UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

# Conjugate Gradient-type Product Methods for Solving Nonsymmetric Linear Systems (Ph.D. Thesis)

Theodore L. Doug Szeto

June 1994

CAM Report 94-19

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

UNIVERSITY OF CALIFORNIA

Los Angeles

# Conjugate Gradient-type Product Methods for Solving Nonsymmetric Linear Systems

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Mathematics

by

Theodore L. Doug Szeto

1994

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

The research presented in this dissertation was supported by grants from the Office of Naval Research N00014-90-J-1695, N00014-92-J-1890, the Department of Energy DE-FG03-87ER25037, the National Science Foundation ASC90-03002, ASC92-01266, the Army Research Office DAAL03-88-K-0085, DAAL03-91-G-150, the Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration and the University Space Research Association, and the U.S. Department of Education Fellowship.

## ABSTRACT OF THE DISSERTATION

Conjugate Gradient-type Product Methods

for Solving Nonsymmetric Linear Systems

by

Theodore L. Doug Szeto

Doctor of Philosophy in Mathematics

University of California, Los Angeles, 1994

Professor Tony Chan, Chair

The classical Conjugate Gradient (CG) method [39] is a powerful iterative method for solving linear systems where the coefficient matrix in symmetric and positive definite, especially when combined with a preconditioner. It is optimal because it uses short recurrences and minimizes the residual at each iteration. Many applications, however, involve nonsymmetric coefficient matrices. The Biconjugate Gradient (BCG) algorithm [44, 24] is the natural extension of CG to nonsymmetric matrices.

Unfortunately, at each step, the BCG method requires a multiplication with the transpose of the coefficient matrix, which may be too difficult to perform or not even available in practice. In any case, this multiplication presents an extra cost per iteration as compared with CG. Moreover, BCG also suffers from wild, oscillatory residual convergence behavior, and possible numerical breakdown. In this dissertation, we present new algorithms which fall in a class of methods, *product methods*, that solves nonsymmetric linear systems without requiring the matrix transpose while curing some of the other problems.

For instance, since BCG does not minimize the residual as CG does due to the fact that the residuals satisfy a Galerkin condition, its convergence is typically erratic. We combine the smoothing properties of the Quasi-minimal Residual (QMR) method [29] with product methods to obtain "transpose-free" methods which do not have the erratic convergence of BCG, thus increasing numerical stability. We will introduce Quasi-minimal Squared (QMRS) and two QMR variants of the Bi-CGSTAB method (QMRCGSTAB and QMRCGSTAB2).

Another problem for BCG is that it can suffer numerical breakdowns from divisions by 0. Recently, the Composite Step technique [5, 6] has been developed to handle one of two types of breakdown in BCG. We extend this technique to transpose-free methods in the following methods: CSCGS, CS-CGSTAB, and CS-CGSTAB2.

Numerical experiments show that the new methods do produce improved performance over existing algorithms. Also, the new methods require only minor changes to the existing algorithms.

Furthermore, we extend the "best approximation" result in [5] to obtain convergence proofs for the product methods CGS, Bi-CGSTAB, and their composite

step stabilized counterparts.

# CHAPTER 1

# Introduction

In this chapter, we give a description of the general problem to be solved (Section 1.1) and we describe the class of Krylov subspace iterative methods (Section 1.2), from which we will be developing algorithms to solve the above mentioned problem. These new algorithms, their performance and their theoretical aspects, are among the contributions to this thesis, all of which will be summarized in Section 1.3.

## 1.1 The Problem

One of the most frequently encountered tasks in numerical computation is the solution of the system of linear equations

$$Ax = b. \tag{1.1}$$

In many cases, the $N \times N$ coefficient matrix $A$ is large, but sparse. Such systems arise from many applications; for example, finite difference or finite element approximations to partial differential equations. More specifically, in computational fluid dynamics, whenever global dependence is inherent in the physical system (for example, elliptic-type problems), one encounters a set of equations like (1.1). These include problems involving almost incompressible flow, implicit time marching schemes, convection-diffusion problems, and Helmholtz problems, to name a few.

Methods to solve these problems fall into two main categories: *Direct solvers* and *Iterative solvers.* Direct methods involve the factorization of the matrix $A$, e.g., via Gaussian Elimination, then solving the resulting triangular systems. These methods are very robust, but unfortunately, can be quite costly. In cases where $A$ is large and sparse, the factors could, in fact, be dense and require much storage, thus making these solvers extremely impractical. For example, 3-D simulations give rise to large problems which oftentimes cannot be solved practically using direct methods.

The alternative is to solve (1.1) using an iterative scheme. This is advantageous because these techniques can exploit the sparsity or special structure of $A$ by involving it only in matrix-vector products $A \cdot v$ or $A^T \cdot v$. The two main aspects of the iterative solvers we are interested in are:

(a) Developing an effective method from basis vectors chosen from the Krylov subspace, and

(b) Selecting a good preconditioner, usually requiring knowledge of the problem.

In this thesis, we will be focusing on the first aspect, presenting methods to solve an already preconditioned system.

## 1.2 Krylov Subspace Methods

The standard classical iterative methods include Jacobi, Gauss-Seidel, and SOR (for references, see [38, 57, 61]), but in this thesis, we focus on the class of iterative methods which produce iterates from basis vectors generated from Krylov subspaces defined by:

$$
\begin{aligned}
K_n(r_0, A) &= \operatorname{span}\{r_0, Ar_0, A^2 r_0, \dots , A^{n-1} r_0\}; \\
K_n^*(\tilde{r}_0, A) &= \operatorname{span}\{\tilde{r}_0, A^T \tilde{r}_0, (A^T)^2 \tilde{r}_0, \dots , (A^T)^{n-1} \tilde{r}_0\},
\end{aligned}
\tag{1.2}
$$

where $r_0 = b - Ax_0$ is the initial residual for initial guess $x_0$, and $\tilde{r}_0$ is arbitrary. Thus, iterates take on the form $x_n = x_0 + y_n$, where $y_n \in K_n(r_0, A)$. The development of Krylov subspace based methods is one of the most studied problems of computational linear algebra. The advantage of this class of methods is that the subspace provides approximations to the solution $x_n$ by polynomials in $A$ yielding good approximation properties and $N$-step convergence. There has been a lot of activity in this field recently with major advancements via algorithms such as Conjugate Gradients Squared (CGS) [54] and Quasi-minimal Residual (QMR) [29].

Different Krylov subspace iterative methods are characterized by certain requirements on the residual vector $r_n = b - Ax_n$ to satisfy some optimal condition such as

$$(i) \qquad \|r_n\| = \text{minimum in some norm, or}$$

$$(ii) \qquad r_n \perp K_n^*(\tilde{r}_0, A) \text{ (Galerkin condition).}$$

Even though iterates $x_n$ are defined by properties like $(i)$ and $(ii)$, they are not computed directly, but via updates. In most cases, this is done using an updating formula $x_n = x_{n-1} + d_{n-1}$, where $d_{n-1} \in K_n(r_0, A)$. Methods differ only in the way $d_{n-1}$ is chosen from the Krylov subspace.

For a symmetric and positive definite coefficient matrix $A$, the classical Conjugate Gradient algorithm (CG) proposed by Hestenes and Stiefel [39] satisfies property $(i)$ using two-term recurrences. By using short recurrences at each iteration, this method efficiently minimizes work and storage. CG is one of the most powerful iterative methods for solving (1.1), especially when combined with a preconditioning strategy. However, for a general nonsymmetric matrix $A$, Faber and Manteuffel [23] proved that property $(i)$ does not hold (i.e., the residual norm cannot be minimized) with an algorithm using short recurrences.

Thus, if $A$ is nonsymmetric and we wanted an algorithm to satisfy $(i)$, such a method cannot have the short recurrences as does CG. For example, the Generalized Minimum Residual (GMRES) scheme by Saad and Schultz [51] is a very popular method which minimizes the residual by creating an orthonormal basis via the Gram-Schmidt process. However, since storage and work per iteration grow quadratically with each iteration in this algorithm, the computation is expensive and in practice, restarts are required. Hence, GMRES may display slow convergence properties for difficult problems.

An extension of the CG method for nonsymmetric matrices which employs property *(ii)* is the Biconjugate Gradient (BCG) algorithm due to Lanczos [44]. In fact, the BCG method is closely related to the nonsymmetric Lanczos process for computing the basis for the Krylov subspaces $K_n(r_0, A)$ and $K_n^*(\tilde{r}_0, A)$. In BCG, short recurrences are used, thereby minimizing work and storage, but it runs into three additional problems: *(a)* since it is not based on a minimization property, the convergence behavior can be quite erratic with wild oscillations in the residual norm, *(b)* breakdowns - divisions by 0 - can occur, and *(c)* an extra multiplication by the transpose of $A$ is needed at each iteration, increasing the work and also requiring $A^T$.

To handle problem *(a)*, techniques such as the Minimum Residual Smoothing (MRS) algorithm (presented in Schönauer [52], Weiss [60], Zhou and Walker [62]) and the Quasi-minimal Residual (QMR) method (developed by Freund and Nachtigal [29]) have been proposed to minimize and quasi-minimize the residual, respectively, resulting in smoother convergence behavior and improved stability.

As for problem *(b)*, BCG can encounter two types of breakdown: one due to the fact that $x_n$ may not be defined, and another due to a breakdown in the underlying Lanczos process used to define the updates. The QMR method also addresses this problem since it is based on a look-ahead variant of the classical nonsymmetric Lanczos process used to define the updates. There are many methods like this which cure problem *(b)* by looking ahead to the next iterates to see whether or not they are affected by possible breakdowns and handling them accordingly (see e.g., [10, 14, 15, 27, 36, 41, 48]).

Recently, Bank and Chan introduced the Composite Step Biconjugate Gradient (CSBCG) algorithm [5, 6], an alternative which cures one of two possible breakdowns in BCG by skipping over steps for which the BCG iterate is not defined. This is done with a simple modification of BCG which needs only a maximum look-ahead step size of 2 to eliminate the (near) breakdown (assuming that the Lanczos process does not break down) and to smooth the sometimes erratic convergence of BCG. Thus, instead of a more complicated (but less prone to breakdown) version, CSBCG cures only one kind of breakdown, but does so with a minimal modification to the usual implementation of BCG in the hope that its empirically observed stability will be inherited.

Problem *(c)* is that BCG requires 2 matrix-vector multiplications ($A \cdot v$ and $A^T \cdot v$) at each iteration. This is a problem because in many applications, it is often complicated and perhaps not even possible to perform multiplications with the transpose matrix. Furthermore, BCG requires twice as many matrix-vector products than CG to achieve the same degree of the Krylov subspace. Chan, de Pillis and Van der Vorst [16] have developed transpose-free implementations of QMR and BCG, but these algorithms require 3 matrix-vector products per step.

Other methods have been developed which overcome problem *(c)* by computing residuals characterized by a product of the BCG residual polynomial with another polynomial of equal degree. In 2 matrix-vector multiplies (both with $A \cdot v$) per iteration, these methods compute two degrees of the Krylov subspace with iterate $x_n \in K_{2n}(r_0, A)$. Hence, we term the class of these transpose-free methods *product methods*. Examples of these include methods due to Sonneveld [54], Van der Vorst [55], Gutknecht [35], Freund [26], and Sleijpen and Fokkema [53]. All of these product methods, with the exception of the first one mentioned, handle in some way the problem of erratic convergence. Still other product methods have been developed which cure breakdowns (e.g., [9, 30]).

## 1.3 Overview of the Thesis

In this thesis, we focus on the class of product Krylov subspace methods for solving nonsymmetric linear systems. New methods are presented which not only eliminate the need for a multiplication with $A^T$, but also smoothes the residuals and handles some breakdowns. We also prove theorems of convergence for some standard product methods.

Specifically, after a review of existing methods in Chapter 2, we present the Quasi-minimal Residual Squared (QMRS) algorithm. As mentioned earlier, the QMR method overcomes some of the problems of BCG stabilizing the residual convergence but unfortunately requires the multiplication of $A^T$. Thus, in the same way the Conjugate Gradients Squared (CGS) algorithm [54] squares the BCG residual to overcome $A^T$ products, we square a non-look ahead version of the QMR residual to formulate the QMRS algorithm. Since it is based on the QMR polynomial, QMRS is more stable than CGS. Unfortunately, each iteration of QMRS costs an extra matrix-vector multiplication (3 instead of 2 for CGS). Note, however, that this is an improvement over BCG and QMR because in one iteration of QMRS, we advance two degrees in the Krylov subspace.

In Chapter 4, we present another transpose-free method which also attempts to stabilize the erratic convergence of CGS without the extra matrix-vector product in QMRS. This is done by applying the quasi-minimization idea of the QMR method to the Bi-CGSTAB algorithm [55], a stabilized version of CGS. We introduce this algorithm, QMRCGSTAB, along with some of its variants and we show that these do in fact yield smoother residuals and improved convergence behavior.

Next, we consider the problem of breakdowns in BCG. Chapter 5 reviews the details of the composite step technique [5, 6] as it is applied to the BCG algorithm to cure one of the two possible causes of numerical instability. Transpose-free methods like CGS and Bi-CGSTAB have similar breakdown problems since they involve products with the BCG polynomial. The composite step technique, then, can also be applied to these product methods to overcome the analogous problem of breakdown.

Chapters 6 and 7 give the details for CSCGS and CS-CGSTAB, the methods where composite step is applied to CGS and Bi-CGSTAB, respectively. Chapter 7 also presents a variant, CS-CGSTAB2, which cures an additional potential breakdown due to the Bi-CGSTAB process in the case of skew-symmetric $A$, thus further improving the stability of these composite step product methods. These composite step methods employ a stepping strategy which allows the step size to vary and does not require user specified tolerance parameters. Furthermore, this is done with only a minimal modification of the existing methods, We remark that 5 matrix-vector products are required per composite step as opposed to 4 in two steps of the standard product methods CGS and Bi-CGSTAB. Note that this is still an improvement over QMRS, where 3 multiplications are required at every step.

As for the theoretical aspects of the convergence of these methods, a recent advancement was the proof of a "best approximation" result for BCG, due to Bank and Chan [6]. Until then, there has been very little theory known on the convergence of the Biconjugate Gradient algorithm or other related methods. In Chapter 8, we show how we can extend this result to prove convergence of product methods CGS, Bi-CGSTAB, and their composite step counterparts, since all of these methods involve the BCG polynomial. Finally, conclusions are presented in Chapter 9.

Each of the new methods in this thesis improves on the standard BCG algorithm, which is widely used and noted for its simplicity and good convergence properties. While maintaining these advantages of BCG, we overcome the problems in BCG that were mentioned and prove convergence as well. In this thesis, the problem of erratic convergence is handled via quasi-minimization and the MRS technique in Chapters 3, 4 and 6. The problem of breakdowns is addressed in Chapters 5 - 7, and all of the new algorithms overcome the problem of multiplying by $A^T$ since all are considered product methods.

# CHAPTER 2

# Nonsymmetric Conjugate Gradient Methods

We now survey several of the nonsymmetric Conjugate Gradient methods mentioned in Chapter 1. We present details on the methods which will be referred to in the subsequent chapters. For more complete overviews, refer to [3, 7, 22].

## 2.1 Minimization Methods

Suppose we have a nonsymmetric linear system and want to solve it by extending the Conjugate Gradient method in such a way that the residual is minimized over $K_n(r_0, A)$ at step $n$ (Property ($i$), Section 1.2). There are several iterative techniques that have been developed which achieve this.

If the symmetric part of $A$ is positive definite, the Generalized Conjugate Residual (GCR) method and the ORTHODIR method can both solve (1.1) efficiently. (For references, and other methods, see [2, 22, 21, 40].) Unfortunately, stability problems due to round-off error occur in these methods when the symmetric part is no longer positive definite. It is this case which motivated the development of the Generalized Minimum Residual (GMRES) algorithm by Saad and Schultz [51].

### 2.1.1 The Generalized Minimum Residual (GMRES) Algorithm

GMRES is based on the Arnoldi process [1] for computing an orthonormal basis of the Krylov subspace $K_n(r_0, A)$. The Arnoldi algorithm uses (modified) Gram-Schmidt orthogonalization to form at the $n$-th step the system of basis vectors

$$V_n = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \tag{2.1}$$

and the $(n+1) \times n$ upper Hessenberg matrix $\bar{H}_n$ which satisfy the relation:

$$AV_n = V_{n+1}\bar{H}_n. \tag{2.2}$$

Recall that the 2-norm of the residual $||r_n|| = ||b - Ax_n||$ is to be minimized. Thus, one needs to find iterates $x_n = x_0 + V_n y_n$, where $y_n$ is chosen to solve the least squares problem

$$\min ||b - Ax_n|| = \min_{y \in \mathrm{R}^n} ||r_0 - AV_n y||. \tag{2.3}$$

If we begin the orthogonalization with $v_1 = r_0/||r_0||$, the quantity to be minimized can be written:

$$||\beta v_1 - V_{n+1}\bar{H}_n y|| = ||V_{n+1}(\beta e_1^{(n+1)} - \bar{H}_n y)||,$$

where $\beta = ||r_0||$ and $e_1^{(n+1)} := \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T \in \mathrm{R}^{n+1}$. Since $V_{k+1}$ is orthonormal, the least squares problem (2.3) can be reduced to:

$$\min_{y \in \mathrm{R}^n} ||\beta e_1^{(n)} - \bar{H}_n y||. \tag{2.4}$$

The solution vector $y_n$ can be updated by updating the QR-factorization of $\bar{H}_n$ progressively. This, in turn, can be used to update the GMRES iterate $x_n$. (See

[51] for details.) This gives rise to the GMRES algorithm, as presented in Table 2.1.

Table 2.1: **Algorithm GMRES**

```
    Choose $x_0$ and compute $r_0 = b - Ax_0$
    Set $v_1 = r_0/||r_0||$
For $j = 1, 2, \ldots, n$ *until convergence*
    $h_{i,j} = (Av_j, v_i), \quad i = 1, 2, \ldots, j$
    $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
    $h_{j+1,j} = ||\hat{v}_{j+1}||$
    $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
    Update $y_n$ to solve (2.4)
    $x_n = x_0 + V_n y_n$,
```

As $n$ increases, note that the amount of storage vectors required grows like $n$ and the number of multiplies grows like $\frac{1}{2}\, n^2 N$. This is not efficient in practice, especially when $A$ is large, unless $A$ is well conditioned (e.g., when there is a good preconditioner), so GMRES is usually implemented with a truncated version, GMRES($m$), which restarts the algorithm every $m$ steps. This means that after $m$ steps, the iteration begins anew using the recently computed data as initial values, no longer requiring storage of the old values. By fixing $m$, one can limit the costs for computation and storage.

Unfortunately, a good choice for $m$ is difficult to find. If $m$ is too small, the method may converge too slowly, or may not converge at all. On the other hand, if $m$ is too large, it could result in costly and inefficient work and storage. For more details and extensions of this method, we refer to studies by Saad [49], Vuik [58], and Walker [59].

## 2.2 Galerkin Methods

Instead of minimization, an alternative requirement on the residual is to apply a Galerkin orthogonality condition on it (e.g., Property (*ii*), Section 1.2). Specifically, we impose the condition

$$r_n \perp K_n^*(\tilde{r}_0, A) \tag{2.5}$$

on $r_n = b - Ax_n$, where $K_n^*(\tilde{r}_0, A)$ is as defined in (1.2). Doing this will eliminate the long recurrences in methods like GMRES, thereby decreasing work and storage.

If we define $\bar{K}_n, \bar{K}_n^*$ to be matrices whose columns are as given in (1.2) and span the Krylov spaces $K_n(r_0, A)$ and $K_n^*(\tilde{r}_0, A)$, condition (2.5) implies that iterates of the form $x_n = x_0 + \bar{K}_n v_n$ are defined by the solution to the linear system $(\bar{K}_n^*)^T A \bar{K}_n v_n = (\bar{K}_n^*)^T r_0$. We note that the iterate $x_n$ exists whenever the Hankel moment matrix

$$(2.6) \qquad H_n^{(1)} = (\bar{K}_n^*)^T A \bar{K}_n = \begin{pmatrix} \mu_1 & \mu_2 & \cdots & \mu_n \\ \mu_2 & \mu_3 & \cdots & \mu_{n+1} \\ \vdots & \vdots & & \vdots \\ \mu_n & \mu_{n+1} & \cdots & \mu_{2n-1} \end{pmatrix},$$

where $\mu_i \equiv \tilde{r}_0^T A^i r_0$, is nonsingular.

The Biconjugate Gradient (BCG) algorithm [44, 24], a natural extension of Conjugate Gradient for nonsymmetric systems, is one such Galerkin method. The BCG iterates are computed from bi-orthogonal basis vectors closely related to the nonsymmetric Lanczos process [43]. In this section, we first give an overview of the Lanczos process, then describe how the BCG iterates are obtained from this. We will also discuss possible breakdowns in the BCG algorithm.

### 2.2.1 The Nonsymmetric Lanczos Algorithm

The nonsymmetric Lanczos process [43] was originally proposed to reduce an arbitrary matrix $A$ to tridiagonal form. Starting with $v_0 = r_0$ and an arbitrary nonzero vector $w_0 \in \mathrm{R}^N$, this process generates two sequences of vectors $v_0, v_1, \ldots,$ and $w_0, w_1, \ldots,$ such that, for $n = 1, 2, \ldots,$

$$(2.7) \qquad \mathrm{span}\{v_0, v_1, \ldots, v_{n-1}\} = K_n(r_0, A),$$

$$(2.8) \qquad \mathrm{span}\{w_0, w_1, \ldots, w_{n-1}\} = K_n(w_0, A^T).$$

It does this via the three-term recurrences:

$$v_{n+1} = Av_n - \alpha_n v_n - \beta_n v_{n-1} \tag{2.9}$$

$$w_{n+1} = A^T w_n - \alpha_n w_n - \beta_n w_{n-1}, \tag{2.10}$$

where

$$\alpha_n = \frac{(w_n, Av_n)}{(w_n, v_n)} \quad \text{and} \quad \beta_n = \frac{(w_n, v_n)}{(w_{n-1}, v_{n-1})}$$

are chosen so that we have the following bi-orthogonality condition:

$$w_i^T v_j = 0, \quad \text{if } i \neq j.$$

We can see that this algorithm will break down if $(w_n, v_n) = 0$ but $v_n \neq 0$, $w_n \neq 0$. We term this the *Lanczos breakdown.* Methods have been developed to overcome this by "looking-ahead", i.e., if one forsees breakdowns, then successive vectors can be computed and block bi-orthogonality conditions imposed [10, 14, 15, 27, 41, 48]. (See Section 5.2 for more details.)

The basis vectors generated can be arranged into the matrices:

$$V_n = \begin{bmatrix} v_0 & v_1 & \cdots & v_{n-1} \end{bmatrix}, \tag{2.11}$$

$$W_n = \begin{bmatrix} w_0 & w_1 & \cdots & w_{n-1} \end{bmatrix}.$$

Thus, the recurrence relations (2.9), (2.10), can be written compactly in matrix form as follows:

$$AV_n = V_n T_n + v_n e_{n+1}^T, \tag{2.12}$$

$$A^T V_n = W_n T_n + w_n e_{n+1}^T,$$

where $T_n$ is the $n \times n$ tridiagonal matrix

$$T_n = \begin{bmatrix} \alpha_0 & \beta_1 & & & \\ 1 & \alpha_1 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & \alpha_{n-2} & \beta_{n-1} \\ & & & 1 & \alpha_{n-1} \end{bmatrix},$$

and $e_{n+1}^T = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \in \mathrm{R}^{n+1}$. Equation (2.12) can be rewritten:

$$AV_n = V_{n+1} T_n^{(e)}, \tag{2.13}$$

where

$$T_n^{(e)} = \begin{bmatrix} T_n \\ e_{n+1}^T \end{bmatrix}.$$

Note that $V_n = K_n(v_0, A)$ and $W_n = K_n^*(w_0, A)$ and also,

$$W_n^T V_n = D_n = \mathrm{diag}(d_1, \cdots, d_n),$$

where $d_i = w_i^T v_i$.

These basis vectors can now be used in computing the solution of (1.1). Solution iterates are of the form

$$x_n = x_0 + V_n y_n, \quad \text{where } y_n \in \mathrm{R}^n.$$

The corresponding residual is as follows:

$$r_n = b - Ax_n = r_0 - AV_ny_n \tag{2.14}$$

$$= r_0 - V_{n+1}T_n^{(e)}y_n \tag{2.15}$$

$$= V_{n+1}(e_1^{(n+1)} - T_n^{(e)}y_n), \tag{2.16}$$

where $e_1^{(n+1)}$ is the first vector of the canonical basis. Imposing the bi-orthogonality of the bases gives:

$$0 = W_n^T r_n = W_n^T V_{n+1}(e_1^{(n+1)} - T_n^{(e)}y_n).$$

Hence, $y_n$ satisfies the equation:

$$(W_n^T V_n)T_ny_n = (w_1^T v_1)e_1^{(n)}.$$

If the matrix $W_n^T V_n$ is nonsingular, then we can find $y_n$ by solving

$$T_ny_n = e_1^{(n)}, \tag{2.17}$$

and update the solution $x_n$.

Note that the singularity of $W_n^T V_n$ corresponds directly to the Lanczos breakdown discussed earlier. In terms of the associated Krylov subspaces given in (1.2), we see that the Lanczos process will break down when the Hankel moment matrix $H_n^{(0)}$ [36] defined by:

$$H_n^{(0)} \equiv \bar{K}_n^*(\tilde{r}_0)\bar{K}_n(r_0) = \begin{pmatrix} \mu_0 & \mu_1 & \cdots & \mu_{n-1} \\ \mu_1 & \mu_2 & \cdots & \mu_n \\ \vdots & \vdots & & \vdots \\ \mu_{n-1} & \mu_n & \cdots & \mu_{2n-2} \end{pmatrix},$$

is singular. ($\bar{K}_n$, $\bar{K}_n^*$, $\mu_i$, are as defined at the beginning of this section.)

### 2.2.2 The Biconjugate Gradient (BCG) Algorithm

The BCG iterate $x_n^{BCG} = x_0 + V_n y_n$ is obtained from solving the tridiagonal system (2.17) by computing the LU factors of $T_n$ without pivoting. This gives the bidiagonal factors $L_n$ and $U_n$ which implies that the iterate $x_n^{BCG} = (V_n U_n^{-1})(L_n^{-1} e_1^{(n)})$ can be updated using two-term recurrences. Thus, $v_i$ and $w_i$ do not need to be stored.

In general, the LU decomposition (without pivoting) may not always exist. If we run into a zero pivot, this will cause a breakdown in the solution of (2.17). Hence, we term this breakdown the *pivot breakdown.* In terms of the Krylov subspaces (1.2), it is easily shown that the pivot breakdown corresponds to the singularity of the Hankel moment matrix $H_n^{(1)}$ (2.6). There are several ways of handling this breakdown, which will be discussed in further detail in Section 5.2.

The BCG algorithm, thus, consists of generating pairs of iterates: $x_n$, which will solve (1.1), and $\tilde{x}_n$, for the auxiliary system, $A^T \tilde{x} = \tilde{b}$. The corresponding residuals satisfy the Galerkin conditions:

$$r_n \perp K_n^*(\tilde{r}_0, A); \qquad \tilde{r}_n \perp K_n(r_0, A), \tag{2.18}$$

and we have shown how the iterates can be updated via two-term recurrences.

We give the standard implementation of BCG in Table 2.2. From this table,

Table 2.2: **Algorithm BCG**

```
    Choose $x_0, \tilde{x}_0$ and compute $r_0 = b - Ax_0, \tilde{r}_0 = \tilde{b} - A^T\tilde{x}_0$
    Set $p_0 = r_0; \tilde{p}_0 = \tilde{r}_0; \rho_0 = \tilde{r}_0^T r_0$
For $n = 0, 1, 2, \dots$, until convergence
    $\sigma_n = \tilde{p}_n^T A p_n;$    $\alpha_n = \rho_n / \sigma_n$
    $r_{n+1} = r_n - \alpha_n A p_n;$    $\tilde{r}_{n+1} = \tilde{r}_n - \alpha_n A^T \tilde{p}_n$
    $x_{n+1} = x_n + \alpha_n p_n;$    $\tilde{x}_{n+1} = \tilde{x}_n + \alpha_n \tilde{p}_n$
    $\rho_{n+1} = \tilde{r}_{n+1}^T r_{n+1};$    $\beta_{n+1} = \rho_{n+1} / \rho_n$
    $p_{n+1} = r_{n+1} + \beta_{n+1} p_n;$    $\tilde{p}_{n+1} = \tilde{r}_{n+1} + \beta_{n+1} \tilde{p}_n$
```

we can see the two possible kinds of numerical breakdowns discussed: (1) $\sigma_n = 0$ *(pivot breakdown)*, and (2) $\rho_n = 0$, but $r_n \neq 0$ *(Lanczos breakdown)*. [1] Although such exact breakdowns are very rare in practice, near breakdowns can cause severe numerical instability.

As mentioned earlier, there are other drawbacks to the BCG algorithm: irregular convergence behavior since the residual is not being minimized, and two matrix-vector multiplications (by $A$ and its transpose $A^T$) are required at each it-

[1] In other literature, what we term the *pivot* and *Lanczos* breakdowns, are also known as *true* and *ghost* breakdowns [11], *Galerkin* and *serious Lanczos* breakdowns [29], *hard* and *soft* breakdowns [41].

eration to gain only one degree of the Krylov subspace. In addition, until recently, there has been little theory known about the convergence of the BCG method. Hence, developing methods to overcome these problems has been of great interest in this field. In this thesis, we will address all of these problems in the development of new algorithms and theory.

## 2.3 Quasi-minimization Methods

The Quasi-minimal Residual (QMR) method, due to Freund and Nachtigal [29], offers another alternative for characterizing the residual of a Krylov subspace method. QMR is also based on the nonsymmetric Lanczos process (Section 2.2.1) for computing a bi-orthogonal basis for the associated Krylov subspaces (1.2).

Note that the implementation of the nonsymmetric Lanczos process in [29] is presented with look-ahead in order to handle the curable Lanczos breakdowns. In this thesis, we will only consider the non-look-ahead version of the Lanczos process (and QMR).

Assuming that this process yields scaled Lanczos vectors $v_j$ which are normalized to have unit length:

$$\|v_j\| = 1, \quad j = 0, 1, 2, \dots , \tag{2.19}$$

QMR uses the quantities $V_{n+1}$ and $T_n^{(e)}$ generated from the look-ahead Lanczos process to produce iterates:

$$x_n = x_0 + V_n z_n, \tag{2.20}$$

where $z_n \in \mathrm{R}^n$ is the solution of the least squares problem similar to GMRES:

$$\|d_1^{(n+1)} - \Omega_{n+1} T_n^{(e)} z_n\| = \min_{z \in \mathrm{R}^n} \|d_1^{(n+1)} - \Omega_{n+1} T_n^{(e)} z\|. \tag{2.21}$$

Here,

$$d_1^{(n+1)} = \omega_1 \|r_0\| \cdot \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T \in \mathrm{R}^{n+1}, \tag{2.22}$$

and

$$\Omega_{n+1} = \mathrm{diag}(\omega_1, \omega_2, \ldots, \omega_{n+1}), \quad \omega_j > 0, \quad j = 1, 2, \ldots, n+1, \tag{2.23}$$

is a still arbitrary scaling matrix. Usually, one chooses unit weights

$$\omega_j \equiv 1, \text{ for all } j. \tag{2.24}$$

Note that, by (2.20) and (2.13), the residual corresponding to the QMR iterate $x_n$ is given by

$$r_n = V_{n+1} \Omega_{n+1}^{-1} \left( d_1^{(n+1)} - \Omega_{n+1} T_n^{(e)} z_n \right). \tag{2.25}$$

Hence, in view of (2.21), the QMR iterates are characterized by a minimization of the second factor in (2.25); this is the quasi-minimal residual property. Note that, by (2.19), the scaling (2.24) is very natural, in the sense that all the columns of $V_{n+1}\Omega_{n+1}^{-1}$ in the representation (2.25) of $r_n$ are treated equally. Note also that the

QMR iterates can be easily updated from step to step because the QR factorization of $\Omega_{n+1}T_n^{(e)}$ can be updated.

Quasi-minimizing will smooth the residual convergence. Furthermore, it overcomes the pivot breakdown in BCG. In cases where BCG does not break down, the convergence of QMR is similar to that of BCG since both methods are generated from the same bi-orthogonal bases. In fact, it can be shown [29] that one can recover the BCG iterates from QMR. However, it is noted that in some cases when BCG stagnates or diverges, QMR may still reduce the residual.

## 2.4 Product Methods

One major drawback of QMR and BCG is that they require a multiplication with the matrix $A^T$. In many cases, the transpose matrix is not readily available or the multiplication is difficult to perform, perhaps due to storage constructs. Hence, there is a need for methods which are *transpose-free.*

Freund and Zha [32] have shown that, in principle, the transpose can always be eliminated by choosing special starting vectors. However, this approach is practical only for special cases, as, e.g., complex symmetric matrices, and in general there is a need for transpose-free BCG-like schemes.

Two such methods are the Transpose-free Implementations of QMR (TFiQMR) and of BCG (TFiBCG) [16] by Chan et al. These methods recover the QMR and BCG iterates without requiring multiplications with $A^T$. However, they require 3 matrix-vector multiplications per iteration.

In fact, another drawback of the standard BCG and QMR algorithms is that they require 2 matrix-vector products per iteration. Although this is less than TFiBCG and TFiQMR, it is still twice as many products as needed in an iteration of CG to advance one degree of the Krylov subspace.

As mentioned before, product methods are a class of methods which handle these problems. We first note that the Krylov subspace $K_n(r_0, A)$ can be written

$$K_n(r_0, A) = \{\Phi(A)r_0 | \Phi \in \mathcal{P}_{n-1}\},$$

where $\mathcal{P}_n$ is the set of all polynomials of degree at most $n$. In view of this, the residual vector $r_n = b - Ax_n$ of any iterate $x_n \in x_0 + K_n(r_0, A)$ is of the form $r_n = \Phi(A)r_0$, where $\Phi \in \mathcal{P}_n$ and $\Phi(0) = 1$. In particular, the BCG residual can be written

$$r_n^{BCG} = \phi_n(A)r_0, \text{ where } \phi_n \in \mathcal{P}_n, \;\; \phi_n(0) = 1. \tag{2.26}$$

Product methods, then, are algorithms characterized by residuals formed from the product of the BCG polynomial with another polynomial of equal degree:

$$r_n^{Product} = \tau_n(A)\phi_n(A)r_0, \text{ where } \tau_n, \phi_n \in \mathcal{P}_n, \;\; \tau_n(0), \phi_n(0) = 1. \tag{2.27}$$

We next give details on the product methods CGS and Bi-CGSTAB, and survey other existing product methods.

### 2.4.1 The Conjugate Gradient Squared (CGS) Algorithm

The Conjugate Gradient Squared (CGS) algorithm, due to Sonneveld [54], is a product method in which the polynomial $\tau_n$ in (2.27) is the BCG polynomial $\phi_n$ itself. CGS produces iterates $x_n$ in which the corresponding residual satisfies

$$r_n = \phi_n^2(A)r_0. \tag{2.28}$$

This can be done by using the polynomial update formulas for the BCG residual and search direction from Table 2.2:

$$\phi_{n+1}(\vartheta) = \phi_n(\vartheta) - \alpha_n \vartheta \psi_n(\vartheta), \tag{2.29}$$

$$\psi_{n+1}(\vartheta) = \phi_{n+1}(\vartheta) + \beta_{n+1} \vartheta \psi_n(\vartheta). \tag{2.30}$$

Squaring these yields:

$$\phi_{n+1}^2(\vartheta) = \phi_n^2 - 2\alpha_n \vartheta \phi_n \psi_n + \alpha_n^2 \vartheta^2 \psi_n^2, \tag{2.31}$$

$$\psi_{n+1}^2(\vartheta) = \phi_{n+1}^2 + 2\beta_{n+1} \phi_{n+1} \psi_n + \beta_{n+1}^2 \psi_n^2. \tag{2.32}$$

Hence, in order to compute the CGS residual $r_n = \phi_n^2(A)r_0$ and search direction $p_n = \psi_n^2(A)r_0$, the mixed terms $\phi_n\psi_n$, $\phi_{n+1}\psi_n$ must be calculated as well. By noting that $\phi_n\psi_n$ are updated

$$\phi_n\psi_n = \phi_n^2 + \beta_n \phi_n \psi_{n-1},$$

it can, in turn, be used to update

$$\phi_{n+1}\psi_n = \phi_n\psi_n - \alpha_n \vartheta \psi_n^2.$$

We denote $q_n = \phi_n(A)\psi_{n-1}(A)r_0$ and $u_n = \phi_n(A)\psi_n(A)r_0$.

The coefficients $\alpha_n, \beta_n$ in the polynomials (2.31), (2.32), can be computed using the BCG constants $\sigma_n, \rho_n$ (see Table 2.2) which can be recovered via:

$$\rho_n^{BCG} = (\phi_n(A^T)\tilde{r}_0, \phi_n r_0) = (\tilde{r}_0, \phi_n^2 r_0) = \tilde{r}_0^T r_n^{CGS},$$

$$\sigma_n^{BCG} = (\psi_n(A^T)\tilde{r}_0, A\psi_n r_0) = (\tilde{r}_0, A\psi_n^2 r_0) = \tilde{r}_0^T A p_n^{CGS}.$$

Note that in the CGS algorithm (Table 2.3), as with other product methods, with every two matrix-vector products, the Krylov subspace is advanced two degrees. CGS, thus, generally has convergence behavior twice as fast as BCG.

Table 2.3: **Algorithm CGS**

```
    Choose $x_0$, and compute $r_0 = b - Ax_0$
    Pick an arbitrary vector $\tilde{r}_0$
    Set $q_0 = p_{-1} = 0; \rho_{-1} = 1$
For $n = 0, 1, 2, \dots$, *until convergence*
    $\rho_n = \tilde{r}_0^T r_n; \quad \beta_n = \rho_n / \rho_{n-1}$
    $u_n = r_n + \beta_n q_n$
    $p_n = u_n + \beta_n (q_n + \beta_n p_{n-1})$
    $v_n = A p_n$
    $\sigma_n = \tilde{r}_0^T v_n; \quad \alpha_n = \rho_n / \sigma_n$
    $q_{n+1} = u_n - \alpha_n v_n$
    $r_{n+1} = r_n - \alpha_n A(u_n + q_{n+1})$
    $x_{n+1} = x_n + \alpha_n (u_n + q_{n+1})$
```

### 2.4.2 The Biconjugate Gradient Stabilized (Bi-CGSTAB) Algorithm

In the squaring process of CGS, the good behavior of BCG is enhanced, but unfortunately, large residual values are amplified resulting in loss of accuracy. As discussed in [56], the irregular convergence behavior of CGS is due to the fact that if $\phi_n(A)$ is viewed as a reduction operator applied twice to $r_0$, since $\phi_n(A)$ is quite dependent on the initial residual $r_0$, it is not likely to be a reduction for any other vector, not even for $\phi_n(A)r_0$ itself.

The Bi-CGSTAB algorithm [55] due to Van der Vorst attempts to stabilize this by multiplying the BCG polynomial $\phi_n(A)$, instead, by another polynomial of equal degree,

$$\tau_n(A) = (I - \omega_1 A)(I - \omega_2 A) \cdots (I - \omega_n A), \tag{2.33}$$

where the $\omega_i$'s are chosen to locally minimize the residual by a steepest descent method. Thus, by computing the residual $r_n^{Bi-CGSTAB} = \tau_n(A)\phi_n(A)r_0$, we obtain a more smoothly converging algorithm.

This residual can be updated by the following polynomial relationship:

$$\begin{aligned}
\tau_{n+1}\phi_{n+1}(\vartheta) &= (1 - \omega_{n+1}\vartheta)\tau_n\phi_{n+1} \\
&= (1 - \omega_{n+1}\vartheta)\tau_n(\phi_n - \alpha_n\vartheta\psi_n) \\
&= (1 - \omega_{n+1}\vartheta)(\tau_n\phi_n - \alpha_n\vartheta\tau_n\psi_n).
\end{aligned}$$

Similarly, for $p_n^{Bi-CGSTAB} = \tau_n(A)\psi_n(A)r_0$,

$$\begin{aligned} \tau_{n+1}\psi_{n+1}(\vartheta) &= (1-\omega_{n+1}\vartheta)\tau_n(\phi_{n+1}+\beta_{n+1}\psi_n) \\ &= (1-\omega_{n+1}\vartheta)(\tau_n\phi_{n+1}+\beta_{n+1}\tau_n\psi_n). \end{aligned}$$

Recovering the BCG coefficients is a bit more involved than for CGS. By construction, $\phi_{n+1}(A)r_0$ is orthogonal with respect to all vectors $\chi_n(A^T)\tilde{r}_0$ where $\chi_n$ is an arbitrary polynomial of degree at most $n$. Thus,

$$\begin{aligned} \rho_{n+1}^{BCG} &= (\tilde{\phi}_{n+1}(A^T)\tilde{r}_0, \phi_{n+1}(A)r_0) \\ &= \alpha_0\cdots\alpha_{n-1}((-A^T)^{n+1}\tilde{r}_0, \phi_{n+1}(A)r_0), \end{aligned}$$

picking out the coefficient of the highest order term. Similarly,

$$\begin{aligned} \rho_n^{Bi-CGSTAB} &= (\tilde{r}_0, \tau_{n+1}(A)\phi_{n+1}(A)r_0) \\ &= (\tau_{n+1}(A^T)\tilde{r}_0, \phi_{n+1}(A)r_0) \\ &= \omega_1\cdots\omega_{n+1}((-A^T)^{n+1}\tilde{r}_0, \phi_{n+1}(A)r_0), \end{aligned}$$

and the relationship between them follows:

$$\rho_{n+1}^{BCG} = \frac{\alpha_0\cdots\alpha_n}{\omega_1\cdots\omega_{n+1}}\ \rho_{n+1}^{Bi-CGSTAB}, \text{ where } \alpha_n = \frac{\rho_n^{BCG}}{\sigma_n^{BCG}}\ .$$

If we let $\mu_{n+1} = (\alpha_0\cdots\alpha_n)/(\omega_1\cdots\omega_{n+1})$, this reduces to:

$$\rho_{n+1}^{BCG} = \mu_{n+1}\rho_{n+1}^{Bi-CGSTAB}$$

where $\mu_{n+1}$ is updated by:

$$\mu_{n+1} = \mu_n\left(\frac{\rho_n^{BCG}}{\sigma_n^{BCG}\omega_{n+1}}\right).$$

The value $\sigma_{n+1}^{BCG}$ can be similarly derived to yield:

$$\sigma_{n+1}^{BCG} = \mu_{n+1}\sigma_{n+1}^{Bi-CGSTAB}$$

Thus, the coefficients $\alpha_n$ and $\beta_{n+1}$ can be recovered:

$$\begin{aligned} \alpha_n &= \rho_n^{Bi-CGSTAB}/\sigma_n^{Bi-CGSTAB}, \\ \beta_{n+1} &= (\rho_{n+1}^{Bi-CGSTAB}/\rho_n^{Bi-CGSTAB})(\alpha_n/\omega_{n+1}). \end{aligned}$$

Finally, we comment on the calculation of the $\omega_i$'s. In [55], the value $\omega_n$ is computed so that the norm

$$||r_n|| = ||\tau_n\phi_n r_0|| = ||(I - \omega_n A)\tau_{n-1}\phi_n r_0||$$

is minimized. By letting $s_n = \tau_{n-1}\phi_n r_0$, $\omega_n$ is the orthogonal projection of $s_n$ onto $As_n$:

$$\omega_n = \frac{(s_n, As_n)}{(As_n, As_n)}\ .$$

The Bi-CGSTAB algorithm is given in Table 2.4. As explained in [55], the first part of the loop corresponds to the BCG step for the matrix $A$, and $s_i$ represents an intermediate residual which could be tested for convergence . From the orthogonality of $\varphi_i(A)$ and $\psi_j(A)$ for $j < i$, it follows that in exact arithmetic Bi-CGSTAB terminates with the true solution after $m \leq n$ steps.

This algorithm overcomes some of the wild, irregular convergence behavior of CGS at convergence rate similar to CGS. Note that since both CGS and Bi-CGSTAB have the BCG polynomial built in, they break down whenever BCG does.

Table 2.4: **Algorithm Bi-CGSTAB**

```
    Choose $x_0$, and compute $r_0 = b - Ax_0$
    Pick an arbitrary vector $\tilde{r}_0$
    Set $v_0 = p_0 = 0; \rho_0 = \alpha_1 = \omega_0 = 1$
For $n = 1, 2, \ldots$, *until convergence*
    $\rho_n = \tilde{r}_0^T r_{n-1};$        $\beta_n = (\rho_n/\rho_{n-1})(\alpha_n/\omega_{n-1})$
    $p_n = r_{n-1} + \beta_n(p_{n-1} - \omega_{n-1}v_{n-1})$
    $v_n = Ap_n$
    $\sigma_n = \tilde{r}_0^T v_n;$        $\alpha_n = \rho_n/\sigma_n$
    $s_n = r_{n-1} - \alpha_n v_n$   $t_n = As_n$
    $\omega_n = (s_n, t_n)/(t_n, t_n)$
    $r_n = s_n - \omega_n t_n$
    $x_n = x_{n-1} + \alpha_n p_n + \omega_n s_n$
```

### 2.4.3 Other Product Methods

Recently, various other transpose-free algorithms have been proposed. Freund [26] has devised a transpose-free QMR algorithm (TFQMR), that uses basis vectors from CGS to generate iterates characterized by the QMR property. The Bi-CGSTAB2 method due to Gutknecht [35] and the more general Bi-CGSTAB($l$) method by Fokkema and Sleijpen [31] handle instability in the Bi-CGSTAB algorithm.

We will not give the derivations of these algorithms in this section because these methods are not used explicitly in the following chapters. However, ideas from TFQMR and BiCGSTAB2 are used in Chapters 4 and 7, respectively, and relevant details will be given then.

To cure breakdowns inherited from BCG, look-ahead techniques can be applied to product methods also. See, for example, Freund and Nachtigal [30], and Brezinski and Redivo-Zaglia [9].

# CHAPTER 3

# The Quasi-minimal Residual Squared (QMRS) Algorithm

## 3.1 Motivation

As discussed in the previous chapter, Freund and Nachtigal [29] have recently proposed the Quasi-minimal Residual algorithm (QMR), based on a look-ahead Lanczos process, which overcomes both problems of breakdown and of erratic convergence in the BCG method by using iterates which are defined by a quasi-minimization of the residual norm.

Since the QMR method is based on the nonsymmetric Lanczos process it requires one matrix-vector product with the coefficient matrix $A$, and one product with its transpose $A^T$. This is a disadvantage for certain applications, where $A^T$ is not readily available. Furthermore, it takes 2 matrix-vector products to advance one degree in the Krylov subspace basis, twice as many as for CG.

In this chapter, we propose a new transpose-free method, the Quasi-minimal Residual Squared (QMRS) algorithm, based on the non-look-ahead version of QMR described in Section 2.3. As in the derivation of CGS from BCG (Section 2.4.1), the QMRS scheme is obtained by squaring the QMR residual polynomials. Like standard QMR, the resulting QMRS algorithm converges smoothly, in contrast to CGS, which typically even amplifies the erratic convergence behavior of BCG in many cases.

The remainder of this chapter is organized as follows. In Section 3.2, we show how the iterates of QMR without look-ahead can be obtained directly from the classical BCG process. This is a new way of looking at QMR using two-term recurrences instead of three-term recurrences. It has been observed in practice that two-term recurrences tend to yield more numerically stable results. We then use this to derive the QMRS algorithm in Section 3.3. In Section 3.4, a few results of numerical experiments are reported.

## 3.2 An Implementation of QMR Based on BCG

We begin by deriving a new implementation of the QMR algorithm without look-ahead, using the connection of QMR and the classical BCG algorithm. By reformulating QMR using two-term recurrences, we are not only preparing for the squaring process, but also obtaining a more stable implementation of QMR, as observed in practice. Figure 3.1 is an example of the superior numerical stability of a two-term recurrence. Shown are the decreasing residual norms of QMR based on the two-term BCG (solid line) and based on the three-term Lanczos process (dashed line) when used to solve a linear system that arises in performance modeling of multiprocessor systems. This example is taken from [28]. $A$ is a matrix of size $N = 3663$ with 23397 non-zero elements, and it is preconditioned by a variant of Saad's ILUT preconditioner [50].

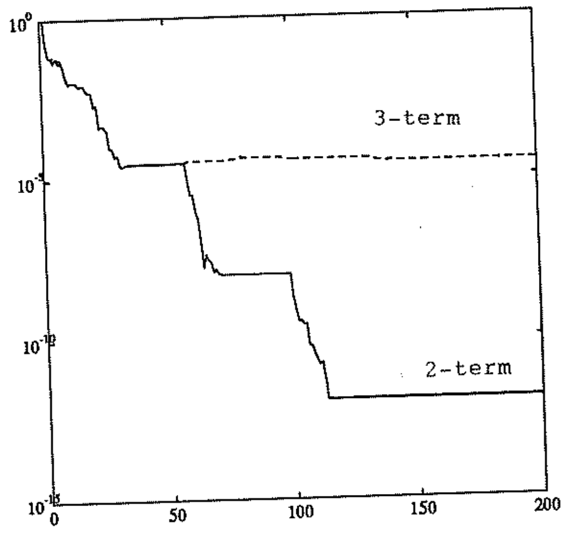

Figure 3.1: QMR/BCG – Two-term vs. three-term recurrences

In deriving this particular implementation of QMR, we will use the notation introduced in Section 2.3. We have already established (Section 2.2.2) that the BCG iterates $x_n^{BCG}$ satisfy

$$x_n^{BCG} = x_0 + V_n \tilde{z}_n, \tag{3.1}$$

where $V_n$ is the set of scaled Lanczos vectors (2.11) defined in Section 2.3, and $\tilde{z}_n$ is the solution of the linear system

$$T_n \tilde{z}_n = \|r_0\| e_1^{(n)}, \quad \text{with} \quad e_1^{(n)} := \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T \in \mathrm{R}^n. \tag{3.2}$$

Here, $T_n$ denotes the $n \times n$ tridiagonal matrix obtained by deleting the last row of the $(n+1)\times n$ scaled Lanczos matrix $T_n^{(e)}$. Multiplying (3.2) by $\Omega_n$, we see that (3.2) is equivalent to

$$\Omega_n T_n \tilde{z}_n = d_1^{(n)}. \tag{3.3}$$

Note that the coefficient matrix $\Omega_n T_n$ of (3.3) is tridiagonal, and the right-hand side is a multiple of $e_1^{(n)}$. Furthermore, the linear system just consists of the first $n$ equations of the least squares problem (2.21).

The solutions $z_n$ and $z_{n-1}$ of two successive Hessenberg least squares problems of the type (2.21) are connected with the solution $\tilde{z}_n$ of the corresponding leading square linear system of the type (3.3) via a lemma proved by Freund [26, Lemma 4.1]. By applying his result to (2.21) and (3.3), we obtain the relation

$$z_n = (1-c_n^2)\begin{bmatrix} z_{n-1} \\ 0 \end{bmatrix} + c_n^2 \tilde{z}_n, \tag{3.4}$$

where $c_n$ is just the cosine of the $n$-th Givens rotation in a QR decomposition of the tridiagonal matrix $\Omega_{n+1}T_n^{(e)}$ (see [26, Section 4]).

Note that $c_n$ can be updated at each step in the following manner:

$$c_n := \frac{1}{\sqrt{1+\vartheta_n^2}}, \quad \vartheta_n := \frac{1}{\tau_{n-1}}\|d_1^{(n+1)} - \Omega_{n+1}T_n^{(e)}\tilde{z}_n\|. \tag{3.5}$$

The $\tau_n$'s can also be updated by setting

$$\tau_n = \tau_{n-1}\vartheta_n c_n, \quad \text{where, for } n=1, \quad \tau_0 := \omega_1\|r_0\|, \tag{3.6}$$

and the norm in (3.5) is directly connected with the norm of the BCG residual vector as follows:

$$\|d_1^{(n+1)} - \Omega_{n+1} T_n^{(e)} \tilde{z}_n\| = \omega_{n+1} \|r_n^{BCG}\|. \tag{3.7}$$

This is due to the fact that

$$d_1^{(n+1)} - \Omega_{n+1} T_n^{(e)} \tilde{z}_n = \zeta_n \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix}^T, \quad \zeta_n \in \mathrm{R},$$

$$r_n^{BCG} = V_{n+1} \Omega_{n+1}^{-1} (d_1^{(n+1)} - \Omega_{n+1} T_n^{(e)} \tilde{z}_n) = \frac{\zeta_n}{\omega_{n+1}} v_{n+1}, \tag{3.8}$$

and $\|v_{n+1}\| = 1$. Hence, we can update the relation (3.4).

Next, we use this to show how the QMR iterates can be updated by means of quantities from the BCG algorithm. Let $x_n$ denote the QMR iterate at the $n$-th step. With (2.20) and (3.1), relationship (3.4) gives

$$\begin{aligned} x_n &= x_{n-1} - c_n^2 x_{n-1} + c_n^2 x_n^{BCG} && (3.9) \\ &= x_{n-1} + \hat{p}_n, && (3.10) \end{aligned}$$

where

$$\begin{aligned} \hat{p}_n &= -c_n^2 x_{n-1} + c_n^2 (x_{n-1}^{BCG} + \alpha_{n-1} p_{n-1}) \\ &= c_n^2 (x_{n-1}^{BCG} - x_{n-1}) + c_n^2 \alpha_{n-1} p_{n-1} \\ &= c_n^2 \vartheta_{n-1}^2 \hat{p}_{n-1} + c_n^2 \alpha_{n-1} p_{n-1}. \end{aligned} \tag{3.11}$$

Here, $\vartheta_{n-1}^2 = 1/c_{n-1}^2 - 1$ and $\alpha_{n-1}$, $p_{n-1}$ are from the BCG iteration (See Table 2.2). We note that $\hat{p}_1 = c_1^2 \alpha_0 p_0$, and thus formula (3.11) remains valid for $n = 1$, if one sets $\vartheta_0 := 0$.

Thus, we can update the QMR iterates, provided that the BCG quantities $\|r_n^{BCG}\|$, $\alpha_{n-1}$, and $p_{n-1}$ are available. By adding the updates (3.4)–(3.7), and (3.11) to the classical BCG algorithm [44], we obtain the following implementation of QMR without look-ahead (Table 3.1).

Note that although we are computing QMR iterates, this algorithm is based on BCG and therefore, can suffer the same breakdowns as BCG.

Table 3.1: **Algorithm QMR without look-ahead from BCG**

```
Choose $x_0, \tilde{r}_0 \neq 0$
Set $x_0^{QMR} = x_0^{BCG} = x_0$, $p_0 = r_0^{BCG} = r_0 = b - Ax_0$,
Set $\hat{p}_0 = 0$, $\tau_0 = \omega_1 \|r_0\|$, $\vartheta_0 = 0$, $\tilde{q}_0 = \tilde{r}_0$, $\rho_0 = \tilde{r}_0^T r_0$
For $n = 1, 2, \ldots,$ do :
    $\sigma_{n-1} = \tilde{p}_{n-1}^T A p_{n-1}$        $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$
    $r_n^{BCG} = r_{n-1}^{BCG} - \alpha_{n-1} A p_{n-1}$        $\tilde{r}_n = \tilde{r}_{n-1} - \alpha_{n-1} A^T \tilde{p}_{n-1}$
    If BCG iterates are desired, set $x_n^{BCG} = x_{n-1}^{BCG} + \alpha_{n-1} p_{n-1}$
    $\vartheta_n = \omega_{n+1} \|r_n^{BCG}\| / \tau_{n-1}$
    $c_n = 1/\sqrt{1 + \vartheta_n^2}$        $\tau_n = \tau_{n-1} \vartheta_n c_n$
    $\hat{p}_n = c_n^2 \vartheta_{n-1}^2 \hat{p}_{n-1} + c_n^2 \alpha_{n-1} p_{n-1}$        $x_n^{QMR} = x_{n-1}^{QMR} + \hat{p}_n$
    $\rho_n = \tilde{r}_n^T r_n^{BCG}$        $\beta_n = \rho_n / \rho_{n-1}$
    $p_n = r_n^{BCG} + \beta_n p_{n-1}$        $\tilde{p}_n = \tilde{r}_n + \beta_n \tilde{p}_{n-1}$
```

## 3.3 The QMR Squared Algorithm

In this section, we derive the QMRS algorithm by squaring the QMR algorithm. We consider the QMR algorithm without look-ahead, where—at the moment—we still allow arbitrary weights $\omega_j > 0$ in (2.23). Let $\varphi_n$ be the polynomial corresponding to the $n$-th QMR residual vector, i.e.,

$$r_n^{QMR} = \varphi_n(A) r_0, \quad \text{where} \quad \varphi_n \in \mathcal{P}_n, \quad \varphi_n(0) = 1. \tag{3.12}$$

Note that from (3.9), the QMR and BCG residuals are connected as follows:

$$r_n^{QMR} = (1 - c_n^2) r_{n-1}^{QMR} + c_n^2 r_n^{BCG}, \tag{3.13}$$

where $c_n$ is the quantity defined in (3.4). Rewriting (3.13) in terms of polynomials, we obtain

$$\varphi_n(\vartheta) \equiv \mu_n \varphi_{n-1}(\vartheta) + \nu_n \phi_n(\vartheta), \tag{3.14}$$

where $\mu_n := 1 - c_n^2$ and $\nu_n = c_n^2$.

In analogy to (2.28), we now define the $n$-th iterate $x_n^{QMRS}$ of the QMR squared algorithm (QMRS) by

$$x_n^{QMRS} \in x_0 + K_{2n}(r_0, A) \tag{3.15}$$

and its residual by

$$r_n^{QMRS} = b - A x_n^{QMRS} = \varphi_n^2(A) r_0. \tag{3.16}$$

Note that the iterate (3.15) can be rewritten in a form involving the polynomial $\hat{\varphi}_n$:

$$x_n^{QMRS} = x_0 + \hat{\varphi}_n(A) r_0, \tag{3.17}$$

where $\hat{\varphi}_n(\vartheta) \equiv (1/\vartheta)(1 - \varphi_n^2(\vartheta))$.

Similarly, we have

$$x_n^{CGS} = x_0 + \hat{\phi}_n(A) r_0, \quad \text{where} \quad \hat{\phi}_n(\vartheta) \equiv (1/\vartheta)(1 - \phi_n^2(\vartheta)). \tag{3.18}$$

Using (3.14) and the fact that $\mu_n + \nu_n = 1$, one readily verifies that the polynomials $\hat{\varphi}_n$, $\hat{\varphi}_{n-1}$, and $\hat{\phi}_n$ in (3.17) and (3.18) are connected as follows:

$$\hat{\varphi}_n(\vartheta) \equiv \mu_n^2 \hat{\varphi}_{n-1}(\vartheta) + 2\mu_n \nu_n (1/\vartheta)(1 - \varphi_{n-1}(\vartheta)\phi_n(\vartheta)) + \nu_n^2 \hat{\phi}_n(\vartheta). \tag{3.19}$$

Furthermore, the fact that $\mu_n^2 + 2\mu_n\nu_n + \nu_n^2 = 1$ gives

$$x_n^{QMRS} = \mu_n^2 x_{n-1}^{QMRS} + 2\mu_n \nu_n f_n + \nu_n^2 x_n^{CGS}, \tag{3.20}$$

where $f_n := x_0 + A^{-1}(I - \varphi_{n-1}(A)\phi_n(A)) r_0$. By means of (3.20), the QMRS iterates can be generated from the standard CGS algorithm, provided that we can also compute the scalar $\mu_n$ and the vector $f_n$.

First, we show how to update $f_n$. Using (3.14), (3.18), and (2.29) (with $n$ replaced by $n-1$),

$$\begin{aligned} (1/\vartheta)(1 - \varphi_{n-1}(\vartheta)\phi_n(\vartheta)) &\equiv \mu_{n-1}(1/\vartheta)(1 - \varphi_{n-2}(\vartheta)\phi_{n-1}(\vartheta)) \\ &+ \nu_{n-1}\hat{\phi}_{n-1}(\vartheta) + \alpha_{n-1}\psi_{n-1}(\vartheta)\varphi_{n-1}(\vartheta). \end{aligned} \tag{3.21}$$

Hence, the fact that $\mu_{n-1} + \nu_{n-1} = 1$ gives

$$f_n = \mu_{n-1} f_{n-1} + \nu_{n-1} x_{n-1}^{CGS} + \alpha_{n-1} s_{n-1}, \tag{3.22}$$

where $s_{n-1} := \psi_{n-1}(A)\varphi_{n-1}(A)r_0$. We note that (3.22) remains valid for $n = 1$, if one sets $\mu_0 = 0$ and $\nu_0 = 1$. To show how to update $s_n$, we set

$$y_n := \varphi_{n-1}(A)\phi_n(A)r_0, \tag{3.23}$$

$$t_n := \varphi_n(A)\phi_n(A)r_0, \tag{3.24}$$

$$g_n := \psi_{n-1}(A)\phi_n(A)r_0. \tag{3.25}$$

Then, with (2.28), (2.29), (2.30), (3.14), (3.22), and (3.23), one readily verifies that

$$y_n = t_{n-1} - \alpha_{n-1} A s_{n-1}, \tag{3.26}$$

$$t_n = \nu_n r_n^{CGS} + \mu_n y_n, \tag{3.27}$$

$$s_n = t_n + \beta_n(\nu_n g_n + \mu_n s_{n-1}). \tag{3.28}$$

The vector $g_n$ also occurs in the CGS process, and its update is part of the standard CGS algorithm.

It remains to show how to obtain $\mu_n$. From (3.14) and (3.4), we have

$$\mu_n = \eta_n \nu_n, \quad \text{where} \quad \nu_n = 1/(1+\eta_n) \quad \text{and} \quad \eta_n := \vartheta_n^2. \tag{3.29}$$

Furthermore, in view of (3.4) and (3.7), the quantity $\eta_n$ is given by

$$\eta_n = \omega_{n+1}^2 \|r_n^{BCG}\|^2 / \xi_{n-1}, \quad \text{where} \quad \xi_{n-1} := \tau_{n-1}^2. \tag{3.30}$$

Note that, by (3.6) and (3.29), the $\xi_n$'s can be updated as follows:

$$\xi_n = \xi_{n-1}\mu_n. \tag{3.31}$$

So far, we have not specified how to select the weight $\omega_{n+1}$ in (3.30). Recall from (2.24) that the natural choice is $\omega_{n+1} = 1$. However, the computation of $\eta_n$ in (3.30) would then require $\|r_n^{BCG}\|$, which is not available in squared algorithms. Instead, we set

$$\omega_{n+1} = (\|r_n^{CGS}\| \cdot \|r_0\|)^{1/2} \,/\, \|r_n^{BCG}\|, \tag{3.32}$$

so that (3.30) reduces to

$$\eta_n = \|r_n^{CGS}\| \cdot \|r_0\| \,/\, \xi_{n-1}. \tag{3.33}$$

Note that, for the choice (3.32), the relations (2.26) and (2.28) suggest that $\omega_{n+1}^2 = \|(\phi_n(A))^2 r_0\| \cdot \|r_0\| / \|\phi_n(A) r_0\|^2 \approx 1$.

In summary, we have shown that the QMRS algorithm corresponding to the weights (3.32) can be implemented with short vector updates. One only has to add the updates (3.20), (3.22), (3.26), (3.29), (3.31), and (3.33) to the standard CGS algorithm [54]. The resulting QMRS algorithm is as follows in Table 3.2.

We remark that the QMRS Algorithm requires 3 matrix-vector products with $A$ per iteration. This will gain two degrees of the Krylov subspace. Although in terms of cost, this improves on the QMR method, both the CGS algorithm and Freund's TFQMR algorithm [26] involve only 2 matrix-vector products with $A$ per iteration to achieve two degrees of the Krylov subspace.. However, squaring the already smooth QMR to obtain QMRS yields an improved residual polynomial over CGS and TFQMR.

Table 3.2: **Algorithm QMRS**

```
Choose $x_0, \tilde{r}_0 \neq 0$
Set $x_0^{QMRS} = x_0^{CGS} = x_0$,
    $s_0 = t_0 = p_0 = u_0 = r_0^{CGS} = r_0 = b - Ax_0$
Set $v_0 = Ap_0$, $f_0 = 0$, $\xi_0 = \|r_0\|^2$, $\eta_0 = \mu_0 = 0$, $\nu_0 = 1$,
    $\tilde{p}_0 = \tilde{r}_0$, $\rho_0 = \tilde{r}_0^T r_0$
For $n = 1, 2, \ldots$, *until convergence*
    $\sigma_{n-1} = \tilde{r}_0^T v_{n-1}$;        $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$
    $g_n = u_{n-1} - \alpha_{n-1} v_{n-1}$;        $f_n = \mu_{n-1} f_{n-1} + \nu_{n-1} x_{n-1}^{CGS} + \alpha_{n-1} s_{n-1}$
    $x_n^{CGS} = x_{n-1}^{CGS} + \alpha_{n-1}(u_{n-1} + g_n)$
    $r_n^{CGS} = r_{n-1}^{CGS} - \alpha_{n-1} A(u_{n-1} + g_n)$
    $\eta_n = \|r_n^{CGS}\| \cdot \|r_0\| / \xi_{n-1}$; $\nu_n = 1/(1 + \eta_n)$; $\mu_n = \eta_n \nu_n$; $\xi_n = \xi_{n-1} \mu_n$
    $x_n^{QMRS} = \mu_n^2 x_{n-1}^{QMRS} + 2\mu_n \nu_n f_n + \nu_n^2 x_n^{CGS}$
    $\rho_n = \tilde{r}_0^T r_n^{CGS}$;        $\beta_n = \rho_n / \rho_{n-1}$
    $u_n = r_n^{CGS} + \beta_n g_{n-1}$
    $p_n = u_n + \beta_n(g_n + \beta_n p_{n-1})$    $v_n = Ap_n$
    $y_n = t_{n-1} - \alpha_{n-1} A s_{n-1}$;        $t_n = \nu_n r_n^{CGS} + \mu_n y_n$
    $s_n = t_n + \beta_n(\nu_n g_n + \mu_n s_{n-1})$
```

## 3.4 Numerical Examples

In this section, we present numerical results for two examples. For both examples, we have compared the QMRS algorithm, the TFQMR algorithm [26], and CGS. For the second example, we have also included standard QMR. The figures show the relative residual norms $\|r_n\|/\|r_0\|$ plotted versus the iteration number $n$, for QMRS (solid line), TFQMR (dashed line), and CGS (dotted line). For the second example, the QMR curve (dash-dotted line) is also depicted.

Our first example is the SHERMAN5 matrix from the Harwell-Boeing collection of sparse test matrices [20]. This matrix comes from a fully implicit black oil simulator on a $16 \times 23 \times 3$ grid, with three unknowns per grid point. Here, $A$ is a matrix of order $N = 3312$, and it has 20793 nonzero elements. This example was run without preconditioning. The convergence curves are shown in Figure 3.2. Note that CGS does diverge, while QMRS and TFQMR converge.

The second example is a system arising from the partial differential equation $-\Delta u + 40(xu_x + yu_y + zu_z) - 240u = f$ on the unit cube $(0,1)^3$. We discretize this equation using centered differences on a uniform $25 \times 25 \times 25$ grid with mesh size $h = 1/26$. The resulting linear system has a sparse coefficient matrix $A$ of order $N = 15625$. This example was run with a variant of an ILU preconditioner. The convergence curves are shown in Figure 3.3.

From the figures, we can see the smoothing effect of QMRS over CGS. This leads

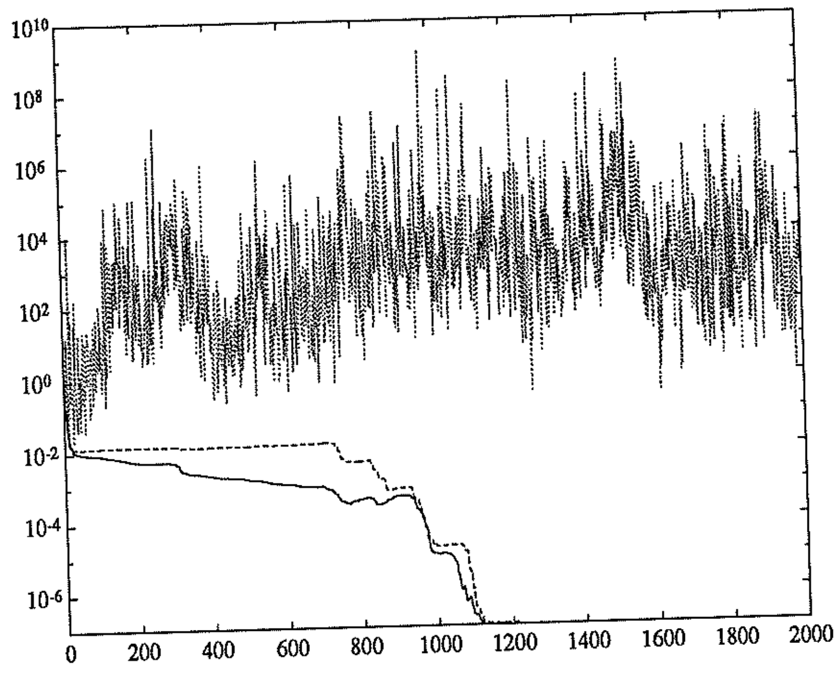

Figure 3.2: QMRS – Sherman5 (Example 3.1)

to convergence in Example 3.1 where CGS diverges. As compared to TFQMR, in Example 3.1, we see some of the additional smoothing of QMRS and in Example 3.2, this leads to convergence in slightly fewer iterations.

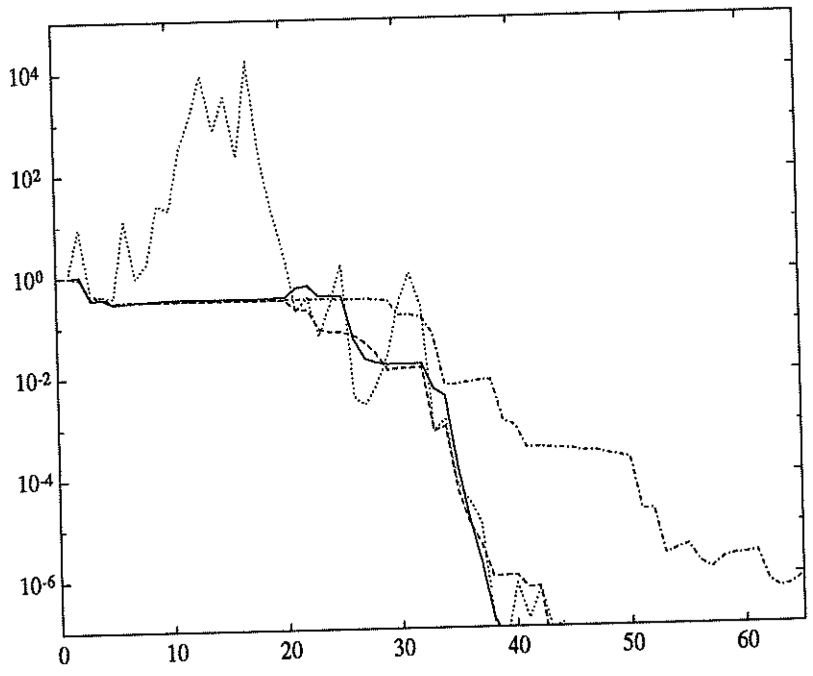

Figure 3.3: QMRS – 3D PDE (Example 3.2)

# CHAPTER 4

# QMR Variants of the Bi-CGSTAB Algorithm

## 4.1 Motivation

In Section 2.4.3, we mentioned a variant of CGS due to Freund, called TFQMR [26], which "quasi-minimizes" [29] the residual in the space spanned by the vectors generated by the CGS iteration. Numerical experiments show that in most cases, TFQMR retains the good convergence features of CGS while correcting its erratic behavior. The transpose-free nature of TFQMR, its low computational cost and its smooth convergence behavior make it an attractive alternative to CGS. On the other hand, since the square of the residual polynomial for BCG is still in the space being quasi-minimized, in many practical examples, CGS and TFQMR converge in about the same number of steps and breaks down in the same way. We note, however, that in contrast to CGS, the asymptotic behavior of TFQMR has been analyzed [25]. It is also well-known that the CGS residual polynomial can be quite polluted by round-off error[56]. One possible remedy would be to combine TFQMR with a look-ahead Lanczos technique as was done in the original QMR method[27].

In this chapter, we take an alternative approach by deriving quasi-minimal residual extensions to Bi-CGSTAB. Our motivation is that the more stable polynomials generated by Bi-CGSTAB will lead to more stable quasi-minimal versions. We call the basic method QMRCGSTAB and illustrate numerically that it does result in smoother convergence than Bi-CGSTAB. Indeed, this smoother behavior parallels the improvement that TFQMR achieves over CGS.

It may appear redundant to combine the local minimization in Bi-CGSTAB with a global quasi-minimization. However, our view is that the local minimization is secondary in nature and is only used as a stable way of generating residual polynomials in the appropriate Krylov subspace over which the residual is being quasi-minimized. In fact, this view allows us some flexibility in modifying the local minimization step in Bi-CGSTAB which leads to other quasi-minimal variants.

## 4.2 The QMRCGSTAB Algorithm

Since our proposed method is based on Bi-CGSTAB, we refer the reader to a review of the details given in Section 2.4.2. The algorithm proposed in this chapter is inspired by TFQMR in that the three term recurrence $x_i = x_{i-1} + \alpha_i p_i + \omega_i s_i$ of Bi-CGSTAB is transformed into a quasi-minimization problem. In other words, we use Bi-CGSTAB to generate the vectors $p_i$ and $s_i$ and quasi-minimize the residual over their span. During each step of Bi-CGSTAB two vector relations hold:

$$s_i = r_{i-1} - \alpha_i A p_i, \qquad r_i = s_i - \omega_i A s_i. \tag{4.1}$$

Let

$$Y_k = \begin{bmatrix} y_1 & y_2 & \cdots & y_k \end{bmatrix},$$

where

$$\begin{cases} y_{2l-1} = p_l & \text{for} \quad l = 1, \dots, [(k+1)/2] \\ y_{2l} = s_l & \text{for} \quad l = 1, \dots, [k/2] \end{cases}$$

( $[k/2]$ is the integer part of $k/2$). In the same way, let

$$W_{k+1} = \begin{bmatrix} w_0 & w_1 & \cdots & w_k \end{bmatrix},$$

with

$$\begin{cases} w_{2l} = r_l & \text{for} \quad l = 0, \dots, [k/2] \\ w_{2l-1} = s_l & \text{for} \quad l = 1, \dots, [(k+1)/2] \end{cases}.$$

We also define

$$\begin{bmatrix} \delta_1 & \cdots & \delta_k \end{bmatrix}, \quad \text{where} \quad \begin{cases} \delta_{2l} = \omega_l & \text{for} \quad l = 1, \dots, [(k+1)/2] \\ \delta_{2l-1} = \alpha_l & \text{for} \quad l = 1, \dots, [(k+1)/2] \end{cases}.$$

In this case, for each vector of $W_{k+1}$ and $Y_k$, (4.1) may be written as

$$A y_j = (w_{j-1} - w_j)\delta_j^{-1}, \quad j = 1, \dots, k,$$

or, using matrix notation,

$$A Y_k = W_{k+1} B_{k+1}^{(e)},$$

where $B^{(e)}_{k+1}$ is the $(k+1) \times k$ bidiagonal matrix

$$B^{(e)}_{k+1} = \begin{bmatrix} \delta_1^{-1} & 0 & 0 & \dots & 0 \\ -\delta_1^{-1} & \delta_2^{-1} & 0 & \dots & 0 \\ 0 & -\delta_2^{-1} & \delta_3^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \delta_{k+1}^{-1} \\ 0 & 0 & 0 & \dots & -\delta_{k+1}^{-1} \end{bmatrix}.$$

It can easily be checked that the degree of the polynomials corresponding to the vectors $r_i, s_i$ and $p_i$ are $2i, 2i-1$ and $2i-2$ respectively. Therefore, $\text{span}(Y_k) = \text{span}(W_k) = K_{k-1}(r_0, A)$, the Krylov subspace of degree $k-1$ generated by $r_0$. The main idea in QMRCGSTAB is to look for an approximation to the solution of (1.1) using the Krylov subspace $K_{k-1}$ in the form

$$x_k = x_0 + Y_k g_k, \quad \text{with} \quad g_k \in \mathrm{R}^k.$$

Hence, we may write the residual $r_k = b - Ax_k$ as

$$r_k = r_0 - AY_k g_k = r_0 - W_{k+1} B^{(e)}_{k+1} g_k.$$

Using the fact that the first vector of $W_{k+1}$ is indeed $r_0$, it follows that

$$r_k = W_{k+1}(e_1^{(k)} - B^{(e)}_{k+1} g_k),$$

where $e_1^{(k)}$ is the first vector of the canonical basis. Since the columns of $W_{k+1}$ are not normalized, it was suggested in [26] to use a $(k+1) \times (k+1)$ scaling matrix

$\Sigma_{k+1}=\text{diag}(\sigma_1,\ldots,\sigma_{k+1})$, with $\sigma_i = \|w_i\|$, in order to make the columns of $W_{k+1}$ to be of unit norm. Then

$$r_k = W_{k+1}\Sigma_{k+1}^{-1}\Sigma_{k+1}(e_1^{(k+1)} - B_{k+1}^{(e)}g_k) = W_{k+1}\Sigma_{k+1}^{-1}(f_{k+1} - H_{k+1}^{(e)}g_k) \tag{4.2}$$

where $H_{k+1}^{(e)} = \Sigma_{k+1}B_{k+1}^{(e)}$ and $f_{k+1} = \sigma_1 e_1^{(k+1)}$.

The quasi-minimal residual approach is defined by the minimization of the pseudo-residual:

$$\tau_k \equiv \|f_{k+1} - H_{k+1}^{(e)}g_k\| = \min_{g\,\in\,\mathrm{R}^k} \|f_{k+1} - H_{k+1}^{(e)}g\|. \tag{4.3}$$

Although the solution of the least squares problem at each step appears to be expensive, it is actually possible to use a recursive formula and update the solution at every step [26]. A QR factorization of $H_{k+1}^{(e)}$:

$$Q_{k+1}H_{k+1}^{(e)} = \begin{bmatrix} R_k \\ 0 \end{bmatrix}$$

can be performed in an incremental manner using $R_k$ as a triangular component of the decomposition. We also note that since $H_{k+1}^{(e)}$ is bidiagonal, only the rotation of the previous step is needed, so no additional memory is required for storing the orthogonal factor $Q_{k+1}$.

In more detail, this is done by applying [26, Lemma 4.1] to (4.3). This results in the following relation for the QMRCGSTAB iterate:

$$x_k = (1 - c_k^2)x_{k-1} + c_k^2\tilde{x}_k,$$

where $c_k$ is the cosine at the $k$-th Givens rotation in the QR factorization, and $\tilde{x}_k$ is the auxiliary iterate:

$$\tilde{x}_k = x_0 + Y_k \tilde{g}_k,$$

where $\tilde{g}_k = H_k^{-1} f_k$ and $H_k$ is obtained from deleting the last row of $H_{k+1}^{(e)}$. It is easily verified that this can be updated:

$$\tilde{x}_k = \tilde{x}_{k-1} + \delta_k y_k.$$

Analogous to TFQMR, we have the values:

$$\begin{aligned} \theta_k &= \sigma_k / \tau_{k-1}, \\ c_k &= 1/\sqrt{1+\theta_k^2} \\ \tau_k &= \tau_{k-1}\theta_k c_k. \end{aligned}$$

Thus, the QMRCGSTAB iterate can be updated:

$$x_k = x_{k-1} + \eta_k d_k,$$

where $\eta_k = c_k^2 \delta_k$ and $d_k = (\tilde{x}_k - x_{k-1})/\delta_k$.

We check for convergence using an estimated bound on $\|r_k\|$ since the true $r_k$ is not generated explicitly.

$$\begin{aligned} \|r_k\| &\leq \|W_{k+1}\Sigma_{k+1}^{-1}\| \|f_{k+1} - H_{k+1}^{(e)} g_k\| \\ &\leq \sqrt{k+1}\,\tau_k. \end{aligned}$$

The pseudocode for the QMRCGSTAB algorithm is given in Table 4.1. Note that the cost per iteration is slightly higher than for Bi-CGSTAB, since 2 additional inner products are needed to compute the elements of $\Sigma_{k+1}$. A more detailed discussion on computational costs is given in Section 4.4.

Table 4.1: **Algorithm QMRCGSTAB**

```
      Choose $x_0$ and compute $r_0 = b - Ax_0$
      Choose $\tilde{r}_0$ such that $(\tilde{r}_0, r_0) \neq 0$
      Set $p_0 = v_0 = d_0 = 0; \rho_0 = \alpha_0 = \omega_0 = 1$
      Set $\tau = \|r_0\|, \theta_0 = \eta_0 = 0$
For   $k = 1, 2, \cdots$ *until convergence*
      $\rho_k = (\tilde{r}_0, r_{k-1});\quad \beta_k = (\rho_k \alpha_{k-1})/(\rho_{k-1}\omega_{k-1})$
      $p_k = r_{k-1} + \beta_k(p_{k-1} - \omega_{k-1}v_{k-1})$
      $v_k = Ap_k$
      $\alpha_k = \rho_k/(\tilde{r}_0, v_k)$
      $s_k = r_k - \alpha_k v_k$
 % First quasi-minimization and update iterate
      $\tilde{\theta}_k = \|s_k\|/\tau;\qquad c = 1/\sqrt{1+\tilde{\theta}_k^2}$
      $\tilde{\tau} = \tau\tilde{\theta}_k c;\qquad \tilde{\eta}_k = c^2\alpha_k$
      $\tilde{d}_k = p_k + \frac{\theta_{k-1}^2\eta_{k-1}}{\alpha_k} d_{k-1}$
      $\tilde{x}_k = x_{k-1} + \tilde{\eta}_k \tilde{d}_k$
 % Compute $t_k$, $\omega_k$ and update $r_k$
      $t_k = As_k$
      $\omega_k = (s_k, t_k)/(t_k, t_k)$
      $r_k = s_k - \omega_k t_k$
 % Second quasi-minimization and update iterate
      $\theta_k = \|r_k\|/\tilde{\tau};\qquad c = 1/\sqrt{1+\theta_k^2}$
      $\tau = \tilde{\tau}\theta_k c;\qquad \eta_k = c^2\omega_k$
      $d_k = s_k + \frac{\tilde{\theta}_k^2\tilde{\eta}_k}{\omega_k}\tilde{d}_k$
      $x_k = \tilde{x}_k + \eta_k d_k$
      If $x_k$ is accurate enough, then quit
```

## 4.3 Some Variants of QMRCGSTAB

The use of quasi-minimization in the product algorithms (such as CGS and Bi-CGSTAB) introduces some flexibility in the underlying methods. For example, the basic product algorithm need not be constrained to generate a residual polynomial that has small norm because, presumably, the quasi-minimization step will handle that. Instead, the basic iteration can be viewed as only generating a (preferably stable) basis for the Krylov subspace over which the quasi-minimization is applied. This view leads us to several variants of QMRCGSTAB which we will briefly describe. Note however that only one of these variants will be used in the numerical experiments.

We make two observations on the QMRCGSTAB method:

(a) It is not crucial that the steepest descent step reduce the norm of the residual as long as it increases the degree of the Krylov subspace associated with $W_{k+1}$.

(b) If $W_{k+1}$ were orthogonal, then quasi-minimization becomes true minimization of the residual.

Therefore, it is natural to choose $\omega_i$ to make $W_{k+1}$ "more orthogonal". For example, one can choose $\omega_i$ to make $r_i$ orthogonal to $s_i$ and equivalently, $W_k$ pairwise orthogonal. This leads to the formula:

$$\omega_i = \frac{(s_i, s_i)}{(s_i, t_i)}$$

which replaces the corresponding formula in Algorithm QMRCGSTAB. We call this variant QMRCGSTAB2. We note that since the inner-product $(s_i, s_i)$ is already needed to compute $\tilde{\theta}_i$, we save one inner-product compared to QMRCGSTAB.

We also note that similarly to Bi-CGSTAB, both QMRCGSTAB and QMR-CGSTAB2 break down if $(s_i, t_i) = 0$ which is possible if $A$ is indefinite (in fact it is always true if $A$ is skew symmetric.) This is an additional breakdown condition over that of BCG. One possible strategy to overcome this is to set a lower bound for the quantity $|(s_i, t_i)|$. However, for matrices with large imaginary parts, Gutknecht [37] observed that Bi-CGSTAB does not perform well because the steepest descent polynomials have only real roots and thus cannot be expected to approximate the spectrum well. In principle, it is possible to derive a quasi-minimal version of Gutknecht's variant of Bi-CGSTAB, but we shall not pursue that here. An alternate strategy for handling this additional breakdown in given in Chapter 7.

## 4.4 Numerical Experiments

We next compare numerically the performance of the QMRCGSTAB variants with that of Bi-CGSTAB, TFQMR, and CGS.

Table 4.2 shows the cost per step of the methods under discussion, excluding the cost for computing the residual norm which is the same for all methods.

In the sequel, we present experiments to show that QMRCGSTAB indeed

Table 4.2: QMRCGSTAB Examples: Cost per step for each method

| | Inner products | DAXPY operations | Matrix-vector multiplications |
|---|---|---|---|
| Bi-CGSTAB | 4 | 6 | 2 |
| CGS | 2 | 7 | 2 |
| QMRCGSTAB | 6 | 8 | 2 |
| QMRCGSTAB2 | 5 | 8 | 2 |
| TFQMR | 4 | 10 | 2 |

achieves a smoothing of the residual compared to Bi-CGSTAB. Note, however, that because the Bi-CGSTAB method already improves the erratic residual convergence of BCG, the effect of QMRCGSTAB is not as impressive as the one of TFQMR on the residual of CGS.

Unless otherwise stated, in all the examples, the right hand side $b$ was generated as a random vector with values distributed uniformly in $(0,1)$, and the starting vector $x_0$ was taken to be zero. All matrices arising from a partial differential operator were obtained using centered, second-order finite differences. The methods were compared on the basis of the number of iterations necessary to achieve relative residual $\frac{\|r_k\|}{\|r_0\|} < 10^{-8}$, with $r_k = b - Ax_k$ being the true residual. Hence, the figures were built with the ordinate axis representing the number of iterations and the coordinate axis representing $\log_{10} \|r_k\|$. Note that all the methods compared require the same number of matrix-vector multiplies at ach iteration (see Table 4.2).

Experiments were conducted using a Beta test version of MATLAB 4.0 [33] running on a Sun Sparc workstation.

**Example 4.1.** We begin by constructing an example which illustrates the superior numerical stability of Bi-CGSTAB over CGS and TFQMR. Matrix $A$ is a modification of an example presented in [47].

$$A = \begin{pmatrix} \epsilon & 1 \\ -25 & 100 \end{pmatrix} \otimes I_{n/2} \tag{4.4}$$

i.e., $A$ is an $n \times n$ block diagonal matrix with $2 \times 2$ blocks and $n = 40$. We choose $b = (1 \quad 0 \quad 1 \quad 0 \quad \ldots)^T$ and $\tilde{r}_0 = r_0$. For such a $b$, the norm of the resulting BCG polynomial satisfies $\|\phi_n\| = \mathcal{O}(\epsilon^{-1})$. Thus, $\|\phi_n^2\| = \mathcal{O}(\epsilon^{-2})$ in the squared methods and we can foresee numerical problems when $\epsilon$ is small.

Each entry of Table 4.3 shows $(i)$ the number of correct digits, $d$, in the relative residual obtained after running each algorithm until the relative residual dropped below $10^{-8}$ but without exceeding 20 matrix-vector multiplications, and $(ii)$ in parentheses, the number of matrix-vector products, $mv$, that is not greater than 20 needed to achieve a relative residual of $10^{-d}$.

In exact arithmetic, finite termination occurs after the second BCG polynomial $\phi_2$ is computed in both the BCG and CGS algorithms. We see from Table 4.3 that all methods behave equally well for $\epsilon = 1.0$. As $\epsilon$ decreases, round-off error causes

CGS and TFQMR, which are based on the squaring of the BCG polynomial, to fail or not to converge within the expected time. Furthermore, both CGS and TFQMR lose about twice as many digits as Bi-CGSTAB and its quasi-minimal variants. We also mark the instances of the quasi-minimal variants whose residuals stagnate before the maximum number of iterations has been reached. We note that although the example is contrived, it does justify the implementation of a QMRCGSTAB-type method.

Table 4.3: Example 4.1 - Correct digits and matrix-vector products at termination: $d(mv)$

| | $\epsilon$ | | | |
|---|---|---|---|---|
| Method | 1.0 | $10^{-4}$ | $10^{-8}$ | $10^{-12}$ |
| CGS | 14(4) | 5(4)‡ | -3(20)* | -1(4)† |
| TFQMR | 13(3) | 5(4)‡ | 1(20)‡ | 1(20)‡ |
| Bi-CGSTAB | 16(3) | 12(3) | 7(3)‡ | 3(3)‡ |
| QMRCGSTAB | 16(3) | 12(3) | 7(3)‡ | 3(3)‡ |
| QMRCGSTAB2 | 16(3) | 12(3) | 7(3)‡ | 3(3)‡ |

* Oscillatory behavior observed

‡ Residual stagnates before max. number of $mv$'s was reached

† Iterations stopped when division by zero was encountered

**Example 4.2.** From [54], the matrix $A$ corresponds to the discretization of the convection-diffusion operator

$$L(u) = -\varepsilon \Delta u + \cos(\alpha) u_x + \sin(\alpha) u_y \tag{4.5}$$

on the unit square with homogeneous Dirichlet conditions on the boundary and parameters $\varepsilon = 0.1$ and $\alpha = -30°$, using 40 grid points per direction, yielding a matrix of order $n = 1600$. Fig. 4.1 shows the convergence histories, from which we can see the smoothing effect of quasi-minimization on the CGS and Bi-CGSTAB residuals. We see that Bi-CGSTAB and its smoothed counterparts converge slightly faster than CGS and TFQMR, with QMRCGSTAB2 winning by a small margin.

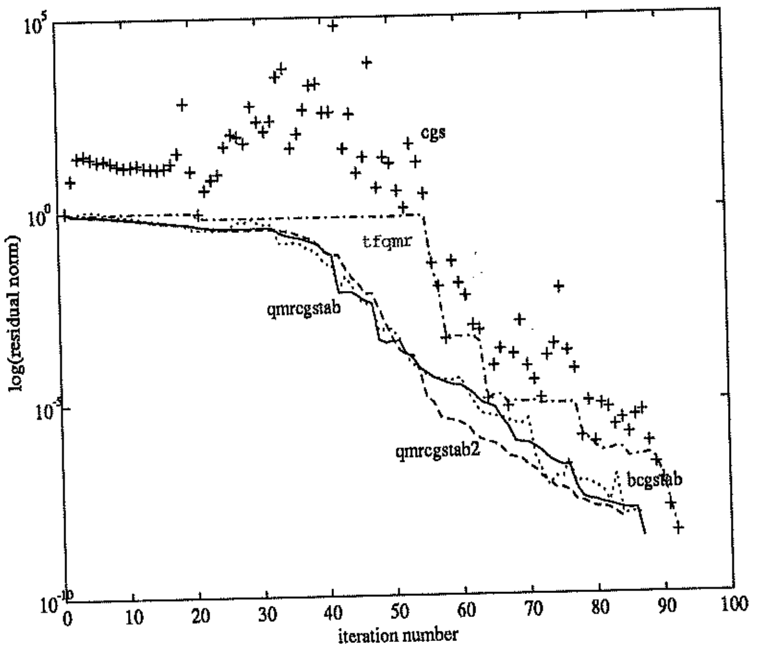

Figure 4.1: QMRCGSTAB – 2D conv-diff (Example 4.2)

**Example 4.3.** This matrix is *Saylr1* from the Harwell-Boeing collection [20]. It is a nonsymmetric matrix $A$ of order $n = 238$. Left diagonal preconditioning was used. Fig. 4.2 demonstrates the significantly better behavior of the residuals in the Bi-CGSTAB versions, especially QMRCGSTAB2.

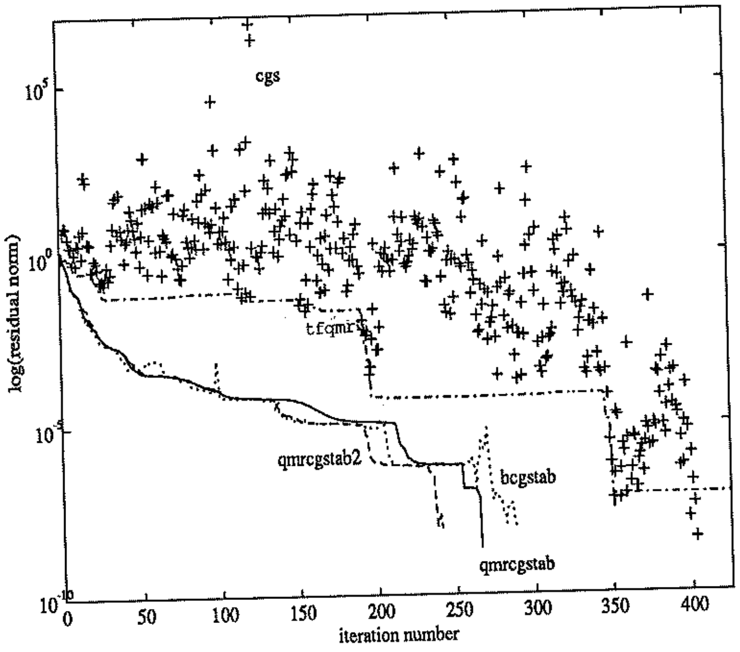

Figure 4.2: QMRCGSTAB – Saylr1 (Example 4.3)

**Example 4.4.** This example comes from the discretization of the convection-diffusion equation

$$L(u) = -\Delta u + \gamma(xu_x + yu_y) + \beta u \tag{4.6}$$

on the unit square where $\gamma = 100, \beta = -100$, for a $63 \times 63$ grid, yielding a matrix of order $n = 3969$. No preconditioning was used. In this example, we see the CGS-based methods converge a little faster than Bi-CGSTAB and QMRCGSTAB, but the pairwise orthogonal variant, QMRCGSTAB2, is the fastest. See Fig. 4.3.

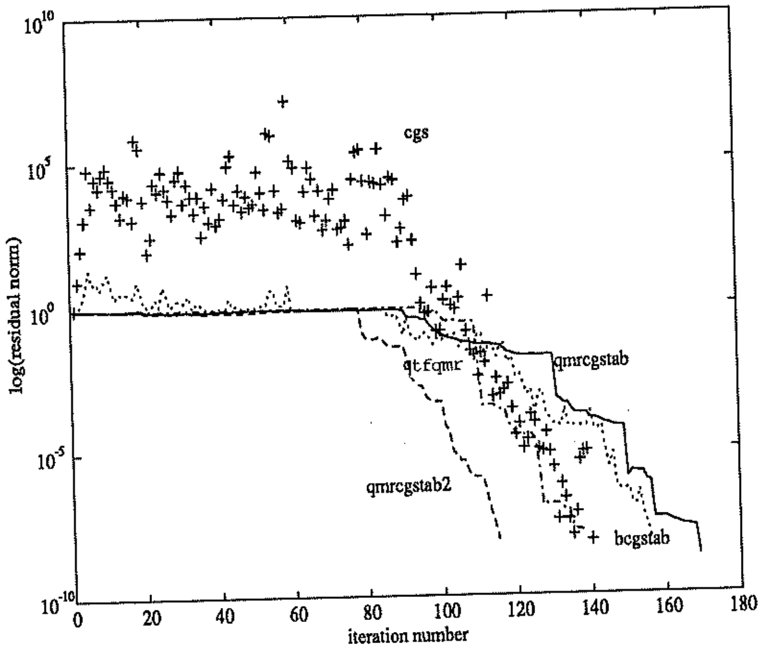

Figure 4.3: QMRCGSTAB – 2D conv-diff (Example 4.4)

**Example 4.5.** Figures 4.4 and 4.5 show the results of a 3-dimensional version of Example 4.4 without preconditioning:

$$L(u) = -\Delta u + \gamma(xu_x + yu_y + zu_z) + \beta u \tag{4.7}$$

on the unit cube where $\beta = -100$, and $\gamma = 50, 1000$ for a $15 \times 15 \times 15$ grid, yielding a matrix of order $n = 3375$. For small values of $\gamma$, all of the methods converge without any problems; see Fig. 4.4. However, for large $\gamma$, this operator will yield a nearly skew-symmetric matrix and the eigenvalues have large imaginary parts.

Therefore, we expect Bi-CGSTAB to have problems [37]. We see in Fig. 4.5 that QMRCGSTAB variants do inherit the problems of Bi-CGSTAB. We will discuss ways to overcome this problem in Chapter 7.

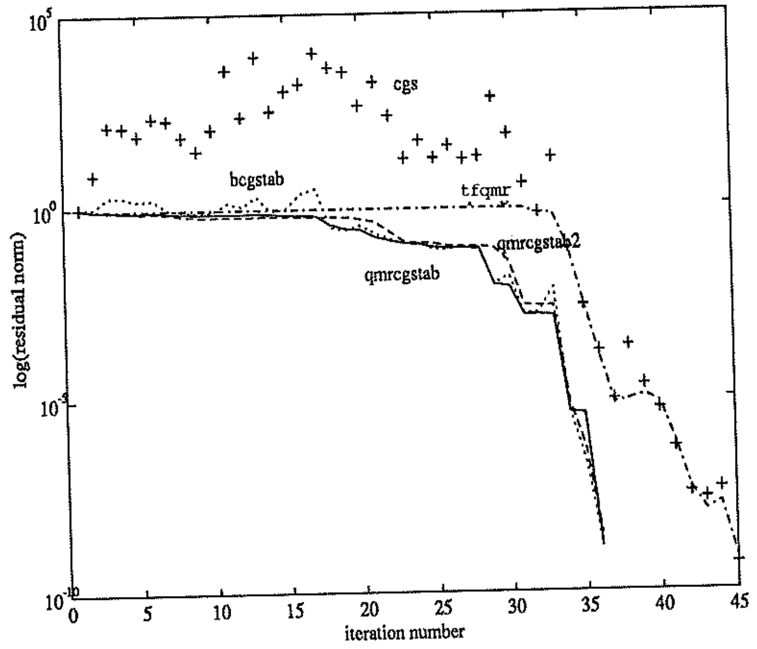

Figure 4.4: QMRCGSTAB – 3D conv-diff, $\gamma = 50$ (Example 4.5a)

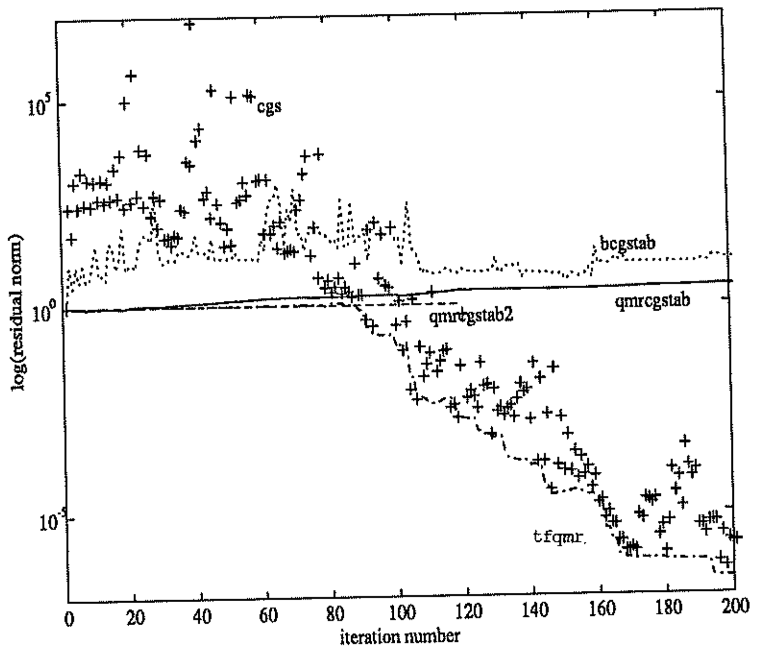

Figure 4.5: QMRCGSTAB – 3D conv-diff, $\gamma = 1000$ (Example 4.5b)

# CHAPTER 5

# The Composite Step Technique

## 5.1 Motivation

Since the product methods presented thus far involve the BCG residual polynomial $\phi_n$, they not only inherit the good properties of BCG, but they also take on some of the problems of BCG. In Section 2.2.2, we discussed the numerical breakdowns that occur in BCG. Recently, many methods have been designed to cure them by "looking ahead" in order to avoid computing iterates where a breakdown can be predicted. The spectrum of these methods ranges from simple modifications of BCG to handle only one of the breakdowns to more complex algorithms which provide almost total breakdown protection using a variable look ahead step.

In this chapter, we consider the *composite step* technique (Bank and Chan [5, 6]) which cures the pivot breakdowns (assuming no Lanczos breakdowns) by simply looking ahead only one step when a (near) breakdown occurs. This technique is attractive because there is no need for user specified tolerance parameters, no variable step sizes, and requires only a minimal modification of existing algorithms (e.g., BCG).

In this chapter we review the details of the Composite Step BCG (CSBCG) algorithm as presented by Bank and Chan [5]. Then, we show that the composite step technique can easily be extended to product methods (e.g., CGS, Bi-CGSTAB) in the following two chapters.

## 5.2 Handling Breakdowns in the BCG Algorithm

Recall, from Section 2.2.2, that a pivot breakdown in BCG occurs when the Hankel moment matrix:

$$H_n^{(1)} = (\bar{K}_n^*)^T A \bar{K}_n = \begin{pmatrix} \mu_1 & \mu_2 & \cdots & \mu_n \\ \mu_2 & \mu_3 & \cdots & \mu_{n+1} \\ \vdots & \vdots & & \vdots \\ \mu_n & \mu_{n+1} & \cdots & \mu_{2n-1} \end{pmatrix},$$

when $\mu_i \equiv \tilde{r}_0^T A^i r_0$, is singular, and the Lanczos breakdown occurs when $H_n^{(0)} \equiv \bar{K}_n^*(\tilde{r}_0)\bar{K}_n(r_0)$ is singular.

To overcome the pivot breakdown, we can "look ahead" in $H_n^{(1)}$ by not computing $x_n$ where it is not defined. Looking ahead means that we build $H_i^{(1)}$ until it is no longer singular and we have an iterate $x_{n+m}, m \geq 1$. There are several approaches to handling this. In the case where $A$ is symmetric, for which Lanczos breakdown cannot occur, it can be treated by the method of hyperbolic pairs due to Luenberger [45], and later expanded by Fletcher [24] (See Section 5.3.2).

For general nonsymmetric matrices, the pivot breakdown can be cured using a three-term recurrence as done in the unnormalized BIORES algorithm of Gutknecht [36]. As previously mentioned, the QMR method [29], if considered without look-ahead Lanczos, avoids pivot breakdowns as it numerically stabilizes the BCG method by computing an iterate defined by a "quasi-minimized" solution (which always exists) instead of the Galerkin condition.

Although the methods described above can cure possible singularities in $H_n^{(1)}$, $H_n^{(0)}$ can still be singular and cause breakdown problems. These breakdowns are harder to fix and many look-ahead methods have been proposed to remedy them as well (see e.g., Freund, Gutknecht and Nachtigal [27], Brezinski, Redivo-Zaglia and Sadok [14, 15], Brezinski and Sadok [10], Joubert [41], and Parlett et al [48]). Although the step size needed to overcome an exact breakdown can be computed in principle, these methods can unfortunately be quite complicated for handling near breakdowns since the sizes of the look-ahead steps are variable (indeed, the breakdowns can be *incurable*).

The Composite Step Biconjugate Gradient (CSBCG) algorithm [5, 6], cures the pivot breakdown (assuming no Lanczos breakdowns) by skipping over steps for which the BCG iterate is not defined. It was shown in [5, Lemma 4.3] that only look-ahead steps of size 2 are needed, but this can also be seen in the relationship between the two Hankel matrices defined above: assuming that $det(H_n^{(0)}) \neq 0$ for all $n$, then no two consecutive principal submatrices of $H_n^{(1)}$ can be singular. (The structure of these Hankel determinants was studied in detail by Draux [19].)

Moreover, to incorporate composite step into BCG requires only a simple modification of the usual implemetation of BCG. By closely relating the composite step version to the BCG recurrence, there ia a better chance that the empirically observed stability of BCG will be inherited.

The composite step idea, then, can in principle be incorporated anywhere the BCG polynomial is used; in particular, in product methods such as CGS [54], Bi-CGSTAB [55], BiCGSTAB2 [37], and TFQMR [26]. Doing this not only cures the breakdown mentioned above, but also takes on the advantages of these product methods over BCG, namely, no multiplications by the transpose matrix and a faster convergence rate. For example, if we take the CSBCG polynomials and square them, we obtain the Composite Step CGS method as will be shown in Chapter 6.

The same can be done for the Bi-CGSTAB algorithm. In Chapter 7, we apply composite step to Bi-CGSTAB by computing products of the CSBCG polynomial with a steepest descent polynomial. This will handle the instability in CSCGS while inheriting the desirable properties of composite step. We call the new method CS-CGSTAB. Moreover, other techniques can be employed to stabilize CS-CGSTAB. For example, the Bi-CGSTAB2 method (Gutknecht, [37]) uses an alternate minimization strategy which can be applied during a composite step to further improve on this method. We will also combine composite step with this strategy from BiCGSTAB2 and give results in Chapter 7.

There are other methods which also employ look-ahead techniques for product methods. The unnormalized $BIORES^2$ [36] squares the $BIORES$ method to handle pivot breakdowns. MRZS and its variants [9, 13] treat both breakdowns in the CGS method, as does the Look-ahead TFQMR method, currently being developed by Freund and Nachtigal [30].

We note that all look-ahead versions of a basic method are essentially equivalent in the sense that, given the same step size, they all compute the same uniquely defined iterate at the end of the look-ahead step. The different methods differ only in the details of the stepping strategy and the recurrences used in carrying out the step.

The composite step approach seeks to achieve this with a minimal modification of the usual implementation of existing methods, such as BCG and CGS, in the hope that the empirically observed numerical stability of these methods will be inherited. In trying to cure only one of the two possible breakdowns in CGS, we make a conscious decision in favor of having a simpler modification of CGS instead of a version which is less prone to breakdown but more complicated. Thus, the CSCGS method should be viewed as one in a spectrum of methods with varying degrees of breakdown protection and complexities of implementation.

## 5.3 The Composite Step BCG (CSBCG) Algortihm

We now review briefly the details of CSBCG. For notation, we refer the reader to the BCG algorithm in Table 2.2. Suppose in running the BCG method, we encounter a situation where $\sigma_n = 0$ at step $n$. We see that the values $x_{n+1}$, $\tilde{x}_{n+1}$,

$r_{n+1}, \tilde{r}_{n+1}$ are not defined. CSBCG overcomes this problem by skipping the $n+1$ update and computing the quantities in step $n+2$ by using scaled versions of $r_{n+1}$ and $\tilde{r}_{n+1}$, which do not require divisions by $\sigma_n$. This is done by defining auxiliary vectors $z_{n+1} \in K_{n+2}(r_0, A)$ and $\tilde{z}_{n+1} \in K^*_{n+2}(\tilde{r}_0, A)$:

$$\begin{aligned} z_{n+1} &= \sigma_n r_{n+1} \\ &= \sigma_n r_n - \rho_n A p_n, \end{aligned} \tag{5.1}$$

$$\begin{aligned} \tilde{z}_{n+1} &= \sigma_n \tilde{r}_{n+1} \\ &= \sigma_n \tilde{r}_n - \rho_n A^T \tilde{p}_n. \end{aligned}$$

These are then used in looking for the step $n+2$ iterates:

$$\begin{aligned} x_{n+2} &= x_n + [p_n, z_{n+1}] f_n, \\ \tilde{x}_{n+2} &= \tilde{x}_n + [\tilde{p}_n, \tilde{z}_{n+1}] \tilde{f}_n, \end{aligned}$$

where $f_n, \tilde{f}_n \in R^2$.

The corresponding residuals are :

$$\begin{aligned} r_{n+2} &= r_n - A[p_n, z_{n+1}] f_n, \\ \tilde{r}_{n+2} &= \tilde{r}_n - A^T[\tilde{p}_n, \tilde{z}_{n+1}] \tilde{f}_n. \end{aligned} \tag{5.2}$$

Similarly, the search directions in a composite step are found using

$$\begin{aligned} p_{n+2} &= r_{n+2} + [p_n, z_{n+1}] g_n, \\ \tilde{p}_{n+2} &= \tilde{r}_{n+2} + [\tilde{p}_n, \tilde{z}_{n+1}] \tilde{g}_n, \end{aligned} \tag{5.3}$$

where $g_n, \tilde{g}_n \in R^2$.

To solve for the unknowns $f_n = \left(f_n^{(1)}, f_n^{(2)}\right)^T$ and $g_n = \left(g_n^{(1)}, g_n^{(2)}\right)^T$, we impose the Galerkin condition of BCG:

$$(\tilde{p}_n, \tilde{z}_{n+1})^T r_{n+2} = 0,$$

and the conjugacy condition:

$$(\tilde{p}_n, \tilde{z}_{n+1})^T A p_{n+2} = 0,$$

and solve the resulting two $2 \times 2$ linear systems:

$$\begin{bmatrix} \tilde{p}_n^T A p_n & \tilde{p}_n^T A z_{n+1} \\ \tilde{z}_{n+1}^T A p_n & \tilde{z}_{n+1}^T A z_{n+1} \end{bmatrix} \begin{bmatrix} f_n^{(1)} \\ f_n^{(2)} \end{bmatrix} = \begin{bmatrix} \tilde{p}_n^T r_n \\ \tilde{z}_{n+1}^T r_n \end{bmatrix} \tag{5.4}$$

$$\begin{bmatrix} \tilde{p}_n^T A p_n & \tilde{p}_n^T A z_{n+1} \\ \tilde{z}_{n+1}^T A p_n & \tilde{z}_{n+1}^T A z_{n+1} \end{bmatrix} \begin{bmatrix} g_n^{(1)} \\ g_n^{(2)} \end{bmatrix} = - \begin{bmatrix} \tilde{p}_n^T A r_{n+2} \\ \tilde{z}_{n+1}^T A r_{n+2} \end{bmatrix}. \tag{5.5}$$

This yields (after some algebra) the quantities:

$$f_n = (\zeta_{n+1}\rho_n, \theta_{n+1})\rho_n^2/\delta_n \text{ and } g_n = (\rho_{n+2}/\rho_n, \sigma_n\rho_{n+2}/\theta_{n+1}),$$

where

$$\zeta_{n+1} = \tilde{z}_{n+1}^T A z_{n+1}, \tag{5.6}$$

$$\theta_{n+1} = \tilde{z}_{n+1}^T z_{n+1}, \tag{5.7}$$

$$\delta_n = \sigma_n \zeta_{n+1} \rho_n^2 - \theta_{n+1}^2. \tag{5.8}$$

Furthermore, Lemma 5.1 in [6] shows $\tilde{f}_n = f_n$ and $\tilde{g}_n = g_n$. It is now possible to compute $x_{n+2}$, $\tilde{x}_{n+2}$, $r_{n+2}$, $\tilde{r}_{n+2}$ and thus, advance from step $n$ to step $n+2$.

The Composite Step BCG algorithm, then, is simply the combination of the $1 \times 1$ and $2 \times 2$ steps. It can be proven [5, 6], provided the underlying Lanczos process doesn't break down, that the use of $2 \times 2$ steps are sufficient for CSBCG to compute exactly those iterates of BCG which are well defined.

### 5.3.1 CSBCG Stepping Strategy

As far as deciding when to actually take a $2 \times 2$ step, CSBCG employs a practical stepping strategy that will skip over exact breakdowns using the criterion:

$$\|r_{n+1}\| > \max\{\|r_n\|, \|r_{n+2}\|\}. \tag{5.9}$$

This will avoid exact breakdowns by skipping over the "peaks" in the residual convergence. Moreover, this strategy will yield a smoother, more stable method. In order to avoid unnecessary computation of $\|r_{n+2}\|$, the inequality (5.9) can be restated in the following algorithm:

```
If ($\|r_{n+1}\| < \|r_n\|$) then          $\longleftarrow$ Condition (5.9a)
    choose $1 \times 1$ step
else
    if ($\|r_{n+1}\| < \|r_{n+2}\|$) then    $\longleftarrow$ Condition (5.9b)
        choose $1 \times 1$ step
    else
```

```
            choose $2 \times 2$ step
        end
    end
```

Since $r_{n+1}$ may be nonexistent, the auxiliary vector $z_{n+1} = \sigma_n r_{n+1}$ can be used to evaluate Condition (5.9a). It can be replaced by the equivalent condition: $\|z_{n+1}\| < |\sigma_n|\|r_n\|$. If this condition is not met, then Condition (5.9b) must be checked. Similarly, $\|r_{n+1}\| < \|r_{n+2}\|$ is restated as $|\delta_n|\|z_{n+1}\| < |\sigma_n|\|\nu_{n+2}\|$, where

(5.10) $$\nu_{n+2} \equiv \delta_n r_{n+2} = \delta_n r_n - \delta_n f_n^{(1)} A p_n - \delta_n f_n^{(2)} A z_{n+1}.$$

It may appear that it will cost more to perform the additional matrix-vector product $Az_{n+1}$ in (5.10) However, note that

$$Ap_{n+1} = Az_{n+1}/\sigma_n + \beta_{n+1} A p_n,$$

and thus, the vector $y_{n+1} \equiv Az_{n+1}$ can be precomputed and used in updating the vector $q_{n+1} \equiv Ap_{n+1}$ :

$$q_{n+1} = y_{n+1}/\sigma_n + \beta_{n+1} q_n.$$

Note that this is updated without any matrix-vector multiplications. In this way, the extra cost of computing the matrix-vector product $Az_{n+1}$ is saved by eliminating the $Ap_n$ multiplication. The additional cost to do this is minimal and note that no user specified tolerance parameters are required for this algorithm. The CSBCG algorithm is given in Table 5.1.

Table 5.1: **Algorithm CSBCG**

```
$p_0 = r_0$;  $\tilde{p}_0 = \tilde{r}_0$;  $q_0 = Ap_0$;  $\tilde{q}_0 = A^T\tilde{p}_0$;  $\rho_0 = \tilde{p}_0^T r_0$
$n \leftarrow 0$
While *method not converged yet* do:
  $\sigma_n = \tilde{p}_n^T q_n$
  $z_{n+1} = \sigma_n r_n - \rho_n q_n$;  $\tilde{z}_{n+1} = \sigma_n \tilde{r}_n - \rho_n \tilde{q}_n$
  $y_{n+1} = Az_{n+1}$;  $\tilde{y}_{n+1} = A^T \tilde{z}_{n+1}$
  $\theta_{n+1} = \tilde{z}_{n+1}^T z_{n+1}$;  $\zeta_{n+1} = \tilde{z}_{n+1}^T y_{n+1}$
  $\xi_{n+1} = \|z_{n+1}\|$;  $\phi_n = \|r_n\|$
  % Decide whether to take a $1 \times 1$ step or a $2 \times 2$ step.
    If $\xi_{n+1} < |\sigma_n|\phi_n$, Then          % $\|r_{n+1}\| < \|r_n\|$
      *one-step* = 1
    Else
      $\nu_{n+2} = \|\delta_n r_n - \rho_n^3 \zeta_{n+1} q_n - \theta_{n+1}\rho_n^2 y_{n+1}\|$
      If $|\delta_n|\xi_{n+1} < |\sigma_n|\nu_{n+2}$, Then          % $\|r_{n+1}\| < \|r_{n+2}\|$
          *one-step* = 1
      Else
          *one-step* = 0
      End If
    End If
  % Compute next iterate.
    If *one-step*, Then          % Usual BCG
      $\alpha_n = \rho_n/\sigma_n$
      $\rho_{n+1} = \theta_{n+1}/\sigma_n^2$;  $\beta_{n+1} = \rho_{n+1}/\rho_n$
      $r_{n+1} = r_n - \alpha_n q_n$;  $\tilde{r}_{n+1} = \tilde{r}_n - \alpha_n \tilde{q}_n$
      $x_{n+1} = x_n + \alpha_n p_n$;  $\tilde{x}_{n+1} = \tilde{x}_n + \alpha_n \tilde{p}_n$
      $p_{n+1} = z_{n+1}/\sigma_n + \beta_{n+1} p_n$;  $\tilde{p}_{n+1} = \tilde{z}_{n+1}/\sigma_n + \beta_{n+1}\tilde{p}_n$
      $q_{n+1} = y_{n+1}/\sigma_n + \beta_{n+1} q_n$;  $\tilde{q}_{n+1} = \tilde{y}_{n+1}/\sigma_n + \beta_{n+1}\tilde{q}_n$
      $n \leftarrow n + 1$
    Else          % $2 \times 2$ step CSBCG
      $\delta_n = \sigma_n \zeta_{n+1}\rho_n^2 - \theta_{n+1}^2$
      $\alpha_n = \zeta_{n+1}\rho_n^3/\delta_n$;  $\alpha_{n+1} = \theta_{n+1}\rho_n^2/\delta_n$
      $r_{n+2} = r_n - \alpha_n q_n - \alpha_{n+1} y_{n+1}$;  $\tilde{r}_{n+2} = \tilde{r}_n - \alpha_n \tilde{q}_n - \alpha_{n+1}\tilde{y}_{n+1}$
      $x_{n+2} = x_n + \alpha_n p_n + \alpha_{n+1} z_{n+1}$;  $\tilde{x}_{n+2} = \tilde{x}_n - \alpha_n \tilde{p}_n - \alpha_{n+1}\tilde{z}_{n+1}$
      $z_{n+2} = r_{n+2}$;  $\tilde{z}_{n+2} = \tilde{r}_{n+2}$
      $\rho_{n+2} = \tilde{z}_{n+2}^T r_{n+2}$;  $\beta_n = \rho_{n+2}/\rho_n$;  $\beta_{n+1} = \sigma_n \rho_{n+2}/\theta_{n+1}$
      $p_{n+2} = z_{n+2} + \beta_n p_n + \beta_{n+1} z_{n+1}$;  $\tilde{p}_{n+2} = \tilde{z}_{n+2} + \beta_n \tilde{p}_n + \beta_{n+1}\tilde{z}_{n+1}$
      $q_{n+2} = Ap_{n+2}$;  $\tilde{q}_{n+2} = A\tilde{p}_{n+2}$
      $n \leftarrow n + 2$
    End If
End While
```

### 5.3.2 The Symmetric Case

Ideas similar to the composite step approach were used earlier by Luenberger [45] and Fletcher [24] in the case where A is symmetric. Note that in this case, there are no Lanczos breakdowns and mathematically, CSBCG and the methods in [24, 45] are, in fact, breakdown-free, and all produce precisely the same iterate $x_{n+2}$.

However, the details of exactly how $x_{n+2}$ is updated are different. In [45], Luenberger treats only the case of exact breakdowns ($\sigma_n = 0$) and computes the iterates from a set of basis vectors different from that of CSBCG. Fletcher [24], on the other hand, handles near breakdowns similar to CSBCG but varies in the actual computation of $f_n$ and $g_n$. For example, $f_n$ is computed as follows:

$$f_n = \left( -\frac{\rho_n}{\theta_n/\rho_n^2} \cdot \frac{\zeta_n/\rho_n^2}{\theta_n/\rho_n^2}, \frac{\rho_n}{\theta_n/\rho_n^2} \cdot \frac{1}{\rho_n} \right) / (1 - \mu_n),$$

where

$$\mu_n = \frac{\sigma_n}{\rho_n} \cdot \frac{\rho_n}{\theta_n/\rho_n^2} \cdot \frac{\zeta_n/\rho_n^2}{\theta_n/\rho_n^2} .$$

This yields different roundoff properties when practically applied and compared to CSBCG on symmetric matrices.

Thus, CSBCG can be viewed as a way of generalizing [24] to nonsymmetric matrices.

# CHAPTER 6

# The Composite Step CGS(CSCGS) Algorithm

## 6.1 Motivation

We now use the composite step idea to derive a more stable variant of the CGS method [54] (see Section 2.4.1) for solving (1.1). The method is based on squaring the Composite Step BCG method, described in the previous chapter, which itself is a stabilized variant of BCG in that it skips over steps for which the BCG iterate is not defined and causes one kind of breakdown in BCG. By doing this, we obtain a method (Composite Step Conjugate Gradients Squared or CSCGS) which not only handles the breakdowns described above, but does so with the advantages of CGS, namely, short recurrences, no multiplications by the transpose matrix, and a faster convergence rate than BCG by advancing two degrees of the Krylov subspace with each iteration. We also propose a practical adaptive stepping strategy to cure both exact and near pivot breakdowns, as well as to provide some smoothing effect to the residuals.

Existing implementations of CGS can be easily modified to incorporate CSCGS, as a maximum look-ahead step size of 2 is sufficient. We will also apply the Minimal

Residual Smoothing algorithm of Schönauer and Weiss [52, 60, 62] to the CSCGS iterates in order to further smooth the convergence behavior.

We begin this chapter by describing the CSCGS algorithm (Section 6.2). Section 6.2.2 gives further details of implementation including the decision strategy for when to take a composite step. This is an important practical issue; as with CSBCG, the strategy we have chosen does not require any user specified tolerance parameters. In Section 6.3, we discuss Minimal Residual Smoothing and its effect on the convergence behavior of CSCGS, and finally, results of numerical experiments are reported in Section 6.4.

## 6.2 The Composite Step CGS Algorithm

The Conjugate Gradients Squared method is an attractive alternate to BCG for reasons mentioned earlier (see Section 2.4.1). However, because it is derived from BCG, breakdowns exist in the CGS algorithm analogous to the breakdowns encountered in BCG. The purpose of CSCGS is to cure the pivot breakdowns using a composite step technique similar to CSBCG (Section 5.3) provided that Lanczos breakdowns do not occur.

Suppose CSBCG were to take only $1 \times 1$ steps, then we could square CSBCG in the same way CGS squares BCG [54]. First, we must express the CSBCG quantities in polynomial form. Since the residual $r_n = b - Ax_n$ of any iterate $x_n \in x_0 + K_n(r_0)$ can be written in the form $r_n = \phi_n(A)r_0$, where $\phi_n$ is a polynomial of degree $\leq n$

and $\phi_n(0) = 1$, we can write the CSBCG residual as

$$r_n^{CSBCG} = \phi_n(A) r_0.$$

Similarly,

$$\begin{aligned} p_n^{CSBCG} &= \psi_n(A) r_0, \\ z_{n+1}^{CSBCG} &= \xi_{n+1}(A) r_0. \end{aligned}$$

We can now obtain squared CSCGS polynomials

$$r_n^{CSCGS} = \phi_n^2(A) r_0$$

$$p_n^{CSCGS} = \psi_n^2(A) r_0$$

with corresponding iterates of the form $x_n^{CSCGS} \in x_0 + K_{2n}(r_0)$.

To handle breakdowns, as in CSBCG, if we encounter $\sigma_n = 0$ in the $n$-th step of the CGS algorithm, we will need to take a $2 \times 2$ step. We formulate the $2 \times 2$ step by first writing the polynomial equivalents of equations (5.2), (5.3), and (5.1):

$$\phi_{n+2}(\vartheta) = \phi_n - \vartheta \psi_n f_n^{(1)} - \vartheta \xi_{n+1} f_n^{(2)}, \tag{6.1}$$

$$\psi_{n+2}(\vartheta) = \phi_{n+2} + \psi_n g_n^{(1)} + \xi_{n+1} g_n^{(2)}, \tag{6.2}$$

$$\xi_{n+1}(\vartheta) = \sigma_n \phi_n - \rho_n \vartheta \psi_n. \tag{6.3}$$

Squaring equations (6.1) and (6.2), we obtain:

$$\begin{aligned}
(6.4)\qquad \phi_{n+2}^2 &= \phi_n^2 + \left[f_n^{(1)}\right]^2 \vartheta^2 \psi_n^2 + \left[f_n^{(2)}\right]^2 \vartheta^2 \xi_{n+1}^2 \\
&\quad -2 f_n^{(1)} \vartheta \phi_n \psi_n - 2 f_n^{(2)} \vartheta \phi_n \xi_{n+1} + 2 f_n^{(1)} f_n^{(2)} \vartheta^2 \psi_n \xi_{n+1}, \\
(6.5)\qquad \psi_{n+2}^2 &= \phi_{n+2}^2 + \left[g_n^{(1)}\right]^2 \psi_n^2 + \left[g_n^{(2)}\right]^2 \xi_{n+1}^2 \\
&\quad +2 g_n^{(1)} \phi_{n+2} \psi_n + 2 g_n^{(2)} \phi_{n+2} \xi_{n+1} + 2 g_n^{(1)} g_n^{(2)} \psi_n \xi_{n+1}.
\end{aligned}$$

Thus, we see that we will need relations to define the quantities $\phi_n \xi_{n+1}$, $\psi_n \xi_{n+1}$, $\xi_{n+1}^2$, $\phi_{n+2}\psi_n$, $\phi_{n+2}\xi_{n+1}$. From equations (6.3) and (6.1), we can write:

$$\begin{aligned}
(6.6)\qquad \phi_n \xi_{n+1} &= \sigma_n \phi_n^2 - \rho_n \vartheta \phi_n \psi_n \\
(6.7)\qquad \psi_n \xi_{n+1} &= \sigma_n \phi_n \psi_n - \rho_n \vartheta \psi_n^2 \\
(6.8)\qquad \xi_{n+1}^2 &= \sigma_n^2 \phi_n^2 - 2\sigma_n \rho_n \vartheta \phi_n \psi_n + \rho_n^2 \vartheta^2 \psi_n^2 \\
(6.9)\qquad \phi_{n+2} \psi_n &= \phi_n \psi_n - f_n^{(1)} \vartheta \psi_n^2 - f_n^{(2)} \vartheta \psi_n \xi_{n+1} \\
(6.10)\qquad \phi_{n+2} \xi_{n+1} &= \phi_n \xi_{n+1} - f_n^{(1)} \vartheta \psi_n \xi_{n+1} - f_n^{(2)} \vartheta \xi_{n+1}^2.
\end{aligned}$$

All of these quantities can be defined in terms of $\phi_n^2$ $(= r_n^{CSCGS})$, $\psi_n^2$ $(= p_n^{CSCGS})$, and $\phi_n \psi_n$. What remains, then, is the updating of $\phi_n \psi_n$. If the previous step was a $1 \times 1$ step, $\phi_n \psi_n$ is defined as

$$(6.11)\qquad \phi_n \psi_n = \phi_n^2 + \beta_n \phi_n \psi_{n-1},$$

where the term $\phi_n \psi_{n-1}$ can be updated (as in CGS) by the relation

$$\phi_n \psi_{n-1} = \phi_{n-1} \psi_{n-1} - \alpha_{n-1} \vartheta \psi_{n-1}^2.$$

However, if the previous step was a $2 \times 2$ step, we cannot use this because the mixed term $\phi_n \psi_{n-1}$ doesn't exist. Instead, we use the fact that

$$\psi_n = \phi_n + \psi_{n-2} g_{n-2}^{(1)} + \xi_{n-1} g_{n-2}^{(2)}.$$

Hence,

$$\phi_n \psi_n = \phi_n^2 + g_{n-2}^{(1)} \phi_n \psi_{n-2} + g_{n-2}^{(2)} \phi_n \xi_{n-1}, \tag{6.12}$$

where the terms $\phi_n \psi_{n-2}$ and $\phi_n \xi_{n-1}$ can be updated by the relations $\phi_{n+2} \psi_n$ and $\phi_{n+2} \xi_{n+1}$ which have already been computed.

One simplification we can perform is in the process of finding coefficients $f_n$ and $g_n$. As shown in [6, Lemma 5.1], the systems (5.4) and (5.5) can be simplified and the two systems can be solved explicitly for $f_n$ and $g_n$:

$$f_n = (\zeta_{n+1} \rho_n, \theta_{n+1}) \rho_n^2 / \delta_n \tag{6.13}$$

$$g_n = (\rho_{n+2}/\rho_n, \sigma_n \rho_{n+2}/\theta_{n+1}), \tag{6.14}$$

where $\rho_n$ and $\sigma_n$ are defined as before. The constants $\zeta_{n+1}$ and $\theta_{n+1}$ can be evaluated using CSCGS vectors that we have already computed in the algorithm.

$$\begin{aligned}
\zeta_{n+1} &= (\xi_{n+1}(A^T)\tilde{r}_0, A\xi_{n+1}(A) r_0) \\
&= (\tilde{r}_0, A\xi_{n+1}^2(A) r_0), \\
\theta_{n+1} &= (\xi_{n+1}(A^T)\tilde{r}_0, \xi_{n+1}(A) r_0) \\
&= (\tilde{r}_0, \xi_{n+1}^2(A) r_0),
\end{aligned}$$

and finally, $\delta_n = \sigma_n \zeta_{n+1} \rho_n^2 - \theta_{n+1}^2$.

### 6.2.1 CSCGS Stepping Strategy

As far as deciding when to take a $2 \times 2$ step, we use the same criterion as in CSBCG [5, 6] in order to develop a strategy that will skip over exact breakdowns. Namely, Condition (5.9)

$$\|r_{n+1}\| > \max\{\|r_n\|, \|r_{n+2}\|\}$$

can be applied.

In order to implement the above test efficiently, and stably, we need to arrange the associated computations in a careful way. Our approach follows that in Section 5.3.1, but the details are somewhat different. Since we do not have $r_{n+1}$ yet, and do not want to compute it unnecessarily (if we decide to skip over it), we use the fact that that $\xi_{n+1} = \sigma_n \phi_{n+1}$ to obtain the relationship $s_{n+1} = \sigma_n^2 r_{n+1}$. Hence, condition (5.9a): $\|r_{n+1}\| < \|r_n\|$, can be written

$$\|s_{n+1}\| < \sigma_n^2 \|r_n\|.$$

To obtain $\|r_{n+2}\|$, we use the fact that we can write $f_n$ explicitly as in (6.13) and scale the $r_{n+2}$ update by defining $\nu_{n+2}$ as follows:

$$(6.15) \quad \nu_{n+2} \equiv \delta_n^2 r_{n+2} = \delta_n^2 r_n - A[\alpha'_n(\delta_n u_n + v'_{n+2}) + \alpha'_{n+1}(\delta_n t_{n+1} + w'_{n+2})],$$

where $\alpha'_n = \zeta_{n+1}\rho_n^3$, $\alpha'_{n+1} = \theta_{n+1}\rho_n^2$, and $v'_{n+2}$, $w'_{n+2}$ are scaled versions of $v_{n+2}$, $w_{n+2}$ :

$$v'_{n+2} \equiv \delta_n v_{n+2} = \delta_n u_n - \alpha'_n b_n - \alpha'_{n+1} c_{n+1},$$

$$w'_{n+2} \equiv \delta_n w_{n+2} = \delta_n t_{n+1} - \alpha'_n c_{n+1} - \alpha'_{n+1} d_{n+1}.$$

Thus, condition (5.9b): $\|r_{n+1}\| < \|r_{n+2}\|$ can be expressed as

$$\delta_n^2 \|s_{n+1}\| < \sigma_n^2 \|\nu_{n+2}\|. \tag{6.16}$$

Unfortunately, evaluating the exact $\|\nu_{n+2}\|$ using (6.15) requires one more matrix-vector multiply. Hence, for practical purposes, we use the following upper bound to approximate $\|\nu_{n+2}\|$:

$$\|\nu_{n+2}\| \leq \|\hat{\delta}_n^2 r_n\| + \kappa \|\hat{\alpha}_n(\hat{\delta}_n u_n + \hat{v}_{n+2}) + \alpha'_{n+1}(\hat{\delta}_n t_{n+1} + \hat{w}_{n+2})\|, \tag{6.17}$$

where $\kappa \approx \|A\|$ (e.g., from power iteration), and where the quantities $\hat{\delta}_n$,$\hat{\alpha}_n$, $\hat{v}_{n+2}$,$\hat{w}_{n+2}$ are values based on the estimate $|\zeta_{n+1}| = |\tilde{r}_0^T A s_{n+1}| \approx \kappa \|\tilde{r}_0\| \|s_{n+1}\|$.

This approximate test may be misleading in the rare case where the $2 \times 2$ matrix in equations (5.4) and (5.5) is near singular, but is not revealed by the approximation (6.17). This will result in a $2 \times 2$ step where a $1 \times 1$ step is desired. Hence, we include an additional check to monitor this case by computing the exact value of $\zeta_{n+1}$ which requires an additional matrix-vector multiplication. In doing this, a matrix-vector multiplication is wasted only in the rare cases where a $2 \times 2$ step has to be aborted.

It is important to note, then, that even with the approximation, our stepping strategy still avoids exact breakdown. Moreover, we want to emphasize that this stepping strategy not only skips exact breakdowns, it is also designed to yield smoother residuals. It may use more composite steps than necessary to overcome exact breakdowns, but it has the advantage of not involving any user specified tolerance parameters.

### 6.2.2 Implementation Details

From Section 6.2.1 we see that at each iteration, the estimated norm of the residual of a $2 \times 2$ step is required even if we decide to take a $1 \times 1$ step. This could result in unnecessary matrix-vector multiplications if a $1 \times 1$ step is chosen. Recall from Section 2.4.1, in one step of CGS, we need to perform the matrix products $A\psi_n^2 r_0$ and $A(\phi_n\psi_n r_0 + \phi_{n+1}\psi_n r_0)$. The CSCGS stepping strategy requires the additional matrix-vector product $As_{n+1} = \sigma^2 A\phi_n^2$ at each step. Hence, using a precomputing strategy analogous to the one used for CSBCG (Section 5.3.1), we can rearrange the matrix-vector multiplications to eliminate the extra product in a $1 \times 1$ step.

Specifically, by precomputing the matrix-vector products $e_n \equiv Au_n$ and $c_{n+1} \equiv Aq_{n+1}$, we can rearrange the usual implementation of the CGS algorithm so that the $1 \times 1$ step requires only these two matrix-vector multiplications. Thus, the $1 \times 1$ step is equivalent to CGS mathematically and also in terms of work per iteration. Furthermore, these two multiplications are all that are necessary to implement our stepping strategy to test for a $2 \times 2$ step.

If a $2 \times 2$ step is chosen, the products $d_{n+1} \equiv As_{n+1}$ and $b_{n+2} \equiv Ap_{n+2}$ must be computed. Note, however, that in order to compute (6.4) we also need the extra matrix multiplications: $A^2\psi_n^2$, $A^2\xi_{n+1}^2$, $A^2\psi_n\xi_{n+1}$, and $A\phi_n\xi_{n+1}$. These can all be absorbed into one extra matrix-vector multiply. Hence, the $2 \times 2$ step requires a total of 5 matrix-vector products, whereas in two steps of CGS, only 4 are needed. This is the price we pay for the composite step. However, it is still a considerable savings from BCG and also significantly less work than QMRS [31] and TFiQMR [16], where an extra matrix-vector multiply is required at *every* step.

In Table 6.1, we summarize the notation we will be using in our implementation of the CSCGS algorithm. We list the vector used in the algorithm, its corresponding polynomial form, and the equations in which it is derived and used. The CSCGS algorithm is given in Table 6.2.

Table 6.1: Notation for CSCGS

| vector | polynomial | where/how derived | where used |
|---|---|---|---|
| $r_n$ | $\phi_n^2 r_0$ | (6.4) | (6.5) (6.6) (6.8) |
| $p_n$ | $\psi_n^2 r_0$ | (6.5) | |
| $u_n$ | $\phi_n \psi_n r_0$ | (6.11) (6.12) | (6.7) (6.9) |
| $s_{n+1}$ | $\xi_{n+1}^2 r_0$ | (6.8) | (6.5) |
| $t_{n+1}$ | $\phi_n \xi_{n+1} r_0$ | (6.6) | (6.10) |
| $q_{n+1}$ | $\psi_n \xi_{n+1} r_0$ | (6.7) | (6.5) (6.11) |
| $v_{n+2}$ | $\phi_{n+2} \psi_n r_0$ | (6.9) | (6.5) (6.12) |
| $w_{n+2}$ | $\phi_{n+2} \xi_{n+1} r_0$ | (6.10) | (6.5) (6.12) |
| $e_n$ | $A\phi_n \psi_n r_0$ | $Au_n$ | (6.4) (6.6) (6.8) |
| $c_{n+1}$ | $A\psi_n \xi_{n+1} r_0$ | $Aq_{n+1}$ | (6.9) (6.10) |
| $d_{n+1}$ | $A\xi_{n+1}^2 r_0$ | $As_{n+1}$ | (6.10) |
| $b_{n+2}$ | $A\psi_{n+2}^2 r_0$ | $Ap_{n+2}$ | (6.7) (6.9) |

Table 6.2: **Algorithm CSCGS**

```
  $\rho_0 = \tilde{r}_0^T r_0$;   $p_0 = u_0 = r_0$;   $b_0 = e_0 = Ap_0$
  Compute $\kappa$ = estimate for $\|A\|$
  $n \leftarrow 0$
While *method not converged yet* do:
  $\sigma_n = \tilde{r}_0^T b_n$
  $q_{n+1} = \sigma_n u_n - \rho_n b_n$;   $c_{n+1} = Aq_{n+1}$
  $s_{n+1} = \sigma_n^2 r_n - \rho_n \sigma_n e_n - \rho_n c_{n+1}$;   $\xi_{n+1} = \|s_{n+1}\|$;   $\phi_n = \|r_n\|$
  % Decide whether to take a 1 × 1 step or a 2 × 2 step.
    If $\xi_{n+1} < \sigma_n^2 \phi_n$, Then          % $\|r_{n+1}\| < \|r_n\|$
        *one-step* = 1
    Else
        $\theta_{n+1} = \tilde{r}_0^T s_{n+1}$;   $\hat{\zeta}_n = \kappa \phi_0 \xi_{n+1}$;   $\hat{\delta}_n = \sigma_n \hat{\zeta}_n \rho_n^2 - \theta_{n+1}^2$
        $\hat{\alpha}_n = \hat{\zeta}_n \rho_n^3$;   $\hat{\alpha}_{n+1} = \theta_{n+1} \rho_n^2$
        $t_{n+1} = \sigma_n r_n - \rho_n e_n$
        $\hat{v}_{n+2} = \hat{\delta}_n u_n - \hat{\alpha}_n b_n - \hat{\alpha}_{n+1} c_{n+1}$;   $\hat{w}_{n+2} = \hat{\delta}_n t_{n+1} - \hat{\alpha}_n c_{n+1} - \hat{\alpha}_{n+1} \kappa s_{n+1}$
        $\hat{\nu}_{n+2} = \|\hat{\delta}_n^2 r_n\| + \kappa \|\hat{\alpha}_n (\hat{\delta}_n u_n + \hat{v}_{n+2}) + \alpha_{n+1} (\hat{\delta}_n t_{n+1} + \hat{w}_{n+2})\|$
        If $\hat{\delta}_n^2 \xi_{n+1} < \sigma_n^2 \hat{\nu}_{n+2}$, Then          % $\|r_{n+1}\| < \|r_{n+2}\|$
            *one-step* = 1
        Else
            $d_{n+1} = As_{n+1}$;   $\zeta_{n+1} = \tilde{r}_0^T d_{n+1}$;   $\delta_n = \sigma_n \zeta_{n+1} \rho_n^2 - \theta_{n+1}^2$
            If $\delta_n^2 \xi_{n+1} < \sigma_n^2 \hat{\nu}_{n+2}$, Then          % Test again with true $\delta_n$
                *one-step* = 1
            Else
                *one-step* = 0
            End If
        End If
    End If
  % Compute next iterate.
    If *one-step*, Then          % Usual CGS
        $\alpha_n = \rho_n / \sigma_n$
        $r_{n+1} = r_n - \alpha_n (e_n + c_{n+1} / \sigma_n)$
        $x_{n+1} = x_n + \alpha_n (u_n + q_{n+1} / \sigma_n)$
        $\rho_{n+1} = \tilde{r}_0^T r_{n+1}$;   $\beta_n = \rho_{n+1} / \rho_n$
        $u_{n+1} = r_{n+1} + \beta_n q_{n+1} / \sigma_n$;   $e_{n+1} = Au_{n+1}$
        $p_{n+1} = u_{n+1} + \beta_n (q_{n+1} / \sigma_n + \beta_n p_n)$
        $b_{n+1} = e_{n+1} + \beta_n (c_{n+1} / \sigma_n + \beta_n b_n)$
        $n \leftarrow n + 1$
    Else          % 2 × 2 step CSCGS
        $\alpha_n = \zeta_{n+1} \rho_n^3 / \delta_n$;   $\alpha_{n+1} = \theta_{n+1} \rho_n^2 / \delta_n$
        $v_{n+2} = u_n - \alpha_n b_n - \alpha_{n+1} c_{n+1}$;   $w_{n+2} = t_{n+1} - \alpha_n c_{n+1} - \alpha_{n+1} d_{n+1}$
        $r_{n+2} = r_n - A[\alpha_n (u_n + v_{n+2}) + \alpha_{n+1} (t_{n+1} + w_{n+2})]$
        $x_{n+2} = x_n + [\alpha_n (u_n + v_{n+2}) + \alpha_{n+1} (t_{n+1} + w_{n+2})]$
        $\rho_{n+2} = \tilde{r}_0^T r_{n+2}$;   $\beta_n = \rho_{n+2} / \rho_n$;   $\beta_{n+1} = \sigma_n \rho_{n+2} / \theta_{n+1}$
        $u_{n+2} = r_{n+2} + \beta_n v_{n+2} + \beta_{n+1} w_{n+2}$;   $e_{n+2} = Au_{n+2}$
        $p_{n+2} = u_{n+2} + \beta_n (\beta_n p_n + \beta_{n+1} q_{n+1} + v_{n+2}) + \beta_{n+1} (\beta_n q_{n+1} + \beta_{n+1} s_{n+1} + w_{n+2})$
        $b_{n+2} = Ap_{n+2}$
        $n \leftarrow n + 2$
    End If
End While
```

## 6.3 Residual Smoothing

One other technique we can incorporate to obtain smoother convergence is the Minimal Residual Smoothing (MRS) algorithm, due to Schönauer and Weiss [52, 60] and further extended by Zhou and Walker [62] and Brezinski and Redivo-Zaglia [12]. The idea is to generate a sequence $\{y_n\}$ using the relation $y_n = (1-\eta_n)y_{n-1} + \eta_n x_n$, where $x_n$ is the iterate of a particular iterative method and $\eta_n$ is chosen so that the new resulting residuals $b - Ay_n$ have monotone decreasing norms and $\|b - Ay_n\|_2 \leq \|b - Ax_n\|_2$, for each $n$. Note that although doing this will not cure the problem of breakdown, it does track the true residual more accurately, and in a smooth fashion, as shown in [62].

To incorporate Minimal Residual Smoothing into the Composite Step algorithm which does handle pivot breakdown, use $\{x_n\}$ from CSCGS. Choosing this set of $\{x_n\}$ will provide a more stable basis for smoothing than would, say, CGS. Then, we generate $y_n$ as described above for $1 \times 1$ steps. For $2 \times 2$ steps, define $y_n = (1-\eta_n)y_{n-2} + \eta_n x_n$ and choose $\eta_n$ to minimize $\|b - Ay_n\|_2$.

The MRS algorithm we have implemented (shown in Table 6.3), is one of several smoothing techniques described in [62]. Although we have selected this particular algorithm, note that the other techniques perform similarly.

Table 6.3: **Algorithm MRS for CSCGS**

```
Set $s_0 = r_0^{CSCGS}, y_0 = \hat{x}_0 = x_0^{CSCGS}$, and $u_0 = v_0 = 0$
Let $\hat{x}_0, \hat{x}_1, \hat{x}_2, \ldots$ be consecutive iterates of the CSCGS method
    (using either $1 \times 1$ steps or $2 \times 2$ steps).
For $n = 1, 2, 3, \ldots$
    $p_n = \hat{x}_n - \hat{x}_{n-1};$
    $u_n = u_{n-1} + Ap_n;$
    $v_n = v_{n-1} + p_n;$
    $\eta_n = s_{n-1}^T u_n / u_n^T u_n;$
    $s_n = s_{n-1} - \eta_n u_n;$
    $y_n = y_{n-1} - \eta_n v_n;$
    $u_n \leftarrow (1 - \eta_n) u_n;$
    $v_n \leftarrow (1 - \eta_n) v_n;$
```

## 6.4 Numerical Experiments

The experiments in this section were run in MATLAB 4.0 on a SUN Sparc station with machine precision about $10^{-16}$. We shall use CSCGS* to refer to an implementation of CSCGS where the norm estimate is *not* used in the composite step decision strategy (i.e., *all* matrix-vector multiplications are performed). We will compare the performance of CSCGS, CSCGS*, MRS(CSCGS) with methods including BCG, CGS, Bi-CGSTAB, TFQMR on a contrived example as well as on matrices which come from several partial differential equation operators.

**Example 6.1.** We begin the numerical experiments with a contrived example to illustrate the superior numerical stability of composite step methods over those without composite step. Let $A$ be a modification of the example used in Section 4.4 and in [47]:

$$A = \begin{pmatrix} \epsilon & 1 \\ -1 & \epsilon \end{pmatrix} \otimes I_{n/2},$$

i.e., $A$ is a $n \times n$ block diagonal with $2 \times 2$ blocks, and $n = 40$. By choosing $b = (1 \quad 0 \quad 1 \quad 0 \quad \cdots)^T$ and a zero initial guess, we set $\sigma_0 = \epsilon$, and thus, we can forsee numerical problems with methods such as BCG and CGS when $\epsilon$ is small. Although these methods converge in 2 steps in exact arithmetic when $\epsilon \neq 0$, in finite precision, convergence gets increasingly unstable as $\epsilon$ decreases. Table 6.4 shows the relative error in the solution after 2 steps of BCG, CGS, and their composite step counterparts. Note that the loss of significant digits in BCG and CGS is proportional to $O(\epsilon^{-1})$ and $O(\epsilon^{-2})$, respectively, whereas the accuracy of CSBCG and CSCGS is insensitive to $\epsilon$.

Table 6.4: Example 6.1

| Rel. error in the soln. after 2 steps (n=40) | | | | |
|---|---|---|---|---|
| | BCG | CSBCG | CGS | CSCGS |
| $\epsilon = 10^{-4}$ | $1.5 \times 10^{-12}$ | $1.1 \times 10^{-16}$ | $2.5 \times 10^{-8}$ | 0 |
| $\epsilon = 10^{-8}$ | $2.5 \times 10^{-8}$ | $1.1 \times 10^{-16}$ | $1.0 \times 10^{0}$ | $1.1 \times 10^{-16}$ |
| $\epsilon = 10^{-12}$ | $4.9 \times 10^{-4}$ | $2.0 \times 10^{-28}$ | $1.3 \times 10^{8}$ | $2.0 \times 10^{-28}$ |

**Example 6.2.** This example comes from the Harwell-Boeing set of sparse test matrices [20]. The matrix is a discretization of the convection-diffusion equation:

$$L(u) = -\Delta u + 100(xu_x + yu_y) - 100u$$

on the unit square for a $63 \times 63$ grid. We use a random right hand side, zero initial guess, and left diagonal preconditioning. Figure 6.1 plots the number of iterations versus the true residual norm for CSCGS* and CGS, and smoothed versions of these: MRS on CSCGS* and TFQMR [26]. This is done in order to illustrate the cutting off of the "peaks" of CGS, as seen in this figure. Specifically, note that the maximum point of the CGS curve is around $10^{10}$ whereas it is only $10^{6}$ for the composite step version. For this particular example, the CGS residual stagnates at $10^{-6}$ and so does its smoothed counterpart TFQMR, whereas CSCGS* reaches the stopping criterion $\|r_n\|/\|r_0\| < 10^{-8}$. We refer the reader to [34] for further discussion and explanation of the behavior and accuracy of CG-like methods.

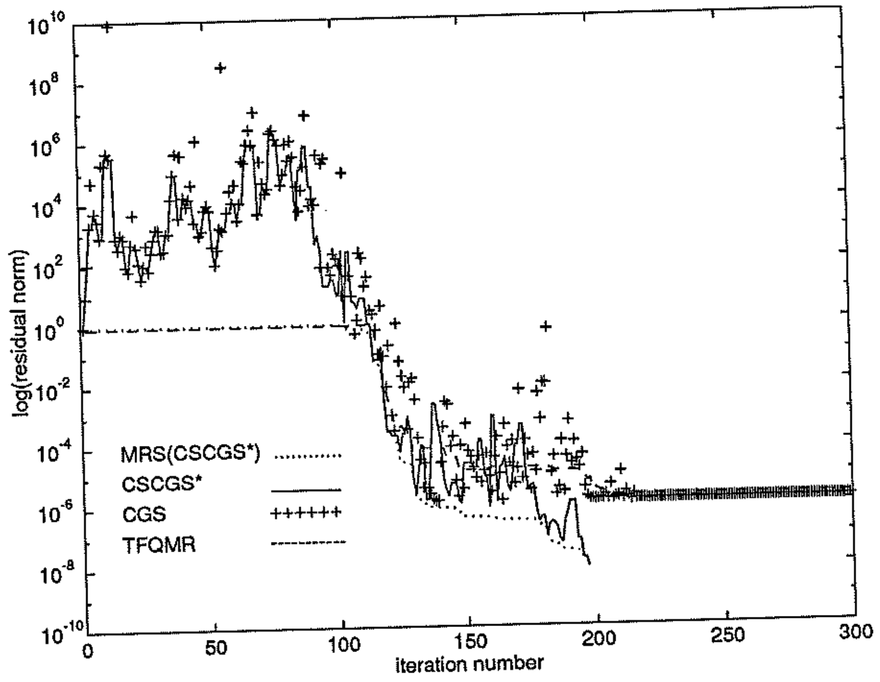

Figure 6.1: CSCGS - 2D conv-diff (Example 6.2a)

The next plot, Fig. 6.2, shows the same example with different axes: the number of matrix-vector multiplications versus the norm of the residual. Instead of CSCGS*, we now illustrate the behavior of CSCGS - using the norm estimation strategy described in section 6.2.2 . Recall that for CSCGS, an extra matrix-vector multiply is required when a $2 \times 2$ step is performed. Note, however, that this does not make a significant impact when compared to the previous figure. We see that even with the approximation, the composite step method manages to control the wild behavior of CGS and eventually converge to the desired residual tolerance. In this example, the total number of $2 \times 2$ steps taken is 23, whereas 133 $1 \times 1$ steps are taken. The $2 \times 2$ step was aborted 2 times.

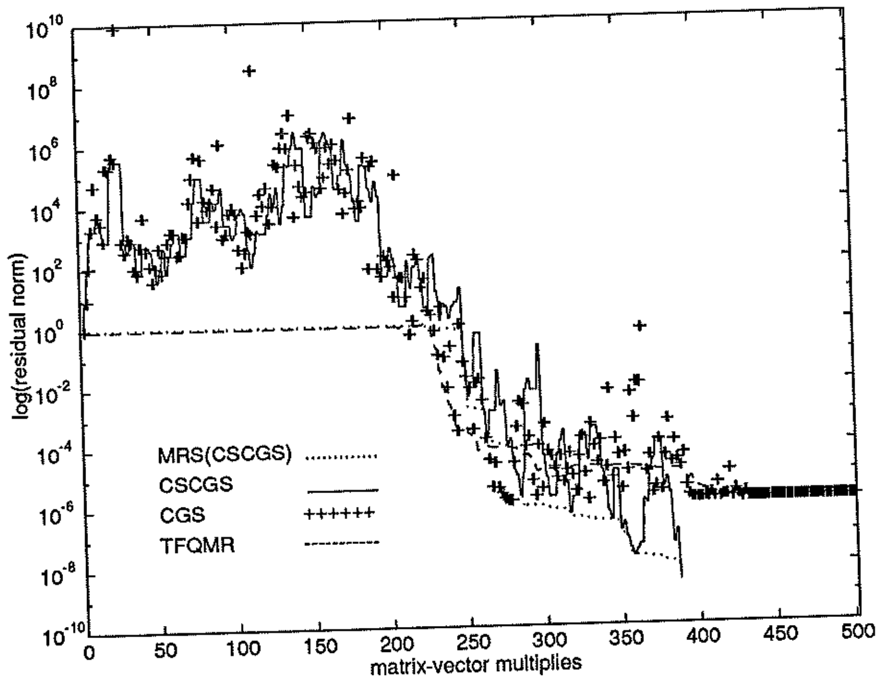

Figure 6.2: CSCGS - 2D conv-diff (Example 6.2b)

Hence, Figures 6.1 and 6.2 clearly depict the advantage of composite step methods over CGS for this example matrix.

**Example 6.3.** The next example, taken from [17], is a discretization of

$$L(u) = -\Delta u + 2e^{2(x^2+y^2)}u_x - 100u,$$

on the unit square for a $40 \times 40$ grid. We use a random right hand side, and the same preconditioning and stopping criterion as in the previous example. Figure 6.3 illustrates the superior behavior of MRS(CSCGS) as compared with not only CGS, but also Bi-CGSTAB [55], another transpose-free product method, as well as CSBCG. We did not include the CSCGS plot, so as not to over-complicate the picture, but mention that it converges in about the same number of matrix-vector multiplications as MRS(CSCGS). We see that the new method reaches the desired tolerance with less work than the other methods. There were 239 $1 \times 1$ steps and 37 $2 \times 2$ steps.

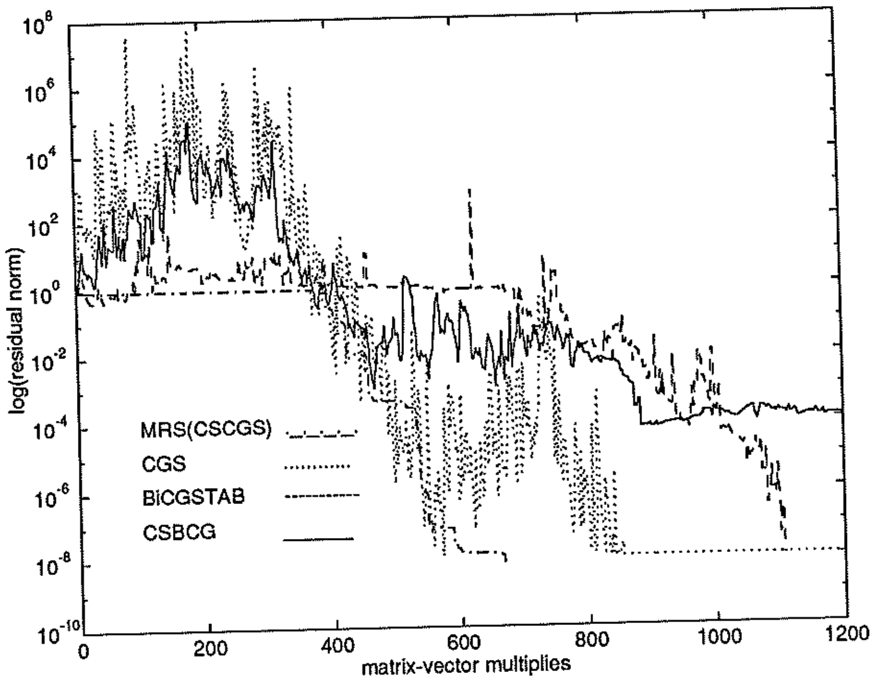

Figure 6.3: CSCGS - PDE (Example 6.3)

# CHAPTER 7

## Composite Step Bi-CGSTAB Algorithms

In the same way composite step was applied to CGS in the previous chapter, it can also be applied to other product methods such as Bi-CGSTAB to overcome analogous pivot breakdowns due to the BCG polynomial involved. As we have already noted the advantages of Bi-CGSTAB over CGS, our hope is that they carry over to their more stable composite step counterparts.

In this chapter, we present CS-CGSTAB, the composite step technique applied to Bi-CGSTAB, due to Van der Vorst [55], (see Section 2.4.2) in Section 7.1. Additionally, we propose a variant of this method CS-CGSTAB2 (Section 7.2) which gives further numerical stability by curing an additional breakdown problem due to the steepest descent part of the Bi-CGSTAB residual.

## 7.1 CS-CGSTAB

For notation, we refer the reader to the Bi-CGSTAB algorithm (Table 2.4) in Section 2.4.2. Since the Bi-CGSTAB residual polynomials are formed from multiplying the BCG polynomial $\phi_n$ with another polynomial $\tau_n$ of the form (2.33), we can use the subset of well-defined $\phi_i$'s from CSBCG to multiply with $\tau_n$ instead.

To do this, recall

$$p_n^{CSBCG} = \phi_n(A)r_0; \quad z_{n+1}^{CSBCG} = \xi_{n+1}(A)r_0.$$

The CS-CGSTAB polynomials, then, take on the form:

$$r_n^{CS-CGSTAB} = \tau_n(A)\phi_n(A)r_0; \quad p_n^{CS-CGSTAB} = \tau_n(A)\psi_n(A)r_0.$$

In the case of a $1 \times 1$ step, we simply use the Bi-CGSTAB update. For the $2 \times 2$ composite step, we will need to evaluate

$$\begin{aligned} r_{n+2}^{CS-CGSTAB} &= \tau_{n+2}\phi_{n+2}r_0 \\ &= (I - \omega_{n+2}A)(I - \omega_{n+1}A)\tau_n\phi_{n+2}r_0 \end{aligned} \tag{7.1}$$

and

$$\begin{aligned} p_{n+2}^{CS-CGSTAB} &= \tau_{n+2}\psi_{n+2}r_0 \\ &= (I - \omega_{n+2}A)(I - \omega_{n+1}A)\tau_n\psi_{n+2}r_0 \end{aligned} \tag{7.2}$$

using the quantities obtained from the $n$-th step: $\tau_n\phi_n$, and $\tau_n\psi_n$. We first show how to compute the polynomials $\tau_n\phi_{n+2}$ and $\tau_n\psi_{n+2}$ appearing in (7.1) and (7.2). We use the CSBCG relationships from (5.1) - (5.3):

$$\xi_{n+1}(\vartheta) = \sigma_n\phi_n - \rho_n\vartheta\psi_n, \tag{7.3}$$

$$\phi_{n+2}(\vartheta) = \phi_n - \vartheta\psi_n f_n^{(1)} - \vartheta\xi_{n+1}f_n^{(2)}, \tag{7.4}$$

$$\psi_{n+2}(\vartheta) = \phi_{n+2} + \psi_n g_n^{(1)} + \xi_{n+1}g_n^{(2)} \tag{7.5}$$

and multiply by the $\tau_n$ polynomial to evaluate the needed quantities:

$$\tau_n(\vartheta)\xi_{n+1}(\vartheta) = \sigma_n\tau_n\phi_n - \rho_n\vartheta\tau_n\psi_n \tag{7.6}$$

$$\begin{aligned}\tau_n(\vartheta)\phi_{n+2}(\vartheta) &= \tau_n(\phi_n - \vartheta\psi_n f_n^{(1)} - \vartheta\xi_{n+1}f_n^{(2)}),\\ &= \tau_n\phi_n - \vartheta\tau_n\psi_n f_n^{(1)} - \vartheta\tau_n\xi_{n+1}f_n^{(2)},\end{aligned} \tag{7.7}$$

$$\begin{aligned}\tau_n(\vartheta)\psi_{n+2}(\vartheta) &= \tau_n(\phi_{n+2} + \psi_n g_n^{(1)} + \xi_{n+1}g_n^{(2)})\\ &= \tau_n\phi_{n+2} + \tau_n\psi_n g_n^{(1)} + \tau_n\xi_{n+1}g_n^{(2)}.\end{aligned} \tag{7.8}$$

We now show how to compute the unknowns $f_n$ and $g_n$ in (7.7) and (7.8). The CSBCG residual $r_{n+2}$ in (5.2) and search direction $p_{n+2}$ in (5.3) are, respectively, orthogonal and $A$-conjugate to $K^*_{n+1}(\tilde{r}_0)$ [5]. By imposing orthogonality and $A$-conjugacy conditions on two specific vectors $\tau_n(A^T)\tilde{r}_0 \in K^*_{n+1}(\tilde{r}_0)$ and $A^T\tau_n(A^T)\tilde{r}_0 \in K^*_{n+1}(\tilde{r}_0)$, we obtain two linear systems which give $f_n$ and $g_n$.

Specifically, by writing equation (5.2) in polynomial form (7.4) and taking an inner product with $\tau_n(A^T)\tilde{r}_0$, we obtain the relation:

$$\begin{aligned}0 &= (\tau_n(A^T)\tilde{r}_0, \phi_{n+2}(A)r_0)\\ &= (\tilde{r}_0, \tau_n(A)\phi_{n+2}(A)r_0)\\ &= (\tilde{r}_0, [\tau_n\phi_n - A\tau_n\psi_n f_n^{(1)} - A\tau_n\xi_{n+1}f_n^{(2)}](A)r_0).\end{aligned}$$

Similarly, for the $A$-conjugacy of the search direction (5.3),

$$\begin{aligned}0 &= (A^T\tau_n(A^T)\tilde{r}_0, \psi_{n+2}r_0)\\ &= (\tilde{r}_0, A\tau_n(A)\psi_{n+2}(A)r_0)\\ &= (\tilde{r}_0, [A\tau_n\phi_{n+2} + A\tau_n\psi_n g_n^{(1)} + A\tau_n\xi_{n+1}g_n^{(2)}](A)r_0).\end{aligned}$$

We derive two similar relations by imposing the orthogonality and $A$-conjugacy conditions on $A^T\tau_n(A^T)\tilde{r}_0$. Combining the four relations, we obtain the following two $2 \times 2$ linear systems:

$$\text{(7.9)} \qquad \begin{bmatrix} (\tilde{r}_0, \vartheta\tau_n\psi_n r_0) & (\tilde{r}_0, \vartheta\tau_n\xi_{n+1} r_0) \\ (\tilde{r}_0, \vartheta^2\tau_n\psi_n r_0) & (\tilde{r}_0, \vartheta^2\tau_n\xi_{n+1} r_0) \end{bmatrix} \begin{bmatrix} f_n^{(1)} \\ f_n^{(2)} \end{bmatrix} = \begin{bmatrix} (\tilde{r}_0, \tau_n\phi_n r_0) \\ (\tilde{r}_0, \vartheta\tau_n\phi_n r_0) \end{bmatrix}$$

$$\text{(7.10)} \qquad \begin{bmatrix} (\tilde{r}_0, \vartheta\tau_n\psi_n r_0) & (\tilde{r}_0, \vartheta\tau_n\xi_{n+1} r_0) \\ (\tilde{r}_0, \vartheta^2\tau_n\psi_n r_0) & (\tilde{r}_0, \vartheta^2\tau_n\xi_{n+1} r_0) \end{bmatrix} \begin{bmatrix} g_n^{(1)} \\ g_n^{(2)} \end{bmatrix} = \begin{bmatrix} (\tilde{r}_0, \vartheta\tau_n\phi_{n+2}) \\ (\tilde{r}_0, \vartheta^2\tau_n\phi_{n+2}) \end{bmatrix}.$$

These are easily solved since all of the entries in the $2 \times 2$ matrix and the right hand sides can be obtained from equations (7.6), (7.7), and quantities from the $n$-th step. Thus, we can update (7.7) and (7.8).

The next step in evaluating (7.1) and (7.2) is to choose $\omega_{n+1}$ and $\omega_{n+2}$ to satisfy some local minimization property. In the case of a $1 \times 1$ step, we imitate the Bi-CGSTAB update steps. Specifically, $\omega_{n+1}$ is chosen to minimize the norm of $r_{n+1} = \tau_{n+1}\phi_{n+1}r_0 = (I - \omega_{n+1}A)\tau_n\phi_{n+1}r_0$. For the $2 \times 2$ step, we employ the same steepest descent rule to compute $\omega_{n+1}$ and $\omega_{n+2}$ by minimizing $\|r_{n+1}\|$ and $\|r_{n+2}\|$. Note that $r_{n+1}$ is not available in a $2 \times 2$ step, but we can use a scaled version for minimization. To do this, let $u_{n+1} = \tau_n\xi_{n+1}r_0 \equiv \frac{1}{\sigma_n}\,\tau_n\phi_{n+1}r_0$. Then we can write

$$\text{(7.11)} \quad r_{n+1} = \tau_{n+1}\phi_{n+1}r_0 = (I - \omega_{n+1}A)\tau_n\phi_{n+1}r_0 = (I - \omega_{n+1}A)\frac{1}{\sigma_n}\,u_{n+1}.$$

The vector $u_{n+1}$ is already computed in the CS-CGSTAB algorithm (see relation

(7.6)) and thus, we minimize

$$\|r_{n+1}^2\| = \frac{1}{\sigma_n^2}\ ((I-\omega_{n+1}A)u_{n+1})^T((I-\omega_{n+1}A)u_{n+1})$$

by choosing $\omega_{n+1} = (Au_{n+1}, u_{n+1})/(Au_{n+1}, Au_{n+1})$, an orthogonal projection of $u_{n+1}$ onto $Au_{n+1}$.

Similarly, let

$$u_{n+2} \equiv \tau_{n+1}\phi_{n+2}r_0 = (I-\omega_{n+1}A)\tau_n\phi_{n+2}r_0 \tag{7.12}$$

which can be computed because $\tau_n\phi_{n+2}$ is available from relation (7.7). Then write

$$r_{n+2} = \tau_{n+2}\phi_{n+2}r_0 = (I-\omega_{n+2}A)\tau_{n+1}\phi_{n+2}r_0 = (I-\omega_{n+2}A)u_{n+2} \tag{7.13}$$

and minimize

$$\|r_{n+2}^2\| = ((I-\omega_{n+2}A)u_{n+2})^T((I-\omega_{n+2}A)u_{n+2})$$

by choosing

$$\omega_{n+2} = (Au_{n+2}, u_{n+2})/(Au_{n+2}, Au_{n+2}). \tag{7.14}$$

Finally, in running CS-CGSTAB, we must be able to recover the BCG constants $\rho_n^{BCG}$ and $\sigma_n^{BCG}$ in order to update the BCG polynomial part of the residual. Recall in Section 2.4.2, we showed the relation due to Van der Vorst [55]:

$$\rho_{n+1}^{BCG} = \frac{\alpha_0\cdots\alpha_n}{\omega_1\cdots\omega_{n+1}}\ \rho_{n+1}^{Bi-CGSTAB},\ \text{where}\ \alpha_n = \frac{\rho_n^{BCG}}{\sigma_n^{BCG}}\ .$$

If we let $\mu_n = (\alpha_0\cdots\alpha_{n-1})/(\omega_1\cdots\omega_n)$, this reduces to the update formula:

$$\rho_{n+1}^{BCG} = \mu_{n+1}\rho_{n+1}^{Bi-CGSTAB} = \mu_n\left(\frac{\rho_n^{BCG}}{\sigma_n^{BCG}\omega_{n+1}}\right)\rho_{n+1}^{Bi-CGSTAB}.$$

In a $2 \times 2$ step, we determine $\rho_{n+2}^{CSBCG}$ similarly. First we use relation (7.4):

$$\begin{aligned}\rho_{n+2}^{CSBCG} &= (\phi_{n+2}(A^T)\tilde{r}_0, \phi_{n+2}(A)r_0) \\ &= ((\phi_n(A^T) - \alpha_n A^T \psi_n(A^T) \\ &\quad -\alpha_{n+1} A^T \xi_{n+1}(A^T))\tilde{r}_0, \phi_{n+2}(A)r_0).\end{aligned} \tag{7.15}$$

Then, the fact that $\phi_{n+2}r_0$ is orthogonal to all vectors $\chi_{n+1}(A^T)\tilde{r}_0$ implies that the inner product (7.15) picks out the coefficient of the highest order term of the $n+2$ degree polynomial that $\phi_{n+2}$ is being orthogonalized against. In this case, the coefficient of the highest order term for the polynomial $\xi_{n+1}(A^T) = \sigma_n^{BCG}\phi_{n+1}(A^T)$ is $\sigma_n^{BCG}\alpha_0 \cdots \alpha_n$, so (7.15) reduces to:

$$\rho_{n+2}^{CSBCG} = -\alpha_0 \cdots \alpha_{n+1} \sigma_n^{BCG}((-A^T)^{n+2}\tilde{r}_0, \phi_{n+2}(A)r_0)$$

which leads to the update formula:

$$\rho_{n+2}^{CSBCG} = \mu_{n+2}\rho_{n+2}^{CS-CGSTAB} = -\mu_n \left( \frac{\rho_n^{CSBCG}\alpha_{n+1}}{\omega_{n+1}\omega_{n+2}} \right) \rho_{n+2}^{CS-CGSTAB}. \tag{7.16}$$

The update for $\sigma_n^{BCG} \equiv (\psi_n(A^T)\tilde{r}_0, A\psi_n(A)r_0)$ can be calculated similarly. We define $\sigma_n^{CS-CGSTAB} \equiv (\tilde{r}_0, A\tau_n(A)\psi_n(A)r_0) = (\tilde{r}_0, Ap_n^{CS-CGSTAB})$ and obtain the analogous update formula:

$$\sigma_{n+2}^{CSBCG} = \mu_{n+2}\sigma_{n+2}^{CS-CGSTAB}. \tag{7.17}$$

### 7.1.1 CS-CGSTAB Stepping Strategy

As far as deciding when to actually take a $2 \times 2$ step, we follow the principles used in Chapters 5 and 6 for the CSBCG and CSCGS methods. As with the other composite step methods, CS-CGSTAB employs a practical stepping strategy that will skip over exact breakdowns using criterion (5.9).

Analogous to CSCGS, in order to avoid repeating work and to do this in a stable way, Condition (5.9a) can be written:

$$\|(I - \omega_{n+1}A)u_{n+1}\| < |\sigma_n|\|r_n\|.$$

For Condition (5.9b), we first rescale the $r_{n+2}$ update in order to estimate $\|r_{n+2}\|$ stably by letting

$$\nu_{n+2} \equiv \delta_n r_{n+2} = \delta_n(I - \omega_{n+1}A)(I - \omega_{n+2}A)s_{n+2},$$

where $\delta_n$ is the determinant of the $2 \times 2$ matrix in (7.9). Evaluating $\nu_{n+2}$ exactly would involve the quantity $A^2 s_{n+2}$ which would require an additional matrix-vector multiplication if we decide to take a $1 \times 1$ step. Hence, for practical purposes, we use an upper bound approximation to estimate $\|\nu_{n+2}\|$. Note that although we do not have $A^2 s_{n+2}$, the quantity $As_{n+2}$ can be made available even if we take a $1 \times 1$ step, and can be done without an additional matrix-vector product using the precomputed values $e_n$, $c_n$, and $d_{n+1}$:

$$
\begin{aligned}
(7.18) \qquad t_{n+2} &\equiv As_{n+2} \\
&= A(\delta_n r_n - \alpha_n q_n - \alpha_{n+1} y_{n+1}) \\
&= \delta_n e_n - \alpha_n c_n - \alpha_{n+1} d_{n+1}.
\end{aligned}
$$

Thus, we would like to estimate $\|\nu_{n+2}\| = \|\delta_n(I - \omega_{n+1}A)(I - \omega_{n+2}A)s_{n+2}\|$ using (7.18) and without any further matrix-vector multiplications. Our strategy is to set $\omega_{n+2} = 0$, thereby eliminating the need to compute $A^2 s_{n+2}$. Doing this gives an upper bound estimate to $\|\nu_{n+2}\|$ :

$$\|\nu_{n+2}\| \leq \|\delta_n(I - \omega_{n+1}A)s_{n+2}\|.$$

Hence, Condition (5.9b): $\|r_{n+1}\| < \|r_{n+2}\|$ is evaluated by the approximated condition

$$(7.19) \qquad |\delta_n| \|(I - \omega_{n+1}A)u_{n+1}\| < |\sigma_n| \|\tilde{\nu}_{n+2}\|,$$

where $\|\tilde{\nu}_{n+2}\|$ is the estimated upper bound for $\|\nu_{n+2}/\delta_n\|$:

$$(7.20) \qquad \|\nu_{n+2}/\delta_n\| < \|\tilde{\nu}_{n+2}\| \equiv \|s_{n+2} - \omega_{n+1}t_{n+2}\|.$$

If a $2 \times 2$ step is chosen, then the true $\nu_{n+2}$ is evaluated. After $\nu_{n+2}$ is computed, we add one final check using the true $\|\nu_{n+2}\|$ to decide whether a $1 \times 1$ step should be taken instead.

Table 7.1: Notation for CS-CGSTAB

| vector | polynomial | where/how derived | where used |
|---|---|---|---|
| $r_n$ | $\tau_n\phi_n r_0$ | (7.1) | (7.6) (7.7) (7.9) |
| $p_n$ | $\tau_n\psi_n r_0$ | (7.2) | (7.8) |
| $u_{n+1}$ | $\tau_n\xi_{n+1} r_0$ | (7.6) | (7.7) (7.8) (7.11) |
| $s_{n+2}$ | $\tau_n\phi_{n+2} r_0$ | (7.7) | (7.1) (7.8) (7.12) |
| $u_{n+2}$ | $\tau_{n+1}\phi_{n+2} r_0$ | (7.12) | (7.13) |
| $e_n$ | $A\tau_n\phi_n r_0$ | $Ar_n$ | (7.9) |
| $q_n$ | $A\tau_n\psi_n r_0$ | $Ap_n$ | (7.6) (7.7) (7.9) (7.10) |
| $y_{n+1}$ | $A\tau_n\xi_{n+1} r_0$ | $Au_{n+1}$ | (7.9) (7.10) (7.11) |
| $t_{n+2}$ | $A\tau_n\phi_{n+2} r_0$ | $As_{n+2}$ | (7.10) (7.12) |
| $y_{n+2}$ | $A\tau_{n+1}\phi_{n+2} r_0$ | $Au_{n+2}$ | (7.13) |
| $c_n$ | $A^2\tau_n\psi_n r_0$ | $Aq_n$ | (7.9) (7.10) |
| $d_{n+1}$ | $A^2\tau_n\xi_{n+1} r_0$ | $Ay_{n+1}$ | (7.9) (7.10) |
| $v_{n+2}$ | $A^2\tau_n\phi_{n+2} r_0$ | $At_{n+2}$ | (7.10) |
| $\alpha_n$ | $f_n^{(1)}$ | (7.9) | (7.7) |
| $\alpha_{n+1}$ | $f_n^{(2)}$ | (7.9) | (7.7) |
| $\beta_n$ | $g_n^{(1)}$ | (7.10) | (7.8) |
| $\beta_{n_1}$ | $g_n^{(2)}$ | (7.10) | (7.8) |

### 7.1.2 Implementation Details

In Table 7.1, we summarize the notation we will be using in our implementation of the CS-CGSTAB algorithm. We list the vector used in the algorithm, its corresponding polynomial form, and the equations in which it is derived and used.

As with CSBCG and CSCGS, at every step, we must anticipate a $2\times2$ step even if we decide to take a $1 \times 1$ step. The stepping strategy is similar to the strategies described in Sections 5.3.1 and 6.2.1, so this implies that we must be able to approximate the residual of a $2 \times 2$ step regardless of the step size we ultimately choose. (The stepping strategy will be discussed in Section 7.1.1.) Recall that in one step of Bi-CGSTAB, only two matrix-vector multiplications are performed: $q_n = Ap_n$ and $y_{n+1} = Au_{n+1}$. From the third column of Table 7.1, we see that 8 matrix-vector products are required to do a $2 \times 2$ step which appears to be 4 more than 2 steps of Bi-CGSTAB.

However, the total 8 products can be reduced to 4 multiplications by precomputing certain values and absorbing them into vector updates rather than explicitly multiplying by $A$. Specifically, by precomputing $c_n \equiv Aq_n$, the multiplication $y_{n+1} \equiv Au_{n+1}$ can be written:

$$\begin{aligned} y_{n+1} &= \sigma_n e_n - \rho_n A q_n \\ &= \sigma_n e_n - \rho_n c_n. \end{aligned}$$

Similarly, if we have $d_{n+1} \equiv Ay_{n+1}$, then the product $e_{n+1} \equiv Ar_{n+1}$ can also be evaluated without having to multiply by $A$:

$$\begin{aligned} e_{n+1} &= (y_{n+1} - \omega_{n+1} Ay_{n+1})/\sigma_n \\ &= (y_{n+1} - \omega_{n+1} d_{n+1})/\sigma_n. \end{aligned}$$

Furthermore, $q_{n+1} \equiv Ap_{n+1}$ can also be updated:

$$\begin{aligned} q_{n+1} &= Ar_{n+1} + \beta_{n+1}(Ap_n - \omega_{n+1} Aq_n) \\ &= e_{n+1} + \beta_{n+1}(q_n - \omega_{n+1} c_n). \end{aligned}$$

Thus, the $1 \times 1$ step can still be performed with only 2 (pre)multiplications with $A$: $Aq_n$ and $Ay_{n+1}$. Moreover, by precomputing $v_{n+2} \equiv At_{n+2}$ and $q_{n+2} \equiv Ap_{n+2}$, we can update the remaining values in the $2 \times 2$ step in a similar fashion.

However, using this precomputing strategy to update

$$e_{n+2} \equiv Ar_{n+2} = A(I - \omega_{n+1} A)(I - \omega_{n+2} A)s_{n+2}$$

implies that we need $A^3 s_{n+2}$ which we do not have. Thus, the $2 \times 2$ step requires an additional matrix-vector multiplication: $w_{n+2} \equiv A^3 s_{n+2} = A(v_{n+2})$.

Hence, as in CSCGS, there are 5 matrix-vector multiplications for a $2 \times 2$ step, whereas in two steps of Bi-CGSTAB, only 4 are needed. This is the price we must pay for composite step. If we do not need to take many $2 \times 2$ steps in practice, this price will not be too costly.

In Table 7.2, we present the CS-CGSTAB algorithm. Note that if only $1 \times 1$ steps are taken, we have exactly the Bi-CGSTAB algorithm.

Table 7.2: **Algorithm CS-CGSTAB**

```
$\rho_0 = \tilde{r}_0^T r_0$;   $p_0 = r_0$;   $\phi_0 = \|r_0\|$;   $e_0 = q_0 = Ar_0$;   $\mu_0 = 1$;   $n \leftarrow 0$
While *method not converged yet* do:
  $\sigma_n = (\tilde{r}_0^T q_n)\mu_n$;   $c_n = Aq_n$                                    % Evaluate (7.17)
  $u_{n+1} = \sigma_n r_n - \rho_n q_n$;   $y_{n+1} = \sigma_n e_n - \rho_n c_n$;   $d_{n+1} = Ay_{n+1}$
  $\omega_{n+1} = (y_{n+1}, u_{n+1})/(y_{n+1}, y_{n+1})$
  $\hat{r}_{n+1} = u_{n+1} - \omega_{n+1} y_{n+1}$;   $\hat{e}_{n+1} = y_{n+1} - \omega_{n+1} d_{n+1}$;   $\psi_{n+1} = \|\hat{r}_{n+1}\|$
  % Decide whether to take a $1 \times 1$ step or a $2 \times 2$ step.
    If $\psi_{n+1} < |\sigma_n|\phi_n$,Then      % Condition (5.9a): $\|r_{n+1}\| < \|r_n\|$
      *one-step* $= 1$
    Else
      $a_{11} = \tilde{r}_0^T q_n$;   $a_{12} = \tilde{r}_0^T y_{n+1}$;   $a_{21} = \tilde{r}_0^T c_n$;   $a_{22} = \tilde{r}_0^T d_{n+1}$
      $\delta_n = a_{11}a_{22} - a_{12}a_{21}$;   $b_1 = \rho_n/\mu_n$;   $b_2 = \tilde{r}_0^T e_{n+1}$
      $\alpha_n = a_{22}b_1 - a_{12}b_2$;   $\alpha_{n+1} = -a_{21}b_1 + a_{11}b_2$                  % Solve (7.9)
      $s_{n+2} = \delta_n r_n - \alpha_n q_n - \alpha_{n+1} y_{n+1}$;   $t_{n+2} = \delta_n e_n - \alpha_n c_n - \alpha_{n+1} d_{n+1}$
      $\tilde{\nu}_{n+2} = \|s_{n+2} - \omega_{n+1} t_{n+2}\|$                                % Evaluate (7.20)
      If $|\delta_n|\psi_{n+1} < |\sigma_n|\tilde{\nu}_{n+2}$, Then     % Condition (5.9b): $\|r_{n+1}\| < \|r_{n+2}\|$
          *one-step* $= 1$
      Else                                                             % True $\nu_{n+2}$
          $v_{n+2} = At_{n+2}$;   $w_{n+2} = Av_{n+2}$;   $z_{n+2} = t_{n+2} - \omega_{n+1} v_{n+2}$
          $\omega_{n+2} = (z_{n+2}, u_{n+2})/(z_{n+2}, z_{n+2})$;   $\gamma_1 = -(\omega_{n+1} + \omega_{n+2})$;   $\gamma_2 = \omega_{n+1}\omega_{n+2}$
          $\hat{r}_{n+2} = s_{n+2} + \gamma_1 t_{n+2} + \gamma_2 v_{n+2}$;   $\nu_{n+2} = \|\hat{r}_{n+2}\|$
          $\hat{e}_{n+2} = t_{n+2} + \gamma_1 v_{n+2} + \gamma_2 w_{n+2}$
          If $|\delta_n|\psi_{n+1} < |\sigma_n|\nu_{n+2}$, Then                    % Re-test w/$\nu_{n+2}$
              *one-step* $= 1$
          Else
              *one-step* $= 0$
    End If;   End If;   End If
  % Compute next iterate.
    If *one-step*, Then          % *** Usual Bi-CGSTAB ***
      $r_{n+1} = \hat{r}_{n+1}/\sigma_n$;   $e_{n+1} = \hat{e}_{n+1}/\sigma_n$;   $\phi_{n+1} = \psi_{n+1}/\sigma_n$
      $x_{n+1} = x_n + (\rho_n p_n + \omega_{n+1} u_{n+1})/\sigma_n$
      $\mu_{n+1} = (\mu_n \rho_n)/(\sigma_n \omega_{n+1})$;   $\rho_{n+1} = (\tilde{r}_0^T r_{n+1})\mu_{n+1}$
      $\beta_{n+1} = \rho_{n+1}/\rho_n$
      $p_{n+1} = r_{n+1} + \beta_{n+1}(p_n - \omega_{n+1} q_n)$
      $q_{n+1} = e_{n+1} + \beta_{n+1}(q_n - \omega_{n+1} c_n)$
      $n \leftarrow n + 1$
    Else       % *** $2 \times 2$ step CSCGSTAB ***
      $r_{n+2} = \hat{r}_{n+2}/\delta_n$;   $e_{n+2} = \hat{e}_{n+2}/\delta_n$;   $\phi_{n+2} = \nu_{n+2}/\delta_n$
      $x_{n+2} = x_n + (\alpha_n p_n + \alpha_{n+1} u_{n+1} - \gamma_1 s_{n+2} - \gamma_2 t_{n+2})/\delta_n$
      $\mu_{n+2} = -\mu_n(\alpha_{n+1}\rho_n/\delta_n\gamma_2)$;   $\rho_{n+2} = (\tilde{r}_0^T r_{n+2})\mu_{n+2}$      % Evaluate (7.16)
      $b_1 = \tilde{r}_0^T t_{n+2}$;   $b_2 = \tilde{r}_0^T v_{n+2}$
      $\beta_n = (a_{22}b_1 - a_{12}b_2)/\delta_n$;   $\beta_{n+1} = (-a_{21}b_1 + a_{11}b_2)/\delta_n$   % Solve (7.10)
      $p_{n+2} = r_{n+2} - \beta_n(p_n + \gamma_1 q_n + \gamma_2 c_n) - \beta_{n+1}(u_{n+1} + \gamma_1 y_{n+1} + \gamma_2 d_{n+1})$
      $q_{n+2} = Ap_{n+2}$
      $n \leftarrow n + 2$
    End If
```

## 7.2 A Variant of CS-CGSTAB

A problem in the CS-CGSTAB method is that additional breakdowns may occur if $A$ is skew-symmetric or near breakdowns if it has complex eigenvalues with large imaginary parts. Suppose we are at step $n$ of the algorithm. In the case that $A$ is skew-symmetric, it can be easily checked that $\omega_{n+1} = \omega_{n+2} = 0$. The $2 \times 2$ step attempts to divide by the quantity $\gamma_2 = \omega_{n+1}\omega_{n+2}$ which causes a breakdown. For nearly skew-symmetric $A$, $\gamma_2$ will be small, thus causing near breakdown and numerical instability.

This can be cured by modifying the local minimization at that particular $2 \times 2$ step. The idea is to require

$$\|r_{n+2}\| = \min_{\varphi \in \mathcal{P}_2} \|\varphi(A)\tau_n(A)\phi_{n+2}(A)r_0\|, \tag{7.21}$$

where $\mathcal{P}_2$ is the set of all polynomials of degree at most 2 and $\varphi(0) = 1$.

Performing this minimization over two degrees of freedom was first presented in the BiCGSTAB2 algorithm by Gutknecht, [37]. The purpose was to cure problems that arise in the steepest descent part of Bi-CGSTAB due to eigenvalues of $A$ in the complex spectrum that are not approximated well with eq. (2.33). Hence, every other step of the BiCGSTAB2 method performs (7.21) in order to handle conjugate pairs of eigenvalues. (In the remaining steps of the BiCGSTAB2 algorithm, the usual Bi-CGSTAB update is taken.) The BiCGSTAB2 algorithm can be summarized:

$$\tau_n\phi_n \xrightarrow{\text{1-D min}} \tau_{n+1}\phi_{n+1} \xrightarrow{\text{2-D min}} \tau_{n+2}\phi_{n+2}.$$

Unfortunately, in the implementation presented in [37], the two-dimensional minimization steps of BiCGSTAB2 are computed based on the Bi-CGSTAB step immediately before them. For skew-symmetric $A$, this poses a similar breakdown problem to the one mentioned above because the Bi-CGSTAB step requires a division by $\omega_{n+1}$, which will be zero in this case.

Note that BiCGSTAB2 is mathematically equivalent to the BiCGSTAB($l$) algorithm, due to Sleijpen and Fokkema [53] in the case where $l = 2$, but the implementation is different. BiCGSTAB(2) does not compute the intermediate Bi-CGSTAB residual $r_{n+1} = \tau_{n+1}\phi_{n+1}r_0$. Instead, it uses an intermediate basis vector, $A\tau_n\phi_{n+1}r_0$, to generate the auxiliary residual $\hat{r}_{n+2} = \tau_n\phi_{n+2}r_0$. This algorithm can be summarized:

$$\tau_n\phi_n \xrightarrow{A\tau_n\phi_{n+1}} \tau_n\phi_{n+2} \xrightarrow{\text{2-D min}} \tau_{n+2}\phi_{n+2}.$$

By not involving $\tau_{n+1}$, BiCGSTAB(2) will not suffer the breakdown mentioned above. However, note that it is still prone to breakdown in the BCG $\phi_{n+1}$ part.

We now show how to overcome this breakdown problem. Recall that CS-CGSTAB already cures the BCG pivot breakdown in step $n + 1$, so all we need to do now is to show how to modify CS-CGSTAB so that it will overcome the $\tau_{n+1}$ breakdowns as well. The idea is to note that CS-CGSTAB has access to the $A\tau_n\phi_{n+1}r_0$ vector through the relationship $\xi_{n+1} = \sigma_n\phi_{n+1}$. We use it to compute the auxiliary residual $s_{n+2} = \tau_n \phi_{n+2} r_0$ to yield :

$$\tau_n \phi_n \xrightarrow{A\tau_n \xi_{n+1}} \tau_n \phi_{n+2} \xrightarrow{\text{2-D min}} \tau_{n+2} \phi_{n+2}.$$

The quantity $s_{n+2}$ is updated by (7.7) and the 2-dimensional minimization step can be performed by first writing the residual

$$r_{n+2} = s_{n+2} + R \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix},$$

where $R = [As_{n+2} \quad A^2 s_{n+2}] \equiv [t_{n+2} \quad v_{n+2}]$, and then minimizing $\|r_{n+2}\|$ by solving the system:

$$R^T R \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = -R^T s_{n+2}. \tag{7.22}$$

In the case where $A$ is skew-symmetric, the attempt to divide by $\gamma_2 = \omega_{n+1}\omega_{n+2}$ in the $2 \times 2$ step can now be performed without breakdown. In particular, if $A = -A^T$, then $\gamma_2 \equiv -(v_{n+2}, s_{n+2})/(v_{n+2}, v_{n+2}) \neq 0$ because the numerator $v_{n+2}^T s_{n+2} = s_{n+2}^T A^2 s_{n+2} > 0$.

We can now form a variant of the CS-CGSTAB algorithm which follows the former method in allowing $1 \times 1$ and $2 \times 2$ steps and differs only in the $2 \times 2$ step, performing instead the minimization over the 2 degrees of freedom described above.

We use the same stepping strategy (see Section 7.1.1), but we adjust the approximation for $\|\nu_{n+2}\|$ to get a tighter bound. In the norm

$$\|\nu_{n+2}\| = \|\delta_n (I - \omega_{n+1} A)(I - \omega_{n+2} A) s_{n+2}\|$$

we once again set $\omega_{n+2} = 0$ to avoid computing $A^2 s_{n+2}$, and additionally, we minimize the resulting

$$||\delta_n(I - \omega_{n+1} A) s_{n+2}||.$$

This is solved by

$$\omega_{n+1} = \tilde{\omega}_{n+1} = (t_{n+2}, s_{n+2})/(t_{n+2}, t_{n+2}),$$

where $t_{n+2} = A s_{n+2}$ is again updated without matrix-vector multiplications (7.18).

We incorporate this into the new variant, CS-CGSTAB2. This is a more stable implementation which will not breakdown when $A$ is skew-symmetric. Rather, in this case, provided there are no pivot breakdowns, it will always take $2 \times 2$ steps and will be equivalent to the BiCGSTAB(2) method. In general, the composite step methods differ from the others because they are variable step methods. Note that in CS-CGSTAB2, the composite step is used to skip over breakdowns in the $\tau_{n+1}$ polynomial as well as $\phi_{n+1}$. However, if there was no $\phi_{n+1}$ breakdown, and we took a $2 \times 2$ step in CS-CGSTAB2, then it is still possible that there could be pivot breakdown due to $\phi_{n+2}$. In principle, we can solve this by applying the composite step idea to Bi-CGSTAB2 and taking a $3 \times 3$ step when we forsee possible pivot breakdown because $\phi_{n+3}$ exists under our assumption of $det(H^{(0)}) \neq 0$. However, we will not pursue this in this thesis.

## 7.3 Numerical Experiments

All experiments are run in MATLAB 4.0 on a SUN Sparc station with machine precision about $10^{-16}$. In most cases, as expected, composite step methods behave similarly to their non-composite step counterparts. In terms of the number of iterations it takes to converge, composite step methods are never worse in almost all cases, and in terms of the number of matrix-vector products performed, the extra cost is minimal. Here, we present a few selected examples where composite step does make a significant improvement.

**Example 7.1.** We begin the numerical experiments with a contrived example to illustrate the superior numerical stability of composite step methods over those without composite step. Let $A$ be a modification of the matrix used in Sections 4.4 and 6.4, taken from [47]:

$$A = \begin{pmatrix} \epsilon & 1 \\ -1 & 2 \end{pmatrix} \otimes I_{N/2},$$

i.e., $A$ is a $N \times N$ block diagonal with $2 \times 2$ blocks, and $N = 40$. By choosing $b = (1 \quad 0 \quad 1 \quad 0 \quad \cdots)^T$ and a zero initial guess, we set $\sigma_0 = \epsilon$, and thus, we can forsee numerical problems with BCG polynomial based methods such as Bi-CGSTAB, BiCGSTAB2, and CGS when $\epsilon$ is small. Although these methods converge in 2 steps in exact arithmetic when $\epsilon \neq 0$, in finite precision, convergence gets increasingly unstable as $\epsilon$ decreases. Table 7.3 shows the relative error in the solution after 2 steps of BCG, CGS, and Bi-CGSTAB. Note that the loss of significant digits in BCG and Bi-CGSTAB is approximately proportional to $O(\epsilon^{-1})$ and the loss of digits in CGS is proportional to $O(\epsilon^{-2})$. The accuracy of CS-CGSTAB, CS-CGSTAB2, and CSCGS, the composite step CGS algorithm [18] is insensitive to $\epsilon$ and these three methods all converge in two steps with errors $\leq 10^{-16}$.

Table 7.3: Example 7.1

| Rel. error in the soln. after 2 steps (N=40) | | | | |
|---|---|---|---|---|
| $\epsilon$ | Bi-CGSTAB | BiCGSTAB2 | BiCGSTAB(2) | CGS |
| $10^{-4}$ | $1.5 \times 10^{-12}$ | $8.6 \times 10^{-13}$ | $2.5 \times 10^{-16}$ | $3.6 \times 10^{-8}$ |
| $10^{-8}$ | $4.9 \times 10^{-9}$ | $4.7 \times 10^{-9}$ | $1.0 \times 10^{-9}$ | $2.2 \times 10^{0}$ |
| $10^{-12}$ | $3.0 \times 10^{-5}$ | $7.3 \times 10^{-4}$ | $2.5 \times 10^{-4}$ | $3.0 \times 10^{8}$ |
| CS-CGSTAB, CS-CGSTAB2, and CSCGS all converge with errors $\leq 10^{-16}$. | | | | |

**Example 7.2.** Next we alter Example 7.1 slightly to show the advantage of CS-CGSTAB2 over CS-CGSTAB. Recall, CS-CGSTAB was developed to overcome breakdowns in cases where A is (nearly) skew-symmetric. Hence, if we change the last example so that

$$A = \begin{pmatrix} \epsilon & 1 \\ -1 & \epsilon \end{pmatrix} \otimes I_{N/2},$$

we see that as $\epsilon$ gets small, CS-CGSTAB exhibits poor numerical results whereas CS-CGSTAB2 converges in the first $2 \times 2$ step as in the Example 7.1. The results are given in Table 7.4.

Table 7.4: Example 7.2

| Rel. error in the soln. after 2 steps (N=40) | | | | | |
|---|---|---|---|---|---|
| $\epsilon$ | Bi-CGSTAB | BiCGSTAB2 | BiCGSTAB(2) | CGS | CS-CGSTAB |
| $10^{-4}$ | $2.1 \times 10^{-12}$ | $2.9 \times 10^{-13}$ | $2.9 \times 10^{-13}$ | $2.5 \times 10^{-8}$ | $2.3 \times 10^{-13}$ |
| $10^{-8}$ | $2.0 \times 10^{-8}$ | bd | $1.0 \times 10^{-8}$ | $1.0 \times 10^{0}$ | $7.6 \times 10^{-10}$ |
| $10^{-12}$ | $1.2 \times 10^{-4}$ | $2.7 \times 10^{1}$ | $1.0 \times 10^{-12}$ | $1.3 \times 10^{8}$ | bd |
| bd: Encountered breakdown | | | | | |
| CS-CGSTAB2 converged each time with error $\leq 10^{-16}$. | | | | | |

**Example 7.3.** To emphasize the point made in Example 7.2, we pick a random skew-symmetric matrix with dimension $N = 20$ and a random right hand side. All the methods mentioned above either diverge or break down, except for CS-CGSTAB2 and BiCGSTAB(2), which achieve residual tolerance $10^{-11}$ in 24 iterations.

**Example 7.4.** We now show an example using a matrix which comes from the Harwell-Boeing set of sparse test matrices [20]. It is a discretization of the convection-diffusion equation:

$$L(u) = -\Delta u + 100(xu_x + yu_y) - 100u$$

on the unit square for a $63 \times 63$ grid. We use a random right hand side, zero initial guess, and no preconditioning. Figure 7.1 plots the true residual norm for Bi-CGSTAB, BiCGSTAB(2), CS-CGSTAB, and CS-CGSTAB2 versus the number of iterations taken, and Figure 7.2 shows the number of matrix-vector products. We have chosen a right hand side which yields some numerical instability for Bi-CGSTAB due to a near pivot breakdown around step 135 which results in convergence stagnation. Taking composite steps in this case overcomes this problem. We also see the advantage of the 2-dimensional minimization steps in the convergence behavior of BiCGSTAB(2) and CS-CGSTAB2.

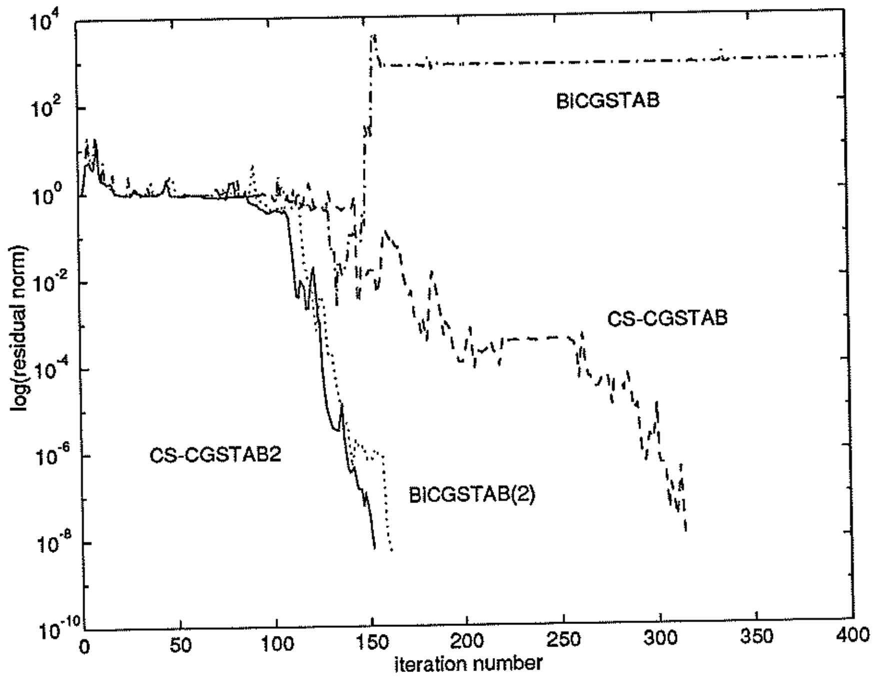

Figure 7.1: CS-CGSTAB – 2D conv-diff (Example 7.4a)

The stepping strategy that we have described and implemented is conservative in that a $2\times2$ step is chosen whenever there is a peak in the residual convergence. If the result of the composite step is only a slight improvement, the extra cost it takes to perform a $2\times2$ step would be wasted. However, in practice, this increase in cost is relatively small. For example, in this particular problem, 96 of the steps taken in CS-CGSTAB are $2\times2$ steps and 47 $2\times2$ steps are taken in CS-CGSTAB2. We see that the composite step methods require only about 13% more matrix-vector products in this example.

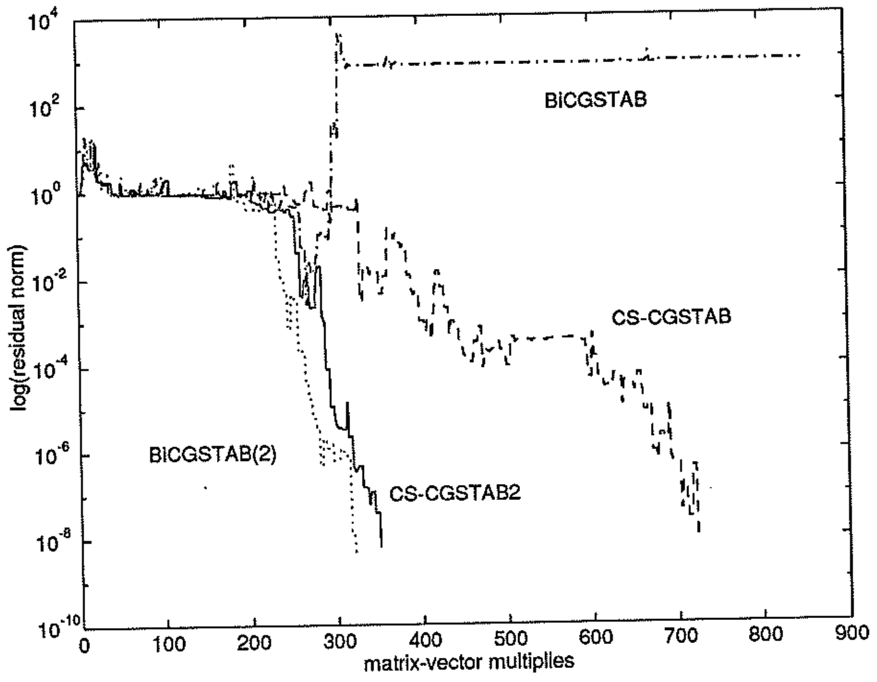

Figure 7.2: CS-CGSTAB – 2D conv-diff (Example 7.4b)

# CHAPTER 8

## Best Approximation Results for Product Methods

Until recently, there has been very little theory known on the convergence of the Biconjugate gradient algorithm or other related methods. When Bank and Chan introduced CSBCG in [6], they also included a proof of a "best approximation" result for BCG. It is based on an analysis by Aziz and Babuška [4] and is similar to the analysis of the Petrov-Galerkin methods in finite element theory. Specifically, if we let $M_k$ be any symmetric positive definite matrix and define the norm $|||v|||_*^2 = v^t M_k v$, then Bank and Chan showed that the BCG error term $e_k^{BCG} = x - x_k^{BCG} = \phi_k(A)e_0$ can be bounded as follows:

$$|||e_k^{BCG}|||_* \leq (1 + \Gamma/\delta) \inf_{\phi_k : \phi_k(0)=1} \|\phi_k(M_k^{\frac{1}{2}} \; AM_k^{-\frac{1}{2}} \;)\|_2 |||e_0|||_*, \tag{8.1}$$

where $\Gamma$ and $\delta$ are constants independent of $k$ determined from inequalities involving $v \in \mathcal{V}_k$ and $w \in \mathcal{W}_k$, where $\mathcal{V}_k$ and $\mathcal{W}_k$ are the Krylov subspaces generated by the Lanczos method at the $k$-th step (see Section 2.2):

$$|w^T Av| \leq \Gamma |||v|||_* |||w|||_*;$$

$$\inf_{\substack{v \in \mathcal{V}_k \\ |||v|||_* = 1}} \; \sup_{\substack{w \in \mathcal{W}_k \\ |||w|||_* = 1}} \; w^T Av \geq \delta_k > 0.$$

Moreover, if we define the Lanczos tridiagonal matrix $T_k = W_k^T AV^k$ and its LU-factorization $T_k = L_k D_k U_k$, and form $M_k = W_k U_k^T (D_k^T D_k)^{1/2} U_k W_k^T$, it can be shown [6] that $\Gamma = \delta = 1$.

This result establishes convergence of BCG in the case where there are no breakdowns because then $M_k$ is well-defined and symmetric positive definite. If this were not the case, the tridiagonal matrix $T_k$ would be singular and such an $M_k$ would not be positive definite. However, this result can be extended to cover situations with breakdown. For example, assuming no Lanczos breakdowns, the composite step approach does yield an $M_k$ matrix based on a factorization of $T_k$ which may involve $2 \times 2$ blocks, and hence, the above result applies to the error $e_k$ corresponding to the well-defined iterates $x_k$ [5]. In principle, if we add to this a look-ahead method to handle the Lanczos breakdowns, we can prove convergence of BCG for cases where both breakdowns occur.

Note that in general, simple upper bounds for the term

$$\inf_{\phi_k:\phi_k(0)=1} \|\phi_k(M_k^{\frac{1}{2}} \, A M_k^{-\frac{1}{2}})\|_2 \tag{8.2}$$

are known only for special cases. For example, if we assume that the eigenvalues of $A$ are contained in an ellipse in the complex plane which does not contain the origin, then, due to a result by Manteuffel [46], the quantity (8.2) can be bounded by a value dependent on the foci of the ellipse.

## 8.1 A Convergence Proof for CGS

The product methods discussed earlier (CGS and Bi-CGSTAB) both involve the BCG polynomial. Hence, we can use the result in (8.1) to establish bounds on these methods as well. We first prove a lemma which will be used in the derivation of both bounds for CGS and Bi-CGSTAB.

**Lemma 1** *For any matrix $A \in R^{N \times N}$ and vector $v \in R^N$,*

$$|||Av|||_* \leq ||M^{\frac{1}{2}} AM^{-\frac{1}{2}}||_2 |||v|||_*$$

*Proof.* *By definition, $|||w|||_* = ||M^{\frac{1}{2}} w||_2$,*

*and thus,*

$$\begin{aligned} |||Av|||_* &= ||M^{\frac{1}{2}} Av||_2 = ||M^{\frac{1}{2}} AM^{-\frac{1}{2}} M^{\frac{1}{2}} v||_2 \\ &\leq ||M^{\frac{1}{2}} AM^{-\frac{1}{2}}||_2 ||M^{\frac{1}{2}} v||_2 = ||M^{\frac{1}{2}} AM^{-\frac{1}{2}}||_2 |||v|||_*. \square \end{aligned}$$

We now use this to estimate a bound on the CGS method and show the squaring effect on the convergence rate.

**Theorem 1** *Let $e_k^{CGS} = \phi_k^2(A)e_0$. Then*

$$|||e_k^{CGS}|||_* \leq c_1 \left( \inf_{\phi_k : \phi_k(0)=1} ||\phi_k(M_k^{\frac{1}{2}} AM_k^{-\frac{1}{2}})||_2 \right)^2 |||e_0|||_*.$$

*Proof.* *Applying Lemma 1 to $|||e_k^{CGS}|||_*$, we get*

$$\begin{aligned}
|||e_k^{CGS}|||_* &= |||\phi_k^2(A)e_0|||_* \\
&\le \|\phi_k(M_k^{\frac{1}{2}}\ AM_k^{-\frac{1}{2}}\ )\|_2 |||\phi_k(A)e_0|||_* \\
&= \|\phi_k(M_k^{\frac{1}{2}}\ AM_k^{-\frac{1}{2}}\ )\|_2 |||e_k^{BCG}|||_* \\
&\le c_1 \left( \inf_{\phi_k:\phi_k(0)=1} \|\phi_k(M_k^{\frac{1}{2}}\ AM_k^{-\frac{1}{2}}\ )\|_2 \right)^2 |||e_0|||_*. \ \square
\end{aligned}$$

## 8.2 A Convergence Proof for Bi-CGSTAB

Next we show convergence of the Bi-CGSTAB method and how the convergence rate of BCG is decreased further by the effect of the steepest descent.

**Theorem 2** *Let $e_k^{BiCGSTAB} = \tau_k(A)\phi_k(A)e_0$. Also, Let $\tilde{A} = M^{\frac{1}{2}}\ AM_k^{-\frac{1}{2}}$ , and define $S$ to be the symmetric part of $\tilde{A}$ (i.e., $S = \frac{1}{2}\ (\tilde{A} + \tilde{A}^T)$). Then if $S$ is positive definite,*

$$|||e_k^{BiCGSTAB}|||_* \le c_2 \left(1 - \frac{\lambda_{min}(S)^2}{\lambda_{max}(\tilde{A}^T\tilde{A})}\right)^{\frac{k}{2}} \left( \inf_{\phi_k:\phi_k(0)=1} \|\phi_k(\tilde{A})\|_2 \right) |||e_0|||_*.$$

*Proof. First note that we can bound*

$$\min_{\tau_k \in P_k} \|\tau_k(\tilde{A})\|_2 \ \le \ \left(1 - \frac{\lambda_{min}(S)^2}{\lambda_{max}(\tilde{A}^T\tilde{A})}\right)^{\frac{k}{2}}$$

*by applying the proof in Theorem 3.3 of (Eisenstat, Elman, and Schultz, [21]) to the matrix $\tilde{A}$. Combining this fact with Lemma 1, we can establish the following* *bound.*

$$\begin{aligned}
|||e_k^{BiCGSTAB}|||_* &= \min_{\tau_k \in P_k} |||\tau_k(A)\phi_k(A)e_0|||_* \\
&\leq \min_{\tau_k \in P_k} \left( \|\tau_k(M_k^{\frac{1}{2}} \; AM_k^{-\frac{1}{2}} \;)\|_2 |||\phi_k(A)e_0|||_* \right) \\
&\leq \left( \min_{\tau_k \in P_k} \|\tau_k(\tilde{A})\|_2 \right) |||\phi_k(A)e_0|||_* \\
&\leq \left( 1 - \frac{\lambda_{min}(S)^2}{\lambda_{max}(\tilde{A}^T\tilde{A})} \right)^{\frac{k}{2}} |||e_k^{BCG}|||_* \\
&\leq c_2 \left( 1 - \frac{\lambda_{min}(S)^2}{\lambda_{max}(\tilde{A}^T\tilde{A})} \right)^{\frac{k}{2}} \left( \inf_{\phi_k : \phi_k(0)=1} \|\phi_k(\tilde{A})\|_2 \right) |||e_0|||_* . \; \square
\end{aligned}$$

Recently, Barth and Manteuffel [8] have shown that the constant $(1 + \Gamma/\delta)$ in (8.1) can be improved to 1. In other words, $e_k^{BCG}$ is minimized over $K_n$ in the $||| \cdot |||_*$ norm. Correspondingly, the constants $c_1$ and $c_2$ in Theorems 1 and 2 can be improved to 1.

# CHAPTER 9

## Conclusions

In this dissertation, we have considered several aspects of solving nonsymmetric linear systems using Krylov subspace iterative methods. We have developed new algorithms, observed their practical convergence behavior, and proved theorems of convergence.

The new algorithms we have introduced can all be classified as product methods. Furthermore, they all share the good properties of being transpose-free and using short recurrences to minimize work and storage. These new methods attempt to stabilize and improve on existing methods by curing problems which are inherited from the individual polynomials which form the residual polynomial product.

We have shown 2 ways of applying the QMR technique to smooth the erratic convergence behavior due to the BCG polynomial. We first squared the QMR polynomial to get the QMRS algorithm. This is a transpose-free method that advances 2 degrees of the Krylov subspace at each iteration with 3 matrix-vector multiplications, an improvement over the 4 multiplications required in QMR and BCG. Practically, QMRS is shown to be more stable than CGS and typically converges at about the same rate as TFQMR. Note, however, that CGS and TFQMR only require 2 matrix-vector products to gain 2 degrees in the Krylov subspace. Hence, QMRS lies between BCG/QMR and CGS in terms of efficiency.

We have also derived two QMR variants of Bi-CGSTAB. Our motivation for these methods was to inherit any potential improvements on performance that Bi-CGSTAB offers over CGS, while at the same time, to provide smoother convergence behavior. We have shown numerically that this is indeed true for many realistic problems. Although, in their present form, the two proposed methods still suffer from some numerical problems, they have make efficient use of matrix-vector multiplications and they demonstrate smooth convergence behavior.

In addition to QMR, we have dicussed other smoothing techniques. It was shown how the MRS algorithm could be applied to the CSCGS algorithm and numerical examples illustrate this smoothing effect.

We have also presented methods which handle breakdown problems inherited from BCG. The composite step technique was originally introduced to cure the pivot breakdown in BCG assuming there is no Lanczos breakdown. Our new methods improve on the product methods CGS and Bi-CGSTAB by using the composite step technique to increase numerical stability. These new methods require only minor modifications to the existing algorithms, overcome near and exact pivot breakdowns, and smooth the residual without requiring any user specified tolerance parameters.

Moreover, a variant of the composite step Bi-CGSTAB algorithm was introduced which simultaneously cures additional breakdown in the underlying Bi-

CGSTAB process due to skew-symmetric $A$. Numerical experiments support the improved convergence behavior of the composite step methods over their non-composite step counterparts.

Finally, we have proven convergence rate estimates for the product methods CGS and Bi-CGSTAB. We have also shown how these proofs also hold for the composite step versions.

## Bibliography

[1] W. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quart. Appl. Math., 9(1851), pp. 17-29.

[2] S. Ashby, T. Manteuffel, and P. Saylor, *A taxonomy for conjugate gradient methods*, SIAM J. Numer. Anal., 27(1990), pp. 1542-1568.

[3] O. Axelsson, *A survey of preconditioned iterative methods for linear systems of algebraic equations*, BIT 25(1985), pp. 166-187.

[4] Babuska and Aziz, *Part I, survey lectures on the mathematical foundations of the finite element method,* in The Mathematical Foundations of the Finite Element Method with Applications to Partial Differential Equations, Academic Press, New York, 1972.

[5] R. E. Bank and T. F. Chan, *An analysis of the composite step bi-conjugate gradient algorithm for solving nonsymmetric systems*, Numer. Math., 66(1993), pp. 295-319.

[6] R. E. Bank and T. F. Chan, *A composite step bi-conjugate gradient algorithm for solving nonsymmetric systems*, UCLA CAM Tech. Report 93-21 (1993). To appear in Numerical Algorithms.

[7] R. Barrett, et al., *Templates for the solution of linear systems: building blocks for iterative methods*, SIAM Publications, Philadelphia, 1994.

[8] T. Barth and T. Manteuffel, *Variable metric conjugate gradient methods*, Paper presented at the Colorado Conference on Iterative Methods, Breckenridge, CO, 1994.

[9] C. Brezinski and H. Sadok, *Avoiding breakdown in the CGS algorithm*, Numer. Alg. 1(1991), pp. 199-206.

[10] C. Brezinski and H. Sadok, *Lanczos-type algorithms for solving systems of linear equations*, Applied Numerical Mathematics, 11(1993), pp. 443-473.

[11] C. Brezinski and M. Redivo-Zaglia, *Breakdowns in the computation of orthogonal polynomials*, Nonlinear Numerical Methods and Rational Approximation, A. Cuyt ed., Kluwer, Dordrecht. To appear.

[12] C. Brezinski and M. Redivo-Zaglia, *Hybrid procedures for solving linear systems*, To appear in Numer. Math. (1993).

[13] C. Brezinski and M. Redivo-Zaglia, *Treatment of near-breakdown in the CGS algorithm*, To appear in Numer. Alg.

[14] C. Brezinski and M. Redivo-Zaglia and H. Sadok, *A breakdown-free Lanczos type algorithm for solving linear systems*, Numer. Math. 63(1992), pp. 29-38.

[15] C. Brezinski and M. Redivo-Zaglia and H. Sadok, *Avoiding breakdown and near-breakdown in Lanczos type algorithms*, Numer. Alg. 1(1991), pp. 261-284.

[16] T.F. Chan, L. de Pillis, and H.A. Van der Vorst, *A transpose-free squared Lanczos algorithm and application to solving nonsymmetric linear systems*, Technical Report CAM 91-17, Department of Mathematics, University of California, Los Angeles, CA, 1991.

[17] T. F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, C. Tong, *QMRCGSTAB: A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems*, SIAM J. Sci. Stat. Comput. 15(1993), pp. 338-347.

[18] T. F. Chan and T. Szeto, *A composite step conjugate gradients squared algorithm for solving nonsymmetric linear systems*, UCLA CAM Tech. Report 93-27 (1993). To appear in Numerical Alg.

[19] A. Draux, *Polynômes orthognaux formels*, Lect. Notes in Math., v. 974, Springer Verlag, Berlin, 1983.

[20] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Softw., 15(1989), pp. 1-14.

[21] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. of Numer.

Anal., 20(1983), pp. 345-357.

[22] H. C. ELMAN, *Iterative methods for large sparse symmetric systems of linear equations*, Ph.D. Thesis, Yale Univ., New Haven, CT, 1982.

[23] V. FABER AND T. MANTEUFFEL, *Necessary and sufficient conditions for the existence of a conjugate gradient method*, SIAM J. Numer. Anal., 21(1984), pp. 352-362.

[24] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, in Numerical Analysis Dundee 1975, G.A. Watson, ed., Lecture Notes in Mathematics 506, Springer, Berlin, 1976, pp. 73–89.

[25] R. W. FREUND, *Quasi-kernal polynomials amd convergence results for quasi-minimal residual iterations*, Numerical Methods in Approximation Theory, Vol. 9, D. Braess and L. L. Schumaker, eds., Birkhäuser Verlag, Basel, 1992, pp. 77-95.

[26] R. W. FREUND, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Stat. Comput., 14(1993), pp. 470-482.

[27] R. W. FREUND AND M.H. GUTKNECHT AND N. M. NACHTIGAL, *An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices*, SIAM J. Sci. Comp., 13 (1992), pp. 137-158.

[28] R. W. FREUND AND N. M. NACHTIGAL, *An implementation of the QMR method based on coupled two-term recurrences*, RIACS Tech. Report 92.15,

June 1992. SIAM J. Sci. Stat. Comput.

[29] R. W. FREUND AND N. M. NACHTIGAL, *QMR: a quasi-minimal residual method for non-Hermitian linear systems*, Numerische Mathematik 60(1991), pp. 315-339.

[30] R. W. FREUND AND N. M. NACHTIGAL, *A Look ahead TFQMR Method*, Presented at the Cornelius Lanczos International Centenary Conference, Raleigh, NC, December 1993.

[31] R. W. FREUND AND T. SZETO, *A Quasi-minimal residual squared algorithm for non-Hermitian linear systems*, UCLA CAM Tech. Report 92-19 (1992). Presented at the Copper Mountain Conference on Iterative Methods, April 1992.

[32] R.W. FREUND AND H. ZHA, *Simplifications of the nonsymmetric Lanczos process and a new algorithm for solving indefinite symmetric linear systems*, Technical Report, RIACS, NASA Ames Research Center, Moffett Field, CA, 1992.

[33] J. R. GILBERT, C. MOLER, AND R. SCHREIBER, *Sparse matrices in MATLAB: Design and implementation*, SIAM J. Matrix Anal. Appl., 13 (Jan. 1992), pp. 333–356.

[34] A. GREENBAUM, *Accuracy of computed solutions from conjugate-gradient-like methods*, Proceedings for the International Symposium PCG'94 on Ma-

trix Analysis and Parallel Computing, Yokohama, Japan, March 1994.

[35] M. H. GUTKNECHT, *A completed theory of the unsymmetric Lanczos process and related algorithms, Part I*, SIAM J. Matrix Anal. Appl. 13(1992), pp. 594-639.

[36] M. H. GUTKNECHT, *The unsymmetric Lanczos algorithms and their relations to Pade approximation, continued fraction and the QD algorithm*, in Proc. of the Copper Mt. Conf. on Iterative Methods, 1990.

[37] M. H. GUTKNECHT, *Variants of BiCGStab for matrices with complex spectrum*, tech. rep., Eidgenössische Technische Hochschule, Zürich, Aug. 1991. IPS Research Report 91-14.

[38] L. HAGEMAN AND D. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[39] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand., 49(1952), pp. 409-436.

[40] K. C. JEA AND D. M. YOUNG, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl, 34(1980), pp. 159-194.

[41] W. JOUBERT, *Generalized conjugate gradient and Lanczos methods for the solution of nonsymmetric systems of linear equations*, Ph.D. Thesis, The University of Texas at Austin, Austin, TX (1990).

[42] W. D. Joubert and T. A. Manteuffel, *Iterative methods for nonsymmetric linear systems*, in Iterative Methods for Large Linear Systems, D. R. Kincaid and L. J. Hayes, eds., Academic Press, Boston, 1990, pp. 149–171.

[43] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Natl. Bur. Stand., 45 (1950), pp. 255–282.

[44] C. Lanczos, *Solution of linear equations by minimized iterations*, J. Res. Natl. Bur. Stand. 49 (1952), pp. 33-53.

[45] D. G. Luenberger, *Hyperbolic Pairs in the Method of Conjugate Gradients*, SIAM J. Appl. Math. 17(1969), pp.1263-1267.

[46] T. Manteuffel, *An iterative method for solving nonsymmetric linear systems with dynamic estimation of parameters*, Ph.D. Thesis, University of Illinois, Urbana, 1975.

[47] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, *How fast are nonsymmetric matrix iterations?*, SIAM J. Matrix Anal. Appl., 13(1992), pp. 778-795.

[48] B. N. Parlett, D. R. Taylor, and Z. A. Liu, *A look-ahead Lanczos algorithm for unsymmetric matrices*, Math. Comp., 44(1985), pp.105-124.

[49] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14(1993), pp. 461-469.

[50] Y. SAAD, *ILUT: a dual threshhold incomplete LU factorization*, Research Report UMSI 92/38, University of Minnesota Supercomputer Institute, Minneapolis, MN, March 1992.

[51] Y. SAAD, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. of Numer. Anal., 19(1982), pp. 485-506.

[52] W. SCHÖNAUER, *Scientific computing on vector computers*, North-Holland, Amsterdam, New York, Oxford, Tokyo (1987).

[53] G. SLEIJPEN AND D. FOKKEMA, *BICGSTAB(L) for linear equations involving unsymmetric matrices with complex spectrum*, ETNA, 1(1993), pp. 11-32.

[54] P. SONNEVELD, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 10(Jan 1989), pp. 36-52.

[55] H. A. VAN DER VORST, *Bi-CGSTAB: A fast and smoothly converging variant of bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13(March 1992), pp. 631-644.

[56] H. A. VAN DER VORST, *The convergence behaviour of preconditioned CG and CG-S in the presence of rounding errors*, in Preconditioned Conjugate Gradient Methods, O. Axelsson and L. Yu. Kolotilina (eds.), Lecture Notes in Mathematics 1457, Springer Verlag, Berlin, 1990.

[57] R. VARGA, *Matrix iterative analysis*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1962.

[58] C. VUIK, *A comparison of some GMRES-like methods*, Tech. rept., Delft University of Technology, Delft.

[59] H. F. WALKER, *Implementation og the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., 9(1988), pp. 152-163.

[60] R. WEISS, *Convergence behavior of generalized conjugate gradient methods*, Ph. D. thesis, University of Karlsruhe (1990).

[61] D. YOUNG, *Iterative solution of large linear systems,* Academic Press, New York, 1971.

[62] L. ZHOU AND H. F. WALKER, *Residual smoothing techniques for iterative methods*, Tech. rep., Dept. of Mathematics and Statistics, Utah State University (1992).