_n\|_2 \). The purpose of this section is to provide an upper bound for \( \|r_n\|_2 \), so that one need not compute \( r_n = b - Ax_n \) and its norm at every step. The bound obtained here will be for algorithm QMR, though similar bounds should be obtainable for other implementations of the QMR approach.

To obtain a bound for \( \|r_n\|_2 \), we again start from (4.1):

\[
r_n = \hat{V}^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \omega_1 d^{(n+1)} - \Omega^{(n)} \hat{H}_e^{(n)} z_n,
\]

which gives

\[
\|r_n\|_2 \leq \left\| \hat{V}^{(n+1)} \right\|_2 \left\| \left( \Omega^{(n)} \right)^{-1} \right\|_2 \left\| \omega_1 d^{(n+1)} - \Omega^{(n)} \hat{H}_e^{(n)} z_n \right\|_2.
\]

Now if \( z_n \) is given by (4.3), then from (4.5)

\[
\left\| \omega_1^{(n+1)} d^{(n+1)} - \Omega^{(n)} \hat{H}_e^{(n)} z_n \right\|_2 = |\tilde{r}_{n+1}|,
\]

and from (4.11) and (4.6),

\[
|\tilde{r}_{n+1}| = |-\bar{s}_n \tilde{r}_n| = |s_1 \ldots s_n | \tilde{r}_1 | = |\beta| \omega_1 |s_1 \ldots s_n|.
\]

Finally, we have for the look-ahead Lanczos algorithm, LAL, \( \|\hat{V}^{(n+1)}\|_2 \leq \sqrt{n + 1} \) and \( \beta = \|r_0\|_2 \), so that

\[
\|r_n\|_2 \leq \sqrt{n + 1} \max_j \left( \frac{1}{\omega_j} \right) \beta \omega_1 |s_1 \ldots s_n|.
\]

We remark that all quantities are available at no extra cost.

Going back to (2.18) and (2.29), we are now in a position to prove that for both of these, the system is inconsistent when \( H^{(n)} \) is singular. To prove it for (2.18), we use (4.19), with \( \Omega^{(n)} = I_{n+1} \). Using the QR decomposition (2.20) of \( H^{(n)} \) and (4.6), we get from (2.18)

\[
R_e^{(n)} z_n = Q^{(n-1)} d^{(n)} = \begin{bmatrix} \ell^{(n-1)} \\ \tilde{r}_n \end{bmatrix}.
\]

Now from (4.19),

\[
|\tilde{r}_n| = \|r_{n-1}^{G M R E S}\|_2 > 0,
\]

as otherwise the algorithm would have converged at step \( n - 1 \). On the other hand, when \( H^{(n)} \) is singular, the lower right-hand corner element of \( R_e^{(n)} \) is zero. Thus, the system is inconsistent. A similar argument shows that (2.29) is inconsistent when \( H^{(n)} \) is singular.
Section 4.5. Residual Vector Updating

Another possibility is to update the residual vector $r_n$ from the previous vector $r_{n-1}$. Let us write from (4.1), (4.4), and (4.6),

$$
r_n = \hat{V}^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \left( \omega_1 d^{(n+1)} - \Omega^{(n)} \hat{H}_e^{(n)} z_n \right)
$$
$$
= \hat{V}^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \left( Q^{(n)} \right)^H \left( \omega_1 Q^{(n)} d^{(n+1)} - \begin{bmatrix} R_e^{(n)} \\ 0 & \cdots & 0 \end{bmatrix} z_n \right)
$$
$$
= \hat{V}^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \left( Q^{(n)} \right)^H \begin{bmatrix} t^{(n)} - R_e^{(n)} z_n \\ \tilde{r}_{n+1} \end{bmatrix}.
$$

Define now

$$Y^{(n+1)} = \begin{bmatrix} y_1 & \cdots & y_n & \tilde{y}_{n+1} \end{bmatrix} = \hat{V}^{(n+1)} \left( \Omega^{(n)} \right)^{-1} \left( Q^{(n)} \right)^H,$$

where

$$
\left( Q^{(n)} \right)^H = \begin{bmatrix}
\begin{array}{cccc}
c_1 & -s_1 & & \\
\overline{s_1} & c_1 & & \\
& & \ddots & \\
& & & 1
\end{array}
\end{bmatrix}
\begin{bmatrix}
1 & & & \\
& \ddots & & \\
& & 1 & \\
& & & \frac{c_n}{s_n} - s_n
\end{bmatrix}.
$$

We remark that the vectors $y_j$ are given by the recurrences

$$y_n = c_n \tilde{y}_n + \frac{1}{\omega_{n+1}} \hat{v}_{n+1},$$

$$\tilde{y}_{n+1} = -s_n \tilde{y}_n + c_n \frac{1}{\omega_{n+1}} \hat{v}_{n+1},$$

started with

$$\tilde{y}_1 = \frac{1}{\omega_1} \hat{v}_1.$$

From (4.22) and (4.23) we obtain

$$r_n = Y^{(n+1)} \begin{bmatrix} t^{(n)} - R_e^{(n)} z_n \\ \tilde{r}_{n+1} \end{bmatrix},$$

which together with (4.7) yields the update formula

$$r_n = \tilde{r}_{n+1} \tilde{y}_{n+1} = \tilde{r}_{n+1} \left( -s_n \tilde{y}_n + c_n \frac{1}{\omega_{n+1}} \hat{v}_{n+1} \right)
$$
$$
= (-s_n) ( -\frac{1}{s_n} \tilde{r}_n ) \tilde{y}_n + c_n \frac{\tilde{r}_{n+1}}{\omega_{n+1}} \hat{v}_{n+1}
$$
$$
= |s_n|^2 r_{n-1} + c_n \frac{\tilde{r}_{n+1}}{\omega_{n+1}} \hat{v}_{n+1}.$$
Section 4.6. Connection with BCG

We recall from our discussion in Chapter 2 that by coupling the Lanczos process with a Galerkin condition on the residual, one gets the biconjugate gradient algorithm (BCG). Proposed by Cornelius Lanczos in [47] and in a mathematically equivalent form by Roger Fletcher in [18], the BCG algorithm obtains its iterates by computing

\[ z_{n}^{BCG} = \left( H^{(n)} \right)^{-1} d^{(n)}, \]  

(4.25)

and then using the standard formula for \( x_{n} \) to get

\[ x_{n}^{BCG} = x_{0} + V^{(n)} z_{n}^{BCG} = x_{0} + V^{(n)} \left( H^{(n)} \right)^{-1} d^{(n)}. \]

The classical BCG algorithm assumes that the underlying Lanczos algorithm has no breakdowns, so that \( H^{(n)} \) is tridiagonal and can be inverted implicitly as was discussed in Chapter 2. This means that the classical BCG algorithm may break down in two situations: if the Lanczos algorithm has a breakdown, or if \( H^{(n)} \) is singular. The first breakdown can be circumvented by using a look-ahead variant of the Lanczos process, but in doing so, \( H^{(n)} \) is no longer guaranteed to be tridiagonal, and hence can no longer be inverted implicitly. Instead, the linear system (4.25) has to be solved by other means, for example by using the QR decomposition of \( H^{(n)} \). In addition, there still remains the possibility of \( H^{(n)} \) being singular, which means that even using a look-ahead Lanczos algorithm, the BCG approach can still break down in exact arithmetic, or lose arbitrarily many significant digits in finite precision arithmetic.

In this section, we investigate the connection between the BCG and QMR approaches, as reflected in the connection between their iterates. It turns out that one can obtain the BCG iterate \( x_{n}^{BCG} \) — when it exists — from the previous QMR iterate \( x_{n-1}^{QMR} \), regardless of whether \( H^{(n)} \) was singular at a previous step or not. The computation of the BCG iterate \( x_{n}^{BCG} \) will not involve previous BCG iterates, which might not exist. In particular, any roundoff errors introduced in computing a BCG iterate when \( H^{(n)} \) is nearly singular will not propagate to affect subsequent iterates. Finally, the QMR process also gives \( \| r_{n}^{BCG} \|_{2} \), without requiring the computation of the residual vector \( r_{n}^{BCG} \). Hence, QMR may also be viewed and can be used as a stable way of computing the BCG iterates.

We note that the expression (4.25) for \( z_{n}^{BCG} \) was obtained under the assumption that \( D^{(n)} \) is nonsingular (see the end of section 2.4). Hence, throughout this section, we will always assume that \( n = n_{j} - 1, \ j = 1, \ldots \). Next, we wish to obtain an
expression for \((H^{(n)})^{-1}\) involving quantities from the QMR process. We start from the QR factorization (4.4) of \(\Omega^{(n)}H_e^{(n)}\):

\[
Q^{(n)}\Omega^{(n)}H_e^{(n)} = \begin{bmatrix}
R_e^{(n)} \\
0 & \cdots & 0
\end{bmatrix}.
\]

Expanding the left-hand side gives

\[
G_n \begin{bmatrix}
Q^{(n-1)} & 0 \\
0 & 1
\end{bmatrix} \begin{bmatrix}
\Omega^{(n-1)} & 0 \\
0 & \omega_{n+1}
\end{bmatrix} \begin{bmatrix}
H^{(n)} \\
0 & \cdots & 0 & \hat{\rho}_{n+1}
\end{bmatrix} = G_n \begin{bmatrix}
Q^{(n-1)}\Omega^{(n-1)}H^{(n)} \\
0 & \cdots & 0 & \omega_{n+1}\hat{\rho}_{n+1}
\end{bmatrix} = \begin{bmatrix}
R_e^{(n)} \\
0 & \cdots & 0
\end{bmatrix},
\]

where \(\hat{\rho}_{n+1}\) denotes the nonzero scaling factor that depends on the variant of the look-ahead Lanczos algorithm used. Now \(G_n\) is always invertible, so

\[
\begin{bmatrix}
Q^{(n-1)}\Omega^{(n-1)}H^{(n)} \\
0 & \cdots & 0 & \omega_{n+1}\hat{\rho}_{n+1}
\end{bmatrix} = G_n^H \begin{bmatrix}
R_e^{(n)} \\
0 & \cdots & 0
\end{bmatrix}
\]

\[
= \begin{bmatrix}
I_{n-1} & c_n \\
\frac{c_n}{s_n} & -s_n
\end{bmatrix} \begin{bmatrix}
R_e^{(n)} \\
0 & \cdots & 0
\end{bmatrix} = \begin{bmatrix}
I_{n-1} & c_n \\
0 & \cdots & 0
\end{bmatrix} \begin{bmatrix}
I_{n-1} & c_n \\
\frac{c_n}{s_n} & -s_n
\end{bmatrix} R_e^{(n)},
\]

which finally gives for \(H^{(n)}\)

\[
H^{(n)} = \left(\Omega^{(n-1)}\right)^{-1} \left(Q^{(n-1)}\right)^{H} \begin{bmatrix}
I_{n-1} \\
1/c_n
\end{bmatrix} R_e^{(n)}.
\]

Note \(H^{(n)}\) is singular if and only if \(c_n = 0\), as all the other matrices are always full rank. Secondly, note that, as seen from (4.11), when \(c_n = 0\) the QMR approach stagnates, with \(\tau_n = c_n\hat{\tau}_n = 0\), so that \(x_n^{\text{QMR}} = x_n^{\text{QMR}}\) in (4.12).

Assuming now that \(c_n \neq 0\), we have from the above

\[
(H^{(n)})^{-1} = (R_e^{(n)})^{-1} \begin{bmatrix}
I_{n-1} \\
1/c_n
\end{bmatrix} Q^{(n-1)}\Omega^{(n-1)},
\]

so that from (4.25) and (4.6),

\[
z_n^{\text{BCG}} = (H^{(n)})^{-1} d^{(n)} = (R_e^{(n)})^{-1} \begin{bmatrix}
I_{n-1} \\
1/c_n
\end{bmatrix} Q^{(n-1)}\Omega^{(n-1)}d^{(n)}
\]

\[
= (R_e^{(n)})^{-1} \begin{bmatrix}
I_{n-1} \\
1/c_n
\end{bmatrix} \begin{bmatrix}
\tilde{t}^{(n-1)} \\
\tilde{\tau}_n
\end{bmatrix} = (R_e^{(n)})^{-1} \begin{bmatrix}
\tilde{t}^{(n-1)} \\
\tilde{\tau}_n/c_n
\end{bmatrix}.
\]

88
Thus, the BCG iterate is given by

\[
x_{n}^{\text{BCG}} = x_{0} + V^{(n)} x_{n}^{\text{BCG}} = x_{0} + P^{(n-1)} t^{(n-1)} + P^{(n)} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \tilde{r}_{n}/c_{n} \end{bmatrix}
\]

\[
= x_{n-1}^{\text{QMR}} + \frac{\tilde{r}_{n}}{c_{n}} p_{n}.
\]

We also show how to obtain \( r_{n}^{\text{BCG}} \) without actually computing \( r_{n}^{\text{BCG}} \). We have

\[
r_{n}^{\text{BCG}} = r_{0} - AV^{(n)} (R_{e}^{(n)})^{-1} \begin{bmatrix} t^{(n-1)} \\ \tilde{r}_{n}/c_{n} \end{bmatrix}
\]

\[
= V^{(n+1)} (\Omega^{(n)})^{-1} \left( \omega_{1} d^{(n+1)} - \Omega^{(n)} H^{(n)}_{e} (R_{e}^{(n)})^{-1} \begin{bmatrix} t^{(n-1)} \\ \tilde{r}_{n}/c_{n} \end{bmatrix} \right)
\]

\[
= Y^{(n+1)} \left( \omega_{1} Q^{(n)} d^{(n+1)} - \begin{bmatrix} R_{e}^{(n)} \\ 0 \ldots \ 0 \end{bmatrix} (R_{e}^{(n)})^{-1} \begin{bmatrix} t^{(n-1)} \\ \tilde{r}_{n}/c_{n} \end{bmatrix} \right)
\]

\[
= Y^{(n+1)} \begin{bmatrix} t^{(n-1)} \\ \tau_{n} \\ \tilde{r}_{n+1} \\ 0 \end{bmatrix} - \begin{bmatrix} t^{(n-1)} \\ \tilde{r}_{n}/c_{n} \end{bmatrix}
\]

\[
= \left( \tau_{n} - \tilde{r}_{n}/c_{n} \right) y_{n} + \tilde{r}_{n+1} \tilde{y}_{n+1},
\]

where \( Y^{(n+1)} \) is defined in (4.23). Expanding \( y_{n} \) and \( \tilde{y}_{n+1} \) according to (4.24) gives

\[
r_{n}^{\text{BCG}} = \left( \tau_{n} - \frac{\tilde{r}_{n}}{c_{n}} \right) \begin{bmatrix} c_{n} \tilde{y}_{n} + \frac{1}{\omega_{n+1}} \tilde{v}_{n+1} \\ \tau_{n} \\ \tilde{r}_{n+1} \end{bmatrix} + \frac{1}{\omega_{n+1}} \left( \frac{1}{s_{n}} \tau_{n} - s_{n} \tau_{n+1} \right) \tilde{v}_{n+1}.
\]

We now use the formulas \( \tau_{n} = c_{n} \tilde{r}_{n} \) and \( \tilde{r}_{n+1} = -s_{n} \tilde{r}_{n} \) from (4.11) to simplify the coefficients:

\[
c_{n} \tau_{n} - \tilde{r}_{n} - s_{n} \tilde{r}_{n+1} = c_{n}^{2} \tilde{r}_{n} - \tilde{r}_{n} + s_{n} s_{n} \tilde{r}_{n} = (c_{n}^{2} + s_{n} s_{n} - 1) \tilde{r}_{n} = 0,
\]

and

\[
\frac{1}{s_{n}} \tau_{n} - \frac{1}{s_{n}} \frac{\tilde{r}_{n}}{c_{n}} + c_{n} \tilde{r}_{n+1} = \frac{1}{s_{n}} \tau_{n} + \frac{1}{c_{n} \left( \frac{1}{s_{n}} \right)} \tilde{r}_{n+1} + c_{n} \left( \frac{1}{s_{n}} \right) \frac{\tilde{r}_{n+1}}{c_{n}} = \frac{\tilde{r}_{n+1}}{c_{n}},
\]

so that finally

\[
r_{n}^{\text{BCG}} = \frac{\tilde{r}_{n+1}}{c_{n} \omega_{n+1}} \tilde{v}_{n+1}.
\]

It should come as no surprise that \( r_{n}^{\text{BCG}} \) is just a multiple of \( \tilde{v}_{n+1} \). We know that \( r_{n}^{\text{BCG}} \in K_{n+1}(r_{0}, A) \) and that \( r_{n}^{\text{BCG}} \perp_{T} W^{(n)} \), which together mean that it can have

89
components only along \( v_{n+1} \). If the look-ahead Lanczos algorithm scales \( v_{n+1} \) in such a way that \( \| \hat{v}_{n+1} \|_2 \) is known, then (4.27) gives the norm of \( r_n^{\text{BCG}} \). For the algorithm \( \text{LAL} \), \( \| \hat{v}_{n+1} \|_2 = 1 \), so that (4.27) together with (4.20) give

\[
\| r_n^{\text{BCG}} \|_2 = \frac{\hat{r}_{n+1}}{c_n \omega_{n+1}} = \| r_0 \|_2 |s_1 \cdots s_n| \frac{\omega_1}{c_n \omega_{n+1}},
\]

where all quantities are known.

We should point out that it may pay to monitor \( \| r_n^{\text{BCG}} \|_2 \) during the course of the QMR algorithm. The QMR iterates are not guaranteed to always give a smaller residual than the BCG iterates, so there might be times when the BCG iterate has a smaller residual than the corresponding QMR iterate, and is better in this sense. At these times, the BCG iterate can be easily computed from (4.26).

We summarize the results of this section in the following:

**Theorem.** Let \( n = n_j - 1, j = 1, \ldots \). Then the following are equivalent:

1. The BCG iterate \( x_n^{\text{BCG}} \) exists,
2. \( H^{(n)} \) is nonsingular,
3. \( c_n \neq 0 \).

Furthermore, if \( x_n^{\text{BCG}} \) exists, then

\[
x_n^{\text{BCG}} = x_{n-1}^{\text{BCG}} + \hat{r}_n c_n p_n,
\]

\[
\| r_n^{\text{BCG}} \|_2 = \| r_0 \|_2 |s_1 \cdots s_n| \frac{\omega_1}{c_n \omega_{n+1}}.
\]

We conclude this section with two examples that illustrate the two possible breakdowns in the classical BCG algorithm, as well as in CGS [73] and in Bi-CGSTAB [81], two algorithms based on BCG. Both algorithms use as a starting point the BCG iteration polynomials, which they modify to obtain a new iteration polynomial. In CGS, the BCG iteration polynomial is multiplied by itself, while in Bi-CGSTAB, the BCG iteration polynomial is multiplied by the iteration polynomial from the steepest descent algorithm [48]. The point is that in exact arithmetic, both CGS and Bi-CGSTAB will break down at exactly the same point as BCG. We should point out that exact breakdowns of the type illustrated in the two examples are unlikely in practice thanks to roundoff. Nevertheless, less dramatic breakdowns are possible in practice and can lead to the rather erratic convergence curves that have been observed for BCG and the other algorithms based on BCG.
In both of the examples below, the matrix $A$ has order $N = 20$. On the vertical axis, we plot the scaled residual norm $\|r_n\|_2/\|r_0\|_2$, while on the horizontal axis, we plot the iteration number.

**Example 5.** This example was constructed so that the Lanczos algorithm would encounter an exact breakdown at step 2. This then leads to a breakdown of BCG, CGS, and Bi-CGSTAB at step 3, as indicated by the convergence curves becoming vertical. QMR using the look-ahead Lanczos algorithm LAL has no problems at the breakdown, and beyond the breakdown, the BCG iterates can again be computed from the QMR iterates. Note that, at least in theory, such a breakdown could be avoided in the BCG algorithm and its variants. One could combine a look-ahead Lanczos algorithm with the BCG approach to obtain a BCG algorithm that would be able to handle breakdowns in the Lanczos algorithm.

![Figure 5. Breakdown of the underlying Lanczos algorithm.](image)

**Example 6.** This example was constructed so that the matrix $H^{(5)}$ generated after 5 steps of the Lanczos process is singular and hence, by the theorem in this section, $x_5^{BCG}$ does not exist. This also leads to a breakdown of the CGS and Bi-CGSTAB algorithms, as indicated by the convergence curves becoming vertical.
at step 5. Once again, QMR does not encounter any problems, and beyond the breakdown, the BCG iterates can be computed from the QMR iterates. In the figure, the CGS convergence curve is the dashed line, and the Bi-CGSTAB convergence curve is the solid line becoming vertical at step 5.

**Figure 6.** Breakdown of the Galerkin condition.
Section 4.7. Numerical Examples (Linear Systems)

In this section, we present a few numerical examples with the algorithm QMR applied to non-Hermitian linear systems. When solving large linear systems with iterative methods, one usually combines the linear system solver with a preconditioning technique, so as to improve the convergence of the solver. In the numerical examples we will discuss, we have used the SSOR and the ILUT(k) preconditioners, hence we first describe briefly the two preconditioning techniques.

Let $M$ be a given nonsingular $N \times N$ matrix which approximates in some sense the matrix $A$ of the linear system (2.1). Suppose that $M$ is decomposed in

$$
M = M_1 M_2.
$$

(4.28)

Instead of solving the original linear system (2.1), we apply the QMR algorithm to the equivalent linear system

$$
A' y = b',
$$

(4.29)

where

$$
A' = M_1^{-1} A M_2^{-1},
$$

$$
b' = M_1^{-1} (b - A x_0),
$$

$$
y = M_2 (x - x_0).
$$

As before, $x_0$ denotes the initial guess for the solution $A^{-1} b$. The iterates $y_n$ and the residuals $r'_n = b' - A' y_n$ of the preconditioned system (4.29) are transformed back into the corresponding quantities for the original system (2.1) by setting

$$
x_n = x_0 + M_2^{-1} y_n, \quad \text{and} \quad r_n = M_1 r'_n.
$$

Depending on the matrices $M_1$ and $M_2$, one can have different types of preconditioning. Thus, if $M_1 = I$ or $M_2 = I$, then one obtains right or left preconditioning, respectively; otherwise, one has two-sided preconditioning. In our case, since the underlying Lanczos algorithm LAL requires both multiplications by $A$ and by $A^T$, in general one has to be able to compute $M_1^{-1} z$, $M_1^{-T} z$, $M_2^{-1} z$, and $M_2^{-T} z$, for arbitrary vectors $z \in \mathbb{C}^N$.

The SSOR preconditioner [2] is based on a decomposition of the matrix $A$ into a nonsingular diagonal matrix $D$, a strictly lower triangular matrix $L$, and a strictly upper triangular matrix $U$, such that

$$
A = L + D + U.
$$
In general, $D$ might have to be block diagonal to ensure that it is nonsingular. The preconditioner matrix $M$ is then given by

$$M = (D + \omega L)D^{-1}(D + \omega U),$$

with $\omega$ the relaxation parameter, and can be used as either a right or a left preconditioner. The preconditioner is fairly insensitive to the value of $\omega$, so one usually sets $\omega = 1$; this was done in the examples below.

The Incomplete LU preconditioner is based on the LU decomposition of the matrix $A$ into a unit lower triangular matrix $L$ and an upper triangular matrix $U$. For a sparse matrix $A$, the full LU decomposition would result in factors $L$ and $U$ that in general have far more nonzero elements than $A$ itself. The incomplete LU factorization then aims to reduce the amount of fill-in in the factors $L$ and $U$.

In ILUT($k$), we use a strategy due to Saad [70] for limiting fill-in in $L$ and $U$. Each row of $L$ is allowed to have at most $k$ more nonzero elements than the corresponding row of $A$, and similarly for each row of $U$. The strategy also drops elements whose contribution to the decomposition is considered to be insignificant. For example, if $n_{\text{max}}$ is the maximum number of elements allowed for some row of $L$, $n$ is the actual number of elements of that row computed by the elimination process, and $\text{tol}$ is the cutoff tolerance, then the algorithm orders the $n$ elements in order of decreasing magnitude, and keeps at most $\min(n, n_{\text{max}})$ elements, but no elements smaller than $\text{tol}$. The resulting matrices $L$ and $U$ can be used as $M_1 = L$ and $M_2 = U$ (two-sided preconditioner), or as $M_1 = I$ and $M_2 = LU$ (right preconditioner), or as $M_1 = LU$ and $M_2 = I$ (left preconditioner).

We note that this variant of ILU is different from the standard one. For a Hermitian matrix $A$, the standard ILU preconditioner proposed by Meijerink and van der Vorst in [53] preserves the sparsity pattern of the matrix. Thus, for $k = 0$, the preconditioner matrices have nonzero elements only in those positions where $A$ itself has nonzero elements. Meijerink and van der Vorst show that this strategy produces a good preconditioner, provided that $A$ is a Hermitian $M$-matrix. For a general non-Hermitian matrix $A$, there is no reason to preserve the sparsity pattern, and hence our ILUT($k$) variant discards elements subject only to the constraints of fill-in and size. However, this does mean that if $A$ is a Hermitian matrix, we do not recover the standard ILU preconditioner.

We present four examples of the algorithm QMR and compare it with BCG and with restarted GMRES. In all the examples, unless otherwise noted, the BCG iterates were obtained from QMR using (4.26). The restart parameter $m$ in GMRES($m$)
was chosen so that the work and storage required by one cycle of GMRES(m) — which comprises m steps of GMRES followed by the computation of the GMRES solution — is roughly comparable to m steps of QMR. For our matrices, a typical value of the restart parameter was \( m = 20 \). Both QMR and GMRES were always started with \( x_0 = 0 \), and the iteration was run until a reduction of \( 10^{-6} \) in the residual norm was obtained, or some arbitrary iteration limit was exceeded. The examples below use right preconditioning; we also tried left preconditioning and obtained similar results in terms of convergence behavior. Of course, both GMRES and QMR were preconditioned using the same preconditioner. Wherever applicable, we indicate the number of blocks of size greater than 1 built during the QMR run by the underlying look-ahead Lanczos algorithm. In all examples, the Lanczos algorithm was allowed to build blocks no larger than 10; the largest block actually built was of size 4. We should point out that the blocks reported are only those built after a reasonable value for \( n(A) \) was obtained. We always started the Lanczos algorithm with the estimate (3.49) for \( n(A) \), and typically the algorithm will immediately adjust \( n(A) \) once or twice before proceeding. All plots show the scaled residual norm \( \|r_n\|_2/\|r_0\|_2 \) on the vertical axis, and the iteration number \( n \) on the horizontal axis. For GMRES, this is the actual iteration number, not the number of restart cycles. Unless otherwise indicated, the residual norm was obtained by actually computing at each step the iterate \( x_n \) and then the norm of its residual \( r_n = b - Ax_n \). In practice, both GMRES and QMR make available estimates for the residual norm, so that one would not normally compute the residual and its norm at every step. Finally, all examples were run on a Cray-2 at the NASA Ames Research Center.

**Example 7.** We consider the 3-D partial differential equation

\[
Lu = f \quad \text{on} \quad (0, 1) \times (0, 1) \times (0, 1),
\]

where

\[
Lu = -\Delta u + \gamma \left( x \frac{\partial u}{\partial x} + y \frac{\partial u}{\partial y} + z \frac{\partial u}{\partial z} \right) + \beta u,
\]

with Dirichlet boundary conditions \( u = 0 \). The operator was discretized using centered differences on a uniform \( 25 \times 25 \times 25 \) grid with mesh size \( h = 1/26 \), leading to a sparse nonsymmetric matrix of order \( N = m^3 = 15625 \), with 105625 nonzero elements. For the parameters, we used \( \beta = -250 \) and \( \gamma = 40 \); this choice of parameters guarantees that the cell Reynolds number is smaller than 1, and hence centered differences yield a stable discretization. The right-hand side \( b \) was
chosen such that the vector of all 1’s is the exact solution of the linear system. The Lanczos process was started with $v_1 = w_1 = b$, and it built 4 blocks of size 2. In this example, we used right SSOR preconditioning. In Figure 7, we show the convergence curves for QMR (the solid line), BCG (the dotted line that oscillates), GMRES(19) (the dash-dotted line), GMRES(20) (the dashed line), and full GMRES (the lowest dotted line). As the plot indicates, the convergence curve for QMR is rather smooth; we also see the typical oscillations in the BCG convergence curve. GMRES, while being optimal as long as it is not restarted, loses its edge once it is restarted, and furthermore its behavior after it is restarted can be quite sensitive to the restart parameter $m$. In this case, we accidentally found an example where GMRES(20) does converge, albeit slowly, while GMRES(19) does not show signs of convergence after 210 steps. For this example, full GMRES converged in 43 steps, and we plot its convergence curve to show that QMR, which converged in 51 steps, was fairly close in performance to the optimal algorithm, while requiring less work and storage.

**Figure 7.** Convergence curves for Example 7.
Example 8. This example was run on a matrix from the Harwell-Boeing set of sparse test matrices [12]. The matrix used, SHERMAN 5, is from the SHERMAN collection and comes from a fully implicit black oil simulator on a $16 \times 23 \times 3$ grid, with three unknowns per grid point. The order of the matrix is 3312 and it has 20793 nonzero elements. The elements of the right-hand side vector $b$ were chosen as random entries from a normal distribution with mean 0.0 and variance 1.0. The Lanczos algorithm was started with $v_1 = w_1 = b$, and it built 49 blocks of size 2, 7 blocks of size 3, and 1 block of size 4. In this example, we used no preconditioning, since we were interested in the convergence behavior of the QMR algorithm over a longer run. The linear system is a fairly difficult one if not preconditioned, and the QMR algorithm needs almost 1500 iterations to converge. By comparison, restarted GMRES($m$), even with an unrealistically large restart value $m = 100$, does not converge after 2500 iterations. In fact, after 4100 iterations, GMRES(100) has only reached the $5.1\times10^{-5}$ level. In this example, we also tried the standard BCG algorithm. In the presence of roundoff, the BCG iterates obtained from QMR and the iterates obtained from the standard algorithm may diverge from each other, especially once the look-ahead Lanczos algorithm builds a block or the
Lanczos matrix $H^{(n)}$ is nearly singular. Furthermore, the standard BCG algorithm builds each iterate from the previous one, using an update formula similar to (4.12). However, this can lead to an accumulation of roundoff errors, which may slow down the convergence of the classical BCG algorithm (as is the case in this example), or indeed, prevent it from converging altogether. In Figure 8, we show the convergence curves for QMR (the solid line), BCG from QMR (the dots), standard BCG (the dotted line), and GMRES(100) (the dashed line).

**Figure 9.** Convergence curves for Example 9.

**Example 9.** This example was run on the same SHEtMAN 5 matrix of Example 8. This time, we preconditioned the linear system with the ILUT(0) preconditioner, with a cutoff tolerance of $tol = 0.001$, used as a two-sided preconditioner. This choice leads to factors $L$ and $U$ which together have 19899 nonzero terms. The right-hand side $b$ was chosen the same as in Example 8. The Lanczos algorithm was again started with $v_1 = w_1 = b$, and it built only blocks of size 1. The main purpose of this example was to see how good is the upper bound (4.21), and how close is the QMR convergence curve to the GMRES convergence curve. In Figure 8, we show the convergence curves for QMR (the solid line), BCG (the dotted line), GM-
RES(20) (the longer dash-dotted line), full GMRES (the lower dash-dotted line), as well as the residual norm upper bound (4.21) (the dashed line). As can be seen, the upper bound is close to the true QMR residual norm, as we have also noticed in other experiments. We also see that QMR is again fairly close to the full GMRES algorithm.

Based on our numerical experience, the convergence behavior of the QMR algorithm illustrated by these examples is fairly typical. The important features are the smooth convergence curves, as opposed to the wildly oscillating BCG convergence curves, and the good quality of the residual norm upper bound (4.21). Finally, the QMR convergence curve is often quite close to the GMRES convergence curve, though this becomes harder to check on the more difficult problems, where GMRES has to be restarted.
Chapter 5. Conclusions

In conclusion, this thesis presents two new results: a new variant of the look-ahead Lanczos algorithm, and a new iterative method for solving non-Hermitian linear systems. The new look-ahead Lanczos algorithm is based on the theory presented by Martin Gutknecht in [34, 35] and is the result of joint work with Roland Freund and Martin Gutknecht. It has the ability to handle look-ahead blocks of arbitrary size, while requiring the same number of inner products per step as the classical Lanczos algorithm. We also discussed some details of a practical implementation; full details and listings of the FORTRAN codes can be found in [21]. We then showed how to combine the new look-ahead Lanczos algorithm with the quasi-minimal residual approach proposed by Freund in [20] to obtain a new method for solving general non-Hermitian linear systems. The new QMR algorithm is the result of joint work with Roland Freund. Thanks to the underlying Lanczos process, the new algorithm requires little storage and essentially constant work per step. In addition, due to the quasi-minimal property, we were able to obtain several convergence results for the algorithm, relating it to the convergence of the generalized minimal residual algorithm, the optimal method in this class of methods. We also showed how the QMR algorithm relates to the biconjugate gradient algorithm, and how the BCG iterates can be stably recovered from the QMR process. The new algorithm has good numerical properties and shows smooth convergence curves. We also presented some of the details of an implementation of the algorithm; complete details and listings of the FORTRAN codes can be found in [23, 24].
Appendix. Notation List

This appendix is a compilation of the notation used in the thesis, grouped according to the section in which it is introduced. We only list notation used globally; notation used only locally in one section or in definitions does not appear in this listing.

Section 2.1

\( A = \) coefficient matrix, \( \in \mathbb{C}^{N \times N} \), with inverse \( A^{-1} \)

\( b = \) right-hand side, \( \in \mathbb{C}^N \)

\( N = \) dimension of the system

\( n = \) iteration number

\( x_0 = \) initial guess

\( r_0 = \) initial residual \( b - Ax_0 \)

\( e_0 = \) initial error \( A^{-1}b - x_0 \)

\( x_n = \) approximate solution at step \( n \)

\( r_n = \) residual \( b - Ax_n \) at step \( n \)

\( e_n = \) error \( A^{-1}b - x_n \) at step \( n \)

\( q_n(z) = \) iteration polynomial, \( x_n = x_0 + q_n(A)r_0 \)

\( p_n(z) = \) residual polynomial, \( p_n(z) = 1 - zq_n(z), \ r_n = p_n(A)r_0 \)

\( I = N \times N \) identity matrix

\( \| \cdot \|_2 = \) 2-norm of a vector or a matrix

\( u_j = \) a right eigenvector of \( A \)

\( \lambda_j = \) an eigenvalue of \( A \)

\( U = \) matrix of right eigenvectors of \( A \)

\( D = \) diagonal matrix of eigenvalues of \( A \)

\( \Lambda = \) spectrum of \( A \)

\( \Lambda_\epsilon(A) = \epsilon \)-pseudospectrum of \( A \)

\( (zI - A)^{-1} = \) resolvent of \( A \)

\( \|p\|_S = \) for a polynomial \( p \) and a compact set \( S \in \mathbb{C}, \max_{z \in S} |p(z)| \)

\( \kappa(A) = \) 2-norm condition number of \( A \)
Section 2.2

\( K_n(v, A) = \) nth Krylov space generated by \( A \) and \( v \)

\[ v_1 = \text{starting basis vector for } K_n(r_0, A) \]

\[ v_n = \text{nth basis vector for } K_n(r_0, A), v_n \in K_n(v_1, A) \setminus K_{n-1}(v_1, A) \]

\[ V^{(n)} = [v_1 \cdots v_n] \]

\[ \beta = \frac{\|r_0\|_2}{\|v_1\|_2} \]

\( h_{jn} = \) coefficient of \( v_j \) in the recurrence for \( v_{n+1} \)

\( H^{(n)} = (h_{ij}), i, j = 1, \ldots, n \)

\( H_e^{(n)} = (n + 1) \times n \) Hessenberg matrix of recurrence coefficients for the \( v_j \)’s

\[ z_n = \text{coefficients of the correction to } x_0 \text{ in the basis } V^{(n)}, x_n = x_0 + V^{(n)} z_n \]

\[ d^{(n)} = [\beta \ 0 \cdots 0]^T \in \mathbb{R}^n \]

Section 2.3

\( I_n = n \times n \) identity matrix

\( Q^{(n)} = \) orthogonal matrix from the QR decomposition of \( H_e^{(n)} \) and \( H^{(n+1)} \)

\( R^{(n)} = \) upper triangular matrix from the QR decomposition of \( H^{(n)} \)

\( R_e^{(n)} = \) upper triangular matrix from the QR decomposition of \( H_e^{(n)} \)

\( v_{i:j} = \) elements \( i \) through \( j \) of vector \( v \)

\( L = \) termination index of the Arnoldi process

Section 2.4

\[ w_1 = \text{starting basis vector for } K_n(w_1, A^T) \]

\[ w_n = \text{nth basis vector for } K_n(w_1, A^T), w_n \in K_n(w_1, A^T) \setminus K_{n-1}(w_1, A^T) \]

\[ W^{(n)} = [w_1 \cdots w_n] \]

\[ \delta_j = w_j^T v_j \]

\( h_{jn} = \) coefficient of \( w_j \) in the recurrence for \( w_{n+1} \)

\( \tilde{H}^{(n)} = (\tilde{h}_{ij}), i, j = 1, \ldots, n \)

\( H_e^{(n)} = (n + 1) \times n \) Hessenberg matrix of recurrence coefficients for the \( w_j \)’s

\[ \alpha_n = \text{coefficient of } v_n \text{ and } w_n \text{ in the recurrence for } v_{n+1} \text{ and } w_{n+1} \]

\[ \beta_n = \text{coefficient of } v_{n-1} \text{ and } w_{n-1} \text{ in the recurrence for } v_{n+1} \text{ and } w_{n+1} \]

\( D^{(n)} = \) biorthogonality matrix \( (W^{(n)})^T (V^{(n)}) \)
Section 2.5

\( \mathcal{P}_n = \text{space of complex polynomials of degree at most } n \)

\( \gamma_j = \text{polynomial coefficient in front of the term } z^i \)

\( \Psi_n = \text{Lanczos or Arnoldi polynomial for } v_{n+1} \text{ or } w_{n+1}, \in \mathcal{P}_n \)

\( \langle \cdot, \cdot \rangle_A = \text{Arnoldi inner product} \)

\( \langle \cdot, \cdot \rangle_L = \text{Lanczos inner product} \)

\( \overline{\Psi}_n = \text{polynomial obtained from } \Psi_n \text{ by conjugating only the coefficients} \)

\( \zeta^{(A)} = \text{coefficients of the decomposition of } v_1 \text{ in the eigenvectors of } A \)

\( \zeta^{(H)} = \text{coefficients of the decomposition of } w_1 \text{ in the eigenvectors of } A^H \)

\( \zeta^{(T)} = \text{coefficients of the decomposition of } w_1 \text{ in the eigenvectors of } A^T \)

Section 3.1

\( L_r = \text{grade of } v_1 \text{ with respect to } A \)

\( L_l = \text{grade of } w_1 \text{ with respect to } A^T \)

\( L_* = \min(L_l, L_r) \)

\( L = \text{termination index of the Lanczos process} \)

Section 3.2

\( \mu_j = j\text{th moment or Schwarz constant, } \mu_j = w_1^T A^j v_1 \)

\( M^{(n)} = (n+1) \times (n+1) \text{ Hankel matrix of moments} \)

\( K^{(n)} = \text{matrix of Krylov vectors } [v_1 \ A v_1 \ \cdots \ A^{n-1} v_1] \)

\( \hat{K}^{(n)} = \text{matrix of Krylov vectors } [w_1 \ A^T w_1 \ \cdots \ (A^T)^{n-1} w_1] \)

\( C^{(n)} = n \times n \text{ upper triangular matrix with the coefficients of } \Psi_{n-1} \text{ in the } n\text{th column} \)

Section 3.3

\( n_j = \text{index for which a regular FOP } \Psi_{n_j-1} \text{ exists} \)

\( J = \text{highest index for which a regular FOP } \Psi_{n_J-1} \text{ exists, } n_J \leq L_* \)

\( j = \text{generic counter for the blocks} \)

\( l = \text{index of the current block, containing the last regular Lanczos vectors} \)

\( h_j = \text{size of the } j\text{th block} \)

\( V_j = \text{matrix containing the Lanczos right vectors from the } j\text{th block} \)
$W_j =$ matrix containing the Lanczos left vectors from the $j$th block

$\delta_j = W_j^T V_j$, diagonal block of $D^{(n)}$ corresponding to the $j$th block

$\tilde{v}_n =$ intermediate Lanczos right vector

$\tilde{w}_n =$ intermediate Lanczos left vector

$\hat{v}_n =$ final scaled Lanczos right vector

$\hat{w}_n =$ final scaled Lanczos left vector

$\rho_n = \frac{\|\hat{v}_n\|_2}{\|\tilde{v}_n\|_2}

\xi_n = \frac{\|\hat{w}_n\|_2}{\|\tilde{w}_n\|_2}$

Section 3.6

$\zeta_n =$ coefficient of $v_n$ and $w_n$ in the inner recurrence for $v_{n+1}$ and $w_{n+1}$

$\eta_n =$ coefficient of $v_{n-1}$ and $w_{n-1}$ in the inner recurrence for $v_{n+1}$ and $w_{n+1}$

$\alpha_j =$ diagonal block of $H^{(n)}$ from the recurrence for the $j$th block

$\beta_j =$ superdiagonal block of $H^{(n)}$ from the recurrence for the $j$th block

$\gamma_j =$ subdiagonal block of $H^{(n)}$ from the recurrence for the $j$th block

Section 3.9

$n(A) =$ factor used in the checks for the look-ahead Lanczos algorithm

Section 4.1

$\omega_j =$ positive scaling factor introduced at step $j - 1$

$\Omega^{(n)} =$ diagonal matrix of scaling factors at step $n$

$\tilde{r}_n =$ norm of the QMR least-squares problem at step $n - 1$

$t^{(n)} =$ first $n$ elements of $\omega_1 Q^{(n)} d^{(n+1)}$

$\tau_j =$ $j$th element of $t^{(n)}$, $j \leq n$

$G_n =$ Givens rotation at step $n$

$c_n =$ cosine of $G_n$, $c_n \in \mathbb{R}$, $c_n > 0$

$s_n =$ sine of $G_n$, $s_n \in \mathbb{C}$

$P_j =$ QMR search directions

$P^{(n)} = [p_1 \cdots p_n]$

$P_j =$ matrix containing the QMR search directions from the $j$th block
Section 4.5

\[ Y^{(n+1)} = \tilde{Y}^{(n+1)} (\Omega^{(n)})^{-1} (Q^{(n)})^H \]

\[ y_j = j^{th} \text{ column of } Y^{(n)} \]

\[ \tilde{y}_j = \text{ auxiliary vector from the recurrence for the } y_j \text{'s} \]
References


