

Krylov Solvers for Linear Algebraic Systems

*Charles George Broyden
Maria Teresa Vespucci*

Elsevier

KRYLOV SOLVERS
FOR LINEAR ALGEBRAIC SYSTEMS

Krylov Solvers

STUDIES IN COMPUTATIONAL MATHEMATICS 11

Editors:

C.K. CHUI

*Stanford University
Stanford, CA, USA*

P. MONK

*University of Delaware
Newark, DE, USA*

L. WUYTACK

*University of Antwerp
Antwerp, Belgium*



ELSEVIER

Amsterdam – Boston – Heidelberg – London – New York – Oxford – Paris
San Diego – San Francisco – Singapore – Sydney – Tokyo

KRYLOV SOLVERS FOR LINEAR ALGEBRAIC SYSTEMS

Krylov Solvers

Charles George BROYDEN
University of Bologna
Italy

Maria Teresa VESPUCCI
University of Bergamo
Italy



2004

Amsterdam – Boston – Heidelberg – London – New York – Oxford – Paris
San Diego – San Francisco – Singapore – Sydney – Tokyo

ELSEVIER B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE
Amsterdam, The Netherlands

ELSEVIER Inc.
525 B Street
Suite 1900, San Diego
CA 92101-4495, USA

ELSEVIER Ltd.
The Boulevard
Langford Lane, Kidlington,
Oxford OX5 1GB, UK

ELSEVIER Ltd.
84 Theobalds Road
London WC1X 8RR
UK

© 2004 Elsevier B.V. All rights reserved.

This work is protected under copyright by Elsevier B.V., and the following terms and conditions apply to its use:

Photocopying

Single photocopies of single chapters may be made for personal use as allowed by national copyright laws. Permission of the Publisher and payment of a fee is required for all other photocopying, including multiple or systematic copying, copying for advertising or promotional purposes, resale, and all forms of document delivery. Special rates are available for educational institutions that wish to make photocopies for non-profit educational classroom use.

Permissions may be sought directly from Elsevier's Rights Department in Oxford, UK: phone (+44) 1865 843830, fax (+44) 1865 853333, e-mail: permissions@elsevier.com. Requests may also be completed on-line via the Elsevier homepage (<http://www.elsevier.com/locate/permissions>).

In the USA, users may clear permissions and make payments through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA; phone: (+1) (978) 7508400, fax: (+1) (978) 7504744, and in the UK through the Copyright Licensing Agency Rapid Clearance Service (CLARCS), 90 Tottenham Court Road, London W1P 0LP, UK; phone: (+44) 20 7631 5555; fax: (+44) 20 7631 5500. Other countries may have a local reprographic rights agency for payments.

Derivative Works

Tables of contents may be reproduced for internal circulation, but permission of the Publisher is required for external resale or distribution of such material. Permission of the Publisher is required for all other derivative works, including compilations and translations.

Electronic Storage or Usage

Permission of the Publisher is required to store or use electronically any material contained in this work, including any chapter or part of a chapter.

Except as outlined above, no part of this work may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the Publisher.
Address permissions requests to: Elsevier's Rights Department, at the fax and e-mail addresses noted above.

Notice

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein. Because of rapid advances in the medical sciences, in particular, independent verification of diagnoses and drug dosages should be made.

First edition 2004

Library of Congress Cataloging in Publication Data
A catalog record is available from the Library of Congress.

British Library Cataloguing in Publication Data
A catalogue record is available from the British Library.

ISBN: 0-444-51474-0
ISSN: 1570-579X

 The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).
Printed in The Netherlands.

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER **BOOK AID**
International **Sabre Foundation**

To Caterina, Tom, Matty, Ben, Will, Francesca and Luc.

This page intentionally left blank

Preface

If any one event could be said to be the starting-point of the Odyssey that led to the publication of this book it would be the meeting of numerical linear algebraists organised by Emilio Spedicato at “Il Ciocco” in the Garfagnana region of Italy in 1990. Among the speakers was Csaba Hegedüs who gave an early description of some of the methods discussed later in the main text. These were sufficiently similar to the well-known biconjugate gradient (BiCG) method to make me wonder just what was the relationship between his “Galerkin” method and BiCG, and how the differences between these two methods were reflected in their relative performances. I soon realised that there were similarities between many of the Krylov methods, most of which had already been noted by other authors. However none of these other comparisons included Csaba’s methods (they were too recent) so I resolved to try to construct a comprehensive and unified theory of all known Krylov methods for solving linear equations. This task, on and off, occupied me for the next twelve years or so, working at first alone and subsequently with Dott.ssa Maria Teresa Vespucci. It was helped by the doctoral courses that I gave during this time at the University of Padua on “Krylov methods”, a somewhat salutary but most useful experience. There is nothing quite like blank faces staring up at you to bring home the fact that your latest clever proof of an important theorem is not as transparent as you had thought. Indeed without the inadvertent help of the students, this book would have been considerably more opaque than it actually is.

John Donne wrote that “No man is an island” and this is particularly true of authors of non-fiction. We are perhaps more indebted than most for outside help. In our specific case this came from many people, ranging from good friends on the one hand to previously unknown colleagues answering pleas for help on the NA-net on the other.

It gives us special pleasure to thank two people in particular for reading and criticising early drafts of the manuscript. They are, in chronological order, Aurel Galantai of the University of Miskolc, Hungary and Michele Benzi of Emory University, Atlanta, USA. Aurel read the initial drafts of the first few chapters and as a result of his advice the style and presentation of the book was established in its present form. The suggestions that he made at the outset determined the tone of the whole enterprise. Later Michele not only made available to us his enormous expertise in preconditioning but went far beyond the call of friendship in reading and re-reading Chapter 11. He rescued us from many errors of fact and interpre-

tation and if, as occasionally happened, we deliberately ignored his advice we can only hope that the resulting gain (if any) of one aspect of our presentation goes some way to compensate for the losses suffered by others. For the consequences of such decisions we accept full responsibility, as indeed we do for any other errors, omissions and shortcomings appearing elsewhere in the text. Even infelicitous page appearances can now, in these days of TeX and camera-ready copy, be laid at our door.

More indirect help, not specifically related to the text, was also important to us. Claude Brezinski encouraged us to write the book in the first place and the staffs of the “biblioteche” in Cesena, Bologna and Bergamo ensured that vital source materials were obtained promptly and efficiently. Some of these references were suggested by others and we thank those colleagues (in the most general sense) who either drew our attention to them or helped track down some of the more obscure ones that we had discovered for ourselves. We are grateful to Renato Zanovello of the University of Padua who allowed me to try out my ideas on his doctoral students and to two more students, Marco Boschetti and Paolo Foschi, for their contributions to chapters ten and twelve respectively.

Although many of our in-house algorithms were checked against those available in MATLAB we were not able to do this in the case of the Hegedüs algorithms. We are therefore grateful to Csaba for giving us access to his own programs and for his comments, generous and enlightening, over the years. We also thank the dozen or so colleagues who responded to our NA-net plea for help with algorithm testing. Although this part of the project had to be abandoned we still appreciate their willingness to help and their encouraging, quirky, comments.

It is unlikely that this project would ever have been started had I not been invited to Italy in the first place. For this and much else I am grateful both to Emilio Spedicato of the University of Bergamo and to Ilio Galligani of the University of Bologna. Emilio originally suggested that I apply for a post in Italy and Ilio made sure that, having been appointed, I came to Bologna. I am also grateful to Aristide Mingozzi of the Cesena campus of the University of Bologna for his help and friendship after my arrival in “il bel paese” and to CNR, the Italian Council of National Research, which gave me far more generous financial support than I ever received in my own country. Another debt of gratitude is owed to Mr. Ian Scott and his colleagues at the Ipswich Hospital, UK. Without his timely and skillful intervention this book would almost certainly never have seen the light of day and it is a pleasure to be able to record my appreciation here.

If we have inadvertently omitted to acknowledge any help received then we offer our apologies; we were only too grateful for all the outside assistance. Finally we thank our families for their understanding and forbearance for the duration of this labour. They can now have us back.

C. G. Broyden
February, 2004.

Contents

Preface	vii
Contents	ix
1 Introduction	1
1.1 Norm-reducing methods	3
1.2 The quasi-minimal residual (QMR) technique	8
1.3 Projection methods	10
1.4 Matrix equations	12
1.5 Some basic theory	13
1.6 The naming of algorithms	17
1.7 Historical notes	18
2 The long recurrences	21
2.1 The Gram-Schmidt method	22
2.2 Causes of breakdown*	24
2.3 Discussion and summary	26
2.4 Arnoldi's method	28
2.5 OrthoDir and GCR	32
2.6 FOM, GMRes and MinRes	35
2.7 Practical considerations	39
3 The short recurrences	43
3.1 The block-CG algorithm (BiCG)	43
3.2 Alternative forms	45
3.3 The original Lanczos method	47
3.4 Simple and compound algorithms	48
3.5 Galerkin algorithms	51
3.5.1 The conjugate gradient algorithm (CG)	51
3.5.2 The biconjugate gradient algorithm (BiCG)	52
3.5.3 The BiCGL algorithm	53
3.5.4 The Hegedüs “Galerkin” algorithm (HGL)	54
3.5.5 The Hegedüs “Galerkin” algorithm (HG)	54
3.5.6 The Concus-Golub-Widlund algorithm (CGW)	55

3.6 Minimum-residual algorithms	57
3.6.1 The conjugate residual algorithm (CR)	57
3.6.2 The modified conjugate residual algorithm (MCR)	58
3.6.3 The CG normal residuals algorithm (CGNR)	59
3.6.4 The biconjugate residual algorithm (BiCR)	60
3.6.5 The biconjugate residual algorithm (BiCRL)	61
3.7 Minimum-error algorithms	61
3.7.1 The method of orthogonal directions (OD)	62
3.7.2 The stabilised OD method (StOD)	63
3.7.3 The CG normal errors, or Craig's, algorithm (CGNE)	64
3.8 Lanczos-based methods	64
3.8.1 SymmLQ	65
3.8.2 LSQR and LSQR2	67
3.8.3 QMR (original version but without look-ahead)	69
3.9 Existence of short recurrences*	70
4 The Krylov aspects	77
4.1 Equivalent algorithms	84
4.2 Rates of convergence	87
4.3 More on GMRes	91
4.4 Special cases*	99
4.4.1 BiCG and BiCGL	99
4.4.2 QMR	102
5 Transpose-free methods	105
5.1 The conjugate-gradient squared method (CGS)	105
5.2 BiCGStab	109
5.3 Other algorithms	113
5.4 Discussion	114
6 More on QMR	117
6.1 The implementation of QMR, GMRes, SymmLQ and LSQR	117
6.2 QMRBiCG - an alternative form of QMR without look-ahead	120
6.3 Simplified (symmetric) QMR	122
6.4 QMR and BiCG	125
6.5 QMR and MRS	126
6.6 Discussion	131
7 Look-ahead methods	133
7.0.1 Note on notation	136
7.1 The computational versions	137
7.1.1 The look-ahead block Lanczos algorithm	138
7.1.2 The Hestenes-Stiefel version	139
7.2 Particular algorithms	142

7.3 More Krylov aspects*	145
7.4 Practical details	148
8 General block methods	151
8.1 Multiple systems	151
8.2 Single systems	157
9 Some numerical considerations	163
10 And in practice...?	173
10.1 Presenting the results	175
10.2 Choosing the examples	176
10.3 Computing the residuals	178
10.4 Scaling and starting	180
10.5 Types of failure	182
10.6 Heads over the parapet	183
10.6.1 HS versus Lanczos	183
10.6.2 Galerkin versus minimum residual	186
10.6.3 Do we need residual smoothing?	187
10.6.4 Do we need look-ahead?	188
10.6.5 Do we need preconditioning?	189
10.6.6 Do we need sophisticated linear algebra?	189
10.6.7 And the best algorithms...?	189
10.6.8 For symmetric systems	192
11 Preconditioning	193
11.1 Galerkin methods	195
11.2 Minimum residual methods*	203
11.3 Notation (again)	207
11.4 Polynomial preconditioning	207
11.4.1 An early example	208
11.4.2 General polynomial preconditioning	208
11.4.3 Neumann preconditioning	210
11.4.4 Chebychev preconditioning	212
11.4.5 Other polynomial preconditioners	215
11.5 Some non-negative matrix theory	219
11.6 (S)SOR preconditioners	226
11.6.1 SSOR	230
11.7 ILU preconditioning	231
11.7.1 Incomplete Cholesky (IC) preconditioning	233
11.7.2 DCR	235
11.7.3 ILU(p)	235
11.7.4 Threshold variants (ILUT)	243
11.7.5 Imposing symmetry	249
11.7.6 Tismenetsky's method	250

11.7.7 Numerical considerations	252
11.7.8 The impact of re-ordering	257
11.8 Methods for parallel computers	262
11.8.1 The AIM methods (SpAI, MR and MRP)	263
11.8.2 The AIF methods (FSAI, AIB and AInv)	267
11.8.3 The PP methods (JP and the TN methods)	270
11.8.4 The AIAF methods	271
11.8.5 Other methods	271
11.9 In conclusion	277
12 Duality	279
12.1 Interpretations	284
A Reduction of upper Hessenberg matrix to upper triangular form	287
B Schur complements	293
C The Jordan Form	295
D Chebychev polynomials	297
E The companion matrix	299
F The algorithms	301
G Guide to the graphs	313
References	315
Index	327

Chapter 1

Introduction

We wish to solve the system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.1}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a real large sparse matrix and \mathbf{b} is a real vector. Such equations occur frequently in science and engineering, either from naturally-occurring *sparse* systems such as girder frameworks or power-cable networks, or from attempts to solve partial differential equations by some sort of discretization procedure using either finite differences or finite elements. Normally \mathbf{A} will be square ($m = n$) and nonsingular but we cannot exclude the possibility that it is singular or even rectangular, in which case the solution we require will generally be a least-squares solution. Since \mathbf{A} is sparse we do not wish, for reasons of storage, to use a method that introduces non-zeroes into the matrix. At the same time, however, we do want to compute a theoretically exact solution after a finite number of steps. These two restrictions effectively rule out methods that modify the matrix, like Gaussian elimination or Cholesky decomposition, since these methods normally generate many extra nonzeroes. They also eliminate iterative methods like Jacobi or SOR, since these methods do not compute the exact solution. Thus the restrictions that we impose effectively exclude all the classical linear equation solvers from our consideration. We must therefore try to devise a completely different type of method, one that perhaps only requires the calculation of the matrix-vector product $\mathbf{A}\mathbf{y}$, (or $\mathbf{A}^T\mathbf{y}$), where \mathbf{y} is some vector that appears in the course of the calculation, since this is a “sparse” operation (even though we always assume vectors to be “dense”) and does not result in the creation of further non-zero elements in the matrix.

Since we do not wish to modify the coefficient matrix \mathbf{A} we are forced to consider methods which generate sequences of approximate solutions by adding the appropriate correction \mathbf{d}_i to the current approximation. Denote this latter by \mathbf{x}_i and let $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{d}_i$. The problem now is how to choose \mathbf{d}_i . There are two basic ways of doing this. The first is choosing a correction that reduces some norm of the error of the approximate solution. Methods based on this idea alone are essentially iterative and do not yield an exact solution after a finite number of steps even using exact

arithmetic. They thus do not satisfy one of the basic requirements proposed above but despite this they must be considered, not only because they include some of the most popular and successful algorithms but because they contain in them the basic idea which, when generalised, leads to the second strategy for choosing \mathbf{d}_i .

The second strategy is the successive reduction of the dimension (or rank) of the subspace in which the error of the approximate solution is forced to lie. This is done by effectively projecting the successive errors into subspaces of ever-decreasing dimension until finally the error is forced to be null. The convergence of this type of algorithm is generally not monotonic but with exact arithmetic the solution will be computed in a finite number of steps. Because there is no monotonic reduction of some error norm, algorithms based on this principle tend to be unstable and it is essential that ways of overcoming this instability be devised.

Finally it is possible to construct algorithms that embody both the above criteria but only for equations that have, or can be treated as having, a symmetric positive definite coefficient matrix. For such equations, reasonably successful algorithms are available but for equations with an indefinite symmetric or nonsymmetric coefficient matrix, compromises are inevitable and at the time of writing no one algorithm emerges as being clearly superior. We note here that the crude substitution of the normal equations

$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b} \quad (1.2)$$

for equation (1.1) is not a satisfactory response. Even though $\mathbf{A}^T \mathbf{A}$ is both symmetric and positive definite so that an algorithm giving the exact solution with monotonic norm reduction is, in principle, possible there can still be severe practical difficulties. For very large n , even with exact arithmetic, the amount of work needed to obtain an exact solution is often prohibitive since to obtain this solution the theoretical maximum of n iterations is often needed. The presence of rounding error generally increases this so that sometimes as many as $10n$ or more iterations are necessary before the current approximate solution is sufficiently accurate. It is therefore imperative that even the “exact” methods should converge rapidly and yield a solution of acceptable accuracy in far fewer than n iterations if at all possible.

Now the rate of convergence of virtually every iterative method, including some of the “exact methods” discussed below, is related to the condition number of the matrix of coefficients. If the condition number of \mathbf{A} is large, since the condition number of $\mathbf{A}^T \mathbf{A}$ is the square of the condition number of \mathbf{A} , the substitution of equation (1.2) for $\mathbf{A}\mathbf{x} = \mathbf{b}$ can do more harm than good. Again a compromise has to be sought and this book is about the properties of such compromises. We begin, then, by examining our two basic strategies, starting with methods that reduce some norm of the error.

1.1. Norm-reducing methods

In these methods the next approximate solution is chosen to reduce some measure of its error, and to do this we need two vectors that represent in some way the discrepancy between an arbitrary vector of independent variables \mathbf{x} and the solution \mathbf{x}^* of equation (1.1). The first of these is the *error vector* \mathbf{e} defined by

$$\mathbf{e} = \mathbf{x} - \mathbf{x}^* \quad (1.3)$$

and the second is the *residual vector* \mathbf{r} defined by

$$\mathbf{r} = \mathbf{Ax} - \mathbf{b} \quad (1.4)$$

where \mathbf{e} , \mathbf{r} and \mathbf{x} may be subscripted as appropriate. Since \mathbf{x}^* is unknown the error has only a theoretical significance but the residual vector may be computed for any \mathbf{x} at the cost of a matrix-vector multiplication. It can therefore be used as the measure of the accuracy of the current approximate solution \mathbf{x}_i . Clearly, if \mathbf{A} is square and nonsingular, the residuals and errors are related by

$$\mathbf{r} = \mathbf{A}\mathbf{e} \quad (1.5)$$

since the solution \mathbf{x}^* of equation (1.1) is then given by $\mathbf{A}^{-1}\mathbf{b}$. Now a scalar measure of the error of any approximate solution is the norm of the error vector, $\|\mathbf{e}\|$. Vector norms satisfy the three conditions

- $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$
- $\|\mathbf{x}\theta\| = \|\mathbf{x}\| |\theta|$ where θ is a scalar, real or complex, and
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (the triangle inequality)

Norms are thus a measure of the “size” of a vector, and an error norm that decreases to zero indicates an iteration that converges to the correct solution. The most common norm, one that will be denoted by $\|\cdot\|$ throughout, is the Euclidean norm defined by

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \mathbf{x}}.$$

Another norm that is of considerable importance in the field of conjugate gradients is the so-called *energy norm* $\|\mathbf{x}\|_G$ defined by

$$\|\mathbf{x}\|_G = \sqrt{\mathbf{x}^T \mathbf{G} \mathbf{x}}$$

where \mathbf{G} is some symmetric positive definite matrix. An alternative way of expressing this norm of the error is by writing $\mathbf{G} = \mathbf{B}^2$ where \mathbf{B} is symmetric, and defining \mathbf{s} by $\mathbf{s} = \mathbf{B}\mathbf{e}$ so that

$$\|\mathbf{e}\|_G = \|\mathbf{s}\|.$$

That such a \mathbf{B} exists follows from the fact that since \mathbf{G} is symmetric positive definite, $\mathbf{G} = \mathbf{X}\Lambda\mathbf{X}^T$ where \mathbf{X} is orthogonal, $\Lambda = \text{diag}(\lambda_i)$ and $\lambda_i > 0$, $1 \leq i \leq n$. Then, trivially, $\mathbf{B} = \mathbf{X}\Lambda^{\frac{1}{2}}\mathbf{X}^T$ where $\Lambda^{\frac{1}{2}}$ is a diagonal matrix whose diagonal elements are the square roots of those of Λ . If we choose these to be positive then

\mathbf{B} is also positive definite, and we shall subsequently refer to such a matrix as the *square root* of \mathbf{G} , or simply as $\mathbf{G}^{\frac{1}{2}}$. This equivalent form of $\|\mathbf{e}\|_G$ is used in Chapter 4 for establishing the rate of convergence for certain conjugate-gradient algorithms.

One reason for the importance of the energy norm $\|\mathbf{e}\|_G$ of the error is the special properties that it enjoys when applied to a system of equations whose matrix of coefficients is \mathbf{G} itself. For this reason we are prompted to consider the *equivalent system*

$$\mathbf{G}\mathbf{x} = \mathbf{h} \quad (1.6)$$

which is equivalent to $\mathbf{A}\mathbf{x} = \mathbf{b}$ in the sense that the solutions of both systems of equations are the same but where \mathbf{G} is symmetric positive definite. This implies that $\mathbf{G}^{-1}\mathbf{h} = \mathbf{A}^{-1}\mathbf{b}$, or

$$\mathbf{h} = \mathbf{G}\mathbf{A}^{-1}\mathbf{b}, \quad (1.7)$$

and one way of solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ is simply to solve the equivalent system. Of course if \mathbf{A} is symmetric and positive definite we can merely put $\mathbf{G} = \mathbf{A}$, but if \mathbf{A} is indefinite or nonsymmetric and we wish to make use of the special properties of $\|\mathbf{e}\|_G$ then we must choose another value for \mathbf{G} . Two common choices are $\mathbf{G} = \mathbf{A}^T\mathbf{A}$ if \mathbf{A} has full column rank and $\mathbf{G} = \mathbf{I}$, for which the corresponding values of \mathbf{h} are $\mathbf{A}^T\mathbf{b}$ and $\mathbf{A}^{-1}\mathbf{b}$. The choice $\mathbf{G} = \mathbf{I}$ presents certain problems since it is impossible to calculate a residual directly as it involves the (unknown) solution of equation (1.6) but these problems can be overcome by a certain amount of trickery.

We denote now the residual of the equivalent system by \mathbf{f} and define it by

$$\mathbf{f} = \mathbf{G}\mathbf{x} - \mathbf{h}, \quad (1.8)$$

subscripting both \mathbf{f} and \mathbf{x} as appropriate (we shall use this notation throughout and reserve the symbol \mathbf{r} , subscripted as necessary, for the true residual $\mathbf{Ax} - \mathbf{b}$). By analogy with equation (1.5), $\mathbf{f} = \mathbf{Ge}$ where $\mathbf{e} = \mathbf{x} - \mathbf{x}^*$ and this implies that

$$\|\mathbf{e}\|_G^2 = \mathbf{f}^T \mathbf{G}^{-1} \mathbf{f} = \|\mathbf{f}\|_{G^{-1}}^2. \quad (1.9)$$

Now the special properties of $\|\mathbf{e}\|_G$ when applied to equation (1.8) are related to the imposition of a restriction on the independent variables \mathbf{x} of the form

$$\mathbf{x} \equiv \mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_i + \mathbf{P}_i \mathbf{z} \quad (1.10)$$

where \mathbf{x}_i is an arbitrary value of \mathbf{x} , $\mathbf{P}_i \in \mathbb{R}^{n \times r}$ and is also arbitrary and \mathbf{z} denotes a vector of independent variables. The number of columns r of \mathbf{P}_i is always less than or equal to n and is usually very much less, one or two being the most common. Note that if \mathbf{P}_i is a vector ($r = 1$) and z a scalar we obtain the classic parameterization that forms the basis of optimisation linesearch routines.

The special properties referred to above are established by the following theorems.

Theorem 1. Let

- (a) $\mathbf{G} \in \mathbb{R}^{n \times n}$ be symmetric positive definite
- (b) $\mathbf{P}_i \in \mathbb{R}^{n \times r}$ have full column rank

- (c) $\mathbf{x}(\mathbf{z})$ be defined by equation (1.10) and
- (d) $\mathbf{f}(\mathbf{z}) \equiv \mathbf{Gx}(\mathbf{z}) - \mathbf{h}$.

Then (A) there exists a unique value \mathbf{z}_i of \mathbf{z} such that if $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{P}_i \mathbf{z}_i$ and $\mathbf{f}_{i+1} = \mathbf{Gx}_{i+1} - \mathbf{h}$ then

$$\mathbf{P}_i^T \mathbf{f}_{i+1} = \mathbf{0}, \quad (1.11)$$

and (B) \mathbf{x}_{i+1} minimises $\|\mathbf{e}(\mathbf{z})\|_G$ over the manifold defined by equation (1.10).

Proof. From (d) and equation (1.10),

$$\mathbf{f}(\mathbf{z}) \equiv \mathbf{f}_i + \mathbf{GP}_i \mathbf{z} \quad (1.12)$$

and since from the hypotheses of the theorem $\mathbf{P}_i^T \mathbf{GP}_i$ is nonsingular, premultiplication of equation (1.12) by \mathbf{P}_i^T and equating $\mathbf{P}_i^T \mathbf{f}(\mathbf{z})$ to the null vector gives

$$\mathbf{z}_i = -(\mathbf{P}_i^T \mathbf{GP}_i)^{-1} \mathbf{P}_i^T \mathbf{f}_i, \quad (1.13)$$

proving (A).

To prove (B) let $\mathbf{z} \equiv \mathbf{z}_i + \mathbf{u}$ so that from equation (1.10),

$$\mathbf{x}(\mathbf{u}) \equiv \mathbf{x}_i + \mathbf{P}_i \mathbf{z}_i + \mathbf{P}_i \mathbf{u} \equiv \mathbf{x}_{i+1} + \mathbf{P}_i \mathbf{u}$$

and, from (d),

$$\mathbf{f}(\mathbf{u}) \equiv \mathbf{f}_{i+1} + \mathbf{GP}_i \mathbf{u}.$$

From equation (1.9) we have $\|\mathbf{e}(\mathbf{u})\|_G^2 = \mathbf{f}(\mathbf{u})^T \mathbf{G}^{-1} \mathbf{f}(\mathbf{u})$ so that, from equation (1.11),

$$\|\mathbf{e}(\mathbf{u})\|_G^2 = \|\mathbf{e}_{i+1}\|_G^2 + \|\mathbf{P}_i \mathbf{u}\|_G^2 \quad (1.14)$$

and $\|\mathbf{e}(\mathbf{u})\|_G$ is minimised if and only if $\mathbf{u} = \mathbf{0}$. ■

This theorem will be used subsequently to derive the convergence properties of the basic CG algorithm. Note also that the choice of notation implies that the next approximate solution \mathbf{x}_{i+1} is obtained by minimising $\|\mathbf{e}(\mathbf{z})\|_G$ over the manifold defined by equation (1.10).

Theorem 2. Let

- (a) $\mathbf{A} \in \mathbb{R}^{n \times n}$ be nonsingular
- (b) $\mathbf{P}_i \in \mathbb{R}^{n \times r}$ have full column rank
- (c) $\mathbf{x}(\mathbf{z})$ be defined by equation (1.10) and
- (d) $\mathbf{r}(\mathbf{z}) \equiv \mathbf{Ax}(\mathbf{z}) - \mathbf{b}$.

Then (A) there exists a unique value \mathbf{z}_i of \mathbf{z} such that if $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{P}_i \mathbf{z}_i$ and $\mathbf{r}_{i+1} = \mathbf{Ax}_{i+1} - \mathbf{b}$ then

$$\mathbf{P}_i^T \mathbf{A}^T \mathbf{r}_{i+1} = \mathbf{0}, \quad (1.15)$$

and (B) \mathbf{x}_{i+1} minimises $\|\mathbf{r}(\mathbf{z})\|$ over the manifold defined by equation (1.10).

Proof. From (d), $\mathbf{A}^T \mathbf{r}(\mathbf{z}) \equiv \mathbf{A}^T \mathbf{A} \mathbf{x}(\mathbf{z}) - \mathbf{A}^T \mathbf{b}$ and putting $\mathbf{G} = \mathbf{A}^T \mathbf{A}$ and $\mathbf{h} = \mathbf{A}^T \mathbf{b}$ gives, from equations (1.4) and (1.8), $\mathbf{f}(\mathbf{z}) \equiv \mathbf{A}^T \mathbf{r}(\mathbf{z})$. Theorem 1 then asserts that there is a unique value \mathbf{z}_i of \mathbf{z} that minimises $\|\mathbf{e}(\mathbf{z})\|_{\mathbf{A}^T \mathbf{A}} = \|\mathbf{r}(\mathbf{z})\|^2$ over the manifold (1.10). Substituting the appropriate values of \mathbf{G} and \mathbf{f}_i into equation (1.13) gives \mathbf{z}_i to be

$$\mathbf{z}_i = -(\mathbf{P}_i^T \mathbf{A}^T \mathbf{A} \mathbf{P}_i)^{-1} \mathbf{P}_i^T \mathbf{A}^T \mathbf{r}_i, \quad (1.16)$$

completing the proof. ■

Equation (1.11) is called a *Galerkin condition* (more properly a *Ritz-Galerkin condition*) and this condition occurs frequently in the study of Krylov methods. Equation (1.15) is similar. In the case where the matrix of coefficients \mathbf{G} is positive definite, Theorem 1 relates equation (1.11) to the minimisation of the energy norm of the error. Theorem 2 performs a similar function for the Euclidean norm of the residuals. However a glance at the proofs of these theorems indicates that the satisfaction of the Galerkin condition and the proofs of the first part of the theorems do not require that \mathbf{G} be positive definite. The only requirement is that $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is nonsingular. In fact, as we shall see, some very well-known algorithms rely on a Galerkin condition where \mathbf{G} is symmetric but only indefinite and for these algorithms there is no minimisation involved. This has important implications for their stability.

If \mathbf{G} is positive definite and $\mathbf{x}(\mathbf{z})$ is defined by identity (1.10) it makes sense to choose the next approximate solution \mathbf{x}_{i+1} to be that value of \mathbf{x} that minimises $\|\mathbf{e}(\mathbf{z})\|_G$. Thus, from equations (1.10) and (1.13),

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{f}_i \quad (1.17)$$

where

$$\mathbf{D}_i = \mathbf{P}_i^T \mathbf{G} \mathbf{P}_i \quad (1.18)$$

so that, from equation (1.8),

$$\mathbf{f}_{i+1} = (\mathbf{I} - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T) \mathbf{f}_i. \quad (1.19)$$

If we wish to choose \mathbf{x}_{i+1} to minimise $\|\mathbf{r}(\mathbf{z})\|$ it follows from equations (1.10) and (1.16) that

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{P}_i (\mathbf{P}_i^T \mathbf{A}^T \mathbf{A} \mathbf{P}_i)^{-1} \mathbf{P}_i^T \mathbf{A}^T \mathbf{r}_i \quad (1.20)$$

so that, since $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$,

$$\mathbf{r}_{i+1} = (\mathbf{I} - \mathbf{A} \mathbf{P}_i (\mathbf{P}_i^T \mathbf{A}^T \mathbf{A} \mathbf{P}_i)^{-1} \mathbf{P}_i^T \mathbf{A}^T) \mathbf{r}_i. \quad (1.21)$$

Note that the matrices in equations (1.19) and (1.21) are both idempotent while that in the second equation is symmetric as well. Matrices \mathbf{M} that are idempotent (a matrix \mathbf{M} is said to be *idempotent* if $\mathbf{M} = \mathbf{M}^2$) are called *projection matrices*, or *projectors*. If such a matrix is also symmetric then it is an *orthogonal projector*,

otherwise an *oblique* projector. Idempotent matrices have eigenvalues equal to either one or zero since $\mathbf{M}(\mathbf{M} - \mathbf{I}) = \mathbf{O}$ and this implies that for orthogonal (symmetric) projectors \mathbf{M} , $\|\mathbf{M}\| = 1$. Thus, from equation (1.21), $\|\mathbf{r}_{i+1}\| \leq \|\mathbf{r}_i\|$, as must be the case for a norm-minimising algorithm.

The simplest way of exploiting Theorem 1 is by replacing the matrix \mathbf{P}_i in equation (1.17) by some suitably-chosen vector \mathbf{p}_i , giving

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{f}_i}{\mathbf{p}_i^T \mathbf{G} \mathbf{p}_i} \right), \quad (1.22)$$

and we will refer to such vectors as *displacement vectors*. If \mathbf{G} is positive definite and $\mathbf{p}_i^T \mathbf{f}_i \neq 0$ this results in a positive decrease of $\|\mathbf{e}\|_G$. If this operation is carried out sequentially with a number of different displacement vectors $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_i$, we can generate a sequence of approximate solutions $\{\mathbf{x}_i\}$, the energy norms of whose errors decrease monotonically. However monotonic convergence does not guarantee that $\|\mathbf{e}\|_G$ decreases to zero, or that any such decrease will be rapid. The vectors \mathbf{p}_i can be the columns of the unit matrix or the columns of \mathbf{A} or \mathbf{A}^T , generally chosen cyclically. All these variations have been tried and discarded. Another possible choice is $\mathbf{p}_i = -\mathbf{f}_i$ giving the method of *steepest descent* (due originally to Cauchy) which at least guarantees strictly monotonic convergence of $\|\mathbf{e}\|_G$ even if, in common with all the other primitive choices of \mathbf{p}_i , convergence is intolerably slow if \mathbf{G} is at all ill-conditioned. Since the large sparse systems which the algorithms here described are designed to solve are often ill-conditioned, methods based only on equation (1.22) are not practically viable and ways must be found of making them more efficient.

One way is to revert to the original form of equation (1.22) and calculate \mathbf{x}_{i+1} from equation (1.17). For this procedure not to break down it is necessary that \mathbf{D}_i is nonsingular and since \mathbf{G} is positive definite by hypothesis it suffices that \mathbf{P}_i has rank r . For large r one might expect a reasonable reduction of $\|\mathbf{e}\|_G$ at each step even for relatively unsophisticated choices of \mathbf{P}_i but this would be partly offset by the amount of work and storage required to compute \mathbf{D}_i and solve $\mathbf{D}_i \mathbf{z}_i = -\mathbf{P}_i^T \mathbf{f}_i$, since \mathbf{D}_i is $(r \times r)$. Now intuitively it would appear that the greater the value of r the better since the restriction on \mathbf{f}_{i+1} imposed by the Galerkin condition is more severe the greater r becomes. Indeed if $r = n$ so that \mathbf{P}_i is square, if it is also nonsingular (and it must be nonsingular for \mathbf{D}_i to be nonsingular) then equation (1.11) implies that $\mathbf{f}_{i+1} = \mathbf{0}$ so that \mathbf{x}_{i+1} is the required solution.

Thus one way of obtaining improved performance of a norm-reducing algorithm is to compute \mathbf{P}_i in such a way that equation (1.13) may be readily solved even for large values of r . The limitations on r are then those imposed by the combination of a fixed computer storage and the fact that, as r increases, so does the size of \mathbf{P}_i and \mathbf{D}_i . However, pursuing a strategy that equation (1.13) must be readily solvable usually means a destruction of sparsity, and conversely. If \mathbf{G} is sparse then \mathbf{D}_i could be made sparse by choosing \mathbf{P}_i to be, say, a selection of the columns of the unit matrix (the block Gauss-Seidel method) but in this case not only would equation (1.13) be more difficult to solve but we would not be fully exploiting the potential

gains made available to us by increasing r . The other alternative is to compute \mathbf{P}_i so that equation (1.13) admits of a relatively easy solution, with \mathbf{D}_i being perhaps upper Hessenberg or tridiagonal and ignoring completely any other considerations of sparsity.

Algorithms that would, with unlimited storage and exact arithmetic, give the exact solution of $\mathbf{Ax} = \mathbf{b}$ but have to be used iteratively¹ because of physical limitations are discussed in the next chapter. Since they are used iteratively they do not yield the exact solution but the best of them, like GMRes(m), can be very effective. If we want the exact solution with the type of algorithm we are considering we have to appeal to the second of the two principles outlined above, and we consider this second principle in the section on projection methods, below. We first, though, look at another method of obtaining an approximation to the solution of $\mathbf{Ax} = \mathbf{b}$.

1.2. The quasi-minimal residual (QMR) technique

This idea was introduced by Freund and Nachtigal [118] in an attempt to overcome the problems posed by the solution of equation (1.17) and is, as its name implies, a modification of the idea of residual norm reduction. If $\{\mathbf{y}_i\}$ denotes a sequence of *arbitrary* vectors then a second sequence $\{\mathbf{s}_i\}$ may be defined by $\mathbf{s}_1 = \hat{\mathbf{r}}_1 = \mathbf{r}_1$, where \mathbf{r}_1 is the residual corresponding to some initial approximation $\hat{\mathbf{x}}_1 = \mathbf{x}_1$ and, for $i \geq 1$,

$$\mathbf{s}_{i+1} = \mathbf{s}_i \alpha_i + \mathbf{s}_{i-1} \beta_{i-1} + \mathbf{Ay}_i \quad (1.23)$$

where α_i and β_{i-1} are known scalars. This idea is quite flexible since additional vectors \mathbf{s}_k , $k = i - 2$, etc., could be included in equation (1.23) without impairing the original concept. If we now define \mathbf{Y}_i by

$$\mathbf{Y}_i = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_i] \quad (1.24)$$

then we can define a linear manifold $\hat{\mathbf{x}}(\mathbf{v})$, where $\mathbf{v} \in \mathbb{R}^i$ denotes a vector of independent variables, by

$$\hat{\mathbf{x}}(\mathbf{v}) \equiv \hat{\mathbf{x}}_1 + \mathbf{Y}_i \mathbf{v} \quad (1.25)$$

with the corresponding residual $\mathbf{A}\hat{\mathbf{x}}(\mathbf{v}) - \mathbf{b}$ being given by

$$\hat{\mathbf{r}}(\mathbf{v}) \equiv \hat{\mathbf{r}}_1 + \mathbf{AY}_i \mathbf{v} \quad (1.26)$$

where \mathbf{A} , the coefficient matrix of the equations we wish to solve, is not required to be symmetric but only nonsingular. Now ideally we would like to minimise $\|\hat{\mathbf{r}}(\mathbf{v})\|$ over \mathbf{v} but this would involve solving for \mathbf{v} the equation $\mathbf{Y}_i^T \mathbf{A}^T \mathbf{AY}_i \mathbf{v} = -\mathbf{Y}_i^T \mathbf{A}^T \hat{\mathbf{r}}_1$ for probably more than one value of i . Since this is quite impractical it is necessary to find an alternative approach.

¹ We say that an exact algorithm is *used iteratively* if it is modified so as not to give the exact solution but to generate a sequence of (hopefully) converging approximate solutions.

If \mathbf{S}_i is defined analogously to \mathbf{Y}_i , equation (1.23) may be written

$$\mathbf{A}\mathbf{Y}_i = \mathbf{S}_{i+1}\mathbf{H}_{i+1} \quad (1.27)$$

where $\mathbf{H}_{i+1} \in \mathbb{R}^{(i+1) \times i}$ and is given explicitly by

$$\mathbf{H}_{i+1} = \begin{bmatrix} -\alpha_1 & -\beta_1 & 0 & \cdots & 0 & 0 \\ 1 & -\alpha_2 & -\beta_2 & \cdots & 0 & 0 \\ 0 & 1 & -\alpha_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -\alpha_{i-1} & -\beta_{i-1} \\ 0 & 0 & 0 & \cdots & 1 & -\alpha_i \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}. \quad (1.28)$$

(Note that equation (1.27) is quite fundamental and matrices \mathbf{Y}_i and \mathbf{S}_{i+1} satisfying this equation are computed as a matter of course by many of the algorithms discussed below. It is therefore a comparatively simple matter to apply the QMR technique to these algorithms.) Since, by definition, $\hat{\mathbf{r}}_1 = \mathbf{s}_1$, identity (1.26) becomes

$$\hat{\mathbf{r}}(\mathbf{v}) \equiv \mathbf{S}_{i+1}(\mathbf{e}_1 + \mathbf{H}_{i+1}\mathbf{v})$$

and one approach to finding an approximate solution of $\mathbf{Ax} = \mathbf{b}$ would be to minimise $\|\mathbf{e}_1 + \mathbf{H}_{i+1}\mathbf{v}\|$ over \mathbf{v} . The motivation for doing this rather than minimising $\|\hat{\mathbf{r}}(\mathbf{v})\|$ is the fact that since \mathbf{H}_{i+1} is upper Hessenberg this is a comparatively simple operation which may be carried out economically using numerically stable transformations (see Appendix A for full details). This minimisation, however, does not take into account any variations in $\|\mathbf{S}_{i+1}\|$ so Freund and Nachtigal suggested normalising each column of \mathbf{S}_{i+1} before carrying out the minimisation. Let then $\Sigma_{i+1} = \text{diag}(\sigma_j)$ where $\sigma_j = \|\mathbf{s}_j\|$, let $\tilde{\mathbf{H}}_{i+1} = \Sigma_{i+1}\mathbf{H}_{i+1}$ and $\tilde{\mathbf{S}}_{i+1} = \mathbf{S}_{i+1}\Sigma_{i+1}^{-1}$. Then

$$\hat{\mathbf{r}}(\mathbf{v}) \equiv \tilde{\mathbf{S}}_{i+1}(\sigma_1\mathbf{e}_1 + \tilde{\mathbf{H}}_{i+1}\mathbf{v}) \quad (1.29)$$

and QMR works by minimising $\|\sigma_1\mathbf{e}_1 + \tilde{\mathbf{H}}_{i+1}\mathbf{v}\|$ over \mathbf{v} . Now $\tilde{\mathbf{S}}_{i+1}^T \tilde{\mathbf{S}}_{i+1}$ is a symmetric matrix whose diagonal elements are unity and whose off-diagonal elements are less than or equal to unity in absolute value (from the Cauchy inequality). Thus, trivially,

$$\left\| \tilde{\mathbf{S}}_{i+1}^T \tilde{\mathbf{S}}_{i+1} \right\|_1 = \left\| \tilde{\mathbf{S}}_{i+1}^T \tilde{\mathbf{S}}_{i+1} \right\|_\infty \leq i+1$$

so that since, for any matrix \mathbf{M} , $\|\mathbf{M}\|^2 \leq \|\mathbf{M}\|_1 \|\mathbf{M}\|_\infty$ we have

$$\left\| \tilde{\mathbf{S}}_{i+1} \right\| \leq \sqrt{i+1}.$$

Thus even if the QMR choice of \mathbf{v} does not minimise $\|\hat{\mathbf{r}}(\mathbf{v})\|$ it does at least guarantee a reasonable upper bound if a sufficiently small value of $\|\sigma_1\mathbf{e}_1 + \tilde{\mathbf{H}}_{i+1}\mathbf{v}\|$ can be achieved.

If, for some value of i , $\mathbf{s}_{i+1} = \mathbf{0}$, equation (1.29) becomes

$$\tilde{\mathbf{r}}(\mathbf{v}) \equiv \tilde{\mathbf{S}}_i(\sigma_1 \mathbf{e}_1 + \bar{\mathbf{H}}_i \mathbf{v}) \quad (1.30)$$

where $\bar{\mathbf{H}}_i$ denotes $\tilde{\mathbf{H}}_{i+1}$ with its final row omitted. If this matrix is nonsingular, minimising $\|\sigma_1 \mathbf{e}_1 + \bar{\mathbf{H}}_i \mathbf{v}\|$ over \mathbf{v} gives a null value for the residual so that the corresponding value of $\hat{\mathbf{x}}$ is the solution of $\mathbf{Ax} = \mathbf{b}$. If not, it is still possible to obtain a minimum-norm solution of equation (1.30) and a consequent approximate solution of $\mathbf{Ax} = \mathbf{b}$.

The actual minimisation of $\|\sigma_1 \mathbf{e}_1 + \tilde{\mathbf{H}}_{i+1} \mathbf{v}\|$ and the subsequent derivation of the recurrence formula for $\{\hat{\mathbf{x}}_i\}$ are somewhat technical and are normally based on the reduction of $\tilde{\mathbf{H}}_{i+1}$ to upper triangular form by orthogonal transformations. These transformations are described in Appendix A while the derivation of the recurrence formula for $\hat{\mathbf{x}}_i$ and further properties are given in Chapter 6. Descriptions of algorithms using the QMR technique are to be found in Chapters 3 and 5.

1.3. Projection methods

None of the classical choices of \mathbf{P}_i guarantees an exact solution of equation (1.1) in a finite number of steps and this problem must now be addressed. Although it is possible to base our arguments on minimising a function such as $\mathbf{e}^T \mathbf{G} \mathbf{e}$ the alternative approach is simpler and more susceptible to generalisation, and for this reason is preferred. It leads to the same final result.

We first note that although equation (1.11) was derived using arguments based on minimising an error norm it does, in fact, constitute a restrictive condition in its own right. It imposes the restriction that \mathbf{f}_{i+1} must lie in a subspace, the particular one in question being the null-space of \mathbf{P}_i^T whose dimension is $(n - r)$. This interpretation is valid even if \mathbf{G} is indefinite or even nonsymmetric although in these cases there will generally be no element of minimisation involved in the calculation of \mathbf{x}_{i+1} and hence no guarantee that any algorithm based on equation (1.11) alone will converge.

Now we have already noted that it is not possible in practice to put $r = n$ since in this case solving the system $\mathbf{D}_i \mathbf{z}_i = -\mathbf{P}_i^T \mathbf{f}_i$ would be just as difficult as solving $\mathbf{Ax} = \mathbf{b}$, but it does suggest a possible way forward. We know that $\mathbf{P}_i^T \mathbf{f}_{i+1} = \mathbf{0}$ (equation (1.11)) but unless specific conditions are imposed upon the matrices \mathbf{P}_j the vectors $\mathbf{P}_j^T \mathbf{f}_{i+1}$, $1 \leq j \leq i - 1$, will not be anything in particular and will certainly not be null. However if we could, simply, compute matrices \mathbf{P}_j such that $\mathbf{P}_j^T \mathbf{f}_{i+1} = \mathbf{0}$ for $1 \leq j \leq i$ and the columns of the collection of matrices \mathbf{P}_j were all linearly independent then \mathbf{f}_{i+1} would, as i increased, be squeezed into a sequence of subspaces of ever-decreasing dimension and would eventually be forced to become null. For this to happen cleanly it would be necessary for n to be an integral multiple of r (or that each \mathbf{P}_j satisfies $\mathbf{P}_j \in \mathbb{R}^{n \times r_j}$ and $\sum_j r_j = n$) but this is not an insurmountable problem.

In order to simplify the algebra we define the matrix $\bar{\mathbf{P}}$, by

$$\bar{\mathbf{P}}_i = [\mathbf{P}_1 \ \mathbf{P}_2 \ \dots \ \mathbf{P}_i] \quad (1.31)$$

so that the conditions that we wish to impose may be expressed as

$$\bar{\mathbf{P}}_i^T \mathbf{f}_{i+1} = \mathbf{0}. \quad (1.32)$$

The problem now facing us is how to construct an algorithm that satisfies this equation, but before we do this we should perhaps look at some of its implications.

One advantage of this dimension-reduction argument is that it may be applied without modification if \mathbf{G} is indefinite or even nonsymmetric. The most obvious reason for choosing an indefinite \mathbf{G} would be if \mathbf{A} were symmetric but indefinite, when we could put $\mathbf{G} = \mathbf{A}$ and effectively find the stationary value of $\mathbf{e}^T \mathbf{G} \mathbf{e}$. Another would be if \mathbf{A} were nonsymmetric when we could put

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}$$

and solve the equations

$$\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix}, \quad (1.33)$$

thus solving $\mathbf{A}^T \mathbf{z} = \mathbf{c}$ as well as $\mathbf{Ax} = \mathbf{b}$ (this system is referred to as the *expanded system* by Brezinski [35]). It might be thought somewhat extravagant electing to solve two systems of equations when the solution of only one is needed, but if the advantages of having a symmetric \mathbf{G} outweigh the disadvantages of doubling the size of the problem then a case for this choice of \mathbf{G} can be made, and many modern algorithms do just this.

Since the dimension-reducing argument is valid for \mathbf{G} nonsymmetric there seems to be no reason why it should not be applied directly to the residuals calculated from the original matrix \mathbf{A} , here assumed nonsymmetric, so that the Galerkin condition becomes

$$\mathbf{P}_j^T \mathbf{r}_{i+1} = \mathbf{0}, \quad j = 1, 2, \dots, i$$

where $\mathbf{r}_i = \mathbf{Ax}_i - \mathbf{b}$. By analogy with equation (1.17), \mathbf{x}_{i+1} is then computed by

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{P}_i (\mathbf{P}_i^T \mathbf{A} \mathbf{P}_i)^{-1} \mathbf{P}_i^T \mathbf{r}_i \quad (1.34)$$

which guarantees that the equation $\mathbf{P}_i^T \mathbf{r}_{i+1} = \mathbf{0}$ is trivially satisfied. The difficulty with trying to extend this approach so that $\mathbf{P}_j^T \mathbf{r}_{i+1} = \mathbf{0}$, $1 \leq j \leq i$, is that it leads to algorithms requiring excessive storage, although at least one such algorithm has been proposed (FOM, see Chapter 2). A similar difficulty occurs if we let $\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_i + \mathbf{P}_i \mathbf{z}$ and impose the condition that $\mathbf{V}_i^T \mathbf{r}_{i+1} = \mathbf{0}$ for some $\mathbf{V}_i \neq \mathbf{P}_i$ in order to compute \mathbf{x}_{i+1} . Again there is no reason in principle why such algorithms should not be constructed, but they are generally too memory-intensive to be practical. For this reason we choose \mathbf{G} to be symmetric, although even this by itself is not enough. Algorithms have been proposed where \mathbf{G} is symmetric and the storage requirements are still too large to permit the exact solution of really large systems. For these problems, this type of algorithm has to be used iteratively.

1.4. Matrix equations

Another advantage of the Galerkin condition is that it is immediately applicable to systems of equations with more than one right-hand side. Suppose we want to solve $\mathbf{Ax} = \mathbf{b}_j$ for $j = 1, 2, \dots, s$, where every \mathbf{b}_j is known at the outset. We can express the equations as $\mathbf{AX} = \mathbf{B}$, where \mathbf{b}_j is the j -th column of \mathbf{B} and the corresponding solution is the j -th column of \mathbf{X} . The symmetric equivalent of this equation is $\mathbf{GX} = \mathbf{H}$, where \mathbf{G} is defined as before, and if \mathbf{X}_i denotes the i -th approximation to the solution then \mathbf{F}_i may be defined as

$$\mathbf{F}_i = \mathbf{GX}_i - \mathbf{H}. \quad (1.35)$$

If \mathbf{X}_{i+1} is then computed by

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i \quad (1.36)$$

which is an obvious generalisation of equation (1.17), it follows from equation (1.35) that

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{GP}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i \quad (1.37)$$

so that, trivially, the Galerkin condition $\mathbf{P}_i^T \mathbf{F}_{i+1} = \mathbf{0}$ is satisfied.

Perhaps if solving equation (1.1) with more than one right-hand side were the only reason for introducing the block form (so-called because \mathbf{F}_i and \mathbf{X}_i are blocks of vectors and not just single ones) we would defer it to some later chapter. However some of the most popular and best-known methods for solving the *vector* equation (1.1) when \mathbf{A} is nonsymmetric are based on solving

$$\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix} \quad (1.38)$$

(the block form of equation (1.33)). These methods are so important practically, and so awkward to deal with individually, that we prefer to express the basic theory in terms of blocks. This causes no great difficulty since the theory is virtually the same in either case, the only difference being that, in the block case, the matrices \mathbf{F}_i can sometimes become rank-deficient without becoming null necessitating certain rescue procedures. If \mathbf{f}_i becomes null then \mathbf{x}_i is the required solution. If \mathbf{F}_i becomes rank-deficient then essentially we have obtained a partial solution of the equation $\mathbf{GX} = \mathbf{H}$ in the sense that we have found a matrix \mathbf{X}_i such that $\mathbf{GX}_i \mathbf{v} = \mathbf{H} \mathbf{v}$ for some vector $\mathbf{v} \neq \mathbf{0}$. Apart from this, though, the theory for both cases is practically identical.

In particular the theory is identical in the applicability of the Galerkin condition and the idea of dimension-reduction. If

$$\bar{\mathbf{P}}_i^T \mathbf{F}_{i+1} = \mathbf{0}, \quad (1.39)$$

then precisely the same argument to that used previously shows that if the columns of $\bar{\mathbf{P}}_i$ are linearly independent then the columns of \mathbf{F}_{i+1} must lie in a subspace

whose dimension reduces as i increases. If therefore we can construct a sequence of matrices $\{\mathbf{P}_j\}$ such that equation (1.39) is satisfied where $\overline{\mathbf{P}}_i$ is defined by equation (1.31) we have a powerful means of obtaining an exact solution of $\mathbf{G}\mathbf{X} = \mathbf{H}$ after a finite number of steps. We describe in the next section some of the key ideas underpinning this type of method.

1.5. Some basic theory

Our next theorem gives sufficient conditions for the matrices \mathbf{P}_j to satisfy in order to generate the sequence of matrices satisfying equation (1.39).

Theorem 3. Let $\mathbf{G} \in \mathbb{R}^{n \times n}$ be symmetric and nonsingular, and let $\mathbf{H}, \mathbf{X} \in \mathbb{R}^{n \times r}$. Let \mathbf{X}_i be an approximate solution of $\mathbf{G}\mathbf{X} = \mathbf{H}$, $\mathbf{F}_i = \mathbf{G}\mathbf{X}_i - \mathbf{H}$ and let $\mathbf{P}_j^T \mathbf{F}_i = \mathbf{O}$, $1 \leq j \leq i-1$. Then if \mathbf{X}_{i+1} is given by equation (1.36), a sufficient condition for $\mathbf{P}_j^T \mathbf{F}_{i+1} = \mathbf{O}$, $1 \leq j \leq i$, is that $\mathbf{P}_j^T \mathbf{G}\mathbf{P}_i = \mathbf{O}$, $1 \leq j \leq i-1$.

Proof. From equation (1.37) we have, trivially, $\mathbf{P}_i^T \mathbf{F}_{i+1} = \mathbf{O}$. For $1 \leq j \leq i-1$, and the same equation, $\mathbf{P}_j^T \mathbf{F}_{i+1} = \mathbf{O}$ follows equally trivially from the hypotheses of the Theorem. ■

Matrices that satisfy the condition

$$\mathbf{P}_j^T \mathbf{G}\mathbf{P}_k = \mathbf{O}, \quad j \neq k \quad (1.40)$$

are said to be *conjugate with respect to \mathbf{G}* and form the basis, normally in their single-column (i.e. vector) versions, of the groups of algorithms known as the *conjugate gradient (CG)* or *conjugate direction (CD)* methods. If $r > 1$ these methods are referred to as *block methods*.

Lemma 4. Let $\mathbf{P}_j^T \mathbf{G}\mathbf{P}_k = \mathbf{O}$ for $j \neq k$ and let the matrices $\mathbf{D}_j = \mathbf{P}_j^T \mathbf{G}\mathbf{P}_j$ be nonsingular for $1 \leq j \leq i$. Let $\overline{\mathbf{P}}_i$ be defined by equation (1.31). Then $\overline{\mathbf{P}}_i$ has full rank.

Proof. Define $\overline{\mathbf{D}}_i$ by

$$\overline{\mathbf{D}}_i = \overline{\mathbf{P}}_i^T \mathbf{G} \overline{\mathbf{P}}_i. \quad (1.41)$$

It will, from the conditions of the Lemma and equation (1.31), be block-diagonal with its diagonal blocks equal to \mathbf{D}_j , $j = 1, 2, \dots, i$. Thus $\overline{\mathbf{D}}_i$ is nonsingular since, by hypothesis, every matrix \mathbf{D}_j is nonsingular and since $\overline{\mathbf{D}}_i$ is nonsingular then, from equation (1.41), $\overline{\mathbf{P}}_i$ must have full rank. ■

Thus if we can construct a sequence of matrices $\{\mathbf{P}_j\}$ that satisfy equation (1.40) and for which $\mathbf{P}_j^T \mathbf{G}\mathbf{P}_j$ is nonsingular for all j then not only can we compute the sequence $\{\mathbf{X}_i\}$ successively using equation (1.36) knowing that equation (1.39) will be automatically satisfied, but since $\overline{\mathbf{P}}_i$ has full rank we can also be certain that the dimension of the space available to \mathbf{F}_{i+1} is being systematically reduced and that we therefore obtain the exact solution of equation (1.1) after a finite number

of steps. This simple idea is the key to the finite termination of the Krylov sequence methods.

The mutual conjugacy of the matrices $\{\mathbf{P}_j\}$ enables us to derive alternative expressions for \mathbf{F}_{i+1} . Equation (1.37) may be written

$$\mathbf{F}_{i+1} = (\mathbf{I} - \mathbf{G}\mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{P}_i^T)\mathbf{F}_i \quad (1.42)$$

so that

$$\mathbf{F}_{i+1} = \prod_{j=1}^i (\mathbf{I} - \mathbf{G}\mathbf{P}_j\mathbf{D}_j^{-1}\mathbf{P}_j^T)\mathbf{F}_1$$

or

$$\mathbf{F}_{i+1} = \mathbf{Q}_i\mathbf{F}_1 \quad (1.43)$$

where $\mathbf{Q}_0 = \mathbf{I}$ and, for $i \geq 1$,

$$\mathbf{Q}_i = \prod_{j=1}^i (\mathbf{I} - \mathbf{G}\mathbf{P}_j\mathbf{D}_j^{-1}\mathbf{P}_j^T). \quad (1.44)$$

If this product is multiplied out, all the terms involving $\mathbf{P}_j^T\mathbf{G}\mathbf{P}_k$, $j \neq k$, vanish and we are left with

$$\mathbf{Q}_i = \mathbf{I} - \mathbf{G} \sum_{j=1}^i \mathbf{P}_j\mathbf{D}_j^{-1}\mathbf{P}_j^T. \quad (1.45)$$

This corresponds to the normal (sum) form of the Gram-Schmidt algorithm as opposed to the modified (product) form implied by equation (1.44). Since the matrix $\overline{\mathbf{D}}_i$ as defined by equation (1.41) is block diagonal it follows from equations (1.31) and (1.45) that

$$\mathbf{Q}_i = \mathbf{I} - \mathbf{G}\overline{\mathbf{P}}_i\overline{\mathbf{D}}_i^{-1}\overline{\mathbf{P}}_i^T. \quad (1.46)$$

This expression for \mathbf{Q}_i is the one we will use most in what follows, and its first rôle is in proving a theorem that underpins the convergence analysis of CG in Chapter 4 (below).

Theorem 5. Let

- (a) $\mathbf{G} \in \mathbb{R}^{n \times n}$ be symmetric positive definite and define $\mathbf{f}(\mathbf{x}) \equiv \mathbf{G}\mathbf{x} - \mathbf{h}$
- (b) the matrices $\mathbf{P}_j \in \mathbb{R}^{n \times r}$, $j = 1, 2, \dots, i$, have full rank and satisfy $\mathbf{P}_j^T\mathbf{G}\mathbf{P}_k = \mathbf{O}$, $j \neq k$
- (c) the error $\mathbf{e}(\mathbf{x})$ be defined by $\mathbf{e}(\mathbf{x}) \equiv \mathbf{x} - \mathbf{G}^{-1}\mathbf{h}$
- (d) $\overline{\mathbf{P}}_i$ and $\overline{\mathbf{D}}_i$ be defined by equations (1.31) and (1.41) and
- (e) $\mathbf{x}_1 \in \mathbb{R}^n$ be an arbitrary initial value of \mathbf{x} .

Then if the sequence of approximate solutions \mathbf{x}_{j+1} , $j = 1, 2, \dots, i$, is computed by equation (1.17), \mathbf{x}_{i+1} minimises $\|\mathbf{e}(\mathbf{x})\|_G$ over the manifold

$$\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \overline{\mathbf{P}}_i\mathbf{z}.$$

Proof. Since \mathbf{x}_{j+1} is computed from \mathbf{x}_j by equation (1.17) the associated residuals satisfy equation (1.19), the vector form of equation (1.42). As this is true for $j = 1, 2, \dots, i$, we obtain, from equations (1.44) and (1.43), $\mathbf{f}_{i+1} = \mathbf{Q}_i \mathbf{f}_1$. It now follows from equation (1.44) and hypothesis (b) of the theorem that \mathbf{Q}_i is given by equation (1.46) so that

$$\mathbf{f}_{i+1} = (\mathbf{I} - \mathbf{G}\bar{\mathbf{P}}_i\bar{\mathbf{D}}_i^{-1}\bar{\mathbf{P}}_i^T)\mathbf{f}_1. \quad (1.47)$$

Hence $\bar{\mathbf{P}}_i^T \mathbf{f}_{i+1} = \mathbf{0}$ and since $\mathbf{f} = \mathbf{Gx} - \mathbf{h}$ and \mathbf{G} is nonsingular, equation (1.47) also implies that

$$\mathbf{x}_{i+1} = \mathbf{x}_1 + \bar{\mathbf{P}}_i \mathbf{z}_i$$

where $\mathbf{z}_i = -\bar{\mathbf{D}}_i^{-1}\bar{\mathbf{P}}_i^T \mathbf{f}_1$. That this value of \mathbf{z} minimises $\|\mathbf{e}(\mathbf{z})\|_G$ over the manifold $\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \bar{\mathbf{P}}_i \mathbf{z}$ follows immediately from Theorem 1 with \mathbf{x}_1 replacing \mathbf{x}_i and $\bar{\mathbf{P}}_i$ replacing \mathbf{P}_i , proving the theorem. ■

This theorem, together with Theorem 1, forms the basis of the classical CG convergence theory where \mathbf{G} was always assumed to be positive definite. Theorem 1 shows that if \mathbf{x}_{i+1} is computed using equation (1.17) then $\|\mathbf{e}(\mathbf{x})\|_G$ is minimised over the manifold defined by equation (1.10) while Theorem 5 shows that if $\{\mathbf{x}_j\}$ is computed by a succession of such steps where the matrices \mathbf{P}_j are mutually conjugate then $\|\mathbf{e}(\mathbf{x})\|_G$ is minimised over the totality of such manifolds. Thus it is possible to achieve the same reduction of $\|\mathbf{e}(\mathbf{x})\|_G$ obtained by one “big step” (over $\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \bar{\mathbf{P}}_i \mathbf{z}$) by a succession of “little steps” (over $\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_j + \mathbf{P}_j \mathbf{z}$ for $j = 1, 2, \dots, i$) if the matrices \mathbf{P}_j are mutually conjugate. Using these ideas we can not only prove finite termination for CG but can give upper bounds for $\|\mathbf{e}(\mathbf{x})\|_G$ at each stage of the iteration (see Chapter 4 below). All this, though, goes by the board if \mathbf{G} is indefinite since then the norm $\|\mathbf{e}(\mathbf{x})\|_G$ is undefined. In this case, provided that the matrices \mathbf{P}_j are mutually conjugate and no breakdown occurs, the same algorithm will still solve $\mathbf{Gx} = \mathbf{h}$ in a finite number of steps, but intermediate errors cannot now be bounded without further assumption and the termination proof must be based on equation (1.39). A solution is usually possible but the advantages of stability and monotonicity conferred by positive definiteness have been lost.

We note that although we have assumed, for simplicity, that all the matrices \mathbf{P}_j have r columns the preceding arguments are equally valid when each \mathbf{P}_j has an arbitrary number of columns. This is of practical importance when certain types of failure occur (see Chapter 8).

Properties of the matrices \mathbf{Q}_i that follow immediately from the above definitions and which will be exploited in the subsequent development of the theory are

$$\bar{\mathbf{P}}_i^T \mathbf{G} \mathbf{Q}_i^T = \mathbf{O} \quad (1.48)$$

$$\mathbf{Q}_i^T \bar{\mathbf{P}}_i = \mathbf{O} \quad (1.49)$$

$$\mathbf{Q}_i \mathbf{Q}_j = \mathbf{Q}_j \mathbf{Q}_i = \mathbf{Q}_i, \quad j \leq i \quad (1.50)$$

and

$$\mathbf{Q}_{j-1} \mathbf{G} \mathbf{P}_i = \mathbf{G} \mathbf{P}_i, \quad j \leq i. \quad (1.51)$$

We now show how we can, in principle, compute a sequence of conjugate matrices. We begin by assuming that $\bar{\mathbf{P}}_i$ and $\bar{\mathbf{D}}_i$ are as previously defined and have already been computed, and that $\bar{\mathbf{D}}_i$ is nonsingular. To construct a matrix \mathbf{P}_{i+1} that satisfies $\bar{\mathbf{P}}_i^T \mathbf{G} \mathbf{P}_{i+1} = \mathbf{O}$ it suffices that

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{W}_{i+1} \quad (1.52)$$

where $\mathbf{W}_{i+1} \in \mathbb{R}^{n \times r}$ is some arbitrary matrix, since for any arbitrary matrix \mathbf{W}_{i+1} we have, from equation (1.48), $\bar{\mathbf{P}}_i^T \mathbf{G} \mathbf{Q}_i^T \mathbf{W}_{i+1} = \mathbf{O}$. Given such a sequence of matrices $\{\mathbf{W}_j\}$ we can, if certain not particularly restrictive conditions are satisfied, use equations (1.46) and (1.52) successively to generate the required sequence of conjugate matrices $\{\mathbf{P}_j\}$.

In practical calculation, of course, the matrices \mathbf{Q}_i are not stored as such but as $\bar{\mathbf{P}}_i$ and perhaps $\bar{\mathbf{D}}_i$. For some algorithms even this is unnecessary as only a degenerate form of \mathbf{Q}_i is needed with just a few terms of the sum in equation (1.45) being required. We obtain this reduction in storage by clever choices of the matrices \mathbf{W}_i . The two principal choices are $\mathbf{W}_i = \mathbf{K} \mathbf{G} \mathbf{P}_{i-1}$ which gives

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{K} \mathbf{G} \mathbf{P}_i \quad (1.53)$$

and $\mathbf{W}_i = \mathbf{K} \mathbf{F}_i$ from which

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{K} \mathbf{F}_{i+1} \quad (1.54)$$

where \mathbf{K} is some arbitrary matrix. Their use leads to two related families of methods which are found to be based on the sequence of matrices $(\mathbf{K} \mathbf{G})^j \mathbf{P}_1$, $j = 0, 1, \dots, i-1$, the sequence being named after the Russian mathematician A. N. Krylov (see Chapter 4, below). We shall refer to the first of these two choices as the *Lanczos* choice as it leads to the characteristic three-term recurrence formulæ devised by C. Lanczos [175]. The second choice will be referred to as the *Hestenes-Stiefel (HS)* choice as from it may be obtained the two-term formulæ associated with M. R. Hestenes and E. Stiefel [154]. These two choices dominate the subject and much of what follows concerns the contrasting properties of the algorithms that they yield.

Not all Krylov sequence algorithms, however, are based on degenerate forms of \mathbf{Q}_i and in the next chapter we explore some algorithms which do not have this property. In order that they may be viable it is necessary to truncate or restart them when they threaten to take too much time per step or too much computer memory. Despite this they can be very effective and some of them are among the most popular and widely-used of all the Krylov methods.

Text	Normal	Name in full
BiCG	Bi-CG	Bi-conjugate gradients
BiCR	Min. Res.	Bi-conjugate residuals
BiCRL	Min. Res.	BiCR, Lanczos version
BiCGStab	Bi-CGSTAB	Bi-CG stabilised
BICG	BICG	Block CG
CG	CG	Conjugate gradients
CGNE	CGNE	CG normal errors
CGNR	CGNR	CG normal residuals
CGS	CGS	Conjugate gradient squared
CGW	CGW	Concus-Golub-Widlund
CR	CR	Conjugate residuals
FOM	FOM	Full orthogonalization method
GCR	GCR	Generalised CR
GMRes	GMRES	Generalised minimal residuals
GODir	GODIR	Generalised OrthoDir
GOMin	GOMIN	Generalised OrthoMin
GORes	GORES	Generalised OrthoRes
HG	Galerkin	Hegedüs Galerkin
HGL	Galerkin	HG Lanczos version
LSQR	LSQR	Least-squares QR
MCR	MCR	Modified CR
MRZ	MRZ	Method of recursive zoom
OD	OD	Orthogonal directions
OrthoDir	Orthodir	Orthogonal directions
OrthoMin	ORTHOMIN	Orthogonal minimisation
OrthoRes	ORTHORES	Orthogonal residuals
QMR	QMR	Quasi-minimal residuals
QMRCGStab	QMRCGSTAB	BiCGStab, QMR version
SymmLQ	SYMMLQ	Symmetric LQ
StOD	STOD	Stabilised OD
TFQMR	TFQMR	Transpose-free QMR

Table 1.

1.6. The naming of algorithms

If the naming of cats merits a poem by T. S. Eliot [102] then the naming of algorithms surely merits a brief discussion in a book devoted to their properties and performance, the more so since many algorithmic names are confusing or even downright misleading. The CGS (conjugate gradient squared) algorithm, for example, as was pointed out in [242], is based not on the square of the conjugate gradient

algorithm but on the square of the *biconjugate* gradient algorithm (BiCG), while the phrase for which MRZ is an acronym seems to have been tailored to fit the acronym rather than the algorithm. The term QMR is applied indiscriminately not only to both versions of the algorithm itself but also to the residual-smoothing technique which is a vital component of the algorithm and which can be, and indeed already has been, tacked on to various other algorithms in attempts to improve their performances. It thus emphasises, perhaps unfairly, just one of the three major innovations described in [118]. The use of upper- and lower-case letters seems to be entirely arbitrary and with some algorithmic abbreviations approaching a dozen or so characters the jargon is becoming increasingly confusing for the non-specialist and neophyte.

While the rôle of this book is not that of a linguistic crusader we have, nevertheless, modified some of even the better-known names in the interests of clarity. One principle we have adopted is that upper-case letters are used only if they are the initial letters of words appearing in the written name. Thus GMRES (generalised minimal residual) becomes GMRes, comprehensible (if perhaps irritating) to the specialist but more helpful to the uninitiated. We sometimes denote the Lanczos form of an algorithm by the letter L and occasionally add a preferred acronym to the usual one if it describes the algorithm more precisely. Thus we may refer to MRZ as MRZ (BiCGL) since MRZ is just the “look-ahead” version of the Lanczos form of the biconjugate gradient algorithm. We have also, very rarely, taken the liberty of using a name not chosen by the originator of the algorithm if this name has appeared before in the literature and more precisely describes its subject. We have done this for the algorithm referred to by Hegedüs as his “minimum residual” algorithm, partly to avoid confusion with all the other minimum residual algorithms with which the field is littered and partly because we think the term *biconjugate residual* better describes the algorithm. In order to clarify matters even further we have appended a table (Table 1, above) which gives the conventional (normal) abbreviation, the abbreviation used in the text and the full name.

1.7. Historical notes

Although the names of Hestenes and Stiefel will always be associated with the conjugate gradient methods, this is because they were the first to show [154] that short recurrences are possible when computing the sequence of displacement vectors $\{\mathbf{p}_i\}$. Prior to 1952, when their seminal paper first appeared, others had also remarked the possibility of using conjugacy when solving systems of linear equations. Stewart [224] states that “The prototype for the class of conjugate direction algorithms was described by Fox, Huskey and Wilkinson [112]” and this prototype was referred to by its proposers as “the method of orthogonal (*sic*) vectors”. Although in 1948 they regarded this method as a direct one their work nevertheless paved the way for later discoveries which developed the more iterative aspects of the algorithm. They computed the displacement vectors \mathbf{p}_i essentially from the vector form of equation (1.52) where the vectors \mathbf{w}_i were taken to be the successive

columns of the unit matrix, and did not assume \mathbf{G} to be positive definite. Neither did they claim to be the originators of the method. Although they arrived at the algorithm independently, the numerical tests that they carried out led them to re-examine the “escalator method” of Morris [187] (not to be confused with an earlier iterative method of the same author [186]). This, they concluded, was essentially the same method as their own despite considerable differences in presentation. The idea therefore of using conjugate directions to solve symmetric systems goes back at least as far as 1946 when [187] was published, or 1942 if we take into account the earlier work of Morris and Head [188], [189] on the solution of the Lagrangian frequency equations.

In the same year that the Hestenes-Stiefel paper appeared Lanczos [175] published a paper describing the solution of linear equations by “the method of minimized iterations”, and this method turned out to be a 3-term equivalent of the 2-term method of Hestenes and Stiefel (see Chapter 4 below for a discussion of the relationship between these two algorithms). Originally these two methods were regarded as quite distinct since the connection between them is by no means obvious, and it was not until twelve years later that Householder [156] showed otherwise. At first Lanczos’s new method was largely ignored, partly because, like the Hestenes and Stiefel algorithm, it was regarded as a direct method and thus a direct rival of Gaussian elimination and Cholesky factorisation. Looked at in this way it could barely compete, and it was not until the advent of larger computers and the need to solve larger, sparse, systems (where the problems of *fill-in* (the replacement of zeroes by nonzeroes and consequent increase of storage requirements) imposed severe handicaps on the classical direct methods) that the method, together with its HS counterpart, assumed greater prominence. It was helped in this by the 1976 paper of Fletcher [110] which made the method more accessible to a wider audience and by the 1971 contribution of Reid [208] who first noted the iterative character of the conjugate gradient methods.

In 1973 Stewart [224] introduced the idea of block conjugacy and pointed out that many of the operations carried out when using a CG algorithm are in fact projections. These two techniques have been exploited extensively in the present volume. Wherever appropriate, an equation or formula is always written as a projection even though the resulting expression differs from the conventional representation found elsewhere.

More recently many attempts have been made to simplify and unify the theory of CG methods, varying from the papers of Dennis and Turner [89], Ashby *et al* [7], Gutknecht [140], [141] and Broyden [47] to several full-blown textbooks. The latter employ various approaches. The monograph of Fischer [107] (which deals only with equations involving symmetric matrices) and the comprehensive studies of Brezinski *et al* [35], [40], are based on the properties of polynomials. This is possible because the Krylov methods, as we shall see in Chapter 4, involve subspaces defined by sequences of matrix polynomials multiplying a given matrix or vector. The books by Weiss [256] and Greenbaum [134], on the other hand, veer more towards linear algebra. Perhaps the most ambitious attempt at the unification of the theory of linear-equation solvers is the 1989 attempt of Abaffy and Spedicato

[1]. The authors consider an extremely general class of projection methods (the so-called ABS methods), some of which were also discussed in 1960 by Egervary [99]. They include the Krylov methods as special cases. Their approach is based on linear algebra but they are less interested in the properties of the Krylov methods as such than in the broader overall picture. Other books of a more general nature include those by Meurant [185] and Saad [217].

The present volume is firmly in the linear algebra camp, invoking polynomials only when it is absolutely necessary to do so. Although superficially the theoretical development resembles that of Weiss there are two significant differences. Both approaches are based on 2×2 block matrices but in [256] the fundamental equations to be solved are

$$\begin{bmatrix} \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{A}^T \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}$$

whereas in the present volume they are taken to be

$$\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix}.$$

Thus our coefficient matrices are symmetric and the analysis is based on the block conjugate gradient algorithm. It stems from a series of papers [46], [47], [49] and [51] published between 1993 and 1999 by one of the present authors.

One aspect of our subject that we wish to emphasize in the present volume is the relationship between the modern Krylov methods and their antecedents. We therefore, in the next chapter, begin by discussing the Gram-Schmidt and Arnoldi algorithms and show in Chapter 3 how the modern algorithms are related to these two precursors. This relationship is valid despite the fact that some of the intermediate methods (GCR and OrthoDir) were only discovered *after* the announcement of the basic CG methods, and as such is a tribute to the insight showed by Hestenes, Stiefel and Lanczos in making their original contributions.

Chapter 2

The long recurrences

We refer to the methods described in this chapter as *the long recurrences* since in order for them to compute a set of vectors \mathbf{p}_i or matrices \mathbf{P}_i whose conjugacy properties are such as to guarantee termination it is necessary to use the full form of the projection matrix \mathbf{Q}_i . The matrix version of this is (equation (1.45))

$$\mathbf{Q}_i = \mathbf{I} - \mathbf{G} \sum_{j=1}^i \mathbf{P}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \quad (2.1)$$

so that as the iteration proceeds and i increases, so do the memory requirements for storing \mathbf{Q}_i and the time taken to carry out each individual iteration. This is in marked contrast with the short recurrences discussed in the next chapter. For these it will be shown that if \mathbf{G} and \mathbf{K} are both symmetric and if the matrices \mathbf{W}_{i+1} of equation (1.52) are chosen appropriately then the above expression for \mathbf{Q}_i may be replaced by

$$\mathbf{Q}_i = \mathbf{I} - \mathbf{G} \mathbf{P}_{i-1} \mathbf{D}_{i-1}^{-1} \mathbf{P}_{i-1}^T - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \quad (2.2)$$

or even by $\mathbf{Q}_i = \mathbf{I} - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T$. Despite these simplifications the methods still generate complete sets of conjugate vectors (or matrices), at least if exact arithmetic is assumed. Thus for the short recurrences both the storage requirements and the time taken for each iteration are significantly reduced and are, moreover, constant. There is, though, a price to be paid for these gains. If we use the full form of \mathbf{Q}_i as defined by equation (2.1) and compute \mathbf{P}_{i+1} by $\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{W}_{i+1}$ then \mathbf{P}_{i+1} is conjugate to all the other matrices \mathbf{P}_j , $1 \leq j \leq i$, regardless of the choice of \mathbf{W}_{i+1} . If, on the other hand, we use the form of equation (2.2) then, for an arbitrary \mathbf{W}_{i+1} , \mathbf{P}_{i+1} is only necessarily conjugate to \mathbf{P}_i and \mathbf{P}_{i-1} and this makes the short recurrences particularly susceptible to the propagation of rounding error. In practice it is found that provided that $|i - j|$ is reasonably small, say less than or equal to ten, then $\|\mathbf{P}_i^T \mathbf{G} \mathbf{P}_j\|$ is not too large but beyond this the conjugacy deteriorates. We return to this matter in Chapter 9.

Despite the problems of space utilisation and execution time noted above, various methods based on the long recurrences have been proposed and we begin this chapter with descriptions of two methods which form the basis of many such algorithms. These methods do not themselves solve the equations $\mathbf{GX} = \mathbf{H}$ but compute the conjugate matrices \mathbf{P}_i by which, using equations

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i$$

and

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i,$$

they may be solved. They are generalised versions of the Gram-Schmidt method and Arnoldi's method (see e.g. [134]). We give the generalised versions partly because they lead naturally to the methods to be discussed in the following chapter but also because we wish to establish the relationship between those methods and methods like OrthoDir and OrthoMin which form the subjects of later sections of the present chapter.

2.1. The Gram-Schmidt method

In its original form the Gram-Schmidt method [131], [220] or more accessibly [155], computes a sequence of orthonormal vectors $\{\mathbf{p}_j\}$ from a given sequence of linearly-independent vectors $\{\mathbf{w}_j\}$. In the following description the normalisation is omitted in the interests of clarity and consistency so the vectors generated by our version of the algorithm are merely orthogonal. We do, though, introduce normalisation when we come to describe a particular version of the Gram-Schmidt method, namely Arnoldi's method, a few pages further on.

The Gram-Schmidt algorithm works by putting $\mathbf{p}_1 = \mathbf{w}_1$ and, for $i = 1, 2, \dots$, calculating

$$\mathbf{p}_{i+1} = \mathbf{w}_{i+1} - \sum_{j=1}^i \mathbf{p}_j \alpha_{j,i+1}$$

where the scalars $\alpha_{j,i+1}$ are chosen so that $\mathbf{p}_j^T \mathbf{p}_{i+1} = 0$, $j = 1, 2, \dots, i$. This condition implies that $\alpha_{j,i+1} = \mathbf{p}_j^T \mathbf{w}_{i+1} / \mathbf{p}_j^T \mathbf{p}_j$ so that the above equation may be written

$$\mathbf{p}_{i+1} = \mathbf{w}_{i+1} - \sum_{j=1}^i \mathbf{p}_j \left(\frac{\mathbf{p}_j^T \mathbf{w}_{i+1}}{\mathbf{p}_j^T \mathbf{p}_j} \right)$$

or

$$\mathbf{p}_{i+1} = \mathbf{Q}_i^T \mathbf{w}_{i+1}$$

where

$$\mathbf{Q}_i = \mathbf{I} - \sum_{j=1}^i \frac{\mathbf{p}_j \mathbf{p}_j^T}{\mathbf{p}_j^T \mathbf{p}_j}$$

This is just the vector form of equation (2.1) with \mathbf{G} replaced by the identity matrix. We thus see that the use of equations (2.1) and (1.52) to generate a sequence of matrices having particular conjugacy properties is just a generalisation of the Gram-Schmidt method, where the generalisation takes two forms. In the first the vectors \mathbf{w}_j are replaced by the matrices \mathbf{W}_j , where $\mathbf{W}_j \in \mathbb{R}^{n \times r_j}$ for some positive integer r_j and in the second the computed matrices \mathbf{P}_j are required to satisfy equation (1.40) where \mathbf{G} is some symmetric but not necessarily definite matrix. The fact that \mathbf{G} may be indefinite introduces an element of uncertainty in the process but before examining this in detail we give the following *formal* description of the algorithm.

The Generalised Gram-Schmidt Algorithm (GGSA)

- (1) Select \mathbf{G} and \mathbf{W}_j , $j = 1, 2, \dots$. Set $\mathbf{Q}_0 = \mathbf{I}$, $\mathbf{P}_1 = \mathbf{W}_1$ and $i = 1$
- (2) while $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is nonsingular
 - (a) compute $\mathbf{Q}_i = \mathbf{Q}_{i-1} - \mathbf{G} \mathbf{P}_i (\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i)^{-1} \mathbf{P}_i^T$
 - (b) compute $\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{W}_{i+1}$
 - (c) set $i = i + 1$
- (3) endwhile
- (4) end GGSA.

We shall use these formal descriptions throughout to give overall but nevertheless precise descriptions of certain algorithms. The descriptions are complete and self-contained but skip over some of the computational details (in the GGSA, for example, one would not actually compute \mathbf{Q}_i but would store $\bar{\mathbf{P}}_i$ and $\bar{\mathbf{D}}_i$ and compute \mathbf{P}_{i+1} from these, probably using a version of the *modified Gram-Schmidt algorithm*² (see e.g. [129])). However the actual (computational) algorithms are to be regarded as refinements of the formal algorithms which would, using exact arithmetic, give identical results.

This deceptively simple algorithm [48] is, nevertheless, an algorithm of considerable power and utility. If we choose the matrices \mathbf{W}_i to be

$$\mathbf{W}_i = \begin{bmatrix} \mathbf{Y}_i & \mathbf{O} \\ \mathbf{O} & \mathbf{Z}_i \end{bmatrix},$$

where $\mathbf{Y}_i \in \mathbb{R}^{n \times r}$ and $\mathbf{Z}_i \in \mathbb{R}^{m \times r}$ have full (column) rank, and if \mathbf{G} is given by

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}$$

² In the *classical Gram-Schmidt algorithm*, \mathbf{P}_{i+1} is computed using a representation of \mathbf{Q}_i based on equation (1.46). In the *modified* version the calculation is based on the representation of equation (1.44) and is generally more stable [33].

where $\mathbf{A} \in \mathbb{R}^{m \times n}$, we obtain the block form of the CFG biconjugation method of Chu *et al* [72], to which it reverts if the matrices \mathbf{Y}_i and \mathbf{Z}_i have only one column. If, in the above, $\mathbf{A} = \mathbf{I}$ we obtain the block form of the two-sided Gram-Schmidt algorithm of Parlett [204]. Moreover, as we shall see, the GGSA forms the basis of practically all the Krylov methods for solving linear systems of equations.

From the above formal description we see that the GGSA does not terminate as long as the matrices $\mathbf{D}_i = \mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ are nonsingular. However if singularity does occur, the algorithm is forced to stop and it is important to know why this happens. In the context of the Krylov sequence methods, where the algorithm is being used to solve $\mathbf{G}\mathbf{x} = \mathbf{H}$, if \mathbf{D}_i is singular because \mathbf{P}_i is rank-deficient then normally a partial solution of the equation can be found. If, *a fortiori*, \mathbf{P}_i is null, usually the full solution may be computed. This type of termination involving the loss of rank of \mathbf{P}_i is called *regular termination*. If, on the other hand, \mathbf{P}_i has full rank and \mathbf{D}_i is singular because \mathbf{G} is indefinite then *serious breakdown* has occurred. This does not lead to the calculation of any sort of approximate solution and rescue procedures have to be initiated. We discuss both these possibilities in more detail later and restrict ourselves now to giving sufficient conditions for the matrices \mathbf{D}_i to be nonsingular. Before doing so, though, we give the following simple but extremely useful technical lemma.

Lemma 6. Let \mathbf{Q}_j be defined by equation (1.46). Then if, in this equation, $\overline{\mathbf{P}}_j$ is replaced by $\overline{\mathbf{P}}_j \overline{\mathbf{M}}_j$ where $\overline{\mathbf{M}}_j$ is any nonsingular matrix, \mathbf{Q}_j remains unchanged.

Proof. Straightforward, by substitution ■

This lemma indicates that projection matrices like \mathbf{Q}_j remain unaltered when the co-ordinate system defining their spaces is changed provided that the spaces themselves remain the same. If $\overline{\mathbf{P}}_j$ is replaced by $\overline{\mathbf{P}}_j \overline{\mathbf{M}}_j$, the basis of the column-space of $\overline{\mathbf{P}}_j$ is altered but the space so defined remains unchanged. This property enables us to choose any basis we please for the column-space of $\overline{\mathbf{P}}_j$ and yet obtain the identical projection matrix \mathbf{Q}_j , and this implies that many representations of \mathbf{Q}_j are possible.

2.2. Causes of breakdown*

In this section³ we examine the fundamental causes of breakdown of the GGSA and see precisely how these affect the matrices \mathbf{P}_i and $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$.

Theorem 7. Let $\mathbf{W}_j \in \mathbb{R}^{n \times r_j}$, $j = 1, 2, \dots, i + 1$, be an arbitrary sequence of matrices and $\mathbf{G} \in \mathbb{R}^{n \times n}$ be some symmetric matrix, let $\overline{\mathbf{W}}_j$ be defined by

$$\overline{\mathbf{W}}_j = [\mathbf{W}_1 \ \mathbf{W}_2 \ \dots \ \mathbf{W}_j] \quad (2.3)$$

³ Sections marked with an asterisk may be omitted at first reading.

and let the matrices $\overline{\mathbf{W}}_j^T \mathbf{G} \overline{\mathbf{W}}_j$, $j = 1, 2, \dots, i$, be nonsingular. Then \mathbf{P}_j , $j = 1, 2, \dots, i+1$, may be computed by the GGSA without breakdown and will satisfy $\overline{\mathbf{P}}_j = \overline{\mathbf{W}}_j \overline{\mathbf{U}}_j$ where $\overline{\mathbf{P}}_j$ is defined by equation (1.31) and $\overline{\mathbf{U}}_j$ is a block unit upper triangular matrix and thus nonsingular.

Proof. The theorem is clearly true for $j = 1$ with $\overline{\mathbf{U}}_1 = \mathbf{I}$ since $\mathbf{Q}_0 = \mathbf{I}$. We show that if it is true for $j = 1, 2, \dots, k$, where $k \leq i$, then it is also true for $j = k+1$.

From the induction hypothesis

$$\overline{\mathbf{P}}_k = \overline{\mathbf{W}}_k \overline{\mathbf{U}}_k \quad (2.4)$$

and since $\overline{\mathbf{U}}_k$ is nonsingular and, by hypothesis, $\overline{\mathbf{W}}_k^T \mathbf{G} \overline{\mathbf{W}}_k$ is also nonsingular it follows that $\overline{\mathbf{P}}_k^T \mathbf{G} \overline{\mathbf{P}}_k$ is nonsingular. Thus, from Lemma 6 and equations (1.46) and (2.4), \mathbf{Q}_k is given by

$$\mathbf{Q}_k = \mathbf{I} - \mathbf{G} \overline{\mathbf{W}}_k (\overline{\mathbf{W}}_k^T \mathbf{G} \overline{\mathbf{W}}_k)^{-1} \overline{\mathbf{W}}_k^T. \quad (2.5)$$

Since $\mathbf{P}_{k+1} = \mathbf{Q}_k^T \mathbf{W}_{k+1}$ we see that $\mathbf{P}_{k+1} = \mathbf{W}_{k+1} + \overline{\mathbf{W}}_k \mathbf{Y}_{k+1}$ for some matrix \mathbf{Y}_{k+1} , and from this equation and equation (2.4) it follows that $\overline{\mathbf{P}}_{k+1} = \overline{\mathbf{W}}_{k+1} \overline{\mathbf{U}}_{k+1}$ where

$$\overline{\mathbf{U}}_{k+1} = \begin{bmatrix} \overline{\mathbf{U}}_k & \mathbf{Y}_{k+1} \\ \mathbf{O} & \mathbf{I} \end{bmatrix}$$

The result follows. ■

Remark 1. We are not particularly concerned with the precise value of $\overline{\mathbf{U}}_k$ although this could be obtained by a more careful version of the above analysis. Its vital property is that it is always nonsingular.

Before proving the next theorem we introduce the idea of the *nullities* of a matrix since these are used in the proof. Whereas the rank of a matrix is unique a matrix has two nullities (row and column) and the column nullity is defined to be the number of columns of the matrix less the rank, with the row nullity similarly defined. The column nullity ν_c of a matrix \mathbf{A} , here denoted $\nu_c(\mathbf{A})$, is also the dimension of its right null-space so that there exist matrices \mathbf{X} of rank ν_c such that $\mathbf{AX} = \mathbf{O}$, but no matrix \mathbf{Y} of rank $\nu_c + 1$ such that $\mathbf{AY} = \mathbf{O}$. Using this idea the following properties are easy to establish:

- (1) If $\mathbf{AX} = \mathbf{O}$ then $\nu_c(\mathbf{A}) \geq \text{rank}(\mathbf{X})$
- (2) If \mathbf{B} is nonsingular then $\nu_c(\mathbf{AB}) = \nu_c(\mathbf{A})$
- (3) $\nu_c(\mathbf{W}^T \mathbf{G} \mathbf{W}) \geq \nu_c(\mathbf{W})$

Clearly if \mathbf{A} is square its row and column nullities are identical and in this case we can just refer to the nullity of \mathbf{A} . Moreover, if both nullities are zero then \mathbf{A} is nonsingular. The usefulness of nullities in the following theorem stems from the fact that we show that two matrices, whose dimensions and ranks are quite different, nevertheless have the same column nullity.

Theorem 8. Let the matrices \mathbf{G}, \mathbf{W}_j and $\overline{\mathbf{W}}_j$, $j = 1, 2, \dots, i+1$, satisfy the conditions of Theorem 7 and let the matrices \mathbf{P}_j , $j = 1, 2, \dots, i+1$, be computed by the GGSA. Then

- (a) $\nu_c(\mathbf{P}_{i+1}) = \nu_c(\overline{\mathbf{W}}_{i+1})$, and
- (b) $\nu_c(\mathbf{P}_{i+1}^T \mathbf{G} \mathbf{P}_{i+1}) = \nu_c(\overline{\mathbf{W}}_{i+1}^T \mathbf{G} \overline{\mathbf{W}}_{i+1})$.

Proof. (a) From Theorem 7, $\overline{\mathbf{P}}_{i+1} = \overline{\mathbf{W}}_{i+1} \overline{\mathbf{U}}_{i+1}$ where $\overline{\mathbf{U}}_{i+1}$ is nonsingular, so that $\nu_c(\overline{\mathbf{P}}_{i+1}) = \nu_c(\overline{\mathbf{W}}_{i+1})$. Let $\nu_c(\overline{\mathbf{P}}_{i+1}) = \nu_p$ and let σ be its total number of columns. Then there exists a matrix $\overline{\mathbf{X}}_{i+1} \in \mathbb{R}^{\sigma \times \nu_p}$ of rank ν_p such that $\overline{\mathbf{P}}_{i+1} \overline{\mathbf{X}}_{i+1} = \mathbf{O}$ so that if we partition $\overline{\mathbf{X}}_{i+1}$ as

$$\overline{\mathbf{X}}_{i+1} = \begin{bmatrix} \overline{\mathbf{X}}_i \\ \mathbf{X}_{i+1} \end{bmatrix},$$

equation (1.31) yields

$$\overline{\mathbf{P}}_i \overline{\mathbf{X}}_i + \mathbf{P}_{i+1} \mathbf{X}_{i+1} = \mathbf{O}.$$

Premultiplying this last equation by $\overline{\mathbf{P}}_i^T \mathbf{G}$ gives $\overline{\mathbf{D}}_i \overline{\mathbf{X}}_i = \mathbf{O}$ and since, by hypothesis $\overline{\mathbf{W}}_i^T \mathbf{G} \overline{\mathbf{W}}_i$ is nonsingular it follows from Theorem 7 that $\overline{\mathbf{D}}_i = \overline{\mathbf{P}}_i^T \mathbf{G} \overline{\mathbf{P}}_i$ is nonsingular and this implies that $\overline{\mathbf{X}}_i = \mathbf{O}$. Thus $\mathbf{P}_{i+1} \mathbf{X}_{i+1} = \mathbf{O}$ where \mathbf{X}_{i+1} has rank ν_p so that $\nu_c(\mathbf{P}_{i+1}) \geq \nu_p$, and as it cannot exceed ν_p (since then $\nu_c(\overline{\mathbf{P}}_{i+1}) > \nu_p$, contrary to hypothesis) it must be equal to ν_p .

(b) Let $\nu_c(\overline{\mathbf{W}}_{i+1}^T \mathbf{G} \overline{\mathbf{W}}_{i+1}) = \nu_w$ so that, from equation (2.4) with $k = i+1$, $\nu_c(\overline{\mathbf{D}}_{i+1}) = \nu_w$. Now

$$\overline{\mathbf{D}}_{i+1} = \begin{bmatrix} \overline{\mathbf{D}}_i & \mathbf{O} \\ \mathbf{O} & \mathbf{D}_{i+1} \end{bmatrix}$$

and since $\overline{\mathbf{D}}_i$ is nonsingular this implies that $\nu_c(\mathbf{D}_{i+1}) = \nu_w$. ■

2.3. Discussion and summary

Theorem 7 indicates that if we have a sequence of matrices $\{\mathbf{W}_j\}$ we can, provided that the matrices $\overline{\mathbf{W}}_j^T \mathbf{G} \overline{\mathbf{W}}_j$ are nonsingular where $\overline{\mathbf{W}}_j$ is defined by equation (2.3), generate a sequence of conjugate matrices $\{\mathbf{P}_j\}$ which may then be used to solve equation (1.1). It also indicates that the GGSA breaks down if, for some j , $\overline{\mathbf{W}}_j^T \mathbf{G} \overline{\mathbf{W}}_j$ becomes singular, and this may happen in either of two ways. The first is if $\overline{\mathbf{W}}_j$ becomes rank-deficient when, from Theorem 8, \mathbf{P}_j becomes rank-deficient and hence \mathbf{D}_j becomes singular. This case thus gives rise to the regular termination referred to previously. We note that regular termination can happen even if \mathbf{W}_j has full rank; all that is necessary for the breakdown to occur is that one of the columns of \mathbf{W}_j is a linear combination of those of $\overline{\mathbf{W}}_{j-1}$. Theorem 8 thus shows that the rank-deficiency of $\overline{\mathbf{W}}_j$, which may not reveal itself unless all its columns are taken

into account, will be detected in $\bar{\mathbf{P}}_j$ by considering only those columns that form its final partition \mathbf{P}_j . Another consequence of Theorem 8 is that the GGSA must eventually terminate since as soon as the number of columns of $\bar{\mathbf{W}}_j$ exceeds n , both $\bar{\mathbf{W}}_j$ and $\bar{\mathbf{P}}_j$ (and hence \mathbf{P}_j) become rank-deficient and again \mathbf{D}_j becomes singular. Finally if $\bar{\mathbf{W}}_j$ becomes prematurely rank-deficient (i.e. becomes rank-deficient when the number of columns of $\bar{\mathbf{W}}_j$ is strictly less than n) then $\bar{\mathbf{W}}_k$ is rank-deficient for all $k \geq j$. There is thus no point in proceeding with a modified form of the algorithm in the hope that, for some $k > j$, $\bar{\mathbf{W}}_k^T \mathbf{G} \bar{\mathbf{W}}_k$ will be nonsingular because in the case of premature rank-deficiency this can never happen. However, with this type of termination it is often possible to obtain a partial, or in some cases even a complete, solution of $\mathbf{G}\mathbf{X} = \mathbf{H}$. (A *partial solution* may be obtained if, for any reason during the course of the calculation, the matrix \mathbf{F}_i becomes rank-deficient so that there exists some matrix \mathbf{U}_i having full column rank such that $\mathbf{F}_i \mathbf{U}_i = \mathbf{O}$. Since $\mathbf{F}_i = \mathbf{G}\mathbf{X}_i - \mathbf{H}$ this implies that $\mathbf{G}\mathbf{X}_i \mathbf{U}_i = \mathbf{H}\mathbf{U}_i$. The matrix $\mathbf{X}_i \mathbf{U}_i$ may be regarded as being a partial solution of $\mathbf{G}\mathbf{X} = \mathbf{H}$ since if \mathbf{V}_i is a matrix such that $[\mathbf{U}_i \ \mathbf{V}_i]$ is square and nonsingular it is only necessary to solve $\mathbf{G}\mathbf{Y} = \mathbf{H}\mathbf{V}_i$ in order to extract the full solution of $\mathbf{G}\mathbf{X} = \mathbf{H}$. See Chapter 8 below.)

Thus regular termination of the algorithm due to the rank-deficiency of $\bar{\mathbf{W}}_j$ usually results in the calculation of at least a partial solution of $\mathbf{G}\mathbf{X} = \mathbf{H}$, even if the termination is premature. This is in marked contrast to the second mode of breakdown which occurs if $\bar{\mathbf{W}}_j$ has full rank but, because \mathbf{G} is indefinite, $\bar{\mathbf{W}}_j^T \mathbf{G} \bar{\mathbf{W}}_j$ is singular. It follows from Theorem 8(b) that this is the underlying cause of serious breakdown. In this case, however, it is possible that if $\bar{\mathbf{W}}_j^T \mathbf{G} \bar{\mathbf{W}}_j$ is singular then, for some $k > j$, $\bar{\mathbf{W}}_k^T \mathbf{G} \bar{\mathbf{W}}_k$ is nonsingular, and this possibility opens the doors to rescue procedures involving the so-called *look-ahead versions* of the algorithms (see Chapter 7).

Suppose now that $\mathbf{W}_j \in \mathbb{R}^{n \times r_j}$, $j = 1, 2, \dots, s$, for some positive integer s and that $\sum_{j=1}^s r_j = n$ so that $\bar{\mathbf{W}}_s$ is square. Suppose further that for some j , $1 \leq j < s$, $\bar{\mathbf{W}}_j^T \mathbf{G} \bar{\mathbf{W}}_j$ is singular but that \mathbf{G} is nonsingular and that $\bar{\mathbf{W}}_s$ has full rank, i.e. is nonsingular. Then, automatically, $\bar{\mathbf{W}}_s^T \mathbf{G} \bar{\mathbf{W}}_s$ will be nonsingular and in principle look-ahead will work. Thus if \mathbf{G} is nonsingular and look-ahead fails, it fails only because $\bar{\mathbf{W}}_s$ is rank-deficient and this implies that, in addition to serious breakdown, premature termination has also taken place. This total failure of the algorithm, due to a combination of serious breakdown and premature termination, is known as *incurable breakdown* and will be discussed no further here beyond saying that its occurrence is extremely rare.

We now summarise the various ways in which the GGSA can terminate.

Types of Termination

- (1) If $\bar{\mathbf{W}}_j^T \mathbf{G} \bar{\mathbf{W}}_j$ is nonsingular for $j \leq i - 1$ and the rank of $\bar{\mathbf{W}}_i$ is equal to that of $\bar{\mathbf{W}}_{i-1}$ then, from Theorem 8, \mathbf{P}_i is null. If \mathbf{G} is positive definite then this type of termination *always* occurs if exact arithmetic is assumed and if

the GGSA is used to generate a vector sequence ($r = 1$), since \mathbf{p}_i is either non-null (full rank) or null. It *can* occur if the system $\mathbf{GX} = \mathbf{H}$ has the form of equation (1.38) even though in this case \mathbf{G} is not positive definite. When this type of termination occurs it is normally possible to use the matrices $\{\mathbf{P}_j\}$ generated by the GGSA to obtain the complete solution of $\mathbf{GX} = \mathbf{H}$, and algorithms exploiting this possibility are described later in this chapter. We should note though that in practice these precise forms of termination are invariably clouded by rounding error and the best that can be hoped for is some form of approximate solution.

- (2) If $\overline{\mathbf{W}}_i$ does not have full rank but $\text{rank}(\overline{\mathbf{W}}_i) > \text{rank}(\overline{\mathbf{W}}_{i-1})$ and

$$\nu_c(\overline{\mathbf{W}}_i^T \mathbf{G} \overline{\mathbf{W}}_i) = \nu_c(\overline{\mathbf{W}}_i),$$

it is often possible to obtain a partial solution of $\mathbf{GX} = \mathbf{H}$. This is usually the case for the more general block methods where $r \geq 2$, and its consequences are described in Chapter 8.

- (3) If $\overline{\mathbf{W}}_i$ has full rank but $\overline{\mathbf{W}}_i^T \mathbf{G} \overline{\mathbf{W}}_i$ is singular then serious breakdown has occurred. In the absence of incurable breakdown it may be overcome by using look-ahead techniques (See Chapter 7)
- (4) If $\overline{\mathbf{W}}_i$ does not have full rank and $\nu_c(\overline{\mathbf{W}}_i^T \mathbf{G} \overline{\mathbf{W}}_i) > \nu_c(\overline{\mathbf{W}}_i)$ then incurable breakdown has occurred. The techniques of Chapter 7 and Chapter 8 are insufficient to rescue the algorithm which must then be restarted from a different initial point.

This completes the description of the GGSA. We now consider a special case (Arnoldi's method) where the matrices \mathbf{W}_j are given by $\mathbf{W}_j = \mathbf{B} \mathbf{P}_{j-1} \mathbf{H}_{j,j-1}^{-1}$ for some $\mathbf{B} \in \mathbb{R}^{n \times n}$ and normalisation matrices $\mathbf{H}_{j,j-1}$. This algorithm forms the basis of about half of the methods to be discussed subsequently.

2.4. Arnoldi's method

Arnoldi's method [6] is merely the original Gram-Schmidt method applied to the vector sequence $\{\mathbf{w}_j\}$, where \mathbf{w}_1 is arbitrary and $\mathbf{w}_{j+1} = \mathbf{B} \mathbf{p}_j$ for $j = 1, 2, \dots, n$, and for some matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$. The method is used to generate a sequence of orthonormal vectors and \mathbf{B} is often symmetric. In its original form it may be expressed as:

The Original Arnoldi Algorithm (OAA)

- (1) Select \mathbf{B} and \mathbf{v}_1 . Set $\mathbf{Q}_0 = \mathbf{I}$ and $j = 1$
- (2) while $\mathbf{v}_j \neq 0$
 - (a) compute $\mathbf{p}_j = \mathbf{v}_j / \|\mathbf{v}_j\|$
 - (b) compute $\mathbf{Q}_j = \mathbf{Q}_{j-1} - \mathbf{p}_j \mathbf{p}_j^T$
 - (c) compute $\mathbf{v}_{j+1} = \mathbf{Q}_j \mathbf{B} \mathbf{p}_j$ (since $\mathbf{Q}_j = \mathbf{Q}_j^T$)
 - (d) set $j = j + 1$

- (3) endwhile
- (4) end OAA.

We consider now the more general case where we compute matrices conjugate with respect to \mathbf{G} using the GGSA but with the matrices \mathbf{W}_{j+1} being given by

$$\mathbf{W}_{j+1} = \mathbf{B}\mathbf{P}_j\mathbf{H}_{j+1,j}^{-1} \quad (2.6)$$

where $\mathbf{H}_{j+1,j}^{-1}$ is some arbitrary normalisation matrix. We place no restriction on \mathbf{B} other than requiring it to be nonsingular. Formally we have

The Generalised Arnoldi Algorithm (GAA)

- (1) Select \mathbf{G} , \mathbf{B} and \mathbf{P}_1 (we assume that \mathbf{G} is symmetric and that \mathbf{P}_1 has already been appropriately normalized). Set $\mathbf{Q}_0 = \mathbf{I}$ and $j = 1$
- (2) while $\mathbf{P}_j^T \mathbf{G} \mathbf{P}_j$ is nonsingular
 - (a) compute $\mathbf{Q}_j = \mathbf{Q}_{j-1} - \mathbf{G} \mathbf{P}_j (\mathbf{P}_j^T \mathbf{G} \mathbf{P}_j)^{-1} \mathbf{P}_j^T$
 - (b) compute $\mathbf{P}_{j+1} = \mathbf{Q}_j^T \mathbf{B} \mathbf{P}_j \mathbf{H}_{j+1,j}^{-1}$ where $\mathbf{H}_{j+1,j}$ is chosen to normalise \mathbf{P}_{j+1} as required
 - (c) set $j = j + 1$
- (3) endwhile
- (4) end GAA.

Let then the sequence $\{\mathbf{P}_j\}$, $1 \leq j \leq i$, be computed by the GAA. Provided that serious breakdown does not occur, equations (1.46), (1.52) and (2.6) give

$$\mathbf{P}_{j+1} \mathbf{H}_{j+1,j} = \mathbf{B} \mathbf{P}_j - \bar{\mathbf{P}}_j \mathbf{Y}_j \quad (2.7)$$

for some matrix $\mathbf{Y}_j \in \mathbb{R}^{rj \times r}$. Equation (2.7) may be written

$$\mathbf{B} \mathbf{P}_j = \bar{\mathbf{P}}_{j+1} \begin{bmatrix} \mathbf{Y}_j \\ \mathbf{H}_{j+1,j} \end{bmatrix} \quad (2.8)$$

and if

$$\mathbf{Y}_j^T = [\mathbf{H}_{1j}^T \ \mathbf{H}_{2j}^T \ \dots \ \mathbf{H}_{jj}^T] \quad (2.9)$$

for some matrices $\mathbf{H}_{ij} \in \mathbb{R}^{r \times r}$, equation (2.8) for $j = 1, 2, \dots, i$, becomes

$$\mathbf{B} \bar{\mathbf{P}}_i = \bar{\mathbf{P}}_{i+1} \tilde{\mathbf{H}}_{i+1} \quad (2.10)$$

where $\tilde{\mathbf{H}}_{i+1} \in \mathbb{R}^{(i+1)r \times ir}$ is given by

$$\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} & \dots & \mathbf{H}_{1,i-1} & \mathbf{H}_{1i} \\ \mathbf{H}_{21} & \mathbf{H}_{22} & \dots & \mathbf{H}_{2,i-1} & \mathbf{H}_{2i} \\ \mathbf{O} & \mathbf{H}_{32} & \dots & \mathbf{H}_{3,i-1} & \mathbf{H}_{3i} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{O} & \mathbf{O} & \dots & \mathbf{H}_{i,i-1} & \mathbf{H}_{ii} \\ \mathbf{O} & \mathbf{O} & \dots & \mathbf{O} & \mathbf{H}_{i+1,i} \end{bmatrix} \quad (2.11)$$

and is thus block upper-Hessenberg. We note from these equations that

$$\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \tilde{\mathbf{H}}_i & \mathbf{Y}_i \\ \mathbf{O} & \mathbf{H}_{i+1,i} \end{bmatrix} \quad (2.12)$$

so that $\tilde{\mathbf{H}}_i$ is a submatrix of $\tilde{\mathbf{H}}_{i+1}$. Another useful matrix is $\bar{\mathbf{H}}_i$ where

$$\bar{\mathbf{H}}_i = \begin{bmatrix} \tilde{\mathbf{H}}_i & \mathbf{Y}_i \end{bmatrix} \quad (2.13)$$

so that, from equation (2.11),

$$\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} & \bar{\mathbf{H}}_i \\ [\mathbf{O} \ \mathbf{O} \ \dots \ \mathbf{O} \ \mathbf{H}_{i+1,i}] \end{bmatrix}. \quad (2.14)$$

Thus $\bar{\mathbf{H}}_i \in \mathbb{R}^{r_i \times r_i}$ is the block upper-Hessenberg matrix that comprises the first i block rows of $\tilde{\mathbf{H}}_{i+1}$.

Suppose now that the sequence of matrices $\{\mathbf{P}_i\}$ is generated by the GAA and assume that the first type of termination occurs for $i = s$, i.e. assume that $\bar{\mathbf{D}}_s$ is nonsingular but that \mathbf{P}_{s+1} is null. Equations (2.7) and (2.10) with $j = s$ and $i = s$ become

$$\mathbf{B}\mathbf{P}_s = \bar{\mathbf{P}}_s \mathbf{Y}_s. \quad (2.15)$$

and

$$\mathbf{B}\bar{\mathbf{P}}_s = \bar{\mathbf{P}}_s \bar{\mathbf{H}}_s \quad (2.16)$$

where $\bar{\mathbf{H}}_s \in \mathbb{R}^{sr \times sr}$ is the block upper Hessenberg matrix defined by equation (2.13). This upper Hessenberg matrix is characteristic of the Arnoldi method and is exploited by at least some of the Krylov sequence algorithms. Its form makes it very easy to triangularize by orthogonal transformations, especially when vector sequences are being generated and its entries are scalars, and we now show that it is nonsingular if \mathbf{B} is nonsingular.

Lemma 9. Let \mathbf{B} be nonsingular and let the matrices $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$, $1 \leq i \leq s$, computed by the GAA be nonsingular. Let $\mathbf{P}_{s+1} = \mathbf{O}$ so that $\bar{\mathbf{P}}_s$ satisfies equation (2.16). Then $\bar{\mathbf{H}}_s$ is nonsingular.

Proof. Assume the contrary so that there exists a vector $\mathbf{y} \neq \mathbf{0}$ such that $\bar{\mathbf{H}}_s \mathbf{y} = \mathbf{0}$. Then, from equation (2.16), $\mathbf{B}\bar{\mathbf{P}}_s \mathbf{y} = \mathbf{0}$. But \mathbf{B} is nonsingular and since, from the hypotheses of the lemma and from Lemma 4, $\bar{\mathbf{P}}_s$ has full rank this leads to a contradiction and the result follows. ■

At this point we might note that if, in the GAA, $\mathbf{G} = \mathbf{I}$ and $\mathbf{H}_{j+1,j}$ is chosen so that $\mathbf{P}_{j+1}^T \mathbf{P}_{j+1} = \mathbf{I}$ (so that the algorithm becomes the block form of the original Arnoldi algorithm) then $\bar{\mathbf{P}}_s^T \bar{\mathbf{P}}_s = \mathbf{I}$. Premultiplication of equation (2.16) by $\bar{\mathbf{P}}_s^T$ then gives

$$\bar{\mathbf{P}}_s^T \mathbf{B}\bar{\mathbf{P}}_s = \bar{\mathbf{H}}_s \quad (2.17)$$

so that if, in addition, \mathbf{B} is symmetric then $\overline{\mathbf{H}}_s$ is also symmetric. But $\overline{\mathbf{H}}_s$ is block upper Hessenberg so if it is also symmetric it has to be block tridiagonal (tridiagonal if vector sequences are being generated), a fact that is exploited by the algorithm MinRes (see below).

Although the use of the GAA with the first type of termination forms the basis of some practical linear equation solvers, other methods based on the GAA do not use the termination properties at all and stop the algorithm after computing \mathbf{P}_{i+1} , where \mathbf{P}_{i+1} has full rank. If \mathbf{P}_{i+1} is not null it cannot be assumed without further hypothesis that $\overline{\mathbf{H}}_i$ is nonsingular and in some algorithms (e.g. FOM) this can be a potential source of instability. We now show, though, that if $\mathbf{B} = \mathbf{KG}$ and \mathbf{K} is positive real then $\overline{\mathbf{H}}_i$ is nonsingular for all i . Since the term *positive real* is perhaps unfamiliar we first define and outline some properties of positive real matrices.

Definition 1. A non-symmetric matrix \mathbf{A} is said to be positive real if, for all $\mathbf{x} \neq \mathbf{0}$, $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$.

This is clearly equivalent to $\mathbf{A} + \mathbf{A}^T$ being positive definite and many properties of positive real matrices and positive definite matrices are similar. Trivially, positive real matrices are nonsingular and if \mathbf{A} is positive real then so is \mathbf{A}^{-1} (since \mathbf{A} is nonsingular, if $\mathbf{x} \neq \mathbf{0}$ then $\mathbf{x}^T \mathbf{A}^{-1} \mathbf{x} = \mathbf{y}^T \mathbf{A}^T \mathbf{A}^{-1} \mathbf{A} \mathbf{y} = \mathbf{y}^T \mathbf{A}^T \mathbf{y} > 0$ for all $\mathbf{y} \neq \mathbf{0}$). More significantly, for our purposes, if \mathbf{M} has full column rank and \mathbf{A} is positive real then $\mathbf{M}^T \mathbf{A} \mathbf{M}$ is nonsingular (since, trivially, it is also positive real).

Note that unlike positive definite matrices, the powers of positive real matrices are not necessarily positive real even though the inverse always is. If

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix} \text{ so that } \mathbf{A}^{-1} = \begin{bmatrix} 1/5 & -2/5 \\ 2/5 & 1/5 \end{bmatrix} \text{ and } \mathbf{A}^2 = \begin{bmatrix} -3 & 4 \\ -4 & -3 \end{bmatrix}$$

then both \mathbf{A} and \mathbf{A}^{-1} are positive real but \mathbf{A}^2 most certainly is not.

Lemma 10. Let $\overline{\mathbf{H}}_i$ be generated by the GAA where $\mathbf{B} = \mathbf{KG}$, \mathbf{G} is symmetric and nonsingular and \mathbf{K} is positive real. Let $\overline{\mathbf{D}}_i = \overline{\mathbf{P}}_i^T \mathbf{G} \overline{\mathbf{P}}_i$ be nonsingular. Then $\overline{\mathbf{H}}_i$ is nonsingular.

Proof. Under the hypotheses of the lemma the GAA generates a sequence of matrices $\{\mathbf{P}_j\}$ conjugate with respect to \mathbf{G} . Premultiplying equation (2.10) by $\overline{\mathbf{P}}_i^T \mathbf{G}$ gives, from equation (2.14) and since $\overline{\mathbf{P}}_i^T \mathbf{G} \overline{\mathbf{P}}_{i+1} = [\overline{\mathbf{D}}_i \ \mathbf{O}]$,

$$\overline{\mathbf{P}}_i^T \mathbf{G} \mathbf{K} \mathbf{G} \overline{\mathbf{P}}_i = \overline{\mathbf{D}}_i \overline{\mathbf{H}}_i.$$

Since $\overline{\mathbf{D}}_i$ is nonsingular by hypothesis it follows from the conditions of the lemma that $\mathbf{G} \overline{\mathbf{P}}_i$ has full column rank. Thus, since \mathbf{K} is positive real, $\overline{\mathbf{P}}_i^T \mathbf{G} \mathbf{K} \mathbf{G} \overline{\mathbf{P}}_i$ is nonsingular and the lemma follows. ■

2.5. OrthoDir and GCR

We are now in a position to derive the methods Orthodir [260] and GCR [101] which are based on the generalised GS and Arnoldi algorithms. They both solve the equation $\mathbf{Ax} = \mathbf{b}$ expressed as $\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b}$ by generating a sequence of vectors conjugate with respect to $\mathbf{A}^T \mathbf{A}$. We first, though, describe generalisations GODir and GOMin which solve the equations $\mathbf{GX} = \mathbf{H}$ by generating sequences of matrices conjugate with respect to \mathbf{G} . We do this in order to construct a unifying theory that includes most algorithms of conjugate-gradient type as special cases as well as OrthoDir and GCR. These latter methods may be derived from GODir and GOMin simply by replacing the matrices \mathbf{P}_j by vectors \mathbf{p}_j and by substituting $\mathbf{A}^T \mathbf{A}$ for \mathbf{G} and \mathbf{A}^{-T} for \mathbf{K} .

GODir and GOMin use the generalised GS and Arnoldi algorithms to generate a sequence of conjugate matrices $\{\mathbf{P}_j\}$. These matrices are then used, via equation (1.36), to generate a sequence of approximate solutions \mathbf{X}_i of $\mathbf{GX} = \mathbf{H}$. The matrix \mathbf{B} in the GAA is set equal to \mathbf{KG} for some arbitrary nonsingular matrix \mathbf{K} and $\mathbf{H}_{i+1,i}$ is chosen to normalise in some sense⁴ \mathbf{P}_{i+1} . Note that \mathbf{P}_1 must be set equal to \mathbf{KF}_1 if GODir is to give the exact solution of $\mathbf{GX} = \mathbf{H}$ on termination.

We now give a formal description of GODir and GOMin and show for GODir that if the first type of termination occurs after s steps, \mathbf{X}_{s+1} is the solution of $\mathbf{GX} = \mathbf{H}$.

Generalised OrthoDir/OrthoMin (GODir/GOMin)

- (1) Select \mathbf{G} , \mathbf{H} , \mathbf{K} and \mathbf{X}_1 . Set $\mathbf{Q}_0 = \mathbf{I}$ and $i = 1$. Compute $\mathbf{F}_1 = \mathbf{GX}_1 - \mathbf{H}$ and $\mathbf{P}_1 = \mathbf{KF}_1$
- (2) while $\mathbf{P}_i^T \mathbf{GP}_i$ is nonsingular
 - (a) compute $\mathbf{D}_i = \mathbf{P}_i^T \mathbf{GP}_i$
 - (b) compute $\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i$
 - (c) compute $\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{GP}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i$
 - (d) compute $\mathbf{Q}_i = \mathbf{Q}_{i-1} - \mathbf{GP}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T$
 - (i) compute $\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{KGP}_i \mathbf{H}_{i+1,i}^{-1}$ (GODir only)
 - (ii) compute $\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{KF}_{i+1}$ (GOMin only)
 - (e) set $i = i + 1$
- (3) endwhile
- (4) end GODir/GOMin.

Theorem 11 (Finite termination GODir). Let $\{\mathbf{X}_i\}$ be computed by GODir, where \mathbf{G} and \mathbf{K} are nonsingular and \mathbf{G} is symmetric, let the matrices $\mathbf{P}_i^T \mathbf{GP}_i$, $1 \leq i \leq s$, computed by GODir be nonsingular and let $\mathbf{P}_{s+1} = \mathbf{O}$. Then $\mathbf{GX}_{s+1} = \mathbf{H}$.

⁴ If $\mathbf{P}_{i+1} = \mathbf{p}_{i+1}$ then $\mathbf{H}_{i+1,i} = h_{i+1,i}$ is usually chosen (at least for the Lanczos algorithms) such that $\|\mathbf{p}_{i+1}\| = 1$. If $\mathbf{P}_{i+1} = \begin{bmatrix} \mathbf{u}_{i+1} & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_{i+1} \end{bmatrix}$ it is chosen so that $\|\mathbf{u}_{i+1}\| = \|\mathbf{v}_{i+1}\| = 1$. For the HS algorithms in both cases it is usually set to be the unit matrix.

Proof. From equation (2.16) and Lemma 9 we have, since $\mathbf{B} = \mathbf{KG}$ and \mathbf{K} is nonsingular by hypothesis,

$$\mathbf{G}\bar{\mathbf{P}}_s\mathbf{H}_s^{-1} = \mathbf{K}^{-1}\bar{\mathbf{P}}_s. \quad (2.18)$$

Now \mathbf{P}_1 is the first block column of $\bar{\mathbf{P}}_s$ so that $\mathbf{P}_1 = \bar{\mathbf{P}}_s\mathbf{E}_1$, where \mathbf{E}_1 consists of the first r columns of the unit matrix of appropriate order. Hence since $\mathbf{P}_1 = \mathbf{KF}_1$ by imposition, $\mathbf{F}_1 = \mathbf{K}^{-1}\bar{\mathbf{P}}_s\mathbf{E}_1$ so that, from equation (2.18),

$$\mathbf{F}_1 = \mathbf{G}\bar{\mathbf{P}}_s\mathbf{H}_s^{-1}\mathbf{E}_1.$$

But from equation (1.43) we have $\mathbf{F}_{s+1} = \mathbf{Q}_s\mathbf{F}_1$ and since, from equation (1.48), $\mathbf{Q}_s\mathbf{G}\bar{\mathbf{P}}_s = \mathbf{O}$ it follows immediately that $\mathbf{F}_{s+1} = \mathbf{O}$ so that \mathbf{X}_{s+1} is the required solution. ■

Corollary 12. If $\mathbf{G} = \mathbf{I}$ then the theorem is valid if $\mathbf{P}_1 = \mathbf{K}^r\mathbf{F}_1$ for any integer r .

Proof. If $\mathbf{G} = \mathbf{I}$ equation (2.18) yields

$$\bar{\mathbf{P}}_s\mathbf{H}_s^{-r} = \mathbf{K}^{-r}\bar{\mathbf{P}}_s.$$

If $\mathbf{P}_1 = \mathbf{K}^r\mathbf{F}_1$ then $\mathbf{F}_1 = \bar{\mathbf{P}}_s\mathbf{H}_s^{-r}\mathbf{E}_1$ and $\mathbf{F}_{s+1} = \mathbf{Q}_s\mathbf{F}_1 = \mathbf{O}$ as before. ■

This somewhat surprising theorem and its corollary are true even if the Arnoldi algorithm terminates after only a few steps. All it requires is that no breakdown occurs and this is guaranteed if \mathbf{G} is definite and the matrices \mathbf{P}_j , $1 \leq j \leq s$, have full rank. Since the latter is always true for the vector version of the algorithm we have, in principle, a breakdown-free method for solving the equation $\mathbf{Gx} = \mathbf{h}$ in the case where \mathbf{G} is positive definite.

The algorithm OrthoDir solves the equation $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is non-singular but not symmetric. It does so by minimising $\|\mathbf{r}\|$ at each step, where the residual vector \mathbf{r} is defined by $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$. In the generalised algorithm described above, therefore, $\mathbf{G} = \mathbf{A}^T\mathbf{A}$ and is positive definite since \mathbf{A} is assumed to be non-singular, $\mathbf{h} = \mathbf{A}^T\mathbf{b}$ and $\mathbf{f} = \mathbf{Gx} - \mathbf{h} = \mathbf{A}^T\mathbf{r}$. Vector sequences are generated ($r = 1$) and \mathbf{K} is chosen to be \mathbf{A}^{-T} . Since $\mathbf{p}_1 = \mathbf{Kf}_1$ it follows from the above definitions that $\mathbf{p}_1 = \mathbf{r}_1$, the residual vector computed at the arbitrary initial value \mathbf{x}_1 of \mathbf{x} . Substituting these values in equations (1.22) and (1.52) then yields, from equations (2.1) and (1.53),

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{A}^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A}^T \mathbf{A} \mathbf{p}_i} \right) \quad (2.19)$$

and

$$\mathbf{p}_{i+1} = \mathbf{Ap}_i - \sum_{j=1}^i \mathbf{p}_j \left(\frac{\mathbf{p}_j^T \mathbf{A}^T \mathbf{A}^2 \mathbf{p}_i}{\mathbf{p}_j^T \mathbf{A}^T \mathbf{A} \mathbf{p}_j} \right). \quad (2.20)$$

Since no serious breakdown can occur the algorithm yields the exact solution in at most n iterations, although if n is sufficiently large it may be necessary to use the algorithm iteratively (see Practical Considerations, below).

For the algorithm GoMin [101]

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{K} \mathbf{F}_{i+1} \quad (2.21)$$

so it is not strictly a version of the GAA. Its termination properties are similar to those of GODir but with one important exception; in order to guarantee that termination yields an exact solution for GOMin it is necessary to assume that \mathbf{K} is positive real.

We shall see that the necessity for \mathbf{K} to be positive real is a feature of the HS algorithms which is not shared by their Lanczos equivalents. It implies that the Lanczos algorithms should be more stable than the HS ones and perhaps, using exact arithmetic, they are. We return to this topic in Chapter 9.

Theorem 13 (Finite termination GOMin). Let $\{\mathbf{X}_i\}$ and $\{\mathbf{P}_i\}$ be computed by GOMin, where \mathbf{K} is positive real and \mathbf{G} is nonsingular and symmetric, and let the algorithm terminate after s steps. Then $\nu_c(\mathbf{P}_{s+1}) = \nu_c(\mathbf{F}_{s+1})$, where ν_c denotes a matrix column nullity.

Proof. (a) Let $\nu_c(\mathbf{F}_{s+1}) = \nu_F$ so that there exists a matrix \mathbf{X} of rank ν_F such that $\mathbf{F}_{s+1}\mathbf{X} = \mathbf{O}$. Hence, from equation (2.21), $\mathbf{P}_{s+1}\mathbf{X} = \mathbf{O}$ so that if $\nu_P = \nu_c(\mathbf{P}_{s+1})$ then $\nu_P \geq \nu_F$.

(b) Let $\nu_c(\mathbf{P}_{s+1}) = \nu_P$ so that there exists a matrix \mathbf{Y} of rank ν_P such that $\mathbf{P}_{s+1}\mathbf{Y} = \mathbf{O}$. Then, from equation (2.21), $\mathbf{Q}_s^T \mathbf{K} \mathbf{F}_{s+1} \mathbf{Y} = \mathbf{O}$ and hence

$$\mathbf{Y}^T \mathbf{F}_{s+1}^T \mathbf{Q}_s^T \mathbf{K} \mathbf{F}_{s+1} \mathbf{Y} = \mathbf{O}. \quad (2.22)$$

Now the sequence $\{\mathbf{X}_j\}$ has been computed by equation (1.36) and since, by construction, the matrices \mathbf{P}_j are conjugate with respect to \mathbf{G} , equation (1.43) is valid so that $\mathbf{F}_{s+1} = \mathbf{Q}_s \mathbf{F}_1$. But \mathbf{Q}_s is idempotent so that $\mathbf{F}_{s+1} = \mathbf{Q}_s \mathbf{F}_{s+1}$ and this implies, from equation (2.22), that $\mathbf{Y}^T \mathbf{F}_{s+1}^T \mathbf{K} \mathbf{F}_{s+1} \mathbf{Y} = \mathbf{O}$. Since, by hypothesis, \mathbf{K} is positive real it follows that $\mathbf{F}_{s+1} \mathbf{Y} = \mathbf{O}$ so that $\nu_F \geq \nu_P$. Thus, from (a), $\nu_F = \nu_P$ proving the theorem. ■

This theorem is valid for all types of termination of GOMin but its applicability here is to give reassurance that \mathbf{P}_i will not be rank-deficient if \mathbf{F}_i has full rank. Thus if GOMin is used to generate vector sequences, \mathbf{G} is positive definite and \mathbf{K} is positive real then GOMin will not terminate until the exact solution of $\mathbf{G}\mathbf{x} = \mathbf{h}$ has been found. If $\mathbf{G} = \mathbf{A}^T \mathbf{A}$, $\mathbf{h} = \mathbf{A}^T \mathbf{b}$, $\mathbf{K} = \mathbf{A}^{-T}$ (the same values as for OrthoDir) and in addition \mathbf{A} is positive real then all the above conditions are satisfied and GOMin reduces to the generalized conjugate residual algorithm (GCR) for which equation (2.20) becomes

$$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} - \sum_{j=1}^i \mathbf{p}_j \left(\frac{\mathbf{p}_j^T \mathbf{A}^T \mathbf{A} \mathbf{r}_{i+1}}{\mathbf{p}_j^T \mathbf{A}^T \mathbf{A} \mathbf{p}_j} \right) \quad (2.23)$$

and for which \mathbf{x}_{i+1} is given by equation (2.19). Since the sum in the above equation runs from 1 to i it is possible that for large n , just as in OrthoDir, the algorithm as described above will need too much time and memory to be practical and restarted

or truncated forms will have to be devised. This may be done precisely as in the case of OrthoDir.

A disadvantage of GODir, shared by all the other algorithms described in this chapter, is the amount of labour and memory involved in the calculation of \mathbf{P}_{i+1} (Step 2d.i of the algorithm). Although for reasons of clarity this step is expressed in terms of \mathbf{Q}_i , since $\mathbf{Q}_i \in \mathbb{R}^{n \times n}$ and is generally full, it would not be stored in the computer as such but probably as $\bar{\mathbf{P}}_i$ (see equation (1.46)), and \mathbf{P}_{i+1} would be computed using a Gram-Schmidt technique. Thus the amount of work needed to compute \mathbf{P}_{i+1} and the amount of memory needed to store $\bar{\mathbf{P}}_i$ increases linearly with i and this *long recurrence* or *long recursion* is sufficient to make the algorithm as described unworkable for large problems. There are two ways of overcoming this. If \mathbf{G} and \mathbf{K} are both symmetric then, as we shall see in Chapter 3, it is necessary to store only one or two terms in the sum in equation (2.1) when computing \mathbf{P}_{i+1} (this is also true, very exceptionally, for some \mathbf{G} and \mathbf{K} nonsymmetric) and this *short recurrence* is enough to make the algorithm viable. Alternatively the long recurrence may be either *truncated* or *restarted*, and these techniques are discussed in the final section of this chapter.

2.6. FOM, GMRes and MinRes

FOM [6], [217], and GMRes [219] both use the same set of orthogonal vectors $\{\mathbf{p}_j\}$ generated by the OAA with $\mathbf{B} = \mathbf{A}$ and $\mathbf{p}_1 = \mathbf{r}_1 / \|\mathbf{r}_1\|$ as the basis of their operations, but in FOM they are used to satisfy the Ritz-Galerkin condition (equation (1.11)) while in GMRes they are used to minimize the residual norm $\|\mathbf{r}\|$. Since the structure of FOM and GMRes differs from that of GODir and GOMin we do not derive their generalised versions as we did for the other two algorithms. For FOM and GMRes the displacement vectors \mathbf{p}_j are orthogonal and we cannot compute \mathbf{x}_{i+1} recursively from \mathbf{x}_i by a variant of Step 2(b) of Algorithm GODir/GOMin (see page 32). It is necessary to compute $\bar{\mathbf{P}}_i$ using Arnoldi and then calculate \mathbf{x}_{i+1} using a different technique to that employed in GODir and GOMin.

Let then $\bar{\mathbf{P}}_i$ be computed by the OAA with $\mathbf{B} = \mathbf{A}$ and $\mathbf{p}_1 = \mathbf{r}_1 / \|\mathbf{r}_1\|$, and let

$$\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \bar{\mathbf{P}}_i \mathbf{z}, \quad (2.24)$$

where \mathbf{z} is a vector of independent variables to be determined. Then if $\mathbf{r}(\mathbf{z}) \equiv \mathbf{Ax}(\mathbf{z}) - \mathbf{b}$,

$$\mathbf{r}(\mathbf{z}) \equiv \mathbf{r}_1 + \mathbf{A} \bar{\mathbf{P}}_i \mathbf{z} \quad (2.25)$$

or, from equation (2.10) with $\mathbf{B} = \mathbf{A}$,

$$\mathbf{r}(\mathbf{z}) \equiv \mathbf{r}_1 + \bar{\mathbf{P}}_{i+1} \tilde{\mathbf{H}}_{i+1} \mathbf{z}.$$

Now since \mathbf{p}_1 is set equal to $\mathbf{r}_1 / \|\mathbf{r}_1\|$ we have $\mathbf{r}_1 = \bar{\mathbf{P}}_{i+1} \mathbf{e}_1 \|\mathbf{r}_1\|$ where $\mathbf{e}_1 \in \mathbb{R}^{i+1}$, and substituting this in the previous equation gives

$$\mathbf{r}(\mathbf{z}) \equiv \bar{\mathbf{P}}_{i+1} \left(\tilde{\mathbf{H}}_{i+1} \mathbf{z} + \mathbf{e}_1 \|\mathbf{r}_1\| \right). \quad (2.26)$$

Since vector sequences are being generated the last row of $\tilde{\mathbf{H}}_{i+1}$ may, from equation (2.14), be expressed as $h_{i+1,i}\mathbf{e}_i^T$ where $h_{i+1,i}$ is the scalar equivalent of $\mathbf{H}_{i+1,i}$. Equation (2.26) may then be written

$$\mathbf{r}(\mathbf{z}) \equiv \overline{\mathbf{P}}_i (\overline{\mathbf{H}}_i \mathbf{z} + \mathbf{e}_1 \|\mathbf{r}_1\|) + \mathbf{p}_{i+1} h_{i+1,i} \mathbf{e}_i^T \mathbf{z} \quad (2.27)$$

where now $\mathbf{e}_1 \in \mathbb{R}^i$. For FOM, $\mathbf{r}(\mathbf{z})$ is required to satisfy the Ritz-Galerkin condition $\overline{\mathbf{P}}_i^T \mathbf{r}(\mathbf{z}) = \mathbf{0}$ and since the columns of $\overline{\mathbf{P}}_i$ are orthonormal by construction it follows from equation (2.27) that this condition is satisfied if

$$\overline{\mathbf{H}}_i \mathbf{z} + \mathbf{e}_1 \|\mathbf{r}_1\| = \mathbf{0}. \quad (2.28)$$

Since $\overline{\mathbf{H}}_i$ is upper Hessenberg, equation (2.28) may be solved simply for \mathbf{z} by premultiplying by the orthogonal matrix \mathbf{Q}_i , where \mathbf{Q}_i is chosen so that

$$\mathbf{Q}_i \overline{\mathbf{H}}_i = \overline{\mathbf{U}}_i \quad (2.29)$$

and $\overline{\mathbf{U}}_i$ is upper triangular (see Appendix A for further details).

Note: the matrices \mathbf{Q}_i of this section and the appendix should *not* be confused with the projection matrices \mathbf{Q}_i defined by equations (1.44), (1.46) and (2.1)).

Thus $\mathbf{z} = -\overline{\mathbf{U}}_i^{-1} \mathbf{Q}_i \mathbf{e}_1 \|\mathbf{r}_1\|$ so that, from equation (2.24),

$$\mathbf{x}_{i+1}^{FOM} = \mathbf{x}_1 - \overline{\mathbf{P}}_i \overline{\mathbf{U}}_i^{-1} \mathbf{Q}_i \mathbf{e}_1 \|\mathbf{r}_1\| \quad (2.30)$$

and, from equations (2.27) and (2.28),

$$\mathbf{r}_{i+1}^{FOM} = -\mathbf{p}_{i+1} h_{i+1,i} \mathbf{e}_i^T \overline{\mathbf{U}}_i^{-1} \mathbf{Q}_i \mathbf{e}_1 \|\mathbf{r}_1\|.$$

If $\overline{\mathbf{U}}_i = [\overline{u}_{jk}]$ then, since $\overline{\mathbf{U}}_i$ is upper triangular, $\mathbf{e}_i^T \overline{\mathbf{U}}_i^{-1} = \overline{u}_{ii}^{-1} \mathbf{e}_i^T$ so that

$$\mathbf{r}_{i+1}^{FOM} = -\mathbf{p}_{i+1} h_{i+1,i} \overline{u}_{ii}^{-1} \mathbf{e}_i^T \mathbf{Q}_i \mathbf{e}_1 \|\mathbf{r}_1\|$$

and, since $\|\mathbf{p}_{i+1}\| = 1$,

$$\|\mathbf{r}_{i+1}^{FOM}\| = |h_{i+1,i} \overline{u}_{ii}^{-1}| |\mathbf{e}_i^T \mathbf{Q}_i \mathbf{e}_1| \|\mathbf{r}_1\|. \quad (2.31)$$

The importance of equation (2.31) is that since the calculation of \mathbf{Q}_i and $\overline{\mathbf{U}}_i$ can be integrated into the GAA, it is possible to compute $\|\mathbf{r}_{i+1}^{FOM}\|$ directly during that phase of the algorithm without computing either \mathbf{x}_{i+1} or \mathbf{r}_{i+1} (see Appendix A). Then, when a suitable value of $\|\mathbf{r}_{i+1}^{FOM}\|$ has been obtained, \mathbf{x}_{i+1} can be computed and the process may, if necessary, be restarted with the new \mathbf{x}_1 being set to the old \mathbf{x}_{i+1} . Alternatively, if the process has to be terminated for practical reasons (shortage of memory, etc.), it could be terminated with the best \mathbf{x}_j to date if, for some $j \leq i$, $\|\mathbf{r}_j^{FOM}\| < \|\mathbf{r}_{i+1}^{FOM}\|$.

Another advantage of this way of proceeding is that it is simple to monitor $\overline{\mathbf{H}}_i$ for singularity or near-singularity. If \mathbf{A} is not positive real, $\overline{\mathbf{H}}_i$ may be singular and if singularity does occur then \overline{u}_{ii} is zero. In this case it is not possible to compute \mathbf{x}_{i+1}^{FOM} . If $|\overline{u}_{ii}|$ is small, it is possible that $\|\mathbf{r}_{i+1}^{FOM}\| > \|\mathbf{r}_1\|$ so that an increase of

residual norm has occurred. For all these reasons FOM is not normally used in practice, its interest being in its relationship with GMRes.

To analyse GMRes we start with equation (2.26). In this case \mathbf{z} is chosen not to satisfy a Ritz-Galerkin condition but to minimise $\|\mathbf{r}(\mathbf{z})\|$, and since the columns of $\tilde{\mathbf{H}}_{i+1}$ are orthonormal by construction it follows from equation (2.26) not only that $\|\mathbf{r}(\mathbf{z})\| \equiv \left\| \left(\tilde{\mathbf{H}}_{i+1}\mathbf{z} + \mathbf{e}_1 \|\mathbf{r}_1\| \right) \right\|$ but also that

$$\|\mathbf{r}(\mathbf{z})\| \equiv \left\| \mathbf{Q} \left(\tilde{\mathbf{H}}_{i+1}\mathbf{z} + \mathbf{e}_1 \|\mathbf{r}_1\| \right) \right\| \quad (2.32)$$

for any orthogonal matrix \mathbf{Q} . Since \mathbf{Q} is arbitrary, let it be equal to \mathbf{Q}_{i+1} where

$$\mathbf{Q}_{i+1}\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \tilde{\mathbf{U}}_i \\ \mathbf{0}^T \end{bmatrix} \quad (2.33)$$

and $\tilde{\mathbf{U}}_i$ is upper triangular (see Appendix A for full details of this transformation). Making the appropriate substitutions in equation (2.32) then gives, if we define $\tilde{\mathbf{c}}_i$ and $\tilde{\gamma}_{i+1}$ by $\begin{bmatrix} \tilde{\mathbf{c}}_i \\ \tilde{\gamma}_{i+1} \end{bmatrix} = \mathbf{Q}_{i+1}\mathbf{e}_1$,

$$\|\mathbf{r}(\mathbf{z})\| \equiv \left\| \begin{bmatrix} \tilde{\mathbf{U}}_i \\ \mathbf{0}^T \end{bmatrix} \mathbf{z} + \begin{bmatrix} \tilde{\mathbf{c}}_i \\ \tilde{\gamma}_{i+1} \end{bmatrix} \|\mathbf{r}_1\| \right\|$$

so that

$$\|\mathbf{r}(\mathbf{z})\|^2 \equiv \left\| \left(\tilde{\mathbf{U}}_i\mathbf{z} + \tilde{\mathbf{c}}_i \|\mathbf{r}_1\| \right) \right\|^2 + (\tilde{\gamma}_{i+1} \|\mathbf{r}_1\|)^2. \quad (2.34)$$

Since $\tilde{\mathbf{U}}_i$ is nonsingular it follows immediately that $\|\mathbf{r}(\mathbf{z})\|$ is minimized if

$$\mathbf{z} = -\tilde{\mathbf{U}}_i^{-1}\tilde{\mathbf{c}}_i \|\mathbf{r}_1\|.$$

With this choice of \mathbf{z} we then have

$$\|\mathbf{r}_{i+1}^{GMRES}\| = |\tilde{\gamma}_{i+1}| \|\mathbf{r}_1\| \quad (2.35)$$

and, from equation (2.24),

$$\mathbf{x}_{i+1} = \mathbf{x}_1 - \tilde{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1} \tilde{\mathbf{c}}_i \|\mathbf{r}_1\|. \quad (2.36)$$

Note that $\tilde{\gamma}_{i+1}$ is the bottom left-hand corner element of \mathbf{Q}_{i+1} and may be expressed as the product of the sines computed by the Givens transformations (see Appendix A).

To see how we actually compute \mathbf{x}_{i+1} from equation (2.36) we note that this equation is virtually identical to equation (6.5) so that, *mutatis mutandis*, equations (6.6) - (6.13) apply to GMRes and GMRes(m). Only the matrix \mathbf{Z}_i and vector \mathbf{y}_i need be stored. The full algorithmic details may be found in Appendix F.

We now examine the principal changes to GMRes that ensue if \mathbf{A} is symmetric. Even if \mathbf{A} is not symmetric, GMRes generates a sequence of orthonormal vectors

$\{\mathbf{p}_j\}$ so that equation (2.17) is satisfied with $\mathbf{B} = \mathbf{A}$, and if \mathbf{A} is symmetric it follows that $\tilde{\mathbf{H}}_s$ is tridiagonal since it is both upper Hessenberg and symmetric. The matrices $\tilde{\mathbf{H}}_i$, $1 \leq i \leq s$, are principal submatrices of $\tilde{\mathbf{H}}_s$ so these too must be tridiagonal. If they are then transformed to upper triangular form by a sequence of plane rotations as described in Appendix A, below, they remain tridiagonal. It follows from the discussion in Chapter 6 that the successive approximations $\{\mathbf{x}_i\}$ to the solution of the equations $\mathbf{Ax} = \mathbf{b}$ and the corresponding residuals $\{\mathbf{r}_i\}$ may, unlike GMRes, be generated recursively using a comparatively small amount of storage. It is thus not necessary to either truncate or restart the iteration. These ideas form the foundation of the algorithm MinRes [202] which is itself a precursor of GMRes.

We conclude this section with a discussion on the implementation of GMRes. GMRes is implemented by putting \mathbf{p}_1 equal to the normalised \mathbf{r}_1 , where \mathbf{r}_1 is the residual corresponding to the arbitrary initial approximate solution \mathbf{x}_1 of $\mathbf{Ax} = \mathbf{b}$. The successive vectors \mathbf{p}_i , $i = 2, 3, \dots$, are then computed by the original Arnoldi algorithm (OAA page 28) where the matrix \mathbf{Q}_i is not stored as such but as the matrix $\bar{\mathbf{P}}_i$ to conserve space. Step 2(c) of the OAA is then effected either by the Gram-Schmidt algorithm (for which \mathbf{Q}_i has essentially the form of equation (1.46)) or by the modified Gram-Schmidt algorithm (where \mathbf{Q}_i has the form of equation (1.44)). The latter is preferable since it has much better numerical properties (see [33]). During this calculation the last column of the upper Hessenberg matrix $\tilde{\mathbf{H}}_{i+1}$ is computed. Since we are only considering the vector form of the algorithm (so that $\mathbf{P}_i = \mathbf{p}_i$ and $\mathbf{Y}_i = \mathbf{y}_i$) it follows from equation (2.9) and the orthonormality of the columns of $\bar{\mathbf{P}}_i$ that premultiplying equation (2.7) with $j = i$ by $\bar{\mathbf{P}}_i^T$ yields

$$\mathbf{y}_i = [h_{1i} \ h_{2i} \ \dots \ h_{ii}]^T = \bar{\mathbf{P}}_i^T \mathbf{B} \mathbf{p}_i.$$

Thus \mathbf{y}_i may be readily computed and equation (2.7) with $j = i$ then gives

$$\mathbf{p}_{i+1} = (\mathbf{B} \mathbf{p}_i - \bar{\mathbf{P}}_i \mathbf{y}_i) / h_{i+1,i}$$

where $h_{i+1,i}$ is chosen so that $\|\mathbf{p}_{i+1}\| = 1$. The last column of $\tilde{\mathbf{H}}_{i+1}$ is then, from equation (2.12), seen to be equal to $[\mathbf{y}_i^T \ h_{i+1,i}]^T$. These calculations represent the use of the “classical” Gram-Schmidt algorithm but can easily be restructured if the modified Gram-Schmidt algorithm is to be implemented.

Now the next stage of the algorithm is the conversion of $\tilde{\mathbf{H}}_{i+1}$ to upper triangular form by the orthogonal transformations described in Appendix A. Since $\tilde{\mathbf{H}}_i$ will already have been so converted it suffices first to apply all the previous rotations to $[\mathbf{y}_i^T \ h_{i+1,i}]^T$ and then determine and apply the new one. The latter involves the matrix $\mathbf{S}_i = \text{diag}(\mathbf{I}_{i-1}, \mathbf{P}_i)$ (see equation A.3) where $\mathbf{S}_i \in \mathbb{R}^{(i+1) \times (i+1)}$ and

$$\mathbf{P}_i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \\ \sin \theta_i & \cos \theta_i \end{bmatrix}.$$

Also, at this stage, the vector $\tilde{\mathbf{c}}_i$ of equation (2.34) is computed, replacing $\tilde{\mathbf{c}}_{i-1}$. The details of both these calculations are given in Appendix A.

We will now have computed the matrix $\bar{\mathbf{P}}_{i+1}$ of equation (2.10), and the matrix $\tilde{\mathbf{U}}_i$ and vector $\tilde{\mathbf{c}}_i$ of equation (2.34). We will not have computed any approximate solution \mathbf{x}_i nor its corresponding residual. Neither will we have computed, explicitly, any orthogonal matrix \mathbf{Q}_j although we will have stored the scalars $s_j = \sin \theta_j$ and $c_j = \cos \theta_j$, $1 \leq j \leq i$, from which these matrices can be recovered. Since no approximate solutions or residuals have been computed it may be thought that we do not know the value of $\|\mathbf{r}\|$ after i steps but the putative value of this may be simply computed from equation (2.35) which, from equation (A.7), becomes

$$\|\mathbf{r}_{i+1}^{GMR}\| = \left(\prod_{j=1}^i |s_j| \right) \|\mathbf{r}_1\|$$

where the scalars s_j are simply the values of $\sin \theta_j$ that have already been computed and stored. Thus when the product of sines becomes sufficiently small this phase of the algorithm can be terminated. The vector \mathbf{z}_i is then computed by solving $\tilde{\mathbf{U}}_i \mathbf{z}_i = -\tilde{\mathbf{c}}_i \|\mathbf{r}_1\|$ and since $\tilde{\mathbf{U}}_i$ is upper triangular this is comparatively straightforward. Then, from equation (2.24), \mathbf{x}_{i+1} is computed by

$$\mathbf{x}_{i+1} = \mathbf{x}_1 + \bar{\mathbf{P}}_i \mathbf{z}_i.$$

This description of the implementation of GMRes was prompted by its practical importance. It is one of the more popular algorithms for solving $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is non-symmetric and as such justifies the inclusion of the above description. FOM, on the other hand, is comparatively rarely used as a practical problem-solver and its interest lies in its theoretical relationship with GMRes. Its residual norms do not converge monotonically as do those of GMRes and can increase abruptly. If graphs are plotted of $\|\mathbf{r}_i^{FOM}\|$ versus i these abrupt increases show up as “peaks” on the graph, while similar graphs of $\|\mathbf{r}_i^{GMR}\|$ exhibit “plateaus” where $\|\mathbf{r}_i^{GMR}\|$ remains virtually constant for several iterations. Moreover there is seen to be a correlation between these peaks and plateaus, and this aspect of the relative behaviour of these two algorithms has been investigated in [42] and [80] and is based on the observation that equations (2.31) and (2.35) yield

$$\|\mathbf{r}_{i+1}^{FOM}\| = |h_{i+1,i} \bar{u}_{ii}^{-1}| \|\mathbf{r}_i^{GMR}\|.$$

It is even possible that FOM would compute an infinite residual (if $\bar{u}_{ii} = 0$ for some i) but this, from Lemma 10 and equation (2.29), could only happen if \mathbf{A} were not positive real. The norms computed by GMRes, on the other hand, always decrease monotonically though not necessarily strictly.

2.7. Practical considerations

The practical problems that arise with OrthoDir as i increases when n is large may be tackled in one of the following two ways.

- (1) *Restarting*. In this variation the calculation is begun afresh after m steps, where m is some pre-assigned positive integer. All the stored matrices are

stripped away and the process is restarted with the new \mathbf{x}_1 being set equal to the old \mathbf{x}_{m+1} . Intuitively, though, it would appear to be undesirable to jettison the results of possibly valuable calculation, and this leads to the second idea which is:

- (2) *Truncating.* In this version, if $i \geq k$ for some predetermined positive integer k , equation (2.20) is replaced by

$$\mathbf{p}_{i+1} = \mathbf{A}\mathbf{p}_i - \sum_{i+1-k}^i \mathbf{p}_j \left(\frac{\mathbf{p}_j^T \mathbf{A}^T \mathbf{A}^2 \mathbf{p}_i}{\mathbf{p}_j^T \mathbf{A}^T \mathbf{A} \mathbf{p}_j} \right)$$

so that the sum is over the last k terms of the sum in equation (2.20). Thus both the storage and the work per step are bounded, but at the cost of exact termination in both cases. The truncated version of GCR is known as OrthoMin and was introduced by Vinsome [248].

These problems apply equally to GCR, FOM and GMRes, and for each of these methods restarted and truncated versions have been proposed [101] [219] [248]. Probably the most popular of these variations is GMRes restarted after a fixed number, m say, steps ($\text{GMRes}(m)$) where m is arbitrary and is usually chosen to be between 10 and 50. However despite its popularity, this version of GMRes can sometimes fail to converge. We have already noted that the residual norms of the full GMRes can form plateaus where, for several iterations, no change is apparent and it was shown by Greenbaum *et al* [135] that in the most extreme case one plateau could last for $(n - 1)$ iterations with the algorithm swooping in on the solution at the last step. This implies that for some problems that are particularly plateau-prone the idea of re-starting GMRes after m iterations, $\text{GMRes}(m)$, is a non-starter since it is possible that no progress at all will be made during each sequence of m steps. To guarantee the convergence of $\text{GMRes}(m)$ it is necessary to impose a further condition upon \mathbf{A} and it will come as no surprise that this condition is positive reality (see [103]) since this is the condition that guarantees that GOMin will not stagnate.

Surprisingly, despite the extra complexity of FOM and GMRes since they both require orthogonal transformations, they both need less work per step than OrthoDir and the GCR algorithm. The reason for this is due to what seems to be an innate weakness of minimum residual algorithms, a weakness that appears to have been first noted by Stoer [226]. If we compare equations (2.20) and (2.23) with the OAA (on which both FOM and GMRes are based) we see that whereas FOM and GMRes only need the calculation of $\mathbf{A}\mathbf{p}_i$ at each step, OrthoDir and GCR require the calculation of both $\mathbf{A}\mathbf{p}_i$ and either $\mathbf{A}^2\mathbf{p}_i$ (OrthoDir) or $\mathbf{A}\mathbf{r}_{i+1}$ (GCR). This extra matrix-vector multiplication seems to be a feature of *all* algorithms of this type that do not use orthogonal transformations, e.g. CR, MCR, and BiCR, but can be avoided by generating recursively the sequence of auxiliary vectors $\{\mathbf{q}_i\} = \{\mathbf{A}\mathbf{p}_i\}$ or some equivalent. Thus the sequence $\{\mathbf{p}_i\}$ of OrthoDir could, from equation (2.20), be computed by first computing $\mathbf{q}_1 = \mathbf{A}\mathbf{p}_1$ and then, for $i \geq 1$, computing

$$\mathbf{q}_{i+1} = \mathbf{A}\mathbf{q}_i - \sum_{j=1}^i \mathbf{q}_j \left(\frac{\mathbf{q}_j^T \mathbf{A} \mathbf{q}_j}{\mathbf{q}_j^T \mathbf{q}_j} \right)$$

and

$$\mathbf{p}_{i+1} = \mathbf{q}_i - \sum_{j=1}^i \mathbf{p}_j \left(\frac{\mathbf{q}_j^T \mathbf{A} \mathbf{q}_j}{\mathbf{q}_j^T \mathbf{q}_j} \right).$$

This device is used in CR, MCR and BiCR but since for these algorithms the recurrences are short, requires the storage of at most one or two additional vectors. In fact for the BiCR algorithms it is not even necessary to compute any vectors of the original sequences $\{\mathbf{v}_i\}$ except the first (see Chapter 3) since the shadow solutions (of $\mathbf{A}^T \mathbf{z} = \mathbf{c}$) are generally not required. However for OrthoDir and GCR the calculation of these auxiliary vectors roughly doubles the storage requirements and as these are excessive anyway this extra imposition makes this version of the algorithms uncompetitive. The only circumstances in which they might be able to challenge GMRes would be if both they and GMRes were to be restarted after only a few steps, and even then any improvement in the relative performance of OrthoDir and GCR is unlikely to be more than marginal.

The above analysis only concerns the relative computational labour of a single step so if it were the case that OrthoDir and GCR required fewer than half the steps needed by GMRes then they could still end up as the overall winners. Unfortunately it can be shown that in exact arithmetic and starting with the same initial approximation, the sequences of approximate solutions generated by OrthoDir, GCR and GMRes are *identical* (see Chapter 4, Equivalent algorithms), and this is sufficient to eliminate OrthoDir and GCR as practical algorithms. However, in their generalised forms of GODir and GOMin, if both \mathbf{G} and \mathbf{K} are symmetric, they undergo a transformation that makes them the basis of all the currently-popular CG-type algorithms. See Chapter 3 for further details.

We are left, therefore, with GMRes which despite its weaknesses is seen to be a reasonably effective and robust algorithm for solving $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is non-symmetric. In its restarted form it is one of the more popular algorithms for solving this problem even though in this form its convergence is not guaranteed for a general matrix \mathbf{A} . Its popularity has stimulated a great deal of recent research aimed at establishing its fundamental properties and determining its rate of convergence. Much of this analysis is based on the Krylov properties of the algorithm and we discuss some of these aspects in Chapter 4 (More on GMRes, page 91).

This page intentionally left blank

Chapter 3

The short recurrences

We now consider the simplifications that occur to both GODir and GOMin when \mathbf{K} is symmetric, and examine the various algorithms given by particular choices of \mathbf{G} and \mathbf{K} . We shall see that if \mathbf{K} is symmetric then both GODir and GOMin reduce to particular versions of the block CG (BICG) algorithm with the expressions for the projection matrices \mathbf{Q}_i being substantially simplified. Although this has certain theoretical implications its principal impact is practical. The simplification means that the storage and time required by each iteration of the algorithm does not increase with the number of iterations as it does for the inexact methods but remains constant, making it possible in principle for the exact solution of $\mathbf{G}\mathbf{X} = \mathbf{H}$ to be found for a far greater range of problems than is the case for the methods of Chapter 2. We shall also see that certain well-known algorithms, for example CG and BiCG, may be regarded simply as particular cases of the BICG algorithm. Others, e.g. QMR and LSQR, use Arnoldi's algorithm or its generalisation to compute the sequence of matrices $\{\mathbf{P}_i\}$ but since these matrices are not \mathbf{A} or $\mathbf{A}^T\mathbf{A}$ conjugate are forced to employ other devices to compute $\{\mathbf{X}_i\}$. This use of GODir and GOMin enables us, comparatively simply, to compare and contrast various apparently quite distinct algorithms and provides a unifying theory for a wide range of individual methods.

3.1. The block-CG algorithm (BICG)

In this section we derive the particular forms of GODir and GOMin when both \mathbf{G} and \mathbf{K} are symmetric and by so doing obtain two versions of the BICG algorithm. These two algorithms may be used to solve $\mathbf{G}\mathbf{X} = \mathbf{H}$ where $\mathbf{H}, \mathbf{X} \in \mathbb{R}^{n \times s}$ and \mathbf{G} has no special properties other than symmetry and nonsingularity. There are various reasons for wanting to solve block systems such as these:

- Some problems occur naturally in this form or can be transformed into it by simple manipulation
- The desire to exploit parallelism

- The desire to solve the vector equation $\mathbf{Ax} = \mathbf{b}$ by solving the compound system of equations (1.38).

We note that if $r = 1$ the two BiCG algorithms reduce to simple algorithms of CG-type.

Theorem 14. Let the sequence $\{\mathbf{P}_j\}$ be computed by GODir with \mathbf{K} symmetric. Then Step 2(e) of GODir becomes

$$\mathbf{P}_{i+1} = (\mathbf{I} - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{G} - \mathbf{P}_{i-1} \mathbf{D}_{i-1}^{-1} \mathbf{P}_{i-1}^T \mathbf{G}) \mathbf{KGP}_i. \quad (3.1)$$

Proof. Since, for both GODir and GOMin, $\mathbf{P}_j = \mathbf{Q}_{j-1}^T \mathbf{W}_j$, premultiplication of equation (1.51) by \mathbf{W}_j^T gives

$$\mathbf{P}_j^T \mathbf{GP}_i = \mathbf{W}_j^T \mathbf{GP}_i, \quad 1 \leq j \leq i. \quad (3.2)$$

Now for GODir, $\mathbf{W}_j = \mathbf{KGP}_{j-1}$ and since \mathbf{K} is symmetric

$$\mathbf{P}_j^T \mathbf{GP}_i = \mathbf{P}_{j-1}^T \mathbf{GKGP}_i, \quad 2 \leq j \leq i. \quad (3.3)$$

Hence, since the matrices \mathbf{P}_j are mutually conjugate with respect to \mathbf{G} ,

$$\mathbf{P}_j^T \mathbf{GKGP}_i = \mathbf{O}, \quad 1 \leq j \leq i-2. \quad (3.4)$$

Now for GODir, $\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{KGP}_i$ and the theorem follows immediately from this equation, the previous equation and equation (2.1). ■

This theorem, or more particularly equation (3.1), forms the basis of the Lanczos version of the block-CG algorithm of O'Leary [193] which is the algorithm from which BiCGL, MRZ and the original version of QMR may all be derived.

Theorem 15. Let the sequences $\{\mathbf{F}_j\}$ and $\{\mathbf{P}_j\}$ be computed by GOMin with \mathbf{K} symmetric. If the matrices $\mathbf{F}_j^T \mathbf{P}_j$, $j = 1, 2, \dots, i-1$, are nonsingular then Step 2(e) of GOMin becomes

$$\mathbf{P}_{i+1} = (\mathbf{I} - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{G}) \mathbf{KF}_{i+1} \quad (3.5)$$

Proof. Since, for both GODir (in effect, the GAA) and GOMin, $\mathbf{P}_j^T \mathbf{F}_{i+1} = \mathbf{O}$, $1 \leq j \leq i$, it follows from equation (2.1) that $\mathbf{Q}_{j-1} \mathbf{F}_{i+1} = \mathbf{F}_{i+1}$, $1 \leq j \leq i+1$, and pre-multiplying this equation by \mathbf{W}_j^T gives, since again for both GODir and GOMin $\mathbf{P}_j = \mathbf{Q}_{j-1}^T \mathbf{W}_j$,

$$\mathbf{P}_j^T \mathbf{F}_{i+1} = \mathbf{W}_j^T \mathbf{F}_{i+1}, \quad 1 \leq j \leq i+1. \quad (3.6)$$

Now for GOMin $\mathbf{W}_j = \mathbf{KF}_j$ and since \mathbf{K} is symmetric,

$$\mathbf{P}_j^T \mathbf{F}_{i+1} = \mathbf{F}_j^T \mathbf{KF}_{i+1}, \quad 1 \leq j \leq i+1, \quad (3.7)$$

from which, immediately,

$$\mathbf{F}_j^T \mathbf{KF}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i. \quad (3.8)$$

Now equation (1.37) may be written

$$\mathbf{F}_{j+1} = \mathbf{F}_j - \mathbf{G}\mathbf{P}_j\mathbf{D}_j^{-1}\mathbf{P}_j^T\mathbf{F}_j$$

and pre-multiplying this by $\mathbf{F}_{i+1}^T\mathbf{K}$ and transposing gives, from equation (3.8) and the symmetry of \mathbf{K} ,

$$\mathbf{F}_j^T\mathbf{P}_j\mathbf{D}_j^{-1}\mathbf{P}_j^T\mathbf{G}\mathbf{K}\mathbf{F}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i-1.$$

But, by hypothesis, $\mathbf{F}_j^T\mathbf{P}_j$ is nonsingular and this implies that

$$\mathbf{P}_j^T\mathbf{G}\mathbf{K}\mathbf{F}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i-1 \tag{3.9}$$

and since, for GOMin, $\mathbf{P}_{i+1} = \mathbf{Q}_i^T\mathbf{K}\mathbf{F}_{i+1}$ the theorem follows immediately from this equation and equations (2.1) and (3.9). ■

This algorithm, or more particularly equation (3.5), forms the basis of the HS version of the block-CG algorithm [193] which is the algorithm from which BiCG, HG and BiCR may all be derived. It therefore represents one of the more important algorithms discussed here, the more so since its vector form also includes CG and CR as special cases.

3.2. Alternative forms

Although the expressions for \mathbf{P}_{i+1} given by equations (3.1) and (3.5) are closest to the original projectors, alternative forms have been proposed for practical implementation that enable matrices already computed to be recycled and used again in a different context in order to avoid unnecessary calculation. Other practical problems that may arise relate to the possibility of over- or under-flow (particularly in the case of the Lanczos versions) and the maintenance of the linear independence of the columns of \mathbf{P}_i for the general block methods discussed in Chapter 8. The latter is necessary since for the block methods the matrices \mathbf{P}_i are given by $\mathbf{P}_i = \mathbf{Q}_{i-1}^T(\mathbf{KG})^{i-1}\mathbf{P}_1$ (see Chapter 4, below) and each column of $(\mathbf{KG})^{i-1}\mathbf{P}_1$ tends to a multiple of the eigenvector corresponding to the dominant eigenvalue of \mathbf{KG} as i increases. Thus without some form of normalisation the matrices $\mathbf{P}_i^T\mathbf{P}_i$ become increasingly ill-conditioned with increasing i and this can give rise to numerical difficulties.

The matrices \mathbf{P}_i may, provided they have full column rank, be “scaled” by post-multiplying them by suitable nonsingular matrices \mathbf{B}_i so that, e.g., $\mathbf{P}_i^T\mathbf{P}_i = \mathbf{I}$ or, if \mathbf{G} is positive definite, $\mathbf{P}_i^T\mathbf{G}\mathbf{P}_i = \mathbf{I}$ (note: in the context of the GAA (see page 29) we have denoted the matrices \mathbf{B}_i by $\mathbf{H}_{i,i-1}^{-1}$). Matrices so scaled define the same vector spaces as their unscaled counterparts so the conjugacy properties remain unaltered. In equations (1.36), (1.37), (3.1) and (3.5), substitution of the scaled versions of \mathbf{P}_i for the unscaled ones leaves the equations unchanged as the scaling matrices, being nonsingular, simply cancel out. With the alternative forms this is no longer the case so in deriving them we have to take into account the effects of scaling, although we can assume that the four equations referred to above are also

valid when the scaled versions of the matrices \mathbf{P}_i are substituted for the unscaled ones.

Consider first the Lanczos (GODir) version for which \mathbf{W}_i would be given by $\mathbf{KGP}_{i-1}\mathbf{B}_i$ for some nonsingular scaling matrix \mathbf{B}_i . Since equation (3.2) is valid for a general \mathbf{W}_i , substitution of the above expression in this equation gives

$$\mathbf{P}_i^T \mathbf{GP}_i = \mathbf{B}_i^T \mathbf{P}_{i-1}^T \mathbf{GKGP}_i.$$

Thus, if

$$\mathbf{D}_i = \mathbf{P}_i^T \mathbf{GP}_i \quad (3.10)$$

then

$$\mathbf{P}_{i-1}^T \mathbf{GKGP}_i = \mathbf{B}_i^{-T} \mathbf{D}_i,$$

and substituting this expression in equation (3.1) and adding the scaling matrix \mathbf{B}_{i+1} gives

$$\mathbf{P}_{i+1} = [(\mathbf{I} - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{G}) \mathbf{KGP}_i - \mathbf{P}_{i-1} \mathbf{D}_{i-1}^{-1} \mathbf{B}_i^{-T} \mathbf{D}_i] \mathbf{B}_{i+1}. \quad (3.11)$$

The new values of \mathbf{X}_i and \mathbf{F}_i are then given by equation (1.36) and (1.37) which, for ease of reference, we duplicate here:

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i \quad (3.12)$$

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{GP}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i \quad (3.13)$$

and these three equations with, for simplicity, $\mathbf{B}_i = \mathbf{I}$ form the basis of the majority of the particular algorithms of Lanczos type derived below. We note that, in equation (3.11), we have replaced $\mathbf{P}_{i-1}^T \mathbf{GKGP}_i$ (which would not otherwise need to be evaluated) by the previously-computed \mathbf{D}_i , thereby eliminating some unnecessary calculation.

For the HS (GOMin) version of the algorithm, \mathbf{W}_i is defined by $\mathbf{KF}_i \mathbf{B}_i$ for some suitable⁵ nonsingular scaling matrix \mathbf{B}_i and, for convenience, \mathbf{C}_i is defined by

$$\mathbf{C}_i = \mathbf{F}_i^T \mathbf{KF}_i. \quad (3.14)$$

Since equation (3.6) is valid for arbitrary \mathbf{W}_i , substituting $\mathbf{KF}_{i+1} \mathbf{B}_{i+1}$ for \mathbf{W}_{i+1} in this equation gives

$$\mathbf{P}_{i+1}^T \mathbf{F}_{i+1} = \mathbf{B}_{i+1}^T \mathbf{C}_{i+1} \quad (3.15)$$

so that equations (3.12) and (3.13) become

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{B}_i^T \mathbf{C}_i \quad (3.16)$$

and

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{GP}_i \mathbf{D}_i^{-1} \mathbf{B}_i^T \mathbf{C}_i. \quad (3.17)$$

⁵ For a first reading it may be assumed that $\mathbf{B}_i = \mathbf{B}_{i+1} = \mathbf{I}$. See the text for an explanation.

Furthermore, transposing equation (3.17) and postmultiplying by $\mathbf{K}\mathbf{F}_{i+1}$ yields, from equations (3.8) and (3.15),

$$\mathbf{C}_{i+1} = -\mathbf{C}_i \mathbf{B}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{G} \mathbf{K} \mathbf{F}_{i+1}$$

so that equation (3.5), with the addition of its scaling matrix \mathbf{B}_{i+1} , becomes

$$\mathbf{P}_{i+1} = (\mathbf{K}\mathbf{F}_{i+1} + \mathbf{P}_i \mathbf{B}_i^{-1} \mathbf{C}_i^{-1} \mathbf{C}_{i+1}) \mathbf{B}_{i+1}. \quad (3.18)$$

Again we note that we have replaced the calculation of $\mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{G} \mathbf{K} \mathbf{F}_i$ by that of $\mathbf{C}_i^{-1} \mathbf{C}_{i+1}$, where both \mathbf{C}_i and \mathbf{C}_{i+1} are comparatively small matrices whose elements have either already been, or will need to be, computed.

In practice it is normally necessary to scale the Lanczos algorithms in order to avoid either underflow or overflow when generating the sequence $\{\mathbf{P}_i\}$. Although this scaling has no effect on the original projector for computing \mathbf{P}_{i+1} (equation (3.1)) it does complicate the alternative version (equation (3.11)) with the introduction of a \mathbf{B}_i^{-T} . This must be included in any practical implementation of the algorithm (see Appendix F for scaled versions of some of the better-known Lanczos algorithms). For the purposes of this chapter, though, where we are more concerned with the theoretical properties of the algorithms, we omit this scaling in the interests of simplicity.

For the HS versions of the algorithms, even in practice, it is not necessary to scale the matrices \mathbf{P}_{i+1} computed by equation (3.18) as underflow or overflow is not normally a problem with these versions. In the particular algorithms detailed below, scaling is only recommended for the variant of QMR based on coupled two-term recurrences (see page 120) and even here the comparative testing reported in Chapter 10 (below) casts doubts on its necessity. We therefore, with this one exception, obtain the equations for the HS versions by substituting the appropriate values of \mathbf{G} and \mathbf{K} in equations (3.16) - (3.18) with $\mathbf{B}_i = \mathbf{B}_{i+1} = \mathbf{I}$, that is into

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{C}_i, \quad (3.19)$$

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{C}_i \quad (3.20)$$

and

$$\mathbf{P}_{i+1} = \mathbf{K}\mathbf{F}_{i+1} + \mathbf{P}_i \mathbf{C}_i^{-1} \mathbf{C}_{i+1}, \quad (3.21)$$

where \mathbf{D}_i and \mathbf{C}_i are defined by equations (3.10) and (3.14).

3.3. The original Lanczos method

Although this method cannot be used as it stands to solve linear equations as it generates orthogonal as opposed to conjugate vectors, it nevertheless forms the basis of methods like SymmLQ and LSQR. The principal recurrence is obtained from the vector form of equation (3.11) by putting $\mathbf{K} = \mathbf{A}$ and $\mathbf{G} = \mathbf{I}$, where \mathbf{A}

is symmetric, and normalising the vectors \mathbf{p}_i so that $\|\mathbf{p}_i\| = 1$. This implies from equation (3.10) that $\mathbf{D}_i = 1$ so if we define \mathbf{q}_i by

$$\mathbf{q}_i = \mathbf{A}\mathbf{p}_i - \mathbf{p}_i (\mathbf{p}_i^T \mathbf{A}\mathbf{p}_i) - \mathbf{p}_{i-1} \|\mathbf{q}_{i-1}\| \quad (3.22)$$

we have $\mathbf{B}_{i+1} = 1/\|\mathbf{q}_i\|$ and

$$\mathbf{p}_{i+1} = \mathbf{q}_i / \|\mathbf{q}_i\|.$$

Thus the Lanczos algorithm is just the vector form of GODir with $\mathbf{G} = \mathbf{I}$ and $\mathbf{K} = \mathbf{K}^T$ and this in turn is just the vector form of the generalised Arnoldi algorithm with $\mathbf{B} = \mathbf{KG} = \mathbf{K}$. If we put $\mathbf{K} = \mathbf{A}$ ($= \mathbf{A}^T$) we can replace \mathbf{B} by \mathbf{A} in the section on Arnoldi's method and the results are valid for the Lanczos method as defined by equation (3.22). In particular, equation (2.10) becomes $\mathbf{A}\bar{\mathbf{P}}_i = \bar{\mathbf{P}}_{i+1}\bar{\mathbf{H}}_{i+1}$ where $\bar{\mathbf{P}}_i$ (see equation (1.31)) is now defined by

$$\bar{\mathbf{P}}_i = [\mathbf{p}_1 \mathbf{p}_2 \dots \mathbf{p}_i]$$

and since the vectors \mathbf{p}_i are orthonormal, $\bar{\mathbf{P}}_i^T \bar{\mathbf{P}}_i = \mathbf{I}$. Premultiplying the previous equation by $\bar{\mathbf{P}}_i^T$ then gives, from equation (2.14),

$$\bar{\mathbf{P}}_i^T \mathbf{A}\bar{\mathbf{P}}_i = \bar{\mathbf{H}}_i \quad (3.23)$$

and the tridiagonality of $\bar{\mathbf{H}}_i$ may be deduced either from the fact that it is simultaneously symmetric and upper Hessenberg, or directly from equation (3.22).

The main application of the Lanczos method is the determination of eigenvalues. Since the columns of $\bar{\mathbf{P}}_i$ are orthonormal there exist certain relationships between the eigenvalues of \mathbf{A} and those of the successive matrices $\bar{\mathbf{H}}_i$, which are known as *Ritz values* of \mathbf{A} . There has been an enormous amount of work done in this area since the original paper of Lanczos [174] and as this is still continuing those interested are referred to the most recently-published book on numerical linear algebra or, better still, any equivalent work on the determination of eigenvalues.

3.4. Simple and compound algorithms

For the algorithms described in this chapter the values of \mathbf{G} and \mathbf{K} fall into two distinct categories. The first is where \mathbf{G} is one of \mathbf{I} , \mathbf{A} or \mathbf{A}^2 if \mathbf{A} is symmetric or \mathbf{I} or $\mathbf{A}^T\mathbf{A}$ if it is not, with similar choices being made for \mathbf{K} . For these methods there is only one right-hand side and the precise form of the algorithm is obtained by substituting the appropriate values of \mathbf{G} , \mathbf{h} and \mathbf{K} in equation (1.8) and in the vector forms of equations (1.17), (3.1) and (3.5) or their equivalents. We refer to these algorithms as *simple algorithms* and clearly they are just special cases of the BICG algorithm with $r = 1$.

Compound algorithms solve the equations $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{A}^T\mathbf{z} = \mathbf{c}$ by expressing them in the form

$$\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix} \quad (3.24)$$

or, in the case of LSQR,

$$\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix},$$

and solving the system thus obtained by the BiCG algorithm. For the compound algorithms, \mathbf{G} is always 2×2 block diagonal or block skew-diagonal, and if \mathbf{X} is chosen to be block diagonal it follows from the equation $\mathbf{GX} = \mathbf{H}$ and from equation (1.35) that both \mathbf{H} and \mathbf{F} must have the same structure as \mathbf{G} . If, in addition, \mathbf{K} is also either 2×2 block diagonal or block skew-diagonal and \mathbf{X}_1 is block diagonal then simple (but tedious) substitution in equations (1.36), (3.1) and (3.5) or their equivalents leads to the result that, for all i , \mathbf{X}_i is given by

$$\mathbf{X}_i = \begin{bmatrix} \mathbf{x}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{z}_i \end{bmatrix} \quad (3.25)$$

and \mathbf{P}_i is given by

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{u}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_i \end{bmatrix} \text{ or } \begin{bmatrix} \mathbf{0} & \mathbf{u}_i \\ \mathbf{v}_i & \mathbf{0} \end{bmatrix} \quad (3.26)$$

depending on the structure of \mathbf{G} and \mathbf{K} (for some algorithms the form of \mathbf{P}_i even alternates from one iteration to the next). This permits us to express the recursion for \mathbf{X}_i as a recursion for \mathbf{x}_i (the recursion for \mathbf{z}_i is omitted unless the solution of $\mathbf{A}^T \mathbf{z} = \mathbf{c}$ is specifically required) and the recursion for \mathbf{P}_i as recursions for \mathbf{u}_i and \mathbf{v}_i , which is the way these algorithms are normally presented. That we can do this is entirely due to the block structure assigned to the two matrices \mathbf{G} and \mathbf{K} .

Another useful matrix that we should define at this point is the block residual matrix \mathbf{R}_i which is given by

$$\mathbf{R}_i = \begin{bmatrix} \mathbf{0} & \mathbf{s}_i \\ \mathbf{r}_i & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{x}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{z}_i \end{bmatrix} - \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix}. \quad (3.27)$$

The residual vector \mathbf{r}_i could be computed by $\mathbf{r}_i = \mathbf{Ax}_i - \mathbf{b}$ but is normally computed recursively to avoid unnecessary matrix-vector multiplications. The vector of *shadow residuals* \mathbf{s}_i has to be computed recursively as the sequence $\{\mathbf{z}_i\}$ is generally not computed at all. The initial value \mathbf{s}_1 of \mathbf{s}_i is chosen arbitrarily.

All combinations of block diagonal/skew-diagonal are possible when choosing \mathbf{G} and \mathbf{K} except that both cannot be simultaneously diagonal (since in this case the algorithm decouples into two totally independent algorithms), and it follows from this that at least one of \mathbf{G} and \mathbf{K} must be indefinite. This has important practical consequences since the indefiniteness of \mathbf{G} or \mathbf{K} can cause the algorithm to be numerically unsatisfactory. Although a full discussion of this problem is deferred until Chapter 4 it is useful to give a brief description of it now in order to facilitate the descriptions of individual algorithms in the following sections.

If $\mathbf{D}_i = \mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is singular for some value of i then \mathbf{X}_{i+1} can be computed neither by equation (3.12) nor equation (3.16) and the algorithm is said to *crash*. The most common cause of a crash or near-crash (which can cause severe numerical instability

due to the resulting large increase in the residual norms) is the indefiniteness of \mathbf{G} which permits serious breakdown to occur (see Discussion and summary, page 26). It usually manifests itself by the appearance of very small divisors in the expressions used to calculate \mathbf{X}_{i+1} and the problem is normally overcome by the use of a look-ahead version of the algorithm concerned (see Chapter 7, below). A more insidious problem occurs when $\mathbf{C}_i = \mathbf{F}_i^T \mathbf{K} \mathbf{F}_i$ is singular or nearly so for some value of i . In the case of the compound algorithms defined above, near-singularity of \mathbf{C}_i is equivalent to \mathbf{C}_i being effectively null so that \mathbf{X}_{i+1} is practically equal to \mathbf{X}_i . This effect is associated with the indefiniteness of \mathbf{K} , (see Chapter 4, below) and in exact arithmetic differs for the two different versions of the algorithm. In the Lanczos one \mathbf{X}_j , $j = i, i+1, \dots, k$, may be unchanged for some positive integer k but eventually the correct solution will be computed. On the other hand if \mathbf{C}_i becomes null in the HS version the algorithm never recovers (see e.g. [50]). In both cases the algorithm is said to *stagnate*. In practice, of course, the presence of rounding error blurs the clean lines of the pure analysis and surprisingly the HS versions usually perform *better* than their Lanczos counterparts. The reasons for this are not well understood at the time of writing and some recent attempts at explanation are reported in Chapter 9 (below). However neither crashing nor stagnation can occur if both \mathbf{G} and \mathbf{K} are definite and the algorithm only generates vector sequences, but very few algorithms used in practice satisfy these somewhat rigorous criteria. For a fuller discussion of these and related matters, see Chapter 4 (below). In our assessment of individual algorithms we make no distinction between "temporary" and "permanent" stagnation, regarding all algorithms for which \mathbf{K} is indefinite simply as being prone to stagnation.

In the majority of cases described below it is necessary only to substitute the appropriate values of \mathbf{G} , \mathbf{H} and \mathbf{K} into equations (3.10) - (3.13) or equations (3.19) - (3.21) to obtain the required formulæ pertaining to a specific algorithm but it should be pointed out that in some cases, notably the Lanczos version of the Hegedüs algorithm (HGL) in which the form of \mathbf{P}_i (see equation (3.26)) alternates from one iteration to another, a certain amount of care is needed. For other algorithms, in particular the minimum error algorithms, direct substitution of \mathbf{G} and \mathbf{K} in the GAA yields the equations for computing the sequences $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ but it is necessary to use other techniques to compute the sequence of approximate solutions $\{\mathbf{x}_i\}$ since in equations (3.19) - (3.21), \mathbf{F}_i depends on \mathbf{G}^{-1} and is thus unknown. Some of these techniques rely on orthogonal transformations. Orthogonal transformations have outstanding numerical properties but these are not their only useful attributes and they can also be used to alter the fundamental structure of an algorithm. This is how they are used here and our discussion focusses on this aspect of the transformations, accepting their numerical properties as a bonus. Finally we note that the forms of the equations given below are the forms that are required after the first one or two iterations. To avoid repetition, any vectors with negative or zero subscripts that would occur in these iterations should be regarded as null unless otherwise stated.

In the next four sections we derive the specific recursions given by particular choices of \mathbf{G} and \mathbf{K} . The algorithms are divided into four groups: Galerkin algo-

rithms (which roughly correspond to $\mathbf{G} = \mathbf{A}$), minimum residual algorithms (which roughly correspond to $\mathbf{G} = \mathbf{A}^T \mathbf{A}$), minimum error algorithms (which roughly correspond to $\mathbf{G} = \mathbf{I}$) and others which cannot be obtained by direct substitution into equations already derived. This corresponds roughly to the classification of Freund [115]. For the algorithms in the first three sections (Galerkin algorithms, Minimum-residual algorithms and Minimum-error algorithms) it is sufficient in general to substitute the parameters quoted in the relevant subsection into equations (3.10) - (3.13) or (3.19) - (3.21) to obtain the basic equations for the corresponding algorithm. In addition, for the minimum-residual and minimum-error algorithms, one or two auxiliary sequences are also usually computed for the purpose of avoiding unnecessary matrix vector multiplications or for obtaining particular scalars. These recurrences may be simply derived from the basic equations once the latter have been obtained by simple substitution.

For the algorithms of the fourth section (Lanczos-based methods) the block-CG equation (3.11) only generates sequences of (orthogonal or bi-orthogonal) vectors, and since they are orthogonal as opposed to conjugate they cannot be substituted directly in equation (3.12) to give the next approximate solution. The calculation of the approximate solutions are specific to each algorithm. The associated descriptions, therefore, are somewhat more comprehensive than those given in the previous three sections except that of QMR, which is dealt with more fully elsewhere.

3.5. Galerkin algorithms

The Galerkin, properly Ritz-Galerkin, algorithms are characterized by choosing \mathbf{G} to be either \mathbf{A} or $\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}$, and there is only an element of minimisation involved if $\mathbf{G} = \mathbf{A}$ and is positive definite. The alternative choice leads to an indefinite \mathbf{G} regardless of the properties of \mathbf{A} , and for algorithms for which \mathbf{G} is not definite there is always the possibility of breakdown. This may usually be avoided, at some cost, by the use of look-ahead techniques.

3.5.1. The conjugate gradient algorithm (CG)

Type: HS

Parameters:

$$\mathbf{A} = \mathbf{A}^T, \quad \mathbf{G} = \mathbf{A}, \quad \mathbf{h} = \mathbf{b}, \quad \mathbf{K} = \mathbf{I}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{p}_1 = \mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$,

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \right) \quad (3.28)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{A}\mathbf{p}_i \left(\frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \right) \quad (3.29)$$

$$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \mathbf{p}_i \left(\frac{\mathbf{r}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{r}_i} \right) \quad (3.30)$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{A} \mathbf{p}_i = 0, j \neq i$

Termination properties: $\mathbf{p}_j^T \mathbf{r}_i = 0, j < i$

Numerical properties: Can neither crash nor stagnate if \mathbf{A} is positive definite but can crash if \mathbf{A} is indefinite.

Notes: ‘The Original and Best’. Look-ahead versions have been proposed by Luenberger [180] and Fletcher [110] in the case where \mathbf{A} is indefinite but their stability has been questioned [227], [226]. Choices of \mathbf{K} other than the identity give the pre-conditioned CG algorithm (see Chapter 11).

Original reference: [154], 1952.

3.5.2. The biconjugate gradient algorithm (BiCG)

Type: HS

Parameters:

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{O} & \mathbf{I} \\ \mathbf{I} & \mathbf{O} \end{bmatrix}$$

Initial values: $\mathbf{x}_1, \mathbf{s}_1$, arbitrary, $\mathbf{u}_1 = \mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{v}_1 = \mathbf{s}_1$,
Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{s}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \quad (3.31)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{A} \mathbf{u}_i \left(\frac{\mathbf{s}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \quad (3.32)$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{A}^T \mathbf{v}_i \left(\frac{\mathbf{s}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \quad (3.33)$$

$$\mathbf{u}_{i+1} = \mathbf{r}_{i+1} + \mathbf{u}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{s}_i^T \mathbf{r}_i} \right) \quad (3.34)$$

$$\mathbf{v}_{i+1} = \mathbf{s}_{i+1} + \mathbf{v}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{s}_i^T \mathbf{r}_i} \right) \quad (3.35)$$

Conjugacy properties: $\mathbf{v}_j^T \mathbf{A} \mathbf{u}_i = 0, j \neq i$, $\mathbf{s}_j^T \mathbf{r}_i = 0, j \neq i$,

Termination properties: $\mathbf{v}_j^T \mathbf{r}_i = 0, j < i, \mathbf{u}_j^T \mathbf{s}_i = 0, j < i,$

Numerical properties: Can both crash and stagnate for any \mathbf{A} , with the remote possibility of incurable breakdown for certain choices of initial values.

Notes: The vectors $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ are biconjugate with respect to \mathbf{A} (giving the algorithm its name) while the residuals are orthogonal to the shadow residuals. The expression of BiCG in this form, with the above value of \mathbf{K} , was hinted at almost as an afterthought in a paper by Joly [164].

Principal references: [175], 1952 and [110], 1976.

3.5.3. The BiCGL algorithm

Type: Lanczos

Parameters:

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{O} & \mathbf{I} \\ \mathbf{I} & \mathbf{O} \end{bmatrix}$$

Initial values: $\mathbf{x}_1, \mathbf{s}_1$ arbitrary, $\mathbf{u}_1 = \mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}, \mathbf{v}_1 = \mathbf{s}_1$

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right) \quad (3.36)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{Au}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right)$$

$$\mathbf{u}_{i+1} = \mathbf{Au}_i - \mathbf{u}_i \left(\frac{\mathbf{v}_i^T \mathbf{A}^2 \mathbf{u}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right) - \mathbf{u}_{i-1} \left(\frac{\mathbf{v}_i^T \mathbf{Au}_i}{\mathbf{v}_{i-1}^T \mathbf{Au}_{i-1}} \right) \quad (3.37)$$

$$\mathbf{v}_{i+1} = \mathbf{A}^T \mathbf{v}_i - \mathbf{v}_i \left(\frac{\mathbf{v}_i^T \mathbf{A}^2 \mathbf{u}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right) - \mathbf{v}_{i-1} \left(\frac{\mathbf{v}_i^T \mathbf{Au}_i}{\mathbf{v}_{i-1}^T \mathbf{Au}_{i-1}} \right) \quad (3.38)$$

Conjugacy properties: $\mathbf{v}_j^T \mathbf{Au}_i = 0, j \neq i$

Termination properties: $\mathbf{v}_j^T \mathbf{r}_i = 0, j < i$

Numerical properties: Can both crash and stagnate for any \mathbf{A} , with the remote possibility of incurable breakdown for certain choices of initial values.

Notes: The Lanczos version of the BiCG algorithm, with similar properties. Its “look-ahead” form is known as the MRZ algorithm.

Principal references: [175], 1952, [110], 1976 and, for the MRZ variant, [38].

3.5.4. The Hegedüs “Galerkin” algorithm (HGL)

Type: Lanczos

Parameters:

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{bmatrix}$$

Initial values: $\mathbf{x}_1, \mathbf{u}_1$ arbitrary, $\mathbf{v}_1 = \mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$,

Recursions:

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right) \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \mathbf{Au}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right) \\ \mathbf{u}_{i+1} &= \mathbf{A}^T \mathbf{v}_i - \mathbf{u}_i \left(\frac{\mathbf{v}_i^T \mathbf{AA}^T \mathbf{v}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right) - \mathbf{u}_{i-1} \left(\frac{\mathbf{v}_{i-1}^T \mathbf{Au}_i}{\mathbf{v}_{i-1}^T \mathbf{Au}_{i-1}} \right) \\ \mathbf{v}_{i+1} &= \mathbf{Au}_i - \mathbf{v}_i \left(\frac{\mathbf{u}_i^T \mathbf{A}^T \mathbf{Au}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right) - \mathbf{v}_{i-1} \left(\frac{\mathbf{v}_{i-1}^T \mathbf{Au}_i}{\mathbf{v}_{i-1}^T \mathbf{Au}_{i-1}} \right) \end{aligned}$$

Conjugacy properties: $\mathbf{v}_j^T \mathbf{Au}_i = 0, j \neq i$

Termination properties: $\mathbf{v}_j^T \mathbf{r}_i = 0, j < i$

Numerical properties: Can crash for any \mathbf{A} , but with one-step recovery from serious breakdown

Notes: The Lanczos version of HG (see below).

Original references: [149], 1990, [150], 1991 and [151], 1991.

3.5.5. The Hegedüs “Galerkin” algorithm (HG)

Type: HS

Parameters:

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{bmatrix}$$

Initial values: $\mathbf{x}_1, \mathbf{s}_1$ arbitrary, $\mathbf{v}_1 = \mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}, \mathbf{u}_1 = \mathbf{s}_1$

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{Au}_i} \right)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{A}\mathbf{u}_i \left(\frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A}\mathbf{u}_i} \right)$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{A}^T \mathbf{v}_i \left(\frac{\mathbf{s}_i^T \mathbf{s}_i}{\mathbf{v}_i^T \mathbf{A}\mathbf{u}_i} \right)$$

$$\mathbf{u}_{i+1} = \mathbf{s}_{i+1} + \mathbf{u}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{s}_{i+1}}{\mathbf{s}_i^T \mathbf{s}_i} \right)$$

$$\mathbf{v}_{i+1} = \mathbf{r}_{i+1} + \mathbf{v}_i \left(\frac{\mathbf{r}_{i+1}^T \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{r}_i} \right)$$

Conjugacy properties: $\mathbf{v}_j^T \mathbf{A}\mathbf{u}_i = 0, j \neq i, \quad \mathbf{r}_j^T \mathbf{r}_i = \mathbf{s}_j^T \mathbf{s}_i = 0, j \neq i,$

Termination properties: $\mathbf{u}_j^T \mathbf{s}_i = \mathbf{v}_j^T \mathbf{r}_i = 0, j < i.$

Numerical properties: Can crash for any \mathbf{A} , but with one-step recovery from serious breakdown.

Notes: Can be regarded as an alternatively-preconditioned BiCG algorithm with convergence properties similar to those of LSQR (see Chapter 4 below). Deserves to be more widely known.

Original references: [149], 1990, [150], 1991 and [151], 1991.

3.5.6. The Concus-Golub-Widlund algorithm (CGW)

Type: See Notes.

Parameters:

$$\mathbf{G} = \mathbf{A} \neq \mathbf{A}^T, \quad \mathbf{K} = \left(\frac{\mathbf{A} + \mathbf{A}^T}{2} \right)^{-1}, \quad \mathbf{h} = \mathbf{b}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{p}_1 = \mathbf{Kr}_1$

Recursions:

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A}\mathbf{p}_i} \right), \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \mathbf{A}\mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A}\mathbf{p}_i} \right), \end{aligned} \tag{3.39}$$

and

$$\mathbf{p}_{i+1} = \mathbf{Kr}_{i+1} - \mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{A}\mathbf{K}\mathbf{r}_{i+1}}{\mathbf{p}_i^T \mathbf{A}\mathbf{p}_i} \right). \tag{3.40}$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{A}\mathbf{p}_i = 0, j < i, \quad \mathbf{r}_j^T \mathbf{K}\mathbf{r}_i = 0, j \neq i,$

Termination properties: $\mathbf{p}_j^T \mathbf{r}_i = 0, j < i,$

Numerical properties: Can neither crash nor stagnate if \mathbf{A} is positive real.

Notes: This algorithm is really two distinct algorithms, the HS form being due to Concus and Golub and the Lanczos form to Widlund. It solves the equations $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \neq \mathbf{A}^T$. It is the only algorithm considered here for which \mathbf{G} is not symmetric so that the general theory derived above does not apply. We describe here particular variants of both forms of the algorithm, the variants being chosen to emphasise the similarity between the algorithm and the original CG algorithm. Termination proofs are given in both cases and we consider first the HS (Concus-Golub) form of the algorithm whose defining equations are given above.

This proof is inductive. Assume that

$$(a) \quad \mathbf{p}_j^T \mathbf{r}_i = 0, \quad 1 \leq j < i, \text{ and}$$

$$(b) \quad \mathbf{p}_j^T \mathbf{Ap}_i = 0, \quad 1 \leq j < i.$$

We show that if equations (3.39) and (3.40) hold then (a) and (b) are true with i replaced by $i+1$. Premultiplying equation (3.39) by \mathbf{p}_i^T gives, trivially, $\mathbf{p}_i^T \mathbf{r}_{i+1} = 0$. A second premultiplication, this time by \mathbf{p}_j^T , gives $\mathbf{p}_j^T \mathbf{r}_{i+1} = 0$ immediately from (a) and (b) so that (a) holds for $i = i+1$.

We show now that (b) may be similarly be extended. Premultiplying equation (3.40) by $\mathbf{p}_i^T \mathbf{A}$ gives, trivially, $\mathbf{p}_i^T \mathbf{Ap}_{i+1} = 0$. Premultiplying by $\mathbf{p}_j^T \mathbf{A}$ gives, from (b),

$$\mathbf{p}_j^T \mathbf{Ap}_{i+1} = \mathbf{p}_j^T \mathbf{AKr}_{i+1}, \quad 1 \leq j \leq i-1. \quad (3.41)$$

Consider now equation (3.40) with $j-1$ substituted for i . Premultiplying this by \mathbf{r}_{i+1}^T yields, from (a) updated,

$$\mathbf{r}_{i+1}^T \mathbf{Kr}_j = 0, \quad 1 \leq j \leq i \quad (3.42)$$

and substituting j for i in equation (3.39) and premultiplying by $\mathbf{r}_{i+1}^T \mathbf{K}$ gives, from equation (3.42), if $\mathbf{p}_j^T \mathbf{r}_j \neq 0$,

$$\mathbf{r}_{i+1}^T \mathbf{KA} \mathbf{p}_j = \mathbf{p}_j^T \mathbf{A}^T \mathbf{Kr}_{i+1} = 0, \quad 1 \leq j \leq i-1. \quad (3.43)$$

Define now \mathbf{N} by

$$\mathbf{N} = \left(\frac{\mathbf{A} - \mathbf{A}^T}{2} \right)$$

so that $\mathbf{AK} = \mathbf{I} + \mathbf{NK}$ and $\mathbf{A}^T \mathbf{K} = \mathbf{I} - \mathbf{NK}$. Substituting the second of these expressions in equation (3.43) yields, from (a), $\mathbf{p}_j^T \mathbf{NKr}_{i+1} = 0$ so that, from the first expression and equation (3.41), $\mathbf{p}_j^T \mathbf{Ap}_{i+1} = 0$ establishing that (b) also holds for $i = i+1$. Since both (a) and (b) hold trivially for $i = 2$ the proof follows as long as $\mathbf{p}_j^T \mathbf{r}_j \neq 0$. Now substituting $j-1$ for i in equation (3.40) and premultiplying the resulting equation by \mathbf{r}_j^T yields $\mathbf{p}_j^T \mathbf{r}_j = \mathbf{r}_j^T \mathbf{Kr}_j$ so that if \mathbf{K} is positive definite, i.e.

\mathbf{A} is positive real, and a solution has not yet been obtained, then $\mathbf{p}_j^T \mathbf{r}_j \neq 0$ and the proof holds. The algorithm is thus completely stable if \mathbf{A} is positive real.

The Lanczos (Widlund) version of the algorithm is obtained by replacing equation (3.40) by

$$\mathbf{p}_{i+1} = \mathbf{K} \mathbf{A} \mathbf{p}_i - \mathbf{p}_i \alpha_i - \mathbf{p}_{i-1} \beta_{i-1} \quad (3.44)$$

where α_i and β_{i-1} are chosen so that $\mathbf{p}_{i-1}^T \mathbf{A} \mathbf{p}_{i+1} = \mathbf{p}_i^T \mathbf{A} \mathbf{p}_{i+1} = 0$. To obtain this equation we merely note that the \mathbf{G} and \mathbf{K} for the CGW algorithm satisfy equation (3.82) since we give below a full proof that this is sufficient for GODir to reduce to a 3-term recurrence.

The remarkable feature of the CGW algorithm is that despite \mathbf{G} ($= \mathbf{A}$) being nonsymmetric it can still in principle compute the exact solution of $\mathbf{Ax} = \mathbf{b}$ using only short recurrences. The forms of these and the associated conjugacy properties are then quite similar to those of the original CG algorithm. However the algorithm can do this only by effectively solving an equation of the form $(\mathbf{A} + \mathbf{A}^T) \mathbf{z} = \mathbf{c}$ at each iteration, and this severely limits its range of applicability. Unless this equation has a particularly simple form as would, for instance, be the case if \mathbf{A} consisted of the unit matrix plus some skew-symmetric perturbation, the method is just not competitive. It does, though, demonstrate that while the symmetry of the matrices \mathbf{G} and \mathbf{K} is sufficient to obtain termination with short recurrences this condition is not necessary. The necessary conditions for short recurrence termination are somewhat more complicated to obtain and were first given by Tytyshnikov and Voevodin [251]. They are considered further in the final section of this chapter.

Original references: [75], 1976 and [257], 1978.

3.6. Minimum-residual algorithms

For these algorithms, $\mathbf{A}^T \mathbf{A}$ is either equal to \mathbf{G} or is a block on its principal diagonal. Minimum-residual algorithms are, on the whole, stable and robust. The down side is that they are often prone to stagnation.

3.6.1. The conjugate residual algorithm (CR)

Type: HS

Parameters:

$$\mathbf{A} = \mathbf{A}^T, \quad \mathbf{G} = \mathbf{A}^2, \quad \mathbf{h} = \mathbf{Ab}, \quad \mathbf{K} = \mathbf{A}^{-1}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{p}_1 = \mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{q}_1 = \mathbf{Ap}_1$

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{r}_i^T \mathbf{A} \mathbf{r}_i}{\mathbf{q}_i^T \mathbf{q}_i} \right)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{q}_i \left(\frac{\mathbf{r}_i^T \mathbf{A} \mathbf{r}_i}{\mathbf{q}_i^T \mathbf{q}_i} \right)$$

$$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \mathbf{p}_i \left(\frac{\mathbf{r}_{i+1}^T \mathbf{A} \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{A} \mathbf{r}_i} \right)$$

$$\mathbf{q}_{i+1} = \mathbf{A} \mathbf{r}_{i+1} + \mathbf{q}_i \left(\frac{\mathbf{r}_{i+1}^T \mathbf{A} \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{A} \mathbf{r}_i} \right)$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{A}^2 \mathbf{p}_i = 0, j \neq i, \quad \mathbf{r}_j^T \mathbf{A} \mathbf{r}_i = 0, j \neq i,$

Termination properties: $\mathbf{p}_j^T \mathbf{A} \mathbf{r}_i = 0, j < i$

Numerical properties: Can neither crash nor stagnate if \mathbf{A} positive definite but can stagnate if \mathbf{A} is indefinite.

Notes: Generates the auxiliary sequence $\{\mathbf{q}_i\} = \{\mathbf{A}\mathbf{p}_i\}$ to avoid the calculation of an extra matrix-vector product.

Original reference: [225], 1955.

3.6.2. The modified conjugate residual algorithm (MCR)

Type: Lanczos

Parameters:

$$\mathbf{A} = \mathbf{A}^T, \quad \mathbf{G} = \mathbf{A}^2, \quad \mathbf{h} = \mathbf{Ab}, \quad \mathbf{K} = \mathbf{A}^{-1}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{p}_1 = \mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}, \quad \mathbf{q}_1 = \mathbf{Ap}_1$

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{q}_i^T \mathbf{r}_i}{\mathbf{q}_i^T \mathbf{q}_i} \right)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{q}_i \left(\frac{\mathbf{q}_i^T \mathbf{r}_i}{\mathbf{q}_i^T \mathbf{q}_i} \right)$$

$$\mathbf{p}_{i+1} = \mathbf{q}_i - \mathbf{p}_i \left(\frac{\mathbf{q}_i^T \mathbf{A} \mathbf{q}_i}{\mathbf{q}_i^T \mathbf{q}_i} \right) - \mathbf{p}_{i-1} \left(\frac{\mathbf{q}_i^T \mathbf{q}_i}{\mathbf{q}_{i-1}^T \mathbf{q}_{i-1}} \right)$$

$$\mathbf{q}_{i+1} = \mathbf{A} \mathbf{q}_i - \mathbf{q}_i \left(\frac{\mathbf{q}_i^T \mathbf{A} \mathbf{q}_i}{\mathbf{q}_i^T \mathbf{q}_i} \right) - \mathbf{q}_{i-1} \left(\frac{\mathbf{q}_i^T \mathbf{q}_i}{\mathbf{q}_{i-1}^T \mathbf{q}_{i-1}} \right)$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{A}^2 \mathbf{p}_i = \mathbf{q}_j^T \mathbf{q}_i = 0, j \neq i,$

Termination properties: $\mathbf{q}_j^T \mathbf{r}_i = 0, j < i$.

Numerical properties: Can neither crash nor stagnate if \mathbf{A} positive definite but temporary stagnation is possible if \mathbf{A} is indefinite.

Notes: The Lanczos version of CR. A second matrix-vector multiplication at each step, since the recursion for \mathbf{p}_i involves $\mathbf{p}_i^T \mathbf{A}^3 \mathbf{p}_i$, is avoided at the expense of some additional vector operations by generating recursively the auxiliary sequence $\{\mathbf{q}_i\} = \{\mathbf{A}\mathbf{p}_i\}$.

Original references: [110], [60], [61].

3.6.3. The CG normal residuals algorithm (CGNR)

Type: HS

Parameters:

$$\mathbf{G} = \mathbf{A}^T \mathbf{A}, \quad \mathbf{h} = \mathbf{A}^T \mathbf{b}, \quad \mathbf{K} = \mathbf{I}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{p}_1 = \mathbf{A}^T \mathbf{r}_1$

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{r}_i^T \mathbf{AA}^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A}^T \mathbf{Ap}_i} \right)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{Ap}_i \left(\frac{\mathbf{r}_i^T \mathbf{AA}^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A}^T \mathbf{Ap}_i} \right)$$

$$\mathbf{p}_{i+1} = \mathbf{A}^T \mathbf{r}_{i+1} + \mathbf{p}_i \left(\frac{\mathbf{r}_{i+1}^T \mathbf{AA}^T \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{AA}^T \mathbf{r}_i} \right)$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{A}^T \mathbf{Ap}_i = \mathbf{r}_j^T \mathbf{A}^T \mathbf{Ar}_i = 0, j \neq i$,

Termination properties: $\mathbf{p}_j^T \mathbf{A}^T \mathbf{r}_i = 0, j < i$.

Numerical properties: Can neither crash nor stagnate for any nonsingular \mathbf{A} but convergence in general is very slow. See Chapter 4 below.

Notes: This algorithm is just CG applied to the normal equations $\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b}$. The matrix $\mathbf{A}^T \mathbf{A}$ is not computed explicitly but each step of CGNR requires two matrix-vector multiplications, one by \mathbf{A} (\mathbf{Ap}_i) and the other by \mathbf{A}^T ($\mathbf{A}^T \mathbf{r}_{i+1}$).

Original references: [154], [217].

3.6.4. The biconjugate residual algorithm (BiCR)

Type: HS

Parameters:

$$\mathbf{G} = \begin{bmatrix} \mathbf{A}^T \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{A} \mathbf{A}^T \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{A}^T \mathbf{b} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \mathbf{c} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^{-1} \\ \mathbf{A}^{-T} & \mathbf{O} \end{bmatrix}$$

Initial values: $\mathbf{x}_1, \mathbf{s}_1$ arbitrary, $\mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$, $\mathbf{u}_1 = \mathbf{s}_1$, $[\mathbf{v}_1 = \mathbf{r}_1]$, $\mathbf{y}_1 = \mathbf{A}^T \mathbf{r}_1$,

Recursions:

$$\mathbf{w}_i = \mathbf{A}\mathbf{u}_i$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{s}_i^T \mathbf{A}^T \mathbf{r}_i}{\mathbf{w}_i^T \mathbf{w}_i} \right)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \left(\frac{\mathbf{s}_i^T \mathbf{A}^T \mathbf{r}_i}{\mathbf{w}_i^T \mathbf{w}_i} \right)$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \left(\frac{\mathbf{s}_i^T \mathbf{A}^T \mathbf{r}_i}{\mathbf{y}_i^T \mathbf{y}_i} \right)$$

$$\mathbf{u}_{i+1} = \mathbf{s}_{i+1} + \mathbf{u}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{A}^T \mathbf{r}_{i+1}}{\mathbf{s}_i^T \mathbf{A}^T \mathbf{r}_i} \right)$$

$$\left[\mathbf{v}_{i+1} = \mathbf{r}_{i+1} + \mathbf{v}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{A}^T \mathbf{r}_{i+1}}{\mathbf{s}_i^T \mathbf{A}^T \mathbf{r}_i} \right) \right]$$

$$\mathbf{y}_{i+1} = \mathbf{A}^T \mathbf{r}_{i+1} + \mathbf{y}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{A}^T \mathbf{r}_{i+1}}{\mathbf{s}_i^T \mathbf{A}^T \mathbf{r}_i} \right)$$

Conjugacy properties: $\mathbf{u}_j^T \mathbf{A}^T \mathbf{A} \mathbf{u}_i = \mathbf{v}_j^T \mathbf{A} \mathbf{A}^T \mathbf{v}_i = \mathbf{s}_j^T \mathbf{A}^T \mathbf{r}_i = 0, j \neq i$

Termination properties: $\mathbf{u}_j^T \mathbf{A}^T \mathbf{r}_i = \mathbf{v}_j^T \mathbf{A} \mathbf{s}_i = 0, j < i$,

Numerical properties: Can stagnate for any \mathbf{A} .

Notes: The compound version of CR. Generates the auxiliary sequence $\{\mathbf{y}_i\} = \{\mathbf{A}^T \mathbf{v}_i\}$ in order to avoid an extra matrix-vector multiplication. This recursion is based on that for $\{\mathbf{v}_i\}$ which is not itself computed. This algorithm may also be used for solving the over-determined linear least-squares problem merely by altering the program to accommodate rectangular matrices. The equations are unchanged.

If we denote quantities pertaining to the BiCG and HG algorithms by the subscript G and those pertaining to the BiCR algorithm by the subscript R, then $\mathbf{G}_R = \mathbf{G}_G^2$, $\mathbf{H}_R = \mathbf{G}_G \mathbf{H}_G$ and $\mathbf{K}_R = \mathbf{G}_G^{-1}$. These relationships will be exploited in Chapter 11 when discussing the preconditioned version of BiCR.

Original references: [149], 1990, [150], 1991 and [151], 1991.

3.6.5. The biconjugate residual algorithm (BiCRL)

Type: Lanczos

Parameters:

$$\mathbf{G} = \begin{bmatrix} \mathbf{A}^T \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{A} \mathbf{A}^T \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{A}^T \mathbf{b} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \mathbf{c} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^{-1} \\ \mathbf{A}^{-T} & \mathbf{O} \end{bmatrix}$$

Initial values: $\mathbf{x}_1, \mathbf{u}_1$ arbitrary, $\mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$, $\mathbf{y}_1 = \mathbf{A}^T\mathbf{r}_1$, $[\mathbf{v}_1 = \mathbf{r}_1]$,

Recursions:

$$\mathbf{w}_i = \mathbf{A}\mathbf{u}_i$$

$$\mathbf{z}_i = \mathbf{A}^T\mathbf{w}_i$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{w}_i^T \mathbf{r}_i}{\mathbf{w}_i^T \mathbf{w}_i} \right)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \left(\frac{\mathbf{w}_i^T \mathbf{r}_i}{\mathbf{w}_i^T \mathbf{w}_i} \right)$$

$$\mathbf{u}_{i+1} = \mathbf{y}_i - \mathbf{u}_i \left(\frac{\mathbf{z}_i^T \mathbf{y}_i}{\mathbf{w}_i^T \mathbf{w}_i} \right) - \mathbf{u}_{i-1} \left(\frac{\mathbf{y}_i^T \mathbf{y}_i}{\mathbf{w}_{i-1}^T \mathbf{w}_{i-1}} \right)$$

$$\left[\mathbf{v}_{i+1} = \mathbf{w}_i - \mathbf{v}_i \left(\frac{\mathbf{z}_i^T \mathbf{y}_i}{\mathbf{y}_i^T \mathbf{y}_i} \right) - \mathbf{v}_{i-1} \left(\frac{\mathbf{w}_i^T \mathbf{w}_i}{\mathbf{y}_{i-1}^T \mathbf{y}_{i-1}} \right) \right]$$

$$\mathbf{y}_{i+1} = \mathbf{z}_i - \mathbf{y}_i \left(\frac{\mathbf{z}_i^T \mathbf{y}_i}{\mathbf{y}_i^T \mathbf{y}_i} \right) - \mathbf{y}_{i-1} \left(\frac{\mathbf{w}_i^T \mathbf{w}_i}{\mathbf{y}_{i-1}^T \mathbf{y}_{i-1}} \right)$$

Conjugacy properties: $\mathbf{u}_j^T \mathbf{A}^T \mathbf{A} \mathbf{u}_i = \mathbf{v}_j^T \mathbf{A} \mathbf{A}^T \mathbf{v}_i = \mathbf{s}_j^T \mathbf{A}^T \mathbf{r}_i = 0, j \neq i$

Termination properties: $\mathbf{u}_j^T \mathbf{A}^T \mathbf{r}_i = \mathbf{v}_j^T \mathbf{A} \mathbf{s}_i = 0, j < i$,

Numerical properties: Temporary stagnation is possible for any \mathbf{A} .

Notes: The compound version of MCR. Generates the auxiliary sequence $\{\mathbf{y}_i\} = \{\mathbf{A}^T \mathbf{v}_i\}$ in order to avoid an extra matrix-vector multiplication. This recursion is based on that for $\{\mathbf{v}_i\}$ which is not itself computed.

Original references: [149], 1990, [150], 1991 and [151], 1991.

3.7. Minimum-error algorithms

These methods minimise the Euclidean norm of the error $\mathbf{e} = \mathbf{x} - \mathbf{x}^*$ at each step and it follows from Theorem 1 that they can do this if they satisfy the Galerkin

condition $\mathbf{p}_i^T \mathbf{f}_{i+1} = 0$ for all i , where $\mathbf{f}_i = \mathbf{G}\mathbf{e}_i$ and $\mathbf{G} = \mathbf{I}$ (we consider here only the versions of these algorithms that generate vector sequences). However, satisfaction of the Galerkin condition is non-trivial as in this case $\mathbf{f}(\mathbf{x}) \equiv \mathbf{x} - \mathbf{A}^{-1}\mathbf{b}$ and difficulties arise since $\mathbf{A}^{-1}\mathbf{b}$ is unknown. If \mathbf{K} is chosen appropriately there are no problems with computing the sequence $\{\mathbf{p}_i\}$. They arise when calculating the sequence $\{\mathbf{x}_i\}$ since the equations by which \mathbf{x}_{i+1} is computed, either (3.12) or (3.16), both involve \mathbf{f}_i and hence $\mathbf{A}^{-1}\mathbf{b}$. It is necessary therefore to use either some equivalent formulation of $\mathbf{p}_i^T \mathbf{f}_i$ (algorithm OD) or to generate a sequence of auxiliary vectors $\{\mathbf{v}_i\}$ to enable $\mathbf{p}_i^T \mathbf{f}_i$ to be computed indirectly (algorithms StOD and CGNE). An additional complication is that, for both OD and StOD, \mathbf{p}_1 must be set equal not to $\mathbf{K}\mathbf{f}_1$ but to $\mathbf{K}^2\mathbf{f}_1$. This does not affect the generation of the sequence $\{\mathbf{p}_i\}$ in the absence of breakdown since this is generated by the GAA, nor the termination properties since $\mathbf{G} = \mathbf{I}$ and these are guaranteed by the corollary to Theorem 11.

All the algorithms in this section except CGNE are of Lanczos type.

3.7.1. The method of orthogonal directions (OD)

Parameters:

$$\mathbf{A} = \mathbf{A}^T, \quad \mathbf{G} = \mathbf{I}, \quad \mathbf{h} = \mathbf{A}^{-1}\mathbf{b}, \quad \mathbf{K} = \mathbf{A}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{p}_1 = \mathbf{Ar}_1$

Recursions:

$$\mathbf{x}_2 = \mathbf{x}_1 - \mathbf{p}_1 \left(\frac{\mathbf{r}_1^T \mathbf{r}_1}{\mathbf{p}_1^T \mathbf{p}_1} \right)$$

$$\mathbf{r}_2 = \mathbf{r}_1 - \mathbf{Ap}_1 \left(\frac{\mathbf{r}_1^T \mathbf{r}_1}{\mathbf{p}_1^T \mathbf{p}_1} \right)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{p}_{i-1}^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right), \quad i \geq 2$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{Ap}_i \left(\frac{\mathbf{p}_{i-1}^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right), \quad i \geq 2$$

$$\mathbf{p}_{i+1} = \mathbf{Ap}_i - \mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{Ap}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right) - \mathbf{p}_{i-1} \left(\frac{\mathbf{p}_i^T \mathbf{p}_i}{\mathbf{p}_{i-1}^T \mathbf{p}_{i-1}} \right)$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{p}_i = \mathbf{r}_j^T \mathbf{A}^{-1} \mathbf{r}_i = 0, j \neq i$

Termination properties: $\mathbf{p}_j^T \mathbf{r}_i = 0, j < i$

Numerical properties: Can neither crash nor stagnate if \mathbf{A} positive definite but can stagnate if \mathbf{A} is indefinite. In practice this algorithm has proved to be numerically

unstable (see [227] and [226]).

Notes: Since equation (3.6) is satisfied for the GAA and since, for the GAA,

$$\mathbf{w}_j = \mathbf{KGp}_{j-1},$$

we have

$$\mathbf{p}_j^T \mathbf{f}_{i+1} = \mathbf{p}_{j-1}^T \mathbf{GKf}_{i+1}$$

and putting $j = i + 1$, $\mathbf{G} = \mathbf{I}$ and $\mathbf{K} = \mathbf{A}$ yields

$$\mathbf{p}_{i+1}^T \mathbf{f}_{i+1} = \mathbf{p}_i^T \mathbf{Af}_{i+1}.$$

Substituting $i - 1$ for i and remembering that if $\mathbf{G} = \mathbf{I}$ then $\mathbf{r}_i = \mathbf{Af}_i$ yields

$$\mathbf{p}_i^T \mathbf{f}_i = \mathbf{p}_{i-1}^T \mathbf{r}_i, \quad (3.45)$$

and it is this form that appears in the recursions for \mathbf{x}_i and \mathbf{r}_i .

Original references: [123], 1963 and [110], 1976.

3.7.2. The stabilised OD method (*StOD*)

Parameters:

$$\mathbf{A} = \mathbf{A}^T, \quad \mathbf{G} = \mathbf{I}, \quad \mathbf{h} = \mathbf{A}^{-1}\mathbf{b}, \quad \mathbf{K} = \mathbf{A}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{v}_1 = \mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{p}_1 = \mathbf{Av}_1$.

Recursions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right),$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{Ap}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right),$$

$$\mathbf{p}_{i+1} = \mathbf{Ap}_i - \mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{Ap}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right) - \mathbf{p}_{i-1} \left(\frac{\mathbf{p}_i^T \mathbf{p}_i}{\mathbf{p}_{i-1}^T \mathbf{p}_{i-1}} \right)$$

$$\mathbf{v}_{i+1} = \mathbf{p}_i - \mathbf{v}_i \left(\frac{\mathbf{p}_i^T \mathbf{Ap}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right) - \mathbf{v}_{i-1} \left(\frac{\mathbf{p}_i^T \mathbf{p}_i}{\mathbf{p}_{i-1}^T \mathbf{p}_{i-1}} \right)$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{p}_i = \mathbf{r}_j^T \mathbf{A}^{-1} \mathbf{r}_i = 0, j \neq i$

Termination properties: $\mathbf{p}_j^T \mathbf{r}_i = 0, j < i$

Numerical properties: Can neither crash nor stagnate if \mathbf{A} positive definite but temporary stagnation is possible if \mathbf{A} is indefinite.

Notes: Generates the sequence $\{\mathbf{p}_i\}$ using the standard Lanczos recursion and the

auxiliary sequence $\{\mathbf{v}_i\} = \{\mathbf{A}^{-1}\mathbf{p}_i\}$ as the alternative way of computing $\mathbf{p}_i^T \mathbf{A}^{-1} \mathbf{r}_i$.
Original reference: [121], 1992.

3.7.3. The CG normal errors, or Craig's, algorithm (CGNE)

Parameters:

$$\mathbf{G} = \mathbf{I}, \quad \mathbf{h} = \mathbf{A}^{-1}\mathbf{b}, \quad \mathbf{K} = \mathbf{A}^T \mathbf{A}$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{v}_1 = \mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$, $\mathbf{p}_1 = \mathbf{A}^T \mathbf{r}_1$.

Recursions:

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right), \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \mathbf{A}\mathbf{p}_i \left(\frac{\mathbf{v}_i^T \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{p}_i} \right), \\ \mathbf{p}_{i+1} &= \mathbf{A}^T \mathbf{r}_{i+1} - \mathbf{p}_i \left(\frac{\mathbf{p}_i^T \mathbf{A}^T \mathbf{r}_{i+1}}{\mathbf{p}_i^T \mathbf{p}_i} \right), \\ \mathbf{v}_{i+1} &= \mathbf{r}_{i+1} - \mathbf{v}_i \left(\frac{\mathbf{p}_i^T \mathbf{A}^T \mathbf{r}_{i+1}}{\mathbf{p}_i^T \mathbf{p}_i} \right).\end{aligned}$$

Conjugacy properties: $\mathbf{p}_j^T \mathbf{p}_i = \mathbf{r}_j^T \mathbf{r}_i = 0$, $j \neq i$

Termination properties: $\mathbf{p}_j^T \mathbf{A}^{-1} \mathbf{r}_i = 0$, $j < i$

Numerical properties: Can neither crash nor stagnate for any nonsingular \mathbf{A} but convergence in general is very slow. See Chapter 4 below.

Notes: Generates the auxiliary sequence $\{\mathbf{v}_i\} = \{\mathbf{A}^{-T}\mathbf{p}_i\}$ in order to compute $\mathbf{p}_i^T \mathbf{A}^{-1} \mathbf{r}_i$. This method is usually regarded as solving the equations $\mathbf{A}\mathbf{A}^T \mathbf{y} = \mathbf{b}$ by CG (since $\mathbf{A}\mathbf{A}^T$ is symmetric and positive definite) with the solution \mathbf{x} being retrieved by $\mathbf{x} = \mathbf{A}^T \mathbf{y}$.

Original reference: [79], 1955.

3.8. Lanczos-based methods

For the algorithms of this section, the vectors $\{\mathbf{p}_i\}$, $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ are either generated by the original Lanczos algorithm (see page 47) and are orthogonal (SymmLQ, LSQR) or by a variation of this and are bi-orthogonal (QMR). The sequences $\{\mathbf{x}_i\}$ and $\{\mathbf{r}_i\}$ therefore have to be computed using devices specific to each individual algorithm. In the following descriptions, the parameters refer only to that part of the algorithm that generates the vector sequences $\{\mathbf{p}_i\}$, $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$.

3.8.1. SymmLQ

Parameters:

$$\mathbf{A} = \mathbf{A}^T, \quad \mathbf{G} = \mathbf{I}, \quad \mathbf{K} = \mathbf{A}, \quad \mathbf{P}_1 = \mathbf{p}_1.$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{p}_1 = \mathbf{r}_1 / \|\mathbf{r}_1\|$.

Notes: This algorithm is a minimum error algorithm which solves the equation $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is symmetric and generally indefinite. It works by generating a sequence of *orthonormal* vectors $\{\mathbf{y}_i\}$ and at each step minimising the Euclidean norm of the error over the manifold

$$\mathbf{x}(z) \equiv \mathbf{x}_i + \mathbf{y}_i z. \quad (3.46)$$

Since $\mathbf{e}(z) \equiv \mathbf{x}(z) - \mathbf{A}^{-1}\mathbf{b}$, $\mathbf{e}(z) \equiv \mathbf{e}_i + \mathbf{y}_i z$ and it straightforward to show that since $\mathbf{y}_i^T \mathbf{y}_i = 1$ the value of z that minimises $\|\mathbf{e}(z)\|^2$ is $-\mathbf{y}_i^T \mathbf{e}_i$. Hence

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{y}_i \mathbf{y}_i^T \mathbf{e}_i \quad (3.47)$$

and

$$\mathbf{e}_{i+1} = (\mathbf{I} - \mathbf{y}_i \mathbf{y}_i^T) \mathbf{e}_i$$

so that

$$\mathbf{e}_{i+1} = (\mathbf{I} - \mathbf{y}_i \mathbf{y}_i^T)(\mathbf{I} - \mathbf{y}_{i-1} \mathbf{y}_{i-1}^T) \dots (\mathbf{I} - \mathbf{y}_1 \mathbf{y}_1^T) \mathbf{e}_1. \quad (3.48)$$

If now

$$\bar{\mathbf{Y}}_i = [\mathbf{y}_1 \ \mathbf{y}_2 \ \dots \ \mathbf{y}_i]$$

it follows that since the vectors $\{\mathbf{y}_j\}$ are orthonormal, equation (3.48) may be written

$$\mathbf{e}_{i+1} = \left(\mathbf{I} - \sum_{j=1}^i \mathbf{y}_j \mathbf{y}_j^T \right) \mathbf{e}_1 = (\mathbf{I} - \bar{\mathbf{Y}}_i \bar{\mathbf{Y}}_i^T) \mathbf{e}_1.$$

Termination in at most n steps is guaranteed since $\bar{\mathbf{Y}}_n$ is a square orthogonal matrix and hence $\mathbf{I} - \bar{\mathbf{Y}}_n \bar{\mathbf{Y}}_n^T = \mathbf{O}$. This, then, is the theory. The problem in practice lies in computing \mathbf{x}_{i+1} from equation (3.47) since it involves the unknown error \mathbf{e}_i .

This problem is solved in SymmLQ by first generating an orthonormal sequence of vectors $\{\mathbf{p}_i\}$ by the OAA with $\mathbf{B} = \mathbf{A}$ (see page 28 above) so that, from equation (2.10) with $\mathbf{B} = \mathbf{A}$,

$$\mathbf{A} \bar{\mathbf{P}}_i = \bar{\mathbf{P}}_{i+1} \tilde{\mathbf{H}}_{i+1} \quad (3.49)$$

where $\bar{\mathbf{P}}_i \in \mathbb{R}^{n \times i}$, $\tilde{\mathbf{H}}_{i+1} \in \mathbb{R}^{(i+1) \times i}$ and $\bar{\mathbf{P}}_i^T \bar{\mathbf{P}}_i = \mathbf{I}$. If \mathbf{Q}_{i+1} denotes the orthogonal matrix such that

$$\mathbf{Q}_{i+1} \tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \tilde{\mathbf{U}}_i \\ \mathbf{0}^T \end{bmatrix} \quad (3.50)$$

where $\tilde{\mathbf{U}}_i \in \mathbb{R}^{i \times i}$ and is upper triangular, $\bar{\mathbf{Y}}_i$ and \mathbf{z}_{i+1} may be defined by

$$\bar{\mathbf{P}}_{i+1} \mathbf{Q}_{i+1}^T = [\bar{\mathbf{Y}}_i \ \mathbf{z}_{i+1}] . \quad (3.51)$$

Since the columns of $\bar{\mathbf{P}}_{i+1}$ are orthonormal by construction it follows that $\bar{\mathbf{Y}}_i^T \bar{\mathbf{Y}}_i = \mathbf{I}$ and, from equations (3.49) - (3.51),

$$\mathbf{A} \bar{\mathbf{P}}_i = \bar{\mathbf{P}}_{i+1} \mathbf{Q}_{i+1}^T \mathbf{Q}_{i+1} \tilde{\mathbf{H}}_{i+1} = \bar{\mathbf{Y}}_i \tilde{\mathbf{U}}_i \quad (3.52)$$

so that

$$\mathbf{A}^{-1} \bar{\mathbf{Y}}_i = \bar{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1}.$$

Now the residual \mathbf{r} and the error \mathbf{e} of the system of equations $\mathbf{Ax} = \mathbf{b}$ are related by $\mathbf{e} = \mathbf{A}^{-1} \mathbf{r}$ (see equation (1.5)) so since \mathbf{A} (and hence \mathbf{A}^{-1}) is symmetric,

$$\mathbf{e}_i^T \bar{\mathbf{Y}}_i = \mathbf{r}_i^T \bar{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1}$$

and, if \mathbf{s}_i denotes the last column of the unit matrix of order i ,

$$\mathbf{e}_i^T \mathbf{y}_i = \mathbf{r}_i^T \bar{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1} \mathbf{s}_i.$$

Equation (3.47) may thus be written

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{y}_i \mathbf{r}_i^T \bar{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1} \mathbf{s}_i \quad (3.53)$$

where all the quantities on the right-hand side are known.

Now the matrix $\tilde{\mathbf{H}}_{i+1}$ of equation (3.49) may be written

$$\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \bar{\mathbf{H}}_i \\ \mathbf{h}_{i+1}^T \end{bmatrix},$$

so that premultiplication of equation (3.49) by $\bar{\mathbf{P}}_i^T$ gives $\bar{\mathbf{P}}_i^T \mathbf{A} \bar{\mathbf{P}}_i = \bar{\mathbf{H}}_i$. Since \mathbf{A} is symmetric and $\bar{\mathbf{H}}_i$ is upper Hessenberg, $\bar{\mathbf{H}}_i$ and $\tilde{\mathbf{H}}_{i+1}$ are tridiagonal. This implies that when $\tilde{\mathbf{U}}_i$ is computed from $\tilde{\mathbf{H}}_{i+1}$ by plane rotations, only its principal diagonal and the two adjacent super-diagonals are non-zero (see Appendix A). The calculation of the matrices $\bar{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1}$ and the subsequent computation of \mathbf{x}_{i+1} are described in Chapter 6, page 117.

Original reference: [202], 1975.

3.8.2. LSQR and LSQR2

Parameters (LSQR):

$$\mathbf{G} = \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}, \quad \mathbf{P}_1 = \begin{bmatrix} \mathbf{0} \\ \mathbf{s}_1 \end{bmatrix}.$$

Initial values: \mathbf{x}_1 arbitrary, $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{s}_1 = \mathbf{r}_1 / \|\mathbf{r}_1\|$.

Notes: LSQR works by constructing a sequence $\{\mathbf{w}_j\}$ of vectors mutually conjugate with respect to the matrix $\mathbf{A}^T \mathbf{A}$. It then uses these to compute the solution of $\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b}$ via equations (1.5) and (1.8) with $\mathbf{G} = \mathbf{A}^T \mathbf{A}$ and $\mathbf{h} = \mathbf{A}^T \mathbf{b}$ so that $\mathbf{f}_j = \mathbf{A}^T \mathbf{r}_j$. This approach is also employed by Orthodir, GCR and BiCR and the sequence of approximate solutions $\{\mathbf{x}_i\}$ is computed by all these algorithms using equation (2.19). Substituting \mathbf{w}_j for \mathbf{p}_i in this equation (a mere notational change) then gives

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \mathbf{w}_j \left(\frac{\mathbf{w}_j^T \mathbf{A}^T \mathbf{r}_j}{\mathbf{w}_j^T \mathbf{A}^T \mathbf{A} \mathbf{w}_j} \right). \quad (3.54)$$

This is completely standard. Where LSQR differs from the other minimum-residual algorithms is the way in which it computes the vectors $\{\mathbf{w}_j\}$, which it does by first computing a sequence of *orthogonal* vectors $\{\mathbf{v}_i\}$.

If the above parameters and initial conditions are substituted in equation (3.11) it follows that \mathbf{p}_j has the form $\begin{bmatrix} \mathbf{v}_j \\ \mathbf{0} \end{bmatrix}$ for j even and $\begin{bmatrix} \mathbf{0} \\ \mathbf{s}_j \end{bmatrix}$ for j odd where

$$\mathbf{v}_{j+1} = (\mathbf{A}^T \mathbf{s}_j - \mathbf{v}_{j-1} \beta_j) / \alpha_{j+1} \quad j \text{ odd}, \quad (3.55)$$

$$\mathbf{s}_{j+1} = (\mathbf{A} \mathbf{v}_j - \mathbf{s}_{j-1} \alpha_j) / \beta_{j+1} \quad j \text{ even} \quad (3.56)$$

and α_{j+1} and β_{j+1} are chosen so that $\|\mathbf{v}_{j+1}\| = \|\mathbf{s}_{j+1}\| = 1$. This form of the Lanczos recursion is due to Golub and Kahan [128] and the vector sequences $\{\mathbf{v}_j\}$ and $\{\mathbf{s}_j\}$ are both orthonormal.

Let now

$$\bar{\mathbf{V}}_i = [\mathbf{v}_2, \mathbf{v}_4, \dots, \mathbf{v}_{2i}] \quad (3.57)$$

and

$$\bar{\mathbf{S}}_{i+1} = [\mathbf{s}_1, \mathbf{s}_3, \dots, \mathbf{s}_{2i+1}] \quad (3.58)$$

(the matrix subscripts i and $i + 1$ denote the number of columns of each matrix). Equation (3.56), $j = 2, 4, \dots, 2i$, gives

$$\mathbf{A} \bar{\mathbf{V}}_i = \bar{\mathbf{S}}_{i+1} \tilde{\mathbf{H}}_{i+1} \quad (3.59)$$

where

$$\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ \beta_3 & \alpha_3 & \cdots & 0 \\ 0 & \beta_5 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{2i-1} \\ 0 & 0 & \cdots & \beta_{2i+1} \end{bmatrix}. \quad (3.60)$$

If \mathbf{Q}_{i+1} denotes the orthogonal matrix such that

$$\mathbf{Q}_{i+1}\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \tilde{\mathbf{U}}_i \\ \mathbf{0}^T \end{bmatrix}$$

where $\tilde{\mathbf{U}}_i$ is upper triangular, and the vector \mathbf{z}_{i+1} (which will be immediately discarded) and matrix $\bar{\mathbf{Y}}_i \in \mathbb{R}^{n \times i}$ are defined by

$$\bar{\mathbf{S}}_{i+1}\mathbf{Q}_{i+1}^T = [\bar{\mathbf{Y}}_i \ \mathbf{z}_{i+1}] \quad (3.61)$$

then equation (3.59) may be written

$$\mathbf{A}\bar{\mathbf{V}}_i = \bar{\mathbf{Y}}_i\tilde{\mathbf{U}}_i. \quad (3.62)$$

Now since \mathbf{Q}_{i+1} is orthogonal and the columns of $\bar{\mathbf{S}}_{i+1}$ are orthonormal by construction it follows from equation (3.61) that the columns of $\bar{\mathbf{Y}}_i$ are orthonormal. Thus if

$$\bar{\mathbf{W}}_i = [\mathbf{w}_1 \ \mathbf{w}_2 \ \dots \ \mathbf{w}_i] \quad (3.63)$$

is given by

$$\bar{\mathbf{W}}_i = \bar{\mathbf{V}}_i\tilde{\mathbf{U}}_i^{-1} \quad (3.64)$$

it follows from equation (3.62) that $\bar{\mathbf{Y}}_i = \mathbf{A}\bar{\mathbf{W}}_i$ so that not only are the vectors \mathbf{w}_j mutually conjugate with respect to $\mathbf{A}^T\mathbf{A}$ but, in addition, $\mathbf{w}_j^T\mathbf{A}^T\mathbf{A}\mathbf{w}_j = 1$. Equation (3.54) thus simplifies to

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \mathbf{w}_j\mathbf{w}_j^T\mathbf{A}^T\mathbf{r}_j. \quad (3.65)$$

As in the case of SymmLQ, the calculation of the matrices $\bar{\mathbf{W}}_i$ from equation (3.64) is described in Chapter 6.

For LSQR2, \mathbf{P}_1 is given by $\mathbf{P}_1 = \begin{bmatrix} \mathbf{v}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{s}_1 \end{bmatrix}$ where \mathbf{v}_1 is arbitrary, and substitution of the parameters in equation (3.1) gives

$$\mathbf{s}_{i+1} = (\mathbf{I} - \mathbf{s}_i\mathbf{s}_i^T - \mathbf{s}_{i-1}\mathbf{s}_{i-1}^T)\mathbf{A}\mathbf{v}_i/\beta_{i+1} \quad (3.66)$$

where β_{i+1} is chosen so that $\|\mathbf{s}_{i+1}\| = 1$. A similar equation may be derived for \mathbf{v}_{i+1} . From equation (3.66) we obtain, analogously to equation (3.59),

$$\mathbf{A}\bar{\mathbf{V}}_i = \bar{\mathbf{S}}_{i+1}\tilde{\mathbf{H}}_{i+1}$$

but where $\tilde{\mathbf{H}}_{i+1}$ is now tridiagonal. The remaining analysis follows that of LSQR but now $\tilde{\mathbf{V}}_{i+1}$ has three rather than two nonzero diagonals. Despite the greater generality of this variation (since it involves an arbitrary \mathbf{v}_1), tests conducted by the authors showed no significant improvement over LSQR.

Original references: [153] 1976, [203] 1982 and [48] 1997.

3.8.3. QMR (original version but without look-ahead)

Parameters:

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{I} \\ \mathbf{I} & \mathbf{O} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{O} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{O} \end{bmatrix}, \quad \mathbf{P}_1 = \begin{bmatrix} \mathbf{u}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_1 \end{bmatrix}.$$

Initial values: $\mathbf{x}_1, \mathbf{v}_1$ arbitrary, $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$, $\mathbf{u}_1 = \mathbf{r}_1$.

Notes: Substitution of the above parameters in equation (3.1) (not the alternative form) gives

$$\mathbf{u}_{i+1} = \mathbf{A}\mathbf{u}_i - \mathbf{u}_i \left(\frac{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i}{\mathbf{v}_i^T \mathbf{u}_i} \right) - \mathbf{u}_{i-1} \left(\frac{\mathbf{v}_{i-1}^T \mathbf{A} \mathbf{u}_i}{\mathbf{v}_{i-1}^T \mathbf{u}_{i-1}} \right) \quad (3.67)$$

and

$$\mathbf{v}_{i+1} = \mathbf{A}^T \mathbf{v}_i - \mathbf{v}_i \left(\frac{\mathbf{u}_i^T \mathbf{A}^T \mathbf{v}_i}{\mathbf{v}_i^T \mathbf{u}_i} \right) - \mathbf{v}_{i-1} \left(\frac{\mathbf{u}_{i-1}^T \mathbf{A}^T \mathbf{v}_i}{\mathbf{v}_{i-1}^T \mathbf{u}_{i-1}} \right). \quad (3.68)$$

The first of these equations has precisely the same form as equation (1.23) with $\mathbf{s}_i = \mathbf{y}_i$ and so can be used to quasi-minimise $\|\mathbf{r}(\mathbf{x}(\mathbf{w}))\|$ over \mathbf{w} where

$$\mathbf{x}(\mathbf{w}) \equiv \mathbf{x}_1 + \bar{\mathbf{S}}_i \mathbf{w} \quad (3.69)$$

and

$$\bar{\mathbf{S}}_i = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_i]$$

(see Chapter 1, The QMR technique and Chapter 6). If no breakdown occurs then eventually, for some value of i , \mathbf{u}_{i+1} becomes null and in this case and with exact arithmetic the QMR algorithm gives the exact solution.

The description given above is that of a simplified form of the algorithm, simplified in order to expose its basic structure. For example, scaling factors have been omitted. Since both \mathbf{G} and \mathbf{K} must necessarily be indefinite regardless of the properties of \mathbf{A} , not only is serious breakdown (or near-breakdown) a potential problem but incurable breakdown is a remote theoretical possibility. In the full form of the algorithm as given in [119] the former is overcome by the use of “look-ahead” techniques (see Chapter 7) while the latter, in the unlikely event that it arises, requires a restart from a different initial approximation.

Original reference: [118], 1991.

This completes our survey of the basic CG algorithms. Some are well-known, some are hardly known at all and some are better known in other guises. All except SymmLQ, LSQR and QMR may usefully be regarded as special cases of the block CG algorithm and all, with the sole exception of CGW, derive their short recurrences from the symmetry of both \mathbf{G} and \mathbf{K} . In the next and final section of this chapter we explore in greater depth the conditions necessary for short recurrences. These conditions are not that much more general than the simple symmetries quoted above and have not, at the time of writing, had much practical impact. They might, however, be useful for constructing algorithms tailored to fit special problems.

3.9. Existence of short recurrences*

Earlier in this chapter we showed that if both \mathbf{G} and \mathbf{K} are symmetric then the long recurrence formula of GODir could be reduced to the 3-term one of equation (3.1), and that of GOMin to the 2-term one of equation (3.5), while still guaranteeing finite termination. Thus the symmetry of both \mathbf{G} and \mathbf{K} is sufficient for algorithms employing short recurrences to terminate, and the question naturally arises as to whether or not these symmetries are necessary for termination. That they are not so is demonstrated by the CGW algorithm which in both forms requires only short recurrences despite \mathbf{G} not being symmetric, so again the question arises as to just what, if any, are the necessary conditions for short recurrences to achieve termination. These questions were resolved independently by two groups of researchers, one working in Russia and one in the United States. We outline the conclusions of both groups, the Russian one for its historical precedence and its greater directness and generality and the American one for its thorough and comprehensive analysis and the greater accessibility of its findings. Before doing this, though, we prove the sufficiency (but not necessity) of our own version of these results.

In order to do this we review the termination theory derived at the beginning of this chapter but without any assumptions of symmetry. Let $\bar{\mathbf{P}}_i$ be given by equation (1.31). To ensure termination we want $\bar{\mathbf{P}}_i^T \mathbf{F}_{i+1} = \mathbf{O}$, or, equivalently,

$$\mathbf{P}_j^T \mathbf{F}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i, \tag{3.70}$$

and if, for $1 \leq j \leq i$, \mathbf{F}_{j+1} is computed from \mathbf{F}_j by equation (1.37) where $\mathbf{D}_j = \mathbf{P}_j^T \mathbf{G} \mathbf{P}_j$ then trivially $\mathbf{P}_j^T \mathbf{F}_{j+1} = \mathbf{O}$. If, in addition, $\mathbf{P}_j^T \mathbf{G} \mathbf{P}_k = \mathbf{O}$ for $1 \leq j \leq k < i$ then equation (3.70) is also satisfied. This implies that a sufficient condition for termination is

$$\bar{\mathbf{P}}_i^T \mathbf{G} \bar{\mathbf{P}}_i = \bar{\mathbf{L}}_i \tag{3.71}$$

where $\bar{\mathbf{L}}_i$ is block lower triangular, the conjugacy of the matrices $\bar{\mathbf{P}}_i$ implied by equation (1.41) now being replaced by semiconjugacy since \mathbf{G} is no longer assumed to be symmetric.

To ensure that the computed matrices $\{\mathbf{P}_j\}$ are semiconjugate we first define the matrix $\widehat{\mathbf{Q}}_i$ by its transpose $\widehat{\mathbf{Q}}_i^T$ which is defined by

$$\widehat{\mathbf{Q}}_i^T = \mathbf{I} - \overline{\mathbf{P}}_i \overline{\mathbf{L}}_i^{-1} \overline{\mathbf{P}}_i^T \mathbf{G}. \quad (3.72)$$

$\widehat{\mathbf{Q}}_i$ is thus analogous to the matrix \mathbf{Q}_i of equation (1.46) but with \mathbf{G} replaced by the (nonsymmetric) \mathbf{G}^T . Clearly, from equations (3.71) and (3.72), $\overline{\mathbf{P}}_i^T \mathbf{G} \widehat{\mathbf{Q}}_i^T = \mathbf{O}$ so if \mathbf{P}_i is computed by

$$\mathbf{P}_i = \widehat{\mathbf{Q}}_{i-1}^T \mathbf{W}_i \quad (3.73)$$

for any arbitrary matrix \mathbf{W}_i the conditions for semiconjugacy are automatically satisfied and

$$\mathbf{P}_j^T \mathbf{G} \mathbf{P}_i = \mathbf{O}, \quad 1 \leq j < i-1. \quad (3.74)$$

Moreover, from equations (1.31), (3.72) and (3.74),

$$\widehat{\mathbf{Q}}_{j-1} \mathbf{G} \mathbf{P}_i = \mathbf{G} \mathbf{P}_i, \quad 1 \leq j \leq i$$

and premultiplying this equation by \mathbf{W}_j^T gives, from equation (3.73) with $i=j$,

$$\mathbf{P}_j^T \mathbf{G} \mathbf{P}_i = \mathbf{W}_j^T \mathbf{G} \mathbf{P}_i, \quad 1 \leq j \leq i.$$

Thus, from semiconjugacy,

$$\mathbf{W}_j^T \mathbf{G} \mathbf{P}_i = \mathbf{O}, \quad 1 \leq j \leq i-1. \quad (3.75)$$

The above analysis is valid for any choice of \mathbf{W}_j . We now consider the Lanczos choice for detailed discussion since the HS analogue is slightly more complicated and relates to the Lanczos version in the same way as it does if \mathbf{G} is symmetric. The Lanczos choice is

$$\mathbf{W}_j = \mathbf{K} \mathbf{G} \mathbf{P}_{j-1} \quad (3.76)$$

where neither \mathbf{G} nor \mathbf{K} are assumed to be symmetric. Equation (3.75) with j replaced by $j+1$ yields immediately

$$\mathbf{P}_j^T \mathbf{G}^T \mathbf{K}^T \mathbf{G} \mathbf{P}_i = \mathbf{O}, \quad 1 \leq j \leq i-2,$$

or, since $\mathbf{P}_j^T \mathbf{G} \mathbf{P}_i = \mathbf{O}$ for $j < i$,

$$\mathbf{P}_j^T (\alpha_0 \mathbf{I} + \alpha_1 \mathbf{G}^T \mathbf{K}^T) \mathbf{G} \mathbf{P}_i = \mathbf{O}, \quad 1 \leq j \leq i-2, \quad (3.77)$$

for any arbitrary scalars α_0 and α_1 .

Define now $\overline{\mathbf{Y}}_i$ by

$$\overline{\mathbf{Y}}_i = \overline{\mathbf{P}}_i \overline{\mathbf{L}}_i^{-1} \quad (3.78)$$

and let

$$\overline{\mathbf{Y}}_i = [\mathbf{Y}_1 \ \mathbf{Y}_2 \ \dots \ \mathbf{Y}_i] \quad (3.79)$$

with the same partitioning for $\bar{\mathbf{Y}}_i$ as for $\bar{\mathbf{P}}_i$ so that, from equation (1.31), equation (3.72) may be written

$$\hat{\mathbf{Q}}_i^T = \mathbf{I} - \sum_{j=1}^i \mathbf{Y}_j \mathbf{P}_j^T \mathbf{G}. \quad (3.80)$$

Equations (3.76) and (3.80) then give

$$\mathbf{P}_{i+1} = \mathbf{K} \mathbf{G} \mathbf{P}_i - \sum_{j=1}^i \mathbf{Y}_j \mathbf{P}_j^T (\mathbf{G} \mathbf{K}) \mathbf{G} \mathbf{P}_i \quad (3.81)$$

and comparison of this equation with equation (3.77) shows that all terms except the last two of the sum in equation (3.81) vanish if

$$\mathbf{G} \mathbf{K} = \alpha_0 \mathbf{I} + \alpha_1 \mathbf{G}^T \mathbf{K}^T \quad (3.82)$$

for any α_0 and α_1 . Since, in equation (3.78), $\bar{\mathbf{L}}_i^{-1}$ is block lower triangular it follows that if equation (3.82) is satisfied the calculation of \mathbf{P}_{i+1} only involves \mathbf{P}_i and \mathbf{P}_{i-1} , precisely as in the case where both \mathbf{G} and \mathbf{K} are symmetric. Thus we may write equation (3.81) as

$$\mathbf{P}_{i+1} = \mathbf{K} \mathbf{G} \mathbf{P}_i - \mathbf{P}_i \mathbf{C}_i - \mathbf{P}_{i-1} \mathbf{D}_{i-1}$$

where, if $\mathbf{P}_i \in \mathbb{R}^{n \times r}$, $\mathbf{C}_i, \mathbf{D}_i \in \mathbb{R}^{r \times r}$ and are chosen so that

$$\mathbf{P}_i^T \mathbf{G} \mathbf{P}_{i+1} = \mathbf{P}_{i-1}^T \mathbf{G} \mathbf{P}_{i+1} = \mathbf{O}.$$

Note that this does *not* give equation (3.1) since, in general, $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_{i-1} \neq \mathbf{O}$ but it will result in a 3-term recursion, albeit a somewhat more complicated one than (3.1).

The comparable analysis for the HS algorithm follows along similar lines but with the equation

$$\mathbf{W}_j^T \mathbf{F}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i,$$

replacing equation (3.75). Since, for the HS version, $\mathbf{W}_j = \mathbf{K} \mathbf{F}_j$ this yields

$$\mathbf{F}_j^T \mathbf{K}^T \mathbf{F}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i,$$

so that from equation (1.37) with $i = j$, and supposing $\mathbf{F}_j^T \mathbf{P}_j$ to be nonsingular,

$$\mathbf{P}_j^T \mathbf{G}^T \mathbf{K}^T \mathbf{F}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i-1.$$

This equation then becomes, from equation (3.70),

$$\mathbf{P}_j^T (\alpha_0 \mathbf{I} + \alpha_1 \mathbf{G}^T \mathbf{K}^T) \mathbf{F}_{i+1} = \mathbf{O}, \quad 1 \leq j \leq i-1, \quad (3.83)$$

where again α_0 and α_1 are arbitrary. Finally, since $\mathbf{P}_{i+1} = \hat{\mathbf{Q}}_i^T \mathbf{K} \mathbf{F}_{i+1}$, replacing $\mathbf{G} \mathbf{P}_i$ by \mathbf{F}_{i+1} in equation (3.81) gives

$$\mathbf{P}_{i+1} = \mathbf{K} \mathbf{F}_{i+1} - \sum_{j=1}^i \mathbf{Y}_j \mathbf{P}_j^T (\mathbf{G} \mathbf{K}) \mathbf{F}_{i+1} \quad (3.84)$$

so that if equation (3.82) is satisfied it follows from equation (3.79) and (3.83) that only the last term of the sum in equation (3.84) is nonzero giving a 2-term recurrence which, in this case, is in fact the same as equation (3.5). Thus for both the Lanczos and the HS forms of the algorithm, equation (3.82) is a sufficient condition for the existence of short recursions, 3-term in the case of Lanczos and 2-term for HS.

We now consider briefly how this equation may be satisfied. Clearly if both \mathbf{G} and \mathbf{K} are symmetric or skew-symmetric it is satisfied if $\alpha_0 = 0$ and $\alpha_1 = \pm 1$ as appropriate, but other possibilities also present themselves. If $\mathbf{G} \neq \mathbf{G}^T$ and $\mathbf{K} = (\mathbf{G} + \mathbf{G}^T)^{-1}$ so that $(\mathbf{G} + \mathbf{G}^T) \mathbf{K} = \mathbf{I}$ then it is satisfied for $\alpha_0 = 1$ and $\alpha_1 = -1$, and putting $\mathbf{G} = \mathbf{A}$ then gives essentially the CGW algorithm. However the increased generality of equation (3.82) over the simple symmetry of \mathbf{G} and \mathbf{K} has not led to a spate of new algorithms, so its value must be regarded as largely theoretical.

A sufficient condition having the form of equation (3.82) was first given (in Russian) by Voevodin [249] in 1979. Voevodin considered the vector equation $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{C}^{n \times n}$ and is nonsingular, and updated the approximate solution \mathbf{x}_i by

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{Bs}_{i+1} \alpha_{i+1}$$

for some nonsingular matrix \mathbf{B} and appropriate choice of α_{i+1} so that

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{Ab}_{i+1} \alpha_{i+1}.$$

The vector \mathbf{s}_{i+1} is given by

$$\mathbf{s}_{i+1} = \mathbf{r}_i + \sum_{j=1}^i \mathbf{s}_j \beta_{j,i+1} \quad (3.85)$$

where the coefficients $\beta_{j,i+1}$ are chosen so that $\mathbf{s}_j^T \mathbf{CABs}_i = 0$ for $j < i$ and some nonsingular matrix \mathbf{C} . A comparison of Voevodin's equations with our own establishes that, if all matrices are real, our \mathbf{G} and \mathbf{K} are given by $\mathbf{G} = \mathbf{B}^{-T} \mathbf{CA}$ and $\mathbf{K} = \mathbf{BC}^{-1} \mathbf{B}^T$ so that

$$\mathbf{KG} = \mathbf{BA} \quad (3.86)$$

and

$$\mathbf{B}^T \mathbf{GKB}^{-T} = \mathbf{CABC}^{-1}. \quad (3.87)$$

Now Voevodin's sufficient condition is given in [249] as

$$(\mathbf{CABC}^{-1})^H = \alpha \mathbf{I} + \beta \mathbf{AB}, \quad (3.88)$$

and for real matrices this becomes, from equations (3.86) and (3.87),

$$\left(\mathbf{B}^T \mathbf{G} \mathbf{K} \mathbf{B}^{-T}\right)^T = \alpha \mathbf{I} + \beta \mathbf{B}^{-1} \mathbf{K} \mathbf{G} \mathbf{B}$$

or, on transposition,

$$\mathbf{B}^T \mathbf{G} \mathbf{K} \mathbf{B}^{-T} = \alpha \mathbf{I} + \beta \mathbf{B}^T \mathbf{G}^T \mathbf{K}^T \mathbf{B}^{-T}.$$

This transforms directly to our equation (3.82) with $\alpha = \alpha_0$ and $\beta = \alpha_1$.

The necessity of equation (3.88) for the existence of short recurrences was subsequently proved by Voevodin and Tyrtyshnikov [251] (also in Russian) in 1981. In fact they gave

$$(\mathbf{C} \mathbf{A} \mathbf{B} \mathbf{C}^{-1})^H = \sum_{j=0}^k \alpha_j (\mathbf{A} \mathbf{B})^j. \quad (3.89)$$

as the necessary and sufficient condition for replacing equation (3.85) by

$$\mathbf{s}_{i+1} = \mathbf{r}_i + \sum_{j=i-k+1}^i \mathbf{s}_j \beta_{j,i+1}$$

(so that there are at most k terms in the sum, where k is some fixed positive integer).

In the analysis of Voevodin and Tyrtyshnikov, no assumptions are made about the nature of the matrices \mathbf{A} , \mathbf{B} and \mathbf{C} beyond their nonsingularity. Of course the algorithm can break down if either \mathbf{G} or \mathbf{K} (essentially \mathbf{CAB} or \mathbf{C} in their notation) is indefinite but in the absence of breakdown the length of the recursion depends neither on the symmetry nor the definiteness of \mathbf{CAB} and \mathbf{C} . Sufficient is nonsingularity and the satisfaction of equation (3.89).

Similar conditions for short recurrences were also given by Faber and Manteuffel [106] in 1984 but in their exposition one or two key ideas were only defined implicitly. These ideas were subsequently elaborated and made explicit by Ashby, Manteuffel and Saylor [7] in 1990 who set out the following definitions:

Definition 2. Let $\mathbf{B} \in \mathbb{C}^{n \times n}$ be Hermitian positive definite, i.e. let $\mathbf{B} = \mathbf{B}^H$ and $\mathbf{x}^H \mathbf{B} \mathbf{x} > 0$ for all $\mathbf{x} \neq 0$. Then

(a) The \mathbf{B} -adjoint $\widehat{\mathbf{M}}$ of $\mathbf{M} \in \mathbb{C}^{n \times n}$ is defined by

$$\widehat{\mathbf{M}} = \mathbf{B}^{-1} \mathbf{M}^H \mathbf{B}, \quad (3.90)$$

- (b) \mathbf{M} is said to be \mathbf{B} -self-adjoint if $\mathbf{M} = \widehat{\mathbf{M}}$, and
(c) \mathbf{M} is said to be \mathbf{B} -normal if $\mathbf{M} \widehat{\mathbf{M}} = \widehat{\mathbf{M}} \mathbf{M}$.

These definitions, together with equation (3.91), are generalisations of results appearing in Gantmacher [125], p. 272, pertaining to conventional normal matrices. The latter may be retrieved from the above by setting \mathbf{B} equal to the unit matrix.

Now it was shown in [106] that \mathbf{M} is B-normal if and only if

$$\widehat{\mathbf{M}} = \sum_{j=0}^k \alpha_j \mathbf{M}^j$$

for some finite positive integer k or, from equation (3.90),

$$\mathbf{M}^H = \widehat{\mathbf{B}} \mathbf{M} \widehat{\mathbf{B}}^{-1} = \sum_{j=0}^k \alpha_j (\mathbf{B} \mathbf{M} \mathbf{B}^{-1})^j. \quad (3.91)$$

If s denotes the smallest value of k for which this equation holds then \mathbf{M} is said to be *B-normal*(s). Ashby *et al* also generalised the analysis of Faber and Manteuffel to include preconditioning. In common with Voevodin they used three matrices to describe the algorithm: the *system matrix* \mathbf{A} , the *inner-product matrix* \mathbf{B} (assumed to be Hermitian positive definite) and the *preconditioning matrix* \mathbf{C} . They then showed that a necessary and sufficient condition for an s -term recursion of Lanczos type is that the *preconditioned system matrix* \mathbf{CA} is B-normal($s-2$). Comparison of their equations for computing \mathbf{x}_{i+1} and \mathbf{p}_{i+1} with our own show that their \mathbf{A} , \mathbf{B} and \mathbf{C} are related to our \mathbf{G} and \mathbf{K} by $\mathbf{G} = \mathbf{B}$ and $\mathbf{K} = \mathbf{CAB}^{-1}$, so that $\mathbf{CA} = \mathbf{KG}$ and $\mathbf{BCAB}^{-1} = \mathbf{GK}$. Their necessary and sufficient condition for an s -term recursion thus becomes, in our notation and from equation (3.91),

$$(\mathbf{KG})^H = \sum_{j=0}^{s-2} \alpha_j (\mathbf{GK})^j$$

where the coefficients α_j are arbitrary. Transposing and taking complex conjugates yields, if the scalars α_j are real,

$$\mathbf{KG} = \sum_{j=0}^{s-2} \alpha_j (\mathbf{K}^H \mathbf{G}^H)^j$$

and pre-multiplying this equation by \mathbf{G} and post-multiplying by \mathbf{G}^{-1} gives, since $\mathbf{G} = \mathbf{B}$ and is assumed to be Hermitian,

$$\mathbf{GK} = \sum_{j=0}^{s-2} \alpha_j (\mathbf{G}^H \mathbf{K}^H)^j, \quad (3.92)$$

which reduces to equation (3.82) if $s = 3$ and both \mathbf{G} and \mathbf{K} are real.

The establishment of necessary and sufficient conditions for the existence of short recurrences has given a considerable additional insight into CG-type methods but this new understanding has not yet resulted in many, if any, useful new algorithms. Indeed it is somewhat ironic that one of the more popular methods for solving $\mathbf{Ax} = \mathbf{b}$ for a general matrix \mathbf{A} is based on a long recurrence which often uses a somewhat less-than-elegant rule of thumb to determine when it should be restarted. Because, though, this method and the original CG method have proved so successful

much work has been done to enhance our understanding of how they function. This better understanding is the principal motive underlying the next chapter.

Chapter 4

The Krylov aspects

In this chapter we look at the underlying structure of the methods discussed above, and show how they are related to Krylov sequences. We do this in order to analyse the convergence and other properties of the algorithms and also to prepare the ground for the discussion of the algorithms in Chapter 5.

We know that the generating matrices of Chapter 2, \mathbf{W}_i , are equal to either \mathbf{KGP}_{i-1} in the case of GODir or \mathbf{KF}_i in the case of GOMin, but because the properties of these matrices are not straightforward it is not possible from this knowledge alone to obtain bounds for the convergence rates of the algorithms. If we seek such bounds we need some simpler forms for these matrices and we show in this chapter that for both GODir and GOMin, \mathbf{W}_i is effectively given by $(\mathbf{KG})^{i-1}\mathbf{P}_1$. It is this property that links these methods both to one another and to the Krylov sequences.

A (vector) Krylov sequence is defined to be the sequence of vectors $\{\mathbf{B}^i \mathbf{p}_1\}$, $i = 0, 1, \dots$, where \mathbf{p}_1 is some initial vector and $\mathbf{B} \in \mathbb{R}^{n \times n}$ is some matrix. The matrix version of this is defined by $\{\mathbf{B}^i \mathbf{P}_1\}$, where $\mathbf{P}_1 \in \mathbb{R}^{n \times r}$ is some initial matrix. In particular we will be concerned with the properties of the three matrices $\widehat{\mathbf{P}}_i$, $\widehat{\mathbf{S}}_i$ and $\widehat{\mathbf{T}}_i$, where

$$\widehat{\mathbf{P}}_i = [\mathbf{P}_1 \ \mathbf{KGP}_1 \ (\mathbf{KG})^2 \mathbf{P}_1 \dots (\mathbf{KG})^{i-1} \mathbf{P}_1] \quad (4.1)$$

(the Krylov matrix),

$$\widehat{\mathbf{S}}_i = \widehat{\mathbf{P}}_i^T \mathbf{G} \widehat{\mathbf{P}}_i \quad (4.2)$$

and

$$\widehat{\mathbf{T}}_i = \widehat{\mathbf{P}}_i^T \mathbf{K}^{-1} \widehat{\mathbf{P}}_i. \quad (4.3)$$

We assume throughout that \mathbf{K} is nonsingular although it is not necessarily always assumed to be symmetric.

Our first theorem shows that for GODir the matrices \mathbf{W}_i of Chapter 2 may be taken to be the matrices $(\mathbf{KG})^{i-1} \mathbf{P}_1$, where \mathbf{P}_1 is the normal starting matrix for

GODir, and gives necessary and sufficient conditions for the algorithm to avoid breakdown.

Theorem 16. Let the matrices $\widehat{\mathbf{S}}_j$ be defined by equation (4.2) and be nonsingular for $1 \leq j \leq i$. Then for $1 \leq j \leq i+1$ the sequence of matrices $\{\mathbf{P}_j\}$ is computable by GODir and the matrices $\widehat{\mathbf{P}}_j$ satisfy

$$\widehat{\mathbf{P}}_j = \widehat{\mathbf{P}}_j \widehat{\mathbf{U}}_j \quad (4.4)$$

for some block unit upper triangular (and hence nonsingular) matrix $\widehat{\mathbf{U}}_j$.

Proof. By induction. We show that if the theorem is true for $1 \leq j \leq i$, it is also true for $j = i+1$.

Since $\widehat{\mathbf{S}}_i$ is nonsingular and equation (4.4) is valid for $j = i$ with $\widehat{\mathbf{U}}_i$ nonsingular, both by hypothesis, it follows from Lemma 6 that \mathbf{Q}_i is computable and may be written

$$\mathbf{Q}_i = \mathbf{I} - \mathbf{G}\widehat{\mathbf{P}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{P}}_i^T. \quad (4.5)$$

Consider now equation (4.4) and denote submatrices of $\widehat{\mathbf{U}}_i$ by \mathbf{U}_{rs} . From equations (1.31), (4.1) and (4.4) and with obvious partitioning,

$$\mathbf{P}_i = \sum_{j=1}^i (\mathbf{KG})^{j-1} \mathbf{P}_1 \mathbf{U}_{ji}$$

so that

$$\mathbf{KGP}_i = \sum_{j=1}^i (\mathbf{KG})^j \mathbf{P}_1 \mathbf{U}_{ji}.$$

Now for GODir $\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{KGP}_i$, and from equations (4.1) and (4.5) we have $\mathbf{Q}_i^T (\mathbf{KG})^j \mathbf{P}_1 = \mathbf{O}$ for $0 \leq j \leq i-1$ so that

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T (\mathbf{KG})^i \mathbf{P}_1 \quad (4.6)$$

since $\mathbf{U}_{ii} = \mathbf{I}$. The same two equations yield

$$\mathbf{P}_{i+1} = \widehat{\mathbf{P}}_{i+1} \begin{bmatrix} \widehat{\mathbf{Y}}_{i+1} \\ \mathbf{I} \end{bmatrix} \quad (4.7)$$

for some $\widehat{\mathbf{Y}}_{i+1} \in \mathbb{R}^{ri \times r}$. If we now define $\widehat{\mathbf{U}}_{i+1}$ by

$$\widehat{\mathbf{U}}_{i+1} = \begin{bmatrix} \widehat{\mathbf{U}}_i & \widehat{\mathbf{Y}}_{i+1} \\ \mathbf{O} & \mathbf{I} \end{bmatrix}$$

it follows that equation (4.4) holds for $j = i+1$ and that $\widehat{\mathbf{U}}_{i+1}$ satisfies the required conditions, establishing the induction. The theorem is trivially true for $i = 1$ ($\widehat{\mathbf{U}}_1 = \mathbf{U}_{11} = \mathbf{I}$) so is true for $1 \leq j \leq i+1$, concluding the proof. ■

The importance of this theorem is twofold. Firstly it shows that the columns of the matrices $\mathbf{P}_1, \mathbf{KG}\mathbf{P}_1, (\mathbf{KG})^2\mathbf{P}_1, \dots$ form a basis for the vector space defined by the columns of the matrices $\mathbf{P}_1, \mathbf{P}_2, \dots$ generated by GODir. In so doing it establishes the fundamental property of these methods, namely their connection with the Krylov sequence. It is this connection that enables many of the properties of GODir to be simply deduced and which determines their basic character. Secondly the theorem yields one of the more fundamental of these properties since it gives us the necessary and sufficient conditions for GODir to be breakdown-free, namely that the matrices $\widehat{\mathbf{S}}_i$ should be nonsingular for all i . Since $\widehat{\mathbf{S}}_i$ depends only upon the initial matrix \mathbf{P}_1 (or vector \mathbf{p}_1) and the matrix product \mathbf{KG} , the analysis of the causes of breakdown is very much simplified, and we now briefly consider this.

There are essentially two causes of breakdown. If \mathbf{G} is indefinite and $\widehat{\mathbf{S}}_i$ is singular while $\widehat{\mathbf{P}}_i$ has full rank then serious breakdown has occurred and the algorithm crashes. This difficulty may be overcome by look-ahead techniques if, for some $j > i$, $\widehat{\mathbf{S}}_j$ is nonsingular (see Chapter 7). It is evident from this that serious breakdown can never occur if \mathbf{G} is positive definite, and algorithms for which this is true do not crash though they may stagnate. If, on the other hand, $\widehat{\mathbf{S}}_j$ is singular for all $j \geq i$ then we have incurable breakdown for which, at the time of writing, there is no satisfactory remedy.

The other type of breakdown occurs when $\widehat{\mathbf{P}}_i$ becomes rank-deficient and where the nullity of $\widehat{\mathbf{S}}_i$ is equal to the column nullity of $\widehat{\mathbf{P}}_i$. If this happens a partial solution of the problem may be determined and the algorithm permitted to continue with a reduced number of columns in the matrices \mathbf{P}_i . There are various reasons why $\widehat{\mathbf{P}}_i$ should become rank-deficient. The most obvious one arises when $\widehat{\mathbf{P}}_i$ has more columns than rows. Termination of the algorithm due to this would be the normal termination in the absence of rounding error and would yield the exact solution of the problem after n/r iterations where n is the order of \mathbf{G} , r is the number of columns of \mathbf{P}_i and n/r is assumed to be an integer. However it is possible for $\widehat{\mathbf{P}}_i$ to become rank-deficient after far fewer iterations than n/r , and the reasons for this are bound up with the eigenvectors of \mathbf{KG} . For simplicity we consider only vector sequences ($r = 1$) so that

$$\widehat{\mathbf{P}}_i = [\mathbf{p}_1 \ \mathbf{KG}\mathbf{p}_1 \ (\mathbf{KG})^2\mathbf{p}_1 \ \dots \ (\mathbf{KG})^{i-1}\mathbf{p}_1]. \quad (4.8)$$

Assume that $\widehat{\mathbf{P}}_i$ has full column rank but that

$$\widehat{\mathbf{P}}_{i+1}\mathbf{a} = \mathbf{0} \quad (4.9)$$

for some $\mathbf{a} \neq \mathbf{0}$. If $\mathbf{a}^T = [\alpha_0, \alpha_1, \dots, \alpha_i]$, equation (4.9) may be written

$$\left[\sum_{j=0}^i \alpha_j (\mathbf{KG})^j \right] \mathbf{p}_1 = \mathbf{0} \quad (4.10)$$

where $\alpha_i \neq 0$ since if $\alpha_i = 0$, $\widehat{\mathbf{P}}_i\mathbf{a}_1 = \mathbf{0}$ for some $\mathbf{a}_1 \neq \mathbf{0}$, contrary to hypothesis. Thus \mathbf{a} may be normalised so that $\alpha_i = 1$ and with this normalisation equation (4.10) may be written

$$\prod_{j=1}^i (\mathbf{KG} - \lambda_j \mathbf{I}) \mathbf{p}_1 = \mathbf{0} \quad (4.11)$$

where the scalars λ_j are the roots of the equation

$$\sum_{j=0}^i \alpha_j \lambda^j = 0.$$

Each factor $(\mathbf{KG} - \lambda_j \mathbf{I})$ is singular since if one, $(\mathbf{KG} - \lambda_k \mathbf{I})$ say, were not, equation (4.11) could be premultiplied by $(\mathbf{KG} - \lambda_k \mathbf{I})^{-1}$ and expanding the resulting product would imply that $\widehat{\mathbf{P}}_i \mathbf{b} = \mathbf{0}$ for some $\mathbf{b} \neq \mathbf{0}$, again contrary to hypothesis.

Now equation (4.11) is satisfied if and only if \mathbf{p}_1 is a linear combination of the eigenvectors of \mathbf{KG} (or principal vectors, since even if \mathbf{K} is symmetric the symmetry of \mathbf{K} and \mathbf{G} does not guarantee the symmetry of their product) corresponding to the eigenvalues λ_j of equation (4.11). Thus if \mathbf{p}_1 is a linear combination of only a few such eigenvectors then $\widehat{\mathbf{P}}_i$ will become rank-deficient for $i \ll n$ and rapid termination will ensue. A similar situation occurs when the degree of the minimal polynomial of \mathbf{KG} is very much less than n (the minimal polynomial of a matrix is the polynomial in that matrix of least degree which is null). If the degree of this polynomial is m and

$$\sum_{j=0}^m \alpha_j (\mathbf{KG})^j = \mathbf{O}$$

for some set of coefficients α_j , the algorithm will terminate for *any* choice of \mathbf{p}_1 in at most m iterations. Thus one criterion for the choice of the arbitrary preconditioning matrix \mathbf{K} would be the reduction of the degree of the minimal polynomial of \mathbf{KG} . If $\mathbf{K} = \mathbf{G}^{-1}$, the ultimate preconditioner, the minimal polynomial would be linear and the algorithm would terminate after a single step.

The degree of the minimal polynomial is at least as great as the number of distinct eigenvalues of \mathbf{KG} (with equality if \mathbf{KG} is symmetric) so one way of reducing the degree might be to reduce the number of distinct eigenvalues, even though this is a necessary rather than a sufficient requirement for degree reduction in the general case where \mathbf{KG} is not symmetric.

We note finally that the nonsingularity of the matrices $\widehat{\mathbf{S}}_i$ is the *only* condition that governs the robustness of the method GODir if exact arithmetic is assumed. In this respect it differs markedly from GOMin, for which other factors come into play, and to understand what these might be we now prove a companion theorem to the previous one that relates to GOMin.

Theorem 17. Let the matrices $\widehat{\mathbf{S}}_j$ be defined by equation (4.2) and be nonsingular for $1 \leq j \leq i$. If the matrices $\mathbf{P}_j^T \mathbf{F}_j$ computed by GOMin are nonsingular for $1 \leq j \leq i$ then for $1 \leq j \leq i+1$ the sequences of matrices $\{\mathbf{F}_j\}$ and $\{\mathbf{P}_j\}$ are computable by GOMin and the matrices $\overline{\mathbf{P}}_j$ satisfy

$$\overline{\mathbf{P}}_j = \widehat{\mathbf{P}}_j \widehat{\mathbf{U}}_j \quad (4.12)$$

for some nonsingular block upper triangular matrix $\widehat{\mathbf{U}}_j$.

Proof. By induction. We show that if the theorem is true for $1 \leq j \leq i$, it is also true for $j = i + 1$.

Since $\widehat{\mathbf{S}}_i$ is nonsingular and equation (4.12) is valid for $j = i$ with $\widehat{\mathbf{U}}_i$ nonsingular, both by hypothesis, it follows from Lemma 6 that \mathbf{Q}_i is computable and is given by equation (4.5). It further follows from equation (1.45) and the computability of \mathbf{Q}_i that the matrices \mathbf{D}_j , $1 \leq j \leq i$, are nonsingular so that equation (1.37) is valid and may be written

$$\mathbf{F}_{i+1} = \mathbf{F}_i + \mathbf{G}\mathbf{P}_i\mathbf{M}_i$$

where

$$\mathbf{M}_i = -\mathbf{D}_i^{-1}\mathbf{P}_i^T\mathbf{F}_i \quad (4.13)$$

and is nonsingular since $\mathbf{P}_i^T\mathbf{F}_i$ is nonsingular by hypothesis. Now for GOMin,

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T\mathbf{K}\mathbf{F}_{i+1} = \mathbf{Q}_i^T\mathbf{K}(\mathbf{F}_i + \mathbf{G}\mathbf{P}_i\mathbf{M}_i).$$

Since, from equation (1.50), $\mathbf{Q}_i^T = \mathbf{Q}_i^T\mathbf{Q}_{i-1}^T$ we have

$$\mathbf{Q}_i^T\mathbf{K}\mathbf{F}_i = \mathbf{Q}_i^T\mathbf{Q}_{i-1}^T\mathbf{K}\mathbf{F}_i = \mathbf{Q}_i^T\mathbf{P}_i = \mathbf{O}$$

so that, for GOMin,

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T\mathbf{K}\mathbf{G}\mathbf{P}_i\mathbf{M}_i.$$

A similar argument to that used to establish equation (4.6) then yields

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T(\mathbf{K}\mathbf{G})^i\mathbf{P}_1\mathbf{U}_{ii}\mathbf{M}_i \quad (4.14)$$

(since we may no longer assume that $\mathbf{U}_{ii} = \mathbf{I}$). Equations (4.1) and (4.5) give

$$\mathbf{P}_{i+1} = \widehat{\mathbf{P}}_{i+1} \begin{bmatrix} \widehat{\mathbf{Y}}_{i+1} \\ \mathbf{U}_{i+1,i+1} \end{bmatrix}$$

for some matrix $\widehat{\mathbf{Y}}_{i+1} \in \mathbb{R}^{ri \times r}$ and for

$$\mathbf{U}_{i+1,i+1} = \mathbf{U}_{ii}\mathbf{M}_i. \quad (4.15)$$

Since \mathbf{M}_i and \mathbf{U}_{ii} are both nonsingular by hypothesis it follows that $\mathbf{U}_{i+1,i+1}$ is also nonsingular. The same arguments used to complete the proof of Theorem 16 now conclude the proof. ■

Corollary 18. If the conditions of Theorem 17 are satisfied then

$$\mathbf{U}_{i+1,i+1} = \mathbf{M}_1\mathbf{M}_2 \dots \mathbf{M}_i \quad (4.16)$$

Proof. From equation (4.15) and the fact that $\mathbf{U}_{11} = \mathbf{I}$. ■

From this theorem and the previous one, and from Lemma 6, we deduce that if \mathbf{P}_1 , \mathbf{G} and \mathbf{K} are the same for GOMin and GODir then the projection matrices \mathbf{Q}_j

generated by both algorithms are identical. The sequences of approximate solutions $\{\mathbf{X}_i\}$ are also identical and the matrices \mathbf{P}_i , although not the same, define identical subspaces. In exact arithmetic the only effective difference between the two algorithms occurs if, for some i , $\mathbf{P}_i^T \mathbf{F}_i$ is singular. When this happens it follows from equations (4.13) and (4.14) that \mathbf{M}_i is singular and \mathbf{P}_{i+1} is rank-deficient. The algorithm then breaks down when computing \mathbf{X}_{i+2} . Moreover, from equation (4.16), \mathbf{U}_{jj} is singular for all $j > i$, making recovery difficult. In practice when $\mathbf{P}_i^T \mathbf{F}_i$ becomes nearly singular the algorithm *stagnates*, with successive values of \mathbf{X}_i being nearly the same.

Now the theoretical analysis of stagnation is difficult if based on the matrices $\mathbf{P}_i^T \mathbf{F}_i$ since the properties of these matrices are insufficiently fundamental for detailed analysis. Our next theorem shows, however, that if $\widehat{\mathbf{T}}_i$ is defined by equation (4.3) then $\mathbf{P}_i^T \mathbf{F}_i$ is singular if and only if $\widehat{\mathbf{T}}_i$ is singular, enabling us once again to relate the stability of one of our principal algorithms to the properties of the original Krylov sequence.

Theorem 19. Let the matrices $\widehat{\mathbf{S}}_j$, $1 \leq j \leq i$, be nonsingular. Then the matrices $\mathbf{P}_j^T \mathbf{F}_j$ computed by GOMin are nonsingular if and only if the matrices $\widehat{\mathbf{T}}_j$ are nonsingular.

Proof. By induction. We show that if the theorem is true for $1 \leq j \leq i$ then it is also true for $j = i + 1$.

Let the matrices $\widehat{\mathbf{T}}_j$ be nonsingular for $1 \leq j \leq i$ so that, from the induction hypothesis, the matrices $\mathbf{P}_j^T \mathbf{F}_j$ too are nonsingular for $1 \leq j \leq i$. This implies, from equations (4.13) and (4.16) that $\mathbf{U}_{i+1,i+1}$ is also nonsingular. Now from equations (4.13), (4.14) and (4.15), and the idempotency of \mathbf{Q}_i ,

$$\mathbf{P}_{i+1}^T \mathbf{F}_{i+1} = \mathbf{P}_{i+1}^T \mathbf{Q}_i \mathbf{F}_1 = \mathbf{U}_{i+1,i+1}^T \mathbf{Z}_i$$

where

$$\mathbf{Z}_i = \mathbf{P}_1^T (\mathbf{GK}^T)^i \mathbf{Q}_i \mathbf{F}_1$$

and since $\mathbf{U}_{i+1,i+1}$ is nonsingular, $\mathbf{P}_{i+1}^T \mathbf{F}_{i+1}$ is nonsingular if and only if \mathbf{Z}_i is nonsingular where, from equation (4.5),

$$\mathbf{Z}_i = \mathbf{P}_1^T (\mathbf{GK}^T)^i \mathbf{F}_1 - \mathbf{P}_1^T (\mathbf{GK}^T)^i \mathbf{G} \widehat{\mathbf{P}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{P}}_i^T \mathbf{F}_1. \quad (4.17)$$

Now from equations (4.1) and (4.3) and since $\mathbf{P}_1 = \mathbf{KF}_1$, $\widehat{\mathbf{T}}_{i+1}$ may be expressed as

$$\widehat{\mathbf{T}}_{i+1} = \begin{bmatrix} \widehat{\mathbf{P}}_i^T \\ \mathbf{P}_1^T (\mathbf{GK}^T)^i \end{bmatrix} \mathbf{K}^{-1} \begin{bmatrix} \mathbf{P}_1 & \mathbf{KG} \widehat{\mathbf{P}}_i \end{bmatrix}$$

so that, from equation (4.2) and since $\mathbf{P}_1 = \mathbf{KF}_1$ by imposition,

$$\widehat{\mathbf{T}}_{i+1} = \begin{bmatrix} \widehat{\mathbf{P}}_i^T \mathbf{F}_1 & \widehat{\mathbf{S}}_i \\ \mathbf{P}_1^T (\mathbf{GK}^T)^i \mathbf{F}_1 & \mathbf{P}_1^T (\mathbf{GK}^T)^i \mathbf{G} \widehat{\mathbf{P}}_i \end{bmatrix}. \quad (4.18)$$

From this equation and equation (4.17) it is seen immediately that \mathbf{Z}_i is merely the Schur complement of $\widehat{\mathbf{S}}_i$ in $\widehat{\mathbf{T}}_{i+1}$ so that, from the properties of Schur complements (see Appendix B), \mathbf{Z}_i is nonsingular if and only if $\widehat{\mathbf{T}}_{i+1}$ is nonsingular. Thus $\mathbf{P}_{i+1}^T \mathbf{F}_{i+1}$ is nonsingular if and only if $\widehat{\mathbf{T}}_{i+1}$ is nonsingular, establishing the theorem for $j = i+1$. The theorem is trivially true for $j = 1$ since $\mathbf{P}_1^T \mathbf{F}_1 = \mathbf{P}_1^T \mathbf{K}^{-1} \mathbf{P}_1 = \widehat{\mathbf{T}}_1$, completing the proof. ■

This theorem, together with Theorems 16 and 17, shows that the stability or otherwise of GODir and GOMin is directly connected to the nonsingularity or otherwise of the matrices $\widehat{\mathbf{S}}_i$ and $\widehat{\mathbf{T}}_i$. As long as these are nonsingular GOMin will not terminate prematurely while for GODir the nonsingularity of the matrices $\widehat{\mathbf{S}}_i$ alone is sufficient to guarantee continuation. If vector sequences are being generated and, for some i , \mathbf{p}_{i+1} is null then, from Theorems 11 and 13, GODir and GOMin will both have computed the exact solution. If the matrix sequence $\{\mathbf{P}_i\}$ is being computed where $\mathbf{P}_i \in \mathbb{R}^{n \times r}$, $r > 1$, and $\widehat{\mathbf{P}}_{i+1}$ is rank-deficient then \mathbf{P}_{i+1} will also be rank-deficient and it will be possible to obtain a partial solution (see Chapter 8). If, on the other hand, $\widehat{\mathbf{S}}_i$ or $\widehat{\mathbf{T}}_i$ is singular and $\widehat{\mathbf{P}}_i$ has full column rank then this singularity can only be due to the indefiniteness of either \mathbf{G} or \mathbf{K} , and we have to have recourse to “look-ahead”. It is thus clearly desirable, from the point of view of robustness, that both \mathbf{G} and \mathbf{K} should be definite, and if either or both are not then there is the possibility of serious breakdown or worse. Unfortunately there are very few algorithms where \mathbf{G} and \mathbf{K} are both positive definite. If \mathbf{A} is positive definite then the original CG method of Hestenes and Stiefel is one such, as are the methods CR and MCR, although the success of CG has virtually eliminated these methods from consideration. Other methods for which \mathbf{G} and \mathbf{K} are both positive definite are CGNR and CGNE, but despite their undoubtedly stability their slow convergence makes them uncompetitive (see below, Rates of convergence). For Orthodir, Orthomin and GMRes, \mathbf{G} is always positive definite and if \mathbf{A} is positive real then \mathbf{K} is also positive real. In this case also all three methods are robust.

Methods for which \mathbf{G} is indefinite and \mathbf{K} definite include CG applied to a symmetric indefinite system and the Galerkin algorithms of Hegedüs. All these algorithms may suffer from serious breakdown and may crash in their unmodified forms, but in principle only a single “look-ahead” step is needed to restore them to normality (see Chapter 7). Although this is still an open question, it is likely that the impossibility of incurable breakdown occurring in these algorithms is due to the definiteness of \mathbf{K} .

Algorithms of GOMin type for which \mathbf{G} is definite and \mathbf{K} indefinite and which are therefore prone to stagnation are CR applied to a symmetric indefinite system and BiCR. The avoidance of stagnation in these algorithms is not easy in practice since it is difficult to distinguish stagnation from slow convergence. It is possible to switch to the Lanczos (GODir) form of the algorithm if stagnation occurs, and this has been proposed for CR [60], but at the time of writing very little is known of this aspect of the behaviour of the BiCRL algorithms.

For the BiCG, BiCGI and QMR algorithms both \mathbf{G} and \mathbf{K} are indefinite and for all these algorithms it is recommended in practice to use some form of look-ahead

to overcome the problems of serious breakdown. In contrast to the algorithms where \mathbf{G} is indefinite but \mathbf{K} is definite, incurable breakdown is possible and this may be due to the indefiniteness of \mathbf{K} . However it should be stressed that in practice the likelihood of incurable breakdown is extremely small so the theoretical susceptibility of these algorithms to this problem should not be regarded as a disadvantage. What matters is how they perform in practice, and this is discussed more fully in Chapter 10.

4.1. Equivalent algorithms

For most of the algorithms of Chapters 2 and 3, \mathbf{F}_{i+1} is given by equation (1.43) with \mathbf{Q}_i given by equation (1.46) (see Steps 2(b) and 2(d) of GODir/GOMin, page 32). Thus

$$\mathbf{F}_{i+1} = \mathbf{F}_1 - \mathbf{G}\bar{\mathbf{P}}_i\bar{\mathbf{D}}_i^{-1}\bar{\mathbf{P}}_i^T\mathbf{F}_1 \quad (4.19)$$

so that, since \mathbf{G} is nonsingular by hypothesis and $\mathbf{F} = \mathbf{GX} - \mathbf{H}$,

$$\mathbf{X}_{i+1} = \mathbf{X}_1 + \bar{\mathbf{P}}_i\mathbf{Z}_i \quad (4.20)$$

where

$$\mathbf{Z}_i = -\bar{\mathbf{D}}_i^{-1}\bar{\mathbf{P}}_i^T\mathbf{F}_1. \quad (4.21)$$

This is clearly a generalisation of equation (1.17) and requires only that $\bar{\mathbf{D}}_i = \bar{\mathbf{P}}_i^T\mathbf{G}\bar{\mathbf{P}}_i$ is nonsingular, although if it is computed by either GODir or GOMin it will be either diagonal or block-diagonal as well. Trivially, from equation (4.19), $\bar{\mathbf{P}}_i^T\mathbf{F}_{i+1}$, the equation that guarantees termination, is satisfied.

Now equation (1.17) was derived by defining a linear manifold by equation (1.10) and a function $\mathbf{f}(\mathbf{z})$ on that manifold, and imposing the condition $\mathbf{P}_i^T\mathbf{f}(\mathbf{z}_i) = \mathbf{0}$ (equation (1.11)). Equation (4.20) may be derived similarly. If, by analogy with equation (1.10), $\mathbf{X}(\mathbf{Z})$ is defined by

$$\mathbf{X}(\mathbf{Z}) \equiv \mathbf{X}_1 + \bar{\mathbf{P}}_i\mathbf{Z} \quad (4.22)$$

with $\mathbf{F}(\mathbf{Z})$ given by $\mathbf{F}(\mathbf{Z}) \equiv \mathbf{GX}(\mathbf{Z}) - \mathbf{H}$ we obtain, with the usual notation,

$$\mathbf{F}(\mathbf{Z}) \equiv \mathbf{F}_1 + \mathbf{G}\bar{\mathbf{P}}_i\mathbf{Z} \quad (4.23)$$

and choosing a value \mathbf{Z}_i of \mathbf{Z} so that $\bar{\mathbf{P}}_i^T\mathbf{F}(\mathbf{Z}_i) = \mathbf{0}$ immediately gives equations (4.21) and (4.20). Alternatively we may define the generalised residual $\mathbf{R}(\mathbf{Z})$ by $\mathbf{R}(\mathbf{Z}) \equiv \mathbf{AX}(\mathbf{Z}) - \mathbf{B}$ and choose \mathbf{Z}_i so that $\bar{\mathbf{P}}_i^T\mathbf{A}^T\mathbf{R}(\mathbf{Z}_i) = \mathbf{0}$ by analogy with equation (1.15), giving a “minimum residual” solution. The Krylov methods may thus be thought of as setting up a linear manifold as in equation (4.22) and then selecting as the next approximate solution a value of \mathbf{X} that lies in that manifold and at the same time satisfies some other condition, e.g. a Galerkin condition.

We now invoke Theorems 16 and 17. Define a second linear manifold by

$$\mathbf{X}(\widehat{\mathbf{Z}}) \equiv \mathbf{X}_1 + \widehat{\mathbf{P}}_i \widehat{\mathbf{Z}} \quad (4.24)$$

where $\widehat{\mathbf{P}}_i$ is given by equation (4.1) and $\widehat{\mathbf{Z}}$ is a matrix of independent variables. Since both $\overline{\mathbf{P}}_i$ and $\widehat{\mathbf{P}}_i$ are assumed to have full rank and since, for equation (4.4), $\widehat{\mathbf{U}}_i$ is assumed to be nonsingular it is straightforward to show that $\mathbf{X}(\mathbf{Z}) = \mathbf{X}(\widehat{\mathbf{Z}})$ if and only if $\widehat{\mathbf{Z}} = \widehat{\mathbf{U}}_i \mathbf{Z}$. Thus the manifolds defined in equations (4.22) and (4.24) are identical (since any \mathbf{X} found in one will be found in the other) and depend only on \mathbf{X}_1 , \mathbf{P}_1 and \mathbf{KG} . This fact is sometimes underlined when considering vector sequences by writing the vector form of equation (4.24) as

$$\mathbf{x} \equiv \mathbf{x}_1 + \mathcal{K}(\mathbf{B}, \mathbf{p}_1)$$

where $\widehat{\mathbf{P}}_i$ is given by equation (4.8) and $\mathbf{B} = \mathbf{KG}$ is the matrix whose powers generate the Krylov sequence.

But this is not all. If equation (4.4) is satisfied then the imposition of a Galerkin or similar condition results in the identical value of \mathbf{X} being obtained regardless of which equation is used to define the linear manifold. As may readily be verified, the imposition of each of the four conditions $\overline{\mathbf{P}}_i^T \mathbf{F}(\mathbf{Z}) = \mathbf{O}$, $\overline{\mathbf{P}}_i^T \mathbf{F}(\widehat{\mathbf{Z}}) = \mathbf{O}$, $\widehat{\mathbf{P}}_i^T \mathbf{F}(\mathbf{Z}) = \mathbf{O}$ and $\widehat{\mathbf{P}}_i^T \mathbf{F}(\widehat{\mathbf{Z}}) = \mathbf{O}$ yields the same value of \mathbf{X} in every case since if $\widehat{\mathbf{P}}_i \widehat{\mathbf{U}}_i$ is substituted for $\overline{\mathbf{P}}_i$ in equation (4.23) and any one of the above four conditions is imposed, the matrices $\widehat{\mathbf{U}}_i$ cancel out. Thus if two apparently quite different algorithms are based on the same Krylov sequence and the same condition of the type $\overline{\mathbf{P}}_i^T \mathbf{F}(\mathbf{Z}) = \mathbf{O}$ is imposed, the algorithms will generate identical sequences $\{\mathbf{X}_i\}$.

It is not perhaps surprising that the Lanczos and HS versions of the same algorithm generate (in exact arithmetic and in the absence of breakdown) identical sequences of approximate solutions. The algorithms are similar and from Lemma 6 and equation (4.4) generate identical sequences of projection matrices $\{\mathbf{Q}_j\}$. Thus, from equations (4.6), (4.14) and (4.15), the displacement matrices $\{\mathbf{P}_i\}$ satisfy

$$\mathbf{P}_i^{HS} = \mathbf{P}_i^{Lan} \mathbf{U}_{i+1,i+1}. \quad (4.25)$$

Since, for both versions, \mathbf{X}_{i+1} is computed by equation (3.12) it follows that if \mathbf{X}_i (and hence \mathbf{F}_i) is the same for both versions then \mathbf{X}_{i+1} will also be the same regardless of whether in equation (3.12) \mathbf{P}_i is set equal to \mathbf{P}_i^{HS} or \mathbf{P}_i^{Lan} . This follows trivially from equation (4.25) since substituting $\mathbf{P}_i^{Lan} \mathbf{U}_{i+1,i+1}$ for \mathbf{P}_i^{HS} in equation (3.12) results in the upper triangular matrices cancelling. The only time this fails is if $\mathbf{U}_{i+1,i+1}$ is singular and this, from equations (4.16) and (4.13) and Theorem 19, only occurs if $\widehat{\mathbf{T}}_i$ is singular.

Thus the HS and Lanczos versions of the same algorithm are equivalent but two algorithms A and B may also be equivalent, in the sense that they generate identical sequences of approximate solutions $\{\mathbf{X}_i\}$, even if the displacement matrices

\mathbf{P}_i do not satisfy $\mathbf{P}_i^A = \mathbf{P}_i^B \mathbf{M}_i$ for $i \geq 2$ and some nonsingular matrix \mathbf{M}_i . Consider algorithms Orthodir, GCR and GMRes. Orthodir and GCR are effectively the Lanczos and HS versions of the same algorithm. For these algorithms $\mathbf{G} = \mathbf{A}^T \mathbf{A}$, $\mathbf{K} = \mathbf{A}^{-T}$ and $\mathbf{f} \equiv \mathbf{A}^T \mathbf{Ax} - \mathbf{A}^T \mathbf{b} \equiv \mathbf{A}^T \mathbf{r}$ where the residual \mathbf{r} is defined as usual to be $\mathbf{Ax} - \mathbf{b}$. Thus $\mathbf{KG} = \mathbf{A}$ and $\mathbf{p}_1 = \mathbf{Kf}_1 = \mathbf{r}_1$ by imposition. The displacement vectors \mathbf{p}_i are mutually conjugate with respect to $\mathbf{A}^T \mathbf{A}$ and the Krylov matrix, as defined by equation (4.8), is given by

$$\widehat{\mathbf{P}}_i = [\mathbf{r}_1, \mathbf{Ar}_1, \dots, \mathbf{A}^{i-1} \mathbf{r}_1]. \quad (4.26)$$

GMRes, on the other hand, generates orthonormal vectors \mathbf{p}_j using the original Arnoldi algorithm (see page 28) so that $\mathbf{G} = \mathbf{I}$. For this algorithm $\mathbf{p}_1 = \mathbf{r}_1 / \|\mathbf{r}_1\|$ and $\mathbf{K} = \mathbf{A}$ and its Krylov matrix is just that of Orthodir and GCR scaled by $1 / \|\mathbf{r}_1\|$. Thus the Krylov matrices of all three algorithms are effectively identical and since each minimises $\|\mathbf{r}\|$ over the same linear manifold, it generates the same sequence of approximate solutions.

These three algorithms are not the only ones to show this sort of equivalence. The following table gives the values of $\mathbf{KG} = \mathbf{B}$ (which determines the particular subspace) and the imposed criteria for the second part of the algorithm (Ritz-Galerkin, minimum residual or minimum error) associated with some popular and not so popular algorithms. One would expect the performances of algorithms occupying the same cell of the table to be broadly similar and this is generally the case. It can happen, though, that in finite arithmetic, algorithms that would be expected to show similar performances do actually give quite different results because of numerical instability. This is particularly true with OD and STOD so that it cannot be assumed that satisfying the same criterion over the same subspace guarantees *in practice* similar performances. Finally algorithms whose second phase is more than a straightforward recursion, due to the unsuitability for this purpose of the basis vectors generated by the first phase, are noted in the table by an asterisk.

KG	Galerkin	Minres	Minerr
\mathbf{A}	CG, FOM*	CR, GMRes*, MCR, OrthoDir, OrthoMin, MinRes*	OD, StOD, SymmLQ*
$\mathbf{A}^T \mathbf{A}$		CGNR	CGNE
$\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}$	HG, HGL	BiCR, BiCRL LSQR*, LSQR2*	
$\begin{bmatrix} \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{A}^T \end{bmatrix}$	BiCG, BiCGL	QMR*	

Table 2

4.2. Rates of convergence

Although in exact arithmetic the Krylov sequence algorithms yield the exact solution after at most n iterations, for large systems this is a somewhat academic property. The effects of rounding error ensure that vectors that should be conjugate are not in fact conjugate and the solution is thus, in practice, not attainable after the theoretical maximum number of steps. Even in the absence of rounding error the sheer size of some problems makes it uneconomic to complete the maximum number of steps, so in these cases the accuracy of the approximate solution obtained after a smaller number of iterations becomes more important. The iterative nature of the algorithms was first noted by Reid [208]. From the discussion in the previous section it follows that the maximum number of iterations needed to attain a given accuracy can be estimated only if both \mathbf{G} and \mathbf{K} are positive definite or positive real, since if one or the other is indefinite the algorithm can either crash or stagnate. We therefore assume in this section that both \mathbf{G} and \mathbf{K} are symmetric (for simplicity), and positive definite, and consider only the case where vector sequences are generated so that $\overline{\mathbf{P}}_i$ is defined by

$$\overline{\mathbf{P}}_i = [\mathbf{p}_1 \ \mathbf{p}_2 \ \dots \ \mathbf{p}_i]. \quad (4.27)$$

Now the CG algorithm is a particular case of GOMin (see page 32) so it generates displacement vectors \mathbf{p}_i using (essentially) equation (1.54), thus guaranteeing that $\mathbf{p}_j^T \mathbf{G} \mathbf{p}_k = 0$ for $j \neq k$. Moreover it also computes \mathbf{x}_{i+1} from \mathbf{x}_i using equation (1.17). The vector sequences $\{\mathbf{x}_i\}$ and $\{\mathbf{p}_i\}$ therefore satisfy the conditions of Theorem 5. It then follows from that theorem that \mathbf{x}_{i+1} minimises $\|\mathbf{e}(\mathbf{z})\|_G$ over the manifold $\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \overline{\mathbf{P}}_i \mathbf{z}$, where $\overline{\mathbf{P}}_i$ is given by equation (4.27) and \mathbf{x}_1 is arbitrary. Finally an appeal to Theorem 17 indicates that the same \mathbf{x}_{i+1} minimises $\|\mathbf{e}(\widehat{\mathbf{z}})\|_G$ over the manifold

$$\mathbf{x}(\widehat{\mathbf{z}}) \equiv \mathbf{x}_1 + \widehat{\mathbf{P}}_i \widehat{\mathbf{z}}$$

where $\widehat{\mathbf{P}}_i$ is defined by equation (4.8).

Let now the “semi-residual” \mathbf{s} be defined as before by $\mathbf{s} = \mathbf{G}^{\frac{1}{2}} \mathbf{x} - \mathbf{G}^{-\frac{1}{2}} \mathbf{h}$, where \mathbf{G} is assumed to be positive definite. Thus

$$\mathbf{s}(\widehat{\mathbf{z}}) \equiv \mathbf{s}_1 + \mathbf{G}^{\frac{1}{2}} \widehat{\mathbf{P}}_i \widehat{\mathbf{z}}$$

and since \mathbf{x}_{i+1} minimises $\|\mathbf{s}(\widehat{\mathbf{z}})\| = \|\mathbf{e}(\widehat{\mathbf{z}})\|_G$ over all $\widehat{\mathbf{z}} \in \mathbb{R}^i$ we obtain

$$\|\mathbf{s}_{i+1}\| = \min_{\widehat{\mathbf{z}} \in \mathbb{R}^i} \left\| \mathbf{s}_1 + \mathbf{G}^{\frac{1}{2}} \widehat{\mathbf{P}}_i \widehat{\mathbf{z}} \right\|. \quad (4.28)$$

Now for the Krylov methods, $\mathbf{p}_1 = \mathbf{Kf}_1 = \mathbf{KG}^{\frac{1}{2}} \mathbf{s}_1$ by imposition so that if \mathbf{C} is defined by

$$\mathbf{C} = \mathbf{G}^{\frac{1}{2}} \mathbf{KG}^{\frac{1}{2}}, \quad (4.29)$$

equation (4.8) gives

$$\mathbf{G}^{\frac{1}{2}} \widehat{\mathbf{P}}_i = [\mathbf{C}\mathbf{s}_1 \ \mathbf{C}^2\mathbf{s}_1 \ \dots \ \mathbf{C}^i\mathbf{s}_1]. \quad (4.30)$$

Assume now that $\widehat{\mathbf{z}} = [\widehat{z}_j]$ and define the set of polynomials $\mathcal{P}_i(\lambda)$ by

$$\mathcal{P}_i(\lambda) \equiv \left[\left(1 + \sum_{j=1}^i \widehat{z}_j \lambda^j \right) \mid \widehat{z}_j \in \mathbb{R} \right]. \quad (4.31)$$

It now follows from equations (4.28) and (4.30) that

$$\|\mathbf{s}_{i+1}\| = \min_{p_i(\mathbf{C}) \in \mathcal{P}_i(\mathbf{C})} \|p_i(\mathbf{C})\mathbf{s}_1\| \quad (4.32)$$

so that, for any $p_i(\mathbf{C}) \in \mathcal{P}_i(\mathbf{C})$,

$$\|\mathbf{s}_{i+1}\| \leq \|p_i(\mathbf{C})\| \|\mathbf{s}_1\|. \quad (4.33)$$

Thus if we can determine such a polynomial whose norm is sufficiently small we will have obtained a useful upper bound for $\|\mathbf{s}_{i+1}\| / \|\mathbf{s}_1\|$.

Now since \mathbf{C} is symmetric we have $\|p_i(\mathbf{C})\| = \rho(p_i(\mathbf{C}))$ where $\rho(\cdot)$ denotes the spectral radius. Thus $\|p_i(\mathbf{C})\| = \max_k |p_i(\lambda_k)|$ where λ_k is an eigenvalue of \mathbf{C} so that, from equation (4.33),

$$\|\mathbf{s}_{i+1}\| \leq \max_k |p_i(\lambda_k)| \|\mathbf{s}_1\| \quad (4.34)$$

Now we obtain the best bound for the reduction of $\|\mathbf{s}\|$ if we can choose a polynomial of degree i that minimises $\max_k |p_i(\lambda_k)|$ subject to $p_i(0) = 1$. This, however, is not possible unless all the eigenvalues λ_k are known, but if \mathbf{C} is positive definite (as it will be if \mathbf{K} is positive definite) with $0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ then it is possible to choose a polynomial that minimises $\max_k |p_i(\lambda_k)|$ over the interval $[\lambda_1, \lambda_n]$ subject to $p_i(0) = 1$. This polynomial may be obtained from a particular Chebychev polynomial in x of degree i , $T_i(x)$, by putting

$$p_i(\lambda) \equiv \frac{T_i\left(\frac{\lambda_n + \lambda_1 - 2\lambda}{\lambda_n - \lambda_1}\right)}{T_i\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right)}. \quad (4.35)$$

Since $T_i(x)$ is a polynomial in x of degree i , it follows from equation (4.35) that $p_i(\lambda)$ is a polynomial in λ of degree i and that $p_i(0) = 1$ so it has all the required properties.

Now the Chebychev polynomial of the first kind, $T_i(x)$, may be defined by $T_0(x) \equiv 1$, $T_1(x) \equiv x$ and, for $i \geq 1$,

$$T_{i+1}(x) \equiv 2xT_i(x) - T_{i-1}(x) \quad (4.36)$$

and a simple inductive argument then shows that

$$T_i(x) \equiv \frac{1}{2} \left[\left(x + \sqrt{x^2 - 1} \right)^i + \left(x - \sqrt{x^2 - 1} \right)^i \right] \quad (4.37)$$

for all i . For $|x| < 1$, $\sqrt{x^2 - 1}$ is imaginary and it is straightforward to show that in this case $|(x + \sqrt{x^2 - 1})| = |(x - \sqrt{x^2 - 1})| = 1$ so that, for $-1 \leq x \leq 1$,

$$|T_i(x)| \leq 1 \quad (4.38)$$

(see Appendix D for a brief account of Chebychev polynomials and their properties).

Thus, for $\lambda_1 \leq \lambda \leq \lambda_n$, $|T_i\left(\frac{\lambda_n + \lambda_1 - 2\lambda}{\lambda_n - \lambda_1}\right)| \leq 1$ so if $p_i(\lambda)$ is defined by equation (4.35) it follows that, for $\lambda_1 \leq \lambda \leq \lambda_n$,

$$|p_i(\lambda)| \leq \frac{1}{T_i\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right)} \quad (4.39)$$

so to complete the proof it remains only to obtain a suitable expression for $T_i\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right)$. To do this, simply substitute $\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}$ for x in equation (4.37). The identity

$$\sqrt{\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right)^2 - 1} \equiv \frac{2\sqrt{\lambda_n \lambda_1}}{\lambda_n - \lambda_1}$$

gives

$$T_i\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right) = \frac{\frac{1}{2} \left[(\lambda_n + \lambda_1 + 2\sqrt{\lambda_n \lambda_1})^i + (\lambda_n + \lambda_1 - 2\sqrt{\lambda_n \lambda_1})^i \right]}{(\lambda_n - \lambda_1)^i}$$

and since $\lambda_n - \lambda_1 \equiv (\lambda_n^{1/2} + \lambda_1^{1/2})(\lambda_n^{1/2} - \lambda_1^{1/2})$ we have

$$T_i\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right) = \frac{1}{2} \left[\left(\frac{\lambda_n^{1/2} + \lambda_1^{1/2}}{\lambda_n^{1/2} - \lambda_1^{1/2}} \right)^i + \left(\frac{\lambda_n^{1/2} - \lambda_1^{1/2}}{\lambda_n^{1/2} + \lambda_1^{1/2}} \right)^i \right].$$

Thus if κ denotes the condition number of \mathbf{C} so that $\kappa = \lambda_n/\lambda_1$ and we define σ by

$$\sigma = \frac{\kappa^{1/2} - 1}{\kappa^{1/2} + 1} = \frac{\lambda_n^{1/2} - \lambda_1^{1/2}}{\lambda_n^{1/2} + \lambda_1^{1/2}} \quad (4.40)$$

we have

$$\frac{1}{T_i\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right)} = \frac{2\sigma^i}{1 + \sigma^{2i}} \quad (4.41)$$

yielding, from equations (4.34) and (4.39) and since $\|\mathbf{s}_i\| = \|\mathbf{e}_i\|_G$, the final result

$$\frac{\|\mathbf{s}_{i+1}\|}{\|\mathbf{s}_1\|} = \frac{\|\mathbf{e}_{i+1}\|_G}{\|\mathbf{e}_1\|_G} \leq \frac{2\sigma^i}{1 + \sigma^{2i}} \leq 2\sigma^i = 2 \left(\frac{\kappa^{1/2} - 1}{\kappa^{1/2} + 1} \right)^i. \quad (4.42)$$

Thus if both \mathbf{G} and \mathbf{K} are symmetric and positive definite, the principal factor determining the convergence rate of the (vector) conjugate gradient algorithm is the spectral condition number of \mathbf{C} , where $\mathbf{C} = \mathbf{G}^{\frac{1}{2}} \mathbf{K} \mathbf{G}^{\frac{1}{2}}$. A simple similarity transform

indicates that this is given by λ_n/λ_1 , where λ_n and λ_1 are the largest and smallest eigenvalues of \mathbf{KG} which, although it is not generally symmetric, has positive real eigenvalues under the above hypotheses. Unfortunately for many of the methods described in Chapter 3, either one of both of \mathbf{G} and \mathbf{K} is indefinite so the above analysis does not apply. In fact, only four non-preconditioned HS algorithms satisfy the conditions and they are CG and CR applied to positive definite systems, CGNE and CGNR.

For CG, $\mathbf{K} = \mathbf{I}$ and $\mathbf{G} = \mathbf{A}$ where \mathbf{A} is symmetric and positive definite. In this case κ is the standard (spectral) condition number of \mathbf{A} , and inequality (4.42) indicates that as \mathbf{A} becomes increasingly ill-conditioned so does the convergence of the CG algorithm deteriorate. A similar analysis applies to CR for which $\mathbf{G} = \mathbf{A}^T$ and $\mathbf{K} = \mathbf{A}^{-1}$. However for CGNR ($\mathbf{G} = \mathbf{A}^T\mathbf{A}$ and $\mathbf{K} = \mathbf{I}$) and CGNE ($\mathbf{G} = \mathbf{I}$ and $\mathbf{K} = \mathbf{A}^T\mathbf{A}$) the position is far worse. If we define k to be the condition number of \mathbf{A} then for these algorithms inequality (4.42) is replaced by

$$\frac{\|\mathbf{s}_{i+1}\|}{\|\mathbf{s}_1\|} \leq 2 \left(\frac{k-1}{k+1} \right)^i \quad (4.43)$$

since in inequality (4.42) κ now denotes the condition number of $\mathbf{A}^T\mathbf{A}$. This is perhaps not a fair comparison because CG solves $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is symmetric positive definite whereas CGNR and CGNE both solve problems where \mathbf{A} is a general matrix, but it does give some indication of the reason for the poor performance of the latter two algorithms when applied to even moderately ill-conditioned matrices.

The inequality (4.42) is well-known and is one of the standard results of Krylov theory, but in order to sharpen its impact we consider the case where κ is moderately large, large enough for $(\frac{\kappa^{\frac{1}{2}}-1}{\kappa^{\frac{1}{2}}+1})$ to be approximated to reasonable accuracy by $1 - 2\kappa^{-\frac{1}{2}}$. Taking natural logarithms of both sides of inequality (4.42) then gives, since these are both negative,

$$\left| \ln \left(\frac{\|\mathbf{s}_{i+1}\|}{2\|\mathbf{s}_1\|} \right) \right| \geq i \left| \ln \left(1 - 2\kappa^{-\frac{1}{2}} \right) \right|$$

and if $\kappa^{-\frac{1}{2}}$ is small enough so that, to a fair degree of approximation,

$$\ln \left(1 - 2\kappa^{-\frac{1}{2}} \right) = -2\kappa^{-\frac{1}{2}}$$

we obtain

$$i \leq \frac{1}{2} \left| \ln \left(\frac{\|\mathbf{s}_{i+1}\|}{2\|\mathbf{s}_1\|} \right) \right| \kappa^{\frac{1}{2}}$$

so that the maximum number of iterations is directly proportional to $\kappa^{\frac{1}{2}}$. Now if we require that $\|\mathbf{s}_{i+1}\| / \|\mathbf{s}_1\| = 10^{-6}$, a not unreasonable requirement for convergence if $\|\mathbf{s}_1\| \approx 1$, this inequality becomes, approximately,

$$i \leq 7.25\kappa^{\frac{1}{2}}$$

so that if the CG method is applied to a symmetric positive-definite system with a condition number of 10^4 , again not excessive for a large, sparse matrix, the maximum number of iterations required to effect this reduction would be in the order of seven hundred. If, on the other hand, CGNR or CGNE were used to solve a general linear system having the same condition number it follows from applying a similar analysis to inequality (4.43) that it could take more than seventy thousand iterations to obtain the same reduction of $\|\mathbf{r}\|$. Since this would probably exceed the size of the matrix, termination would already have occurred in exact arithmetic. However, with inexact arithmetic, it is likely that the number of iterations required to obtain the above reduction of $\|\mathbf{r}\|$ would be in excess of the quoted figure.

This extreme number of iterations that might be needed to solve a moderately difficult problem well illustrates the deficiencies of CGNR and CGNE. It also illustrates the vital importance of choosing a good preconditioning matrix \mathbf{K} in order to reduce the “condition number” of \mathbf{KG} (strictly of $\mathbf{G}^{\frac{1}{2}}\mathbf{KG}^{\frac{1}{2}}$). If such a matrix could be found, even CGNR and CGNE might be rehabilitated. We return to these matters in Chapter 11.

4.3. More on GMRes

GMRes solves the equation $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is nonsingular and nonsymmetric, by generating a sequence of orthonormal vectors $\{\mathbf{p}_j\}$ using the original Arnoldi algorithm (the OAA, see page 28) with $\mathbf{p}_1 = \mathbf{r}_1 / \|\mathbf{r}_1\|$ and $\mathbf{B} = \mathbf{A}$. It then computes \mathbf{x}_{i+1} to be that value of \mathbf{x} that minimises $\|\mathbf{r}(\mathbf{z})\|$ over the manifold

$$\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \bar{\mathbf{P}}_i \mathbf{z} \quad (4.44)$$

where $\bar{\mathbf{P}}_i$ is given by equation (4.27). If we apply Theorem 2 to the manifold defined by equation (4.44) instead of to that defined by equation (1.10) we have, from equation (1.20),

$$\mathbf{x}_{i+1} = \mathbf{x}_1 - \bar{\mathbf{P}}_i \left(\bar{\mathbf{P}}_i^T \mathbf{A}^T \mathbf{A} \bar{\mathbf{P}}_i \right)^{-1} \bar{\mathbf{P}}_i^T \mathbf{A}^T \mathbf{r}_1$$

so that, since $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$,

$$\mathbf{r}_{i+1} = \left(\mathbf{I} - \mathbf{A} \bar{\mathbf{P}}_i \left(\bar{\mathbf{P}}_i^T \mathbf{A}^T \mathbf{A} \bar{\mathbf{P}}_i \right)^{-1} \bar{\mathbf{P}}_i^T \mathbf{A}^T \right) \mathbf{r}_1. \quad (4.45)$$

Since the OAA is effectively that part of GODir that generates the sequence $\{\mathbf{p}_j\}$ both Theorem 16 and a modified version of equation (4.4) apply, the modification being needed because the OAA generates normalised vectors \mathbf{p}_j so that the matrix $\hat{\mathbf{U}}_j$ of equation (4.4) is no longer unit upper triangular (though it is both upper triangular and nonsingular). Substituting the expression for $\bar{\mathbf{P}}_i$ given by equation (4.4) in equation (4.45) then gives

$$\mathbf{r}_{i+1} = \left(\mathbf{I} - \mathbf{A} \hat{\mathbf{P}}_i \left(\hat{\mathbf{P}}_i^T \mathbf{A}^T \mathbf{A} \hat{\mathbf{P}}_i \right)^{-1} \hat{\mathbf{P}}_i^T \mathbf{A}^T \right) \mathbf{r}_1 \quad (4.46)$$

where $\widehat{\mathbf{P}}_i$ is given by equation (4.26).

Since \mathbf{r}_{i+1} minimises $\|\mathbf{r}\|$ over the manifold $\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \overline{\mathbf{P}}_i \mathbf{z}$, it also minimises $\|\mathbf{r}\|$ over the manifold $\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_1 + \widehat{\mathbf{P}}_i \mathbf{z}$. Thus

$$\|\mathbf{r}_{i+1}\| = \min_{\widehat{\mathbf{z}} \in \mathbb{R}^i} \left\| \mathbf{r}_1 + \mathbf{A} \widehat{\mathbf{P}}_i \widehat{\mathbf{z}} \right\|$$

and equation (4.26) then gives

$$\|\mathbf{r}_{i+1}\| \leq \|p_i(\mathbf{A})\| \|\mathbf{r}_1\| \quad (4.47)$$

where $p_i(\mathbf{A}) \in \mathcal{P}_i(\mathbf{A})$ and $\mathcal{P}_i(\lambda)$ is defined by equation (4.31) (the polynomial $p_i(\mathbf{A})$ in this case is called the *residual polynomial*). Thus GMRes behaves in a similar manner to CG since inequalities (4.33) and (4.47) are essentially the same. The only differences are that for CG it is the norm of the “semi-residual” that is minimised and, for GMRes, \mathbf{A} is neither positive definite nor symmetric. The analysis used to establish the convergence of CG does not therefore apply to GMRes.

One of the consequences of the popularity and success of GMRes has been the amount of work devoted to disentangling its properties, and it is becoming increasingly clear that the performance of GMRes depends strongly on the departure from normality of the matrix \mathbf{A} . A (real) matrix \mathbf{A} is said to be *normal* if $\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T$ but there seems to be no general agreement on a suitable measure for any lack of normality of a matrix. A cheap-and-cheerful possibility would be the ratio of its spectral radius to its spectral norm since this would never be greater than unity and would be less than unity only if \mathbf{A} were not normal. Unfortunately this ratio can also be unity for a non-normal \mathbf{A} so a better measure of the normality of \mathbf{A} might be $\nu(\mathbf{A})$ where

$$\nu(\mathbf{A}) = \frac{\sqrt{\sum_{i=1}^n |\lambda_i|^2}}{\|\mathbf{A}\|_F} \quad (4.48)$$

and where $\mathbf{A} \in \mathbb{C}^{n \times n}$, $\|\cdot\|_F$ denotes the Frobenius norm and λ_i , $i = 1, 2, \dots, n$, are the n zeroes of the characteristic polynomial of \mathbf{A} . This is essentially Wilkinson’s ‘departure from normality’ if the norms in [258], p.169, are taken to be Frobenius ones. It satisfies $0 \leq \nu(\mathbf{A}) \leq 1$ and is unity if and only if \mathbf{A} is normal. It becomes zero if and only if \mathbf{A} is *nilpotent*, i.e. $\mathbf{A}^p = \mathbf{O}$ for some integer p , $1 < p \leq n$, and satisfies $\nu(\theta \mathbf{A}) = \nu(\mathbf{A})$ for any complex scalar θ .

One of the features of non-normal matrices is that their modal matrices can be extremely ill-conditioned. However the condition number of the modal matrix of \mathbf{A} also depends on the separation of the eigenvalues of \mathbf{A} , the condition number increasing as the eigenvalue separation decreases. Thus matrices that are not, according to the definition (4.48), excessively non-normal may nevertheless have extremely ill-conditioned modal matrices (for some matrices \mathbf{A} which have equal eigenvalues, the modal matrix may not even exist. Such matrices are referred to as *defective*).

We now prove two apparently contradictory theorems about the performance of GMRes, the first by Greenbaum and Strakoš [136] in the version of Greenbaum,

Pták and Strakoš [135] and the second by Campbell *et al* [55], and then show how the apparent contradiction may be resolved. This resolution depends heavily on the properties of non-normal matrices.

Theorem 20. Let ρ_i , $1 \leq i \leq n$, be a sequence of positive real numbers satisfying $\rho_i \geq \rho_{i+1}$ but otherwise arbitrary, and let λ_i , $1 \leq i \leq n$, be a set of nonzero real or complex numbers arbitrary apart from the restriction that any complex numbers must occur as conjugate pairs. Then there exists a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with eigenvalues λ_i , and $\mathbf{x}_1 \in \mathbb{R}^n$, such that if GMRes is used to solve $\mathbf{Ax} = \mathbf{b}$ starting at \mathbf{x}_1 , and $\|\mathbf{r}_1\| = \|\mathbf{Ax}_1 - \mathbf{b}\| = \rho_1$, then $\|\mathbf{r}_i\| = \|\mathbf{Ax}_i - \mathbf{b}\| = \rho_i$ for $2 \leq i \leq n-1$.

Proof. Let

$$\mathbf{C} = \begin{bmatrix} \mathbf{0}^T & \alpha_0 \\ \mathbf{I} & \mathbf{a} \end{bmatrix} \quad (4.49)$$

for arbitrary $\mathbf{a} \in \mathbb{R}^{n-1}$ and α_0 , and let \mathbf{v}_j , $j = 1, 2, \dots, n$, be an arbitrary set of orthonormal vectors. Define \mathbf{B} by

$$\mathbf{B} = [\mathbf{r}_1 \ \mathbf{V}_{n-1}]$$

where $\mathbf{V}_i = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_i]$ and where \mathbf{r}_1 , the residual corresponding to \mathbf{x}_1 , is chosen so that \mathbf{B} is nonsingular. Then, trivially,

$$\mathbf{Be}_1 = \mathbf{r}_1 \quad (4.50)$$

where \mathbf{e}_1 is the first column of the unit matrix, and

$$\mathbf{BC} = [\mathbf{V}_{n-1} \ \mathbf{d}] \quad (4.51)$$

for some vector \mathbf{d} . Let now $\mathbf{A} = \mathbf{BCB}^{-1}$. Substituting this expression for \mathbf{A} in equation (4.26) gives, from equation (4.50),

$$\widehat{\mathbf{AP}}_i = \mathbf{BC} [\mathbf{e}_1 \ \mathbf{Ce}_1 \ \dots \ \mathbf{C}^{i-1}\mathbf{e}_1], \quad 1 \leq i \leq n-1,$$

so that trivially, from equations (4.49) and (4.51),

$$\widehat{\mathbf{AP}}_i = \mathbf{BC} \begin{bmatrix} \mathbf{I}_i \\ \mathbf{0} \end{bmatrix} = \mathbf{V}_i, \quad 1 \leq i \leq n-1,$$

where the subscript on the identity indicates its order. Now for GMRes the residual is given by equation (4.46) so that, since $\mathbf{V}_i^T \mathbf{V}_i = \mathbf{I}_i$,

$$\mathbf{r}_{i+1} = (\mathbf{I} - \mathbf{V}_i \mathbf{V}_i^T) \mathbf{r}_1 = \mathbf{r}_1 - \sum_{j=1}^i \mathbf{v}_j \mathbf{v}_j^T \mathbf{r}_1.$$

Thus

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{v}_i \mathbf{v}_i^T \mathbf{r}_1.$$

so that

$$\|\mathbf{r}_{i+1}\|^2 = \|\mathbf{r}_i\|^2 - (\mathbf{v}_i^T \mathbf{r}_1)^2.$$

Assume now that the arbitrary initial residual \mathbf{r}_1 is given by

$$\mathbf{r}_1 = \sum_j^n \mathbf{v}_j \sqrt{\rho_j^2 - \rho_{j+1}^2}.$$

Then $(\mathbf{v}_i^T \mathbf{r}_1)^2 = \rho_i^2 - \rho_{i+1}^2$ so that $\|\mathbf{r}_i\|^2 - \|\mathbf{r}_{i+1}\|^2 = \rho_i^2 - \rho_{i+1}^2$, and since $\rho_{n+1} = 0$ this implies that $\|\mathbf{r}_j\| = \rho_j$ for $1 \leq j \leq n$. By hypothesis, $\rho_n > 0$ and this guarantees the nonsingularity of \mathbf{B} . If $\mathbf{a} = [\alpha_j]$ the eigenvalues of \mathbf{C} , and hence \mathbf{A} , are simply the roots of $\lambda^n = \sum_{j=1}^n \alpha_{j-1} \lambda^{j-1}$ (see Appendix E). It is thus straightforward to construct a family of matrices \mathbf{A} having any required set of eigenvalues and for which $\|\mathbf{r}_j\| = \rho_j$, proving the theorem. ■

The importance of this theorem is that it tells us, for GMRes applied to a general matrix \mathbf{A} , that the convergence of the algorithm does not depend on the eigenvalues of \mathbf{A} alone. In fact almost any desired convergent sequence can be matched to any choice of eigenvalues, a result that is totally at variance with the convergence behaviour of CG (see previous section). For CG the spectrum is all, and if we know the eigenvalues of \mathbf{A} we can give a very fair estimate of how CG will perform. For GMRes a similar knowledge of the eigenvalues tells us nothing. Other factors are at work and need to be taken into account in any realistic analysis. Theorem 20 was generalised to the case where $\rho_r = 0$ for $r < n + 1$ by Arioli *et al* [5].

We now prove one of a group of similar theorems that attempt to relate the convergence rate of GMRes to the eigenvalues, or Ritz values, of \mathbf{A} . The one we have chosen is a simplified version of a theorem of Campbell *et al* [55], the original theorem applying to general linear operators in Hilbert space. We prove it here for matrices but in order to keep to the spirit of the original as far as possible we allow the matrices to be complex. Similar theorems have been proved by the same authors and also by Van der Vorst and Vuik [243] but all include an unknown factor, essentially the condition number of the modal matrix of \mathbf{A} , in their bounds. It is the uncertainty of this factor, together with its extreme variability, that makes this type of theorem possible but at the same time seriously undermines its impact.

Theorem 21. Let $\mathbf{A} \in \mathbb{C}^{n \times n}$ and assume it has m distinct eigenvalues λ_i with corresponding eigenvectors \mathbf{x}_i , $i = 1, 2, \dots, m$, clustered so that for some $\rho > 0$ and $z \in \mathbb{C}$, $|\lambda_i - z| \leq \rho$, together with p eigenvalues μ_j (the outliers), each of multiplicity m_j and each lying outside the cluster ($|\mu_j - z| > \rho$). Let $d = \sum_{j=1}^p m_j$ so that $n = d + m$. Then if GMRes is applied to a set of equations with such a matrix of coefficients it follows that, for $k \geq 0$,

$$\|\mathbf{r}_{d+k+1}\| \leq C \left(\frac{\rho}{|z|} \right)^k \|\mathbf{r}_1\|, \quad (4.52)$$

where C is a constant not depending upon k .

Proof. For GMRes, \mathbf{r}_{i+1} satisfies equation (4.47) for any $p_i(\mathbf{A}) \in \mathcal{P}_i(\mathbf{A})$. Let $r = d + k$ and define the $r - th$ degree polynomial $p_r(\mathbf{A})$ by

$$p_r(\mathbf{A}) \equiv (\mathbf{I} - z^{-1}\mathbf{A})^k \prod_{j=1}^p (\mathbf{I} - \mu_j^{-1}\mathbf{A})^{m_j}. \quad (4.53)$$

If we express \mathbf{A} in the form of equation (C.2) this gives, from the hypotheses of the theorem,

$$\mathbf{A} = \sum_{i=1}^m \mathbf{x}_i \lambda_i \mathbf{y}_i^H + \sum_{j=1}^p \mathbf{X}_j \mathbf{U}_j \mathbf{Y}_j^H \quad (4.54)$$

so that, from equation (4.53),

$$p_r(\mathbf{A}) \equiv \sum_{i=1}^m \mathbf{x}_i p_r(\lambda_i) \mathbf{y}_i^H \quad (4.55)$$

where the remaining terms have been annihilated on account of the nilpotency of the factors $(\mathbf{I} - \mu_j^{-1}\mathbf{U}_j)$.

Let now

$$\bar{\mathbf{X}}_i = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_i]$$

for $1 \leq i \leq m$ with $\bar{\mathbf{Y}}_i$ being similarly defined, and let $\bar{\mathbf{D}}_m = \text{diag}(d_i)$ where $d_i = p_r(\lambda_i)$, $i = 1, 2, \dots, m$. Equation (4.55) may then be written

$$p_r(\mathbf{A}) = \bar{\mathbf{X}}_m \bar{\mathbf{D}}_m \bar{\mathbf{Y}}_m^H$$

so that

$$\|p_r(\mathbf{A})\| \leq C_m \|\bar{\mathbf{D}}_m\| \quad (4.56)$$

where

$$C_m = \|\bar{\mathbf{X}}_m\| \|\bar{\mathbf{Y}}_m^H\|.$$

Since $\bar{\mathbf{D}}_m$ is diagonal its spectral norm is the largest absolute value of its diagonal elements, and we now obtain an upper bound for this. From equation (4.53) we have

$$p_r(\lambda_i) \equiv \left(1 - \frac{\lambda_i}{z}\right)^k \prod_{j=1}^p \left(1 - \frac{\lambda_i}{\mu_j}\right)^{m_j} \quad (4.57)$$

and we first obtain bounds for the individual factors of this expression. These are, from the hypotheses of the theorem,

$$\left|1 - \frac{\lambda_i}{z}\right| = \left|\frac{z - \lambda_i}{z}\right| \leq \frac{\rho}{|z|}$$

and

$$\begin{aligned} \left| 1 - \frac{\lambda_i}{\mu_j} \right| &= \left| \frac{\mu_j - z - (\lambda_i - z)}{\mu_j} \right| \\ &\leq \frac{|\mu_j - z| + \rho}{|\mu_j|} = \beta_j \quad (\text{say}) \end{aligned} \tag{4.58}$$

since, by hypothesis, $|\lambda_i - z| \leq \rho$. Thus, independently of i , equation (4.57) gives

$$|p_r(\lambda_i)| \leq C_1 \left(\frac{\rho}{|z|} \right)^k \tag{4.59}$$

where

$$C_1 = \prod_{j=1}^p \beta_j^{m_j}$$

and from inequality (4.58) depends only on the eigenvalues of \mathbf{A} . Thus $\|\bar{\mathbf{D}}_m\| \leq C_1 (\rho/|z|)^k$ so that, from equation (4.56),

$$\|p_r(\mathbf{A})\| \leq C_1 C_m (\rho/|z|)^k. \tag{4.60}$$

Now if $j < i$, $\|\bar{\mathbf{X}}_j\| \leq \|\bar{\mathbf{X}}_i\|$ and similarly for $\|\bar{\mathbf{Y}}_j^H\|$ so that, since $m \leq n$, $\|\bar{\mathbf{X}}_m\| \leq \|\bar{\mathbf{X}}_n\|$ where $\bar{\mathbf{X}}_n$ is the modal matrix of \mathbf{A} or, if \mathbf{A} is defective, the matrix that transforms \mathbf{A} to Jordan canonical form (see Appendix C). In either case $\bar{\mathbf{Y}}_n^H = \bar{\mathbf{X}}_n^{-1}$ so that $C_m \leq C_n$ where $C_n = k(\bar{\mathbf{X}}_n)$, the condition number of $\bar{\mathbf{X}}_n$. Inequality (4.60) thus becomes

$$\|p_r(\mathbf{A})\| \leq C (\rho/|z|)^k \tag{4.61}$$

where $C = C_1 C_n$ and is independent of k , depending only on the eigenvalues of \mathbf{A} and the condition number of its modal matrix. Since $r = d+k$ the theorem follows immediately from inequality (4.47). ■

This theorem states that under certain circumstances GMRes may behave as if it were a two-stage process. In the first phase the terms due to the outliers are eliminated before rapid linear convergence sets in for the second phase. The quantities β_j which govern the first phase of the algorithm are essentially ratios of the distance of the j -th outlier to the part of the cluster furthest from it, to the distance of the j -th outlier to the origin (in the complex plane). Their effective impact on the second phase is limited to setting the bound on the norm of the residual already computed when that phase begins, and if in the complex plane the outliers were further out than, and in roughly the same direction as, the cluster this could represent a significant reduction in respect of the original norm $\|\mathbf{r}_1\|$.

The rate of convergence in the second phase is determined by $\rho/|z|$ and to make this as small as possible z would be chosen to minimise ρ subject to $|\lambda_i - z| \leq \rho$ for all eigenvalues λ_i in the cluster. Thus z in some sense represents the centre of the cluster and ρ its radius. We note that the closer z is to the origin (in the

complex plane), the slower is the final convergence of the algorithm. This corresponds to the behaviour of CG applied to a positive definite symmetric system. In this case all the eigenvalues of \mathbf{A} lie on the positive real axis and if they are regarded as one giant cluster then $\rho = \frac{1}{2}(\lambda_{\max} - \lambda_{\min})$ and $z = \frac{1}{2}(\lambda_{\max} + \lambda_{\min})$ so that $\rho/|z| = (\kappa - 1)/(\kappa + 1)$, where $\kappa = \lambda_{\max}/\lambda_{\min}$ is the condition number of \mathbf{A} . This is precisely the factor that occurs in inequality (4.43) which applies to the comparable residual minimising algorithm CGNR. Only the constant factor is different.

However it must be remembered that this theorem only provides an upper bound for the ratio $\|\mathbf{r}_{d+k+1}\|/\|\mathbf{r}_1\|$ and that the behaviour described above refers strictly only to this. The extent to which it applies to the algorithm itself depends heavily on how realistic this upper bound is, and this in turn depends on the size of C_n , the condition number of the modal matrix of \mathbf{A} . If C_n is not too large so that after, say, 5% of the theoretical maximum number of iterations the bound given by the theorem is less than unity then the theorem may give useful guidance on the performance of GMRes. If, on the other hand, C_n is so large that after, say, 95% of the theoretical maximum number of iterations the bound given by the theorem is still greater than unity then the theorem does not do a great deal for us. Unfortunately, as we show below, this can happen.

The theorem could easily be sharpened. If $(\mathbf{U}_j - \lambda_j \mathbf{I})^{k_j} = \mathbf{O}$ for some $k_j < m_j$ (as is quite possible - see Appendix C) then the d of the theorem could be replaced by $\sum_{j=1}^p k_j$ and the final phase would be activated that much sooner. If, for some values of j , $\mathbf{Y}_j^H \mathbf{r}_1 = \mathbf{0}$ then these terms could be ignored in the first phase and d could be further reduced. In the extreme case where $\mathbf{Y}_j^H \mathbf{r}_1 = \mathbf{0}$ for all j , $1 \leq j \leq p$, d could be taken to be zero and the final phase of the algorithm would begin immediately.

The theorem may also be generalised. Campbell *et al* also consider the case where there is more than one cluster of eigenvalues and it is not difficult to see how the above proof could be modified to deal with this eventuality. It also gives some pointers to the analysis of the convergence of CG. For that algorithm the eigenvalues of \mathbf{A} are real and positive and are essentially considered to consist of one cluster, the interval $[\lambda_{\min}, \lambda_{\max}]$. However, suppose that the eigenvalues lie principally in the smaller sub-interval $[\lambda_r, \lambda_s]$ with a few outliers μ_j say, for $1 \leq j \leq p$. If, in deducing inequality (4.42), the polynomial $p_i(\lambda)$ of equation (4.35) were to be replaced by

$$p_i(\lambda) \equiv \frac{T_{i-p} \left(\frac{\lambda_s + \lambda_r - 2\lambda}{\lambda_s - \lambda_r} \right) \prod_{j=1}^p \left(1 - \frac{\lambda}{\mu_j} \right)}{T_{i-p} \left(\frac{\lambda_s + \lambda_r}{\lambda_s - \lambda_r} \right)}, \quad (4.62)$$

and β_j were defined as in equation (4.58) with $z = \frac{1}{2}(\lambda_s + \lambda_r)$ and $\rho = \frac{1}{2}(\lambda_s - \lambda_r)$, then inequality (4.42) would be replaced by

$$\frac{\|\mathbf{s}_{i+1}\|}{\|\mathbf{s}_1\|} \leq 2 \left(\frac{\kappa^{\frac{1}{2}} - 1}{\kappa^{\frac{1}{2}} + 1} \right)^{i-p} \prod_{j=1}^p \beta_j,$$

where $\kappa = \lambda_s/\lambda_r$ and only the first powers of β_j appear since, for a symmetric matrix, all the matrices \mathbf{U}_j of equation (4.54) are equal to $\mu_j \mathbf{I}$. Since λ_s/λ_r is assumed to be considerably smaller than λ_n/λ_1 we would, in fact, have traded a possibly larger value $\prod_{j=1}^p \beta_j$ of the constant for an improved bound during the later stages of the algorithm.

So if the hypotheses of Theorem 21 are valid and $\rho/|z|$ is small, is rapid convergence of the second phase of GMRes guaranteed? Unfortunately not. Theorem 20 states that even with the eigenvalues of \mathbf{A} chosen so that $\rho/|z|$ is arbitrarily small it is still possible to construct a matrix \mathbf{A} having these eigenvalues but for which *there is no norm reduction at all during the first $(n-1)$ steps of the algorithm*. In this case $\|\mathbf{r}_n\| = \|\mathbf{r}_1\|$ and if there are no outliers ($p = d = 0$) the theorem states that

$$\|\mathbf{r}_1\| = \|\mathbf{r}_n\| \leq C \left(\frac{\rho}{|z|} \right)^{n-1} \|\mathbf{r}_1\|,$$

or

$$C \geq (|z|/\rho)^{n-1}. \quad (4.63)$$

Now $C = C_1 C_n$ where, as already noted, C_1 depends only on the eigenvalues of \mathbf{A} and is in fact unity in the absence of outliers. If the eigenvalues in the cluster are all distinct then $\overline{\mathbf{X}}_n$ is the modal matrix of \mathbf{A} , and since $C_1 = 1$ and $C_n = k(\overline{\mathbf{X}}_n)$, inequality (4.63) gives

$$k(\overline{\mathbf{X}}_n) \geq (|z|/\rho)^{n-1}.$$

Thus under quite modest assumptions about the eigenvalues, e.g. $\rho = 0.1$, $z = 1$ and $n = 10$, the condition number of $\overline{\mathbf{X}}_n$ can exceed 10^9 . Now if \mathbf{A} is *normal* (i.e. $\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T$), $k(\overline{\mathbf{X}}_n) = 1$ so it follows from the above analysis that the matrix \mathbf{A} constructed to satisfy the conditions of Theorem 20 is probably very far from normal. However, the excessive ill-condition of the modal matrix of \mathbf{A} depends on two factors, the lack of normality of \mathbf{A} and the closeness of its eigenvalues, and in the example quoted above the latter property could well be the more important given that the relevant eigenvalues are huddled together in a circle of radius 0.1.

The reliance on “designer polynomials” similar to that used in equation (4.62) for obtaining upper bounds for $\|\mathbf{r}_{i+1}\|$ is standard procedure. One selects a problem having certain characteristics and tries to find a polynomial that matches them in some way. If the polynomial has a norm that reduces as i increases then a genuine convergence proof, as opposed to a termination proof, will have been devised. An analysis of this type was carried out by Van der Vorst and Vuik [243]. The coefficients of their polynomials were derived from the *Ritz-values* of \mathbf{A} , i.e. the eigenvalues of $\overline{\mathbf{H}}_i$ where $\overline{\mathbf{H}}_i$ is defined in equations (2.7) - (2.14), and they showed that for certain problems the convergence of GMRes could be superlinear. This behaviour is apparently often observed in practice [243]. However, Theorem 20 shows that this cannot be true in general. But, on the other hand, Theorem 20 also demonstrates the existence of problems for which convergence of GMRes is quadratic, even cubic,

although this behaviour would not normally be attributed to the algorithm. The matter of the nature of its convergence is as yet unresolved, although it is the focus of much current research. One crucial factor is likely to be the non-normality of \mathbf{A} , and thus no significant improvements to the theory are likely to emerge until a way has been found of including this property in the analysis. This is not, though, the whole story. It was recently shown [52] that MinRes, for which \mathbf{A} is symmetric and hence normal, can also converge intolerably slowly. Clearly there is some way yet to go before the convergence properties of GMRes are fully understood.

In the final section of this chapter we return to the relationship between the vectors generated by the CG-type algorithms and Krylov sequences, but for suitably-chosen block algorithms. The motivation for this is to be found in Chapter 5.

4.4. Special cases*

In the earlier part of this chapter we showed how vector sequences generated by both the Lanczos and HS versions of the conjugate gradient algorithm were related to a (vector) Krylov sequence. We now extend this analysis to certain compound (block) methods, namely BiCG, BiCGL and QMR. The extension to BiCG in particular is important since it forms the basis of the CGS and other methods to be described in the next chapter.

4.4.1. BiCG and BiCGL

From the parameters associated with these methods (see page 52 above *et seq*) we see that

$$\mathbf{KG} = \begin{bmatrix} \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{A}^T \end{bmatrix} \quad \text{and} \quad \mathbf{KF} = \begin{bmatrix} \mathbf{r} & \mathbf{0} \\ \mathbf{0} & \mathbf{s} \end{bmatrix} \quad (4.64)$$

so that if

$$\mathbf{P}_1 = \begin{bmatrix} \mathbf{u}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_1 \end{bmatrix} \quad (4.65)$$

the matrix $\widehat{\mathbf{P}}_i$ assumes, from equation (4.1), the structure

$$\widehat{\mathbf{P}}_i = \begin{bmatrix} \mathbf{u}_1 & \mathbf{0} & \mathbf{A}\mathbf{u}_1 & \mathbf{0} & \dots & \mathbf{A}^{i-1}\mathbf{u}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_1 & \mathbf{0} & \mathbf{A}^T\mathbf{v}_1 & \dots & \mathbf{0} & (\mathbf{A}^T)^{i-1}\mathbf{v}_1 \end{bmatrix}.$$

It is readily seen that a simple column permutation of $\widehat{\mathbf{P}}_i$ leads to the alternative form

$$\widehat{\mathbf{P}}_i^{ALT} = \begin{bmatrix} \widehat{\mathbf{U}}_i & \mathbf{O} \\ \mathbf{O} & \widehat{\mathbf{V}}_i \end{bmatrix}$$

where

$$\widehat{\mathbf{U}}_i = [\mathbf{u}_1 \ \mathbf{A}\mathbf{u}_1 \ \dots \ \mathbf{A}^{i-1}\mathbf{u}_1] \quad (4.66)$$

and

$$\widehat{\mathbf{V}}_i = \begin{bmatrix} \mathbf{v}_1 & \mathbf{A}^T \mathbf{v}_1 & \dots & (\mathbf{A}^T)^{i-1} \mathbf{v}_1 \end{bmatrix}. \quad (4.67)$$

Now the above column permutation is equivalent to post-multiplying $\widehat{\mathbf{P}}_i$ by some nonsingular matrix so from Lemma 6, \mathbf{Q}_i is unchanged by this operation. Substituting the expression for \mathbf{G} appropriate to BiCG and BiCGL together with $\widehat{\mathbf{P}}_i^{ALT}$ in equation (4.5) then gives

$$\mathbf{Q}_i = \begin{bmatrix} \mathbf{I} - \mathbf{A}^T \widehat{\mathbf{V}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{U}}_i^T & \mathbf{O} \\ \mathbf{O} & \mathbf{I} - \mathbf{A} \widehat{\mathbf{U}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{V}}_i^T \end{bmatrix} \quad (4.68)$$

where

$$\widehat{\mathbf{S}}_i = \widehat{\mathbf{V}}_i^T \mathbf{A} \widehat{\mathbf{U}}_i.$$

Surprisingly, even though \mathbf{u}_1 and \mathbf{v}_1 are essentially arbitrary and \mathbf{A} is nonsymmetric, $\widehat{\mathbf{S}}_i$ is symmetric since if $\mathbf{T} = [t_{pq}] = \widehat{\mathbf{V}}_i^T \mathbf{A}^k \widehat{\mathbf{U}}_i$ for any integer k , $t_{pq} = \mathbf{v}_1^T \mathbf{A}^{(p+q+k-2)} \mathbf{u}_1$ and this expression is symmetric in p and q . Consequently, or directly from equations (4.66) and (4.67),

$$\widehat{\mathbf{U}}_i^T (\mathbf{A}^T)^k \mathbf{v}_1 = \widehat{\mathbf{V}}_i^T \mathbf{A}^k \mathbf{u}_1. \quad (4.69)$$

Moreover, since $\mathbf{P}_1 = \mathbf{K}\mathbf{F}_1$,

$$\mathbf{u}_1 = \mathbf{r}_1 \quad \text{and} \quad \mathbf{v}_1 = \mathbf{s}_1 \quad (4.70)$$

so that equation (4.69) may be written

$$\widehat{\mathbf{U}}_i^T (\mathbf{A}^T)^k \mathbf{s}_1 = \widehat{\mathbf{V}}_i^T \mathbf{A}^k \mathbf{r}_1. \quad (4.71)$$

Now for the two algorithms being considered

$$\mathbf{F}_i = \begin{bmatrix} \mathbf{0} & \mathbf{s}_i \\ \mathbf{r}_i & \mathbf{0} \end{bmatrix} \quad (4.72)$$

and if $\mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$ is the initial residual and \mathbf{s}_1 is the initial shadow residual, equations (1.43) and (4.70) yield, in the absence of breakdown and since $\widehat{\mathbf{S}}_i$ is symmetric,

$$\mathbf{r}_{i+1} = (\mathbf{I} - \mathbf{A} \widehat{\mathbf{U}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{V}}_i^T) \mathbf{r}_1 \quad (4.73)$$

and

$$\mathbf{s}_{i+1} = (\mathbf{I} - \mathbf{A}^T \widehat{\mathbf{V}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{U}}_i^T) \mathbf{s}_1. \quad (4.74)$$

These equations may be expressed as

$$\mathbf{r}_{i+1} = \mathbf{r}_1 + \mathbf{A} \widehat{\mathbf{U}}_i \mathbf{b} \quad \text{and} \quad \mathbf{s}_{i+1} = \mathbf{s}_1 + \mathbf{A}^T \widehat{\mathbf{V}}_i \mathbf{b}$$

where from equations (4.71) with $k = 0$ the vector $\mathbf{b} = [\beta_j]$ is the same for both equations. They then become, from equations (4.66) and (4.67) with \mathbf{r}_1 and \mathbf{s}_1 being substituted for \mathbf{u}_1 and \mathbf{v}_1 ,

$$\mathbf{r}_{i+1} = p_i(\mathbf{A})\mathbf{r}_1 \quad \text{and} \quad \mathbf{s}_{i+1} = p_i(\mathbf{A}^T)\mathbf{s}_1 \quad (4.75)$$

where, for any matrix \mathbf{M} ,

$$p_i(\mathbf{M}) \equiv \mathbf{I} + \sum_{j=1}^i \beta_j \mathbf{M}^j.$$

Thus the residuals \mathbf{r}_{i+1} and \mathbf{s}_{i+1} may be obtained by premultiplying the initial residuals \mathbf{r}_1 and \mathbf{s}_1 by *identical* polynomials, one in \mathbf{A} and the other in \mathbf{A}^T .

Now for BiCGL, equation (4.6) implies that $\mathbf{P}_{i+1} = \mathbf{Q}_i^T (\mathbf{K}\mathbf{G})^i \mathbf{P}_1$ so that, from equations (4.64), (4.65) and (4.68),

$$\mathbf{u}_{i+1} = (\mathbf{I} - \widehat{\mathbf{U}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{V}}_i^T \mathbf{A}) \mathbf{A}^i \mathbf{u}_1 \quad (4.76)$$

with a similar expression for \mathbf{v}_{i+1} . A similar argument to that used to establish equations (4.75) then gives

$$\mathbf{u}_{i+1} = q_i(\mathbf{A})\mathbf{u}_1 \quad \text{and} \quad \mathbf{v}_{i+1} = q_i(\mathbf{A}^T)\mathbf{v}_1 \quad (4.77)$$

where again the polynomials are the same in each case. If we now define $\overline{\mathbf{U}}_k$ by

$$\overline{\mathbf{U}}_k = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_k]$$

and apply equation (4.76) for $i = 1, 2, \dots, k-1$, we obtain

$$\overline{\mathbf{U}}_k = \widehat{\mathbf{U}}_k \widehat{\mathbf{M}}_k \quad (4.78)$$

where $\widehat{\mathbf{M}}_k$ (but not $\overline{\mathbf{U}}_k$ and $\widehat{\mathbf{U}}_k$) is some unit upper-triangular matrix. We thus obtain a version of Theorem 16 that reflects more accurately the block structure of $\mathbf{K}\mathbf{G}$.

Finally, for the BiCG method, equations (4.14) and (4.15) imply that

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T (\mathbf{K}\mathbf{G})^i \mathbf{P}_1 \mathbf{U}_{i+1,i+1}.$$

Equation (3.7) with $i+1$ replaced by i and with $j = i$ together with equation (4.13) give

$$\mathbf{M}_i = -\mathbf{D}_i^{-1} \mathbf{F}_i^T \mathbf{K} \mathbf{F}_i$$

and substituting the appropriate values of \mathbf{G} , \mathbf{K} , \mathbf{P}_i and \mathbf{F}_i into this equation gives $\mathbf{M}_i = \gamma_i \mathbf{I}$ where

$$\gamma_i = -\mathbf{s}_i^T \mathbf{r}_i / \mathbf{v}_i^T \mathbf{A} \mathbf{u}_i.$$

Thus, if $\theta_i = \prod_{j=1}^i \gamma_j$ we have, for the BiCG method,

$$\mathbf{u}_{i+1} = \theta_i q_i(\mathbf{A}) \mathbf{u}_1 \quad \text{and} \quad \mathbf{v}_{i+1} = \theta_i q_i(\mathbf{A}^T) \mathbf{v}_1 \quad (4.79)$$

where $q_i(\mathbf{A})$ is the identical polynomial to that appearing in equation (4.77), and again the vectors \mathbf{u}_{i+1} and \mathbf{v}_{i+1} are obtained by multiplying their original values by identical polynomials in \mathbf{A} and \mathbf{A}^T . It follows from equation (4.79) that equation (4.78) also holds for BiCG but in this case $\widehat{\mathbf{M}}_k$ is now upper (as opposed to unit upper) triangular with diagonal elements equal to θ_i . These results form the basis of the CGS algorithm to be discussed in the next chapter.

4.4.2. QMR

From the parameters associated with QMR (see page 69 above) we see that both **KG** and **KF** are the same for QMR as they are for BiCG and BiCGL and since, in every case, $\mathbf{P}_1 = \mathbf{KF}_1$ it follows that \mathbf{P}_1 is also the same. Thus equations (4.64) - (4.67) are also valid for QMR but since $\mathbf{G}^{QMR} \neq \mathbf{G}^{BiCG}$ we obtain a different form for \mathbf{Q}_i . The same argument used to establish equation (4.68) then gives, for QMR,

$$\mathbf{Q}_i = \begin{bmatrix} \mathbf{I} - \widehat{\mathbf{V}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{U}}_i^T & \mathbf{O} \\ \mathbf{O} & \mathbf{I} - \widehat{\mathbf{U}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{V}}_i^T \end{bmatrix} \quad (4.80)$$

where

$$\widehat{\mathbf{S}}_i = \widehat{\mathbf{V}}_i^T \widehat{\mathbf{U}}_i.$$

The arguments of the preceding section show the QMR equivalent of equations (4.73) and (4.74) to be

$$\mathbf{r}_{i+1} = (\mathbf{I} - \mathbf{A} \widehat{\mathbf{U}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{V}}_i^T \mathbf{A}^{-1}) \mathbf{r}_1$$

and

$$\mathbf{s}_{i+1} = (\mathbf{I} - \mathbf{A}^T \widehat{\mathbf{V}}_i \widehat{\mathbf{S}}_i^{-1} \widehat{\mathbf{U}}_i^T \mathbf{A}^{-T}) \mathbf{s}_1$$

so that equations (4.75), (4.77) and (4.79) are also valid for QMR but with, of course, different polynomials $p_i(\mathbf{A})$ and $q_i(\mathbf{A})$, and scalars θ_i .

Most of the material in this section has little practical relevance and is included only for completeness, but this is emphatically not the case for equations (4.75) and (4.79) since they underpin the seminal CGS algorithm. That this is so follows from the equations defining the BiCG algorithm (see equations (3.31) - (3.34)), all of which involve either $\mathbf{s}_i^T \mathbf{r}_i$ or $\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i$ or, from equations (4.75) and (4.79), $\mathbf{s}_1^T [p_{i-1}(\mathbf{A})]^2 \mathbf{r}_1$ or $\theta_{i-1}^2 \mathbf{v}_1^T \mathbf{A} [q_{i-1}(\mathbf{A})]^2 \mathbf{u}_1$. The CGS algorithm works by generating the sequences $\{[p_j(\mathbf{A})]^2 \mathbf{r}_1\}$ and $\{[q_j(\mathbf{A})]^2 \mathbf{u}_1\}$ as opposed to (effectively) the sequences $\{p_j(\mathbf{A}) \mathbf{r}_1\}$, $\{p_j(\mathbf{A}^T) \mathbf{s}_1\}$, $\{q_j(\mathbf{A}) \mathbf{u}_1\}$ and $\{q_j(\mathbf{A}^T) \mathbf{v}_1\}$ generated by BiCG.

Now the generation of the “squared” sequences is possible since, for BiCG,

$$\mathbf{KG} = \begin{bmatrix} \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{A}^T \end{bmatrix}$$

(equation (4.64)), i.e. \mathbf{KG} is block diagonal. Thus $(\mathbf{KG})^j$ is also block diagonal for all j and

$$p_j(\mathbf{KG}) = \begin{bmatrix} p_j(\mathbf{A}) & \mathbf{O} \\ \mathbf{O} & p_j(\mathbf{A}^T) \end{bmatrix}.$$

This is in marked distinction to the Hegedüs algorithms for which

$$\mathbf{KG} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}$$

and for which $(\mathbf{KG})^j$ is block diagonal for j even and block skew-diagonal for j odd. It might be possible to develop a “squared” version of HG based on the even values of j for which

$$p_j(\mathbf{KG}) = \begin{bmatrix} p_{j/2}(\mathbf{A}^T \mathbf{A}) & \mathbf{O} \\ \mathbf{O} & p_{j/2}(\mathbf{A} \mathbf{A}^T) \end{bmatrix}$$

but even so one of the main advantages of CGS, that of being “transpose-free”, would be lost. The form of $(\mathbf{KG})^j$ for BiCG is somewhat awkward because it is always non-symmetric (that for HG is always symmetric) and hence may have complex eigenvalues or other undesirable properties but this surprisingly turns out to be one of its strengths. The precise way in which the block-diagonal nature of $(\mathbf{KG})^j$ may be exploited to derive transpose-free algorithms is described fully in the next chapter.

This page intentionally left blank

Chapter 5

Transpose-free methods

One of the disadvantages of the methods of Chapter 3 for solving equations with nonsymmetric matrices is that they require the calculation of both \mathbf{Ax} and $\mathbf{A}^T\mathbf{y}$ for essentially arbitrary vectors \mathbf{x} and \mathbf{y} . Even in the absence of other considerations this requires two distinct subroutines for performing the matrix-vector multiplications, and if the data structure used for storing \mathbf{A} favours the calculation of \mathbf{Ax} it is unlikely to be efficient when computing $\mathbf{A}^T\mathbf{y}$. Moreover for some applications, e.g. problems derived from ordinary differential equations [116], [126], the rows of \mathbf{A} arise naturally from the finite-difference formulæ used to approximate the derivatives and thus the matrix-vector products might be more readily computable than those involving \mathbf{A}^T which would enjoy no such advantage.

Another feature of BiCG in particular that might be regarded as a disadvantage is the extremely tenuous connection between the two sequences of displacement vectors $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ generated by the algorithm. Examination of equations (3.31) - (3.35) (see page 52) shows that the only rôle played by the sequences $\{\mathbf{s}_i\}$ and $\{\mathbf{v}_i\}$ in generating the sequences $\{\mathbf{u}_i\}$, $\{\mathbf{r}_i\}$ and $\{\mathbf{x}_i\}$ lies in the calculation of the scalars $\mathbf{s}_i^T \mathbf{r}_i$ and $\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i$ for substitution in equations (3.31), (3.32) and (3.34). If another way could be found of determining these quantities then almost half the calculation could be omitted. Alternatively, if this were not possible, the calculation could perhaps be rearranged and put to better use. These considerations prompted the search for “transpose free” methods that do not require the calculation of $\mathbf{A}^T\mathbf{y}$, and this chapter discusses several such methods starting with the oldest, the CGS algorithm of Sonneveld [222], published in 1989 but submitted for publication nearly five years earlier.

5.1. The conjugate-gradient squared method (CGS)

Despite its name, the CGS method is based on the BiCG method and not the original CG algorithm, and its derivation is made possible by the tenuous connection already noted between the two sequences of displacement vectors generated by BiCG. We begin the derivation of CGS by obtaining alternative expressions for

the vectors generated by BiCG (see page 52 above).

We saw in Chapter 4 (equation (4.75)) that for this method the residual \mathbf{r}_{i+1} can be expressed as a polynomial in \mathbf{A} of degree i premultiplying \mathbf{r}_1 . If this polynomial is denoted by Φ_i then

$$\mathbf{r}_{i+1} = \Phi_i \mathbf{r}_1. \quad (5.1)$$

We saw also that the corresponding shadow residual \mathbf{s}_{i+1} can be expressed by premultiplying \mathbf{s}_1 by the *identical* polynomial in \mathbf{A}^T . Since the polynomial of a transposed matrix is the transpose of the original polynomial this implies that

$$\mathbf{s}_{i+1} = \Phi_i^T \mathbf{s}_1. \quad (5.2)$$

Similarly, from equation (4.79), \mathbf{u}_{i+1} and \mathbf{v}_{i+1} may be obtained by premultiplying \mathbf{u}_1 and \mathbf{v}_1 respectively by another polynomial of degree i in \mathbf{A} and by the same polynomial in \mathbf{A}^T . If we denote this first polynomial by Θ_i then

$$\mathbf{u}_{i+1} = \Theta_i \mathbf{u}_1 \quad (5.3)$$

and

$$\mathbf{v}_{i+1} = \Theta_i^T \mathbf{v}_1. \quad (5.4)$$

Define now $\widehat{\mathbf{r}}_{i+1}$ and $\widehat{\mathbf{u}}_{i+1}$ by

$$\widehat{\mathbf{r}}_{i+1} = \Phi_i^2 \mathbf{r}_1 \quad (5.5)$$

and

$$\widehat{\mathbf{u}}_{i+1} = \Theta_i^2 \mathbf{u}_1 = \Theta_i^2 \mathbf{r}_1 \quad (5.6)$$

(since $\mathbf{u}_1 = \mathbf{r}_1$ from the definition of BiCG, see page 52 above). The vector sequences $\{\widehat{\mathbf{r}}_i\}$ and $\{\widehat{\mathbf{u}}_i\}$ are the ones actually computed by CGS and are generated recursively as shown below. Before describing the algorithm proper though we complete the preliminaries by first obtaining expressions in terms of $\widehat{\mathbf{r}}_i$ and $\widehat{\mathbf{u}}_i$ of the two scalar quantities that appear in BiCG and by proving a simple lemma that validates one of the steps used in the derivation of the algorithm. The two scalars are $\mathbf{s}_i^T \mathbf{r}_i = \rho_i$ (say) and $\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i = \sigma_i$ (say), and these, from equations (5.1) - (5.6), are given by

$$\rho_i = \mathbf{s}_1^T \Phi_{i-1}^2 \mathbf{r}_1 = \mathbf{s}_1^T \widehat{\mathbf{r}}_i \quad (5.7)$$

and

$$\sigma_i = \mathbf{v}_1^T \Theta_{i-1} \mathbf{A} \Theta_{i-1} \mathbf{u}_1 = \mathbf{v}_1^T \mathbf{A} \Theta_{i-1}^2 \mathbf{u}_1 = \mathbf{s}_1^T \mathbf{A} \widehat{\mathbf{u}}_i \quad (5.8)$$

since all powers of a matrix commute and since, by definition, $\mathbf{v}_1 = \mathbf{s}_1$. The lemma is:

Lemma 22. Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ and let $\mathbf{AB} = \mathbf{BA}$. Let $\mathbf{c} \in \mathbb{R}^n$ be a vector such that $\mathbf{Ac} = \mathbf{Bc}$. Then $\mathbf{A}^2 \mathbf{c} = \mathbf{B}^2 \mathbf{c}$.

Proof. $\mathbf{A}^2 \mathbf{c} = \mathbf{A}(\mathbf{Ac}) = \mathbf{ABc} = \mathbf{BAc} = \mathbf{B}^2 \mathbf{c}$

as required. \blacksquare

Note: The commutativity of \mathbf{A} and \mathbf{B} is essential for this result as may be demonstrated by putting

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{c} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

We now resume our description of CGS. From equations (3.32), (3.34), (5.1) and (5.3) we obtain, since $\mathbf{u}_1 = \mathbf{r}_1$,

$$\Phi_i \mathbf{r}_1 = (\Phi_{i-1} - \alpha_i \mathbf{A} \Theta_{i-1}) \mathbf{r}_1 \quad (5.9)$$

and

$$\Theta_{i-1} \mathbf{r}_1 = (\Phi_{i-1} + \beta_i \Theta_{i-2}) \mathbf{r}_1 \quad (5.10)$$

where, from equations (5.7) and (5.8),

$$\alpha_i = \frac{\mathbf{s}_1^T \widehat{\mathbf{r}}_i}{\mathbf{s}_1^T \mathbf{A} \widehat{\mathbf{u}}_i} \quad \text{and} \quad \beta_i = \frac{\mathbf{s}_1^T \widehat{\mathbf{r}}_i}{\mathbf{s}_1^T \widehat{\mathbf{r}}_{i-1}}. \quad (5.11)$$

Define now $\widehat{\mathbf{p}}_{i+1}$ and $\widehat{\mathbf{q}}_i$ by

$$\widehat{\mathbf{p}}_{i+1} = \Phi_i \Theta_{i-1} \mathbf{r}_1 \quad (5.12)$$

and

$$\widehat{\mathbf{q}}_i = \Phi_{i-1} \Theta_{i-1} \mathbf{r}_1. \quad (5.13)$$

Premultiplying equation (5.9) by Θ_{i-1} and equation (5.10) by Φ_{i-1} gives, since all relevant matrices commute,

$$\Phi_i \Theta_{i-1} \mathbf{r}_1 = (\Phi_{i-1} \Theta_{i-1} - \alpha_i \mathbf{A} \Theta_{i-1}^2) \mathbf{r}_1$$

and

$$\Phi_{i-1} \Theta_{i-1} \mathbf{r}_1 = (\Phi_{i-1}^2 + \beta_i \Phi_{i-1} \Theta_{i-2}) \mathbf{r}_1 \quad (5.14)$$

so that, from equations (5.5), (5.6), (5.12) and (5.13),

$$\widehat{\mathbf{p}}_{i+1} = \widehat{\mathbf{q}}_i - \alpha_i \mathbf{A} \widehat{\mathbf{u}}_i \quad (5.15)$$

and

$$\widehat{\mathbf{q}}_i = \widehat{\mathbf{r}}_i + \beta_i \widehat{\mathbf{p}}_i. \quad (5.16)$$

Similarly all matrices in equations (5.9) and (5.10) commute so that, from Lemma 22,

$$\Phi_i^2 \mathbf{r}_1 = (\Phi_{i-1}^2 - 2\alpha_i \mathbf{A} \Phi_{i-1} \Theta_{i-1} + \alpha_i^2 \mathbf{A}^2 \Theta_{i-1}^2) \mathbf{r}_1$$

and

$$\Theta_{i-1}^2 \mathbf{r}_1 = (\Phi_{i-1}^2 + 2\beta_i \Phi_{i-1} \Theta_{i-2} + \beta_i^2 \Theta_{i-2}^2) \mathbf{r}_1.$$

These equations may be written, from equations (5.5), (5.6), (5.12) and (5.13),

$$\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{r}}_i - \alpha_i \mathbf{A} (2\hat{\mathbf{q}}_i - \alpha_i \mathbf{A} \hat{\mathbf{u}}_i)$$

and

$$\hat{\mathbf{u}}_i = \hat{\mathbf{r}}_i + \beta_i \hat{\mathbf{p}}_i + \beta_i (\hat{\mathbf{p}}_i + \beta_i \hat{\mathbf{u}}_{i-1})$$

or, from equations (5.15) and (5.16),

$$\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{r}}_i - \alpha_i \mathbf{A} (\hat{\mathbf{q}}_i + \hat{\mathbf{p}}_{i+1}) \quad (5.17)$$

and

$$\hat{\mathbf{u}}_i = \hat{\mathbf{q}}_i + \beta_i (\hat{\mathbf{p}}_i + \beta_i \hat{\mathbf{u}}_{i-1}). \quad (5.18)$$

If we look at the equations (5.11) and (5.15) - (5.18) we see that, given $\hat{\mathbf{u}}_{i-1}$, $\hat{\mathbf{p}}_i$ and $\hat{\mathbf{r}}_i$ we can first compute β_i and $\hat{\mathbf{q}}_i$, then $\hat{\mathbf{u}}_i$ followed by α_i , and finally $\hat{\mathbf{p}}_{i+1}$ and $\hat{\mathbf{r}}_{i+1}$. This closes the loop and we can go on to compute the next round of approximations. The algorithm is initialised by choosing arbitrary values of \mathbf{x}_1 and \mathbf{s}_1 , computing $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$ and setting $\hat{\mathbf{r}}_1 = \hat{\mathbf{u}}_1 = \mathbf{r}_1$. The initial values $\hat{\mathbf{p}}_1$ and $\hat{\mathbf{u}}_0$ are chosen to be null. Finally we need to compute the sequence $\{\hat{\mathbf{x}}_i\}$ which we do from the sequence $\{\hat{\mathbf{r}}_i\}$. If $\mathbf{A}\hat{\mathbf{x}}_i - \mathbf{b} = \hat{\mathbf{r}}_i$ it follows from equation (5.17) that $\hat{\mathbf{x}}_{i+1}$ is given by

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - \alpha_i (\hat{\mathbf{q}}_i + \hat{\mathbf{p}}_{i+1}). \quad (5.19)$$

The final form of the algorithm thus becomes:

The Conjugate-gradient squared (CGS) algorithm

- (1) Select $\hat{\mathbf{x}}_1$, \mathbf{s}_1 arbitrarily, compute $\hat{\mathbf{r}}_1 = \mathbf{A}\hat{\mathbf{x}}_1 - \mathbf{b}$,
set $\hat{\mathbf{p}}_1 = \hat{\mathbf{u}}_0 = 0$, $\rho_0 = 1$ and $i = 1$
- (2) while not converged
 - (a) compute $\rho_i = \mathbf{s}_1^T \hat{\mathbf{r}}_i$ and $\beta_i = \rho_i / \rho_{i-1}$
 - (b) compute $\hat{\mathbf{q}}_i = \hat{\mathbf{r}}_i + \beta_i \hat{\mathbf{p}}_i$
 - (c) compute $\hat{\mathbf{u}}_i = \hat{\mathbf{q}}_i + \beta_i (\hat{\mathbf{p}}_i + \beta_i \hat{\mathbf{u}}_{i-1})$ and $\hat{\mathbf{w}}_i = \mathbf{A} \hat{\mathbf{u}}_i$
 - (d) compute $\sigma_i = \mathbf{s}_1^T \hat{\mathbf{w}}_i$ and $\alpha_i = \rho_i / \sigma_i$
 - (e) compute $\hat{\mathbf{p}}_{i+1} = \hat{\mathbf{q}}_i - \alpha_i \hat{\mathbf{w}}_i$
 - (f) compute $\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{r}}_i - \alpha_i \mathbf{A} (\hat{\mathbf{q}}_i + \hat{\mathbf{p}}_{i+1})$
 - (g) compute $\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - \alpha_i (\hat{\mathbf{q}}_i + \hat{\mathbf{p}}_{i+1})$
 - (h) set $i = i + 1$
- (3) endwhile
- (4) end CGS

The calculation of $\hat{\mathbf{r}}_{i+1}$ and $\hat{\mathbf{u}}_{i+1}$ only involves premultiplication of a vector by \mathbf{A} . Moreover, if $\|\Phi_i\| < 1$ then $\|\Phi_i^2\| \leq \|\Phi_i\|^2 < \|\Phi_i\|$ and there is a reasonable presumption (although not a certainty) that since $\mathbf{r}_{i+1} = \Phi_i \mathbf{r}_1$ and $\hat{\mathbf{r}}_{i+1} = \Phi_i^2 \mathbf{r}_1$ then $\|\hat{\mathbf{r}}_{i+1}\| < \|\mathbf{r}_{i+1}\|$. Thus the norm of the vector $\hat{\mathbf{r}}_{i+1}$ obtained by CGS may generally be expected to be less than the norm of the corresponding residual obtained by

BiCG. This suggests that if the vector $\widehat{\mathbf{x}}_{i+1}$ might be a better approximation to the solution of the equation $\mathbf{Ax} = \mathbf{b}$ than the \mathbf{x}_{i+1} computed by the BiCG.

The motivation underlying the CGS algorithm is largely justified by its performance. If BiCG converges then CGS converges more rapidly and the algorithm is remarkably efficient, but there is another side to this particular coin. If $\|\Phi_i\| > 1$ then $\|\Phi_i\|^2 > \|\Phi_i\|$ and it is quite possible that $\|\Phi_i^2\| > \|\Phi_i\|$. Since, from equations (5.1) and (5.5),

$$\|\mathbf{r}_{i+1}\| \leq \|\Phi_i\| \|\mathbf{r}_1\| \quad \text{and} \quad \|\widehat{\mathbf{r}}_{i+1}\| \leq \|\Phi_i^2\| \|\mathbf{r}_1\|$$

it follows in this case that $\|\widehat{\mathbf{r}}_{i+1}\|$ could be substantially *greater* than $\|\mathbf{r}_{i+1}\|$. This too happens in practice and it is found that if the behaviour of BiCG is erratic then that of CGS tends to be even more erratic. Now for a general nonsingular matrix \mathbf{A} there is no particular reason why BiCG should converge in a uniform manner. The proof that it will eventually find a solution is a *termination* proof rather than a *convergence* proof (convergence implies some degree of monotonicity, no matter how tenuous) and if any of the computed quantities $\sigma_i = \mathbf{v}_i^T \mathbf{A} \mathbf{u}_i$ or $\rho_i = \mathbf{s}_i^T \mathbf{r}_i$ is small then the progress of the algorithm is anything but monotonic. This behaviour is exaggerated in the CGS algorithm and gives rise to excessive rounding errors that can prevent accurate termination [241]. Thus as the basis of a general problem solver, CGS is not satisfactory. Its (very great) importance lies in its presenting a different perspective on a whole class of Krylov algorithms.

5.2. BiCGStab

The algorithm BiCGStab [241] was proposed in an attempt to curb the excesses of CGS while at the same time retaining the advantages of more rapid convergence and the non-use of transposes. It works by constructing two sequences of vectors $\{\tilde{\mathbf{r}}_i\}$ and $\{\tilde{\mathbf{u}}_i\}$ based on a specially-chosen polynomial in \mathbf{A} denoted by Ψ_i , where

$$\Psi_i \equiv \sum_{j=0}^i \eta_{ij} \mathbf{A}^j, \tag{5.20}$$

and defining $\tilde{\mathbf{r}}_{i+1}$ and $\tilde{\mathbf{u}}_{i+1}$ by

$$\tilde{\mathbf{r}}_{i+1} = \Psi_i \Phi_i \mathbf{r}_1 \quad \text{and} \quad \tilde{\mathbf{u}}_{i+1} = \Psi_i \Theta_i \mathbf{r}_1. \tag{5.21}$$

The matrix polynomials Φ_i and Θ_i are as defined in the previous section. Before, though, we describe BiCGStab we prove two technical lemmas that pertain to the BiCG algorithm, the algorithm that underpins most of the algorithms described in this chapter.

Lemma 23. Let the sequences $\{\mathbf{r}_i\}$ and $\{\mathbf{s}_i\}$ be computed by the BiCG algorithm, let $\rho_i = \mathbf{s}_i^T \mathbf{r}_i$ and denote the polynomial $p_i(\mathbf{A})$ of equation (4.75) by

$$p_i(\mathbf{A}) \equiv \Phi_i \equiv \sum_{j=0}^i \lambda_{ij} \mathbf{A}^j. \quad (5.22)$$

Then if $\rho_j \neq 0$ for $j = 1, 2, \dots, i$,

- (a) $\mathbf{s}_1^T \mathbf{A}^k \mathbf{r}_j = 0, \quad 0 \leq k \leq j-2$, and
- (b) $\lambda_{j-1,j-1} \mathbf{s}_1^T \mathbf{A}^{j-1} \mathbf{r}_j = \rho_j$.

Proof. By induction. We show that if the lemma is true for $j = 1, 2, \dots, i$, it is also true for $j = i + 1$.

From (b) and the hypothesis that $\rho_j \neq 0$ we infer that $\lambda_{j-1,j-1} \neq 0$ for $j = 1, 2, \dots, i$. Now for these values of j we have, from the properties of the BiCG algorithm, $\mathbf{s}_j^T \mathbf{r}_{i+1} = 0$ so that, from equations (4.75) and (5.22),

$$\mathbf{s}_1^T \left(\sum_{k=0}^{j-1} \lambda_{j-1,k} \mathbf{A}^k \right) \mathbf{r}_{i+1} = 0.$$

Putting $j = 1, 2, \dots, i$ in turn then yields, since $\lambda_{j-1,j-1} \neq 0$,

$$\mathbf{s}_1^T \mathbf{A}^j \mathbf{r}_{i+1} = 0, \quad j = 0, 1, \dots, i-1,$$

so that (a) is true for $j = i + 1$. But, since $\mathbf{s}_{i+1}^T \mathbf{r}_{i+1} = \rho_{i+1}$ we obtain again from equation (4.75)

$$\mathbf{s}_1^T \left(\sum_{k=0}^i \lambda_{ik} \mathbf{A}^k \right) \mathbf{r}_{i+1} = \rho_{i+1}$$

so that, from (a) updated,

$$\lambda_{ii} \mathbf{s}_1^T \mathbf{A}^i \mathbf{r}_{i+1} = \rho_{i+1}$$

completing the proof ■

Corollary 24. For any arbitrary polynomial Ψ_i in \mathbf{A} , where

$$\Psi_i \equiv \sum_{j=0}^i \eta_{ij} \mathbf{A}^j,$$

$$\mathbf{s}_1^T \Psi_i \mathbf{r}_{i+1} = \eta_{ii} \mathbf{s}_1^T \mathbf{A}^i \mathbf{r}_{i+1}.$$

Proof. Straightforward, from the lemma. ■

Lemma 25. Let the sequences $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ be computed by the BiCG algorithm, let $\sigma_i = \mathbf{v}_i^T \mathbf{A} \mathbf{u}_i$ and denote the polynomial $q_i(\mathbf{A})$ of equation (4.79) by

$$\theta_i q_i(\mathbf{A}) \equiv \Theta_i \equiv \sum_{j=0}^i \mu_{ij} \mathbf{A}^j. \quad (5.23)$$

Then, if $\sigma_j \neq 0$ for $j = 1, 2, \dots, i$,

- (a) $\mathbf{v}_1^T \mathbf{A}^{k+1} \mathbf{u}_j = 0, \quad 0 \leq k \leq j-2$, and
(b) $\mu_{j-1,j-1} \mathbf{v}_1^T \mathbf{A}^j \mathbf{u}_j = \sigma_j$.

Proof. Similar to the proof of the previous lemma. \blacksquare

Corollary 26. For any arbitrary polynomial Ψ_i in \mathbf{A} as defined in the corollary to the previous lemma,

$$\mathbf{v}_1^T \mathbf{A} \Psi_i \mathbf{u}_{i+1} = \eta_{ii} \mathbf{v}_1^T \mathbf{A}^{i+1} \mathbf{u}_{i+1}.$$

Proof. Straightforward, from the lemma. \blacksquare

With these lemmas established we are now in a position to describe BiCGStab.

Let the matrix polynomials Φ_i and Θ_i be as defined in the previous section and satisfy the recursion formulæ derived from the BiCG algorithm. Define a new polynomial in \mathbf{A} , Ψ_i , by $\Psi_0 = \mathbf{I}$ and

$$\Psi_i \equiv (\mathbf{I} - \omega_i \mathbf{A}) \Psi_{i-1} \quad (5.24)$$

for $i > 0$, where the scalars ω_i are to be determined. From equations (5.9), (5.21) and (5.24) we obtain

$$\Psi_i \Phi_i \mathbf{r}_1 = (\mathbf{I} - \omega_i \mathbf{A}) (\Psi_{i-1} \Phi_{i-1} - \alpha_i \mathbf{A} \Psi_{i-1} \Theta_{i-1}) \mathbf{r}_1$$

or

$$\tilde{\mathbf{r}}_{i+1} = (\mathbf{I} - \omega_i \mathbf{A}) (\tilde{\mathbf{r}}_i - \alpha_i \mathbf{A} \tilde{\mathbf{u}}_i) \quad (5.25)$$

while from equations (5.10), (5.21) and (5.24) we have

$$\begin{aligned} \Psi_i \Theta_i \mathbf{r}_1 &= \Psi_i (\Phi_i + \beta_{i+1} \Theta_{i-1}) \mathbf{r}_1 \\ &= \Psi_i \Phi_i \mathbf{r}_1 + \beta_{i+1} (\mathbf{I} - \omega_i \mathbf{A}) \Psi_{i-1} \Theta_{i-1} \mathbf{r}_1 \end{aligned}$$

or

$$\tilde{\mathbf{u}}_{i+1} = \tilde{\mathbf{r}}_{i+1} + \beta_{i+1} (\mathbf{I} - \omega_i \mathbf{A}) \tilde{\mathbf{u}}_i. \quad (5.26)$$

We have thus obtained two formulæ by which the sequences $\{\tilde{\mathbf{r}}_i\}$ and $\{\tilde{\mathbf{u}}_i\}$ may be computed recursively and from which we will be able to compute a sequence of approximate solutions $\{\hat{\mathbf{x}}_i\}$ provided that we can compute the scalars α_i and β_i .

Now these scalars, from equations (5.7), (5.8) and (5.11), are obtained from ρ_i and σ_i which are themselves computed by $\rho_i = \mathbf{s}_1^T \mathbf{r}_i$ and $\sigma_i = \mathbf{v}_1^T \mathbf{A} \mathbf{u}_i$ in the BiCG algorithm and by $\rho_i = \mathbf{s}_1^T \hat{\mathbf{r}}_i$ and $\sigma_i = \mathbf{v}_1^T \mathbf{A} \hat{\mathbf{u}}_i$ in the CGS algorithm. Since for BiCGStab none of these sequences is available it is necessary to compute $\{\rho_i\}$ and $\{\sigma_i\}$ using an alternative approach.

From Lemma 23(b) we have

$$\rho_{i+1} = \lambda_{ii} \mathbf{s}_1^T \mathbf{A}^i \mathbf{r}_{i+1}$$

and from equations (5.1) and (5.21), $\tilde{\mathbf{r}}_{i+1} = \Psi_i \mathbf{r}_{i+1}$ so that choosing the arbitrary polynomial in the corollary of this lemma to be Ψ_i yields, from equation (5.20),

$$\mathbf{s}_1^T \tilde{\mathbf{r}}_{i+1} = \eta_{ii} \mathbf{s}_1^T \mathbf{A}^i \mathbf{r}_{i+1}.$$

This gives, from the previous equation,

$$\rho_{i+1} = \frac{\lambda_{ii} \mathbf{s}_1^T \tilde{\mathbf{r}}_{i+1}}{\eta_{ii}} \quad (5.27)$$

so that if λ_{ii} and η_{ii} are known, ρ_{i+1} may be computed. Similarly, from Lemma 25(b),

$$\sigma_{i+1} = \mu_{ii} \mathbf{v}_1^T \mathbf{A}^{i+1} \mathbf{u}_{i+1}$$

and from equations (5.3) and (5.21), $\tilde{\mathbf{u}}_{i+1} = \Psi_i \mathbf{u}_{i+1}$. Choosing the arbitrary polynomial in the corollary of this lemma to be Ψ_i yields, from equation (5.20),

$$\mathbf{v}_1^T \mathbf{A} \tilde{\mathbf{u}}_{i+1} = \eta_{ii} \mathbf{v}_1^T \mathbf{A}^{i+1} \mathbf{u}_{i+1}$$

which gives, from the previous equation,

$$\sigma_{i+1} = \frac{\mu_{ii} \mathbf{v}_1^T \mathbf{A} \tilde{\mathbf{u}}_{i+1}}{\eta_{ii}}. \quad (5.28)$$

It now follows immediately from equations (5.11), (5.27) and (5.28) that

$$\alpha_{i+1} = \frac{\lambda_{ii} \mathbf{s}_1^T \tilde{\mathbf{r}}_{i+1}}{\mu_{ii} \mathbf{v}_1^T \mathbf{A} \tilde{\mathbf{u}}_{i+1}} \quad \text{and} \quad \beta_{i+1} = \frac{\lambda_{ii} \eta_{i-1,i-1} \mathbf{s}_1^T \tilde{\mathbf{r}}_{i+1}}{\lambda_{i-1,i-1} \eta_{ii} \mathbf{s}_1^T \tilde{\mathbf{r}}_i} \quad (5.29)$$

so that, since $\mathbf{s}_1^T \tilde{\mathbf{r}}_i$, $\mathbf{s}_1^T \tilde{\mathbf{r}}_{i+1}$ and $\mathbf{v}_1^T \mathbf{A} \tilde{\mathbf{u}}_{i+1}$ may be computed, α_{i+1} and β_{i+1} may be determined if the ratios $\lambda_{ii}/\lambda_{i-1,i-1}$, λ_{ii}/μ_{ii} and $\eta_{ii}/\eta_{i-1,i-1}$ are known. Now λ_{ii} , μ_{ii} and η_{ii} are the coefficients of \mathbf{A}^i in the polynomials Φ_i , Θ_i and Ψ_i respectively and the required ratios may be obtained from the appropriate recursions. Thus, from equations (5.9), (5.10) and (5.24) we have $\lambda_{ii}/\mu_{i-1,i-1} = -\alpha_i$, $\mu_{ii} = \lambda_{ii}$ and $\eta_{ii}/\eta_{i-1,i-1} = -\omega_i$ so that, from equation (5.29),

$$\alpha_i = \frac{\mathbf{s}_1^T \tilde{\mathbf{r}}_i}{\mathbf{v}_1^T \mathbf{A} \tilde{\mathbf{u}}_i} \quad \text{and} \quad \beta_{i+1} = \frac{\alpha_i \mathbf{s}_1^T \tilde{\mathbf{r}}_{i+1}}{\omega_i \mathbf{s}_1^T \tilde{\mathbf{r}}_i}. \quad (5.30)$$

It remains now only to find a suitable choice for the scalars $\{\omega_i\}$ and to determine the recursion for the sequence $\{\tilde{\mathbf{x}}_i\}$ to complete the definition of the algorithm.

If we define $\tilde{\mathbf{z}}_i$ by

$$\tilde{\mathbf{z}}_i = \tilde{\mathbf{r}}_i - \alpha_i \mathbf{A} \tilde{\mathbf{u}}_i, \quad (5.31)$$

equation (5.25) may be written

$$\tilde{\mathbf{r}}_{i+1} = (\mathbf{I} - \omega_i \mathbf{A}) \tilde{\mathbf{z}}_i. \quad (5.32)$$

Since we propose to construct a sequence of approximate solutions $\{\tilde{\mathbf{x}}_i\}$ where $\tilde{\mathbf{r}}_i = \mathbf{A} \tilde{\mathbf{x}}_i - \mathbf{b}$ and since we wish to reduce the residuals $\tilde{\mathbf{r}}_i$ to zero the obvious criterion for the choice of ω_i is the minimisation of $\|\tilde{\mathbf{r}}_{i+1}\|$. Simple calculus then gives

$$\omega_i = \frac{\tilde{\mathbf{z}}_i^T \mathbf{A} \tilde{\mathbf{z}}_i}{\tilde{\mathbf{z}}_i^T \mathbf{A}^T \mathbf{A} \tilde{\mathbf{z}}_i}.$$

Finally, equation (5.25) may be written, from equation (5.31),

$$\tilde{\mathbf{r}}_{i+1} = \tilde{\mathbf{r}}_i - \mathbf{A}(\alpha_i \tilde{\mathbf{u}}_i + \omega_i \tilde{\mathbf{z}}_i) \quad (5.33)$$

so that the same argument used to establish equation (5.19) yields

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - (\alpha_i \tilde{\mathbf{u}}_i + \omega_i \tilde{\mathbf{z}}_i).$$

The algorithm finally becomes

Algorithm BiCGStab

- (1) Select $\hat{\mathbf{x}}_1$, \mathbf{s}_1 arbitrarily, compute $\tilde{\mathbf{r}}_1 = \mathbf{A}\hat{\mathbf{x}}_1 - \mathbf{b}$ and set $\tilde{\mathbf{u}}_1 = \tilde{\mathbf{r}}_1$ and $i = 1$
- (2) while not converged
 - (a) compute $\tilde{\mathbf{w}}_i = \mathbf{A}\tilde{\mathbf{u}}_i$ and $\alpha_i = \frac{\mathbf{s}_1^T \tilde{\mathbf{r}}_i}{\mathbf{s}_1^T \tilde{\mathbf{w}}_i}$
 - (b) compute $\tilde{\mathbf{z}}_i = \tilde{\mathbf{r}}_i - \alpha_i \tilde{\mathbf{w}}_i$ and $\tilde{\mathbf{q}}_i = \mathbf{A}\tilde{\mathbf{z}}_i$
 - (c) compute $\omega_i = \frac{\tilde{\mathbf{z}}_i^T \tilde{\mathbf{q}}_i}{\tilde{\mathbf{q}}_i^T \tilde{\mathbf{q}}_i}$ and $\tilde{\mathbf{r}}_{i+1} = \tilde{\mathbf{z}}_i - \omega_i \tilde{\mathbf{q}}_i$
 - (d) compute $\beta_{i+1} = \frac{\alpha_i \mathbf{s}_1^T \tilde{\mathbf{r}}_{i+1}}{\omega_i \mathbf{s}_1^T \tilde{\mathbf{r}}_i}$
 - (e) compute $\tilde{\mathbf{u}}_{i+1} = \tilde{\mathbf{r}}_{i+1} + \beta_{i+1}(\tilde{\mathbf{u}}_i - \omega_i \tilde{\mathbf{w}}_i)$
 - (f) compute $\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - (\alpha_i \tilde{\mathbf{u}}_i + \omega_i \tilde{\mathbf{z}}_i)$
 - (g) set $i = i + 1$
- (3) endwhile
- (4) end BiCGStab

5.3. Other algorithms

The algorithm QMR incorporates three innovations. One is the idea of using a quasi-minimal residual technique as a means of computing \mathbf{x}_{i+1} , the others being the unique choice of \mathbf{K} and \mathbf{G} and the use of look-ahead. In TFQMR only the first of these ideas is employed so that, as was pointed out in the original paper [116], the algorithm is not so much the transpose-free version of QMR as is implied by its acronym but more the application of the QMR technique to a variant of CGS (some authors, in fact [57], actually refer to this algorithm more logically as QMRCGS). In this algorithm a vector $\hat{\mathbf{w}}_i$ is defined by $\hat{\mathbf{w}}_i = \mathbf{A}\tilde{\mathbf{u}}_i$ and equations (5.15) and (5.18), which define certain operations performed by CGS, are replaced by

$$\hat{\mathbf{p}}_{i+1} = \hat{\mathbf{q}}_i - \alpha_i \hat{\mathbf{w}}_i \quad (5.34)$$

and

$$\hat{\mathbf{w}}_i = \mathbf{A}\hat{\mathbf{q}}_i + \beta_i(\mathbf{A}\hat{\mathbf{p}}_i + \beta_{i-1}\hat{\mathbf{w}}_{i-1}). \quad (5.35)$$

The matrix multiplications $\mathbf{A}\hat{\mathbf{u}}_i$ and $\mathbf{A}(\hat{\mathbf{q}}_i + \hat{\mathbf{p}}_{i+1})$ of equations (5.15) and (5.17) are similarly replaced by $\mathbf{A}\hat{\mathbf{p}}_{i+1}$ and $\mathbf{A}\hat{\mathbf{q}}_i$. The effect of this is to permit $\hat{\mathbf{r}}_{i+1}$ to be computed by equation (5.17) in two stages:

$$\hat{\mathbf{r}}_{i+\frac{1}{2}} = \hat{\mathbf{r}}_i - \alpha_i \mathbf{A}\hat{\mathbf{q}}_i \quad (5.36)$$

and

$$\widehat{\mathbf{r}}_{i+1} = \widehat{\mathbf{r}}_{i+\frac{1}{2}} - \alpha_i \mathbf{A} \widehat{\mathbf{p}}_{i+1}. \quad (5.37)$$

We may now define the vectors \mathbf{y} and \mathbf{s} of equation (1.23) by

$$\begin{aligned} \mathbf{y}_{2i-1} &= -\alpha_i \widehat{\mathbf{q}}_i, & \mathbf{y}_{2i} &= -\alpha_i \widehat{\mathbf{p}}_{i+1}, \\ \mathbf{s}_{2i-1} &= \widehat{\mathbf{r}}_i & \text{and} & \mathbf{s}_{2i} = \widehat{\mathbf{r}}_{i+\frac{1}{2}}. \end{aligned}$$

Equations (5.36) and (5.37) then give

$$\mathbf{s}_j = \mathbf{s}_{j-1} + \mathbf{A} \mathbf{y}_{j-1} \quad (5.38)$$

for $j = 2i$ and $j = 2i + 1$, and since this equation is just a version of equation (1.23) we may apply the techniques described in Appendix A and Chapter 6 to obtain the required sequence of approximate solutions.

An even more straightforward use of the QMR technique is its application to BiCGStab [57] since in this case no modification of the original algorithm is necessary. From Steps 2(a) - 2(c) of BiCGStab we see that if \mathbf{y}_{2i-1} , etc., are defined by

$$\begin{aligned} \mathbf{y}_{2i-1} &= -\widehat{\mathbf{u}}_i \alpha_i, & \mathbf{y}_{2i} &= -\widehat{\mathbf{z}}_i \omega_i, \\ \mathbf{s}_{2i} &= \widehat{\mathbf{z}}_i & \text{and} & \mathbf{s}_{2i+1} = \widehat{\mathbf{r}}_{i+1} \end{aligned}$$

then again the vectors \mathbf{s}_j and \mathbf{y}_j , $j = 2i$ and $j = 2i + 1$, satisfy equation (5.38) enabling the immediate application of QMR. The resulting algorithm is known as QMRCGStab.

Two other transpose-free algorithms may be mentioned here. These are the transpose-free version of BiCG due to Chan *et al* [59] and the QMRS algorithm of Freund and Szeto [121]. Since these algorithms require three matrix-vector multiplications at each step we limit the discussion to this one brief comment, referring those interested to the original publications.

5.4. Discussion

All the methods described in this chapter are based on or are directly related to the BiCG algorithm, and reflect the latter's strengths and weaknesses. Specifically they all depend on the sequences $\{\rho_i\}$ and $\{\sigma_i\}$ defined by equations (5.7) and (5.8), the only difference being in the way in which these sequences are computed. These quantities are crucial to the stability of the algorithms. If $\sigma_i = 0$ for some i then BiCG crashes due to a division by zero while if $\rho_i = 0$ it stagnates, and if either becomes small then in practice some degree of numerical instability seems to be inevitable. We now consider the effect of these possible causes of instability on the algorithms discussed in this chapter.

The CGS algorithm, the oldest of these algorithms, completely ignores the problems of instability. The quantities ρ_i and σ_i that it computes are mathematically

equivalent to those determined by BiCG, and small values lead to instabilities just as in the case of the parent algorithm. In fact the instabilities are often exaggerated in CGS to the extent that it is not usually regarded as a reliable equation solver. Its importance lies in the breaking of new ground for exploitation by others.

BiCGStab sets out consciously to reduce the instabilities inherent in CGS by replacing the matrix polynomial Φ_i^2 by $\Psi_i \Psi_i$, where Ψ_i has better numerical properties than Φ_i . However it too can fail (crash) if $\mathbf{v}_i^T \mathbf{A} \widehat{\mathbf{u}}_i = 0$ which is equivalent to BiCG crashing if $\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i = 0$. If $\mathbf{s}_i^T \widehat{\mathbf{r}}_i = 0$ and BiCG stagnates the behaviour of BiCGStab is by no means clear although it should be pointed out that if this happens the proof of Lemma 23, on which the termination proof of BiCGStab depends, breaks down. In practice the algorithm is more stable than both BiCG and CGS and is indeed one of the more successful Krylov algorithms (see Chapter 10 below). A look-ahead version that should avoid all these numerical difficulties has been proposed by Graves-Morris and Salam [132].

The same considerations apply to TFQMR (or QMRCGS). If BiCG crashes then so does TFQMR but the application of the QMR technique smooths out the algorithm in cases of near-breakdown and improves its convergence properties though not its overall rate of convergence. In the most severe cases of near-breakdown, look-ahead techniques are recommended [116].

Thus all the algorithms based on BiCG (which might reasonably be referred to collectively as the *Sonneveld algorithms* since they are all derivatives of CGS) break down if BiCG breaks down, but with the exception of CGS the behaviour of the derived algorithms is more acceptable in the cases of near-breakdown. However since failures can still occur in all these methods, the devices introduced to combat instability should perhaps be properly regarded as palliatives rather than cures. The only algorithms that can guarantee the complete avoidance of breakdown seem to be those based on the normal equations and these have their own disadvantages, either recurrences of increasing length or generally poor convergence.

This page intentionally left blank

Chapter 6

More on QMR

In this chapter we examine various aspects of QMR and its relationship to other algorithms, and begin by looking at some of the problems associated with its implementation. A crucial practical step that forms a key component of the QMR technique is the computation of the sequence of matrices $\mathbf{Y}_i \tilde{\mathbf{U}}_i^{-1}$, where $\tilde{\mathbf{U}}_i$ satisfies equation (6.8) and \mathbf{Y}_i satisfies equation (6.9). It follows from these two equations that \mathbf{Y}_{i-1} is a submatrix of \mathbf{Y}_i and $\tilde{\mathbf{U}}_{i-1}^{-1}$ is a submatrix of $\tilde{\mathbf{U}}_i^{-1}$, and it is this serendipitous property that enables us to construct a simple and effective algorithm for computing the sequence of approximate solutions generated by QMR. In fact it is convenient to give this property a name, which we do in the following definition.

Definition 3. A sequence of matrices will be called a nested sequence if every member of the sequence except the last is a submatrix of its immediate successor. A similar definition holds for vector sequences.

We have already encountered such sequences. Trivially the matrices $\bar{\mathbf{P}}_i$ of equation (1.31) form a nested sequence as do the matrices $\tilde{\mathbf{H}}_i$ defined in Chapter 2 above, but the usefulness of the concept comes when we have a sequence of vectors or matrices defined by some other criterion but which is also found to be a nested sequence. This occurs with the matrices $\tilde{\mathbf{U}}_i$ (but not the matrices $\bar{\mathbf{U}}_i$) that are computed when the matrices $\tilde{\mathbf{H}}_i$ are reduced to upper triangular form (see Appendix A). Since they form nested sequences it is possible to derive a very simple recursion formula for the calculation of the successive values of $\mathbf{Y}_i \tilde{\mathbf{U}}_i^{-1}$ and this is done in the first section of the chapter. The following sections deal with certain relationships between QMR and similar algorithms, and describe some properties of a particular application of the QMR technique.

6.1. The implementation of QMR, GMRes, SymmLQ and LSQR

The QMR technique is essentially defined by equations (1.23), (1.25) and (1.29) (see Chapter 1, the Quasi-minimal residual technique, for the relevant background).

These equations, slightly modified, are

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_1 + \mathbf{Y}_i \mathbf{v}_i, \quad (6.1)$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i \alpha_i + \mathbf{s}_{i-1} \beta_{i-1} + \mathbf{A} \mathbf{y}_i \quad (6.2)$$

and

$$\hat{\mathbf{r}}(\mathbf{v}) \equiv \tilde{\mathbf{S}}_{i+1} (\mathbf{e}_1 \sigma_1 + \tilde{\mathbf{H}}_{i+1} \mathbf{v}), \quad (6.3)$$

and the implementational task is the efficient calculation of \mathbf{v}_i and $\hat{\mathbf{x}}_{i+1}$, where \mathbf{v}_i is the value of \mathbf{v} that minimises $\|\mathbf{e}_1 \sigma_1 + \tilde{\mathbf{H}}_{i+1} \mathbf{v}\|$ over \mathbf{v} . The matrix $\mathbf{Y}_i \in \mathbb{R}^{n \times i}$ is essentially arbitrary and the columns of $\tilde{\mathbf{S}}_{i+1}$ are the normalised versions of the vectors \mathbf{s}_i , where $\mathbf{s}_1 = \hat{\mathbf{r}}_1 = \mathbf{A} \hat{\mathbf{x}}_1 - \mathbf{b}$ for some arbitrary initial approximate solution $\hat{\mathbf{x}}_1$ of $\mathbf{A} \mathbf{x} = \mathbf{b}$, and \mathbf{s}_{i+1} is computed using equation (6.2) for $i \geq 1$. The matrix $\tilde{\mathbf{H}}_{i+1} \in \mathbb{R}^{(i+1) \times i}$ is upper Hessenberg and is also derived from this equation (see equation (1.28) and the text immediately following) and $\sigma_i = \|\mathbf{s}_i\|$.

We now establish the precise algorithmic mechanism for determining \mathbf{v}_i and $\hat{\mathbf{x}}_{i+1}$. The first step is the derivation of a particular expression for \mathbf{v}_i . To effect this we reduce the vector $\sigma_1 \mathbf{e}_1 + \tilde{\mathbf{H}}_{i+1} \mathbf{v}$ to the form

$$\begin{bmatrix} \tilde{\mathbf{c}}_i \\ \bar{\gamma}_{i+1} \end{bmatrix} + \begin{bmatrix} \tilde{\mathbf{U}}_i \\ \mathbf{0}^T \end{bmatrix} \mathbf{v}$$

by orthogonal transformations as these leave its norm unchanged (full details of this process are given in Appendix A). It then follows immediately that this norm is minimised by choosing \mathbf{v}_i to satisfy

$$\tilde{\mathbf{U}}_i \mathbf{v}_i + \tilde{\mathbf{c}}_i = \mathbf{0} \quad (6.4)$$

and its value at the minimum is $|\bar{\gamma}_{i+1}|$. It would now be quite possible in principle, since $\tilde{\mathbf{U}}_i$ is upper triangular, to compute \mathbf{v}_i by solving equation (6.4) and then to compute $\hat{\mathbf{x}}_{i+1}$ by substituting the value of \mathbf{v}_i so obtained in equation (6.1). However this approach would require the storage of both $\tilde{\mathbf{U}}_i$ and \mathbf{Y}_i in full, and would not be feasible for large problems. A better method is based on the observation that equations (6.1) and (6.4) give

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_1 - \mathbf{Y}_i \tilde{\mathbf{U}}_i^{-1} \tilde{\mathbf{c}}_i \quad (6.5)$$

or, if we define \mathbf{Z}_i by

$$\mathbf{Z}_i = \mathbf{Y}_i \tilde{\mathbf{U}}_i^{-1}, \quad (6.6)$$

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_1 - \mathbf{Z}_i \tilde{\mathbf{c}}_i. \quad (6.7)$$

We now show how the matrices $\{\mathbf{Z}_i\}$ can be generated recursively, and that only one or two columns of the previously computed matrix need be stored at any one time.

The matrices $\{\tilde{\mathbf{U}}_i\}$ form a nested sequence and satisfy (see Appendix A, equation (A.4))

$$\tilde{\mathbf{U}}_i = \begin{bmatrix} \tilde{\mathbf{U}}_{i-1} & \tilde{\mathbf{u}}_i \\ \mathbf{0}^T & \tilde{u}_{ii} \end{bmatrix} \quad (6.8)$$

so that

$$\tilde{\mathbf{U}}_i^{-1} = \begin{bmatrix} \tilde{\mathbf{U}}_{i-1}^{-1} & -\tilde{\mathbf{U}}_{i-1}^{-1}\tilde{\mathbf{u}}_i\tilde{u}_{ii}^{-1} \\ \mathbf{0}^T & \tilde{u}_{ii}^{-1} \end{bmatrix}$$

Thus, from equation (6.6), and since

$$\mathbf{Y}_i = [\mathbf{Y}_{i-1} \ \mathbf{y}_i] \quad (6.9)$$

we have

$$\mathbf{Z}_i = [\mathbf{Z}_{i-1} \ \mathbf{z}_i]$$

where

$$\mathbf{z}_i = (\mathbf{y}_i - \mathbf{Z}_{i-1}\tilde{\mathbf{u}}_i)\tilde{u}_{ii}^{-1}. \quad (6.10)$$

The matrices $\{\mathbf{Z}_i\}$ thus also form a nested sequence. But, from Appendix A, the vectors $\{\tilde{\mathbf{c}}_i\}$ too form a nested sequence so that

$$\tilde{\mathbf{c}}_i^T = [\tilde{\mathbf{c}}_{i-1}^T \ \tilde{\gamma}_i] \quad (6.11)$$

and it follows from this and equation (6.7) that

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_1 - \mathbf{Z}_{i-1}\tilde{\mathbf{c}}_{i-1} - \mathbf{z}_i\tilde{\gamma}_i$$

or

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - \mathbf{z}_i\tilde{\gamma}_i. \quad (6.12)$$

If the sequence $\{\mathbf{s}_i\}$ has been computed from equation (6.2) the matrix $\tilde{\mathbf{H}}_{i+1}$ will be tridiagonal so that, from the discussion in Appendix A, $\tilde{u}_{jk} = 0$ for $1 \leq j \leq k-3$. It follows from equations (6.8) and (6.10) that it is then necessary to store only the last two columns of \mathbf{Z}_{i-1} when computing \mathbf{z}_i since only the last two elements of $\tilde{\mathbf{u}}_i$ are nonzero. *A fortiori* if, in equation (6.2), $\beta_{i-1} = 0$ for all i then

$$\mathbf{z}_i = (\mathbf{y}_i - \mathbf{z}_{i-1}\tilde{u}_{i-1,i})\tilde{u}_{ii}^{-1} \quad (6.13)$$

and only \mathbf{z}_{i-1} need be stored.

In the QMR methods described here the vectors \mathbf{y}_j are generated using one of the Krylov subspace methods although there seems to be no compelling reason why this should be so in general. The particular choices of the vectors \mathbf{y}_j for the individual methods discussed here are given in the sections relating to these methods.

For SymmLQ, in order to evaluate \mathbf{x}_{i+1} from equation (3.53), we have to compute, $\mathbf{r}_i^T \tilde{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1} \mathbf{e}_i$ where \mathbf{e}_i denotes the final column of the unit matrix of order i . If, by analogy with equation (6.6), we define \mathbf{Z}_i by $\mathbf{Z}_i = \tilde{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1}$ we obtain

$$\mathbf{r}_i^T \tilde{\mathbf{P}}_i \tilde{\mathbf{U}}_i^{-1} \mathbf{e}_i = \mathbf{r}_i^T \mathbf{z}_i$$

where, by analogy with equation (6.10),

$$\mathbf{z}_i = (\mathbf{p}_i - \mathbf{Z}_{i-1} \tilde{\mathbf{u}}_i) \tilde{u}_{ii}^{-1}$$

since both $\{\tilde{\mathbf{P}}_i\}$ and $\{\mathbf{Z}_i\}$ are nested sequences. The calculation of \mathbf{x}_{i+1} then follows, and for LSQR the calculation of \mathbf{x}_{i+1} by equations (3.64) and (3.65) is similar.

6.2. QMRBiCG - an alternative form of QMR without look-ahead

In the original version of QMR [118] the displacement vectors \mathbf{u}_i and \mathbf{v}_i are computed using the Lanczos equations (3.67) and (3.68). With exact arithmetic this process only breaks down if one of the matrices $\hat{\mathbf{S}}_j$ defined by equations (4.1) and (4.2) is singular and such breakdowns can usually be resolved by look-ahead. Despite the fact that, in exact arithmetic, there is a greater chance of breakdown using the HS equations it is observed in practice (see [144] and Chapter 9 below) that in the presence of rounding errors the HS versions actually perform better than their Lanczos counterparts. In the light of this it is hardly surprising that attempts were made to apply the QMR technique to vectors generated by an HS version of QMR, the main problem being that no such version exists. The reason for this is that the QMR choice of \mathbf{G} leads to displacement vectors that are bi-orthogonal rather than bi-conjugate (as they would be for BiCG) so that equations (3.12) and (3.13) involve \mathbf{A}^{-1} and are hence inadmissible. A possible way out of the problem is to apply the QMR technique to vectors generated by the raw version of BiCG (which is closely related to QMR - see below), and this has indeed been suggested. Freund and Nachtigal, however [119], proposed a version that they regard as being based on coupled two-term recurrences. Our derivation shows that this version may equally be regarded as the application of the QMR technique to a scaled implementation of BiCG.

We start with equations (3.17) and (3.18) which define the alternative form of the block CG method, but we now regard the matrices \mathbf{F}_i not as residuals of the equation $\mathbf{GX} = \mathbf{H}$ but as matrices in their own right. Note that if \mathbf{G} , \mathbf{K} , \mathbf{P}_1 and \mathbf{F}_1 are given, together with the nonsingular but otherwise arbitrary scaling matrices \mathbf{B}_i , then equations (3.17) and (3.18) completely define the sequences $\{\mathbf{F}_i\}$ and $\{\mathbf{P}_i\}$ so that from the point of view of generating these sequences the matrices \mathbf{X}_i are irrelevant. Define now the matrices $\tilde{\mathbf{F}}_i$ and $\tilde{\mathbf{C}}_i$ by

$$\tilde{\mathbf{F}}_i = \mathbf{F}_i \mathbf{B}_i \quad \text{and} \quad \tilde{\mathbf{C}}_i = \tilde{\mathbf{F}}_i^T \mathbf{K} \tilde{\mathbf{F}}_i. \quad (6.14)$$

Then, since $\mathbf{C}_i = \mathbf{F}_i^T \mathbf{K} \mathbf{F}_i$,

$$\mathbf{F}_i = \tilde{\mathbf{F}}_i \mathbf{B}_i^{-1} \quad \text{and} \quad \mathbf{C}_i = \mathbf{B}_i^{-T} \tilde{\mathbf{C}}_i \mathbf{B}_i^{-1},$$

and substituting these values of \mathbf{F}_i and \mathbf{C}_i into equations (3.17) and (3.18) gives, after a little manipulation,

$$\tilde{\mathbf{F}}_{i+1} = \left(\tilde{\mathbf{F}}_i \tilde{\mathbf{C}}_i^{-1} \mathbf{D}_i - \mathbf{G} \mathbf{P}_i \right) \mathbf{D}_i^{-1} \tilde{\mathbf{C}}_i \mathbf{B}_i^{-1} \mathbf{B}_{i+1} \quad (6.15)$$

and

$$\mathbf{P}_{i+1} = \mathbf{K} \tilde{\mathbf{F}}_{i+1} + \mathbf{P}_i \tilde{\mathbf{C}}_i^{-1} \mathbf{B}_i^T \mathbf{B}_{i+1}^{-T} \tilde{\mathbf{C}}_{i+1}. \quad (6.16)$$

Define now \mathbf{S}_{i+1} by

$$\mathbf{S}_{i+1} = \mathbf{D}_i^{-1} \tilde{\mathbf{C}}_i \mathbf{B}_i^{-1} \mathbf{B}_{i+1} \quad (6.17)$$

so that equation (6.15) becomes

$$\tilde{\mathbf{F}}_{i+1} = \left(\tilde{\mathbf{F}}_i \tilde{\mathbf{C}}_i^{-1} \mathbf{D}_i - \mathbf{G} \mathbf{P}_i \right) \mathbf{S}_{i+1}. \quad (6.18)$$

Now both \mathbf{B}_i and \mathbf{B}_{i+1} are arbitrary subject only to being nonsingular so we may assume that \mathbf{B}_1 and \mathbf{S}_{i+1} , $i = 1, 2, \dots$, are arbitrary and use equation (6.17) to express the matrices \mathbf{B}_{i+1} in terms of \mathbf{S}_{i+1} , $i \geq 1$. Since $\tilde{\mathbf{C}}_i$ and \mathbf{D}_i are both symmetric this equation may be written

$$\mathbf{D}_i^{-1} \mathbf{S}_{i+1}^{-T} = \tilde{\mathbf{C}}_i^{-1} \mathbf{B}_i^T \mathbf{B}_{i+1}^{-T}$$

and substituting this expression in equation (6.16) gives

$$\mathbf{P}_{i+1} = \mathbf{K} \tilde{\mathbf{F}}_{i+1} + \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{S}_{i+1}^{-T} \tilde{\mathbf{C}}_{i+1}. \quad (6.19)$$

Equations (6.18) and (6.19) thus enable us to compute the sequences $\{\tilde{\mathbf{F}}_i\}$ and $\{\mathbf{P}_i\}$ in terms of the initial values and the arbitrary scaling matrices \mathbf{S}_i , the original arbitrary scaling matrices \mathbf{B}_i having been eliminated.

Let now

$$\begin{aligned} \tilde{\mathbf{F}}_i &= \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{s}}_i \\ \tilde{\mathbf{r}}_i & \mathbf{0} \end{bmatrix}, & \mathbf{P}_i &= \begin{bmatrix} \mathbf{u}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_i \end{bmatrix}, \\ \mathbf{K} &= \begin{bmatrix} \mathbf{O} & \mathbf{M}^{-1} \\ \mathbf{M}^{-T} & \mathbf{O} \end{bmatrix} \quad \text{and} \quad \mathbf{S}_i &= \begin{bmatrix} 1/\rho_i & 0 \\ 0 & 1/\xi_i \end{bmatrix} \end{aligned}$$

where \mathbf{M} is nonsingular but otherwise arbitrary and ρ_i and ξ_i are non-zero but otherwise arbitrary. Note that we have included the preconditioning matrix \mathbf{M} to follow more closely the discussion in [119]. Note also that the scaling matrix \mathbf{S}_i represents a simple scaling of the sequences $\{\tilde{\mathbf{r}}_i\}$ and $\{\tilde{\mathbf{s}}_i\}$. Substituting these values, together with the value of \mathbf{G} appropriate to the BiCG algorithm and given on page 52, into equations (6.18) and (6.19) then yields

$$\mathbf{u}_{i+1} = \mathbf{M}^{-1} \tilde{\mathbf{r}}_{i+1} + \mathbf{u}_i \left(\frac{\tilde{\mathbf{s}}_{i+1}^T \mathbf{M}^{-1} \tilde{\mathbf{r}}_{i+1}}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \xi_{i+1} \quad (6.20)$$

with a similar equation for \mathbf{v}_{i+1} , and

$$\tilde{\mathbf{r}}_{i+1} = \left[\tilde{\mathbf{r}}_i \left(\frac{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i}{\tilde{s}_i^T \mathbf{M}^{-1} \tilde{\mathbf{r}}_i} \right) - \mathbf{A} \mathbf{u}_i \right] / \rho_{i+1}, \quad (6.21)$$

with a similar equation for \tilde{s}_{i+1} . If we now choose our arbitrary scaling factors ρ_i and ξ_i so that $\|\mathbf{M}_1^{-1} \tilde{\mathbf{r}}_i\| = \|\mathbf{M}_2^{-T} \tilde{s}_i\| = 1$, where \mathbf{M}_1 and \mathbf{M}_2 are yet more arbitrary matrices satisfying $\mathbf{M}_1 \mathbf{M}_2 = \mathbf{M}$, we obtain the identical equations, *mutatis mutandis*, to those obtained by Freund and Nachtigal [119] for their Algorithm 7.1.

We can now use equation (6.17) to examine the properties of the matrices $\{\tilde{\mathbf{F}}_i\}$. Let

$$\mathbf{B}_1 = \begin{bmatrix} \beta_1 & 0 \\ 0 & \gamma_1 \end{bmatrix}.$$

This is chosen so that $\tilde{\mathbf{r}}_1 = \mathbf{r}_1 \beta_1$, $\tilde{s}_1 = \mathbf{s}_1 \gamma_1$ and $\|\mathbf{M}_1^{-1} \tilde{\mathbf{r}}_1\| = \|\mathbf{M}_2^{-T} \tilde{s}_1\| = 1$, where \mathbf{s}_1 is arbitrary, $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$ and \mathbf{x}_1 is the arbitrary initial approximation of the solution of $\mathbf{Ax} = \mathbf{b}$. From equation (6.17) the matrices \mathbf{B}_{i+1} , $i = 1, 2, \dots$, are given by $\mathbf{B}_{i+1} = \mathbf{B}_i \tilde{\mathbf{C}}_i^{-1} \mathbf{D}_i \mathbf{S}_{i+1}$. Since $\tilde{\mathbf{C}}_i$ and \mathbf{D}_i are both skew-diagonal, \mathbf{B}_1 is diagonal and \mathbf{S}_{i+1} is diagonal for all i it follows that the matrices \mathbf{B}_i are also diagonal for all i . This implies from the first equation of (6.14) that the vectors $\tilde{\mathbf{r}}_i$ and \tilde{s}_i are merely scaled versions of the vectors \mathbf{r}_i and \mathbf{s}_i that would be generated by BiCG. Whether or not the scaling suggested in [119] gives any material advantage over a simple scaling (e.g. $\|\tilde{\mathbf{r}}_i\| = \|\tilde{s}_i\| = 1$), or even no scaling at all, is an open question since the precise rôles of \mathbf{M}_1 and \mathbf{M}_2 are unclear. Our limited comparisons suggest not (see Chapter 10 below). However experimental results [119] do show that QMR applied to a (scaled) version of BiCG is usually more reliable than the original QMR algorithm [118], where QMR is applied to vectors generated by a Lanczos-type recursion.

The remainder of the calculation (the evaluation of the sequence of approximate solutions $\{\mathbf{x}_i\}$) is carried out by noting that equation (6.21) is merely equation (6.2) with $\mathbf{s}_i = \tilde{\mathbf{r}}_i$ (this \mathbf{s}_i is not to be confused with the shadow residuals of the present section, also denoted by \mathbf{s}_i), $\mathbf{y}_i = -\mathbf{u}_i / \rho_{i+1}$ and $\beta_{i-1} = 0$. The QMR technique is then applied to those vectors as described in the previous section.

Finally we note that for simplicity we have only described the version of the algorithm without look-ahead. This refinement is fully described in [119].

6.3. Simplified (symmetric) QMR

Another variant of the QMR algorithm is that designed to solve symmetric indefinite systems [122], [120]. Such systems are often surprisingly difficult to solve. Definite systems normally yield to PCG (preconditioned CG), especially if a good positive definite preconditioner is available, but indefinite systems have proved far less tractable. Simply applying CG with the Luenberger-Fletcher look-ahead technique [110], [180], does not seem to have been too successful and the performances

of algorithms like SymmLQ and MinRes, algorithms that would appear to be ideally suited to such systems, have been distinctly patchy. This is partly because, as was pointed out in ([120]), they cannot be used with indefinite preconditioners (see Symmetric systems and Galerkin methods, pages 192 and 195, below). This effectively eliminates the use of the “perfect preconditioner” \mathbf{A}^{-1} , which is of course indefinite if \mathbf{A} is indefinite, and this aspect of the algorithms casts doubts on their ultimate viability. This is not the case with the algorithm we are about to describe which applies QMR (either the original Lanczos version or QMRBiCG) to a system of *nonsymmetric* equations derived from the symmetric one. The special nature of the nonsymmetric systems thus created is then exploited to modify QMR by using an ingenious device described by Parlett and Chen [205] in order to reduce computing time and storage requirements.

Assume then that we wish to solve

$$\mathbf{B}\mathbf{z} = \mathbf{c} \quad (6.22)$$

where $\mathbf{B} \in \mathbb{R}^{n \times n}$ is symmetric but not definite and $\mathbf{c}, \mathbf{z} \in \mathbb{R}^n$. Let \mathbf{M} be a symmetric nonsingular indefinite preconditioner which may be expressed as the product of two (generally unsymmetric) real and nonsingular factors \mathbf{M}_1 and \mathbf{M}_2 . Hence, from symmetry,

$$\mathbf{M} = \mathbf{M}_1 \mathbf{M}_2 = \mathbf{M}_2^T \mathbf{M}_1^T. \quad (6.23)$$

Equation (6.22) may now be written

$$(\mathbf{M}_1^{-1} \mathbf{B} \mathbf{M}_2^{-1}) \mathbf{M}_2 \mathbf{z} = \mathbf{M}_1^{-1} \mathbf{c}$$

and putting

$$\mathbf{A} = \mathbf{M}_1^{-1} \mathbf{B} \mathbf{M}_2^{-1}, \quad (6.24)$$

$\mathbf{b} = \mathbf{M}_1^{-1} \mathbf{c}$ and $\mathbf{x} = \mathbf{M}_2 \mathbf{z}$ gives simply $\mathbf{Ax} = \mathbf{b}$, the system of equations to be solved by QMR.

For simplicity we consider only the original QMR algorithm, without scaling or look-ahead, as defined by equations (3.67) and (3.68). These may be written

$$\mathbf{u}_{i+1} = \mathbf{A} \mathbf{u}_i - \mathbf{u}_i \alpha_i - \mathbf{u}_{i-1} \beta_{i-1} \quad (6.25)$$

and

$$\mathbf{v}_{i+1} = \mathbf{A}^T \mathbf{v}_i - \mathbf{v}_i \alpha_i - \mathbf{v}_{i-1} \beta_{i-1}, \quad (6.26)$$

where the coefficients multiplying \mathbf{u}_{i-1} in equation (6.25) and \mathbf{v}_{i-1} in equation (6.26) are the same. This follows from equation (3.3) with $j = i$. Substituting in this equation

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{u}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_i \end{bmatrix}$$

and the values of \mathbf{G} and \mathbf{K} appropriate to QMR (see page 69) gives

$$\mathbf{u}_{i-1}^T \mathbf{A}^T \mathbf{v}_i = \mathbf{v}_{i-1}^T \mathbf{A} \mathbf{u}_i = \mathbf{v}_i^T \mathbf{u}_i$$

so that $\beta_{i-1} = \mathbf{v}_i^T \mathbf{u}_i / \mathbf{v}_{i-1}^T \mathbf{u}_{i-1}$ in both cases⁶.

We now see how the particular structure of \mathbf{A} allows us to simplify QMR. Define a matrix \mathbf{J} to be $\mathbf{M}_1^T \mathbf{M}_2^{-1}$ so that, from equation (6.23),

$$\mathbf{J} = \mathbf{M}_1^T \mathbf{M}_2^{-1} = \mathbf{M}_2^{-T} \mathbf{M}_1. \quad (6.27)$$

Since \mathbf{B} is symmetric we have, from equation (6.24), $\mathbf{A}^T = \mathbf{M}_2^{-T} \mathbf{B} \mathbf{M}_1^{-T}$ so that, from equations (6.24) and (6.27),

$$\mathbf{JA} = \mathbf{M}_2^{-T} \mathbf{B} \mathbf{M}_2^{-1} = \mathbf{A}^T \mathbf{J}.$$

The equation

$$\mathbf{JA} = \mathbf{A}^T \mathbf{J}. \quad (6.28)$$

is the key equation permitting the required simplification of QMR.

To exploit this equation, premultiply equation (6.25) by \mathbf{J} to give

$$\mathbf{Ju}_{i+1} = \mathbf{A}^T \mathbf{Ju}_i - \mathbf{Ju}_i \alpha_i - \mathbf{Ju}_{i-1} \beta_{i-1}.$$

If $\mathbf{Ju}_j = \mathbf{v}_j$ for $j = i-1, i$, this gives

$$\mathbf{Ju}_{i+1} = \mathbf{A}^T \mathbf{v}_i - \mathbf{v}_i \alpha_i - \mathbf{v}_{i-1} \beta_{i-1}$$

so that, from equation (6.26), $\mathbf{Ju}_{i+1} = \mathbf{v}_{i+1}$. Thus to generate the sequence $\{\mathbf{v}_j\}$ it suffices merely to compute the vectors \mathbf{Ju}_j for all j .

In practice the equations used to compute \mathbf{u}_{i+1} and \mathbf{v}_{i+1} are more complicated, even without look-ahead, as it is necessary to normalise the vectors \mathbf{u}_i and \mathbf{v}_i . This changes the values of the scalars α_i and β_{i-1} and makes the algebra more untidy, but the basic algorithm remains unaltered. The matrix \mathbf{J} usually has a simple structure and fewer nonzero elements than \mathbf{A} , making the calculation of \mathbf{v}_{i+1} even simpler. For a particular example given in [120] the diagonal elements of \mathbf{B} were all nonzero but of varying sign. The matrix \mathbf{M}_1 was set to the identity and \mathbf{M}_2 was chosen to be that diagonal matrix for which the diagonal elements of \mathbf{BM}_2^{-1} were unity. Thus \mathbf{J} was indefinite diagonal and the calculation of \mathbf{v}_{i+1} from \mathbf{u}_{i+1} was trivial. Matrices that satisfy equation (6.28) are called *J-symmetric*, or *J-Hermitian* if $\mathbf{JA} = \mathbf{A}^H \mathbf{J}$, and Freund and Nachtigal showed that a simplified scheme can be devised for these matrices as well. Certain classes of matrix are naturally *J-symmetric*. *Persymmetric* matrices satisfy equation (6.26) by definition where

$$\mathbf{J} = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \dots & 0 & 0 \\ 1 & 0 & \dots & 0 & 0 \end{bmatrix}$$

⁶ This simplification of QMR is presumably not normally recommended on numerical grounds.

and since *Toeplitz* matrices are persymmetric they too are J -symmetric. Such systems may be preconditioned by *circulant matrices* which are themselves particular kinds of Toeplitz matrices. See [120] for further examples. Finally we note that although we have described the technique in the context of the original Lanczos-based algorithm it may equally, with appropriate modification, be applied to the one based on coupled recurrences (QMRBiCG) or indeed to other algorithms like BiCG. Again we refer the reader to the original paper for a fuller discussion.

6.4. QMR and BiCG

In this section we examine more closely the relationships between the original three-term Lanczos version of QMR and BiCG, and we do this by comparing the basic recurrence relations for QMR and BiCGL. Since BiCG and BiCGL are equivalent (they are just the HS and Lanczos forms of the same algorithm), any connection that we can establish between QMR and BiCGL carries over into BiCG, and we shall show that if the initial conditions of QMR and BiCGL are chosen appropriately then these two algorithms generate identical sequences of displacement vectors $\{\mathbf{u}_i\}$. Other aspects of the relationship between QMR and BiCG are considered in [80] and [134].

If we denote quantities pertaining to QMR by a tilde, equation (3.68) may be written

$$\tilde{\mathbf{v}}_{i+1} = \mathbf{A}^T \tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_i \left(\frac{\tilde{\mathbf{v}}_i^T \mathbf{A} \tilde{\mathbf{u}}_i}{\tilde{\mathbf{v}}_i^T \tilde{\mathbf{u}}_i} \right) - \tilde{\mathbf{v}}_{i-1} \left(\frac{\tilde{\mathbf{v}}_i^T \mathbf{A} \tilde{\mathbf{u}}_{i-1}}{\tilde{\mathbf{v}}_{i-1}^T \tilde{\mathbf{u}}_{i-1}} \right) \quad (6.29)$$

while the comparable equation for BiCGL, which in its alternative form appears as equation (3.38), is

$$\mathbf{v}_{i+1} = \mathbf{A}^T \mathbf{v}_i - \mathbf{v}_i \left(\frac{\mathbf{v}_i^T \mathbf{A}^2 \mathbf{u}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) - \mathbf{v}_{i-1} \left(\frac{\mathbf{v}_i^T \mathbf{A}^2 \mathbf{u}_{i-1}}{\mathbf{v}_{i-1}^T \mathbf{A} \mathbf{u}_{i-1}} \right). \quad (6.30)$$

Let now $\tilde{\mathbf{u}}_j = \mathbf{u}_j$ and $\tilde{\mathbf{v}}_j = \mathbf{A}^T \mathbf{v}_j$ for $j = i-1, i$. Equation (6.29) becomes

$$\tilde{\mathbf{v}}_{i+1} = \mathbf{A}^T \left[\mathbf{A}^T \mathbf{v}_i - \mathbf{v}_i \left(\frac{\mathbf{v}_i^T \mathbf{A}^2 \mathbf{u}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) - \mathbf{v}_{i-1} \left(\frac{\mathbf{v}_i^T \mathbf{A}^2 \mathbf{u}_{i-1}}{\mathbf{v}_{i-1}^T \mathbf{A} \mathbf{u}_{i-1}} \right) \right]$$

or, from equation (6.30), $\tilde{\mathbf{v}}_{i+1} = \mathbf{A}^T \mathbf{v}_{i+1}$ and a similar argument based on the recursions for $\tilde{\mathbf{u}}_i$ and \mathbf{u}_i shows that $\tilde{\mathbf{u}}_{i+1} = \mathbf{u}_{i+1}$. An even simpler argument shows that if $\tilde{\mathbf{u}}_1 = \mathbf{u}_1$ and $\tilde{\mathbf{v}}_1 = \mathbf{A}^T \mathbf{v}_1$ then $\tilde{\mathbf{u}}_2 = \mathbf{u}_2$ and $\tilde{\mathbf{v}}_2 = \mathbf{A}^T \mathbf{v}_2$ so that if these initial conditions are satisfied then $\tilde{\mathbf{u}}_i = \mathbf{u}_i$ and $\tilde{\mathbf{v}}_i = \mathbf{A}^T \mathbf{v}_i$ for all i . Thus in terms of the calculation of the sequences $\{\tilde{\mathbf{u}}_i\}$ and $\{\mathbf{u}_i\}$, QMR and BiCGL are essentially the same algorithm. They only differ in the way they compute $\{\mathbf{x}_i\}$.

For BiCGL the vectors $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ are biconjugate with respect to \mathbf{A} and it is thus possible to generate the sequence $\{\mathbf{x}_i\}$ using equation (3.36). For QMR, $\{\tilde{\mathbf{u}}_i\}$ and $\{\tilde{\mathbf{v}}_i\}$ are bi-orthogonal so this option for computing $\{\mathbf{x}_i\}$ is not available. In this case an additional technique like the QMR technique has to be employed. Now

there is no reason why this same technique should not be applied to BiCGL since equation (3.37) has the same form as equation (3.67), but for BiCGL and BiCG another possibility presents itself, and this is the application of the QMR technique to equation (3.32) with $s_i = r_i$ and $y_i = -u_i \left(\frac{s_i^T r_i}{v_i^T A u_i} \right)$. When this version of equation (6.2) is used the properties of QMR are particularly simple, and in our next section we explore some of the consequences of this choice.

6.5. QMR and MRS

We now consider the behaviour of the QMR technique when, in equation (6.2), $\alpha_i = 1$ and $\beta_{i-1} = 0$ for all i so that

$$s_{i+1} = s_i + A y_i. \quad (6.31)$$

This would be the case if we applied the QMR technique to any algorithm that itself generates a sequence of approximate residuals r_i using an equation similar to equation (3.32), e.g. BiCG, HG or even the original CG algorithm. It could equally be applied if the vectors y_i were random provided that the vectors s_i satisfied equation (6.31).

Let then v_{i-1} be the value of v that minimises $\|e_1 \sigma_1 + \tilde{H}_i v\|$ over v , where i has been substituted for $i + 1$ in equation (6.3) in order to simplify the notation. From the discussion in Chapter 1 (Norm-reducing methods) or from first principles it follows that

$$v_{i-1} = - \left(\tilde{H}_i^T \tilde{H}_i \right)^{-1} \tilde{H}_i^T e_1 \sigma_1,$$

and if the corresponding quasi-minimal residual is denoted by \hat{r}_i equation (6.3) yields

$$\hat{r}_i = \tilde{S}_i \tilde{Q}_i e_1 \sigma_1 \quad (6.32)$$

where

$$Q_i = I - \tilde{H}_i (\tilde{H}_i^T \tilde{H}_i)^{-1} \tilde{H}_i^T.$$

The matrix Q_i is of order i since $\tilde{H}_i \in \mathbb{R}^{i \times (i-1)}$ but since, trivially, $Q_i \tilde{H}_i = \mathbf{0}$ and since \tilde{H}_i has rank $i - 1$ the nullity of Q_i cannot be less than $i - 1$. Its rank therefore must be either unity or zero and since it is not null its rank can only be unity. Thus

$$Q_i = \frac{\mathbf{z}_i \mathbf{z}_i^T}{\mathbf{z}_i^T \mathbf{z}_i} \quad (6.33)$$

where \mathbf{z}_i is the non-null vector, unique apart from a scaling factor, that satisfies

$$\mathbf{z}_i^T \tilde{H}_i = \mathbf{0}^T. \quad (6.34)$$

Let $\mathbf{z}_i = [\zeta_j]$ and normalise \mathbf{z}_i so that $\zeta_1 = 1$. Equation (6.32) becomes

$$\hat{\mathbf{r}}_i = \tilde{\mathbf{S}}_i \mathbf{z}_i \frac{\sigma_1}{\rho_i^2} \quad (6.35)$$

where

$$\rho_i^2 = \mathbf{z}_i^T \mathbf{z}_i. \quad (6.36)$$

Now $\tilde{\mathbf{H}}_i$ is upper Hessenberg so that

$$\tilde{\mathbf{H}}_i = \begin{bmatrix} \tilde{\mathbf{H}}_{i-1} & \tilde{\mathbf{h}}_i \\ \mathbf{0}^T & \tilde{h}_{i,i-1} \end{bmatrix}$$

and it follows that if $\mathbf{z}_{i-1}^T \tilde{\mathbf{H}}_{i-1} = \mathbf{0}^T$ and $\mathbf{z}_i^T = [\mathbf{z}_{i-1}^T \ \zeta_i]$ then $\mathbf{z}_i^T \tilde{\mathbf{H}}_i = \mathbf{0}^T$ if

$$\zeta_i = \frac{-\mathbf{z}_{i-1}^T \tilde{\mathbf{h}}_i}{\tilde{h}_{i,i-1}}. \quad (6.37)$$

Thus the vectors \mathbf{z}_i form a nested sequence and it is only necessary to compute ζ_i in order to compute \mathbf{z}_i from \mathbf{z}_{i-1} . It also follows from this and equation (6.36) that $\rho_i^2 = \rho_{i-1}^2 + \zeta_i^2$.

Now if, as we assumed,

$$\mathbf{s}_i = \mathbf{s}_{i-1} + \mathbf{A} \mathbf{y}_{i-1} \quad (6.38)$$

it follows from the discussion of QMR in Chapter 1 that

$$\tilde{\mathbf{H}}_i = \begin{bmatrix} -\sigma_1 & 0 & \dots & 0 \\ \sigma_2 & -\sigma_2 & \dots & 0 \\ 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\sigma_{i-1} \\ 0 & 0 & \dots & \sigma_i \end{bmatrix}$$

where $\sigma_i = \|\mathbf{s}_i\|$ so that equation (6.37) becomes $\zeta_i = \zeta_{i-1}(\sigma_{i-1}/\sigma_i)$. From this it is trivial to deduce that, since $\zeta_1 = 1$,

$$\zeta_i = \sigma_1/\sigma_i.$$

Thus if $\tilde{\mathbf{s}}_i$ denotes the $i-th$ column of $\tilde{\mathbf{S}}_i$ we have, from equation (6.35) and since $\tilde{\mathbf{s}}_i = \mathbf{s}_i/\sigma_i$,

$$\begin{aligned} \hat{\mathbf{r}}_i \rho_i^2 &= \left(\tilde{\mathbf{S}}_{i-1} \mathbf{z}_{i-1} + \tilde{\mathbf{s}}_i \zeta_i \right) \sigma_1 \\ &= \tilde{\mathbf{S}}_{i-1} \mathbf{z}_{i-1} \sigma_1 + \mathbf{s}_i \zeta_i \left(\frac{\sigma_1}{\sigma_i} \right) \\ &= \hat{\mathbf{r}}_{i-1} \rho_{i-1}^2 + \mathbf{s}_i \zeta_i^2 \end{aligned}$$

$$= \sum_{j=1}^i s_j \zeta_j^2$$

so that

$$\hat{\mathbf{r}}_i = \hat{\mathbf{r}}_{i-1} \left(\frac{\rho_{i-1}^2}{\rho_i^2} \right) + s_i \left(\frac{\zeta_i^2}{\rho_i^2} \right) \quad (6.39)$$

or

$$\hat{\mathbf{r}}_i = \sum_{j=1}^i s_j \left(\frac{\zeta_j^2}{\rho_i^2} \right). \quad (6.40)$$

Since $\sum_{j=1}^i (\zeta_j^2 / \rho_i^2) = 1$ this equation shows that the residual $\hat{\mathbf{r}}_i$ computed by applying the this version of the QMR technique to BiCG, Hegedüs, etc. is a convex combination of the residuals s_j that would be normally computed by the original algorithm. Moreover, since $\zeta_j = \sigma_1 / \sigma_j$, the weights of the individual components are inversely proportional to the squares of their norms, guaranteeing an effective and stable algorithm.

Since the coefficients in the right-hand side of equation (6.39) sum to unity it follows that if $\hat{\mathbf{r}}_{i-1} = \mathbf{A}\hat{\mathbf{x}}_{i-1} - \mathbf{b}$ and $s_i = \mathbf{A}\mathbf{x}_i - \mathbf{b}$ then

$$\hat{\mathbf{x}}_i = \hat{\mathbf{x}}_{i-1} \left(\frac{\rho_{i-1}^2}{\rho_i^2} \right) + \mathbf{x}_i \left(\frac{\zeta_i^2}{\rho_i^2} \right) \quad (6.41)$$

and this equation may be used to compute $\hat{\mathbf{x}}_i$. This approach therefore gives us an alternative to the method based on orthogonal transformations for implementing QMR in the case where equation (6.31) holds, and is loosely based on the work of Zhou and Walker [261].

Now the application of QMR to a basic algorithm undeniably improves the behaviour of the residual norms but it does not guarantee that they converge monotonically. Since monotonic convergence is often regarded as desirable it is not surprising that “add-ons” have been devised that do guarantee this property, one such being the minimal residual smoothing algorithm (MRS) proposed by Schonhauer [221] and developed by Weiss [255]. Suppose then that we already have a sequence $\{\mathbf{x}_i\}$ of approximate solutions of $\mathbf{Ax} = \mathbf{b}$ together with the corresponding residuals $\{\mathbf{r}_i\}$ (this would be the case if we had already attempted to solve $\mathbf{Ax} = \mathbf{b}$ using say BiCG or the Hegedüs Galerkin algorithm). The MRS algorithm would then compute two further sequences $\{\hat{\mathbf{x}}_i\}$ and $\{\hat{\mathbf{r}}_i\}$ by $\hat{\mathbf{x}}_1 = \mathbf{x}_1$, $\hat{\mathbf{r}}_1 = \mathbf{r}_1$ and, for $i \geq 1$,

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i (1 - \eta_i) + \mathbf{x}_{i+1} \eta_i \quad (6.42)$$

and, with obvious notation and since in this equation the coefficients of $\hat{\mathbf{x}}_i$ and \mathbf{x}_{i+1} sum to unity,

$$\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{r}}_i (1 - \eta_i) + \mathbf{r}_{i+1} \eta_i. \quad (6.43)$$

Comparison of equations (6.41) and (6.42) show that these are identical if $\eta_i = \zeta_{i+1}^2 / \rho_{i+1}^2$, but the natural *a priori* choice of η_i would be that value of η that minimises $\|\widehat{\mathbf{r}}_i(1 - \eta) + \mathbf{r}_{i+1}\eta\|$ over η . If we define \mathbf{u}_i by

$$\mathbf{u}_i = \widehat{\mathbf{r}}_i - \mathbf{r}_{i+1} \quad (6.44)$$

then

$$\widehat{\mathbf{r}}_i(1 - \eta) + \mathbf{r}_{i+1}\eta \equiv \widehat{\mathbf{r}}_i - \mathbf{u}_i\eta$$

and a straightforward calculation gives the minimising value η_i of η to be

$$\eta_i = \frac{\mathbf{u}_i^T \widehat{\mathbf{r}}_i}{\mathbf{u}_i^T \mathbf{u}_i} \quad (6.45)$$

so that

$$\widehat{\mathbf{r}}_{i+1} = \left(\mathbf{I} - \frac{\mathbf{u}_i \mathbf{u}_i^T}{\mathbf{u}_i^T \mathbf{u}_i} \right) \widehat{\mathbf{r}}_i. \quad (6.46)$$

or

$$\widehat{\mathbf{r}}_{i+1} = \widehat{\mathbf{r}}_i - \mathbf{u}_i\eta_i \quad (6.47)$$

Thus, from equation (6.46), $\widehat{\mathbf{r}}_{i+1}$ is obtained from $\widehat{\mathbf{r}}_i$ by an orthogonal projection, emphasising that $\|\widehat{\mathbf{r}}_{i+1}\| \leq \|\widehat{\mathbf{r}}_i\|$.

Now we also need to compute the sequence $\{\widehat{\mathbf{x}}_i\}$ of approximate solutions corresponding to $\{\widehat{\mathbf{r}}_i\}$ and if we define, analogously to equation (6.44), \mathbf{v}_i by

$$\mathbf{v}_i = \widehat{\mathbf{x}}_i - \mathbf{x}_{i+1} \quad (6.48)$$

then $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i$ and equation (6.47) gives

$$\widehat{\mathbf{x}}_{i+1} = \widehat{\mathbf{x}}_i - \mathbf{v}_i\eta_i. \quad (6.49)$$

We thus obtain the following algorithm.

Algorithm MRS

- (1) Given \mathbf{x}_i and \mathbf{r}_i , $i = 1, 2, \dots$, set $\widehat{\mathbf{x}}_1 = \mathbf{x}_1$, $\widehat{\mathbf{r}}_1 = \mathbf{r}_1$ and $i = 1$.
- (2) while there still remain values of \mathbf{x}_{i+1} and \mathbf{r}_{i+1}
 - (a) compute $\mathbf{u}_i = \widehat{\mathbf{r}}_i - \mathbf{r}_{i+1}$ and $\mathbf{v}_i = \widehat{\mathbf{x}}_i - \mathbf{x}_{i+1}$
 - (b) compute $\eta_i = \frac{\mathbf{u}_i^T \widehat{\mathbf{r}}_i}{\mathbf{u}_i^T \mathbf{u}_i}$
 - (c) compute $\widehat{\mathbf{x}}_{i+1} = \widehat{\mathbf{x}}_i - \mathbf{v}_i\eta_i$ and $\widehat{\mathbf{r}}_{i+1} = \widehat{\mathbf{r}}_i - \mathbf{u}_i\eta_i$
 - (d) set $i = i + 1$
- (3) endwhile
- (4) end MRS

Now this algorithm, like all other algorithms, relies on the accuracy of its data for its own accuracy, the data in this case being the sequences $\{\mathbf{x}_i\}$ and $\{\mathbf{r}_i\}$. In order

to save computational labour the residuals \mathbf{r}_i are normally not computed as $\mathbf{Ax}_i - \mathbf{b}$ but recursively, so that because of rounding error the recursively-computed residual \mathbf{r}_i is not normally the same as $\mathbf{Ax}_i - \mathbf{b}$. This discrepancy tends to increase dramatically if the residual norms fluctuate strongly in magnitude and has a damaging effect on MRS. This is particularly likely to occur if the original sequences were produced by BiCG or even more so by CGS (see Chapter 9 for further discussion) but unfortunately it is precisely such algorithms that most need the smoothing effect of MRS or an equivalent algorithm. As a result of this, the original MRS does not always compute compatible values of $\{\hat{\mathbf{x}}_i\}$ and $\{\hat{\mathbf{r}}_i\}$.

To overcome this problem, Zhou and Walker [261] propose an alternative version of MRS, MRS(Modified) say. They take as their data the sequence $\{\mathbf{x}_i\}$ but replace $\{\mathbf{r}_i\}$ by the sequence $\{\mathbf{Ap}_i\}$, where $\mathbf{p}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$, so that

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{Ap}_i. \quad (6.50)$$

The sequence $\{\mathbf{Ap}_i\}$ too is normally available from BiCG or similar algorithms so no extra matrix-vector multiplications are involved. In MRS(M) the sequences $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ are not computed using equations (6.44) and (6.48) but are generated recursively as follows.

From equations (6.44), (6.46) and (6.50) we obtain

$$\begin{aligned} \mathbf{u}_{i+1} &= \hat{\mathbf{r}}_i - \mathbf{u}_i \eta_i - \mathbf{r}_{i+1} - \mathbf{Ap}_{i+1} \\ &= \mathbf{u}_i (1 - \eta_i) - \mathbf{Ap}_{i+1} \end{aligned} \quad (6.51)$$

so that, from equation (6.48),

$$\mathbf{v}_{i+1} = \mathbf{v}_i (1 - \eta_i) - \mathbf{p}_{i+1}. \quad (6.52)$$

These last two equations enable us to restructure MRS into a form where the original (inaccurate) residuals do not appear and where the approximate solution \mathbf{x}_i is used only to compute the next approximation \mathbf{x}_{i+1} . Together with equations (6.46) and (6.49) they give the following algorithm:

Algorithm MRS(M)

- (1) Given \mathbf{p}_i and \mathbf{Ap}_i (or equivalently \mathbf{x}_i and \mathbf{Ap}_i), $i = 1, 2, \dots$, set $\hat{\mathbf{x}}_1 = \mathbf{x}_1$, $\hat{\mathbf{r}}_1 = \mathbf{r}_1$, $\mathbf{u}_1 = -\mathbf{Ap}_1$ and $\mathbf{v}_1 = -\mathbf{p}_1$. Set $i = 1$.
- (2) while there still remain values of \mathbf{p}_{i+1} and \mathbf{Ap}_{i+1}
 - (a) compute $\eta_i = \frac{\mathbf{u}_i^T \hat{\mathbf{r}}_i}{\mathbf{u}_i^T \mathbf{u}_i}$
 - (b) compute $\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - \mathbf{v}_i \eta_i$ and $\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{r}}_i - \mathbf{u}_i \eta_i$
 - (c) compute $\mathbf{u}_{i+1} = \mathbf{u}_i (1 - \eta_i) - \mathbf{Ap}_{i+1}$
 - (d) compute $\mathbf{v}_{i+1} = \mathbf{v}_i (1 - \eta_i) - \mathbf{p}_{i+1}$
 - (e) set $i = i + 1$
- (3) endwhile
- (4) end MRS(M).

6.6. Discussion

Although the algorithms obtained by applying the QMR technique to existing algorithms, e.g. QMRCGStab and TFQMR, are tolerably well known the MRS versions are considerably less so, despite their apparently superior monotone properties. In particular, at the time of writing, there seem to have been very few direct comparisons between the two techniques so it is not possible to say with any degree of certainty which of the two yields the better results. If, though, the convergence of the Krylov methods is dominated by the properties of the matrix \mathbf{KG} then it is likely that there will be very little difference in actual performance between the two of them so that the choice may well depend on simplicity of implementation. However in the broad context of the properties of the Krylov algorithms, this is a matter of very little importance. A more serious question is whether or not this type of modification is needed at all. To put it bluntly, if two algorithms achieve similar results with a similar amount of work, does it matter if the convergence of one is smooth and that of the other is erratic?

The answer to this question is probably “no” provided that the requirements of similar work and similar accuracy are maintained. Although the smooth algorithm will undoubtedly have a better “feelgood factor” this will hardly concern a user for whom the algorithm is a black box and whose only interest is getting quick and accurate answers. Only if the algorithm fails to deliver in one of these areas will he be troubled, but therein lies the rub. Because the residuals \mathbf{r}_i are computed recursively (and not directly by $\mathbf{r}_i = \mathbf{Ax}_i - \mathbf{b}$ in order to avoid an extra matrix-vector multiplication at each step) they may be quite different from (and much smaller than) the true ones. It is thus possible for such an algorithm to converge to an inaccurate solution and for this reason we believe that, in practice, it is imperative to perform a direct calculation of the final residual before accepting the associated \mathbf{x}_i as the approximate solution (see Chapter 10, below).

Now both theory and experiment show that the differences between the true and computed residuals are more pronounced if the convergence is erratic (see Chapter 9 below). This would imply that smooth convergence is desirable but if it is not inherent in the algorithm it can be achieved in one of two ways. In the first the actual structure of the algorithm is modified to give a smoother sequence. This is what is done in BiCGStab and for this type of algorithm the argument is probably valid. For TFQMR and the MRS algorithms, on the other hand, although there may be minor changes made to the fundamental algorithm (see page 113 for those in the case of TFQMR) the smooth sequence is computed *in addition to* and not *instead of* the original sequence which is computed in essentially the same way as it is in the unsmoothed version of the algorithm. Unless therefore the smoothed sequence converges to a different value to that attained by the original algorithm (which is unlikely) the effect of the smoothing will not be to produce a more accurate solution than the original. It will merely induce a false sense of security in the mind of the observer. We should also note that residual smoothing does not seem to *accelerate* convergence, only to smooth it. In the two examples quoted in [116] the algorithm TFQMR (which is essentially the QMR technique tacked on to CGS)

computes residuals that seem to act as a smooth lower bound for the far more bouncy CGS ones, but the former do not reduce further until dragged down by the next favourable CGS iteration. In other words, at any stage of the iteration the smallest value so far achieved by TFQMR is no smaller than that achieved by CGS. If this behaviour is true in general then any convergence criterion in the form of $\|\mathbf{r}\| \leq \epsilon$ for some tolerance ϵ is just as likely to be satisfied after s iterations by CGS as it is by TFQMR. However, for the two examples cited in [116], CGS with or without QMR did outperform BiCGStab but to what extent this behaviour is typical is not known. However the basic version of all these algorithms can break down. It is desirable therefore to have alternative procedures available that can be called upon in the event of imminent failure and the commonest of these, look ahead, is described in the next chapter.

Chapter 7

Look-ahead methods

As we have already mentioned, under certain circumstances (see Causes of breakdown, page 24) both GODir and GOMin break down because $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$, whose inverse is required by the algorithm, is either singular or too badly conditioned for accurate inversion. It is possible in certain circumstances to overcome this problem by taking the appropriate remedial action, the nature of this remedial action being dictated by the precise cause of the singularity or near-singularity. If it is due to the columns of \mathbf{P}_i becoming linearly dependent or nearly so then we have one type of instability which is dealt with in the next chapter. If, on the other hand, \mathbf{P}_i has full rank and the singularity of $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is due solely to the indefiniteness of \mathbf{G} , then we have *serious breakdown*, and a different course of action is required. This action is referred to as “look-ahead” and it is this approach that we now describe. We assume in this chapter, therefore, that \mathbf{P}_i always has full rank except on termination.

It is convenient to use another of our formal algorithms to give an overview of the look-ahead technique but before doing so we first, for purposes of comparison, give a formal definition of the basic algorithm, i.e. without look-ahead. This basic algorithm is just GODir (or GOMin) with Step 2(e) replaced by

$$\text{compute } \mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{W}_{i+1}$$

where \mathbf{W}_{i+1} is either \mathbf{KGP}_i or \mathbf{KF}_{i+1} as appropriate, and is as follows:

The basic algorithm

- (1) Select \mathbf{G} , \mathbf{H} , \mathbf{K} and \mathbf{X}_1 . Set $\mathbf{Q}_0 = \mathbf{I}$ and $i = 1$. Compute $\mathbf{F}_1 = \mathbf{GX}_1 - \mathbf{H}$ and $\mathbf{P}_1 = \mathbf{KF}_1$
- (2) while $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is nonsingular
 - (a) compute $\mathbf{D}_i = \mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$
 - (b) compute $\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i$
 - (c) compute $\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i$
 - (d) compute $\mathbf{Q}_i = \mathbf{Q}_{i-1} - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T$
 - (e) compute $\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{W}_{i+1}$ for the appropriate \mathbf{W}_{i+1}

- (f) set $i = i + 1$
- (3) endwhile
- (4) end the basic algorithm.

This algorithm terminates when $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ becomes singular, something which must inevitably occur sooner or later (see Chapter 2). When this happens prematurely it is not possible to compute the new projection matrix \mathbf{Q}_i , but there is no reason why the matrices \mathbf{P}_{i+j} , $j = 1, 2, \dots$, should not be computed using the *old* oblique projector \mathbf{Q}_{i-1} instead of the updated one. This at least ensures that the matrices \mathbf{P}_{i+j} , $j = 0, 1, \dots$, are conjugate to the matrices \mathbf{P}_k , $k = 1, 2, \dots, i - 1$, even though they will not be conjugate to each other. However this process cannot be allowed to continue indefinitely since the algorithm functions by constructing a sequence of conjugate matrices, the linear-algebraic costs of not doing so being prohibitive. The obvious way forward is to keep adjoining the matrices \mathbf{P}_{i+j} to the matrix $\tilde{\mathbf{P}}_i$ just as in the basic algorithm in the hope that, sooner or later, \mathbf{Q}_i as defined by equation (1.46) will become computable. If it does then it may be calculated and the algorithm can proceed happily on its way. If it does not then we have the problem of *incurable breakdown* for which, as the name implies, there is no known remedy apart from restarting from a different initial approximate solution. The formal representation of the look-ahead version is

The look-ahead version

- (1) Select \mathbf{G} , \mathbf{H} , \mathbf{K} and \mathbf{X}_1 . Set $\tilde{\mathbf{Q}}_0 = \mathbf{I}$, $i = 1$ and $k = 1$. Compute $\mathbf{F}_1 = \mathbf{G}\mathbf{X}_1 - \mathbf{H}$ and $\mathbf{P}_1 = \mathbf{K}\mathbf{F}_1$
- (2) set $\tilde{\mathbf{P}}_k = \mathbf{P}_i$
- (3) while $\tilde{\mathbf{P}}_k^T \mathbf{G} \tilde{\mathbf{P}}_k$ is singular
 - (a) if $\tilde{\mathbf{P}}_k$ is rank-deficient, stop, else
 - (b) compute $\mathbf{P}_{i+1} = \tilde{\mathbf{Q}}_{k-1}^T \mathbf{K} \mathbf{G} \mathbf{P}_i$
 - (c) set $\tilde{\mathbf{P}}_k = \begin{bmatrix} \tilde{\mathbf{P}}_k & \mathbf{P}_{i+1} \end{bmatrix}$
 - (d) set $i = i + 1$
- (4) endwhile
- (5) compute $\tilde{\mathbf{D}}_k = \tilde{\mathbf{P}}_k^T \mathbf{G} \tilde{\mathbf{P}}_k$
- (6) compute $\mathbf{X}_{k+1} = \mathbf{X}_k - \tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1} \tilde{\mathbf{P}}_k^T \mathbf{F}_k$
- (7) compute $\mathbf{F}_{k+1} = \mathbf{F}_k - \mathbf{G} \tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1} \tilde{\mathbf{P}}_k^T \mathbf{F}_k$
- (8) compute $\tilde{\mathbf{Q}}_k = \tilde{\mathbf{Q}}_{k-1} - \mathbf{G} \tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1} \tilde{\mathbf{P}}_k^T$
- (9) compute $\mathbf{P}_{i+1} = \tilde{\mathbf{Q}}_k^T \mathbf{W}_{i+1}$ for the appropriate \mathbf{W}_{i+1}
- (10) set $i = i + 1$, $k = k + 1$
- (11) return to Step 2
- (12) end the look-ahead version.

Remark 2. This algorithm terminates when the columns of $\tilde{\mathbf{P}}_k$ become linearly dependent, as they must eventually do. If the rank of $\tilde{\mathbf{P}}_k$ is zero then the solution has been computed. If it is equal to that of $\tilde{\mathbf{D}}_k$ the retrieval of a partial solution

should be possible. If it exceeds that of $\tilde{\mathbf{D}}_k$ then incurable breakdown has occurred. See Chapter 2, and Chapter 8 for further details of obtaining a partial solution. Note also that in Step 3(b) it is not possible to replace \mathbf{KGP}_i by \mathbf{KF}_{k+1} since \mathbf{F}_{k+1} can only be calculated after the while-loop has terminated.

It follows from the above description of the look-ahead version that if $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is nonsingular for all i the while-loop is never entered and the algorithm reduces to the basic algorithm. If, on the other hand, $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is singular for some i then the while-loop will be activated and the resulting matrix computed, $\tilde{\mathbf{P}}_k$, will have dimensions determined by the number of times the while-loop is traversed. Suppose that the loop is traversed $t_k - 1$ times. Then $\tilde{\mathbf{P}}_k$ will consist of t_k blocks each consisting of one of the matrices \mathbf{P}_i for appropriate values of i . In fact, if we define $s_0 = 0$ and s_k , $k = 1, 2, \dots$, by

$$s_k = \sum_{j=1}^k t_j \quad (7.1)$$

it is readily seen that

$$\tilde{\mathbf{P}}_k = [\mathbf{P}_{s_{k-1}+1} \ \mathbf{P}_{s_{k-1}+2} \ \dots \ \mathbf{P}_{s_k}]. \quad (7.2)$$

It follows from this and equation (1.31) that an alternative expression for $\bar{\mathbf{P}}_{s_k}$ when look-ahead is used is

$$\bar{\mathbf{P}}_{s_k} = [\tilde{\mathbf{P}}_1 \ \tilde{\mathbf{P}}_2 \ \dots \ \tilde{\mathbf{P}}_k]. \quad (7.3)$$

Now a glance at Step 3 and Step 9 of the algorithm indicates that $\tilde{\mathbf{P}}_k = \tilde{\mathbf{Q}}_{k-1}^T \tilde{\mathbf{W}}_k$ for some matrix $\tilde{\mathbf{W}}_k$ while from Step 8,

$$\tilde{\mathbf{Q}}_k = \mathbf{I} - \sum_{j=1}^k \mathbf{G} \tilde{\mathbf{P}}_j \tilde{\mathbf{D}}_j^{-1} \tilde{\mathbf{P}}_j^T. \quad (7.4)$$

Comparison of these two equations with Steps 2a and 2b of the generalised Gram-Schmidt algorithm (see page 23) indicates that the look-ahead versions of both GODir and GOMin are simply more sophisticated examples of this algorithm. They thus generate sequences of conjugate matrices $\{\tilde{\mathbf{P}}_i\}$ and their projectors, by analogy with equations (1.49) and (1.50), satisfy

$$\tilde{\mathbf{Q}}_k^T \bar{\mathbf{P}}_{s_k} = \mathbf{O} \quad (7.5)$$

and

$$\tilde{\mathbf{Q}}_i \tilde{\mathbf{Q}}_j = \tilde{\mathbf{Q}}_j \tilde{\mathbf{Q}}_i = \tilde{\mathbf{Q}}_i, \quad j \leq i. \quad (7.6)$$

The matrices \mathbf{F}_{k+1} and \mathbf{X}_{k+1} are computed by Steps 6 and 7 of the algorithm only when a new nonsingular $\tilde{\mathbf{D}}_k$ has been computed, and are given by

$$\mathbf{X}_{k+1} = \mathbf{X}_k - \tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1} \tilde{\mathbf{P}}_k^T \mathbf{F}_k$$

and

$$\mathbf{F}_{k+1} = \mathbf{F}_k - \mathbf{G}\tilde{\mathbf{P}}_k\tilde{\mathbf{D}}_k^{-1}\tilde{\mathbf{P}}_k^T\mathbf{F}_k, \quad (7.7)$$

or

$$\mathbf{F}_{k+1} = \left(\mathbf{I} - \mathbf{G}\tilde{\mathbf{P}}_k\tilde{\mathbf{D}}_k^{-1}\tilde{\mathbf{P}}_k^T \right) \mathbf{F}_k.$$

Thus

$$\mathbf{F}_{k+1} = \prod_{j=1}^k \left(\mathbf{I} - \mathbf{G}\tilde{\mathbf{P}}_j\tilde{\mathbf{D}}_j^{-1}\tilde{\mathbf{P}}_j^T \right) \mathbf{F}_1$$

or, from the conjugacy of the matrices $\tilde{\mathbf{P}}_j$,

$$\mathbf{F}_{k+1} = \tilde{\mathbf{Q}}_k \mathbf{F}_1 \quad (7.8)$$

where $\tilde{\mathbf{Q}}_k$ is defined by equation (7.4) so that

$$\tilde{\mathbf{P}}_j^T \mathbf{F}_{k+1} = \mathbf{O}, \quad 1 \leq j \leq k. \quad (7.9)$$

Equation (7.8) is essentially the same as equation (1.43) except that the rank of $\tilde{\mathbf{Q}}_k$ is less than that of \mathbf{Q}_i if any look-ahead steps have been taken. Finally, it is convenient at this point to define $\tilde{\mathbf{T}}_k$ by

$$\tilde{\mathbf{T}}_k = \tilde{\mathbf{P}}_k\tilde{\mathbf{D}}_k^{-1} = [\mathbf{T}_{s_{k-1}+1} \ \mathbf{T}_{s_{k-1}+2} \ \dots \ \mathbf{T}_{s_k}] \quad (7.10)$$

where the partitioning of $\tilde{\mathbf{T}}_k$ echoes of that of $\tilde{\mathbf{P}}_k$ in equation (7.2), and to express $\tilde{\mathbf{Q}}_k$ in terms of $\tilde{\mathbf{T}}_k$. With $\tilde{\mathbf{T}}_k$ thus defined equation (7.7) may be written

$$\mathbf{F}_{k+1} = \mathbf{F}_k - \mathbf{G} \left(\sum_{j=s_{k-1}+1}^{s_k} \mathbf{P}_j \mathbf{T}_j^T \right) \mathbf{F}_k \quad (7.11)$$

(note: since $\tilde{\mathbf{D}}_k$ is a full matrix rather than a block-diagonal one there is no simple relationship between the matrices \mathbf{P}_j and \mathbf{T}_j). The projector $\tilde{\mathbf{Q}}_k$ becomes, since $\tilde{\mathbf{D}}_k$ is symmetric,

$$\tilde{\mathbf{Q}}_k = \mathbf{I} - \mathbf{G} \sum_{j=1}^k \tilde{\mathbf{P}}_j \tilde{\mathbf{T}}_j^T = \mathbf{I} - \mathbf{G} \sum_{j=1}^{s_k} \mathbf{P}_j \mathbf{T}_j^T. \quad (7.12)$$

7.0.1. Note on notation

From equations (7.2) - (7.4) and the fact that $\tilde{\mathbf{P}}_j^T \mathbf{G} \tilde{\mathbf{P}}_k = \mathbf{O}$, $j \neq k$, it follows that

$$\tilde{\mathbf{Q}}_k = \mathbf{I} - \mathbf{G} \bar{\mathbf{P}}_{s_k} \left(\bar{\mathbf{P}}_{s_k}^T \mathbf{G} \bar{\mathbf{P}}_{s_k} \right)^{-1} \bar{\mathbf{P}}_{s_k}^T$$

and so might be designated \mathbf{Q}_{s_k} . However in this context \mathbf{Q}_j only exists for $j = s_i$, $i = 1, 2, \dots, k$, and even for these values of j is not in general given by equation (1.45). For these reasons the designation $\tilde{\mathbf{Q}}_k$ is preferred.

7.1. The computational versions

In just the same way that, if \mathbf{G} and \mathbf{K} are both symmetric, GODir and GOMin reduce to the block Lanczos and block HS algorithms respectively, similar simplifications may be made to the look-ahead versions of the basic algorithm in the case when these two matrices are symmetric. Since the matrices \mathbf{P}_j are mutually conjugate it follows from equation (7.4) that

$$\tilde{\mathbf{Q}}_{i-1} \mathbf{G} \tilde{\mathbf{P}}_{k+1} = \mathbf{G} \tilde{\mathbf{P}}_{k+1}, \quad 1 \leq i \leq k+1. \quad (7.13)$$

Now for $s_{i-1} + 1 \leq j \leq s_i$ the matrices \mathbf{P}_j are computed by

$$\mathbf{P}_j = \tilde{\mathbf{Q}}_{i-1}^T \mathbf{W}_j \quad (7.14)$$

so that premultiplying equation (7.13) by \mathbf{W}_j^T gives

$$\mathbf{P}_j^T \mathbf{G} \tilde{\mathbf{P}}_{k+1} = \mathbf{W}_j^T \mathbf{G} \tilde{\mathbf{P}}_{k+1}.$$

This equation holds for both $s_{i-1} + 1 \leq j \leq s_i$ and $i = 1, 2, \dots, k+1$, and since $s_0 = 0$ by definition (see text immediately preceding equation (7.1)) it follows that

$$\mathbf{P}_j^T \mathbf{G} \tilde{\mathbf{P}}_{k+1} = \mathbf{W}_j^T \mathbf{G} \tilde{\mathbf{P}}_{k+1}, \quad 1 \leq j \leq s_{k+1}. \quad (7.15)$$

Hence, since $\tilde{\mathbf{P}}_i^T \mathbf{G} \tilde{\mathbf{P}}_{k+1} = \mathbf{O}$ for $1 \leq i \leq k$,

$$\mathbf{W}_j^T \mathbf{G} \tilde{\mathbf{P}}_{k+1} = \mathbf{O}, \quad 1 \leq j \leq s_k. \quad (7.16)$$

Similarly, by analogy with equation (7.13), equations (7.4) and (7.9) give

$$\tilde{\mathbf{Q}}_{i-1} \mathbf{F}_{k+1} = \mathbf{F}_{k+1}, \quad 1 \leq i \leq k+1. \quad (7.17)$$

Premultiplying equation (7.17) by \mathbf{W}_j^T then yields, from equation (7.14),

$$\mathbf{P}_j^T \mathbf{F}_{k+1} = \mathbf{W}_j^T \mathbf{F}_{k+1}, \quad s_{i-1} + 1 \leq j \leq s_i.$$

A parallel argument to that used to establish equation (7.15) then gives

$$\mathbf{P}_j^T \mathbf{F}_{k+1} = \mathbf{W}_j^T \mathbf{F}_{k+1}, \quad 1 \leq j \leq s_{k+1}$$

so that, from equations (7.2) and (7.9),

$$\mathbf{W}_j^T \mathbf{F}_{k+1} = \mathbf{O}, \quad 1 \leq j \leq s_k. \quad (7.18)$$

Equations (7.16) and (7.18) permeate the whole of conjugate gradient theory. They were used implicitly in Chapter 3 to derive the short recurrences and will

be called upon to perform the same office again in Chapter 8. It is the properties of these equations that, if the matrices \mathbf{W}_k are chosen appropriately, enable us to obtain short recurrences when computing the matrix sequences $\{\mathbf{P}_i\}$ for the look-ahead algorithms.

Just as in the case without look-ahead we have essentially two choices for \mathbf{W}_k leading again to the Lanczos and to the HS versions of the algorithms, and as before the Lanczos choice leads to a slightly more complicated algorithm but by a simpler route. The analysis for the look-ahead methods is inherently more complicated than that for the standard ones as not only do we have to consider the existence of the while-loop (Step 3 of the algorithm) but also the fact that this loop is not executed a fixed number of times. As a result of this the notation required for the look-ahead methods is significantly more complicated than that for the standard ones. Moreover, for the HS versions of the algorithms, we are forced to use two different values of \mathbf{W}_k since the one that essentially defines the HS method is not available until after the while-loop is terminated.

The approach here will be similar to that of Chapter 3. We first obtain look-ahead variants of both main versions, HS and Lanczos, of the block-CG algorithm in terms of \mathbf{G} and \mathbf{K} . We then substitute in the equations so obtained the particular values of these two matrices appropriate to the specific algorithms.

7.1.1. The look-ahead block Lanczos algorithm

For this version we choose \mathbf{W}_j by

$$\mathbf{W}_j = \mathbf{K}\mathbf{G}\mathbf{P}_{j-1} \quad (7.19)$$

for all values of j so equation (7.16) becomes, on replacing j by $j + 1$ for clarity,

$$\mathbf{P}_j^T \mathbf{G} \mathbf{K} \mathbf{G} \tilde{\mathbf{P}}_{k+1} = \mathbf{O}, \quad 1 \leq j \leq s_k - 1. \quad (7.20)$$

Suppose now that the matrices $\tilde{\mathbf{P}}_j$, $j = 1, 2, \dots, k$, have already been computed, and consider the calculation of $\tilde{\mathbf{P}}_{k+1}$. Equations (7.14) and (7.19) yield

$$\mathbf{P}_i = \tilde{\mathbf{Q}}_k^T \mathbf{K} \mathbf{G} \mathbf{P}_{i-1}, \quad s_k + 1 \leq i \leq s_{k+1}. \quad (7.21)$$

Putting $i = s_k + 1$ then gives, from equation (7.12),

$$\mathbf{P}_{s_k+1} = \left(\mathbf{I} - \sum_{j=1}^{s_k} \mathbf{T}_j \mathbf{P}_j^T \mathbf{G} \right) \mathbf{K} \mathbf{G} \mathbf{P}_{s_k} \quad (7.22)$$

or, from equation (7.20) and since \mathbf{P}_{s_k} is a block column of $\tilde{\mathbf{P}}_k$,

$$\mathbf{P}_{s_k+1} = \left(\mathbf{I} - \sum_{j=s_{k-1}}^{s_k} \mathbf{T}_j \mathbf{P}_j^T \mathbf{G} \right) \mathbf{K} \mathbf{G} \mathbf{P}_{s_k}. \quad (7.23)$$

An appeal to equation (7.10) then yields

$$\mathbf{P}_{s_k+1} = \left(\mathbf{I} - \mathbf{T}_{s_{k-1}} \mathbf{P}_{s_{k-1}}^T \mathbf{G} - \tilde{\mathbf{T}}_k \tilde{\mathbf{P}}_k^T \mathbf{G} \right) \mathbf{KGP}_{s_k}. \quad (7.24)$$

For $s_k + 2 \leq i \leq s_{k+1}$ equation (7.21) still holds but the matrices \mathbf{P}_{i-1} are now block-columns of $\tilde{\mathbf{P}}_{k+1}$. This equation and equation (7.12) give

$$\mathbf{P}_i = \left(\mathbf{I} - \sum_{j=1}^{s_k} \mathbf{T}_j \mathbf{P}_j^T \mathbf{G} \right) \mathbf{KGP}_{i-1}$$

and this, from equation (7.20), becomes

$$\mathbf{P}_i = (\mathbf{I} - \mathbf{T}_{s_k} \mathbf{P}_{s_k}^T \mathbf{G}) \mathbf{KGP}_{i-1}. \quad (7.25)$$

Note that, if the while-loop (Step 3) of the look-ahead version is never invoked, equation (7.24) reverts to the standard block Lanczos algorithm.

7.1.2. The Hestenes-Stiefel version

In this version, based on GOMin, the matrices \mathbf{W}_i are chosen by

$$\mathbf{W}_i = \mathbf{KF}_i \quad (7.26)$$

in the absence of breakdown but if, for some i , $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ is singular then it is not possible to compute \mathbf{F}_{i+1} and we thus cannot choose \mathbf{W}_{i+1} using this equation. We can, though, make the Lanczos choice, i.e. $\mathbf{W}_{i+1} = \mathbf{KGP}_i$, so that if we want to apply look-ahead to the HS version we arrive at a hybrid algorithm, owing something to both the Lanczos and the Hestenes-Stiefel ideas.

The derivation of the computational form of the look-ahead HS version of the algorithm is, notwithstanding the greater simplicity of the final result, considerably more complicated than that of the Lanczos form. An inspection of the formal look-ahead version of the algorithm shows that, just as in the Lanczos form, it generates sequences of matrices $\{\tilde{\mathbf{P}}_i\}$ and $\{\tilde{\mathbf{Q}}_i\}$, defined by equations (7.2) and (7.4), where the matrices $\tilde{\mathbf{P}}_i$ are mutually conjugate with respect to \mathbf{G} . If the integers s_i are as previously defined it follows from the algorithm and equation (7.2) that the block columns of $\tilde{\mathbf{P}}_i$ are computed by $\mathbf{P}_j = \tilde{\mathbf{Q}}_{i-1}^T \mathbf{W}_j$ where

$$\mathbf{W}_j = \mathbf{KF}_i \quad (7.27)$$

for $j = s_{i-1} + 1$ and

$$\mathbf{W}_j = \mathbf{KGP}_{j-1}. \quad (7.28)$$

for $s_{i-1} + 2 \leq j \leq s_i$, where $i = 1, 2, \dots, k$. It now follows from these two equations and from equations (7.18) and (7.27) that

$$\mathbf{F}_i^T \mathbf{KF}_{k+1} = \mathbf{O} \quad (7.29)$$

for $i = 1, 2, \dots, k$, and from equations (7.18) and (7.28) that

$$\mathbf{P}_j^T \mathbf{GKF}_{k+1} = \mathbf{O} \quad (7.30)$$

for all values of j between 1 and $s_k - 1$ inclusive except s_1, s_2, \dots, s_{k-1} . We shall now show that equation (7.30) holds even for these values of j if certain conditions are satisfied.

Substituting i for k in equation (7.11) and transposing gives

$$\mathbf{F}_{i+1}^T - \mathbf{F}_i^T = -\mathbf{F}_i^T \left(\sum_{j=s_{i-1}+1}^{s_i} \mathbf{T}_j \mathbf{P}_j^T \right) \mathbf{G}$$

which on postmultiplying by \mathbf{KF}_{k+1} gives, from equation (7.29),

$$\mathbf{F}_i^T \left(\sum_{j=s_{i-1}+1}^{s_i} \mathbf{T}_j \mathbf{P}_j^T \right) \mathbf{GKF}_{k+1} = \mathbf{O}, \quad 1 \leq i \leq k-1.$$

Hence, from equation (7.30),

$$\mathbf{F}_i^T \mathbf{T}_{s_i} \mathbf{P}_{s_i}^T \mathbf{GKF}_{k+1} = \mathbf{O}, \quad 1 \leq i \leq k-1. \quad (7.31)$$

If we now make the assumption (see below) that $\mathbf{F}_i^T \mathbf{T}_{s_i}$ is nonsingular for $1 \leq i \leq k-1$ this leads immediately to the conclusion that

$$\mathbf{P}_{s_i}^T \mathbf{GKF}_{k+1} = \mathbf{O}, \quad 1 \leq i \leq k-1,$$

and combining this equation with equation (7.30) gives

$$\mathbf{P}_j^T \mathbf{GKF}_{k+1} = \mathbf{O}, \quad 1 \leq j \leq s_k - 1. \quad (7.32)$$

This is one of the key results that enable us to obtain a simple computational implementation of the look-ahead version of the block HS algorithm. The other key result is equation (7.20), which is proved for the HS methods by noting that equation (7.16) is simply equation (7.18) with \mathbf{GP}_{k+1} substituted for \mathbf{F}_{k+1} . Making the same substitution in equations (7.29) - (7.32) and following the same argument used to derive equation (7.32) gives the desired result. With these two equations established we can now obtain the computational form of the HS version.

Suppose then that the matrices $\tilde{\mathbf{P}}_i$, $i = 1, 2, \dots, k$, together with $\tilde{\mathbf{Q}}_k$ and \mathbf{F}_{k+1} have already been computed. The first (and possibly only) block column of $\tilde{\mathbf{P}}_{k+1}$ is then given by

$$\mathbf{P}_{s_k+1} = \tilde{\mathbf{Q}}_k^T \mathbf{KF}_{k+1}. \quad (7.33)$$

or, from equation (7.12),

$$\mathbf{P}_{s_k+1} = \left(\mathbf{I} - \sum_{j=1}^{s_k} \mathbf{T}_j \mathbf{P}_j^T \mathbf{G} \right) \mathbf{KF}_{k+1}.$$

It then follows immediately from equation (7.32) that

$$\mathbf{P}_{s_k+1} = (\mathbf{I} - \mathbf{T}_{s_k} \mathbf{P}_{s_k}^T \mathbf{G}) \mathbf{K} \mathbf{F}_{k+1}. \quad (7.34)$$

The remaining blocks of $\tilde{\mathbf{P}}_{k+1}$, if any, are computed by

$$\mathbf{P}_i = \tilde{\mathbf{Q}}_k^T \mathbf{K} \mathbf{G} \mathbf{P}_{i-1}, \quad s_k + 2 \leq i \leq s_{k+1},$$

where the matrices \mathbf{P}_{i-1} are themselves submatrices of $\tilde{\mathbf{P}}_{k+1}$. Equations (7.12) and (7.20) then give

$$\mathbf{P}_i = (\mathbf{I} - \mathbf{T}_{s_k} \mathbf{P}_{s_k}^T \mathbf{G}) \mathbf{K} \mathbf{G} \mathbf{P}_{i-1}, \quad s_k + 2 \leq i \leq s_{k+1}, \quad (7.35)$$

and this equation, together with equation (7.34), defines the final computational form of the algorithm. Note that unlike the Lanczos case, the projection matrices of equations (7.34) and (7.35) are identical.

We note that in order to obtain these two equations it is necessary to assume that $\mathbf{F}_k^T \mathbf{T}_{s_k}$ is nonsingular. For the basic version (no look-ahead) a similar assumption is necessary in order to derive the corresponding formula but in that case we have been able to relate the nonsingularity of $\mathbf{P}_k^T \mathbf{F}_k$ to the original Krylov sequence (see Theorem 19 above). Although we conjecture that a similar result holds in the look-ahead case we know of no comparable theorem to Theorem 19. To this extent our analysis of look-ahead methods is incomplete.

The above analysis completes the derivation of the basic computational forms of the two look-ahead algorithms but before we go on to examine particular examples we prove the following theorem.

Theorem 27. Let \mathbf{G} be symmetric and nonsingular and \mathbf{K} be symmetric positive definite. Assume that $\tilde{\mathbf{P}}_k$, \mathbf{X}_{k+1} and \mathbf{F}_{k+1} , $k = 1, 2, \dots, i$, have been successfully computed by one or other version of the look-ahead algorithm and let the first block column of $\tilde{\mathbf{P}}_{i+1}$, \mathbf{P}_{s_i+1} say, be computed either by

$$\mathbf{P}_{s_i+1} = \tilde{\mathbf{Q}}_i^T \mathbf{K} \mathbf{G} \mathbf{P}_{s_i}$$

(Lanczos) or

$$\mathbf{P}_{s_i+1} = \tilde{\mathbf{Q}}_i^T \mathbf{K} \mathbf{F}_{i+1}$$

(HS). Then if \mathbf{P}_{s_i+1} has full rank and $\mathbf{P}_{s_i+1}^T \mathbf{G} \mathbf{P}_{s_i+1} = \mathbf{O}$ (serious breakdown), only one look-ahead step will be necessary to restore the algorithm to normality.

Proof. Since \mathbf{P}_{s_i+1} is a submatrix of $\tilde{\mathbf{P}}_{i+1}$ it follows from conjugacy and equation (7.4) that

$$\tilde{\mathbf{Q}}_i \mathbf{G} \mathbf{P}_{s_i+1} = \mathbf{G} \mathbf{P}_{s_i+1}. \quad (7.36)$$

Now \mathbf{P}_{s_i+2} is computed by $\mathbf{P}_{s_i+2} = \tilde{\mathbf{Q}}_i^T \mathbf{K} \mathbf{G} \mathbf{P}_{s_i+1}$ so that

$$\tilde{\mathbf{D}}_{i+1} = \begin{bmatrix} \mathbf{P}_{s_i+1}^T \\ \mathbf{P}_{s_i+2}^T \end{bmatrix} \mathbf{G} [\mathbf{P}_{s_i+1} \ \mathbf{P}_{s_i+2}]$$

or

$$\tilde{\mathbf{D}}_{i+1} = \begin{bmatrix} \mathbf{P}_{s_i+1}^T \mathbf{G} \mathbf{P}_{s_i+1} & \mathbf{P}_{s_i+1}^T \mathbf{G} \tilde{\mathbf{Q}}_i^T \mathbf{K} \mathbf{G} \mathbf{P}_{s_i+1} \\ \mathbf{P}_{s_i+1}^T \mathbf{G} \mathbf{K} \tilde{\mathbf{Q}}_i \mathbf{G} \mathbf{P}_{s_i+1} & \mathbf{P}_{s_i+2}^T \mathbf{G} \mathbf{P}_{s_i+2} \end{bmatrix}, \quad (7.37)$$

and it follows from hypothesis and equation (7.36) that

$$\tilde{\mathbf{D}}_{i+1} = \begin{bmatrix} \mathbf{O} & \mathbf{B} \\ \mathbf{B} & \mathbf{C} \end{bmatrix}$$

where $\mathbf{B} = \mathbf{P}_{s_i+1}^T \mathbf{G} \mathbf{K} \mathbf{G} \mathbf{P}_{s_i+1}$. From the hypotheses of the theorem, \mathbf{B} is nonsingular and hence $\tilde{\mathbf{D}}_{i+1}$ is nonsingular so that \mathbf{X}_{i+2} , \mathbf{F}_{i+2} and $\tilde{\mathbf{Q}}_{i+1}$ may be duly computed.

■ This theorem underlines yet again the crucial importance of the properties of the matrices \mathbf{G} and \mathbf{K} to the functioning of conjugate-gradient algorithms and indicates why it is desirable (at least in theory) for at least one of these two matrices to be positive definite. It may be thought that the requirement that $\mathbf{P}_{s_i+1}^T \mathbf{G} \mathbf{P}_{s_i+1} = \mathbf{O}$ (rather than being merely singular) is a weakness of the theorem but in most practical cases, as we shall see, $\mathbf{P}_{s_i+1}^T \mathbf{G} \mathbf{P}_{s_i+1}$ is either itself a scalar or has the form $\begin{bmatrix} 0 & \alpha \\ \alpha & 0 \end{bmatrix}$ for some scalar α . It is thus either nonsingular (in which case there is no breakdown) or null (when the theorem applies).

7.2. Particular algorithms

We turn now to the look-ahead versions of some common algorithms, which we obtain from equations (7.24) etc. by making the appropriate substitutions for \mathbf{G} and \mathbf{K} (see Chapter 3 for the precise values of these two matrices).

We begin with the basic CG algorithm generating vector sequences and applied to a symmetric but indefinite system $\mathbf{A}\mathbf{x} = \mathbf{b}$. Thus $\mathbf{G} = \mathbf{A}$ and is indefinite and $\mathbf{p}_j^T \mathbf{G} \mathbf{p}_j$ is either zero or nonzero. \mathbf{K} is the identity matrix and is thus positive definite so that, from Theorem 27, at most only one step of look-ahead is ever needed.

If \mathbf{t}_{s_k} is the last column (first or second as appropriate) of $\tilde{\mathbf{T}}_k$ where $\tilde{\mathbf{T}}_k = \tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1}$, equations (7.34) and (7.35) give

$$\mathbf{p}_{s_k+1} = (\mathbf{I} - \mathbf{t}_{s_k} \mathbf{p}_{s_k}^T \mathbf{A}) \mathbf{r}_{k+1} \quad (7.38)$$

and, if \mathbf{p}_{s_k+2} needs to be computed,

$$\mathbf{p}_{s_k+2} = \mathbf{p}_{s_k+1} = (\mathbf{I} - \mathbf{t}_{s_k} \mathbf{p}_{s_k}^T \mathbf{A}) \mathbf{A} \mathbf{p}_{s_k+1}. \quad (7.39)$$

A method similar to this for overcoming serious breakdown was first proposed by Luenberger [180] who called it the *method of hyperbolic pairs*. The method given above is equivalent to the improved version of Luenberger's algorithm given by Fletcher [110].

Now for all the other look-ahead versions outlined here (BiCG, MRZ, the Hegedüs-Galerkin algorithm and QMR), \mathbf{P}_j has the form of equation (3.26) and \mathbf{G} is given either by

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \mathbf{O} & \mathbf{I} \\ \mathbf{I} & \mathbf{O} \end{bmatrix}.$$

This implies two things. The first is that $\mathbf{P}_j^T \mathbf{G} \mathbf{P}_j$ is equal to $\begin{bmatrix} 0 & \alpha \\ \alpha & 0 \end{bmatrix}$ for some scalar α so that $\mathbf{P}_j^T \mathbf{G} \mathbf{P}_j$ is either nonsingular or null, making possible an appeal to Theorem 27. The second is that

$$\tilde{\mathbf{P}}_k = \begin{bmatrix} \mathbf{u}_{s_{k-1}+1} & \mathbf{0} & \mathbf{u}_{s_{k-1}+2} & \mathbf{0} & \dots & \mathbf{u}_{s_k} & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_{s_{k-1}+1} & \mathbf{0} & \mathbf{v}_{s_{k-1}+2} & \dots & \mathbf{0} & \mathbf{v}_{s_k} \end{bmatrix}.$$

Now the matrix $\tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1} \tilde{\mathbf{P}}_k^T$ is unchanged if $\tilde{\mathbf{P}}_k \mathbf{M}$ is substituted for $\tilde{\mathbf{P}}_k$, where \mathbf{M} is any nonsingular matrix, and since column permutation is equivalent to postmultiplication by such a matrix we may assume when evaluating $\tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1} \tilde{\mathbf{P}}_k^T$ that

$$\tilde{\mathbf{P}}_k = \begin{bmatrix} \tilde{\mathbf{U}}_k & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{V}}_k \end{bmatrix},$$

where

$$\tilde{\mathbf{U}}_k = [\mathbf{u}_{s_{k-1}+1}, \mathbf{u}_{s_{k-1}+2}, \dots, \mathbf{u}_{s_k}]$$

with a similar expression for $\tilde{\mathbf{V}}_k$. Thus for the BiCG, MRZ and the Hegedüs-Galerkin algorithms, where $\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}$, we have

$$\tilde{\mathbf{Q}}_k^T = \begin{bmatrix} \mathbf{I} - \tilde{\mathbf{U}} \tilde{\mathbf{C}}_k^{-1} \tilde{\mathbf{V}}_k^T \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} - \tilde{\mathbf{V}}_k \tilde{\mathbf{C}}_k^{-T} \tilde{\mathbf{U}}_k^T \mathbf{A}^T \end{bmatrix}. \quad (7.40)$$

where

$$\tilde{\mathbf{C}}_k = \tilde{\mathbf{V}}_k^T \mathbf{A} \tilde{\mathbf{U}}_k.$$

Let now

$$\tilde{\mathbf{Z}}_k = \tilde{\mathbf{U}}_k \tilde{\mathbf{C}}_k^{-1} \quad (7.41)$$

and denote its last column by \mathbf{z}_{s_k} . The look-ahead version of BiCG becomes, from equation (7.34),

$$\mathbf{u}_{s_k+1} = (\mathbf{I} - \mathbf{z}_{s_k} \mathbf{v}_{s_k}^T \mathbf{A}) \mathbf{r}_{k+1} \quad (7.42)$$

while the corresponding equation for MRZ is, from equation (7.24),

$$\mathbf{u}_{s_k+1} = (\mathbf{I} - \tilde{\mathbf{Z}}_k \tilde{\mathbf{V}}_k^T \mathbf{A} - \mathbf{z}_{s_{k-1}} \mathbf{v}_{s_{k-1}}^T \mathbf{A}) \mathbf{A} \mathbf{u}_{s_k}. \quad (7.43)$$

For both these algorithms, for $s_k + 1 \leq i \leq s_{k+1} - 1$, \mathbf{u}_{i+1} is given by

$$\mathbf{u}_{i+1} = (\mathbf{I} - \mathbf{z}_{s_k} \mathbf{v}_{s_k}^T \mathbf{A}) \mathbf{A} \mathbf{u}_i. \quad (7.44)$$

with corresponding equations for \mathbf{v}_{s_k+1} and \mathbf{v}_{i+1} . For the BiCG and MRZ algorithms \mathbf{K} is indefinite so that Theorem 27 does not apply, and for these algorithms incurable breakdown is a remote possibility. For the Hegedüs algorithms, on the other hand, \mathbf{K} is equal to the unit matrix so that only one step of look-ahead is necessary [150]. Making the substitutions appropriate to the Hegedüs-Galerkin algorithm in equations (7.24) and (7.34) then yields, for the HS version (HG),

$$\mathbf{u}_{s_k+1} = (\mathbf{I} - \mathbf{z}_{s_k} \mathbf{v}_{s_k}^T \mathbf{A}) \mathbf{s}_{k+1}$$

and for the Lanczos version (HGL),

$$\mathbf{u}_{s_k+1} = (\mathbf{I} - \tilde{\mathbf{Z}}_k \tilde{\mathbf{V}}_k^T \mathbf{A} - \mathbf{z}_{s_{k-1}} \mathbf{v}_{s_{k-1}}^T \mathbf{A}) \mathbf{A}^T \mathbf{v}_{s_k}$$

while for both versions, if needed, equation (7.35) yields

$$\mathbf{u}_{s_k+2} = \mathbf{u}_{s_{k+1}} = (\mathbf{I} - \mathbf{z}_{s_k} \mathbf{v}_{s_k}^T \mathbf{A}) \mathbf{A}^T \mathbf{v}_{s_k+1}.$$

We finally derive the equations for the look-ahead version of QMR. For this algorithm equation (7.40) becomes

$$\tilde{\mathbf{Q}}_k^T = \begin{bmatrix} \mathbf{I} - \tilde{\mathbf{U}}_k \left(\tilde{\mathbf{V}}_k^T \tilde{\mathbf{U}}_k \right)^{-1} \tilde{\mathbf{V}}_k^T & \mathbf{O} \\ \mathbf{O} & \mathbf{I} - \tilde{\mathbf{V}}_k \left(\tilde{\mathbf{U}}_k^T \tilde{\mathbf{V}}_k \right)^{-1} \tilde{\mathbf{U}}_k^T \end{bmatrix}$$

and if $\tilde{\mathbf{Z}}_k$ is now defined by

$$\tilde{\mathbf{Z}}_k = \tilde{\mathbf{U}}_k \left(\tilde{\mathbf{V}}_k^T \tilde{\mathbf{U}}_k \right)^{-1} \quad (7.45)$$

with, as before, its last column denoted by \mathbf{z}_{s_k} we obtain

$$\mathbf{u}_{s_k+1} = \left(\mathbf{I} - \tilde{\mathbf{Z}}_k \tilde{\mathbf{V}}_k^T - \mathbf{z}_{s_{k-1}} \mathbf{v}_{s_{k-1}}^T \right) \mathbf{A} \mathbf{u}_{s_k} \quad (7.46)$$

and, for $s_k + 1 \leq j \leq s_{k+1} - 1$,

$$\mathbf{u}_{j+1} = (\mathbf{I} - \mathbf{z}_{s_k} \mathbf{v}_{s_k}^T) \mathbf{A} \mathbf{u}_j \quad (7.47)$$

with corresponding equations for \mathbf{v}_{s_k+1} and \mathbf{v}_{j+1} . These equations are somewhat simpler than those given in [117], which take into account certain numerical considerations designed to make the matrices $\tilde{\mathbf{V}}_k^T \tilde{\mathbf{U}}_k$ less ill-conditioned. For further details see the original reference. However, since \mathbf{G} and \mathbf{K} are both indefinite it is still remotely possible that QMR can suffer from incurable breakdown even if look-ahead is fully implemented.

7.3. More Krylov aspects*

We now consider the look-ahead algorithms from the point of view of Chapter 4, and see how Theorems 16 and 17 have to be modified for the more sophisticated algorithms of this chapter. We then discuss the implications of the analysis for the look-ahead algorithms.

Theorem 28. Let the matrices $\widehat{\mathbf{S}}_j$ be defined by equation (4.2) and be nonsingular for $j = s_1, s_2, \dots, s_{k+1}$ where $s_1 < s_2 < \dots < s_{k+1}$. Then the sequence of matrices $\{\mathbf{P}_j\}$ is computable by the look-ahead version of GODir for $1 \leq j \leq s_{k+1}$ and the matrices $\overline{\mathbf{P}}_j$ satisfy

$$\overline{\mathbf{P}}_j = \widehat{\mathbf{P}}_j \widehat{\mathbf{U}}_j \quad (7.48)$$

for some block unit upper triangular, and hence nonsingular, matrix $\widehat{\mathbf{U}}_j$.

Proof. By induction. We show that if the theorem is true for $1 \leq j \leq i$, where i is any integer such that $s_k \leq i \leq s_{k+1} - 1$, then it is also true for $j = i + 1$.

Since $\widehat{\mathbf{S}}_{s_k}$ is nonsingular by hypothesis it follows from Lemma 6 and equations (4.2), (7.3) and (7.4) that $\tilde{\mathbf{Q}}_k$ is computable and is given by

$$\tilde{\mathbf{Q}}_k = \mathbf{I} - \mathbf{G}\widehat{\mathbf{P}}_{s_k} \widehat{\mathbf{S}}_{s_k}^{-1} \widehat{\mathbf{P}}_{s_k}^T. \quad (7.49)$$

Denote now the submatrices of $\widehat{\mathbf{U}}_j$ by \mathbf{U}_{rs} , etc. Since equation (7.48) is assumed to hold for $j = i$ we have, from equations (1.31) and (4.1),

$$\mathbf{P}_i = \sum_{j=1}^i (\mathbf{KG})^{j-1} \mathbf{P}_1 \mathbf{U}_{ji}$$

so that

$$\mathbf{KG}\mathbf{P}_i = \sum_{j=1}^i (\mathbf{KG})^j \mathbf{P}_1 \mathbf{U}_{ji}. \quad (7.50)$$

For the look-ahead version of GODir and for $s_k \leq i \leq s_{k+1} - 1$,

$$\mathbf{P}_{i+1} = \tilde{\mathbf{Q}}_k^T \mathbf{KG}\mathbf{P}_i \quad (7.51)$$

so that since, from equations (4.1) and (7.49),

$$\tilde{\mathbf{Q}}_k^T (\mathbf{KG})^j \mathbf{P}_1 = \mathbf{O}, \quad 0 \leq j \leq s_k - 1, \quad (7.52)$$

it follows that

$$\mathbf{P}_{i+1} = \tilde{\mathbf{Q}}_k^T \sum_{j=s_k}^i (\mathbf{KG})^j \mathbf{P}_1 \mathbf{U}_{ji}. \quad (7.53)$$

Hence since $\mathbf{U}_{ii} = \mathbf{I}$ by hypothesis it follows from equation (7.49) that

$$\mathbf{P}_{i+1} = \widehat{\mathbf{P}}_{i+1} \begin{bmatrix} \mathbf{Y}_{i+1} \\ \mathbf{I} \end{bmatrix}$$

for some $\mathbf{Y}_{i+1} \in \mathbb{R}^{r^i \times r}$. But since equation (7.48) is true for $j = i$ this gives $\overline{\mathbf{P}}_{i+1} = \widehat{\mathbf{P}}_{i+1} \widehat{\mathbf{U}}_{i+1}$ where

$$\widehat{\mathbf{U}}_{i+1} = \begin{bmatrix} \widehat{\mathbf{U}}_i & \mathbf{Y}_{i+1} \\ \mathbf{O} & \mathbf{I} \end{bmatrix} \quad (7.54)$$

so that equation (7.48) is true for $i+1$, establishing the induction. Since the theorem is trivially true for $j = 1$ ($\widehat{\mathbf{U}}_1 = \mathbf{U}_{11} = \mathbf{I}$) it is true for $1 \leq j \leq s_{k+1} - 1$, completing the proof. ■

The corresponding theorem for GOMin is:

Theorem 29. Let $\widehat{\mathbf{S}}_j$ be defined by equation (4.2) and be nonsingular for $j = s_1, s_2, \dots, s_{k+1}$ where $s_1 < s_2 < \dots < s_{k+1}$. If the matrices \mathbf{T}_{s_i} are defined by equation (7.10) and the matrices $\mathbf{T}_{s_i}^T \mathbf{F}_i$ computed by the look-ahead version of GOMin are nonsingular for $1 \leq i \leq k$ then the sequences of matrices $\{\mathbf{P}_j\}$ and $\{\mathbf{F}_j\}$ are computable by GOMin for $1 \leq j \leq s_{k+1}$, and the matrices $\overline{\mathbf{P}}_j$ satisfy

$$\overline{\mathbf{P}}_j = \widehat{\mathbf{P}}_j \widehat{\mathbf{U}}_j \quad (7.55)$$

for some nonsingular block upper triangular matrix $\widehat{\mathbf{U}}_j$.

Proof. By induction. We show that if the theorem is true for $1 \leq j \leq i$, where i is any integer such that $s_k \leq i \leq s_{k+1} - 1$, then it is also true for $j = i + 1$.

Since $\widehat{\mathbf{S}}_{s_k}$ is nonsingular from the hypotheses of the theorem it follows from Lemma 6 and equations (4.2), (7.3) and (7.4) that $\tilde{\mathbf{Q}}_k$ is computable and is given by equation (7.49). Let first $i = s_k$. For this case

$$\mathbf{P}_{s_k+1} = \tilde{\mathbf{Q}}_k^T \mathbf{K} \mathbf{F}_{k+1} \quad (7.56)$$

so that, from equation (7.7),

$$\mathbf{P}_{s_k+1} = \tilde{\mathbf{Q}}_k^T \mathbf{K} \left(\mathbf{F}_k - \mathbf{G} \tilde{\mathbf{P}}_k \tilde{\mathbf{D}}_k^{-1} \tilde{\mathbf{P}}_k^T \mathbf{F}_k \right).$$

Now from equations (7.6) and (7.56),

$$\tilde{\mathbf{Q}}_k^T \mathbf{K} \mathbf{F}_k = \tilde{\mathbf{Q}}_k^T \tilde{\mathbf{Q}}_{k-1}^T \mathbf{K} \mathbf{F}_k = \tilde{\mathbf{Q}}_k^T \mathbf{P}_{s_{k-1}+1} = \mathbf{O}$$

since $\tilde{\mathbf{Q}}_k^T \tilde{\mathbf{P}}_k = \mathbf{O}$ and $\mathbf{P}_{s_{k-1}+1}$ is a submatrix of $\tilde{\mathbf{P}}_k$. Hence, from equation (7.10),

$$\mathbf{P}_{s_k+1} = -\tilde{\mathbf{Q}}_k^T \mathbf{K} \mathbf{G} \tilde{\mathbf{P}}_k \tilde{\mathbf{T}}_k^T \mathbf{F}_k. \quad (7.57)$$

Now equation (7.55) is assumed to be satisfied for $j = s_k$ so that

$$\tilde{\mathbf{Q}}_k^T \mathbf{K} \mathbf{G} \overline{\mathbf{P}}_{s_k} = \tilde{\mathbf{Q}}_k^T \mathbf{K} \mathbf{G} \widehat{\mathbf{P}}_{s_k} \widehat{\mathbf{U}}_{s_k}.$$

Hence, from equation (7.52),

$$\tilde{\mathbf{Q}}_k^T \mathbf{KG} \bar{\mathbf{P}}_{s_k} = \left[\mathbf{O} \dots \mathbf{O} \quad \tilde{\mathbf{Q}}_k^T (\mathbf{KG})^{s_k} \mathbf{P}_1 \right] \hat{\mathbf{U}}_{s_k}$$

and if

$$\hat{\mathbf{U}}_{s_k} = \begin{bmatrix} \mathbf{U}_{11} & \cdots & \mathbf{U}_{1,s_{k-1}} & \mathbf{U}_{1,s_k} \\ \vdots & \ddots & \cdots & \vdots \\ \mathbf{O} & \cdots & \mathbf{U}_{s_{k-1},s_{k-1}} & \mathbf{U}_{s_{k-1},s_k} \\ \mathbf{O} & \cdots & \mathbf{O} & \mathbf{U}_{s_k,s_k} \end{bmatrix}$$

we have

$$\tilde{\mathbf{Q}}_k^T \mathbf{KG} \bar{\mathbf{P}}_{s_k} = \left[\mathbf{O} \dots \mathbf{O} \quad \tilde{\mathbf{Q}}_k^T (\mathbf{KG})^{s_k} \mathbf{P}_1 \mathbf{U}_{s_k,s_k} \right]$$

where the number of null block columns in the right-hand side is $s_k - 1$. Since $\tilde{\mathbf{P}}_k$ is a submatrix of $\bar{\mathbf{P}}_{s_k}$ (see equation (7.3)) this implies that

$$\tilde{\mathbf{Q}}_k^T \mathbf{KG} \tilde{\mathbf{P}}_k = \left[\mathbf{O} \dots \mathbf{O} \quad \tilde{\mathbf{Q}}_k^T (\mathbf{KG})^{s_k} \mathbf{P}_1 \mathbf{U}_{s_k,s_k} \right]$$

where the right-hand side now has $s_k - s_{k-1} - 1$ null block columns. Postmultiplying this equation by $\tilde{\mathbf{T}}_k^T$ then gives, from equation (7.10),

$$\tilde{\mathbf{Q}}_k^T \mathbf{KG} \tilde{\mathbf{P}}_k \tilde{\mathbf{T}}_k^T = \tilde{\mathbf{Q}}_k^T (\mathbf{KG})^{s_k} \mathbf{P}_1 \mathbf{U}_{s_k,s_k} \mathbf{T}_{s_k}^T$$

so that, from equation (7.57),

$$\mathbf{P}_{s_k+1} = -\tilde{\mathbf{Q}}_k^T (\mathbf{KG})^{s_k} \mathbf{P}_1 \mathbf{U}_{s_k,s_k} \mathbf{T}_{s_k}^T \mathbf{F}_k.$$

This equation may be written, from equations (4.1) and (7.49), as

$$\mathbf{P}_{s_k+1} = \hat{\mathbf{P}}_{s_k+1} \begin{bmatrix} \mathbf{Y}_{s_k+1} \\ \mathbf{U}_{s_k+1,s_k+1} \end{bmatrix} \quad (7.58)$$

for some matrix \mathbf{Y}_{s_k+1} and where

$$\mathbf{U}_{s_k+1,s_k+1} = -\mathbf{U}_{s_k,s_k} \mathbf{T}_{s_k}^T \mathbf{F}_k. \quad (7.59)$$

Combining equation (7.58) and equation (7.55) with $j = s_k$ gives

$$\bar{\mathbf{P}}_{s_k+1} = \hat{\mathbf{P}}_{s_k+1} \hat{\mathbf{U}}_{s_k+1}$$

where

$$\hat{\mathbf{U}}_{s_k+1} = \begin{bmatrix} \hat{\mathbf{U}}_{s_k} & \mathbf{Y}_{s_k+1} \\ \mathbf{O} & \mathbf{U}_{s_k+1,s_k+1} \end{bmatrix}, \quad (7.60)$$

and since this is equation (7.55) with $j = s_k + 1$, the induction is established for $j = s_k$.

For $s_k + 1 \leq i \leq s_{k+1} - 1$, \mathbf{P}_{i+1} is given by equation (7.51) and the same argument used to derive equation (7.54) gives

$$\widehat{\mathbf{U}}_{i+1} = \begin{bmatrix} \widehat{\mathbf{U}}_i & \mathbf{Y}_{i+1} \\ \mathbf{O} & \mathbf{U}_{i+1,i+1} \end{bmatrix}$$

where

$$\mathbf{U}_{i+1,i+1} = \mathbf{U}_{ii} \quad (7.61)$$

(as we cannot now assume that $\mathbf{U}_{ii} = \mathbf{I}$). Since \mathbf{U}_{s_k, s_k} and \mathbf{U}_{ii} are both nonsingular by the induction hypothesis and since $\mathbf{T}_{s_k}^T \mathbf{F}_k$ is nonsingular from the hypotheses of the theorem it follows from equations (7.59) and (7.61) that both $\widehat{\mathbf{U}}_{s_k+1}$ and $\widehat{\mathbf{U}}_{i+1}$ are nonsingular block upper triangular. This establishes the induction for $s_k \leq i \leq s_{k+1} - 1$. Since the theorem is true for $j = 1$ ($\widehat{\mathbf{U}}_1 = \mathbf{U}_{11} = \mathbf{I}$) it is true for $1 \leq j \leq s_{k+1}$, completing the proof. ■

The last two theorems, together with Theorems 16 and 17, provide most of the theoretical basis underlying the look-ahead methods. If the matrices $\widehat{\mathbf{S}}_k$ as defined by equation (4.2) are nonsingular for $k = 1, 2, \dots, n$, the Lanczos forms of the original algorithm, i.e. the forms with no look-ahead, will always compute the exact solution of the relevant problem if exact arithmetic is used. This is also the case for the HS methods if, in addition, the matrices $\widehat{\mathbf{T}}_k$ defined by equation (4.3) are nonsingular. Thus in theory look-ahead is only needed if for some k , $1 \leq k \leq n$, $\widehat{\mathbf{S}}_k$ is singular. If this happens, Theorem 28 states that as long as there exists a nonsingular $\widehat{\mathbf{S}}_j$ for some j satisfying $k \leq j \leq n$ then the Lanczos form of the look-ahead version will also solve the relevant problem. In addition to this, Theorem 27 implies that for the algorithms considered in Chapter 3, if \mathbf{K} is positive definite, then $\widehat{\mathbf{S}}_{k+1}$ is always nonsingular if $\widehat{\mathbf{S}}_k$ is singular. Thus for those algorithms only one step of look-ahead is ever needed.

Unfortunately, at the time of writing, there seems to be no simple equivalent of Theorem 19 giving necessary and sufficient conditions for the matrix $\mathbf{F}_i^T \mathbf{T}_{s_i}$ of equation (7.31) to be nonsingular, and although the nonsingularity of these matrices is only sufficient for the existence of short recurrences there must always be the suspicion that their singularity is associated with stagnation of the algorithm. It would thus again appear that the Lanczos versions are more stable than the HS ones since they are not susceptible to this type of stagnation, but this analysis does not take rounding errors into account. Experience shows that rounding errors change everything and that in practice the HS methods are in fact the more reliable (see Chapter 9 below).

7.4. Practical details

One of the most difficult problems in numerical analysis is deciding whether or not a small computed quantity should be regarded as zero, and it is effectively this problem that has to be resolved at every iteration of a look-ahead Lanczos

or HS method. In this context the problem presents itself in Step 3 of the look-ahead version of the basic algorithm (see page 134) as the determination of the singularity or otherwise of $\tilde{\mathbf{P}}_i^T \mathbf{G} \tilde{\mathbf{P}}_i$, where \mathbf{G} is indefinite. There appears to be no single diagnostic procedure to cover the general case though specific tests have been devised for particular versions of the algorithm ([180], [36], [37], [150], [151], [117]).

Luenberger [180] appears to have been the first to propose what is in effect a look-ahead version of a CG method when he suggested the method of hyperbolic pairs. This is applied to the original algorithm of Hestenes and Stiefel when the latter is used to solve $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is symmetric but indefinite, and is based on the proof of Theorem 27. If vector sequences are being generated then, after one step of look-ahead, equation (7.37) may be written, from equation (7.36) with $\mathbf{G} = \mathbf{A}$, as

$$\tilde{\mathbf{D}}_{i+1} = \begin{bmatrix} \mathbf{p}_{s_i+1}^T \mathbf{A} \mathbf{p}_{s_i+1} & \mathbf{p}_{s_i+1}^T \mathbf{A} \mathbf{K} \mathbf{A} \mathbf{p}_{s_i+1} \\ \mathbf{p}_{s_i+1}^T \mathbf{A} \mathbf{K} \mathbf{A} \mathbf{p}_{s_i+1} & \mathbf{p}_{s_i+2}^T \mathbf{A} \mathbf{p}_{s_i+2} \end{bmatrix}.$$

If $\mathbf{p}_{s_i+1}^T \mathbf{A} \mathbf{p}_{s_i+1}$ is exactly zero and \mathbf{K} is positive definite then $\tilde{\mathbf{D}}_{i+1}$ is nonsingular and the next values of \mathbf{x} and \mathbf{p} can be determined. However if $\mathbf{p}_{s_i+1}^T \mathbf{A} \mathbf{p}_{s_i+1}$ is only small, Luenberger in effect set it equal to zero and computed the new \mathbf{x} and \mathbf{p} using the mutilated value of $\tilde{\mathbf{D}}_{i+1}$. Under certain circumstances, if $\tilde{\mathbf{D}}_{i+1}$ is near to singularity, this can introduce serious inaccuracies and the algorithm in this form was never really satisfactory. Fletcher [110] proposed, in effect, using the true matrix $\tilde{\mathbf{D}}_{i+1}$ but did not give a specific test for deciding when $\mathbf{p}_{s_i+1}^T \mathbf{A} \mathbf{p}_{s_i+1}$ is too small to be used as a simple divisor. He did, though, note an analogy between his algorithm and an algorithm by Bunch and Parlett [54] for factorising a symmetric indefinite matrix which experiences similar problems, and suggested that “it might be possible to adapt the test of Bunch and Parlett to work here”. Fletcher’s lack of enthusiasm for a look-ahead version of CG was based on the not unreasonable premise that it might be better to use a method like OD or CR that is incapable of crashing rather than try to shore up an inherently unstable algorithm. However both OD and CR can stagnate if \mathbf{A} is indefinite and to date nobody else seems to have taken up the challenge of finding an efficient and effective look-ahead version of CG when applied to indefinite systems.

An algorithm for which this challenge certainly has been taken up is the original version of QMR, but for this algorithm it was found that in order to achieve reliability it was necessary for three different tests to be simultaneously satisfied before taking a normal step. If in equations (7.45) - (7.47) we scale the matrices $\tilde{\mathbf{U}}_i$ and $\tilde{\mathbf{V}}_i$ so that their columns have unit length, denote the scaled versions by $\bar{\mathbf{U}}_i$ and $\bar{\mathbf{V}}_i$ and define $\bar{\mathbf{Y}}_i$ and $\bar{\mathbf{Z}}_i$ by

$$\bar{\mathbf{Y}}_i^T = (\bar{\mathbf{V}}_i^T \bar{\mathbf{U}}_i)^{-1} \bar{\mathbf{V}}_i^T \quad \text{and} \quad \bar{\mathbf{Z}}_i = \bar{\mathbf{U}}_i (\bar{\mathbf{V}}_i^T \bar{\mathbf{U}}_i)^{-1}$$

then since the matrix $\bar{\mathbf{V}}_i^T \bar{\mathbf{U}}_i$ must be nonsingular in order for the algorithm to function the first thing that could be checked is its smallest singular value, σ_1 say. If this is greater than some prescribed tolerance tol (say $tol = \varepsilon^{1/4}$ or $tol = \varepsilon^{1/3}$

where ε is the machine precision - see [204]) then the while-loop in Step 3 can be safely abandoned. However one problem with this approach is that in some cases σ_1 is *never* greater than the required tolerance. The algorithm remains trapped in the while-loop which has, in effect, created an *artificial* incurable breakdown. Even if this ultimate disaster is avoided the algorithm as described can show a marked sensitivity to the value of tol , not a desirable attribute in a “black-box” algorithm.

To overcome these problems it was suggested by Freund et al [117] that the above criterion be relaxed by reducing tol to ε but at the same time strengthening the procedure by the imposition of two further checks. The improved version of the algorithm escapes from the while-loop only if all three of the following conditions are satisfied:

- (1) $\sigma_1 \geq \varepsilon$
- (2) $\left\| \bar{\mathbf{Y}}_j^T \mathbf{A} \mathbf{u}_k \right\|_1 \leq n(\mathbf{A}) \|\mathbf{u}_k\|, \quad j = i-1, i, \quad \text{and}$
- (3) $\left\| \bar{\mathbf{Z}}_j^T \mathbf{A}^T \mathbf{v}_k \right\|_1 \leq n(\mathbf{A}) \|\mathbf{v}_k\|, \quad j = i-1, i,$

where $n(\mathbf{A})$ is some function of $\|\mathbf{A}\|$ whose detailed calculation is discussed in [117] and where the notation used here is that of equations (7.45) - (7.47).

Similar tests have been incorporated into the final versions of other algorithms [36], [37], [150], [151], with various degrees of success. Unfortunately there seems to be very little in the way of comparative testing to indicate which, if any, method is the best for a particular type of problem and some of the testing that has been done has lacked rigour. Like has occasionally not been tested against like, e.g. a version of algorithm A incorporating look-ahead being compared with a version of algorithm B that lacks this refinement, and this makes evaluation difficult. However the look-ahead version of QMR seems to perform reasonably satisfactorily, and this makes it all the more surprising that we still await an equally satisfactory look-ahead version of CG when the latter is applied to indefinite systems.

Chapter 8

General block methods

In previous chapters the block CG algorithm with block size greater than one has been used either as a theoretical device for unifying the discussion of methods as diverse as CG and MRZ, or it has been called upon to restore normality when the original methods have broken down. In the first of these applications the block size is constant, either one or two depending on the particular algorithm being analysed. In the second one it varies, being equal to either t_i or $2t_i$ where $t_i - 1$ is the number of look-ahead steps needed to rescue the algorithm should it crash during the i -th step. Other applications of block methods are the simultaneous solution of several systems of equations having the same coefficient matrix [193] or the solution of a single system of equations where block techniques have been introduced in the hope of exploiting the greater efficiency of parallel computers [70], [71], [146], [191].

Although the aims of these two applications are different, many of the techniques employed are the same and in both cases are based on the general block algorithm described in the first four sections of Chapter 3. In the next two sections we see how this basic algorithm has to be modified to overcome the different problems posed by each of these two applications.

8.1. Multiple systems

The interest here is in solving more than one system of equations having the same coefficient matrix,

$$\mathbf{G}\mathbf{X} = \mathbf{H}, \quad (8.1)$$

where \mathbf{H} is given. The original block size r is thus equal to the (arbitrary) number of systems we wish to solve and unlike the methods described in the following section may not be chosen to satisfy some other criteria. The basic equations defining the iterations are (3.1), (3.12) and (3.13) for the Lanczos version and, for the HS one,

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i \quad (8.2)$$

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i \quad (8.3)$$

$$\mathbf{P}_{i+1} = (\mathbf{I} - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{G}) \mathbf{K} \mathbf{F}_{i+1} \quad (8.4)$$

(equations (3.5), (3.12) and (3.13)). We consider here only the latter case and restrict our attention to positive definite matrices \mathbf{G} .

We first define the error matrix \mathbf{E} by

$$\mathbf{E} = \mathbf{X} - \mathbf{G}^{-1} \mathbf{H} = \mathbf{G}^{-1} \mathbf{F}$$

subscripting it as necessary, and denote the j -th columns of \mathbf{X} , \mathbf{F} and \mathbf{E} by \mathbf{x} , \mathbf{f} and \mathbf{e} respectively (we omit the subscript j for notational simplicity but use the subscript i to denote the iteration number). Taking the j th columns of \mathbf{X}_{i+1} and \mathbf{F}_{i+1} in equations (8.2) and (8.3) then yields

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{f}_i$$

and

$$\mathbf{f}_{i+1} = \mathbf{f}_i - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{f}_i$$

so that since, from equation (8.3), $\mathbf{P}_i^T \mathbf{F}_{i+1} = \mathbf{0}$, a Galerkin condition $\mathbf{P}_i^T \mathbf{f}_{i+1} = \mathbf{0}$ is satisfied for every column \mathbf{f}_{i+1} of \mathbf{F}_{i+1} . It then follows from Theorem 1 that since we have assumed \mathbf{G} to be positive definite, each step results in the minimisation of $\|\mathbf{e}\|_G$ for each individual column \mathbf{e} of \mathbf{E} over the manifold defined by

$$\mathbf{x}(\mathbf{z}) \equiv \mathbf{x}_i - \mathbf{P}_i \mathbf{z}$$

where, as we have noted, \mathbf{x} denotes the j th column of \mathbf{X} . From the definition of the energy norm $\|\mathbf{e}\|_G$ this is the same as minimising $\mathbf{e}^T \mathbf{G} \mathbf{e}$ for every column of \mathbf{E} or, equivalently, every diagonal element of $\mathbf{E}^T \mathbf{G} \mathbf{E}$. Note that these minimisations are independent since in each individual case a different Galerkin condition $\mathbf{P}_i^T \mathbf{f}_{i+1} = \mathbf{0}$ is satisfied. Less stringently this implies that $\text{trace}(\mathbf{E}^T \mathbf{G} \mathbf{E})$ is also minimised since the minimisations are independent. Since this error minimisation is so similar to that of the vector version of the algorithm whose convergence was discussed in Chapter 4 it is not surprising that a similar result to inequality (4.42) may also be obtained for the block version. If the energy norm of the error $\|\mathbf{e}\|_G$ is as defined in Chapter 1 then O’Leary [193] showed that

$$\frac{\|\mathbf{e}_{i+1}\|_G}{\|\mathbf{e}_1\|_G} \leq c \left(\frac{\kappa^{1/2} - 1}{\kappa^{1/2} + 1} \right)^i \quad (8.5)$$

where c , $0 < c \leq 2$, is some constant. The value of the constant varies with the particular column of \mathbf{F} but does not depend upon i . Its full expression is given in [193].

It must be admitted that the above upper bound for $\|\mathbf{e}_{i+1}\|_G / \|\mathbf{e}_1\|_G$ is something of a disappointment. If $\mathbf{G} \in \mathbb{R}^{n \times n}$ and the block-size is r then the number of iterations required for termination is n for the vector version but only n/r for the block one. We might therefore have expected that the factor $\left(\frac{\kappa^{1/2} - 1}{\kappa^{1/2} + 1} \right)^i$ of the

vector version (equation (4.42)) would be replaced by something like $\left(\frac{\kappa^{1/2}-1}{\kappa^{1/2}+1}\right)^r$ for the block one, but inequality (8.5) is probably fairly tight. For their version of BICG, Chronopoulos and Gear [71] report poor convergence for $r > 5$ while Nikishin and Yeremin [191] assert that the algorithm may become numerically unstable with increasing block size. In some unpublished work of one of the present authors (CGB) it was found that, for admittedly rather small systems, convergence was adversely affected unless n was an exact integer multiple of r . This is not surprising given the nature of the termination proof and may be related to the problem suffered by BICG that we next discuss, the loss of rank of the matrix \mathbf{P}_i .

As we have already observed, the basic BICG algorithm can crash if $\mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$ becomes singular either due to the indefiniteness of \mathbf{G} or to the loss of rank of \mathbf{P}_i . We dealt with the former problem in the previous chapter and now consider how to deal with the latter. The possibility of rescuing the algorithm when \mathbf{P}_i , or equivalently \mathbf{F}_i , becomes rank-deficient was pointed out by O'Leary [193] although no details of how the calculations should actually be performed were given in that paper. Nikishin and Yeremin [191] spelled these details out at some length, and although they were principally concerned with solving the vector problem $\mathbf{Gx} = \mathbf{h}$ and were interested in block methods mainly from the viewpoint of exploiting parallelism, they did show how it was possible to continue the algorithm when rank-deficiency arose. The following treatment is loosely based on their work but the details of recovering the complete solution of $\mathbf{GX} = \mathbf{H}$ are new.

We start by recalling the oblique projector \mathbf{Q}_i which is defined by

$$\mathbf{Q}_i = \mathbf{I} - \sum_{j=1}^i \mathbf{GP}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T, \quad (8.6)$$

where

$$\mathbf{D}_j = \mathbf{P}_j^T \mathbf{G} \mathbf{P}_j \quad (8.7)$$

and $\mathbf{P}_j^T \mathbf{G} \mathbf{P}_k = \mathbf{O}$, $1 \leq j < k \leq i$. This is exactly the same as equation (2.1) but here the matrices \mathbf{P}_j will not all have the same number of columns. As we are considering only the HS versions they are computed by

$$\mathbf{P}_j = \mathbf{Q}_{j-1}^T \mathbf{K} \mathbf{F}_j \mathbf{V}_j \quad (8.8)$$

for some matrices \mathbf{V}_j whose properties we consider below, thereby guaranteeing their mutual conjugacy. We then use the matrices \mathbf{P}_j to compute a sequence of approximate solutions \mathbf{X}_i and residuals \mathbf{F}_i which are defined not by but by $\mathbf{F}_i = \mathbf{GX}_i - \mathbf{H}$ where \mathbf{H} is constant but by

$$\mathbf{F}_i = \mathbf{GX}_i - \mathbf{H}_i \quad (8.9)$$

where $\mathbf{H}_1 = \mathbf{H}$ and, for $i \geq 1$,

$$\mathbf{H}_{i+1} = \mathbf{H}_i \mathbf{V}_i. \quad (8.10)$$

The matrices \mathbf{X}_i are computed by

$$\mathbf{X}_{i+1} = (\mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T \mathbf{F}_i) \mathbf{V}_i \quad (8.11)$$

and with the above definitions of \mathbf{F}_i and \mathbf{H}_i , and choice of \mathbf{X}_{i+1} , it follows that

$$\mathbf{F}_{i+1} = (\mathbf{I} - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T) \mathbf{F}_i \mathbf{V}_i. \quad (8.12)$$

We now consider the choice of \mathbf{V}_i . Let $\mathbf{F}_i \in \mathbb{R}^{n \times r_i}$. Equation (8.12) then implies that

$$\mathbf{V}_i \in \mathbb{R}^{r_i \times r_{i+1}} \quad (8.13)$$

and if \mathbf{F}_i has full column rank then we set $\mathbf{V}_i = \mathbf{I}$ so that from equation (8.13), $r_{i+1} = r_i$ and the steps defined by equations (8.11) and (8.12) are the normal block-CG steps. If, on the other hand, \mathbf{F}_i does not have full column rank assume that it has a column nullity $\nu_i > 0$. Then there exists a matrix $\mathbf{U}_i \in \mathbb{R}^{r_i \times \nu_i}$ of rank ν_i such that

$$\mathbf{F}_i \mathbf{U}_i = \mathbf{O} \quad (8.14)$$

or, from equation (8.9),

$$\mathbf{G} \mathbf{X}_i \mathbf{U}_i = \mathbf{H}_i \mathbf{U}_i. \quad (8.15)$$

Thus in a sense, $\mathbf{X}_i \mathbf{U}_i$ is a partial solution of the equations $\mathbf{G} \mathbf{X} = \mathbf{H}_i$. The matrix \mathbf{V}_i is now chosen so that \mathbf{T}_i , where

$$\mathbf{T}_i = [\mathbf{U}_i \ \mathbf{V}_i], \quad (8.16)$$

is square and nonsingular and dimensional considerations now imply that

$$r_{i+1} = r_i - \nu_i. \quad (8.17)$$

It follows that $\mathbf{F}_i \mathbf{V}_i$ must have full column rank for if not, since \mathbf{T}_i is nonsingular and $\mathbf{F}_i \mathbf{U}_i = \mathbf{O}$, \mathbf{F}_i would have a nullity greater than ν_i giving a contradiction.

We now derive an alternative expression for $\mathbf{P}_i^T \mathbf{F}_i \mathbf{V}_i$ in order to obtain a simpler version of equation (8.8). Trivially, from equation (8.12) with $i = j$, $\mathbf{P}_j^T \mathbf{F}_{j+1} = \mathbf{O}$ so that since the matrices $\{\mathbf{P}_j\}$ are assumed to be computed using equations (8.6) - (8.8) and are thus block-conjugate it follows from Theorem 3 that

$$\mathbf{P}_j^T \mathbf{F}_i = \mathbf{O}, \quad 1 \leq j \leq i-1. \quad (8.18)$$

Hence, from equation (2.1),

$$\mathbf{Q}_{j-1} \mathbf{F}_i = \mathbf{F}_i, \quad 1 \leq j \leq i. \quad (8.19)$$

Premultiplying this equation by $\mathbf{V}_j^T \mathbf{F}_j^T \mathbf{K}$ then gives, from equation (8.8),

$$\mathbf{P}_j^T \mathbf{F}_i = \mathbf{V}_j^T \mathbf{F}_j^T \mathbf{K} \mathbf{F}_i, \quad 1 \leq j \leq i, \quad (8.20)$$

so that

$$\mathbf{P}_i^T \mathbf{F}_i = \mathbf{V}_i^T \mathbf{F}_i^T \mathbf{K} \mathbf{F}_i \quad (8.21)$$

and, from equations (8.18) and (8.20),

$$\mathbf{V}_j^T \mathbf{F}_j^T \mathbf{K} \mathbf{F}_i = \mathbf{O}, \quad 1 \leq j \leq i-1. \quad (8.22)$$

Now, from the definition of \mathbf{U}_j ,

$$\mathbf{U}_j^T \mathbf{F}_j^T = \mathbf{O}$$

so that, from equation (8.16), $\mathbf{T}_j^T \mathbf{F}_j^T \mathbf{K} \mathbf{F}_i = \mathbf{O}$ or

$$\mathbf{F}_j^T \mathbf{K} \mathbf{F}_i = \mathbf{O}, \quad 1 \leq j \leq i-1, \quad (8.23)$$

since \mathbf{T}_j is nonsingular (note that the null matrix in this equation is an $r_j \times r_i$ matrix and since $r_j \geq r_i$ may have more rows than columns).

Consider now equation (8.12). This may be written

$$\mathbf{F}_{j+1}^T = \mathbf{V}_j^T \mathbf{F}_j^T - \mathbf{C}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{G}. \quad (8.24)$$

where

$$\mathbf{C}_i = \mathbf{P}_i^T \mathbf{F}_i \mathbf{V}_i$$

and is symmetric since, from equation (8.21), $\mathbf{C}_i = \mathbf{V}_i^T \mathbf{F}_i^T \mathbf{K} \mathbf{F}_i \mathbf{V}_i$. If \mathbf{K} is positive definite it then follows from the fact that $\mathbf{F}_i \mathbf{V}_i$ has full rank that \mathbf{C}_i is also positive definite. Moreover, from the definition of \mathbf{C}_i and its nonsingularity, \mathbf{P}_i has full column rank so that, if \mathbf{G} is positive definite and \mathbf{D}_i is defined by equation (8.7) then \mathbf{D}_i is also positive definite. The algorithm does not therefore break down.

We now post-multiply equation (8.24) by $\mathbf{K} \mathbf{F}_i$. This gives, from equation (8.23) and the symmetry of \mathbf{K} ,

$$\mathbf{C}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{G} \mathbf{K} \mathbf{F}_i = \mathbf{O}, \quad 1 \leq j \leq i-2, \quad (8.25)$$

or, since \mathbf{C}_j is nonsingular,

$$\mathbf{P}_j^T \mathbf{G} \mathbf{K} \mathbf{F}_i = \mathbf{O}, \quad 1 \leq j \leq i-2.$$

Thus, from equation (8.8) with $j = i$ and equation (8.6),

$$\mathbf{P}_i = (\mathbf{I} - \mathbf{P}_{i-1} \mathbf{D}_{i-1}^{-1} \mathbf{P}_{i-1}^T \mathbf{G}) \mathbf{K} \mathbf{F}_i \mathbf{V}_i$$

and we have essentially the same result that we would have obtained (equation (3.5)) had there been no cases where \mathbf{F}_i was rank-deficient. Provided therefore that we monitor the matrices \mathbf{F}_i for any loss of rank and choose the matrices \mathbf{V}_i appropriately we can very simply modify the original algorithm to take any such loss of rank into account. This initial phase of the algorithm terminates when, for some positive integer s_m , $\mathbf{F}_{s_m} = \mathbf{O}$.

We now define the scalars s_j , $1 \leq s_1 < s_2 \dots < s_m$ to be those values of i for which \mathbf{F}_i is rank-deficient so that, from equation (8.15),

$$\mathbf{G}\mathbf{X}_{s_j}\mathbf{U}_{s_j} = \mathbf{H}_{s_j}\mathbf{U}_{s_j}, \quad 1 \leq j \leq m, \quad (8.26)$$

where \mathbf{U}_{s_m} may be taken to be the unit matrix of appropriate order. At this point all the matrices \mathbf{X}_{s_j} and \mathbf{U}_{s_j} will have already been computed during the first phase of algorithm and it remains to show how the solution of $\mathbf{GX} = \mathbf{H}$ may be reconstructed from these components.

Since $\mathbf{V}_i = \mathbf{I}$ by definition for $s_{j-1} < i < s_j$ it follows from equation (8.10) that $\mathbf{H}_{s_j} = \mathbf{H}_{s_{j-1}}\mathbf{V}_{s_{j-1}}$. If $\mathbf{H}_1 = \mathbf{H}$ then since $\mathbf{H}_{s_1} = \mathbf{H}_1$ ($\mathbf{V}_i = \mathbf{I}$ for $1 \leq i < s_1$ if $s_1 > 1$), this gives

$$\mathbf{H}_{s_j} = \mathbf{H}\tilde{\mathbf{V}}_{j-1} \quad (8.27)$$

where $\tilde{\mathbf{V}}_0 = \mathbf{I}$ and

$$\tilde{\mathbf{V}}_j = \mathbf{V}_{s_1}\mathbf{V}_{s_2} \dots \mathbf{V}_{s_j}, \quad 1 \leq j \leq m-1. \quad (8.28)$$

Equation (8.26) may thus be written

$$\mathbf{G}\mathbf{X}_{s_j}\mathbf{U}_{s_j} = \mathbf{H}\tilde{\mathbf{V}}_{j-1}\mathbf{U}_{s_j}, \quad 1 \leq j \leq m, \quad (8.29)$$

so that if we define $\bar{\mathbf{V}}_j$ by

$$\bar{\mathbf{V}}_j = [\tilde{\mathbf{V}}_{j-1}\mathbf{U}_{s_j} \ \tilde{\mathbf{V}}_j\mathbf{U}_{s_{j+1}} \ \dots \ \tilde{\mathbf{V}}_{m-1}\mathbf{U}_{s_m}] \quad (8.30)$$

it follows that

$$\mathbf{G}[\mathbf{X}_{s_1}\mathbf{U}_{s_1} \ \mathbf{X}_{s_2}\mathbf{U}_{s_2} \ \dots \ \mathbf{X}_{s_m}\mathbf{U}_{s_m}] = \mathbf{H}\bar{\mathbf{V}}_1.$$

Thus if $\bar{\mathbf{V}}_1$ is square and nonsingular the solution \mathbf{X}^* of $\mathbf{GX} = \mathbf{H}$ is given by

$$\mathbf{X}^* = [\mathbf{X}_{s_1}\mathbf{U}_{s_1} \ \mathbf{X}_{s_2}\mathbf{U}_{s_2} \ \dots \ \mathbf{X}_{s_m}\mathbf{U}_{s_m}] \bar{\mathbf{V}}_1^{-1}. \quad (8.31)$$

We now show that $\bar{\mathbf{V}}_1$ is indeed square and nonsingular, and we do this by proving that

$$\bar{\mathbf{V}}_j = \tilde{\mathbf{V}}_{j-1}\tilde{\mathbf{T}}_j, \quad 1 \leq j \leq m, \quad (8.32)$$

where $\tilde{\mathbf{T}}_{s_m} = \mathbf{U}_{s_m}$ and

$$\tilde{\mathbf{T}}_j = \mathbf{T}_{s_j} \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \tilde{\mathbf{T}}_{s_{j+1}} \end{bmatrix}. \quad (8.33)$$

The proof is by reverse induction. Assume that equation (8.32) holds for $j = m, m-1, \dots, k+1$. We then show it to be true for $j = k$.

From equation (8.30),

$$\bar{\mathbf{V}}_k = \begin{bmatrix} \tilde{\mathbf{V}}_{k-1} \mathbf{U}_{s_k} & \bar{\mathbf{V}}_{k+1} \end{bmatrix}$$

so that, by the induction hypothesis,

$$\bar{\mathbf{V}}_k = \begin{bmatrix} \tilde{\mathbf{V}}_{k-1} \mathbf{U}_{s_k} & \tilde{\mathbf{V}}_k \tilde{\mathbf{T}}_{k+1} \end{bmatrix}$$

or, from equation (8.28),

$$\begin{aligned} \bar{\mathbf{V}}_k &= \tilde{\mathbf{V}}_{k-1} \begin{bmatrix} \mathbf{U}_{s_k} & \mathbf{V}_{s_k} \tilde{\mathbf{T}}_{k+1} \end{bmatrix} \\ &= \tilde{\mathbf{V}}_{k-1} \begin{bmatrix} \mathbf{U}_{s_k} & \mathbf{V}_{s_k} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \tilde{\mathbf{T}}_{k+1} \end{bmatrix} \\ &= \tilde{\mathbf{V}}_{k-1} \mathbf{T}_{s_k} \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \tilde{\mathbf{T}}_{k+1} \end{bmatrix} = \tilde{\mathbf{V}}_{k-1} \tilde{\mathbf{T}}_k \end{aligned}$$

from equations (8.16) and (8.33), establishing the induction. Now it follows from equation (8.30) with $j = m$ that equation (8.32) is trivially true for $j = m$ if $\tilde{\mathbf{T}}_m = \mathbf{U}_{s_m}$. Since, by construction, \mathbf{U}_{s_m} is nonsingular and \mathbf{T}_{s_k} is nonsingular for $1 \leq k \leq m-1$, equation (8.33) implies that $\tilde{\mathbf{T}}_j$ is nonsingular for $1 \leq j \leq m$ and hence that $\tilde{\mathbf{T}}_1$ is nonsingular. Finally, since $\tilde{\mathbf{V}}_0 = \mathbf{I}$ by definition, equation (8.32) yields $\bar{\mathbf{V}}_1 = \tilde{\mathbf{T}}_1$ so that $\bar{\mathbf{V}}_1$ is both square and nonsingular. Hence the solution of the original block system $\mathbf{G}\mathbf{x} = \mathbf{h}$ is given by equation (8.31).

8.2. Single systems

The block CG method can be used to solve single systems of equations, but although the maximum number of steps is reduced the increased amount of work per step more than compensates for any savings due to this reduction. Only in the context of parallel computing are these algorithms faster than their vector counterparts. They are, moreover, somewhat more complicated. Normally only one initial solution \mathbf{x}_1 to the original vector problem is given so before the block phase can begin, an initial block \mathbf{X}_1 has to be computed. Then at the end of the main (block) iteration the final (vector) solution has to be determined from the last block. An alternative procedure is to reduce progressively the block size during the iteration itself but this, as we shall see, brings its own problems.

The simplest of these methods is that of Hackbusch [146]. He starts with a given initial approximation \mathbf{x}_1 to the solution of $\mathbf{G}\mathbf{x} = \mathbf{h}$ and computes the columns \mathbf{x}_{1s} of the initial block \mathbf{X}_1 by

$$\mathbf{x}_{1s} = \mathbf{x}_1 - \mathbf{N}_s \mathbf{f}_1$$

where $\mathbf{f}_1 = \mathbf{G}\mathbf{x}_1 - \mathbf{h}$ and the matrices \mathbf{N}_s , $1 \leq s \leq r$, are arbitrary. Since these matrices play no further part in the proceedings this is tantamount to making an

arbitrary choice of \mathbf{X}_1 . Hackbusch sets $\mathbf{N}_r = \mathbf{O}$ so that the last column of \mathbf{X}_1 is the original initial approximation \mathbf{x}_1 . The second phase of the algorithm is a variant of BiCG (HS version) which is terminated after the calculation of \mathbf{X}_i and \mathbf{P}_i for some positive i . At this point the simplest way of determining the final approximate (vector) solution, \mathbf{x}_{i+1} say, would be to set $\mathbf{x}_{i+1} = \mathbf{X}_i \mathbf{a} + \mathbf{P}_i \mathbf{b}$, where \mathbf{a} and \mathbf{b} are chosen to satisfy some given criterion, e.g. the minimisation of $\|\mathbf{G}\mathbf{x}_{i+1} - \mathbf{h}\|$, but because the last column of \mathbf{X}_i is special (since $\mathbf{N}_r = \mathbf{O}$) Hackbusch sets

$$\mathbf{x}_{i+1} = \mathbf{x}_{ir} (1 - \mathbf{e}^T \mathbf{a}) + \bar{\mathbf{X}}_i \mathbf{a} + \mathbf{P} \mathbf{b}$$

where $\mathbf{a}, \mathbf{e} \in \mathbb{R}^{r-1}$, \mathbf{e} denotes the vector of ones and $\bar{\mathbf{X}}_i$ is \mathbf{X}_i with its last column (\mathbf{x}_{ir}) omitted. The vectors \mathbf{a} and \mathbf{b} are then chosen to satisfy the required criterion. Note that, in this case, only $2r - 1$ equations need to be solved to compute \mathbf{a} and \mathbf{b} .

Unlike the previous algorithm the next algorithm we describe, that of Chronopoulos and Gear [71], only requires one initial starting vector \mathbf{x}_1 and chooses \mathbf{W}_i by

$$\mathbf{W}_i = [\mathbf{Kf}_i \ \mathbf{KGKf}_i \ \dots \ \mathbf{K}(\mathbf{GK})^{r-1} \mathbf{f}_i]$$

for some nonsingular symmetric matrix \mathbf{K} where $\mathbf{f}_i = \mathbf{Gx}_i - \mathbf{h}$. The matrices \mathbf{P}_i are then computed by equation (1.52) and \mathbf{x}_{i+1} by equations (1.18) and (1.17). This algorithm is clearly a generalisation of the basic CG algorithm and is perhaps closer in spirit to the look-ahead versions of this algorithm than to the true block methods for which the columns of \mathbf{P}_1 are essentially arbitrary. Like the original algorithm it can be shown [71] that equations (2.1) and (1.52) may be replaced by

$$\mathbf{P}_{i+1} = \left(\mathbf{I} - \mathbf{P}_i (\mathbf{P}_i^T \mathbf{GP}_i)^{-1} \mathbf{P}_i^T \mathbf{G} \right) \mathbf{W}_{i+1},$$

and if $\bar{\mathbf{P}}_i$ is defined by equation (1.31), $\mathbf{p}_1 = \mathbf{Kf}_1$ and

$$\hat{\mathbf{P}}_i = [\mathbf{p}_1 \ \mathbf{KGp}_1 \ (\mathbf{KG})^2 \mathbf{p}_1 \ \dots \ (\mathbf{KG})^{r-1} \mathbf{p}_1]$$

then $\bar{\mathbf{P}}_i = \hat{\mathbf{P}}_i \hat{\mathbf{U}}_i$ for some upper triangular matrix $\hat{\mathbf{U}}_i$. If both \mathbf{G} and \mathbf{K} are positive definite the algorithm is robust as long as the columns of $\hat{\mathbf{P}}_i$ are linearly independent. The appropriate choices of \mathbf{G} and \mathbf{K} then yield algorithms based either on CG or CR, and both these versions are discussed in [71]. The basic ideas were subsequently extended to nonsymmetric systems by Chronopoulos [70].

In the above two algorithms the block size is either established initially, remaining constant throughout until the completion of the algorithm [146] or is re-established anew at the beginning of each step [71]. In the last block method that we describe the block size is progressively reduced throughout the iteration until it eventually becomes zero on completion. This algorithm (VBCG - we omit the "P" of Nikishin and Yeremin [191] since all our algorithms are preconditioned by the matrix \mathbf{K}) is based on a generalisation of the algorithm described earlier in this chapter. We assume that \mathbf{F}_i has full rank throughout since if not the algorithm may be modified as described in the previous section. That analysis is valid up to and including equation (8.24) but if we now decide to reduce the dimension of \mathbf{F}_{i+1} by choosing

$\mathbf{V}_i \neq \mathbf{I}$ then, from equation (8.12), \mathbf{V}_i will be rectangular and equation (8.22) will no longer imply equation (8.23). Moreover, since \mathbf{F}_i has full rank, there is no matrix \mathbf{U}_i such that $\mathbf{F}_i \mathbf{U}_i = \mathbf{O}$. The matrices \mathbf{U}_i and \mathbf{V}_i must therefore be chosen to satisfy other criteria.

For VBCG then we choose \mathbf{V}_i and \mathbf{U}_i arbitrarily subject only to the two conditions that \mathbf{T}_i , as defined by equation (8.16), is square and nonsingular and that

$$\mathbf{V}_i^T \mathbf{C}_{i-1} \mathbf{U}_i = \mathbf{O}. \quad (8.34)$$

Since \mathbf{C}_{i-1} is positive definite this implies that the columns of \mathbf{U}_i span the null space of $\mathbf{V}_i^T \mathbf{C}_{i-1}$ so that any matrix \mathbf{Z} that satisfies $\mathbf{V}_i^T \mathbf{C}_{i-1} \mathbf{Z} = \mathbf{O}$ may be expressed as $\mathbf{Z} = \mathbf{U}_i \mathbf{M}$ for some matrix \mathbf{M} .

There are now two possibilities, either that \mathbf{V}_{j+1} is square (and by definition equal to the identity matrix) or it is not, and to accommodate these two possibilities we now define the set S_i by

$$S_i = \{k \mid \mathbf{V}_k \neq \mathbf{I} \cap 1 \leq k \leq i\}.$$

Premultiply now equation (8.24) by \mathbf{V}_{j+1}^T and post-multiply it by $\mathbf{K}\mathbf{F}_i$. This gives, from equation (8.22),

$$\mathbf{V}_{j+1}^T \mathbf{C}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{G}\mathbf{K}\mathbf{F}_i = \mathbf{O}, \quad 1 \leq j \leq i-2$$

and this implies that if $j+1 \notin S_{i-1}$ then

$$\mathbf{P}_j^T \mathbf{G}\mathbf{K}\mathbf{F}_i = \mathbf{O}. \quad (8.35)$$

If, on the other hand, $j+1 \in S_{i-1}$ then the argument immediately following equation (8.34) implies that

$$\mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{G}\mathbf{K}\mathbf{F}_i = \mathbf{U}_{j+1} \mathbf{M}_{j+1} \quad (8.36)$$

for some matrix \mathbf{M}_{j+1} , or

$$\mathbf{P}_j^T \mathbf{G}\mathbf{K}\mathbf{F}_i = \mathbf{D}_j \mathbf{U}_{j+1} \mathbf{M}_{j+1}, \quad j+1 \in S_{i-1}. \quad (8.37)$$

It remains now only to determine \mathbf{M}_{j+1} .

If we define $\widehat{\mathbf{P}}_j$ and $\widehat{\mathbf{D}}_j$ by

$$\widehat{\mathbf{P}}_j = \mathbf{P}_j \mathbf{U}_{j+1} \quad (8.38)$$

and

$$\widehat{\mathbf{D}}_j = \widehat{\mathbf{P}}_j^T \mathbf{G}\widehat{\mathbf{P}}_j$$

then, from equation (8.7),

$$\mathbf{U}_{j+1}^T \mathbf{D}_j \mathbf{U}_{j+1} = \widehat{\mathbf{D}}_j \quad (8.39)$$

so that premultiplying equation (8.37) by \mathbf{U}_{j+1}^T yields, from equations (8.38) and (8.39),

$$\mathbf{M}_{j+1} = \widehat{\mathbf{D}}_j^{-1} \widehat{\mathbf{P}}_j^T \mathbf{G}\mathbf{K}\mathbf{F}_i. \quad (8.40)$$

Premultiplying equation (8.36) by \mathbf{P}_j gives, from equations (8.38) and (8.40),

$$\mathbf{P}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{GKF}_i = \widehat{\mathbf{P}}_j \widehat{\mathbf{D}}_j^{-1} \widehat{\mathbf{P}}_j^T \mathbf{GKF}_i, \quad j+1 \in S_{i-1}, \quad (8.41)$$

so that from equations (8.6), (8.8), (8.35) and (8.41),

$$\mathbf{P}_i = \left(\mathbf{I} - \mathbf{P}_{i-1} \mathbf{D}_{i-1}^{-1} \mathbf{P}_{i-1}^T \mathbf{G} - \sum_{j+1 \in S_{i-1}} \widehat{\mathbf{P}}_j \widehat{\mathbf{D}}_j^{-1} \widehat{\mathbf{P}}_j^T \mathbf{G} \right) \mathbf{K} \mathbf{F}_i \mathbf{V}_i \quad (8.42)$$

which is our final result. Note that the matrix in parentheses in the above equation is our customary oblique projector.

We consider now the storage aspects of this algorithm, in particular the memory required for storing the projector, and to assist us in this we define $n_c(\mathbf{M})$ to be the number of columns of \mathbf{M} where \mathbf{M} is an arbitrary matrix. From equation (8.13) we have $n_c(\mathbf{V}_i) = r_{i+1}$ so that, from equation (8.8),

$$n_c(\mathbf{P}_j) = r_{j+1}. \quad (8.43)$$

For $j+1 \in S_{i-1}$, since \mathbf{U}_{j+1} is assumed to satisfy equation (8.16) where $\mathbf{T}_{j+1} \in \mathbb{R}^{r_{j+1} \times r_{j+1}}$ and $\mathbf{U}_{j+1} \in \mathbb{R}^{r_{j+1} \times v_{j+1}}$, it follows that

$$n_c(\mathbf{U}_{j+1}) = r_{j+1} - r_{j+2}$$

so that, from equation (8.38),

$$n_c(\widehat{\mathbf{P}}_j) = r_{j+1} - r_{j+2}.$$

This, from equation (8.43), gives

$$n_c(\mathbf{P}_{j+1}) = n_c(\mathbf{P}_j) - n_c(\widehat{\mathbf{P}}_j).$$

Thus for every j for which $j+1 \in S_{i-1}$ implying that an additional matrix $\widehat{\mathbf{P}}_j$ appears in the sum in equation (8.42) the number of columns of \mathbf{P}_j is reduced by the precise number of columns of $\widehat{\mathbf{P}}_j$. Since the largest value of $j+1$ consistent with $j+1 \in S_{i-1}$ is $i-1$, even the most recent addition to the sum is reflected in the number of columns of \mathbf{P}_{i-1} and it follows from this that the total number of columns of \mathbf{P}_{i-1} and all the matrices $\widehat{\mathbf{P}}_j$ appearing in the sum in equation (8.42) is constant. If we assume that $\mathbf{V}_1 = \mathbf{I}$, i.e. that no reduction in the dimension of \mathbf{P}_i occurs until after the first step has been executed, this total number of columns will be r_1 , the number that would apply if no reduction in column numbers ever took place. Thus the storage required is essentially independent of the nature of the reduction scheme, the only variations being due to the different sizes of the matrices $\widehat{\mathbf{D}}_j$ if different numbers of columns are eliminated at any one time.

Clearly when the initial conditions for this algorithm are set up the matrices \mathbf{H} and \mathbf{V}_i must be chosen so that we do in fact solve the problem we want to solve. If, for example, we wish to solve the vector equation $\mathbf{Gx} = \mathbf{h}$ we can choose the

first column of \mathbf{H}_1 to be \mathbf{h} with the other columns being selected to satisfy other criteria. If \mathbf{V}_i , $i = 1, 2, \dots$, then has the form

$$\mathbf{V}_i = \begin{bmatrix} 1 & \mathbf{b}_i^T \\ \mathbf{0} & \mathbf{B}_{ii} \end{bmatrix}$$

for some vector \mathbf{b}_i and matrix \mathbf{B}_{ii} the first column of \mathbf{H}_i will be \mathbf{h} for all i so that if, for some i , $\mathbf{F}_i = \mathbf{O}$ the first column of \mathbf{X}_i will be the required solution.

Of these three algorithms, those of Hackbusch and of Nikishin and Yeremin are based on r distinct Krylov sequences, each commencing with one of the (effectively) arbitrary r columns of \mathbf{P}_1 whereas that of Chronopoulos and Gear is based on the same Krylov sequence that forms the basis of the vector version. This is r times as long as each of the r sequences on which the block algorithms are based. Now it is well-known that if a matrix \mathbf{B} has a simple dominant eigenvalue then the vectors of any Krylov sequence generated by \mathbf{B} tend to the eigenvector associated with that eigenvalue and this effect is more pronounced as the number of terms in the sequence increases. We would therefore expect that the "Krylov" matrix $\widehat{\mathbf{P}}_i$ defined by equation (4.1) and formed from several short Krylov sequences would have much better numerical properties than one formed from a single much longer sequence. That this is no false expectation is demonstrated by the following trivial example. Let $\widehat{\mathbf{P}}_1 = [\mathbf{e}_1 \ \mathbf{Be}_1 \ \mathbf{B}^2\mathbf{e}_1 \ \mathbf{B}^3\mathbf{e}_1]$ and $\widehat{\mathbf{P}}_2 = [\mathbf{e}_1 \ \mathbf{e}_4 \ \mathbf{Be}_1 \ \mathbf{Be}_4]$. Then, if

$$\mathbf{B} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

we have

$$\widehat{\mathbf{P}}_1 = \begin{bmatrix} 1 & 2 & 5 & 14 \\ 0 & -1 & -4 & -14 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad \text{and} \quad \widehat{\mathbf{P}}_2 = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 2 \end{bmatrix},$$

and $\text{cond}(\widehat{\mathbf{P}}_1) = 540.36$ and $\text{cond}(\widehat{\mathbf{P}}_2) = 5.83$.

This suggests that a possible strategy when choosing \mathbf{X}_1 would be the minimisation of the condition number of $\mathbf{P}_1^T \mathbf{P}_1$ insofar as this is practicable, a strategy not open to the Chronopoulos-Gear algorithms. The obvious choice would be $\mathbf{P}_1^T \mathbf{P}_1 = \mathbf{I}$ and this would be perfectly practicable provided that r , the number of columns of \mathbf{P}_1 , were not too large.

In the above discussion of block algorithms no particular structure was assumed for \mathbf{G} apart from symmetry so the algorithms described in this chapter are essentially modifications of the original CG method. However block versions of most if not all the algorithms of Chapter 3 (above) can be devised simply by replacing the vectors \mathbf{x} , \mathbf{z} , \mathbf{b} , \mathbf{c} , \mathbf{u} and \mathbf{v} of equations (4.65) - (4.67) by matrices and making the corresponding changes to the algorithms described in the four sections following those equations. Many of the algorithms so obtained are either untried or

do not feature in the literature. Saad [217] describes briefly a version of the GAA where $\mathbf{G} = \mathbf{I}$, $\mathbf{B} = \mathbf{A}$ and $\mathbf{H}_{j+1,j}$ is chosen to be upper triangular and to satisfy $\mathbf{P}_{j+1}^T \mathbf{P}_{j+1} = \mathbf{I}$. This is always possible provided that the columns of $\mathbf{Q}_j^T \mathbf{A} \mathbf{P}_j$ computed by Step 2(b) of the algorithm (see page 29) are linearly independent. With this choice of normalisation of the matrices \mathbf{P}_j it follows that the collected columns of all such matrices form an orthonormal set ready to be utilised in a block version of GMRES. The upper triangular nature of the blocks $\mathbf{H}_{j+1,j}$ leads to the matrix $\tilde{\mathbf{H}}_{i+1}$ defined by equation (2.12) having a band-Hessenberg rather than a block-Hessenberg structure. The reduction of this matrix to upper-triangular form is a straightforward generalisation of the procedure described in Appendix 1 (below) and is given in more detail in [217]. Saad also gives a modified-Gram-Schmidt version of the GAA together with a variation due to Ruhe [212], the latter approach resulting naturally in a band form for the upper Hessenberg matrix obtained by the algorithm. However the use of the block CG algorithm in [217] seems limited to obtaining a block version of GMRES. It is not seen as central to the whole theory as it is in our approach.

In deciding which of these algorithms to use there is little to guide us. If we want to solve a multiple system (more than one right-hand side) then we are forced to make some provisions for the matrices \mathbf{F}_i losing rank so that some procedure analogous to that described by Nikishin and Yeremin [191] is essential. This procedure will inevitably involve monitoring the matrices \mathbf{F}_i and then reconstructing the full solution from the partial one. We are therefore compelled to use some variation of the algorithm described in pages 153 - 157 above. If, on the other hand, we wish to use a block method to solve a single system then this will normally be because of a desire to exploit parallelism. Our choice will therefore be coloured by implementational considerations and also by the warnings of both Nikishin and Yeremin [191] and Chronopoulos and Gear [71] that the greater the number of columns of \mathbf{F}_i then the greater the probability of numerical instability. Given this it might be better to go for the algorithm VBCG by Nikishin and Yeremin because of its greater flexibility, despite its greater complexity compared with its two competitors. However in the absence of any comparative testing, any such choice is somewhat in the nature of a lottery.

Chapter 9

Some numerical considerations

Compared with the direct methods for solving linear equations, like Gaussian elimination and Cholesky decomposition, there is very little in the way of error analysis available for the Krylov algorithms. This is partly because the latter are more recent, dating from 1951 as opposed to for Cholesky, but is possibly also due to their greater complexity. For the HS versions in particular any error in the displacement matrix \mathbf{P}_j feeds through into the residual matrix \mathbf{F}_{j+1} which, in turn, feeds back into \mathbf{P}_{j+1} . It is thus not possible to analyse the two parts of the algorithm independently as it is for the Lanczos versions. However there are one or two areas where the analysis is sufficiently simple to have been attempted. One of these is the loss of orthogonality of the vectors generated by the Lanczos version of the algorithm with $\mathbf{K} = \mathbf{A}$ and $\mathbf{G} = \mathbf{I}$ [201]. This loss of orthogonality is related specifically to the use of the three-term as opposed to the full formula. Effectively if $\mathbf{A} = \mathbf{A}^T$ the projection matrix \mathbf{Q}_i of equation (2.1), which in vector form is given by

$$\mathbf{Q}_i = \mathbf{I} - \sum_{j=1}^i \frac{\mathbf{p}_j \mathbf{p}_j^T}{\mathbf{p}_j^T \mathbf{p}_j} \quad (9.1)$$

is replaced by the shorter

$$\mathbf{Q}_i = \mathbf{I} - \frac{\mathbf{p}_{i-1} \mathbf{p}_{i-1}^T}{\mathbf{p}_{i-1}^T \mathbf{p}_{i-1}} - \frac{\mathbf{p}_i \mathbf{p}_i^T}{\mathbf{p}_i^T \mathbf{p}_i}.$$

With this choice of \mathbf{Q}_i the vector \mathbf{p}_{i+1} is not forced to be orthogonal to the vectors \mathbf{p}_j , $1 \leq j \leq i-2$, as it would be if equation (9.1) were used. Any such orthogonality is merely implicit and is based on the supposition that $\mathbf{p}_j^T \mathbf{p}_k = 0$, $1 \leq j < k \leq i$ and that certain other conditions are satisfied. It may be guessed that the implicit nature of this calculation favours the propagation of rounding error and in practice it is found that as $|j - k|$ increases so does the loss of orthogonality until it reaches the level expected of random vectors. This loss (of conjugacy in the case where $\mathbf{G} \neq \mathbf{I}$)

can have disastrous effects on termination. Problems for which the exact solution should have been computed after n iterations sometimes need ten or more times that number (see Chapter 10 below). For these problems it is imperative to find a good preconditioner in order to obtain an acceptable solution in a reasonable time. Despite this, though, very few algorithms have been written off as unusable because of this type of instability. Even the best algorithms sometimes fail to converge unless a suitable preconditioner can be found.

Another area which has attracted some theoretical attention is the relationship between the *computed residuals* $\{\bar{\mathbf{r}}_i\}$ and the *true residuals* $\{\mathbf{A}\bar{\mathbf{x}}_i - \mathbf{b}\}$. In contrast with [133] we use $\bar{\mathbf{x}}_i$ and $\bar{\mathbf{r}}_i$ to denote the floating-point approximations to \mathbf{x}_i and \mathbf{r}_i (i.e. the values actually calculated by and stored in the computer), reserving the notation \mathbf{x}_i and \mathbf{r}_i for the values that would have been computed had exact arithmetic been used throughout. In many algorithms, e.g. the original CG algorithm (see page 51), the residuals and the approximate solutions are computed separately by two independent recursions in order to avoid additional matrix/vector multiplications. The concern here is that, after many iterations, the computed residual $\bar{\mathbf{r}}_i$ does not even approximate the true residual $\mathbf{A}\bar{\mathbf{x}}_i - \mathbf{b}$. This effect can be seen graphically if an algorithm is left to run for a very large number of iterations. The corrections to the approximate solutions together with the computed residuals often become smaller and ever smaller even though the corrections are so small that the approximate solutions (and hence the true residuals) remain constant. However for many algorithms, until such time as these corrections fall below the level of rounding error, there is a surprising measure of agreement between the true and the computed residuals (see Figures 9 - 14 and page 178 below for a more detailed discussion). An explanation of this phenomenon was first given by Greenbaum [133] who showed that for the HS methods the difference between the true and recursively-computed residuals is governed both by the basic accuracy to which a matrix/vector product may be computed and by the maximum over j of the ratio $\|\bar{\mathbf{x}}_j\| / \|\mathbf{x}^*\|$, where $\bar{\mathbf{x}}_j$ is the j -th approximate and \mathbf{x}^* is the true solution. Thus if the initial approximate solution is taken to be the null vector and the norms of its successors increase monotonically to their final value, the discrepancy between the true and the computed residuals will be quite small. A similar analysis for the Lanczos methods, on the other hand [144], reveals that the same basic behaviour is complicated by the inclusion of certain scalar multipliers and that these multipliers can become arbitrarily large. The effect on the approximate solution of large multipliers is similar to that of having a large value of $\|\bar{\mathbf{x}}_j\| / \|\mathbf{x}^*\|$ and is discussed in more detail below after deriving bounds for the discrepancy between the true and computed residuals.

Let then the sequence of approximate solutions $\{\mathbf{x}_i\}$ be computed by

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{p}_i \beta_i \quad (9.2)$$

and the residuals $\{\mathbf{r}_i\}$ by

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{A}\mathbf{p}_i \beta_i. \quad (9.3)$$

The vectors \mathbf{p}_i and scalars β_i may, for the purposes of this exercise, be regarded as being both arbitrary and exact. In fact of course they are neither, both being the

results of calculation, but since the identical values are used in both the computation of \mathbf{x}_{i+1} and that of \mathbf{r}_{i+1} any error in \mathbf{p}_i and β_i will affect the calculation of \mathbf{x}_{i+1} and \mathbf{r}_{i+1} equally and will not alter their relative discrepancy. There is no need, therefore, to refer to them by anything other than \mathbf{p}_i and β_i . This is not the case for \mathbf{x}_i and \mathbf{r}_i and we will use the notation $\bar{\mathbf{x}}_i$ and $\bar{\mathbf{r}}_i$ as described above for the computed values. With this notation the two basic operations that are carried out by the computer are

$$\bar{\mathbf{x}}_{i+1} = fl(\bar{\mathbf{x}}_i + \mathbf{p}_i \beta_i) \quad (9.4)$$

and

$$\bar{\mathbf{r}}_{i+1} = fl(\bar{\mathbf{r}}_i + \mathbf{A}\mathbf{p}_i \beta_i) \quad (9.5)$$

and we first need to bound the errors introduced by these two fundamental calculations. To do this we have to make certain assumptions about the nature of floating-point addition and multiplication and following Greenbaum [133], on whose work the following analysis is based, we assume that if α , β and γ are floating-point numbers, \mathbf{p} is a floating-point vector and \mathbf{A} a floating-point matrix then

$$fl(\alpha + \beta) = (1 + \varepsilon_1)\alpha + (1 + \varepsilon_2)\beta, \quad (9.6)$$

$$fl(\alpha\beta) = (1 + \varepsilon_3)\alpha\beta \quad (9.7)$$

and

$$fl(\mathbf{A}\mathbf{p}) = \mathbf{A}\mathbf{p} + \mathbf{q} \quad (9.8)$$

where ε_r , $r = 1, 2, 3$, satisfy $|\varepsilon_r| \leq \varepsilon$ and

$$\|\mathbf{q}\| \leq c \|\mathbf{A}\| \|\mathbf{p}\| \varepsilon \quad (9.9)$$

for some constant c . The quantity ε is the machine precision and is defined to be, for the particular computer under consideration, the smallest positive number δ for which $fl(1 + \delta) > 1$. If the vector $\mathbf{A}\mathbf{p}$ is computed by forming the product of the matrix \mathbf{A} and vector \mathbf{p} then the value of c may be deduced from inequalities (9.6) and (9.7) and depends on the dimensions of \mathbf{A} and the maximum number of non-zero elements in any one row. This vector though is sometimes evaluated implicitly (see [133]) and the more general form given above covers this case as well.

We start the analysis proper by obtaining an expression for $fl(\mathbf{v} + \mathbf{p}\beta)$ where \mathbf{v} is a floating-point vector. Since each element of the final vector is computed independently of all the others it follows from inequalities (9.6) and (9.7) that if $\mathbf{v} = [v_j] \in \mathbb{R}^n$ and $\mathbf{p} = [p_j] \in \mathbb{R}^n$ then, with obvious notation,

$$fl(v_j + p_j \beta) = (1 + \varepsilon_{1j})v_j + (1 + \varepsilon_{2j})(1 + \varepsilon_{3j})p_j \beta, \quad 1 \leq j \leq n.$$

This equation may be written in vector form

$$fl(\mathbf{v} + \mathbf{p}\beta) = (\mathbf{I} + \mathbf{E}_1)\mathbf{v} + (\mathbf{I} + \mathbf{E}_2)(\mathbf{I} + \mathbf{E}_3)\mathbf{p}\beta \quad (9.10)$$

where, for $1 \leq r \leq 3$, $\mathbf{E}_r = \text{diag}(\varepsilon_{rj})$. Since $|\varepsilon_{rj}| \leq \varepsilon$ it follows immediately that

$$\|\mathbf{E}_r\| \leq \varepsilon. \quad (9.11)$$

Applying equations (9.8) and (9.10) to equations (9.4) and (9.5) gives

$$\bar{\mathbf{x}}_{i+1} = (\mathbf{I} + \mathbf{E}_{1i}) \bar{\mathbf{x}}_i + (\mathbf{I} + \mathbf{E}_{2i}) (\mathbf{I} + \mathbf{E}_{3i}) \mathbf{p}_i \beta_i. \quad (9.12)$$

and

$$\bar{\mathbf{r}}_{i+1} = (\mathbf{I} + \mathbf{E}_{4i}) \bar{\mathbf{r}}_i + (\mathbf{I} + \mathbf{E}_{5i}) (\mathbf{I} + \mathbf{E}_{6i}) (\mathbf{A} \mathbf{p}_i + \mathbf{q}_i) \beta_i \quad (9.13)$$

for some matrices \mathbf{E}_{ri} , $r = 4, 5, 6$ and vector \mathbf{q}_i . Now $\|\mathbf{E}_{ri}\| \leq \varepsilon$, $1 \leq r \leq 6$, and \mathbf{q}_i satisfies inequality (9.9) so it is convenient to write equations (9.12) and (9.13) as

$$\bar{\mathbf{x}}_{i+1} = \bar{\mathbf{x}}_i + \mathbf{p}_i \beta_i + [\mathbf{E}_{1i} \bar{\mathbf{x}}_i + (\mathbf{E}_{2i} + \mathbf{E}_{3i}) \mathbf{p}_i \beta_i] + \mathbf{O}(\varepsilon^2) \quad (9.14)$$

and

$$\bar{\mathbf{r}}_{i+1} = \bar{\mathbf{r}}_i + \mathbf{A} \mathbf{p}_i \beta_i + [\mathbf{E}_{4i} \bar{\mathbf{r}}_i + (\mathbf{E}_{5i} + \mathbf{E}_{6i}) \mathbf{A} \mathbf{p}_i \beta_i + \mathbf{q}_i \beta_i] + \mathbf{O}(\varepsilon^2) \quad (9.15)$$

where the terms enclosed in square brackets are $\mathbf{O}(\varepsilon)$ (the notation $\mathbf{O}(\varepsilon^r)$ denotes a vector $\mathbf{y}(\varepsilon)$ such that $\lim_{\varepsilon \rightarrow 0} \|\mathbf{y} \varepsilon^{-r}\|$ is bounded above).

Define now the *discrepancy* \mathbf{d}_i by

$$\mathbf{d}_i = \bar{\mathbf{r}}_i - (\mathbf{A} \bar{\mathbf{x}}_i - \mathbf{b}). \quad (9.16)$$

Note that although a non-zero discrepancy implies that there is error in at least one of $\bar{\mathbf{r}}_i$ and $\bar{\mathbf{x}}_i$, it is quite possible for both of these to have large errors and yet for the discrepancy to be quite small or even zero. If we now premultiply equation (9.14) by \mathbf{A} , subtract \mathbf{b} from both sides and subtract the resulting equation from equation (9.15) we obtain

$$\mathbf{d}_{i+1} = \mathbf{d}_i + \mathbf{s}_i \quad (9.17)$$

where

$$\mathbf{s}_i = \mathbf{E}_{4i} \bar{\mathbf{r}}_i + (\mathbf{E}_{5i} + \mathbf{E}_{6i}) \mathbf{A} \mathbf{p}_i \beta_i + \mathbf{q}_i \beta_i - \mathbf{A} [\mathbf{E}_{1i} \bar{\mathbf{x}}_i + (\mathbf{E}_{2i} + \mathbf{E}_{3i}) \mathbf{p}_i \beta_i] + \mathbf{O}(\varepsilon^2)$$

Taking norms then yields, from inequality (9.9) and since $\|\mathbf{E}_{ri}\| \leq \varepsilon$,

$$\|\mathbf{s}_i\| \leq [\|\bar{\mathbf{r}}_i\| + \|\mathbf{A}\| (\|\mathbf{x}_i\| + (4+c) \|\mathbf{p}_i \beta_i\|)] \varepsilon + \mathbf{O}(\varepsilon^2). \quad (9.18)$$

This expression gives an upper bound for the error introduced into the calculation by the use of an auxiliary sequence, but in the above form it is somewhat unwieldy. To simplify it we first express $\|\mathbf{p}_i \beta_i\|$ in terms of $\|\mathbf{x}_i\|$ by appealing to equation (9.14) which, if written as

$$\mathbf{p}_i \beta_i = \bar{\mathbf{x}}_{i+1} - \bar{\mathbf{x}}_i - [\mathbf{E}_{1i} \bar{\mathbf{x}}_i + (\mathbf{E}_{2i} + \mathbf{E}_{3i}) \mathbf{p}_i \beta_i] + \mathbf{O}(\varepsilon^2),$$

immediately gives

$$\|\mathbf{p}_i \beta_i\| \leq \|\bar{\mathbf{x}}_{i+1}\| + \|\bar{\mathbf{x}}_i\| + O(\varepsilon). \quad (9.19)$$

Now at this stage it is convenient to define three more quantities, σ_i , ρ_i and θ_i by

$$\sigma_i = \frac{\|\mathbf{s}_i\|}{\|\mathbf{A}\|} \quad (9.20)$$

$$\rho_i = \frac{\|\mathbf{r}_i\|}{\|\mathbf{A}\|} \quad (9.21)$$

and

$$\theta_i = \max_{1 \leq j \leq i} \|\bar{\mathbf{x}}_j\|. \quad (9.22)$$

Substituting the expression for $\|\mathbf{p}_i \beta_i\|$ from inequality (9.19) into (9.18) and dividing by $\|\mathbf{A}\|$ then gives, for $i \geq 1$ and since $\theta_i \leq \theta_{i+1}$,

$$\sigma_i \leq [\rho_i + (9 + 2c) \theta_{i+1}] \varepsilon + O(\varepsilon^2). \quad (9.23)$$

We now turn our attention to the approximate residuals. Equation (9.17) with i replaced by $i - 1$, together with equation (9.16), gives

$$\bar{\mathbf{r}}_i = \bar{\mathbf{r}}_{i-1} + \mathbf{A} (\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_{i-1}) + \mathbf{s}_{i-1}$$

from which we obtain recursively, since $\bar{\mathbf{x}}_1 = \mathbf{x}_1$,

$$\bar{\mathbf{r}}_i = \bar{\mathbf{r}}_1 + \mathbf{A} (\bar{\mathbf{x}}_i - \mathbf{x}_1) + \sum_{j=1}^{i-1} \mathbf{s}_j. \quad (9.24)$$

Now the vector $\bar{\mathbf{r}}_1$ is not computed from equation (9.5) but by

$$\bar{\mathbf{r}}_1 = fl(\mathbf{Ax}_1 - \mathbf{b}). \quad (9.25)$$

Hence, from equations (9.8) and (9.10),

$$\bar{\mathbf{r}}_1 = (\mathbf{I} + \mathbf{E}_1)(\mathbf{Ax}_1 + \mathbf{q}_1) - (\mathbf{I} + \mathbf{E}_2)\mathbf{b} \quad (9.26)$$

for some matrices \mathbf{E}_1 and \mathbf{E}_2 and vector \mathbf{q}_1 where $\|\mathbf{E}_r\| \leq \varepsilon$, $r = 1, 2$ and, from equations (9.9) and (9.25),

$$\|\mathbf{q}_1\| \leq c \|\mathbf{A}\| \|\mathbf{x}_1\| \varepsilon. \quad (9.27)$$

Equation (9.26) may be written

$$\bar{\mathbf{r}}_1 = \mathbf{Ax}_1 - \mathbf{Ax}^* + \mathbf{s}_0 + \mathbf{O}(\varepsilon^2) \quad (9.28)$$

where

$$\mathbf{s}_0 = \mathbf{E}_1 \mathbf{Ax}_1 - \mathbf{E}_2 \mathbf{Ax}^* + \mathbf{q}_1, \quad (9.29)$$

and combining equations (9.24) and (9.28) then gives

$$\bar{\mathbf{r}}_i = \mathbf{Ax}_i - \mathbf{Ax}^* + \sum_{j=0}^{i-1} \mathbf{s}_j + \mathbf{O}(\varepsilon^2). \quad (9.30)$$

We note in parentheses that if $\mathbf{x}_1 = \mathbf{0}$ there is no rounding error when computing $\bar{\mathbf{r}}_1$ so that $\mathbf{E}_1 = \mathbf{E}_2 = \mathbf{O}$ and $\mathbf{q}_1 = \mathbf{0}$. Hence $\mathbf{s}_0 = \mathbf{0}$ and

$$\bar{\mathbf{r}}_i = \mathbf{A}\bar{\mathbf{x}}_i - \mathbf{A}\mathbf{x}^* + \sum_{j=1}^{i-1} \mathbf{s}_j. \quad (9.31)$$

since the term $\mathbf{O}(\varepsilon^2)$ also vanishes. Reverting to the principal argument, we now take norms of equation (9.30), divide by $\|\mathbf{A}\|$ and apply equations (9.20) - (9.22) to give

$$\rho_i \leq \theta_i + \theta^* + \sum_{j=0}^{i-1} \sigma_j + O(\varepsilon^2) \quad (9.32)$$

where $\theta^* = \|\mathbf{x}^*\|$. Inequality (9.23) then becomes, for $i \geq 1$,

$$\sigma_i \leq \left[\theta^* + (10 + 2c)\theta_{i+1} + \sum_{j=0}^{i-1} \sigma_j \right] \varepsilon \quad (9.33)$$

where, from equation (9.29) and inequality (9.27),

$$\sigma_0 \leq [\theta^* + (1 + c)\theta_1] \varepsilon. \quad (9.34)$$

We can further simplify expression (9.33). Let

$$k_i = \theta^* + (10 + 2c)\theta_i. \quad (9.35)$$

Inequality (9.33) becomes, for $i \geq 1$,

$$\sigma_i \leq \left(k_{i+1} + \sum_{j=0}^{i-1} \sigma_j \right) \varepsilon \quad (9.36)$$

and if inequality (9.34) is weakened to $\sigma_0 \leq k_1 \varepsilon$ a simple induction argument based on this inequality and inequality (9.36) shows that, since $k_i \leq k_{i+1}$,

$$\sigma_i \leq k_{i+1} (1 + \varepsilon)^i \varepsilon. \quad (9.37)$$

Now from equations (9.16) and (9.28),

$$\mathbf{d}_1 = \mathbf{s}_0 + \mathbf{O}(\varepsilon^2)$$

so it follows from equation (9.17) that

$$\mathbf{d}_i = \sum_{j=0}^{i-1} \mathbf{s}_j + \mathbf{O}(\varepsilon^2).$$

Thus, from inequality (9.37),

$$\frac{\|\mathbf{d}_i\|}{\|\mathbf{A}\|} \leq k_i [(1 + \varepsilon)^i - 1] + O(\varepsilon^2). \quad (9.38)$$

If $\mathbf{x}_1 = \mathbf{0}$, taking norms of equation (9.31) yields

$$\rho_i \leq \theta^* + \theta_1 + \sum_{j=0}^{i-1} \sigma_j$$

where now, since $\mathbf{s}_0 = \mathbf{0}$, $\sigma_0 = 0$. A similar argument to that used to establish inequalities (9.37) and (9.38) shows that these two inequalities are still valid but with i replaced by $i - 1$. The effect on the discrepancy of starting with $\mathbf{x}_1 = \mathbf{0}$ is thus equivalent to the effect of performing one fewer iteration.

We now examine the implications of inequality (9.38). Since $|\varepsilon|$ is small it reduces to

$$\frac{\|\mathbf{d}_i\|}{\|\mathbf{A}\|} \leq k_i \varepsilon i + O(\varepsilon^2) \quad (9.39)$$

so that, as might be expected, the norm of the discrepancy is proportional to both the number of iterations and to the machine precision. However it is convenient to express inequality (9.39) approximately as

$$\frac{\|\mathbf{d}_i\|}{\|\mathbf{A}\|} \leq \left(K_1 + K_2 \max_{1 \leq j \leq i} \|\mathbf{x}_j\| \right) i \quad (9.40)$$

where, from equations (9.22) and (9.35), $K_1 = \theta^* \varepsilon$ and $K_2 = (10 + 2c) \varepsilon$ and are independent of the algorithm. The algorithmic factors that affect the discrepancy are thus the number of iterations already performed and the size of the largest intermediate solution hitherto calculated. For an unstable algorithm like CGS this latter can become very large (a residual norm increase of 10^{10} was monitored in one of our tests, see Graph 101).

Of the two factors influencing $\|\mathbf{d}_i\|$, $\max_{1 \leq j \leq i} \|\bar{\mathbf{x}}_j\|$ is probably the more important. This certainty appears to be the case when solving the artificial problem⁷ $\mathbf{Ax} = \mathbf{0}$ starting with $\mathbf{x}_1 \neq \mathbf{0}$. In graphs 19 - 24, 27 and 28 the scaled discrepancies $diff$ defined by

$$diff = \frac{\|(A\bar{\mathbf{x}}_i - \mathbf{b}) - \bar{\mathbf{r}}_i\|}{\|A\bar{\mathbf{x}}_1\|} \quad (9.41)$$

and plotted against i yield a substantially flat line after a few minor oscillations at the beginning of the iteration. It thus appears that for this type of artificial problem the discrepancy, and with it the attainable accuracy without re-starting, is established at or near the beginning of the iteration and remains constant thereafter in the absence of strong instability. The greater importance of $\max_{1 \leq j \leq i} \|\bar{\mathbf{x}}_j\|$ is also evident when the algorithm is unstable and sudden increases of this quantity associated with increases in $\|A\bar{\mathbf{x}}_j - \mathbf{b}\|$ occur. In Graph 25, step increases in the discrepancy are clearly correlated with peaks in the relative true residual norm (RTR) occurring at around 5, 10 and 95 iterations, with a lesser increase at about 50 iterations. In Graph 26 the same pattern occurs at about 400 iterations, and is

⁷ See Scaling and starting, page 180 (below).

repeated with small but discernable steps corresponding to peaks at around 2400, 4600 and 5100 iterations.

When solving the more natural problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{b} \neq \mathbf{0}$ starting at $\mathbf{x}_1 = \mathbf{0}$ (Graphs 27a, 28a) the discrepancies of the more stable algorithms BiCG, BiCGL and BiCR increased steadily but slowly as the iteration proceeded. This could have been due to either of the two terms in equation (9.40) (i and $\max_{1 \leq j \leq i} \|\bar{\mathbf{x}}_j\|$), both of which could be expected to increase slowly and, in one case, monotonically. However even for this type of problem it is possible for the discrepancy to experience sudden jumps if the algorithm is unstable (Graph 27a). This phenomenon was also observed by van der Vorst. He reported in [241] that in one case involving 40,000 unknowns, the norm of the true residuals computed by CGS after two hundred iterations was some thousand times greater than that computed recursively. This certainly implies that for some j , $\|\bar{\mathbf{x}}_j\|$ must have been fairly large and indicates that under unfavourable conditions the norm of the recursively-computed residuals is not a true reflection of the accuracy of the computed solution. The two conclusions that can be drawn from this are that unstable algorithms like CGS should be avoided, and that recursive residuals should be distrusted as convergence indicators. Prudence dictates that the true residuals should always be computed before accepting the current $\bar{\mathbf{x}}_i$ as the required approximate solution.

A similar analysis to that carried out by Greenbaum for the HS algorithms was undertaken by Gutknecht and Strakoš [144] for a particular version of the Lanczos algorithms. In this version and using our own notation the successive approximate solutions are calculated by

$$\tilde{\mathbf{x}}_{i+1} = fl \left[(\tilde{\mathbf{r}}_i - \tilde{\mathbf{x}}_i \alpha_i - \tilde{\mathbf{x}}_{i-1} \beta_{i-1}) / \gamma_i \right] \quad (9.42)$$

and

$$\tilde{\mathbf{r}}_{i+1} = fl \left[(\mathbf{A}\tilde{\mathbf{x}}_i - \tilde{\mathbf{r}}_i \alpha_i - \tilde{\mathbf{r}}_{i-1} \beta_{i-1}) / \gamma_i \right], \quad (9.43)$$

where the quantities actually computed are denoted by tildes. For the purposes of error analysis the scalars α_i and β_i may be regarded as being arbitrary but in order that $\mathbf{r}_j = \mathbf{A}\mathbf{x}_j - \mathbf{b}$ for $j = i-1, i$ and $i+1$ it is necessary that $\alpha_i + \beta_{i-1} + \gamma_i = 0$. Hence, in floating-point arithmetic, $\alpha_i + \beta_{i-1} + \gamma_i = \varepsilon_i$ where $|\varepsilon_i|$ is suitably bounded. It was shown in [144] that if we define the discrepancy $\tilde{\mathbf{d}}_i$ by

$$\tilde{\mathbf{d}}_i = \tilde{\mathbf{r}}_i - (\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}) \quad (9.44)$$

then

$$\tilde{\mathbf{d}}_{i+1} = \tilde{\mathbf{d}}_1 - \sum_{j=0}^i \mathbf{l}_j \omega_j, \quad (9.45)$$

where $\omega_i = 1$ and, for $j = i-1, i-2, \dots, 1$,

$$\omega_j = 1 + \frac{\beta_j \omega_{j+1}}{\gamma_{j+1}}.$$

The vectors \mathbf{l}_j are local errors and have similar properties to the vectors \mathbf{s}_j of the 2-term analysis. The scalars ω_j are multiplication factors that govern the size of the error as it propagates through the calculation. It follows from equation (9.45) that there is always the possibility of a large discrepancy if any of the scalars ω_j is large. However equations (9.42) and (9.43) are not the same as our 3-term (Lanczos) recurrences which are given in block form by equations (3.11) - (3.13). They refer more appropriately to OrthoRes and similar algorithms.

Graphs 19 - 24 and 27, 27a, 28 and 28a indicate that in general the discrepancies for the Lanczos versions are almost invariably larger than those for their HS counterparts, that they are frequently established at the outset of the iteration and that they generally remain larger throughout the rest of the calculation. This affects the attainable accuracy of the algorithm and implies that the HS versions are inherently more accurate than the Lanczos ones. This belief is reinforced by experience (see [144]), by Graphs 13 - 18 and, in the context of QMR, by Graphs 3 - 8 and 29 - 34. In Graphs 13 - 18 in particular (three problems times two methods) two features stood out. The first is that the accuracy achieved by the HS versions was always better than that achieved by the Lanczos ones (sometimes considerably so, see Graph 14), and the second is that for the Lanczos versions the true and recursive relative residual norms (RTR and RRR) were identical (to graphical accuracy) throughout the iteration while for the HS versions they were identical only until achieving the attainable accuracy. A qualitative explanation of this behaviour on the part of the HS versions is given below (see Computing the residuals, page 178).

This page intentionally left blank

Chapter 10

And in practice...?

In this chapter we discuss the results of experimental comparisons carried out on the basic forms of the algorithms (i.e. without look-ahead or preconditioning). Our aim was not so much to determine the “best” algorithm but to identify if possible those attributes of both algorithms and matrices that give rise to particular behaviour patterns. Since resources were limited and much is already known about the behaviour of Krylov methods when applied to definite symmetric systems our efforts were concentrated on algorithms that solve general sets of linear equations involving real, nonsingular but nonsymmetric matrices. The same limitations on resources have prevented us carrying out our own tests on look-ahead and preconditioned versions of the algorithms.

We wrote our own software (using MATLAB) in order not to introduce differences that might arise from different implementations of some of the ancillary operations, like reducing an upper Hessenberg matrix to triangular form. Any observed differences in performance therefore were due solely to the properties of the basic Krylov methods unless otherwise stated. We did, though, test our own implementations against the publicly-available ones in order to verify the correct functioning of our programs. Thus our versions of BiCG, QMRBiCG, CGS, BiCGStab and GMRes were checked against MATLAB’s built-in versions for our six original test problems (see next section) and complete agreement was obtained in all cases except for GMRes and QMRBiCG. The algorithms HG, HGL, BiCR and BiCRL are not available as MATLAB routines and were checked against the original programs of Hegedüs [148]. Again there was excellent agreement between the two sets of implementations.

Whereas the programs for implementing most of the above algorithms followed fairly closely their published descriptions this was not the case for GMRes(m) and QMR. For the first of these algorithms our own implementation was based on the descriptions given in Appendix F. The MATLAB implementation is similar to ours but at the end of every block of m iterations (all implementations considered here are of GMRes(m)) the correction added to the intermediate solution computed at the end of the previous block of m iterations is recomputed using a more stable linear algebra routine than the one described in our Chapter 2. This is the essen-

tial difference between $\text{GMRes}(m)$ and $\text{GMRes}(m)^*$ but the MATLAB algorithm⁸ also performs other operations aimed at validating rather than obtaining a solution. These have not been taken into account when assessing the performance of $\text{GMRes}(m)^*$.

Clearly the MATLAB algorithm is considerably more elaborate than simple basic $\text{GMRes}(m)$ and in some cases this results in a quite significant reduction in the number of iterations required to achieve a given accuracy. Graphs 1, 2 and 9 - 11 show full details of five cases where the results differed, not all favouring the more sophisticated routine. However the inbuilt MATLAB version always performs some additional work after each cycle of m steps and this reduces its superiority. The comparative figures for workloads in terms of BEIs are given for both versions of the algorithm in Tables 5 - 8. These indicate that, out of the nine problems solved by all six versions of $\text{GMRes}(m)$ and $\text{GMRes}(m)^*$, $\text{GMRes}(m)^*$ was superior in terms of iterations in eleven cases and inferior in four, with both algorithms recording the same number of iterations in twelve cases. If on the other hand we compare BEIs, all twelve tied cases together with two of the eleven wins for MATLAB become wins for the simpler algorithm, making the final tally 18 - 9 in its favour. However the differences were small and it is possible that the more complex algorithm would prevail when presented with more challenging problems.

For QMR, in both the original (Lanczos) version [118] and the later (HS) version [119], Freund and Nachtigal used a superior routine for reducing an upper Hessenberg matrix to upper triangular form than the one we had used in conjunction with GMRes. Our equivalent algorithm, which we refer to as QMRBiCG, is essentially the second Freund-Nachtigal algorithm (Algorithm 7.1 of [119]) but with scaling omitted and using a simpler Givens procedure to effect the reductions to upper Hessenberg form (see page 120 above and Appendix A). It was tested against the implementation of the HS version of QMR in MATLAB 5 (which appears to be a straightforward transcription of Algorithm 7.1) and our own realisation of the original Lanczos algorithm. We refer to these as QMRBiCG* and QMR respectively. None of these three implementations uses look-ahead. Tests were carried out using the twelve matrices obtained by omitting nnc666 from those given in Table 4. They produced virtually identical results for QMRBiCG and QMRBiCG* in terms of number of iterations (see Tables 5 - 8) with the sole exception of e05r0500 for which the performances are shown in Graph 12. Further tests using the matrices listed in Table 3 (Graphs 3 - 8) together with six other matrices (Graphs 29 - 34) confirm that whereas there is little to choose between the two versions of QMRBiCG, they both comfortably out-perform QMR. This is consistent with the relative performances of the Lanczos and HS versions of other algorithms, see HS versus Lanczos, page 183 (below).

⁸ Throughout this chapter we use an asterisk to denote the built-in MATLAB implementation of an algorithm as opposed to the implementation, also in MATLAB, due to the present authors.

10.1. Presenting the results

The tests carried out by the authors were of two types. The first, where the results are displayed graphically with⁹ $\|\mathbf{A}\bar{\mathbf{x}}_i\| / \|\mathbf{A}\bar{\mathbf{x}}_1\|$ being plotted against the number of iterations needed to compute $\bar{\mathbf{x}}_i$, were intended to show differences in the behaviour of particular algorithms due to their structural differences. They include comparisons between the HS and Lanczos versions and the Galerkin and Minimum Residual versions of particular algorithms, and comparisons that demonstrate the effects of scaling. The second set of tests, whose results are presented in tabular form, are used to compare the performances of several algorithms on a collection of a dozen or so test problems.

When presenting the results, both graphically and in tabular form, we invariably use the true residuals (see Computing the residuals, page 178 (below)). Thus even though the residuals were computed recursively in the implementation of the algorithms, for the purposes of comparison and termination (and only for these purposes) the true residuals were used. The costs of the extra matrix-vector multiplications were not, of course, charged against the individual algorithms.

Another feature of our testing procedures when presenting the results graphically is that we let the algorithms continue calculating long after they would, in exact arithmetic, have solved the problems and thus for far more iterations than would be possible in any normal problem-solving environment. We could afford to do this because we were only solving a few problems, and we needed to do it in order to examine more closely the properties of the algorithms. For the compound algorithms like BiCG, BiCR and QMRBiCG for which both \mathbf{Ax} and $\mathbf{A}^T\mathbf{y}$ need to be computed, the number of matrix-vector products required to give an exact solution in the worst case and assuming exact arithmetic is $2n$, where n is the order of the matrix of coefficients. For GMRES this number is n but in our tests we often did not halt the algorithms until after 30,000 iterations, it being thought that this was sufficiently generous to permit even the least effective algorithms to demonstrate their capabilities.

Comparisons between different algorithms are displayed by means of tables. These give the number of iterations required to reduce the relative residuals $\|\mathbf{A}\bar{\mathbf{x}}_i\| / \|\mathbf{A}\bar{\mathbf{x}}_1\|$ to below a given tolerance tol , but since the amount of work per iteration varies considerably between algorithms we give another standard of comparison as well. For this extra comparison we first considered the number of matrix-vector products needed to solve a particular problem but discarded this as it does not take into account the (sometimes not inconsiderable) amount of additional routine linear algebra required by GMRes(m) if m is at all large. The use of flops was rejected as although these give a precise measure of the actual workload involved there is no immediate connection between the number of flops and the size of the problem. Comparisons based on CPU time were similarly rejected, the more so as their use would involve another variable, the speed of the computer. We finally settled

⁹ We set \mathbf{b} , the vector of right-hand sides, to be the null vector - see Scaling and starting, page 180 (below).

upon *BiCG equivalent iterations (BEIs)*. This number is essentially the number of iterations that would be required by BiCG to execute the same number of flops as the algorithm under test¹⁰. It is problem dependent since it includes the cost of performing at least one matrix-vector product per iteration, but involves only the mean number μ of nonzero elements per row of the matrix \mathbf{A} . It was chosen because it gives an extremely precise measure in readily-comprehensible terms of the work performed by an algorithm and relates immediately to the size of the problem since to solve $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{A} \in \mathbb{R}^{n \times n}$ by BiCG requires at most n iterations ($2n$ matrix-vector products) using exact arithmetic.

Let the number of flops per iteration be f_B for BiCG and f_T for the method under test. If the latter method requires I_T iterations for convergence then I_E , the number of BEIs, is given by

$$I_E = I_T (f_T / f_B) \quad (10.1)$$

(we have assumed here for simplicity that the number of flops for the first iteration is the same as for successive iterations). For all methods tested, with the exception of GMRes(m), it is possible to express f_T / f_B approximately as

$$f_T / f_B = 1 + k_T / (\mu + 4)$$

where $\mu = N/n$, n and N are respectively the number of rows (columns) and the number of nonzero elements of \mathbf{A} and k_T is a constant depending on the method. For GMRes(m), I_E is defined by

$$I_E = F_T / f_B$$

where F_T , the total number of flops required for the convergence of GMRes(m), replaces $I_T f_T$ in equation (10.1) and is computed individually for each problem. The values of f_T and f_B are given in “Algorithmic Details” (see Appendix F below) where they appear naturally in conjunction with the detailed descriptions of the algorithms.

10.2. Choosing the examples

The six examples used in our preliminary tests were selected on criteria like condition number, normality and symmetry. They were all taken from the Harwell-Boeing collection [94] so are based on real problems. They are also readily accessible to those who might wish to carry out further comparisons of the algorithms tested. The condition numbers of these matrices varied from 140 (jpwh991) to 6.1×10^{10} (arc130) and form part of the data of the Harwell-Boeing collection. It was therefore fairly straightforward to select matrices having the required range of condition numbers but somewhat less so when it came to choosing matrices according to their normality. No figures on normality are published with the Harwell-Boeing data,

¹⁰A similar device was used by Meijerink and van der Vorst [183].

perhaps because there is no generally-agreed measure for this particular attribute. The measure that we used in our selection process was

$$\nu = \frac{\sqrt{\sum_{i=1}^n |\lambda_i|^2}}{\|\mathbf{A}\|_F}.$$

This is the one described in More on GMRes (see page 91 above) and returns a figure between zero and unity, the former implying that the matrix is nilpotent while the latter characterising a normal matrix. Using this measure, the normality of the chosen matrices varied from 2.6×10^{-5} (arc130) to 0.999998 (steam2).

Even more surprisingly, there seems to be no single figure that acts as a measure of the symmetry of the matrices in the Harwell-Boeing collection, again perhaps because there is no agreed way of measuring this property. We therefore define the *symmetry* of a square matrix \mathbf{A} to be

$$\sigma = \frac{\|\mathbf{S}\|_F}{\|\mathbf{A}\|_F}$$

where $\mathbf{S} = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$ and the subscript F denotes the Frobenius matrix norm. Now it is well-known that any real square matrix \mathbf{A} can be expressed uniquely as the sum of a symmetric matrix \mathbf{S} and a skew-symmetric matrix \mathbf{T} but it is perhaps less well-known (although not difficult to show) that for this particular decomposition

$$\|\mathbf{A}\|_F^2 = \|\mathbf{S}\|_F^2 + \|\mathbf{T}\|_F^2.$$

This result indicates that for the proposed definition of symmetry, $0 \leq \sigma \leq 1$, the extreme values corresponding respectively to a skew-symmetric and a symmetric matrix. Using this definition, the selected matrices have symmetries varying from 0.4708 (e05r0500) to 0.999999 (steam2).

Another property of potential test matrices that we thought to monitor was the “positive reality” of the matrices since this property does seem to have some bearing on the likelihood that an algorithm will stagnate (see Chapter 2 above). A positive real matrix \mathbf{A} is one for which $\mathbf{A} + \mathbf{A}^T$ is positive definite but of all the nonsymmetric matrices that we examined only one (jpwh991) had the properties of a positive real matrix (and that, perversely, was negative real). We therefore decided that it was not worth while pursuing positive reality further at this stage.

The six original matrices together with their basic properties are listed below in Table 3, the condition numbers κ and the eigenvalues required for the normality ν being computed by MATLAB. These matrices were used in most of the preliminary tests including the comparisons between the Lanczos and HS versions of the algorithms and between the recursively-computed and “true” residuals. They were used for these tests in their “raw” form just as retrieved from the Harwell-Boeing collection, i.e. without any scaling to improve their numerical properties. The parameters given in Table 3 refer to these raw matrices. For the *comparison tests* between algorithms one of the six original matrices (nnc666) was dropped as even after scaling none of our algorithms could handle it, and the set was made up to

twelve matrices by the addition of seven further examples. These in general were of higher dimensions than those in the first set and were chosen to test the algorithms against larger problems. All the matrices used in the comparison tests were first subjected to Euclidean scaling and their parameters after scaling, together with those of nnc666, are given in Table 4. The test results are given in Tables 5 - 8.

10.3. Computing the residuals

In implementing the Krylov methods the generalised residuals \mathbf{F}_i may be computed in one of two ways, either recursively from equation (3.13)¹¹ which we repeat here for convenience

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G}\mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{P}_i^T\mathbf{F}_i \quad (10.2)$$

where $\mathbf{D}_i = \mathbf{P}_i^T\mathbf{G}\mathbf{P}_i$, or alternatively from the most recent intermediate solution \mathbf{X}_{i+1} by

$$\mathbf{F}_{i+1} = \mathbf{G}\mathbf{X}_{i+1} - \mathbf{H}.$$

In practice the residuals are invariably computed recursively in order to avoid the extra multiplications (matrix-matrix or matrix-vector) needed to compute the true residuals and this has some important consequences. To examine these we note that the HS version of the algorithm is effectively defined by equation (10.2) together with equations (3.12) and (3.21), which we again repeat here for convenience

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{P}_i^T\mathbf{F}_i \quad (10.3)$$

$$\mathbf{P}_{i+1} = \mathbf{K}\mathbf{F}_{i+1} + \mathbf{P}_i\mathbf{C}_i^{-1}\mathbf{C}_{i+1} \quad (10.4)$$

where $\mathbf{C}_i = \mathbf{F}_i^T\mathbf{K}\mathbf{F}_i$ (we consider only the HS forms since, as we shall see, they are almost always superior to the Lanczos ones). Once \mathbf{F}_1 has been obtained, either arbitrarily, or from an arbitrary \mathbf{X}_1 , or by some combination of these as is done for BiCG, BiCR, etc., \mathbf{P}_1 is set equal to $\mathbf{K}\mathbf{F}_1$ and \mathbf{F}_{i+1} and \mathbf{P}_{i+1} , $i = 1, 2, \dots$, may then be computed using *only* equations (10.2) and (10.4). Nothing at all is fed back into the calculation from equation (10.3). Thus the “motor” that drives the algorithm consists solely of the equations for computing \mathbf{F}_{i+1} and \mathbf{P}_{i+1} . The calculation of the intermediate solutions \mathbf{X}_{i+1} is, by comparison, little more than a sideshow.

In exact arithmetic the Krylov algorithms terminate automatically after a finite number of steps but with rounding error the algorithm can proceed indefinitely and the recursively-computed residuals can, if the iteration is not stopped arbitrarily, be progressively reduced until the algorithm fails through underflow. As the computed

¹¹In fact \mathbf{F}_{i+1} and \mathbf{X}_{i+1} are not computed from equations (3.13) and (3.12) but from the mathematically equivalent equations (3.20) and (3.19). We use the alternative versions here to simplify the derivation of inequality (10.6).

norms $\|\mathbf{F}_i\|$ decrease so do the corrections to \mathbf{X}_i . These are computed by equation (10.3) which may be written

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{M}_i \mathbf{F}_i \quad (10.5)$$

where

$$\mathbf{M}_i = \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{P}_i^T.$$

If \mathbf{G} is positive definite it is straightforward to verify that $\mathbf{G}^{\frac{1}{2}} \mathbf{M}_i \mathbf{G}^{\frac{1}{2}}$ is both symmetric and idempotent so that $\left\| \mathbf{G}^{\frac{1}{2}} \mathbf{M}_i \mathbf{G}^{\frac{1}{2}} \right\| = 1$. Since

$$\mathbf{M}_i \equiv \mathbf{G}^{-\frac{1}{2}} \left(\mathbf{G}^{\frac{1}{2}} \mathbf{M}_i \mathbf{G}^{\frac{1}{2}} \right) \mathbf{G}^{-\frac{1}{2}}$$

we have

$$\|\mathbf{M}_i\| \leq \left\| \mathbf{G}^{-\frac{1}{2}} \right\|^2 = \|\mathbf{G}^{-1}\|$$

so that, from equation (10.5),

$$\|\mathbf{X}_i - \mathbf{X}_{i+1}\| \leq \|\mathbf{G}^{-1}\| \|\mathbf{F}_i\|. \quad (10.6)$$

Thus if $\|\mathbf{F}_i\|$ decreases indefinitely then so does $\|\mathbf{X}_i - \mathbf{X}_{i+1}\|$. Now if the algorithm is permitted to run for a sufficient number of iterations the norm $\|\mathbf{F}_i\|$ can become so small that the increment added to \mathbf{X}_i to give \mathbf{X}_{i+1} is less than the amount needed to change \mathbf{X}_i when this is stored as a floating-point matrix or vector. When this happens $fl(\mathbf{X}_{i+1}) = fl(\mathbf{X}_i)$ and the computed approximate solutions cease to change. The “true” residuals remain constant and we have the spectacle that the recursively-computed residuals become smaller and smaller while the true ones remain unaltered. See Graphs 13 - 18 for illustrations of this and Gutknecht and Rosložník [142], [143] for a more detailed discussion of this and related phenomena.

Inequality (10.6) depends on \mathbf{G} being positive definite but even if this is not so, as in the case of BiCG, the residuals tend on the whole to behave in this fashion although now there is always the possibility that near-breakdown can cause a severe increase in the norm of \mathbf{X}_i . This can lead, via equations (9.22), (9.35) and (9.38), to a sharp increase in the upper bound for the norm of the discrepancy between the true and the computed residuals. The practical importance of this discrepancy arises from the fact that whereas the computed solution is related to the true residuals, the termination criteria are usually based on the already-available computed ones. If their norms are significantly smaller than those of the true residuals it is possible for the computed residuals to give a totally spurious estimate of the accuracy of the computed solution, and for this reason it is strongly urged that the true residuals are always evaluated before accepting any computed results as an approximate solution of a problem.

10.4. Scaling and starting

Although it is possible to apply the algorithms of the previous chapters to the original problems, significant improvements in performance can often be achieved by a simple scaling of the rows of the coefficient matrix and the corresponding elements of the right-hand side vector. This is equivalent to premultiplying the original equations $\bar{\mathbf{A}}\mathbf{x} = \bar{\mathbf{b}}$ by a diagonal matrix \mathbf{D} to give $\mathbf{A}\mathbf{x} = \mathbf{b}$ and is in fact a form of diagonal preconditioning. There are many possible strategies. One is to scale $\bar{\mathbf{A}} = [\bar{a}_{ij}]$ so that $\bar{a}_{ii} = 1$ for all i , where $\mathbf{A} = [a_{ij}]$, but this is only possible if the original matrix $\bar{\mathbf{A}}$ has no zeroes on its principal diagonal and may not be a good choice if some of these elements are small. It could be used in conjunction with column scaling if \mathbf{A} were symmetric positive definite. A popular alternative is to scale $\bar{\mathbf{A}}$ so that $\sum_{j=1}^n |\bar{a}_{ij}| = 1$ for all i , which we will refer to as *absolute* scaling. A third possibility is to scale $\bar{\mathbf{A}}$ so that $\sum_{j=1}^n \bar{a}_{ij}^2 = 1$ for all i , a process that we will call *Euclidean* scaling.

matrix	n	k	σ	ν
arc130	130	6.1×10^{10}	0.7071	2.6×10^{-5}
e05r0500	236	1.2×10^6	0.4708	0.9052
steam2	600	3.8×10^6	0.999999	0.999998
shl400	663	8.6×10^8	0.7071	0.0030
nnc666	666	2.0×10^{10}	0.9130	0.8169
jpwh991	991	140	0.9979	0.9957

Table 3: unscaled matrices

matrix	n	k	σ	ν
arc130	130	6.2×10^5	0.9838	0.9675
e05r0500	236	1.3×10^4	0.6079	0.7477
steam2	600	4986	0.9354	0.8660
shl400	663	2.3×10^5	0.7076	0.8254
nnc666	666	1.8×10^{10}	0.8172	0.6271
jpwh991	991	88	0.9958	0.9917
saylr3	1000	4.3×10^{18}	0.9829	0.9655
orsirr1	1030	7806	0.9910	0.9820
sherman4	1104	1213	0.9991	0.9982
mahindas	1258	6.3×10^6	0.7053	0.7183
watt2	1856	1.4×10^{11}	0.9869	0.8176
orsreg1	2205	6745	0.9910	0.9491
add20	2395	7074	0.9793	0.9581

Table 4: scaled (Euclidean) matrices

For each of our four basic algorithms (BiCG, QMRBiCG, HG and BiCR) and six problems (see Table 3) we tested three possibilities, namely no scaling, absolute

scaling and Euclidean scaling. The results are given in Graphs 35 - 58 and indicate that

- Scaling benefited all four methods of solution tested except for those problems involving the maverick matrix arc130
- There was no significant difference between absolute and Euclidean scaling for BiCG and QMRBiCG
- There was a small but fairly consistent advantage to be gained by using Euclidean scaling with HG and BiCR.

As a result of this, all further tests were carried out using Euclidean scaling. Of course scaling of the problem changes it somewhat. Since we plot residuals computed as $\mathbf{Ax}_i - \mathbf{b}$ rather than $\bar{\mathbf{A}}\bar{\mathbf{x}}_i - \bar{\mathbf{b}}$ we are not strictly comparing like with like but we assume that when comparing the results of absolute with Euclidean scaling this is comparatively unimportant. We might add here in parenthesis that the general problem of the equilibration of matrices when solving linear equations is beyond the scope of this work. Discussions may be found in e.g. [111] Chapter 11, [129] p. 124 or [258] pp. 192 - 194. It concerns us only as a special case of preconditioning (see Chapter 11 below).

We now consider the initialisation of the test problems and what results to present. A quantity that is both useful and simple to compute is

$$\rho_i = \frac{\|\mathbf{Ax}_i - \mathbf{b}\|}{\|\mathbf{Ax}_1 - \mathbf{b}\|} \quad (10.7)$$

where as before a bar indicates a computed value. The quantity ρ_i is not normally calculated since it requires an extra matrix-vector product and contributes nothing to the progress of the algorithm (see Computing the residuals, above). It is though, in its recursively-computed form ($\|\bar{\mathbf{r}}_i\| / \|\bar{\mathbf{r}}_1\|$) one of the more common measures of algorithmic performance but the information it provides is restricted to the residuals. Ideally we would like to know the behaviour of the errors $\bar{\mathbf{x}}_i - \mathbf{x}^*$, but this is only available if we know the solution \mathbf{x}^* of the problem we are trying to solve.

One way of obtaining \mathbf{x}^* is by solving $\mathbf{Ax} = \mathbf{b}$ using multiple-length arithmetic but this method is beset with difficulties. A common alternative is to choose an arbitrary vector for \mathbf{x}^* and then to compute the corresponding right-hand side \mathbf{b} by $\mathbf{b} = fl(\mathbf{Ax}^*)$. Often \mathbf{x}^* is chosen to be the vector of ones in order to reduce the possibility that $fl(\mathbf{Ax}^*) \neq \mathbf{Ax}^*$, and $\bar{\mathbf{x}}_1 = \mathbf{x}_1$ is set to be the null vector when carrying out the Krylov iterations. With these choices of initial values the ratio ρ_i becomes

$$\rho_i = \frac{\|\mathbf{Ax}_i - \mathbf{Ax}^*\|}{\|\mathbf{Ax}^*\|}.$$

Since we know \mathbf{x}^* the errors $\bar{\mathbf{x}}_i - \mathbf{x}^*$ may also be computed but if $\mathbf{b} = fl(\mathbf{Ax}^*) \neq \mathbf{Ax}^*$ then \mathbf{x}^* is *not* the solution of $\mathbf{Ax} = \mathbf{b}$ since \mathbf{b} will contain rounding error. The computed errors would not then be exact even if $fl(\bar{\mathbf{x}}_i - \mathbf{x}^*) = \bar{\mathbf{x}}_i - \mathbf{x}^*$. To overcome this problem, instead of putting $\mathbf{b} = fl(\mathbf{Ax}^*)$ and $\mathbf{x}_1 = \mathbf{0}$ we put $\mathbf{b} = \mathbf{0}$ and $\mathbf{x}_1 = \mathbf{x}^*$. Since the solution of $\mathbf{Ax} = \mathbf{0}$ is the null vector the errors are simply

the computed intermediate solutions $\bar{\mathbf{x}}_i$. Moreover as $\mathbf{b} = \mathbf{0}$ we have, from equation (10.7),

$$\rho_i = \frac{\|\mathbf{A}\bar{\mathbf{x}}_i\|}{\|\mathbf{A}\bar{\mathbf{x}}_1\|}.$$

Now suppose that in the previous scheme we put

$$\mathbf{b} = -fl(\mathbf{Ax}^*) \quad (10.8)$$

instead of $fl(\mathbf{Ax}^*)$. The initial residual would, since for this scheme $\mathbf{x}_1 = \mathbf{0}$, be given by $\bar{\mathbf{r}}_1 = fl(\mathbf{Ax}^*)$ which is *identical*, even to rounding error, to that computed in the scheme that we propose. Not only that, since the recursions defined by equations (10.2) and (10.4) depend only on \mathbf{K} , \mathbf{G} and \mathbf{F}_1 , it follows that if the initial shadow residuals \mathbf{s}_1 are chosen appropriately, not only $\bar{\mathbf{r}}_1$ but all subsequent residuals $\bar{\mathbf{r}}_i$ and $\bar{\mathbf{s}}_i$ as computed recursively will be identical (even down to rounding error) for both strategies. Hence if we solve $\mathbf{Ax} = \mathbf{0}$ rather than $\mathbf{Ax} = \mathbf{b}$ where \mathbf{b} is given by equation (10.8) the most important part of the calculation, the recursive computation of the sequences $\{\mathbf{r}_i\}$ and $\{\mathbf{p}_i\}$, will be identical for both right-hand sides. We can thus, by solving $\mathbf{Ax} = \mathbf{0}$, examine the exact performance (in terms of the perceived rate of convergence, i.e. the recursive residuals) of an algorithm solving $\mathbf{Ax} = -fl(\mathbf{Ax}^*)$ for arbitrary \mathbf{x}^* . The computed intermediate solutions $\{\bar{\mathbf{x}}_i\}$ though and the true residuals $\mathbf{A}\bar{\mathbf{x}}_i$ will be different. However since $\bar{\mathbf{x}}_i = \mathbf{e}_i$ it is straightforward to monitor the sequence of errors $\{\mathbf{e}_i\}$ and check if it is converging to the null vector.

To initiate the iteration we chose $\mathbf{x}_1^T = [1, 1, \dots, 1]$ in all cases except jpwh991 which suffered an immediate crash with this starting vector. For this problem we put $\mathbf{x}_1^T = [1, -1, 1, -1, \dots, 1]$. For all test problems for which it was needed we put $\mathbf{s}_1 = \mathbf{r}_1 = \mathbf{Ax}_1$.

10.5. Types of failure

We recognised three types of failure when assessing the performance of the algorithms. The first is instability which leads eventually to over- or underflow, usually because of a division by zero or a very small quantity. In this case the residual norms oscillate violently, sometimes by as much as 10^{10} . In the tables this type of failure is denoted by I. An (admittedly extreme) form of this type of instability is shown in Graphs 101 and 114. The second type of failure, denoted by S, is stagnation. The residual norms remain constant either indefinitely or until the program is terminated either by overflow or underflow. Examples of stagnation may be observed in Graphs 96 - 98 and 111 - 113. The third type of failure, which we denote by X, is convergence to an inaccurate (incorrect) solution and is due to the phenomenon described in the section on Computing the residuals, see page 178 (above). Here the norms of the recursive residuals still decrease but the true residual norms stay constant at some value greater than the prescribed tolerance. Examples are to be

found in Graphs 107 and 108. Finally if an algorithm fails after alternating periods of stagnation and instability it is denoted in the tables by I/S.

No causes of failure are attributed to the QMR variants since these failures were due to the breakdown of the algorithm whose residuals they were smoothing.

Problem	arc130	sherman4	steam2			
n	130	1104	600			
N	1037	3786	5660			
<i>symmetry</i>	0.9838	0.9991	0.9354			
<i>normality</i>	0.9675	0.9982	0.8660			
<i>cond</i>	6.2×10^5	1213	4986			
<i>tol</i>	10^{-14}	10^{-14}	10^{-14}			
GMRes(10)	140	159	642	949	22	23
GMRes(10)*	140	166	642	993	16	16
GMRes(20)	57	96	571	1402	29	41
GMRes(20)*	39	69	571	1444	16	22
GMRes(30)	99	223	478	1656	39	74
GMRes(30)*	24	48	478	1694	16	22
BiCG	38	38	124	124	16	16
QMRCBiCG	38	44	123	156	16	18
QMRCBiCG*	38	46	123	164	16	19
CGS	35	37	97	107	9	10
QMRCGS	39	57	97	169	9	13
BiCGStab	116	131	83	100	13	15
QMRBiCGStab	116	170	84	146	13	19
LSQR	112	108	542	506	32	31
HG	82	82	770	770	38	38
BiCR	84	88	786	839	38	40
$\frac{\ e\ }{\ \mathbf{x}_1\ }$:	largest	1.9×10^{-9}	3.5×10^{-12}	2.4×10^{-11}		
	smallest	2.6×10^{-11}	1.6×10^{-14}	1.7×10^{-13}		

Table 5: Number of iterations and of *BiCG-equivalent-iterations*

10.6. Heads over the parapet

10.6.1. HS versus Lanczos

Although we have shown in Chapter 4 that in exact arithmetic the Lanczos versions are more stable than the HS ones (since they are not prone to unlimited stagnation) the “folklore” of the subject, based on various pieces of anecdotal evidence and occasional numerical comparisons, suggests that in the presence of rounding error the very opposite is true. Theoretical support for such a belief came

with the papers of Greenbaum [133], Gutknecht and Strakoš [144] and Gutknecht and Rosložník [142], [143], and it is now a generally-accepted feature of the Krylov methods. Examination of Graphs 13 - 18 shows that for every combination of the featured two methods and three problems the final residual norm obtained by the HS version was always smaller than that obtained by the Lanczos one, sometimes by a considerable margin (Graphs 14, 18). If we set the convergence criterion to be 10^{-14} then the Lanczos version of the BiCG failed on jpwh991 and also (just) on steam2 while the same version of BiCR failed on arc130 and steam2. For the HS versions, on the other hand, the final residual norm was of the order of 10^{-16} for all six problem/method combinations.

Problem	add20	jpwh991	saylr3			
n	2395	991	1000			
N	13151	6027	3750			
<i>symmetry</i>	0.9793	0.9958	0.9829			
<i>normality</i>	0.9581	0.9917	0.9655			
<i>cond</i>	7074	88	$3.7 \times 10^{17}(\infty)$			
<i>tol</i>	10^{-13}	10^{-12}	10^{-13}			
GMRes(10)	727	927	105	128	3299	4755
GMRes(10)*	727	966	105	133	4637	6991
GMRes(20)	514	1046	95	181	1758	4201
GMRes(20)*	514	1074	95	185	1876	4616
GMRes(30)	407	1135	78	195	1973	6602
GMRes(30)*	407	1158	78	198	1794	6151
BiCG	276	276	70	70	358	358
QMRBiCG	263	318	69	83	349	439
QMRBiCG*	268	332	69	86	349	462
CGS	149	161	42	45	261	286
QMRCGS	152	240	41	64	262	448
BiCGStab	411	476	38	44	237	283
QMRBiCGStab	436	689	38	59	237	405
LSQR	3428	3248	282	268	3950	3697
HG	5966	5966	364	364	6065	6065
BiCR	5445	5732	360	378	4469	4758
$\frac{\ \mathbf{e}\ }{\ \mathbf{x}_1\ }$: largest		1.8×10^{-10}	9.9×10^{-12}	6.3×10^{-2}		
		6.3×10^{-12}	2.2×10^{-13}	6.3×10^{-2}		

Table 6: Number of iterations and of *BiCG-equivalent-iterations*

Similar results were obtained for equivalent versions of QMR. For a different set of test problems (see Graphs 29 - 34) the original 3-term (Lanczos) QMR failed to reduce the residual norm below 10^{-8} in all six cases whereas the two HS versions (the MATLAB implementation QMRBiCG* of the second Freund-Nachtigal algorithm and QMRBiCG, our own realisation of the same algorithm) reduced the

residual norm in four out of the six cases to 10^{-12} - 10^{-14} . For two of the problems, however (IMPCOLA and West0381) none of the three algorithms tested found a solution.

Problem	watt2	orsirr1	orsreg1			
n	1856	1030	2205			
N	11550	6858	14133			
<i>symmetry</i>	0.9869	0.9910	0.9910			
<i>normality</i>	0.8176	0.9820	0.9491			
<i>cond</i>	1.4×10^{11}	7806	6745			
<i>tol</i>	10^{-12}	10^{-12}	10^{-11}			
GMRes(10)	5160	6323	1079	1292	976	1183
GMRes(10)*	4420	5636	1013	1262	1081	1364
GMRes(20)	2090	4046	748	1401	708	1348
GMRes(20)*	1947	3871	745	1436	707	1382
GMRes(30)	1451	3854	677	1731	596	1559
GMRes(30)*	1451	3936	677	1771	600	1609
BiCG	I/S		533	533	433	433
QMRCBiCG	I		532	632	433	516
QMRCBiCG*	I		532	657	433	537
CGS	X		X		I	
QMRCGS	203	312		S		S
BiCGStab	208	239	504	575	395	452
QMRCBiCGStab	213	328	506	768	413	632
LSQR	2352	2238	X		10094	9611
HG	3810	3810	X		15899	15899
BiCR	2910	3053	14834	15530	15256	15989
$\frac{\ e\ }{\ \mathbf{x}_1\ }$:	largest	6.2×10^{-10}	1.0×10^{-12}	9.2×10^{-12}		
	smallest	2.0×10^{-12}	6.5×10^{-14}	1.3×10^{-13}		

Table 7: Number of iterations and of *BiCG-equivalent-iterations*

Although the evidence quoted above indicates that the HS form will compute a smaller residual norm than the Lanczos one, comparisons of LSQR against BiCR were less clear-cut. Both these algorithms share the same matrix \mathbf{KG} so although other factors (see equation (3.24) and the one immediately following) also come into play it might be expected that the final residual norm is smaller for BiCR (the HS version) than for LSQR (the Lanczos one). This does not always happen, possibly because LSQR generates its conjugate vectors \mathbf{w}_i from the orthogonal vectors \mathbf{v}_i via an orthogonal transformation (see page 67) whereas BiCR uses auxiliary sequences to perform this operation. LSQR is, moreover, apparently more robust. Its performances on one particular example under Matlab 5 (version 5.2.0.3084) and Matlab 6 (version 6.0.0.88 release 12) were virtually identical whereas BiCR was unable to reduce the residual norm to less than 10^{-14} under Matlab 5 but succeeded in doing

so under Matlab 6. Thus if (as is the case for some of our tests) the algorithm is halted when $\|\mathbf{r}\| < 10^{-14}$, LSQR eventually terminates under Matlab 5 whereas BiCR fails to do so. Under Matlab 6, BiCR terminates in fewer iterations than LSQR. Graph 12a gives the whole story.

Problem	e05r0500	shl400	mahindas
n	236	663	1258
N	5846	1712	7682
<i>symmetry</i>	0.6079	0.7076	0.7053
<i>normality</i>	0.7477	0.8254	0.7183
<i>cond</i>	13230	2.3×10^5	6.3×10^6
<i>tol</i>	10^{-11}	10^{-14}	10^{-14}
GMRes(10)	S	S	S
GMRes(10)*	S	S	S
GMRes(20)	S	S	S
GMRes(20)*	S	S	S
GMRes(30)	S	S	S
GMRes(30)*	S	S	S
BiCG	8167	8167	I
QMRBiCG	8014	8575	I
QMRBiCG*	7058	7676	I
CGS	X	I	I
QMRCGS	S	I	I
BiCGStab	I	I	I
QMRBiCGStab	I	I	I
LSQR	2147	2111	636
HG	2224	2224	520
BiCR	1838	1870	518
$\frac{\ \mathbf{e}\ }{\ \mathbf{x}_1\ }$: largest	1.3×10^{-9}	1.3×10^{-9}	2.0×10^{-10}
smallest	1.7×10^{-10}	1.7×10^{-10}	2.4×10^{-12}

Table 8: Number of iterations and of *BiCG-equivalent-iterations*

10.6.2. Galerkin versus minimum residual

The principal algorithms discussed in Chapters 2 and 3 which have both Galerkin and (minimum residual) versions are FOM (GMRes), CG (CR) and HG (BiCR). Of these we have elected not to test CG and CR as these algorithms apply only to symmetric matrices, and our testing of GMRes extends only to the restarted version GMRes(m) where $m = 10, 20, 30$. To compare this with a restarted version of FOM, FOM(m) say, could result in additional uncertainty precisely due to the effects of restarting (see [42] and [80] for further comparisons of FOM and GMRes). Thus the only evidence that we have concerning the relative performances of the

Galerkin and minimum residual versions comes from our tests on HG and BiCR, and the observed (and contradictory) evidence that whereas CG (Galerkin) is more popular than CR, GMRes(m) (minimum residual) is probably more popular than FOM(m).

An examination of Tables 5 - 8 indicates that for our own set of test examples, BiCR tends to outperform its Galerkin equivalent but not uniformly so. This conclusion is reinforced by Graphs 59 - 64 which show that for most of the time there is a small but visible gap between the smooth BiCR curve and the troughs of the more jagged HG one. If, however, we take the costs of the individual iterations into account by comparing BEIs there is very little to choose between them. This might be expected since both the Galerkin and minimum residual variants have the same value of \mathbf{KG} so that any significant differences in overall performance could be attributed to other numerical effects. These do not seem to be prominent in the cases tested but have been reported upon unfavourably by van der Vorst [241]. Perhaps the prudent course is to opt for minimum residuals and stability, and accept the small amount of extra work that this entails. Similar considerations to the above also apply to the closely-related pair of algorithms BiCG and QMRBiCG.

10.6.3. *Do we need residual smoothing?*

If the test problems we have chosen are in any way representative of problems as a whole then the answer to this question is that we probably do not. Comparisons of BiCG and QMRBiCG* using Tables 5 - 8 show that even though it reduced the number of iterations in six of the nine problems solved by both methods, only in one of these cases (e05r0500) was the reduction sufficient to offset the extra work caused by its implementation. This implementation increased the work per iteration by some 20% for most of the problems tested. The effects of applying residual smoothing to CGS and BiCGStab were even less encouraging. With CGS the number of iterations actually increased in three cases of the six that both methods solved, and was reduced in only one (jpw991) and then by only a single iteration. The actual workload was increased in every case, usually by some 30% - 40%. However residual smoothing did in one case permit the solution of a problem that the unmodified algorithm was unable to solve (CGS applied to watt2 - see Graphs 122 and 122a) and only one other method (BiCGStab) solved it more efficiently. The least successful application of residual smoothing was to BiCGStab, an algorithm with its own inbuilt stability. In no case did it reduce the number of iterations needed for a solution and in every case it increased the total workload, sometimes by more than 40%.

Overall the picture is that for most problems tested, the benefits of residual smoothing did not outweigh the disadvantages. However it must be stressed that the programs used in the tests (even the inbuilt MATLAB one) did not implement look-ahead, and look-ahead was a key feature of both the original papers on QMR [118], [119]. However look-ahead is a feature not of the QMR add-on but of the original Krylov algorithm. If our comparisons did not include look-ahead in the QMR versions, neither did they in the basic unsmoothed algorithms. The playing-

field may have been inappropriate but it was level.

10.6.4. Do we need look-ahead?

The evidence is conflicting. On the one hand we have the comparative lack of interest in the Fletcher-Luenberger look-ahead version of CG applied to symmetric indefinite systems but on the other we have algorithms like QMR [118], [119] and HG [150], [151] for which look-ahead was presumably regarded as essential since it formed part of the original descriptions of the algorithms. However the first QMR paper cited above contained several new ideas including, of course, the eponymous residual smoothing technique. Our tests suggest that this latter was perhaps not the most important innovation of that innovative algorithm and that a version without residual smoothing but with look-ahead might be equally successful. Such an algorithm, though, would be very similar to MRZ, again not one of the more popular algorithms. However the differences between QMR and MRZ may be significant. QMR generates two sets of vectors $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ that are mutually orthogonal, i.e. $\mathbf{v}_j^T \mathbf{u}_i = 0$ for $i \neq j$, whereas the vectors $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_i\}$ generated by MRZ are mutually conjugate, i.e. $\mathbf{v}_j^T \mathbf{A} \mathbf{u}_i = 0$, for $i \neq j$. We have already noted in the context of LSQR and BiCR that algorithms based on orthogonal vectors appear to be more stable than those based on conjugate ones and the same observation might apply here. However QMRBiCG also generates conjugate vectors and out-performs QMR, but the picture here is clouded by the accepted superiority of the HS algorithms over the Lanczos ones.

Another possibility is that the original MRZ algorithm was not as stable as it might have been, especially if many look-ahead steps were needed, and the more recent (and more stable) versions [39] have yet to make an impact. Again the paucity of comparative testing makes it difficult to give a fair assessment of the algorithms' relative potentials.

A similar paradox is associated with another QMR algorithm, the symmetric one for solving symmetric indefinite systems [120]. This fairly recent algorithm relies on look-ahead to overcome any problems caused by its double instability. Its basic numerical properties are thus no different from those of CG with an indefinite preconditioner and with Fletcher-Luenberger look-ahead yet the difference in perception of the two algorithms is considerable.

One of the major problems associated with look-ahead is the determination of precisely when it should be invoked, and it could be that differences in performance of various look-ahead algorithms are due principally to using different strategies when making this decision. Experiments suggest that true breakdowns (with divisors exactly equal to zero) are hardly ever encountered. Numerical difficulties arise when the divisors are small, resulting in cancellation and subsequent loss of conjugacy. The decision whether or not a divisor is small enough to justify a look-ahead step is not straightforward, often requiring more than one test for its implementation (see e.g. [118]). This would appear to be an area where more work to clarify the effects of different versions of an algorithm (e.g. with or without look-ahead

and/or residual smoothing) could be quite useful.

10.6.5. Do we need preconditioning?

There can be no equivocation about this, the answer is a resounding affirmative. Without some improvement in the properties of the matrix of coefficients even the best Krylov methods can be struggling, and sometimes even the simplest diagonal scaling can make a significant improvement in performance (see Graphs 35 - 58). But there is, we believe, a deeper reason than mere computational efficiency for introducing preconditioning at the outset of any discussion of the Krylov methods. The development of the theory without preconditioning, i.e. with $\mathbf{K} = \mathbf{I}$, results in virtually no simplification but destroys its symmetry. No longer can we contrast $\mathbf{p}_i^T \mathbf{G} \mathbf{p}_j = 0$ if $i \neq j$ with $\mathbf{f}_i^T \mathbf{K} \mathbf{f}_j = 0$ if $i \neq j$, or note that whereas an indefinite \mathbf{G} can result in crashing an indefinite \mathbf{K} can cause stagnation. Without both \mathbf{G} and \mathbf{K} the whole idea of duality is lost, and with it a simple way of accessing algorithms like OrthoRes and the reverse algorithms of Hegedüs.

When something gives both better numerical performance and deeper theoretical insight with an insignificant amount of extra effort, it is absolutely indispensable.

10.6.6. Do we need sophisticated linear algebra?

In the majority of the tests that we carried out, the use of more sophisticated linear algebra routines made very little difference in terms of numbers of iterations but did increase the workload per iteration. When it did reduce the number of iterations, though, the reduction was usually quite substantial and seemed to be more marked for GMRes than for QMRBiCG. Clearly some algorithms are going to be more sensitive to implementation details than others and it is important to know which are the more fragile. However it is also important to make the appropriate allowances for any extra workload when assessing the results of numerical tests so that accurate comparisons can be made. On balance it is probably better to go for superior linear algebra and pay the price in terms of reduced efficiency, especially if by so doing we can increase the number of problems that can be solved.

10.6.7. And the best algorithms...?

We have, so far, described four broad categories of algorithm. These are:

- (1) The basic HS algorithms which use two-term recurrences and generate displacement vectors conjugate to \mathbf{G} . Examples include CR and BiCG.
- (2) The basic “Lanczos-type” algorithms, referred to in the text as “Lanczos algorithms” and which use three-term recurrences. They generate conjugate displacement vectors which can be used without further modification to compute sequences of intermediate solutions. Examples of this type of algorithm include MCR and, in a look-ahead form, MRZ.
- (3) The true Lanczos algorithms which generate *orthonormal* vectors and which use some additional linear algebra, usually based on orthogonal transforma-

tions, to compute the sequence of intermediate solutions. Examples include GMRes and LSQR.

- (4) The Sonneveld algorithms. These are derived from BiCG (a first-category algorithm) and do not require premultiplication by \mathbf{A}^T . The two-best known examples of this type of algorithm are CGS and BiCGStab.

To all the above algorithms may be appended residual smoothing techniques of which the dominant but not only one is QMR.

Consideration of the test performances quoted in Tables 5 - 8 and the associated graphs indicate that the most robust algorithms are BiCR with 12/12 solutions followed by HG and LSQR with 11/12. These three algorithms would be expected to perform similarly as they share the same Krylov matrix \mathbf{KG} , and our tests confirm this expectation. The results for problem orsrr1 were a little unfortunate for both HG and LSQR in that had the stopping criterion been a little less stringent not only would they both have computed the solution but would probably have done so in fewer iterations than were needed by BiCR. Of these three algorithms, LSQR seems to have the edge. This is rather curious because our other tests (see Graphs 13 - 18) indicate that the Lanczos methods of the second category are inferior to the corresponding HS ones of the first category. We can only surmise that the calculation of orthogonal vectors by a Krylov method is inherently less prone to error than the similar calculation of conjugate ones, and that this increased accuracy is then preserved by the excellent numerical properties of the subsequent orthogonal transformations.

Although LSQR, HG and BiCR solved all, or nearly all, of our twelve test problems they did so in a somewhat leisurely fashion. Despite there being differences between the three individual algorithms, overall they form a recognisable group in terms of performance. The common Krylov matrix \mathbf{KG} for the unpreconditioned algorithms is

$$\mathbf{KG} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix}$$

and as was pointed out in the section on Galerkin methods, see page 195, below, there is no “perfect preconditioner” for these three algorithms (if the “perfect preconditioner” is applied to HG it turns into BiCG). This feature may adversely affect their performances. Another possibility is that the crucial factor determining the performance of a Krylov algorithm is the spectrum of \mathbf{KG} and this consists, for these three algorithms and from the above equation, of the positive and negative square roots of the *singular* values of \mathbf{A} . Even though these are real it could be that their distribution (along the real axis) is less favourable to the Krylov algorithms than that of the more undisciplined eigenvalues of \mathbf{A} which govern the performance of BiCG. Despite their robustness, the slow convergence of these methods makes them unsuitable as routine problem solvers in the forms that we have tested them. However we did not examine the look-ahead versions and it is virtually certain that look-ahead would lead to improved performance. It would, though, lead to improved performance of the other algorithms considered which, in their simple forms, are very much more efficient. In the absence of comparative testing no firm conclusions

can be drawn. Perhaps the rôle of these three methods is to be called upon to solve problems that are too difficult for their faster but less robust competitors.

The next most robust algorithms are GMRes(m), BiCG and BiCGStab, all with 9/12 success rates. In terms of BEIs there was not a great deal to choose between BiCG and BiCGStab, their common success in eight of the nine problems solved underlining the derivation of the second algorithm from the first. The features of BiCGStab that give it an edge over BiCG are that it is transpose-free and that its convergence is considerably smoother (see Graphs 77 - 88). This latter property may make it less prone to converge to an inaccurate solution when solving large and difficult problems, something that did not occur with BiCG when solving the comparatively small problems ($n < 2500$) of our tests. The failures of BiCG occurred after fairly long periods of either stagnation or strong instability (see Graphs 99, 114 and 120) while those of BiCGStab followed bouts of either mild instability or of stagnation punctuated by brief unstable episodes (Graphs 94, 103 and 118). Given the relationship between BiCG and BiCGStab and given that **G** and **K** for the former algorithm are both indefinite, these patterns of failure are not surprising.

The behaviour of the equally-successful GMRes(m) was quite different, failure when it occurred being due to prolonged stagnation. GMRes can stagnate if **A** is not positive real and none of the three difficult matrices were. Graphs 89 - 91, 96 - 98 and 111 - 113 show just how little progress the algorithms made. For the nine problems that it did solve, the best GMRes(m) results were inferior to those of BiCGStab in terms of iterations in 7/9 cases and in terms of BEIs in 8/9 cases (see Tables 3 - 5). The former result was surprising given the popularity of GMRes and the random choice of test problems, but if the choice of examples was fortuitously unfavourable to GMRes, the poor results in terms of workload were not. In particular, for large m , GMRes(m) has to perform a great deal of linear algebra per iteration in addition to computing \mathbf{Ap} for some vector \mathbf{p} . For some problems these overheads amounted to more than ten times as much work as the matrix-vector product and this contributed strongly to the number of BEIs exceeding the number of iterations by factors of more than three. This relatively high overhead is characteristic of GMRes(m) but can be reduced if m is small and the ratio N/n , the mean number of nonzero elements per row of the matrix, is large. It implies that GMRes(m) should record relatively better performances in terms of BiCG equivalents for comparatively dense matrices.

The other algorithm that performed as well as GMRes and BiCGStab in terms of number of problems solved was QMRBiCG/QMRBiCG*. As we have seen, this algorithm may be regarded as a scaled BiCG algorithm with a QMR add-on. Moreover our (albeit limited) testing indicates that at least for some problems, residual smoothing is not particularly effective when used with this simplified version of the algorithm. This suggests that we should perhaps use the full version (i.e. with look-ahead) for any serious calculation. If we were to omit the residual smoothing part of the full version we would obtain an HS version of MRZ [38], a Lanczos-based algorithm with look-ahead, and if the Lanczos *versus* HS comparisons reported on page 183 above are valid for look-ahead algorithms we would expect this version of QMR to outperform MRZ. However in the absence of comparative testing this

assessment must remain somewhat tentative.

The least successful algorithm tested was CGS. It failed on 6/12 tests but of these six failures, three were of type X and the tests would have been regarded as successes had the tolerances been less stringent or had the algorithm been allowed to restart (see Graphs 92, 105 and 122). These comparatively poor results should not be taken as a denigration of CGS. Its great contributions were to establish that transpose-free algorithms based on short recurrences were possible for nonsymmetric \mathbf{A} and, in so doing, to pave the way for BiCGStab.

Thus for non-symmetric systems we have four effective choices. For problems that are not excessively difficult we may choose between the version of QMR based on the coupled recurrences (QMRBiCG), GMRes(m) or BiCGStab. Of these, GMRes(m) and BiCGStab are transpose-free. None is clearly superior to the others and all have their adherents (and their weaknesses). For very difficult problems, one of LSQR, HG or BiCR could be considered but probably only as a last resort when all other methods fail. For more detailed comparisons of the performances of the first three methods on the twelve test problems see Graphs 77 - 88.

10.6.8. For symmetric systems

If \mathbf{A} is not only symmetric but positive definite the original CG algorithm is the algorithm of choice. Used with a good positive definite preconditioner it is both fast and stable, and is the outstanding Krylov method. For \mathbf{A} indefinite, matters are far less clear-cut. Methods that reduce the error norms (SymmLQ) or the residual norms (MinRes) take advantage of symmetry to generate short recurrences, but their performances vary considerably from problem to problem despite stagnation being restricted to one iteration at any one occurrence. If the eigenvalues of \mathbf{A} form distinct clusters an analysis along the lines of the proof of Theorem 21 indicates quite reasonable convergence, but examples have been found for which convergence is intolerably slow [52]. It would appear that the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ where \mathbf{A} is symmetric indefinite is far more difficult than when \mathbf{A} is definite, and possibly even more difficult than when \mathbf{A} is positive real.

Another disadvantage of SymmLQ and MinRes, as noted in [120], is that their nature only permits positive definite preconditioners. Since for indefinite problems the best preconditioners are likely to be indefinite this is a severe restriction which does not apply to the symmetric version of QMR. This algorithm is gaining in popularity and gives users an additional choice of weapon with which to tackle a deceptively innocent problem. It should certainly be considered when a solution of a symmetric indefinite system is required.

Chapter 11

Preconditioning

“X’s algorithms work very well provided that you have X on the end of a telephone line telling you how to choose the arbitrary constants.”

Anon.

As we have seen from the previous chapter, the basic forms of the Krylov methods are not particularly reliable as general-purpose linear equation solvers. Even the original method of Hestenes and Stiefel applied to symmetric positive definite systems can be surprisingly lethargic if the coefficient matrix \mathbf{A} is at all ill-conditioned. These practical observations are corroborated by theory. Inequality (4.42) shows that the bounds on the energy norm of the errors for CG are given directly in terms of the condition number of \mathbf{A} , with poorly-conditioned matrices giving the worst performance. It is not surprising, therefore, that after the first flush of enthusiasm for this new method had died down somewhat, attempts were made to accelerate the rate of convergence.

As the performance of the basic version of CG (with $\mathbf{K} = \mathbf{I}$) depends on the properties of the coefficient matrix an obvious way of proceeding is to try to transform the equations to take these properties into account, but we first need to know what properties lead to an efficient algorithm. As noted in the previous paragraph, if \mathbf{A} is symmetric and positive definite and we wish to use CG, then one strategy would be to try to reduce the condition number of the transformed matrix. If this could be achieved to the extent that the new condition number were in the low hundreds then the energy norm of the error would be reduced by at least a factor of 0.9 at each step, a rate of convergence that would be acceptable for most problems. However if such a transformation were not practicable another possibility would be to obtain a matrix that only had a few distinct eigenvalues. This would not give fast *convergence* but would guarantee rapid *termination*, with the number of iterations not exceeding the number of distinct eigenvalues.

These two approaches represent two opposing strategies, but a more reasonable scenario for a rapid solution would be a sort of linear combination of them both. This transformation would aim at an eigenvalue distribution where the majority of the eigenvalues were clustered within a small number of intervals on the real

line, preferably at some distance from the origin and from each other, and with a few outliers occurring outside the clusters. This would result in rapid convergence after a possibly slow start. For a formal justification of this claim an analogue of Theorem 21 can be proved for CG (see page 94 and the ensuing discussion).

Now laying down conditions that the transformed matrix should satisfy is one thing, implementing them in practice is another. In the case of a general large sparse symmetric matrix this may be difficult if not impossible. If, however, we note that in each case quoted above the transformed matrix is some approximation to the identity matrix (either in norm or rank) then this at least offers some suggestion as to how to proceed. We examine such possibilities later in the chapter but first we look at some general problems regarding preconditioning.

Suppose then that we wish to solve

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (11.1)$$

where \mathbf{A} is symmetric positive definite, and consider how the equation might be modified to improve the performance of CG. The simplest transformation is obtained by premultiplying the equation by some matrix, \mathbf{M}_L^{-1} say, so that

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}_L^{-1}\mathbf{b}, \quad (11.2)$$

a process known as *left-preconditioning*. Since we want $\mathbf{M}_L^{-1}\mathbf{A}$ to be as close to the identity matrix as possible we try to choose \mathbf{M}_L to be as nearly like \mathbf{A} as possible consistent with its inverse being easily computable. In particular, since we have assumed \mathbf{A} to be symmetric positive definite, we would like \mathbf{M}_L also to have this property but we would hope too that $\mathbf{M}_L^{-1}\mathbf{A}$ is symmetric positive definite since we could then solve equation (11.2) for \mathbf{x} using CG. Now the product of two symmetric matrices is itself symmetric if and only if the two matrices commute (see [155]) so if equation (11.2) is to be solvable by CG we must have $\mathbf{M}_L^{-1}\mathbf{A} = \mathbf{A}\mathbf{M}_L^{-1}$. One way of ensuring this, and the only reasonable way of ensuring it in practice, is to choose \mathbf{M}_L so that \mathbf{M}_L^{-1} is a polynomial in \mathbf{A} . This is one consideration that led to the early popularity of polynomial preconditioners.

Another method of preconditioning that uses ideas very similar to those just described is based on writing equation (11.1) as

$$\mathbf{A}\mathbf{M}_R^{-1}\mathbf{M}_R\mathbf{x} = \mathbf{b} \quad (11.3)$$

and is known as *right-preconditioning*. If $\hat{\mathbf{x}}$ is defined by

$$\hat{\mathbf{x}} = \mathbf{M}_R\mathbf{x}, \quad (11.4)$$

equation (11.3) becomes

$$\mathbf{A}\mathbf{M}_R^{-1}\hat{\mathbf{x}} = \mathbf{b} \quad (11.5)$$

and provided that $\mathbf{A}\mathbf{M}_R^{-1}$ is symmetric positive definite may be solved for $\hat{\mathbf{x}}$ using CG. The solution \mathbf{x} may then be recovered from $\hat{\mathbf{x}}$ by solving equation (11.4). We note too that if \mathbf{A} and \mathbf{M}_R^{-1} commute and if $\mathbf{M}_R = \mathbf{M}_L$ then equation (11.5) is

merely equation (11.2) with a different right-hand side. The rates of convergence obtained by applying CG to both equations should therefore be identical.

Given left and right preconditioning, the obvious next step is to combine the two to give *split-preconditioning* where

$$\mathbf{M}_L^{-1} \mathbf{A} \mathbf{M}_R^{-1} \hat{\mathbf{x}} = \mathbf{M}_L^{-1} \mathbf{b} \quad (11.6)$$

and $\hat{\mathbf{x}}$ is given by equation (11.4), and this trick yields a bonus. If \mathbf{A} is symmetric positive definite and $\mathbf{M}_R^{-1} = \mathbf{M}_L^{-T}$ and is nonsingular, $\mathbf{M}_L^{-1} \mathbf{A} \mathbf{M}_R^{-1}$ is also symmetric positive definite so that, regardless of the choice of \mathbf{M}_L^{-1} , equation (11.6) may be solved for $\hat{\mathbf{x}}$ by CG. No longer is the preconditioning matrix restricted to be a polynomial in \mathbf{A} . If the condition number of $\mathbf{M}_L^{-1} \mathbf{A} \mathbf{M}_R^{-1}$ is small then equation (11.6) may be solved efficiently and the true solution \mathbf{x} recovered from $\hat{\mathbf{x}}$ using equation (11.4).

We can now begin to consider how to choose \mathbf{M}_L and \mathbf{M}_R . Ideally we would like $\mathbf{M}_L^{-1} \mathbf{A} \mathbf{M}_R^{-1} = \mathbf{I}$, or $\mathbf{A} = \mathbf{M}_L \mathbf{M}_R$, as this would give one-step termination for CG. This suggests that we might choose \mathbf{M}_L and \mathbf{M}_R to be respectively lower and upper triangular and that they approximate in some way the lower and upper triangular factors of \mathbf{A} occurring in LU-decomposition. This idea has been investigated in [182], [183], [190], [215], [254] and is discussed further later in this chapter. Other choices are also possible, one being that \mathbf{M}_L is orthogonal with \mathbf{M}_R again being upper triangular. See [29], [160], [214] and [253].

Although the ideas of the previous two paragraphs have a certain attraction there are disadvantages, one being that a basic CG algorithm for solving equation (11.6) would generate not the sequences of true residuals $\{\mathbf{r}_i\}$ and approximate solutions $\{\mathbf{x}_i\}$ but the sequences $\{\hat{\mathbf{r}}_i\}$ and $\{\hat{\mathbf{x}}_i\}$ where, from equations (11.4) and (11.6),

$$\hat{\mathbf{r}}_i = \mathbf{M}_L^{-1} \mathbf{A} \mathbf{M}_R^{-1} \hat{\mathbf{x}}_i - \mathbf{M}_L^{-1} \mathbf{b} = \mathbf{M}_L^{-1} \mathbf{r}_i.$$

Since the convergence criterion will normally be specified in terms of $\|\mathbf{r}_i\|$ and since, from the above equation, $\|\mathbf{r}_i\| \leq \|\mathbf{M}_L\| \|\hat{\mathbf{r}}_i\|$, unless $\mathbf{M}_L = \mathbf{I}$ (right preconditioner only) or $\|\mathbf{r}_i\|$ is computed at each step it will be difficult to know when to terminate the iteration. For this and other reasons it is useful to construct an algorithm which solves equation (11.6) for $\hat{\mathbf{x}} = \mathbf{M}_R \mathbf{x}$ but which does so in terms of \mathbf{x}_i and \mathbf{r}_i . We now describe such an algorithm.

11.1. Galerkin methods

For greater generality we consider the solution of the block system $\mathbf{G}\mathbf{X} = \mathbf{H}$ where, as always, \mathbf{G} is assumed to be symmetric. After transformation this equation becomes

$$\mathbf{M}_S^{-1} \mathbf{G} \mathbf{M}_S^{-T} \mathbf{M}_S^T \mathbf{X} = \mathbf{M}_S^{-1} \mathbf{H} \quad (11.7)$$

where we have assumed that $\mathbf{M}_L^{-1} = \mathbf{M}_R^{-T}$ and have written \mathbf{M}_S^{-1} for both these matrices, choosing the subscript S to indicate split preconditioning. Equation (11.7) may now be written

$$\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}} \quad (11.8)$$

where

$$\widehat{\mathbf{G}} = \mathbf{M}_S^{-1} \mathbf{G} \mathbf{M}_S^{-T}, \quad (11.9)$$

$$\widehat{\mathbf{X}} = \mathbf{M}_S^T \mathbf{X} \quad (11.10)$$

and

$$\widehat{\mathbf{H}} = \mathbf{M}_S^{-1} \mathbf{H}, \quad (11.11)$$

and may be solved for $\widehat{\mathbf{X}}$ using the block CG algorithm. Since ideally we would like $\widehat{\mathbf{G}}$ to be the identity matrix, equation (11.9) implies that we should, as far as possible, choose \mathbf{M}_S to satisfy $\mathbf{M}_S \mathbf{M}_S^T = \mathbf{G}$.

Now the block CG algorithm for solving $\mathbf{G}\mathbf{X} = \mathbf{H}$ is given by equations (3.10), (3.14) and (3.19) - (3.21). If we denote by “hatted” symbols the quantities pertaining to and generated by the block CG algorithm when used to solve $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ we can obtain expressions for these simply by substituting the “hatted” symbols for the plain ones in equations (3.19) - (3.21), etc. This gives, if we define $\widehat{\mathbf{F}}_i$ by

$$\widehat{\mathbf{F}}_i = \widehat{\mathbf{G}}\widehat{\mathbf{X}}_i - \widehat{\mathbf{H}}, \quad (11.12)$$

the following:

$$\widehat{\mathbf{C}}_i = \widehat{\mathbf{F}}_i^T \widehat{\mathbf{K}} \widehat{\mathbf{F}}_i, \quad (11.13)$$

$$\widehat{\mathbf{D}}_i = \widehat{\mathbf{P}}_i^T \widehat{\mathbf{G}} \widehat{\mathbf{P}}_i, \quad (11.14)$$

$$\widehat{\mathbf{X}}_{i+1} = \widehat{\mathbf{X}}_i - \widehat{\mathbf{P}}_i \widehat{\mathbf{D}}_i^{-1} \widehat{\mathbf{C}}_i, \quad (11.15)$$

$$\widehat{\mathbf{F}}_{i+1} = \widehat{\mathbf{F}}_i - \widehat{\mathbf{G}} \widehat{\mathbf{P}}_i \widehat{\mathbf{D}}_i^{-1} \widehat{\mathbf{C}}_i \quad (11.16)$$

and

$$\widehat{\mathbf{P}}_{i+1} = \widehat{\mathbf{K}} \widehat{\mathbf{F}}_{i+1} + \widehat{\mathbf{P}}_i \widehat{\mathbf{C}}_i^{-1} \widehat{\mathbf{C}}_{i+1}. \quad (11.17)$$

We can now prove the following theorem.

Theorem 30. Let $\mathbf{G} \in \mathbb{R}^{n \times n}$ be symmetric and nonsingular and let $\mathbf{H} \in \mathbb{R}^{n \times m}$, $m < n$. Let $\widehat{\mathbf{G}}$, $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{H}}$ be defined by equations (11.9) - (11.11), where \mathbf{M}_S is assumed to be nonsingular. If $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ is solved by the block CG method as defined by equations (11.12) - (11.17) for some symmetric matrix $\widehat{\mathbf{K}}$ starting with $\widehat{\mathbf{X}}_1$, and the matrices \mathbf{X}_i , \mathbf{F}_i and \mathbf{P}_i are defined by

$$\mathbf{X}_i = \mathbf{M}_S^{-T} \widehat{\mathbf{X}}_i, \quad (11.18)$$

$$\mathbf{F}_i = \mathbf{G}\mathbf{X}_i - \mathbf{H}, \quad (11.19)$$

and

$$\mathbf{P}_i = \mathbf{M}_S^{-T} \widehat{\mathbf{P}}_i, \quad (11.20)$$

then these matrices are precisely the matrices \mathbf{X}_i , \mathbf{F}_i and \mathbf{P}_i that would be obtained by solving $\mathbf{G}\mathbf{X} = \mathbf{H}$ by the block CG algorithm with

$$\mathbf{K} = \mathbf{M}_S^{-T} \widehat{\mathbf{K}} \mathbf{M}_S^{-1} \quad (11.21)$$

and starting with $\mathbf{X}_1 = \mathbf{M}_S^{-T} \widehat{\mathbf{X}}_1$.

Proof. From equations (11.9), (11.14) and (11.20),

$$\widehat{\mathbf{D}}_i = \widehat{\mathbf{P}}_i^T \widehat{\mathbf{G}} \widehat{\mathbf{P}}_i = \mathbf{P}_i^T \mathbf{G} \mathbf{P}_i$$

so that, from equation (3.10), $\widehat{\mathbf{D}}_i = \mathbf{D}_i$. From equations (11.9), (11.11) and (11.12),

$$\widehat{\mathbf{F}}_i = \mathbf{M}_S^{-1} (\mathbf{G} \mathbf{M}_S^{-T} \widehat{\mathbf{X}}_i - \mathbf{H})$$

so that from equations (11.18) and (11.19),

$$\widehat{\mathbf{F}}_i = \mathbf{M}_S^{-1} \mathbf{F}_i. \quad (11.22)$$

Hence, if we define \mathbf{K} by equation (11.21) we have, from equations (11.13) and (11.22),

$$\widehat{\mathbf{C}}_i = \widehat{\mathbf{F}}_i^T \widehat{\mathbf{K}} \widehat{\mathbf{F}}_i = \mathbf{F}_i^T \mathbf{K} \mathbf{F}_i$$

so that, from equation (3.14), $\widehat{\mathbf{C}}_i = \mathbf{C}_i$.

We now premultiply equation (11.15) by \mathbf{M}_S^{-T} . This gives, from equations (11.18) and (11.20) and since $\widehat{\mathbf{C}}_i = \mathbf{C}_i$ and $\widehat{\mathbf{D}}_i = \mathbf{D}_i$,

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{C}_i. \quad (11.23)$$

Premultiplying equation (11.16) by \mathbf{M}_S similarly gives, from equations (11.9), (11.20) and (11.22),

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G} \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{C}_i \quad (11.24)$$

and finally premultiplying equation (11.17) by \mathbf{M}_S^{-T} gives, from equations (11.21) - (11.22),

$$\mathbf{P}_{i+1} = \mathbf{K} \mathbf{F}_{i+1} + \mathbf{P}_i \mathbf{C}_i^{-1} \mathbf{C}_{i+1}. \quad (11.25)$$

The theorem now follows since equations (11.23) - (11.25) are identical to equations (3.19) - (3.21). ■

Corollary 31. If $\widehat{\mathbf{K}} = \mathbf{I}$ then $\mathbf{K} = \mathbf{M}^{-1}$ where \mathbf{M} is defined by

$$\mathbf{M} = \mathbf{M}_S \mathbf{M}_S^T. \quad (11.26)$$

Proof. Follows immediately from equation (11.21). ■

The matrix \mathbf{M} is normally referred to as the *preconditioning matrix* or *preconditioner*, and this theorem and its corollary establishes a relationship between \mathbf{M}

and \mathbf{K} . In particular, if $\widehat{\mathbf{K}} = \mathbf{I}$, $\mathbf{K} = \mathbf{M}^{-1}$ and may equally be regarded as a preconditioner.

Theorem 30 and its corollary state that the same sequence of approximate solutions $\{\mathbf{X}_i\}$ is generated either by applying the unpreconditioned CG method ($\widehat{\mathbf{K}} = \mathbf{I}$) to the equations $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$, and recovering \mathbf{X}_i from $\widehat{\mathbf{X}}_i$ by using equation (11.10), or from applying the preconditioned form of the algorithm to the original equation $\mathbf{G}\mathbf{X} = \mathbf{H}$ with

$$\mathbf{K} = (\mathbf{M}_S \mathbf{M}_S^T)^{-1}. \quad (11.27)$$

Thus from the point of view of convergence, in exact arithmetic the two algorithms give identical results. Now the theory of Chapter 4 (above) indicates that the convergence of the first of these methods depends on the spectrum of $\widehat{\mathbf{G}}$ whereas that of the second depends on the spectrum of $\mathbf{KG} = (\mathbf{M}_S \mathbf{M}_S^T)^{-1}\mathbf{G}$. However since from equation (11.9) $\widehat{\mathbf{G}}$ is similar to \mathbf{KG} , the spectra are identical and there is no conflict.

We note that if $\widehat{\mathbf{K}} = \mathbf{I}$, preconditioners derived from equation (11.7) are forced by equation (11.27) to be positive definite and although this may be desirable on grounds of stability it does somewhat restrict their choice. The ideal preconditioner would satisfy $\mathbf{G} = \mathbf{M}_S \mathbf{M}_S^T$ which implies that, for the most effective preconditioning, \mathbf{G} too should be positive definite and that, from equation (11.26), \mathbf{K} should resemble \mathbf{G}^{-1} as far as possible. It also suggests that if \mathbf{G} is positive definite then \mathbf{M}_S could be lower triangular and approximate the Cholesky factor \mathbf{L} of \mathbf{G} . The implications of these ideas have been examined in ([182], [183] and [170]) and are discussed below. If, on the other hand, \mathbf{G} is symmetric indefinite then either it is possible to have a semi-stable algorithm (\mathbf{K} positive definite) and give up the idea of good preconditioning, or aim for good preconditioning at the expense of stability. In this case \mathbf{K} would be chosen to resemble \mathbf{G}^{-1} as far as possible. That this second strategy need not be unreasonably optimistic is indicated by the success of PBiCG, either with or without residual smoothing.

A similar theorem to Theorem 30 holds for the Lanczos methods and may be proved by replacing equation (11.17) by

$$\widehat{\mathbf{P}}_{i+1} = \left(\mathbf{I} - \widehat{\mathbf{P}}_i \widehat{\mathbf{D}}_i^{-1} \widehat{\mathbf{P}}_i^T \widehat{\mathbf{G}} \right) \widehat{\mathbf{K}} \widehat{\mathbf{G}} \widehat{\mathbf{P}}_i - \widehat{\mathbf{P}}_{i-1} \widehat{\mathbf{D}}_{i-1}^{-1} \widehat{\mathbf{D}}_i$$

(the “hatted” version of equation (3.11)) in the proof of that theorem. Just as for the HS algorithms, the derivation of preconditioning from equations (11.7) - (11.11) is unduly restrictive if $\widehat{\mathbf{K}} = \mathbf{I}$ since again from equation (11.27) the preconditioners are always positive definite. Now from Theorems 16 and 17, the impact of an indefinite preconditioner on a Lanczos algorithm is considerably less disruptive than it can be for an HS one. Despite this, though, the Lanczos algorithms are still not that popular, even for symmetric indefinite systems where the Lanczos version of the CR algorithm (CRL or MCR) has all the theoretical advantages enjoyed by the symmetric version of QMR except look-ahead. It could be that for CRL these advantages are insufficient to overcome the observed poor performance of Lanczos algorithms compared with their HS counterparts.

For algorithms like SymmLQ and MinRes that are based on the original Lanczos algorithm (one that generates orthonormal vectors) the above comments do not apply. These algorithms cannot crash (since $\mathbf{G} = \mathbf{I}$) and any stagnation lasts for at most one iteration (this follows from Theorem 2.2 of [136]). But since \mathbf{G} is forced to be the identity it follows that the solution of $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is symmetric requires \mathbf{A} to be somehow included in \mathbf{K} . In fact $\mathbf{K} = \mathbf{A}$ for the unpreconditioned versions of both SymmLQ and MinRes. However since \mathbf{G} is sacrosanct, any preconditioning must also be introduced via \mathbf{K} . Now in order to guarantee a short recurrence both \mathbf{G} and \mathbf{K} should be symmetric and this effectively forces any preconditioning to be implemented via equation (11.7). This in turn implies that the preconditioner will be positive definite even if \mathbf{A} is indefinite, and this is seen as a weakness of algorithms that use the original Lanczos method to generate short recurrences¹².

We can now obtain the preconditioned version of CG (PCG - where here, and later, “P” denotes “preconditioned”) for solving $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is symmetric and positive definite. The relevant expressions may be obtained by substituting \mathbf{A} for \mathbf{G} and \mathbf{r}_i for \mathbf{f}_i in the vector forms of equations (11.23) - (11.25), giving

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_i - \mathbf{p}_i \left(\frac{\mathbf{r}_i^T \mathbf{K} \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \right) \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \mathbf{A} \mathbf{p}_i \left(\frac{\mathbf{r}_i^T \mathbf{K} \mathbf{r}_i}{\mathbf{p}_i^T \mathbf{A} \mathbf{p}_i} \right)\end{aligned}\tag{11.28}$$

and

$$\mathbf{p}_{i+1} = \mathbf{K} \mathbf{r}_{i+1} + \mathbf{p}_i \left(\frac{\mathbf{r}_{i+1}^T \mathbf{K} \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{K} \mathbf{r}_i} \right)$$

(c.f. equations (3.28) and (3.30)).

If \mathbf{A} is not symmetric and we have to solve the compound equations

$$\begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix}$$

we precondition the equation $\mathbf{Ax} = \mathbf{b}$ as in equation (11.6), and this suggests that the shadow system of equations $\mathbf{A}^T \mathbf{z} = \mathbf{c}$ be preconditioned by

$$\mathbf{M}_R^{-T} \mathbf{A}^T \mathbf{M}_L^{-T} \mathbf{M}_L^T \mathbf{z} = \mathbf{M}_R^{-T} \mathbf{c}. \tag{11.29}$$

We now define \mathbf{X} as before (equation (3.25)) by

$$\mathbf{X} = \begin{bmatrix} \mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathbf{z} \end{bmatrix},$$

\mathbf{G} and \mathbf{H} by

$$\mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{O} \end{bmatrix} \tag{11.30}$$

¹²But not for GMRes which generates long recurrences.

and

$$\mathbf{H} = \begin{bmatrix} \mathbf{0} & \mathbf{c} \\ \mathbf{b} & \mathbf{0} \end{bmatrix} \quad (11.31)$$

(these are the values of \mathbf{G} and \mathbf{H} appropriate both to the BiCG and HG algorithms, see pages 52 and 54), and \mathbf{M}_S by

$$\mathbf{M}_S = \begin{bmatrix} \mathbf{M}_R^T & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_L \end{bmatrix}. \quad (11.32)$$

Substituting these values of \mathbf{X} , \mathbf{G} , \mathbf{H} , and \mathbf{M}_S in equation (11.7) then gives $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ where, from equations (11.9) - (11.11),

$$\widehat{\mathbf{G}} = \begin{bmatrix} \mathbf{0} & \mathbf{M}_R^{-T} \mathbf{A}^T \mathbf{M}_L^{-T} \\ \mathbf{M}_L^{-1} \mathbf{A} \mathbf{M}_R^{-1} & \mathbf{0} \end{bmatrix}, \quad (11.33)$$

$$\widehat{\mathbf{X}} = \begin{bmatrix} \mathbf{M}_R \mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_L^T \mathbf{z} \end{bmatrix} \quad (11.34)$$

and

$$\widehat{\mathbf{H}} = \begin{bmatrix} \mathbf{0} & \mathbf{M}_R^{-T} \mathbf{c} \\ \mathbf{M}_L^{-1} \mathbf{b} & \mathbf{0} \end{bmatrix}, \quad (11.35)$$

and writing the individual partitions of $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ then yields both equation (11.6) and equation (11.29).

Thus for compound equations where the matrix is given by equation (11.30) and where preconditioning is applied directly to the equation as it is in equation (11.7), it is not possible to find a preconditioner such that the preconditioned coefficient matrix is the identity. No matter how \mathbf{M}_S is chosen, as long as it is real then half the eigenvalues of $\mathbf{M}_S^{-1} \mathbf{G} \mathbf{M}_S^{-T}$ will be negative. This occurs because \mathbf{G} has Young's "property A" (see e.g. [259]) and its eigenvalues (all nonzero since \mathbf{G} is assumed to be nonsingular) consequently occur in pairs $\pm\lambda$, and from the fact that the transformation $\mathbf{M}_S^{-1} \mathbf{G} \mathbf{M}_S^{-T}$ preserves the inertia of the matrix (see e.g. [125], page 296 or [155], page 223).

We can now use Theorem 30 to obtain the preconditioned forms of BiCG and HG by first appealing to equation (11.21) to give the preconditioning matrices \mathbf{K}_{BiCG} and \mathbf{K}_{HG} . If we solve $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ using the block CG algorithm defined by equations (11.13) - (11.18), the preconditioners $\widehat{\mathbf{K}}_{BiCG}$ and $\widehat{\mathbf{K}}_{HG}$ appropriate to the two algorithms are given on pages 52 and 54 and are

$$\widehat{\mathbf{K}}_{BiCG} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \quad \text{and} \quad \widehat{\mathbf{K}}_{HG} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}.$$

It then follows from equations (11.21) and (11.32) that

$$\mathbf{K}_{BiCG} = \begin{bmatrix} \mathbf{0} & \mathbf{K}_H \\ \mathbf{K}_H^T & \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{K}_{HG} = \begin{bmatrix} \mathbf{K}_R & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_L \end{bmatrix}, \quad (11.36)$$

where

$$\mathbf{K}_H = (\mathbf{M}_L \mathbf{M}_R)^{-1}, \quad (11.37)$$

$$\mathbf{K}_R = (\mathbf{M}_R^T \mathbf{M}_R)^{-1} \quad \text{and} \quad \mathbf{K}_L = (\mathbf{M}_L \mathbf{M}_L^T)^{-1}. \quad (11.38)$$

We note here that for BiCG the matrix \mathbf{K} depends only upon the product $\mathbf{M}_L \mathbf{M}_R$. Provided this is constant, \mathbf{M}_L and \mathbf{M}_R may be chosen arbitrarily and the algorithm will generate the same sequence of approximate solutions $\{\mathbf{X}_i\}$. Thus the distinction between left- and right-preconditioning is, at least for this algorithm, somewhat meaningless in the context of exact arithmetic. This is not the case for HG. If \mathbf{M}_L and \mathbf{M}_R change, even if the product $\mathbf{M}_L \mathbf{M}_R$ remains constant, \mathbf{K}_{HG} changes and this will (presumably) affect the performance of the algorithm. A simple way of changing \mathbf{K}_{HG} while keeping $\mathbf{M}_L \mathbf{M}_R$ constant would be, from equations (11.36) and (11.38), to replace the identity by

$$\mathbf{K} = \begin{bmatrix} \alpha \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \alpha^{-1} \mathbf{I} \end{bmatrix}$$

for some scalar $\alpha \neq 0$, but given the generally poor showing of the unpreconditioned Krylov methods a more sophisticated choice would seem to be necessary to make the algorithm competitive at the highest level.

Although the existing theory is not too helpful in the case of indefinite or non-symmetric matrices, it does indicate the importance of the matrix \mathbf{KG} , the matrix that generates the Krylov sequence, for the analysis of convergence. For BiCG and HG this matrix is given by

$$\mathbf{K}_{BiCG} \mathbf{G} = \begin{bmatrix} \mathbf{K}_H \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{K}_H^T \mathbf{A}^T \end{bmatrix} \quad (11.39)$$

and

$$\mathbf{K}_{HG} \mathbf{G} = \begin{bmatrix} \mathbf{O} & \mathbf{K}_R \mathbf{A}^T \\ \mathbf{K}_L \mathbf{A} & \mathbf{O} \end{bmatrix}.$$

Thus for BiCG, \mathbf{KG} is block diagonal and not necessarily symmetric so that it is quite possible for this algorithm that the Krylov sequence be generated by a matrix having complex eigenvalues. On the other hand, it is possible to find a preconditioner such that \mathbf{KG} is the unit matrix and indeed this occurs, from equations (11.37) and (11.39), if $\mathbf{A} = \mathbf{M}_L \mathbf{M}_R$, the ideal preconditioner.

For HG, in contrast, even though \mathbf{KG} is not generally symmetric unless $\mathbf{K}_R = \mathbf{K}_L = \mathbf{I}$, the eigenvalues of \mathbf{KG} are always real. To prove this note that for HG, the eigenvalues of $(\mathbf{KG})^2$ are those of $\mathbf{K}_R \mathbf{A}^T \mathbf{K}_L \mathbf{A}$. Now both \mathbf{K}_L and \mathbf{K}_R are symmetric positive definite so that $\mathbf{K}_R \mathbf{A}^T \mathbf{K}_L \mathbf{A}$ is similar to $\mathbf{B}^T \mathbf{B}$ where $\mathbf{B} = \mathbf{K}_L^{\frac{1}{2}} \mathbf{A}^T \mathbf{K}_R^{\frac{1}{2}}$, and since the eigenvalues of $\mathbf{B}^T \mathbf{B}$ are always real and positive, those of $(\mathbf{KG})^2$ are also real and positive so that those of \mathbf{KG} are real. But if the eigenvalues of the ‘‘Krylov matrix’’ are real, the downside is that it is never possible for \mathbf{KG} to be equal to the unit matrix. Not only is $\mathbf{K}_{HG} \mathbf{G}$ skew rather than normal block diagonal

but $\mathbf{K}_L \mathbf{A}$ and $\mathbf{K}_R \mathbf{A}^T$ can never equal the identity since, from equation (11.38), both \mathbf{K}_L and \mathbf{K}_R are symmetric positive definite whereas \mathbf{A} is non-symmetric. However matters are not quite as black as they seem. If we choose \mathbf{M}_L and \mathbf{M}_R so that $\mathbf{A} = \mathbf{M}_L \mathbf{M}_R$, and \mathbf{K}_L and \mathbf{K}_R are given by equation (11.38), then $(\mathbf{K}_{HG} \mathbf{G})^2 = \mathbf{I}$. This is probably the best we can do, but this quirk might explain the relatively poor showing of HG in our comparative testing (see Chapter 10 below).

We should perhaps note here that in the original presentation of his algorithms, Hegedüs [149], [151] gave the full preconditioned (and also scaled) versions of his equations rather than the simplified forms outlined in Chapter 3 (above). He required the matrices \mathbf{K}_L and \mathbf{K}_R to be symmetric positive definite, noted that they “may serve as preconditioners” and effectively suggested that they be computed from equation (11.38). The matrices \mathbf{M}_L and \mathbf{M}_R would satisfy approximately $\mathbf{M}_L \mathbf{M}_R = \mathbf{A}$ and be determined by incomplete factorisation (see below). However there is insufficient material in the published literature to enable a critical assessment of this technique to be attempted.

With the values of the matrices \mathbf{K}_{BiCG} and \mathbf{K}_{HG} now being available from equation (11.36) we can give the formulæ for the preconditioned forms of both BiCG and HG by substituting the appropriate values in equations (11.23) - (11.25). For PBiCG we obtain

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{s}_i^T \mathbf{K}_H \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \mathbf{A} \mathbf{u}_i \left(\frac{\mathbf{s}_i^T \mathbf{K}_H \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \\ \mathbf{s}_{i+1} &= \mathbf{s}_i - \mathbf{A}^T \mathbf{v}_i \left(\frac{\mathbf{s}_i^T \mathbf{K}_H \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \\ \mathbf{u}_{i+1} &= \mathbf{K}_H \mathbf{r}_{i+1} + \mathbf{u}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{K}_H \mathbf{r}_{i+1}}{\mathbf{s}_i^T \mathbf{K}_H \mathbf{r}_i} \right) \\ \mathbf{v}_{i+1} &= \mathbf{K}_H^T \mathbf{s}_{i+1} + \mathbf{v}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{K}_H \mathbf{r}_{i+1}}{\mathbf{s}_i^T \mathbf{K}_H \mathbf{r}_i} \right).\end{aligned}\tag{11.40}$$

while for PHG we have

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_i - \mathbf{u}_i \left(\frac{\mathbf{r}_i^T \mathbf{K}_L \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \mathbf{A} \mathbf{u}_i \left(\frac{\mathbf{r}_i^T \mathbf{K}_L \mathbf{r}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \\ \mathbf{s}_{i+1} &= \mathbf{s}_i - \mathbf{A}^T \mathbf{v}_i \left(\frac{\mathbf{s}_i^T \mathbf{K}_R \mathbf{s}_i}{\mathbf{v}_i^T \mathbf{A} \mathbf{u}_i} \right) \\ \mathbf{u}_{i+1} &= \mathbf{K}_R \mathbf{s}_{i+1} + \mathbf{u}_i \left(\frac{\mathbf{s}_{i+1}^T \mathbf{K}_R \mathbf{s}_{i+1}}{\mathbf{s}_i^T \mathbf{K}_R \mathbf{s}_i} \right)\end{aligned}\tag{11.41}$$

$$\mathbf{v}_{i+1} = \mathbf{K}_L \mathbf{r}_{i+1} + \mathbf{v}_i \left(\frac{\mathbf{r}_{i+1}^T \mathbf{K}_L \mathbf{r}_{i+1}}{\mathbf{r}_i^T \mathbf{K}_L \mathbf{r}_i} \right).$$

Ways of choosing \mathbf{K}_H for PBiCG are discussed below.

11.2. Minimum residual methods*

In our derivation of preconditioning techniques for Galerkin algorithms we replaced \mathbf{G} by $\widehat{\mathbf{G}} = \mathbf{M}_S^{-1} \mathbf{G} \mathbf{M}_S^{-T}$ but assigned $\widehat{\mathbf{K}}$ to be the values of \mathbf{K} given in the description of each individual method (see Chapter 3 above). This device does not work for minimum residual algorithms so instead we define $\widehat{\mathbf{G}}$ as before (equation (11.9)) and solve $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ by the CR method, that is we solve equations

$$\widetilde{\mathbf{G}}\widetilde{\mathbf{X}} = \widetilde{\mathbf{H}}, \quad (11.42)$$

where $\widetilde{\mathbf{G}} = \widehat{\mathbf{G}}^2$ and $\widetilde{\mathbf{H}} = \widehat{\mathbf{G}}\widehat{\mathbf{H}}$, by an algorithm defined by equations (11.12) - (11.17), but with the hats replaced by tildes (distinguish equations thus modified by appending a “T” to their equation numbers) and with $\widetilde{\mathbf{K}} = \widehat{\mathbf{G}}^{-1}$. A possible disadvantage of this approach, if the algorithm is to be used to solve a least-squares problem, is that it changes the residual norm being minimised but this should not matter unduly if \mathbf{G} is square and nonsingular and a zero-norm solution is sought. We now show that this algorithm too is equivalent to CG applied to the original equation $\mathbf{G}\mathbf{X} = \mathbf{H}$ but with \mathbf{C}_i and \mathbf{D}_i now being defined by equations (11.44) and (11.45).

Theorem 32. Let \mathbf{G} , \mathbf{H} , $\widehat{\mathbf{G}}$, $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{H}}$ be as defined in Theorem 30. If equation (11.42) is solved by the block CG method as defined by equations (11.12T) - (11.17T) with $\widetilde{\mathbf{K}} = \widehat{\mathbf{G}}^{-1}$ and starting with $\widetilde{\mathbf{X}}_1$, and the matrices \mathbf{X}_i , \mathbf{F}_i and \mathbf{P}_i are defined by equations (11.18T), (11.19) and (11.20T), then these matrices are precisely the matrices \mathbf{X}_i , \mathbf{F}_i and \mathbf{P}_i that would be computed by solving $\mathbf{G}\mathbf{X} = \mathbf{H}$ by the block CG algorithm with

$$\mathbf{K} = (\mathbf{M}_S \mathbf{M}_S^T)^{-1}, \quad (11.43)$$

starting with $\mathbf{X}_1 = \mathbf{M}_S^{-T} \widetilde{\mathbf{X}}_1$, but with \mathbf{C}_i and \mathbf{D}_i now given by

$$\mathbf{C}_i = \mathbf{F}_i^T \mathbf{K} \mathbf{G} \mathbf{K} \mathbf{F}_i \quad (11.44)$$

and

$$\mathbf{D}_i = \mathbf{P}_i^T \mathbf{G} \mathbf{K} \mathbf{G} \mathbf{P}_i. \quad (11.45)$$

Proof. Since $\widetilde{\mathbf{G}} = \widehat{\mathbf{G}}^2$ and $\widetilde{\mathbf{H}} = \widehat{\mathbf{G}}\widehat{\mathbf{H}}$, equations (11.9), (11.11) and (11.43) yield

$$\widetilde{\mathbf{G}} = \mathbf{M}_S^{-1} \mathbf{G} \mathbf{K} \mathbf{G} \mathbf{M}_S^{-T} \quad (11.46)$$

and

$$\widetilde{\mathbf{H}} = \mathbf{M}_S^{-1} \mathbf{G} \mathbf{K} \mathbf{H}$$

so that, from equation (11.12T)

$$\tilde{\mathbf{F}}_i = \mathbf{M}_S^{-1} \mathbf{GK} \left(\mathbf{GM}_S^{-T} \tilde{\mathbf{X}}_i - \mathbf{H} \right).$$

Hence, from equations (11.18T) and (11.19),

$$\tilde{\mathbf{F}}_i = \mathbf{M}_S^{-1} \mathbf{GKF}_i. \quad (11.47)$$

Since $\tilde{\mathbf{G}} = \hat{\mathbf{G}}^2$, equations (11.14T) and (11.46) yield

$$\tilde{\mathbf{D}}_i = \tilde{\mathbf{P}}_i^T \mathbf{M}_S^{-1} \mathbf{GKGM}_S^{-T} \tilde{\mathbf{P}}_i.$$

so that, from equations (11.20T) and (11.45),

$$\tilde{\mathbf{D}}_i = \mathbf{P}_i^T \mathbf{GKGP}_i = \mathbf{D}_i. \quad (11.48)$$

Since for CR, $\tilde{\mathbf{K}} = \hat{\mathbf{G}}^{-1}$ by definition, equation (11.9) gives

$$\tilde{\mathbf{K}} = \mathbf{M}_S^T \mathbf{G}^{-1} \mathbf{M}_S. \quad (11.49)$$

Equation (11.13T) gives $\tilde{\mathbf{C}}_i = \tilde{\mathbf{F}}_i^T \tilde{\mathbf{K}} \tilde{\mathbf{F}}_i$ so that, from equations (11.44), (11.47) and (11.49),

$$\tilde{\mathbf{C}}_i = \mathbf{F}_i^T \mathbf{KGKF}_i = \mathbf{C}_i. \quad (11.50)$$

Premultiplying now equation (11.15T) by \mathbf{M}_S^{-T} then gives, from equations (11.18T), (11.20T), (11.48) and (11.50),

$$\mathbf{X}_{i+1} = \mathbf{X}_i - \mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{C}_i. \quad (11.51)$$

Similarly premultiplying equation (11.16T) by \mathbf{M}_S gives, from equations (11.20T), (11.46), (11.47) and the assumed nonsingularity of \mathbf{G} and \mathbf{K} ,

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{GP}_i \mathbf{D}_i^{-1} \mathbf{C}_i. \quad (11.52)$$

Finally, premultiplying equation (11.17T) by \mathbf{M}_S^{-T} gives, from equations (11.20T) and (11.49),

$$\mathbf{P}_{i+1} = \mathbf{G}^{-1} \mathbf{M}_S \tilde{\mathbf{F}}_{i+1} + \mathbf{P}_i \mathbf{C}_i^{-1} \mathbf{C}_{i+1}$$

which reduces, from equation (11.47), to

$$\mathbf{P}_{i+1} = \mathbf{KF}_{i+1} + \mathbf{P}_i \mathbf{C}_i^{-1} \mathbf{C}_{i+1}. \quad (11.53)$$

The theorem now follows since equations (11.51) - (11.53) are identical to equations (3.19) - (3.21). ■

This theorem indicates that if the equation $\hat{\mathbf{G}}\hat{\mathbf{X}} = \hat{\mathbf{H}}$, where $\hat{\mathbf{G}}$, $\hat{\mathbf{X}}$ and $\hat{\mathbf{H}}$ are defined by equations (11.33) - (11.35), is solved by the basic CR algorithm, i.e. if $\tilde{\mathbf{G}}\tilde{\mathbf{X}} = \tilde{\mathbf{H}}$ is solved by the basic block CG algorithm as defined by equations (11.12T) - (11.17T) with $\tilde{\mathbf{G}} = \hat{\mathbf{G}}^2$, $\tilde{\mathbf{H}} = \hat{\mathbf{G}}\hat{\mathbf{H}}$ and $\tilde{\mathbf{K}} = \hat{\mathbf{G}}^{-1}$, then the sequence of

approximate solutions $\{\tilde{\mathbf{X}}_i\}$ so obtained is simply related, by equation (11.18T), to the sequence of approximate solutions $\{\mathbf{X}_i\}$ obtained by solving $\mathbf{G}\mathbf{X} = \mathbf{H}$ by the basic block CG algorithm with $\mathbf{K} = (\mathbf{M}_S \mathbf{M}_S^T)^{-1}$ but with \mathbf{C}_i and \mathbf{D}_i now defined by equations (11.44) and (11.45). We can thus, at least in exact arithmetic, obtain all the benefits of solving $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ (as opposed to solving $\mathbf{G}\mathbf{X} = \mathbf{H}$) by CR merely by solving $\mathbf{G}\mathbf{X} = \mathbf{H}$ by a slightly modified version of the basic block CG algorithm and appropriately choosing \mathbf{K} .

Theorem 30 is even less restrictive. It tells us that we can obtain all the benefits of solving $\widehat{\mathbf{G}}\widehat{\mathbf{X}} = \widehat{\mathbf{H}}$ by a Galerkin algorithm merely by solving $\mathbf{G}\mathbf{X} = \mathbf{H}$ by the basic block CG algorithm with the appropriate choice of \mathbf{K} . However the choice of Galerkin algorithms is wider than that of CR algorithms since in Theorem 30 \mathbf{K} is given by $\mathbf{K} = \mathbf{M}_S^{-T} \widehat{\mathbf{K}} \mathbf{M}_S^{-1}$ where $\widehat{\mathbf{K}}$ is any symmetric matrix but for Theorem 32, \mathbf{K} is restricted to be equal to $(\mathbf{M}_S \mathbf{M}_S^T)^{-1}$. It is this extra freedom that permits us to obtain a preconditioned version of BiCG for which there is no CR equivalent.

Now Theorems 30 and 32 imply that if the same value of \mathbf{K} is chosen, then since the sequences $\{\widehat{\mathbf{X}}_i\}$ and $\{\tilde{\mathbf{X}}_i\}$ are similarly related to $\{\mathbf{X}_i\}$ (by equations (11.18) and (11.18T) respectively) we can switch from a Galerkin algorithm to a CR one simply by changing the way we compute \mathbf{C}_i and \mathbf{D}_i , an attribute previously noted by Rutishauser ([213]). However the restriction on \mathbf{K} implies that if we try to solve $\mathbf{Ax} = \mathbf{b}$ for nonsymmetric \mathbf{A} by solving the compound system by CR then only the “Hagedüs” values of \mathbf{K} are available. The matrix $\widetilde{\mathbf{K}}\widetilde{\mathbf{G}}$ that generates the Krylov sequence is, since $\widetilde{\mathbf{G}} = \widehat{\mathbf{G}}^2$ and $\widetilde{\mathbf{K}} = \widehat{\mathbf{G}}^{-1}$, simply the matrix $\widehat{\mathbf{G}}$. This is defined by equation (11.33) and is, by its very nature, skew block-diagonal. The above comments regarding the convergence of HG thus apply.

We can now obtain the formulæ for PCR and PBiCR simply by substituting the appropriate values in equations (3.19) - (3.21). For solving $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} is symmetric, if we define $\bar{\mathbf{r}}_i = \mathbf{K}\mathbf{r}_i$ we obtain, for PCR,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{p}_i \left(\frac{\bar{\mathbf{r}}_i^T \mathbf{A} \bar{\mathbf{r}}_i}{\mathbf{q}_i^T \mathbf{K} \mathbf{q}_i} \right) \quad (11.54)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{q}_i \left(\frac{\bar{\mathbf{r}}_i^T \mathbf{A} \bar{\mathbf{r}}_i}{\mathbf{q}_i^T \mathbf{K} \mathbf{q}_i} \right)$$

$$\mathbf{p}_{i+1} = \bar{\mathbf{r}}_{i+1} + \mathbf{p}_i \left(\frac{\bar{\mathbf{r}}_{i+1}^T \mathbf{A} \bar{\mathbf{r}}_{i+1}}{\bar{\mathbf{r}}_i^T \mathbf{A} \bar{\mathbf{r}}_i} \right)$$

$$\mathbf{q}_{i+1} = \mathbf{A} \bar{\mathbf{r}}_{i+1} + \mathbf{q}_i \left(\frac{\bar{\mathbf{r}}_{i+1}^T \mathbf{A} \bar{\mathbf{r}}_{i+1}}{\bar{\mathbf{r}}_i^T \mathbf{A} \bar{\mathbf{r}}_i} \right)$$

Similarly, for the compound system for non-symmetric \mathbf{A} , we define $\bar{\mathbf{r}}_i$ and $\bar{\mathbf{s}}_i$ by $\bar{\mathbf{r}}_i = \mathbf{K}_L \mathbf{r}_i$ and $\bar{\mathbf{s}}_i = \mathbf{K}_R \mathbf{s}_i$. Making the appropriate substitution then gives, for PBiCR,

$$\mathbf{w}_i = \mathbf{A} \mathbf{u}_i$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \left(\frac{\bar{\mathbf{s}}_i^T \mathbf{A}^T \bar{\mathbf{r}}_i}{\mathbf{w}_i^T \mathbf{K}_L \mathbf{w}_i} \right) \quad (11.55)$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \left(\frac{\bar{\mathbf{s}}_i^T \mathbf{A}^T \bar{\mathbf{r}}_i}{\mathbf{w}_i^T \mathbf{K}_L \mathbf{w}_i} \right)$$

$$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \left(\frac{\bar{\mathbf{s}}_i^T \mathbf{A}^T \bar{\mathbf{r}}_i}{\mathbf{y}_i^T \mathbf{K}_R \mathbf{y}_i} \right)$$

$$\mathbf{u}_{i+1} = \bar{\mathbf{s}}_{i+1} + \mathbf{u}_i \left(\frac{\bar{\mathbf{s}}_{i+1}^T \mathbf{A}^T \bar{\mathbf{r}}_{i+1}}{\bar{\mathbf{s}}_i^T \mathbf{A}^T \bar{\mathbf{r}}_i} \right)$$

$$\mathbf{y}_{i+1} = \mathbf{A}^T \bar{\mathbf{r}}_{i+1} + \mathbf{y}_i \left(\frac{\bar{\mathbf{s}}_{i+1}^T \mathbf{A}^T \bar{\mathbf{r}}_{i+1}}{\bar{\mathbf{s}}_i^T \mathbf{A}^T \bar{\mathbf{r}}_i} \right)$$

Note that we have, for both these algorithms, introduced auxiliary sequences to avoid extra matrix-vector products as we did in the unpreconditioned versions.

AIAF	approximate inverse of approximate factor
AIB	\mathbf{A} inverse by bordering
AIF	approximate inverse factor
AIM	approximate inverse matrix
AIInv	\mathbf{A} inverse
FSAI	factorised sparse approximate inverse
IC(p)	level-based incomplete Cholesky
ICT	incomplete Cholesky with threshold
IKJ	order of nesting of the for-loops
ILU(p)	level-based incomplete LU
ILUT	incomplete LU with threshold
ILUTP	ILUT with pivoting
JP	Jacobi polynomial
MDF	minimum discarded fill
MR	minimal residual
MRP	MR (self) preconditioned
PCG	preconditioned conjugate gradient
SGS	symmetric Gauss-Seidel
SOR	successive over-relaxation
SSOR	symmetric successive over-relaxation
ST	simple triangularisation
TN	truncated Neumann

Table 9

11.3. Notation (again)

The considerable number of new preconditioners that have emerged over the past few years have spawned a similar number of acronyms and abbreviations. For convenience some of these are presented in tabular form above, using the same conventions that were adopted in Chapter 1.

The term IC should be taken to include both $IC(p)$ and ICT, and similarly for ILU. TN is normally used in conjunction with other acronyms, e.g. TNSOR or TNSGS.

11.4. Polynomial preconditioning

Polynomial preconditioning is normally applied to positive definite symmetric systems, and involves pre- or post-multiplication of the coefficient matrix \mathbf{A} by some low-degree polynomial in \mathbf{A} . The motivation for this has changed over the years. In the first example of polynomial preconditioning known to the authors it is clear and straightforward, and is simply the reduction in the condition number of the coefficient matrix in order that the method of solution subsequently applied would converge more rapidly. Then came a period in the early 1950's when many new methods of solution were being proposed (including, of course, the Krylov methods themselves). Some of these were "pure" methods based on a single idea, like the methods of Frankel [113] and Flanders and Shortley [109]. Other suggested methods were hybrid methods in which two or more ideas alternated. One incentive for the development of these methods was the expectation that one component of the hybrid algorithm would deal with some of the excesses committed by the other. An example of this kind of thinking is to be found in Rutishauser's discussion of an algorithm credited to Lanczos [175] in which CG or CR is interspersed with an algorithm based on the properties of Chebychev polynomials (see [213], section 5). These hybrid algorithms were not too different from the preconditioned algorithms being proposed at the time. In the next section of Rutishauser's paper the stated objective is to "Replace the system given by another system with the same solution but with a coefficient matrix \mathbf{A} of smaller condition number" but the resulting algorithm, based on a transformation by Stiefel [225], is then described in terms of "inner methods" and "outer methods" and is quite similar to those discussed in the previous section of the paper.

Thus the distinction between hybrid methods and preconditioned methods was becoming blurred. Moreover, in the context of the Krylov algorithms, where the residual at any stage may be expressed as a matrix polynomial multiplying the original residual (see Chapter 4, above), it was gradually being realized that since CG itself minimises the energy norm of the error over a particular space of polynomials, the insertion of additional polynomials into the process was something of a distraction. The motivation for this type of preconditioning was becoming harder to find.

This view changed with the advent of the large vector computers. For these

computers, which can carry out several vector operation in parallel, essentially serial operations like back-substitution and the calculation of inner products are an anathema since they do not permit such computers to utilise their full capabilities. When these considerations are taken into account, polynomial preconditioners become more attractive and are currently the subject of continuing investigations.

11.4.1. An early example

The first proposal for a polynomial preconditioner is usually attributed to Lanczos ([176], 1953) but the idea of using specific polynomials to improve the condition number of a coefficient matrix goes back at least to 1937 and a paper by Cesari [56]. To be sure the method of solution to which it was applied was not a gradient method, and the number of equations solved was only three, but the thinking behind the proposal was thoroughly modern. Equally modern was Cesari's handling of matrices at a time when many of his contemporaries were drowning in seas of \sum s.

The underlying method that benefited from the preconditioning was that of von Mises [252]. This solves $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is symmetric and positive definite, essentially by successive approximation where the matrix \mathbf{M} in the splitting defined by equation (11.95) is $\theta^{-1}\mathbf{I}$ for some positive scalar θ . Thus $\mathbf{N} = \theta^{-1}\mathbf{I} - \mathbf{A}$ and the crucial matrix that determines the rate of convergence, $\mathbf{M}^{-1}\mathbf{N}$, is given by $\mathbf{I} - \theta\mathbf{A}$. Now if the eigenvalues λ_j of \mathbf{A} satisfy $0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, the value of θ that minimises the spectral radius of $\mathbf{I} - \theta\mathbf{A}$ is obtained by solving $|1 - \theta\lambda_n| = 1 - \theta\lambda_1$ and is $2/(\lambda_n + \lambda_1)$. The corresponding value of the spectral radius is $(\lambda_n - \lambda_1)/(\lambda_n + \lambda_1)$ or $(\kappa - 1)/(\kappa + 1)$, where κ is the condition number of \mathbf{A} . Hence a reduction in the condition number combined with an opportune choice of θ will increase the rate of convergence of the von Mises algorithm, and Cesari achieves this by premultiplying the equation $\mathbf{Ax} = \mathbf{b}$ by a polynomial in \mathbf{A} of degree $r - 1$, $\psi_{r-1}(\mathbf{A})$ say, to give

$$\varphi_r(\mathbf{A})\mathbf{x} = \psi_{r-1}(\mathbf{A})\mathbf{b} \quad (11.56)$$

where

$$\varphi_r(\mathbf{A}) \equiv \mathbf{A}\psi_{r-1}(\mathbf{A}). \quad (11.57)$$

He observes that for $\lambda_1 \leq \lambda \leq \lambda_n$, $\varphi_r(\lambda)$ should be positive (in order that $\varphi_r(\mathbf{A})$ should be positive definite) and gives explicit expressions for polynomials of degrees 2, 3 and 4 that satisfy this requirement. He further notes that these transformations may be applied recursively and finishes with a numerical example.

11.4.2. General polynomial preconditioning

Now while the motivation of Cesari is unexceptionable, matters become less clear-cut when the same technique is applied to sets of equations which are to be solved by CG, and it is appropriate here to recall some results from Chapter 4. Suppose that \mathbf{A} is symmetric positive definite and we solve $\mathbf{Ax} = \mathbf{b}$ using the vector forms

of equations (3.19) - (3.21) with $\mathbf{G} = \mathbf{A}$ (as we will if we solve $\mathbf{Ax} = \mathbf{b}$ by PCG). It was shown in that chapter that if \mathbf{x}_{i+1} denotes the intermediate solution computed after i iterations of PCG and if $\mathbf{s}_j = \mathbf{A}^{\frac{1}{2}}\mathbf{x}_j - \mathbf{A}^{-\frac{1}{2}}\mathbf{b}$ then (equation (4.32))

$$\|\mathbf{s}_{i+1}\| = \min_{p_i(\mathbf{C}) \in \mathcal{P}_i(\mathbf{C})} \|p_i(\mathbf{C}) \mathbf{s}_1\| \quad (11.58)$$

where $\mathcal{P}_i(\mathbf{C})$ is defined by equation (4.31) and $\mathbf{C} = \mathbf{A}^{\frac{1}{2}}\mathbf{K}\mathbf{A}^{\frac{1}{2}}$ (equation (4.29)). Now for polynomial preconditioning

$$\mathbf{K} = \psi_{r-1}(\mathbf{A}), \quad (11.59)$$

where $\psi_{r-1}(\mathbf{A})$ is some given polynomial in \mathbf{A} of degree $(r-1)$, and since $\mathbf{C} = \mathbf{A}^{\frac{1}{2}}\mathbf{K}\mathbf{A}^{\frac{1}{2}}$ it follows that $\mathbf{C} = \varphi_r(\mathbf{A})$ where $\varphi_r(\mathbf{A})$ may be identified with the polynomial $\varphi_r(\mathbf{A})$ of equation (11.57). Denote this particular value of \mathbf{C} by Φ_r . Its crucial property is that it is a polynomial in \mathbf{C} of degree r whose coefficient of \mathbf{A}^0 is zero. Hence

$$\Phi_r = \beta_1 \mathbf{A} + \beta_2 \mathbf{A}^2 + \cdots + \beta_r \mathbf{A}^r \quad (11.60)$$

for some set of scalars $\beta_j \in \mathbb{R}$, $j = 1, 2, \dots, r$, determined by the choice of \mathbf{K} . Substituting Φ_r for \mathbf{C} in equation (11.58) then gives

$$\|\mathbf{s}_{i+1}\| = \min_{p_i(\Phi_r) \in \mathcal{P}_i(\Phi_r)} \|p_i(\Phi_r) \mathbf{s}_1\|. \quad (11.61)$$

Now any polynomial $p_i(\Phi_r)$ is some polynomial in \mathbf{A} of degree at most ir , $p_{ir}(\mathbf{A})$ say, and since $p_i(\Phi_r) \in \mathcal{P}_i(\Phi_r)$ it follows from equation (11.60) that the coefficient of the unit matrix in $p_{ir}(\mathbf{A})$ is unity. Thus $p_{ir}(\mathbf{A}) \in \mathcal{P}_{ir}(\mathbf{A})$ and the minimum specified by equation (11.61) is performed over a subset of the polynomials in $\mathcal{P}_{ir}(\mathbf{A})$, it being only a subset since Φ_r is a fixed polynomial.

Let now $\hat{\mathbf{e}}_{i+1}$ denote the error obtained as a result of this partial minimisation and \mathbf{e}_{ir+1} the error obtained by performing ir steps of unpreconditioned CG on the original system $\mathbf{Ax} = \mathbf{b}$ and starting at the same initial value of \mathbf{x} . From the discussion in Chapter 4 it follows that \mathbf{e}_{ir+1} is the error obtained by minimising $\|\mathbf{e}\|_A$ over all polynomials in $\mathcal{P}_{ir}(\mathbf{A})$. Since minimising over a subset can never be more effective than minimising over the whole set this implies that¹³

$$\|\mathbf{e}_{ir+1}\|_A \leq \|\hat{\mathbf{e}}_{i+1}\|_A.$$

Thus, in exact arithmetic, we do at least as well by performing ir CG iterations on the original system as we do by performing i iterations on the same system that has been subject to *any* polynomial preconditioning. Since the amount of work in each case is roughly the same (both cases require the equivalent of ir multiplications of some full vector by \mathbf{A}) there is little to be gained either from the point of view of convergence or of reducing the total workload by a polynomial preconditioning of the original system. Only when the calculations are performed on computers

¹³Recall that $\|\mathbf{e}\|_A = \|\mathbf{s}\|$.

whose architectures permit parallel vector operations can any significant advantage be achieved.

Finally we note that if the condition number of Φ_r is K_r and we define Σ_r by

$$\Sigma_r = \frac{K_r^{1/2} - 1}{K_r^{1/2} + 1} \quad (11.62)$$

then, by analogy with inequality (4.42),

$$\frac{\|\hat{\mathbf{e}}_{i+1}\|_A}{\|\mathbf{e}_1\|_A} \leq \frac{2\Sigma_r^i}{1 + \Sigma_r^{2i}}. \quad (11.63)$$

This result is valid for any polynomial preconditioner $\psi_{r-1}(\mathbf{A})$.

For further discussion of general polynomial preconditioning see Axelsson [9].

11.4.3. Neumann preconditioning

Equation (4.40) and inequality (4.42) indicate that if \mathbf{G} ($= \mathbf{A}$) is symmetric positive definite then any similar preconditioner \mathbf{K} should be chosen to minimise the “condition number” of \mathbf{KA} (strictly of $\mathbf{A}^{\frac{1}{2}}\mathbf{KA}^{\frac{1}{2}}$). Even if symmetry and positive definiteness are absent, all the evidence shows that \mathbf{K} should approximate \mathbf{A}^{-1} as closely as possible to achieve effective preconditioning. One choice of \mathbf{K} is given by $\mathbf{K} = (\overline{\mathbf{L}}\overline{\mathbf{U}})^{-1}$ where $\overline{\mathbf{L}}$ and $\overline{\mathbf{U}}$ are incomplete triangular factors of \mathbf{A} and stored as such (see page 231 below), but this choice has the disadvantage that the calculation of \mathbf{Kr} as is required by all preconditioned Krylov algorithms (see equations (11.28), (11.40), (11.41), (11.54) and (11.55)) relies on a double back substitution. On scalar machines the cost of two back substitutions with triangular matrices and that of premultiplying a vector by a matrix with the same number of nonzero elements is approximately the same. On vector machines, though, back substitution is far more time-consuming than matrix multiplication because of its inherently sequential nature so for these machines it is far better to store the matrix \mathbf{K} and compute \mathbf{Kr} as a matrix vector product. The problem then is how to compute \mathbf{K} . One solution of this is the use of truncated Neumann series to generate a sequence of improving approximate inverses as suggested by Dubois, Greenbaum and Rodrigue [92].

Let then \mathbf{K}_1 be an initial approximate inverse and define \mathbf{B} by

$$\mathbf{B} \equiv \mathbf{I} - \mathbf{K}_1 \mathbf{A} \quad (11.64)$$

so that

$$\mathbf{A}^{-1} \equiv (\mathbf{I} - \mathbf{B})^{-1} \mathbf{K}_1. \quad (11.65)$$

It follows from equation (11.64) that \mathbf{B} may be regarded as a matrix of residuals. If \mathbf{B} is convergent, equation (11.65) may be written

$$\mathbf{A}^{-1} \equiv \left(\sum_{j=0}^{\infty} \mathbf{B}^j \right) \mathbf{K}_1$$

and this suggests that a better approximation to \mathbf{A}^{-1} may be obtained by truncating the sum after $r \geq 2$ terms ($r = 1$ gives the original approximation \mathbf{K}_1). Define then \mathbf{K}_r by

$$\mathbf{K}_r = \left(\sum_{j=0}^{r-1} \mathbf{B}^j \right) \mathbf{K}_1. \quad (11.66)$$

We now show that \mathbf{K}_r thus defined is both symmetric and positive definite for all r .

Since $\sum_{j=0}^{r-1} \mathbf{B}^j \equiv (\mathbf{I} - \mathbf{B}^r)(\mathbf{I} - \mathbf{B})^{-1}$ equations (11.65) and (11.66) give

$$\mathbf{K}_r = (\mathbf{I} - \mathbf{B}^r) \mathbf{A}^{-1}. \quad (11.67)$$

By hypothesis \mathbf{A} is positive definite so its square root $\mathbf{A}^{\frac{1}{2}}$ exists. If we denote $\mathbf{A}^{\frac{1}{2}} \mathbf{B} \mathbf{A}^{-\frac{1}{2}}$ by \mathbf{S} not only does

$$\mathbf{S}^r = \mathbf{A}^{\frac{1}{2}} \mathbf{B}^r \mathbf{A}^{-\frac{1}{2}} \quad (11.68)$$

but, from equation (11.64),

$$\mathbf{S} = \mathbf{I} - \mathbf{A}^{\frac{1}{2}} \mathbf{K}_1 \mathbf{A}^{\frac{1}{2}}.$$

Thus \mathbf{S} is symmetric since \mathbf{K}_1 is symmetric, and since it is similar to \mathbf{B} it is also convergent. Equations (11.67) and (11.68) then give

$$\mathbf{A}^{\frac{1}{2}} \mathbf{K}_r \mathbf{A}^{\frac{1}{2}} = \mathbf{I} - \mathbf{S}^r$$

so that, since \mathbf{S} is both symmetric and convergent, \mathbf{K}_r is symmetric and positive definite.

We now turn to practical matters. Equation (11.66) yields

$$\mathbf{K}_{r+1} = \mathbf{B} \mathbf{K}_r + \mathbf{K}_1$$

so that, if we define \mathbf{w}_r to be $\mathbf{K}_r \mathbf{r}$, then

$$\mathbf{w}_{r+1} = \mathbf{B} \mathbf{w}_r + \mathbf{w}_1 \quad (11.69)$$

where $\mathbf{w}_1 = \mathbf{K}_1 \mathbf{r}$. The sequence $\{\mathbf{w}_r\}$ may thus be computed recursively. The choice of \mathbf{K}_1 is arbitrary. Any matrix for which $\rho(\mathbf{I} - \mathbf{K}_1 \mathbf{A}) < 1$ would do and Dubois *et al* recommended $\mathbf{K}_1 = [\text{diag}(a_{ii})]^{-1}$ if $\mathbf{A} = [a_{ij}]$ and is symmetric positive definite. It can be shown that the spectral radius of this particular \mathbf{K}_1 is less than unity if either \mathbf{A} is an M-matrix or is strictly or irreducibly diagonally dominant, and for the test cases given in [92] at least one of these conditions is satisfied. A more elaborate value of \mathbf{K}_1 for more highly-structured¹⁴ matrices was given by van der Vorst [239]. If \mathbf{A} and \mathbf{K}_1 are sparse, an iteration with \mathbf{K}_r may be much faster

¹⁴In this chapter the term *structure* refers to the disposition of the zero and non-zero elements of the matrix. It does not depend on the actual values of the non-zeroes.

than r iterations with \mathbf{K}_1 , because it avoids the additional vector sums and inner products. If the number of iterations is reduced by almost a factor of r , this can give significant savings.

It was shown in [92] that using \mathbf{K}_r as a preconditioner instead of \mathbf{K}_1 can reduce the number of iterations for the PCG method by no more than a factor of r (see also the discussion on generalised polynomial preconditioning, above). This reduction in general will not actually be achieved although numerical experiments have shown that for $r = 2$ it often nearly is. Those quoted in [92] refer to two sets of problems involving the application of the well-known five-point approximation to $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ on a square mesh and indicate that, for the problems tested and for a vector machine,

- (1) The use of the preconditioner based on the truncated Neumann series for $r = 2, 3$ or 4 nearly always reduced the time taken compared with $r = 1$. The choice $r = 2$ gave the best results overall with time savings of between 10% and 25%
- (2) The reduction in the number of iterations achieved by using the truncated Neumann series was less than that obtained by using an incomplete Cholesky preconditioner
- (3) Despite this, the time taken by the incomplete Cholesky preconditioner was always greater than that for the truncated Neumann preconditioner because the latter exploited the chosen computers more effectively (but see [239] where it was shown that for certain regularly-structured matrices, IC preconditioning could in fact be faster than Neumann preconditioning even on a vector machine).

We note that for Neumann preconditioning it is not necessary to estimate λ_1 and λ_n , respectively the smallest and largest eigenvalue of \mathbf{A} . It is only necessary to have some suitable initial approximation to \mathbf{A}^{-1} and this, in many cases, may be obtained simply from the diagonal elements of the original matrix (see above).

11.4.4. Chebychev preconditioning

Even though the advantages of polynomial preconditioning are mainly confined to vector computers, it is still advisable to choose the preconditioner to minimise the condition number of $\varphi_r(\mathbf{A})$. Johnson, Micchelli and Paul [163] approached the problem by considering the case where \mathbf{A} has been symmetrically scaled so that $a_{ii} = 1$ for all i , and choosing $\mathbf{K}_1 = \mathbf{I}$ in equation (11.64). Equations (11.64) and (11.66) then yield $\mathbf{K}_r = \sum_{j=0}^{r-1} (\mathbf{I} - \mathbf{A})^j$ but the authors pointed out that a better way of choosing a preconditioner would be to replace \mathbf{K}_r by an arbitrary polynomial in \mathbf{A} of degree $r - 1$, $\psi_{r-1}(\mathbf{A})$ say, choosing its coefficients so that it “looks like” \mathbf{A}^{-1} . One way of making this idea more precise is by minimising the condition number of $\mathbf{A}\psi_{r-1}(\mathbf{A})$ since the unit matrix is perfectly conditioned. We are back, in fact, to Cesari’s idea of premultiplying $\mathbf{Ax} = \mathbf{b}$ by a suitably-chosen polynomial $\psi_{r-1}(\mathbf{A})$.

Let then $\varphi_r(\mathbf{A}) \equiv \mathbf{A}\psi_{r-1}(\mathbf{A})$. It then follows trivially that any such polynomial $\varphi_r(\mathbf{A})$ may be expressed as

$$\varphi_r(\mathbf{A}) \equiv \mathbf{I} - p_r(\mathbf{A}) \quad (11.70)$$

for some $p_r(\mathbf{A}) \in \mathcal{P}_r(\mathbf{A})$, where $\mathcal{P}_r(\mathbf{A})$ is defined by equation (4.31). Hence minimising the condition number of $\varphi_r(\mathbf{A})$ is equivalent to minimising the condition number of $\mathbf{I} - p_r(\mathbf{A})$ over all $p_r(\mathbf{A}) \in \mathcal{P}_r(\mathbf{A})$. Now \mathbf{A} is symmetric by hypothesis so $\varphi_r(\mathbf{A})$ is also symmetric, and if its eigenvalues are positive its condition number is the ratio of the largest and smallest of these. If K_r and ρ_r denote the condition number of $\varphi_r(\mathbf{A})$ and the spectral radius of $p_r(\mathbf{A})$ respectively and if $\rho_r < 1$ then, from equation (11.70),

$$K_r \leq \frac{1 + \rho_r}{1 - \rho_r} \quad (11.71)$$

with equality if the largest and smallest eigenvalues of $p_r(\mathbf{A})$ are $\pm \rho_r$. Thus in order to minimise K_r it suffices to minimise ρ_r .

Now the minimisation of the spectral radius of $p_r(\mathbf{A})$ requires knowledge of all the eigenvalues of \mathbf{A} and is thus not possible. However if λ_i , $0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$, are the eigenvalues of \mathbf{A} and we know (or can estimate) λ_1 and λ_n we can determine a polynomial $p_r(\lambda)$ in $\mathcal{P}_r(\lambda)$ such that $|p_r(\lambda)|$ is minimised over the interval $[\lambda_1, \lambda_n]$. This polynomial may then be used to bound the eigenvalues, and hence the spectral radius, of $p_r(\mathbf{A})$.

Define then δ_r by

$$\delta_r = \min_{p_r(\lambda) \in \mathcal{P}_r(\lambda)} \max_{\lambda_1 \leq \lambda \leq \lambda_n} |p_r(\lambda)|$$

and denote the polynomial that effects this minimisation by $P_r(\lambda)$. It is now well-known (see any book on approximation theory) that

$$\delta_r = 1/T_r(\bar{\mu})$$

where

$$\bar{\mu} = \frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}$$

and that

$$P_r(\lambda) \equiv \frac{T_r\left(\frac{\lambda_n + \lambda_1 - 2\lambda}{\lambda_n - \lambda_1}\right)}{T_r(\bar{\mu})}$$

where $T_r(x)$ is the Chebychev polynomial of the first kind of degree r (see Appendix D). Note that $P_r(0) = 1$ as is required by equation (4.31) ($P_r(\lambda)$ is plotted for $\lambda_1 = 1$, $\lambda_n = 4$ and $r = 4, 10$ and 25 in Graphs 123 - 125).

Consider now the matrix polynomial

$$P_r(\mathbf{A}) \equiv \frac{T_r\left(\frac{(\lambda_n + \lambda_1)\mathbf{I} - 2\mathbf{A}}{\lambda_n - \lambda_1}\right)}{T_r(\bar{\mu})}. \quad (11.72)$$

If \mathbf{x}_k is the eigenvector of \mathbf{A} corresponding to the eigenvalue λ_k we have

$$\left(\frac{(\lambda_n + \lambda_1) \mathbf{I} - 2\mathbf{A}}{\lambda_n - \lambda_1} \right) \mathbf{x}_k = \mathbf{x}_k \mu_k$$

where

$$\mu_k = \frac{\lambda_n + \lambda_1 - 2\lambda_k}{\lambda_n - \lambda_1}. \quad (11.73)$$

Thus, from equation (11.72),

$$P_r(\mathbf{A}) \mathbf{x}_k = \mathbf{x}_k (T_r(\mu_k) / T_r(\bar{\mu})) \quad (11.74)$$

so that the eigenvalues of $P_r(\mathbf{A})$ are $T_r(\mu_k) / T_r(\bar{\mu})$. Now λ_1 and λ_n are, by definition, the extreme eigenvalues of \mathbf{A} so that, from equation (11.73), $-1 \leq \mu_k \leq 1$ for all k . It now follows from the properties of Chebychev polynomials that $|T_r(\mu_k)| \leq 1$ for all μ_k with equality for $\mu_k = \pm 1$. Hence, from equation (11.74), the spectral radius ρ_r of $P_r(\mathbf{A})$ is $1/T_r(\bar{\mu})$. The preconditioner $\psi_{r-1}(\mathbf{A})$ can then be found from equation (11.57) since, from equation (11.70),

$$\varphi_r(\mathbf{A}) \equiv \mathbf{I} - \frac{T_r \left(\frac{(\lambda_n + \lambda_1) \mathbf{I} - 2\mathbf{A}}{\lambda_n - \lambda_1} \right)}{T_r(\bar{\mu})}.$$

This expression for the polynomial $\varphi_r(\mathbf{A})$ was obtained in its scalar form by Johnson *et al* [163] but it was also known to Rutishauser [213]. This was acknowledged in [163] but the authors claim that in [213] “its optimality property ... was not observed”.

It must be appreciated that the polynomial of equation (11.72) does not, in general, solve the problem

$$\min_{p_r(\mathbf{A}) \in \mathcal{P}_r(\mathbf{A})} \rho(p_r(\mathbf{A})) \quad (11.75)$$

but only gives an upper bound for this on the assumption that the largest and smallest eigenvalue of \mathbf{A} are known. Given λ_1 and λ_n , the solution of (11.75) depends on the distribution of the $n - 2$ remaining eigenvalues in $[\lambda_1, \lambda_n]$, and although distributions exist for which the bound given by equation (11.72) is sharp (the worst-case scenario), for other distributions it can be quite pessimistic. This opens the way for other approximate solutions of problem (11.75), some of which are discussed below. It also underlines the fact that given λ_1 and λ_n , $(T_r(\bar{\mu}) + 1) / (T_r(\bar{\mu}) - 1)$ is only an upper bound for the condition number of $\varphi_r(\mathbf{A})$.

In order to use these results we appeal to the expression for $T_r(\bar{\mu})$ derived in Chapter 4, above. From equation (4.41) we get

$$\rho_r \leq \frac{2\sigma^r}{1 + \sigma^{2r}} \quad (11.76)$$

where, from equation (4.40),

$$\sigma = \frac{\kappa^{1/2} - 1}{\kappa^{1/2} + 1} \quad (11.77)$$

and where κ is the condition number of \mathbf{A} . It now follows from inequalities (11.71) and (11.76) that

$$K_r \leq \frac{1 + \frac{2\sigma^r}{1+\sigma^{2r}}}{1 - \frac{2\sigma^r}{1+\sigma^{2r}}}$$

or

$$K_r \leq \left(\frac{1 + \sigma^r}{1 - \sigma^r} \right)^2. \quad (11.78)$$

Thus if the polynomial $\varphi_r(\mathbf{A})$ is defined specifically by equations (11.57) and (11.72) we can obtain a bound on its condition number K_r in terms of its degree and of the condition number of \mathbf{A} . This bound is straightforward but not overly simple and there is a much simpler relationship between Σ_r and σ . This, from equations (11.62) and (11.78), is

$$\Sigma_r \leq \sigma^r \quad (11.79)$$

and substituting this value of Σ_r in inequality (11.63) gives

$$\frac{\|\hat{\mathbf{e}}_{i+1}\|_A}{\|\mathbf{e}_1\|_A} \leq \frac{2\sigma^{ir}}{1 + \sigma^{2ir}}. \quad (11.80)$$

But, from inequality (4.42) with $G = A$,

$$\frac{\|\mathbf{e}_{ir+1}\|_A}{\|\mathbf{e}_1\|_A} \leq \frac{2\sigma^{ir}}{1 + \sigma^{2ir}} \quad (11.81)$$

so that we obtain the same upper bound for the error norms generated by performing ir unpreconditioned iterations on the original system as we do by performing i iterations on the same system but preconditioned by using a Chebychev polynomial of degree r . Since the amount of work in each case is roughly the same, any advantage gained from this type of preconditioning stems in general from the use of vector computers. This analysis is based on that given in [163].

We might note here in parentheses that solving $\mathbf{Ax} = \mathbf{b}$ by PCG is not the same as solving $\Phi_r \mathbf{x} = \mathbf{b}_1$ by CG even if the same polynomial is used in both cases as a preconditioner. In the first case $\|\mathbf{e}\|_A$ is minimised whereas in the second the norm to be minimised is $\|\mathbf{e}\|_{\Phi_r}$.

11.4.5. Other polynomial preconditioners

The fact that, in inequalities (11.80) and (11.81), both $\|\mathbf{e}_{ir+1}\|_A$ and $\|\hat{\mathbf{e}}_{i+1}\|_A$ have the same upper bound does not imply that the two norms necessarily have similar values. The bounds depend only on the smallest and largest eigenvalues of \mathbf{A} but the inequalities are valid for any distribution of eigenvalues in the interval $[\lambda_1, \lambda_n]$. It is not unreasonable that for some distributions of eigenvalues the inequalities could be very pessimistic and for many practical problems, notably those arising from

the numerical solution of partial differential equations, this seems to be the case for inequality (11.81) (see [108]). Inequality (11.80) on the other hand consistently tends to be a much better bound since the eigenvalue distribution resulting from Chebychev preconditioning is nearly always unfavourable to CG solution [2]. In these circumstances, when (11.81) is pessimistic and (11.80) is reasonably tight, Chebychev preconditioning is not optimal and better strategies may be found.

One such strategy due to Johnson *et al* [163] is replacing the polynomial given by equation (11.72) by one based on solving

$$\min_{p(\lambda) \in \mathcal{P}_r(\lambda)} \int_{\lambda_1}^{\lambda_n} [p(\lambda)]^2 w(\lambda) d\lambda, \quad (11.82)$$

where $w(\lambda)$ is some weight function chosen “to emphasize the portion of the spectrum which is most important”. It must be positive on the open interval (λ_1, λ_n) . The authors then used this expression to derive various preconditioners together with the recurrence formulæ by which they may be computed. These formulæ only require approximations to the largest and smallest eigenvalues of \mathbf{A} for their definition (the same information as that required by the Chebychev preconditioners). One possible weight function is

$$\omega(\lambda) \equiv [(\lambda - \lambda_1)(\lambda_n - \lambda)]^{-\frac{1}{2}}. \quad (11.83)$$

This leads to a particular set of polynomials (*Jacobi polynomials*) which may then be used as preconditioners. The success of these methods relies on a favourable distribution of eigenvalues since Chebychev preconditioning is optimal for the worst case. Unfortunately no experimental results were given in [163]. They were relegated to [162] and are not currently available [159].

Other strategies may be based on generalised Chebychev polynomials. They may be derived by solving

$$\min_{p(\lambda) \in \mathcal{P}_r} \max_{\lambda_1 \leq \lambda \leq \lambda_n} |p(\lambda) \omega(\lambda)| \quad (11.84)$$

for some weight function $\omega(\lambda)$ (if $\omega(\lambda) \equiv 1$ we recover the normal Chebychev polynomial). From the minimising polynomial $P_r(\lambda)$ and equation (11.70) we can obtain $\varphi_r(\lambda)$ and then, from equation (11.57), the polynomial preconditioner $\mathbf{K} = \psi_{r-1}(\mathbf{A})$. The function $w(\lambda)$ is computed using a more detailed knowledge of the spectrum of \mathbf{A} than just λ_1 and λ_n , or could be chosen to satisfy other criteria as it was when minimising expression (11.82).

An example of such a method is one due to Fischer and Freund [108]. This requires considerably more information about the problem than the previous methods but which ultimately gives a preconditioner which requires fewer iterations for convergence. It is a generalisation of the Chebychev preconditioner described above, and is based on the idea of a distribution function (defined below). Although this function would most naturally be expressed in terms of the eigenvalues of \mathbf{A} , much of the underlying theory relates to Chebychev polynomials whose properties are

closely linked to the interval $[-1, 1]$. This makes it convenient to express the distribution function too in terms of this interval and we first establish the necessary correspondence.

Assume then that the initial residual \mathbf{r}_1 may be expressed as

$$\mathbf{r}_1 = \sum_{j=1}^L \mathbf{v}_j \sigma_j \quad (11.85)$$

where the vectors \mathbf{v}_j are a subset of the eigenvectors of \mathbf{A} (hence $L \leq n$). It is further assumed that these are scaled so that $\|\mathbf{v}_j\| = 1$ and that

$$0 \leq \alpha \leq \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_L \leq \beta,$$

where α and β are estimates of λ_1 and λ_L respectively and at least one of the first two, and one of the last $L+1$, inequalities is strict implying that $0 < \lambda_1$ and $\alpha < \beta$. Let now x be given by $x = (2\lambda - \beta - \alpha)/(\beta - \alpha)$ and define the *distribution function* $\sigma(x)$ by

$$\sigma(x) \equiv \sum_{x_j \leq x} \sigma_j^2.$$

The distribution function is clearly discontinuous at $x = x_j$ and resembles an irregularly rising staircase, the positions of the verticals corresponding to the shifted eigenvalues x_j and their heights reflecting the values of σ_j^2 , i.e. the amounts that the corresponding eigenvectors contribute to the make-up of \mathbf{r}_1 .

Now $\sigma(x)$ is not known and its calculation would be prohibitively expensive but it may be approximated by applying the Lanczos algorithm (see page 47) to the original problem starting at the same initial approximation. A distribution function may then be defined for the symmetric tridiagonal matrix $\bar{\mathbf{H}}_i$ given by equation (3.23) and it can be shown [168] that this is related in a very precise manner to that of the original problem. It has the same general characteristics as $\sigma(x)$ and can readily be computed if i is not too large. The practical outcome of this is that if we take the mid-points of the horizontals and verticals of the distribution function generated by $\bar{\mathbf{H}}_i$ and interpolate them by some strictly monotonically-increasing differentiable function (in [108] a piecewise cubic was used) then we have some sort of approximation to $\sigma(x)$, the required distribution function. The derivative of this function, $\delta(x)$, is known as the *density function* and can be used to construct new polynomial preconditioners. This is done via the weight function $\omega(x)$ which Fischer and Freund chose to be

$$\omega(x) = \sqrt{\delta(x) \sqrt{1-x^2}},$$

(see the original paper for the reasons for this choice). They then either solved equation (11.84) for $P_r(\lambda)$ using the Remez algorithm or computed it using the Bernstein-Szegő weight functions. From $P_r(\lambda)$ the preconditioner $\psi_{r-1}(\mathbf{A})$ may be simply recovered as described above. The complete process may be summed up by

- (1) Choose $\mathbf{x}_1 \in \mathbb{R}^n$ and compute $\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$.

- (2) Estimate the distribution function $\sigma(x)$ by
 - (a) performing i steps of the Lanczos algorithm starting with $\mathbf{r}_1 / \|\mathbf{r}_1\|$,
 - (b) computing the Lanczos distribution function $\tau(x)$ which has the same “rising staircase” shape as $\sigma(x)$
 - (c) setting up the interpolation problem where the interpolation points are the mid-points of the horizontals and verticals of $\tau(x)$
- (3) Determine the polynomial preconditioner $\psi_{r-1}(\mathbf{A})$ by
 - (a) computing the monotone piecewise cubic function that interpolates the points obtained in the previous step, and
 - (b) either solving the associated weighted Chebychev approximation problems or computing the Bernstein-Szegő weight functions.

Numerical tests carried out on six examples based on solving Dirichlet-type problems for various mesh spacings were described in [108]. Both forms of the method (Remez and Bernstein-Szegő (BS)) were tested against Chebychev preconditioning and the least squares polynomial preconditioner of equation (11.82) with $w(\lambda) \equiv 1$. This was referred to as the *Legendre preconditioner* in [108]. In all cases twenty Lanczos steps were taken to establish the approximate distribution function, a choice that permitted the BS approximation to be computed for $r \geq 9$.

Each problem/method combination was tested using three values of the degree r of the preconditioner ($r = 5, 10, 15$) so that the total number of tests was $6 \times (4 \times 3 - 1)$, the loss of six tests being due to the inapplicability of BS for $r = 5$. The aggregate number of iterations for each of the eleven method/degree combinations is given in the following table:

r	5	10	15
BS	n/a	332	239
Remez	664	359	263
Chebychev	>2912	2186	1484
Legendre	1335	942	768

Table 10

The iterations for the individual problems were roughly similar and showed that:

- (1) For every problem/method combination quoted, an increase in the degree of the preconditioner resulted in a reduction in the number of iterations required to satisfy the convergence criterion
- (2) For every problem/degree combination quoted, the BS or the Remez algorithm always out-performed the corresponding Chebychev or Legendre algorithm
- (3) For every problem/degree combination quoted, in only one case was the Remez version superior to the BS version
- (4) For exactly half of the problem/degree combinations quoted, the Chebychev method was inferior to the Legendre method.

Observation (4) was due principally to the less erratic behaviour of the Legendre preconditioner compared with the Chebychev one which on two occasions was halted after 1000 iterations without the convergence criterion having been satisfied.

The comparable figure for the total number of iterations required by the un-preconditioned CG method (which managed to solve all problems) was 2527, and although this compares very unfavourably with most of the entries in the table it must be remembered that the amount of work per CG iteration when the coefficient matrix is a single sparse matrix is significantly less than that needed when the coefficient matrix is a matrix polynomial of degree ten, say. However this problem may be ameliorated to some extent by the use of vector or parallel computers.

11.5. Some non-negative matrix theory

This theory is important in practice since many of the matrices derived from the discretisation of partial differential equations are related to non-negative matrices. To the extent that Krylov methods are applied to equations derived from such problems the theory applies also to them. It principally affects preconditioning and justifies some of the operations used in the construction of preconditioners. However it has also been applied to other iterative processes, notably to successive approximation, and we begin by giving an outline of the more general aspects of the theory.

Definition 4. A matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$ is non-negative (positive or strictly positive), written $\mathbf{B} \geq \mathbf{O}$ ($\mathbf{B} > \mathbf{O}$), if all its elements are non-negative (strictly positive). We also write $\mathbf{A} \geq \mathbf{B}$ for $\mathbf{A} - \mathbf{B} \geq \mathbf{O}$, etc.

Definition 5. A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is said to be reducible if there exists a permutation matrix \mathbf{P} such that

$$\mathbf{P}^T \mathbf{A} \mathbf{P} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{O} & \mathbf{A}_{22} \end{bmatrix}$$

where \mathbf{A}_{11} and \mathbf{A}_{22} are both square. If no such matrix exists then \mathbf{A} is said to be irreducible.

Since we are interested in the convergence of iterative processes involving non-negative matrices we first consider some theorems involving their spectral radii.

Theorem 33. For any non-negative matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ the following two conditions hold:

1. $\rho(\mathbf{B})$ is an eigenvalue of \mathbf{B} , and
2. There exists a non-negative vector $\mathbf{x} \neq \mathbf{0}$ such that $\mathbf{Bx} = \mathbf{x}\rho(\mathbf{B})$.

Proof. See Horn and Johnson [155], page 503 (Theorem 8.3.1). ■

Note that the possibility that $\rho(\mathbf{B}) = 0$ is not excluded even if $\mathbf{B} \neq \mathbf{O}$, as would occur if for example $\mathbf{B} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

This theorem is a weaker form of two very much better-known theorems that between them constitute one of the linchpins of non-negative matrix theory. If, in

the above theorem, we impose the further conditions that $\mathbf{B} \neq \mathbf{O}$ and is irreducible we obtain the additional results that

3. $\rho(\mathbf{B}) > 0$, and
4. $\rho(\mathbf{B})$ is a simple eigenvalue of \mathbf{B} .

In this form the theorem is known as the *Perron-Frobenius Theorem* and was due to Frobenius [124]. An earlier version for strictly positive matrices was given by Perron [206].

We now consider the iterative solution of $\bar{\mathbf{A}}\mathbf{x} = \bar{\mathbf{b}}$ where $\bar{\mathbf{A}} \in \mathbb{R}^{n \times n}$ and is nonsingular. Assume, for simplicity and without loss of generality, that $\bar{\mathbf{A}} = [\bar{a}_{ij}]$ has already been scaled so that its diagonal elements are equal to unity. The scaled version \mathbf{A} may thus be expressed as

$$\mathbf{A} = \mathbf{I} - \mathbf{L} - \mathbf{U} \quad (11.86)$$

where \mathbf{U} is a strictly upper and \mathbf{L} a strictly lower triangular matrix. This scaling may be effected either by row scaling or, if $\bar{\mathbf{A}}$ is symmetric and it is desired to maintain symmetry, by combined row and column scaling of the form $\mathbf{A} = \mathbf{D}\bar{\mathbf{A}}\mathbf{D}$ where $\mathbf{D} \in \mathbb{R}^{n \times n}$ is diagonal. Implicit in this assumption are the further assumptions that the diagonal elements of the original matrix are nonzero and, if symmetry is to be preserved, strictly positive. This may seem to be somewhat extreme for very sparse matrices which might be expected to have many zero diagonal entries but if these matrices are derived, as they frequently are, from partial differential equations and both the equations and elements of the solution are ordered in a natural manner, then usually at least one and often all three of the following conditions are satisfied:

$$\bar{a}_{ii} > 0, \quad i = 1, 2, \dots, n,$$

$$\bar{a}_{ij} \leq 0, \quad j \neq i$$

and

$$\sum_{j \neq i} |\bar{a}_{ij}| \leq |\bar{a}_{ii}|, \quad i = 1, 2, \dots, n.$$

The assumption that $\bar{\mathbf{A}}$ may be scaled to satisfy equation (11.86) is thus not quite as outrageous as might at first appear.

Definition 6. If a matrix $\bar{\mathbf{A}} = [\bar{a}_{ij}] \in \mathbb{R}^{n \times n}$ satisfies the third of the above three conditions it is said to be diagonally dominant.

If the first of these three conditions is satisfied it is possible to obtain the form of equation (11.86) simply by scaling each row of $\bar{\mathbf{A}}$ by the appropriate positive factor. The three conditions then become

$$a_{ii} = 1, \quad i = 1, 2, \dots, n, \quad (11.87)$$

$$a_{ij} \leq 0, \quad j \neq i \quad (11.88)$$

and

$$\sum_{j \neq i} |a_{ij}| \leq 1, \quad i = 1, 2, \dots, n. \quad (11.89)$$

Note that if $\mathbf{A} = \mathbf{I} - \mathbf{L} - \mathbf{U}$ and $\mathbf{B} = \mathbf{L} + \mathbf{U}$ then inequality (11.89) implies that $\|\mathbf{B}\|_\infty \leq 1$ so that $\rho(\mathbf{B}) \leq 1$. Now it is often important (see below) to know that $\rho(\mathbf{B})$ is strictly less than unity but this cannot be deduced from the above assumptions alone. To show this we need two further assumptions.

Theorem 34. Let¹⁵ $\mathbf{B} = [b_{ij}] \in \mathbb{R}^{n \times n}$ be irreducible, $\|\mathbf{B}\|_\infty = 1$ and

$$\sum_{j=1}^n |b_{ij}| < 1 \quad (11.90)$$

for at least one value of i . Then $\rho(\mathbf{B}) < 1$.

Proof. See e.g. Horn and Johnson [155] page 363 (corollary 6.2.28), Ortega [198] or Varga [246]. ■

There are several proofs of this theorem. One is based on the observation that as \mathbf{B} is raised to successively higher powers the number of rows for which inequality (11.90) is satisfied increases so that, for some finite r , all rows of \mathbf{B}^r satisfy it and $\|\mathbf{B}^r\|_\infty < 1$. Hence $\rho(\mathbf{B}^r) < 1$ and $\rho(\mathbf{B}) < 1$. If though \mathbf{B} is reducible it can happen that some rows of \mathbf{B}^r never satisfy inequality (11.90) no matter how large we make r . In this case reducibility destroys the connection between the rows, preventing the influence of those that do satisfy (11.90) from spreading (this view is reinforced by the lemma (see [198], page 219) which states that if $\mathbf{B} \in \mathbb{R}^{n \times n} \geq \mathbf{O}$, has strictly positive diagonal elements and is irreducible then $\mathbf{B}^{n-1} > \mathbf{O}$). If we consider the two examples

$$\mathbf{B}_1 = \begin{bmatrix} 1 & 1 \\ \frac{1}{4} & \frac{1}{4} \\ 1 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad \text{and} \quad \mathbf{B}_2 = \begin{bmatrix} 1 & 1 \\ \frac{1}{4} & \frac{1}{4} \\ 0 & 1 \end{bmatrix}$$

we see that for both matrices $\|\mathbf{B}\|_\infty = 1$ and that inequality (11.90) is satisfied for their first row. However \mathbf{B}_1 is irreducible while \mathbf{B}_2 is reducible, and $\rho(\mathbf{B}_1) = \frac{3}{4}$ whereas $\rho(\mathbf{B}_2) = 1$.

We now look at some of the implications of a spectral radius being less than unity.

Definition 7. A matrix \mathbf{B} is said to be convergent if $\lim_{i \rightarrow \infty} \|\mathbf{B}^i\| = 0$.

It is well known (see [155]) that a necessary and sufficient condition for \mathbf{B} to be convergent is that $\rho(\mathbf{B}) < 1$. This of course implies that a matrix is convergent if and only if the absolute values of all its eigenvalues are strictly less than unity.

¹⁵The matrix $\mathbf{I} - \mathbf{B}$, where \mathbf{B} satisfies the conditions of this theorem, is said to be *irreducibly diagonally dominant*.

Let \mathbf{A} be given by equation (11.86) and define \mathbf{B} by $\mathbf{B} = \mathbf{L} + \mathbf{U}$. Then $\mathbf{A} = \mathbf{I} - \mathbf{B}$ and since

$$(\mathbf{I} - \mathbf{B})(\mathbf{I} + \mathbf{B} + \mathbf{B}^2 + \cdots + \mathbf{B}^i) \equiv \mathbf{I} - \mathbf{B}^{i+1}$$

it follows that, if \mathbf{B} is convergent,

$$\mathbf{A}^{-1} = (\mathbf{I} - \mathbf{B})^{-1} = \sum_{i=0}^{\infty} \mathbf{B}^i. \quad (11.91)$$

If the elements of \mathbf{B} are, in addition, nonnegative then trivially \mathbf{A}^{-1} is also non-negative. Thus if $\mathbf{A} = \mathbf{I} - \mathbf{B}$ and \mathbf{B} satisfies the conditions of Theorem 34 then \mathbf{B} is convergent and \mathbf{A}^{-1} is non-negative.

The assumption that inequality (11.89) is strict for at least one value of i is routinely satisfied by matrices derived from the idealised forms of many practical problems. If, for example, we put

$$\overline{\mathbf{A}} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{bmatrix}, \quad (11.92)$$

(a simple example that springs from the solution of $\frac{d^2y}{dt^2} = f(t)$ among other problems) and scale it by one-half, the matrix \mathbf{A} thus obtained satisfies equations (11.87) - (11.89) with (11.89) being strict for its first and last rows. Since it is also irreducible (see [198], page 223) it follows that \mathbf{A}^{-1} is given by equation (11.91) and is non-negative.

Consider now the solution of the equation $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n \times n}$ and is nonsingular. If $\mathbf{A} = \mathbf{I} - \mathbf{B}$ we may write this equation

$$\mathbf{x} = \mathbf{Bx} + \mathbf{b}, \quad (11.93)$$

a form that suggests the iteration

$$\mathbf{x}_{i+1} = \mathbf{Bx}_i + \mathbf{b}. \quad (11.94)$$

To determine whether or not this iteration converges it suffices to define the error \mathbf{e}_i by $\mathbf{e}_i = \mathbf{x}_i - \mathbf{x}$, where \mathbf{x} is the solution, and subtract equation (11.93) from equation (11.94). This gives $\mathbf{e}_{i+1} = \mathbf{Be}_i$ so that $\mathbf{e}_{i+1} = \mathbf{B}^i \mathbf{e}_1$ and the iteration converges from any arbitrary initial approximation \mathbf{x}_1 if and only if \mathbf{B} is convergent. This simple iteration is called the *Method of Jacobi* or sometimes the *method of simultaneous displacements*.

Now Jacobi's method is simple and easy to program (but see the next section on (S)SOR preconditioners) but in general its rate of convergence is poor (if indeed it converges at all) since $\rho(\mathbf{B})$ is usually nearly equal to unity. Clearly a more sophisticated method with faster convergence would be desirable. One way of achieving

this, which includes the method of Jacobi as a special case, is based on the idea of a *matrix splitting*.

Definition 8. The expression of $\mathbf{A} \in \mathbb{R}^{n \times n}$ as

$$\mathbf{A} = \mathbf{M} - \mathbf{N} \quad (11.95)$$

where \mathbf{M} is nonsingular (and in practice readily invertible) is called a splitting of \mathbf{A} .

Given such a splitting, the equation $\mathbf{Ax} = \mathbf{b}$ may be written

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{Nx} + \mathbf{M}^{-1}\mathbf{b} \quad (11.96)$$

and since \mathbf{M} is assumed to be readily invertible this suggests that $\mathbf{Ax} = \mathbf{b}$ could perhaps be solved by the iteration

$$\mathbf{x}_{i+1} = \mathbf{M}^{-1}\mathbf{Nx}_i + \mathbf{M}^{-1}\mathbf{b}. \quad (11.97)$$

If we now define as before the error \mathbf{e}_i to be $\mathbf{x}_i - \mathbf{x}$, where \mathbf{x} is the solution of $\mathbf{Ax} = \mathbf{b}$, subtracting equation (11.96) from (11.97) yields

$$\mathbf{e}_{i+1} = \mathbf{M}^{-1}\mathbf{Ne}_i \quad (11.98)$$

so that the above algorithm converges from any arbitrary initial error \mathbf{e}_1 if and only if $\mathbf{M}^{-1}\mathbf{N}$ is convergent. Moreover the smaller we can make $\|\mathbf{N}\|$, the more closely \mathbf{M} approximates \mathbf{A} , the smaller $\rho(\mathbf{M}^{-1}\mathbf{N})$ becomes and the more rapidly the algorithm converges. Thus the “splitting” matrix \mathbf{M} requires just the same properties to be effective when used to solve linear equations by successive approximation as it does when used as a preconditioner for solving the same equations using a Krylov method. This suggests that the splittings that have established themselves in the context of successive approximation could perhaps be impressed into service as Krylov preconditioners, a case in point being the SSOR preconditioner (see below).

Definition 9. The splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ is called a convergent splitting if $\mathbf{M}^{-1}\mathbf{N}$ is convergent.

We now look at three closely-related types of matrix, namely monotone matrices, M-matrices and H-matrices, that have been used in the construction and analysis of preconditioners.

Definition 10. A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is said to be monotone if, for any $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{Ax} \geq \mathbf{0}$ implies $\mathbf{x} \geq \mathbf{0}$.

The notion of a monotone matrix is due to Collatz [74].

Theorem 35. A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is monotone if and only if it is nonsingular and its inverse is non-negative.

Proof. We first prove that if $\mathbf{Ax} \geq \mathbf{0}$ implies that $\mathbf{x} \geq \mathbf{0}$ then \mathbf{A} is nonsingular. Assume that \mathbf{A} is singular so that there exists a vector $\mathbf{z} \neq \mathbf{0}$ such that $\mathbf{Az} = \mathbf{0}$.

Thus $\mathbf{A}\mathbf{z} \geq \mathbf{0}$ but this does not imply that $\mathbf{z} \geq \mathbf{0}$ since if $\mathbf{z} \geq \mathbf{0}$ we merely replace it by $-\mathbf{z}$ to give $\mathbf{A}(-\mathbf{z}) \geq \mathbf{0}$. Hence if $\mathbf{Ax} \geq \mathbf{0}$ implies that $\mathbf{x} \geq \mathbf{0}$, \mathbf{A} must be nonsingular.

We now show that $\mathbf{A}^{-1} \geq \mathbf{O}$. Since \mathbf{A} is nonsingular, put $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$ so that we need to show that $\mathbf{y} \geq \mathbf{0}$ implies $\mathbf{A}^{-1}\mathbf{y} \geq \mathbf{0}$. If we put \mathbf{y} equal to the columns of the unit matrix in turn we see that the implication is only valid if each column in turn of \mathbf{A}^{-1} is non-negative, i.e. if \mathbf{A}^{-1} is itself non-negative. Thus if $\mathbf{Ax} \geq \mathbf{0}$ implies $\mathbf{x} \geq \mathbf{0}$ then \mathbf{A} is nonsingular and \mathbf{A}^{-1} is non-negative.

To prove that if \mathbf{A} is both nonsingular and satisfies $\mathbf{A}^{-1} \geq \mathbf{O}$ then $\mathbf{Ax} \geq \mathbf{0}$ implies $\mathbf{x} \geq \mathbf{0}$ is trivial, completing the proof. ■

It is clear from equation (11.91) and the above theorem that if $\mathbf{A} = \mathbf{I} - \mathbf{B}$, $\mathbf{B} \geq \mathbf{O}$ and $\rho(\mathbf{B}) < 1$ then \mathbf{A} is monotone. Thus the matrix $\overline{\mathbf{A}}$ of equation (11.92) is monotone.

Theorem 36. If \mathbf{A} is monotone, \mathbf{M} is monotone and $\mathbf{M}^{-1}\mathbf{N} \geq \mathbf{O}$ the splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ is convergent.

Proof. Since both \mathbf{A} and \mathbf{M} are monotone by hypothesis they are both nonsingular from Theorem 35 so that

$$\mathbf{A}^{-1} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{N})^{-1}\mathbf{M}^{-1}. \quad (11.99)$$

From Theorem 33 with $\mathbf{B} = (\mathbf{M}^{-1}\mathbf{N})^T$ there exists a nonnegative vector $\mathbf{x} \neq \mathbf{0}$ such that $\mathbf{x}^T\mathbf{M}^{-1}\mathbf{N} = \rho(\mathbf{M}^{-1}\mathbf{N})\mathbf{x}^T$. Pre-multiplying equation (11.99) by \mathbf{x}^T gives

$$\mathbf{x}^T\mathbf{A}^{-1} = \left(\frac{1}{1 - \rho(\mathbf{M}^{-1}\mathbf{N})} \right) \mathbf{x}^T\mathbf{M}^{-1}$$

and since \mathbf{x} , \mathbf{A}^{-1} and \mathbf{M}^{-1} are all non-negative and the two matrices are nonsingular we must have $\rho(\mathbf{M}^{-1}\mathbf{N}) < 1$ to avoid contradiction. The splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ is thus convergent. ■

Corollary 37. If \mathbf{A} is monotone, \mathbf{M} is monotone and $\mathbf{N} \geq \mathbf{O}$ the splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ is convergent.

Proof. If \mathbf{M} is monotone and $\mathbf{N} \geq \mathbf{O}$ then $\mathbf{M}^{-1}\mathbf{N} \geq \mathbf{O}$ and the proof follows from the theorem. ■

Definition 11. A splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ for which \mathbf{M} is monotone and $\mathbf{N} \geq \mathbf{O}$ is called a regular splitting.

To see how the theorem and its corollary may be used, let $\overline{\mathbf{A}}$ be the matrix \mathbf{A} of equation (11.120) where the tridiagonal matrices \mathbf{T} are given by

$$\mathbf{T} = \begin{bmatrix} 4 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 4 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 4 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 4 \end{bmatrix},$$

and let $\mathbf{A} = \frac{1}{4}\mathbf{T}$. This matrix \mathbf{A} then satisfies equation (11.87) and may be expressed as $\mathbf{A} = \mathbf{I} - \mathbf{B}_1 - \mathbf{B}_2$ where \mathbf{B}_1 is bidiagonal with nonzero elements (equal to $\frac{1}{4}$) only in the two diagonals adjacent to the principal diagonal and \mathbf{B}_2 is also bidiagonal but with nonzero elements (again equal to $\frac{1}{4}$) corresponding to the remaining nonzero off-diagonal elements of \mathbf{A} . Thus $\mathbf{I} - \mathbf{B}_1$ is tridiagonal (and hence comparatively easy to invert) and $\|\mathbf{B}_1\|_\infty = \frac{1}{2}$ so that, trivially, from equation (11.91), $\mathbf{I} - \mathbf{B}_1$ is monotone. Hence if we put $\mathbf{M} = \mathbf{I} - \mathbf{B}_1$ and $\mathbf{N} = \mathbf{B}_2$ then two of the conditions of the corollary of Theorem 36 are satisfied.

To show that \mathbf{A} is monotone we note that it may be shown to be irreducible so that, if we put $\mathbf{B} = \mathbf{B}_1 + \mathbf{B}_2$, $\|\mathbf{B}\|_\infty = 1$ and $\rho(\mathbf{B}) < 1$ from Theorem 34. Since $\mathbf{A} = \mathbf{I} - \mathbf{B}$, $\mathbf{B} \geq \mathbf{O}$ and $\rho(\mathbf{B}) < 1$ it follows immediately from equation (11.91) that $\mathbf{A}^{-1} \geq \mathbf{O}$ so that \mathbf{A} is indeed monotone. Thus all three conditions of the corollary are satisfied and the splitting $\mathbf{M} = \mathbf{I} - \mathbf{B}_1$ and $\mathbf{N} = \mathbf{B}_2$ is convergent (the splitting $\mathbf{M} = \mathbf{I} - \mathbf{B}_2$ and $\mathbf{N} = \mathbf{B}_1$ is similarly convergent but much less useful as $\mathbf{I} - \mathbf{B}_2$ is much less readily invertible than $\mathbf{I} - \mathbf{B}_1$).

One way that this splitting could be exploited as a preconditioner is by choosing \mathbf{K} to be $(\mathbf{I} - \mathbf{B}_1)^{-1}$. However since this is in general a full matrix and $\mathbf{I} - \mathbf{B}_1$ is tridiagonal it is simpler, if $\mathbf{I} - \mathbf{B}_1$ is symmetric positive definite, to express it by its Cholesky factors and to compute $\mathbf{K}\mathbf{y}$ for any vector \mathbf{y} by a double-substitution. This gives a particular example of incomplete Cholesky (IC) preconditioning. IC preconditioning is often very effective in reducing the number of Krylov iterations but the sequential nature of the forward- and back-substitutions makes its implementation on parallel or vector computers somewhat inefficient (but see below, page 273).

Another convergent splitting of the same matrix is $\mathbf{M} = \mathbf{I} - \mathbf{L}$ and $\mathbf{N} = \mathbf{U}$, a splitting that is related to the *Gauss-Seidel* method (see equation (11.105) with $\omega = 1$). Although it is possible to base a preconditioner directly on this splitting, a more popular alternative is to start with a somewhat more sophisticated splitting derived from the method of SSOR. Details are given in the section on (S)SOR preconditioners, below.

Now there is nothing in the definition of monotone matrices that imposes conditions on the signs of the elements of \mathbf{A} . In the example of equation (11.92) the diagonal elements are positive and the off-diagonal ones are negative or zero, but these signs could easily be reversed as may be seen from the following example for which

$$\mathbf{A} = \begin{bmatrix} -1 & 2 \\ 2 & -1 \end{bmatrix} \quad \text{and} \quad \mathbf{A}^{-1} = \begin{bmatrix} 1 & 2 \\ \frac{3}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{1}{2} \\ \frac{3}{2} & \frac{3}{2} \end{bmatrix}. \quad (11.100)$$

Moreover if this matrix is scaled by -1 and expressed as $\mathbf{I} - \mathbf{B}$ then $\rho(\mathbf{B}) = 2$ and equation (11.91) does not hold. Perhaps for these reasons the class of monotone matrices is considered too broad to be of much practical value without further restrictions. These restrictions are introduced in the case of M-matrices by imposing sign requirements on the off-diagonal elements.

Definition 12. A matrix $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$ is called an M-matrix if it is monotone and if $a_{ij} \leq 0$, $i \neq j$.

Note that even in this definition the signs of the diagonal elements of \mathbf{A} remain unspecified, but a brief consideration of the identity $\mathbf{A}\mathbf{A}^{-1} \equiv \mathbf{I}$ indicates that for an M-matrix they must be strictly positive. Clearly the matrix $\overline{\mathbf{A}}$ of equation (11.92) is an M-matrix¹⁶ while that of equation (11.100) is not, despite it being monotone.

Although M-matrices have several useful properties they are somewhat restricted, so it was not surprising that they would be generalised by relaxing some of the sign constraints and discarding the idea that they should be monotone. One particular way of doing this was via H-matrices, which may be defined informally as matrices that can be transformed into M-matrices by reversing the signs of some or all of their off-diagonal elements. More formally we have

Definition 13. Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n} = [a_{ij}], [b_{ij}]$, let $b_{ii} = a_{ii}$ and $b_{ij} = -|a_{ij}|$, $i \neq j$. Then \mathbf{A} is an H-matrix if \mathbf{B} is an M-matrix.

H-matrices have many of the properties of M-matrices and both these types of matrix have been used to validate certain IC and ILU preconditioners. As we shall see, these preconditioners are among the more effective for reducing the number of Krylov iterations.

For a fuller discussion of non-negative matrices see e.g. Horn and Johnson [155], Chapter 8.

11.6. (S)SOR preconditioners

The actual formulation of these two preconditioners is so bizarre that they would be unlikely to be recognized as such without some foreknowledge. This is particularly true of the SOR preconditioner. This matrix is lower triangular and yet is supposed to be a good approximation to a (possibly) symmetric positive definite matrix. To see how this state of affairs came about it is necessary to go back to the days when the linear equations derived from elliptic partial differential equations in two space variables were solved by hand with the help of electric calculating machines, and to look at other iterative methods besides conjugate gradients.

The simplest splitting of equation (11.86) is to put $\mathbf{M} = \mathbf{I}$ and $\mathbf{N} = \mathbf{L} + \mathbf{U}$, and the algorithm based on this splitting is called the *Jacobi method*, or sometimes the

¹⁶It is also, for good measure, a Stieltjes matrix (a symmetric M-matrix).

method of *simultaneous displacements*. From equation (11.97) it may be expressed as

$$\mathbf{x}_{i+1} = (\mathbf{L} + \mathbf{U}) \mathbf{x}_i + \mathbf{b}$$

or

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{r}_i \quad (11.101)$$

where \mathbf{r}_i is the vector of residuals given by

$$\mathbf{r}_i = \mathbf{A}\mathbf{x}_i - \mathbf{b}. \quad (11.102)$$

The matrix $\mathbf{B} = \mathbf{L} + \mathbf{U}$ is referred to as the *Jacobi operator* and the method converges if $\rho(\mathbf{B}) < 1$. This condition is usually (but not always) satisfied if conditions (11.87) - (11.89) hold for the original matrix.

Now as the alternative name of the Jacobi method emphasises, every element of the new vector \mathbf{x}_{i+1} is first computed in terms of all the elements of \mathbf{x}_i and the latter are then replaced lock, stock and barrel by the new elements. This procedure is somewhat awkward when carrying out hand calculation where the simplest approach is the replacement of each individual element as soon as it is re-computed. Hence let $\mathbf{x} = [x_k]$ denote the current approximate solution and compute the next approximation $\mathbf{x}' = [x'_k]$ by subtracting from x_j the j th element of the residual \mathbf{r} corresponding to \mathbf{x} . This is the single-element analogue of equation (11.101) and may be expressed as

$$\mathbf{x}' = \mathbf{x} - \mathbf{e}_j r_j, \quad (11.103)$$

where \mathbf{e}_j denotes the j -th column of the identity matrix and r_j denotes the j th element of $\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b}$. Since $\mathbf{r}' = \mathbf{A}\mathbf{x}' - \mathbf{b}$, equation (11.86) gives

$$\mathbf{r}' = \mathbf{r} - (\mathbf{I} - \mathbf{L} - \mathbf{U}) \mathbf{e}_j r_j$$

so that

$$r'_j = \mathbf{e}_j^T \mathbf{r}' = \mathbf{e}_j^T (\mathbf{L} + \mathbf{U}) \mathbf{e}_j r_j = 0$$

since all the diagonal elements of $\mathbf{L} + \mathbf{U}$ are zero. Thus the effect of carrying out a Jacobi-type update on just one element of \mathbf{x} is to set the corresponding element of the residual vector equal to zero, and would have been seen as a perfectly natural step by those engaged in hand calculation. It was indeed the basic step of that occupation and came to be known as a relaxation step since zeroing the residual was seen, in a sense, to be reducing the overall "stress" of the approximate solution.

Now the hand-calculators used various devices to accelerate convergence. If, for example, a particular sub-group of equations proved to be particularly troublesome this group could be processed several times over before attention was turned to the other equations. Usually the largest residual was relaxed at any one time although this would be changed if it were thought desirable. It was also found that, in some circumstances, over- or under-correcting would result in very considerable improvements in the rate of convergence and experienced problem-solvers developed a flair

for knowing just what tricks to apply and when to apply them (see [223] for further details). However the electronic computer could not cope with such undisciplined intelligence and when these techniques were mechanised they had to be made more rigid and susceptible to theoretical investigation. Art was transformed into Science.

One change that was made was to update each element of the solution just once at every iteration, regardless of how troublesome the corresponding equation happened to be. An “order of solution” would be selected that would remain unchanged throughout the iteration and the equations would always be relaxed in this order. Symbolically the matrix \mathbf{A} would be replaced by $\mathbf{P}^T \mathbf{A} \mathbf{P}$, where \mathbf{P} was a permutation matrix chosen to effect the required reordering and where the post-multiplication by \mathbf{P} was carried out not to maintain symmetry (since \mathbf{A} is not necessarily symmetric) but to keep the diagonal elements of \mathbf{A} on the diagonal. Thus if \mathbf{A} satisfies equation (11.86) then so does $\mathbf{P}^T \mathbf{A} \mathbf{P}$, but with different values for \mathbf{L} and \mathbf{U} . It turns out that the ordering of the equations is a crucial element of one particular branch of the theory (see below).

The other major change when passing from hand to machine calculation was the amount of over-relaxation that could be applied. Instead of it varying for different elements and from one iteration to another it was now, in general, held constant for the whole solution process. However the one thing that did not change in the transition from hand computing was the replacing of each element of \mathbf{x}_i by its updated value as soon as it was computed.

Let then $\mathbf{x}_i = [x_k^{(i)}]$ denote the current approximate solution and $\mathbf{x}_{i+1} = [x_k^{(i+1)}]$ the approximation about to be computed. Then \mathbf{x}_{i+1} is given by (c.f. equation (11.101))

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \omega \mathbf{s} \quad (11.104)$$

where ω is an arbitrary constant and $\mathbf{s} = [s_j]$ denotes not the residual $\mathbf{Ax}_i - \mathbf{b}$ but a residual computed partly from the elements of \mathbf{x}_i and partly from those of \mathbf{x}_{i+1} . If the first $(j-1)$ elements of \mathbf{x}_{i+1} have already been computed, s_j will be given by

$$s_j = \sum_{k=1}^{j-1} a_{jk} x_k^{(i+1)} + \sum_{k=j}^n a_{jk} x_k^{(i)} - b_j.$$

Thus, from equation (11.86),

$$\mathbf{s} = (\mathbf{I} - \mathbf{U}) \mathbf{x}_i - \mathbf{L} \mathbf{x}_{i+1} - \mathbf{b}$$

so that, from equation (11.104),

$$(\mathbf{I} - \omega \mathbf{L}) \mathbf{x}_{i+1} = [(1 - \omega) \mathbf{I} + \omega \mathbf{U}] \mathbf{x}_i + \omega \mathbf{b}. \quad (11.105)$$

The scalar ω is known as the *relaxation factor* and the method based on equation (11.105) is called the method of *successive over-relaxation* or *SOR* if, as is usually the case, $\omega > 1$. If $\omega = 1$ it is known as the *Gauss-Seidel method* or sometimes

the *method of successive displacements*. It follows from equation (11.105) that SOR converges if the spectral radius of

$$\mathbf{H} \equiv (\mathbf{I} - \omega \mathbf{L})^{-1} [(1 - \omega) \mathbf{I} + \omega \mathbf{U}] \quad (11.106)$$

is less than unity. Comparison of equations (11.97) and (11.106) show that SOR is equivalent to successive approximation based on a splitting of $\omega \mathbf{A}$ defined by $\mathbf{M} = \mathbf{I} - \omega \mathbf{L}$ and $\mathbf{N} = (1 - \omega) \mathbf{I} + \omega \mathbf{U}$. Thus a possible preconditioner for $\mathbf{Ax} = \mathbf{b}$ based on SOR is $\omega^{-1} \mathbf{M} = \omega^{-1} \mathbf{I} - \mathbf{L}$.

Let now \mathbf{z} be an eigenvector of \mathbf{H} with an eigenvalue λ so that $\mathbf{Hz} = \mathbf{z}\lambda$. If $\lambda \neq 0$, premultiplication of this equation by $(\mathbf{I} - \omega \mathbf{L})$ gives, from equation (11.106) and after some rearrangement,

$$\left(\lambda^{\frac{1}{2}} \mathbf{L} + \lambda^{-\frac{1}{2}} \mathbf{U} \right) \mathbf{z} = \mathbf{z} \left(\frac{\lambda + \omega - 1}{\omega \lambda^{\frac{1}{2}}} \right). \quad (11.107)$$

Without further assumptions about the nature of \mathbf{A} it is difficult to proceed further, but two cases may be mentioned. If \mathbf{A} is symmetric and positive definite then \mathbf{H} is convergent for $0 < \omega < 2$. We give now an outline proof.

Let \mathbf{z} be a (possibly complex) normalised eigenvector of \mathbf{H} corresponding to the eigenvalue λ . Premultiplication of equation (11.107) by $\lambda^{\frac{1}{2}} \mathbf{z}^H$, where the superscript H denotes the conjugate transpose, gives

$$\lambda \mathbf{z}^H \mathbf{L} \mathbf{z} + \mathbf{z}^H \mathbf{L}^T \mathbf{z} = \frac{\lambda + \omega - 1}{\omega}.$$

Let now

$$\mathbf{z}^H \mathbf{L} \mathbf{z} = \alpha + i\beta \quad \text{so that} \quad \mathbf{z}^H \mathbf{L}^T \mathbf{z} = \alpha - i\beta \quad (11.108)$$

where α and β are real. Substituting these values in the previous equation gives

$$\lambda \left(\alpha - \frac{1}{\omega} + i\beta \right) = 1 - \alpha - \frac{1}{\omega} + i\beta.$$

Assume now that $|\lambda| \geq 1$. Then

$$\left| \alpha - \frac{1}{\omega} + i\beta \right| \leq \left| 1 - \alpha - \frac{1}{\omega} + i\beta \right|$$

so that, since both expressions have a common imaginary part,

$$\left| \alpha - \frac{1}{\omega} \right| \leq \left| 1 - \alpha - \frac{1}{\omega} \right|.$$

Squaring this and simplifying gives

$$\frac{2}{\omega} (1 - 2\alpha) \leq (1 - 2\alpha). \quad (11.109)$$

Now \mathbf{A} is assumed to be positive definite so that $\mathbf{z}^H \mathbf{A} \mathbf{z} > 0$. But, from equation (11.86) and symmetry, $\mathbf{A} = \mathbf{I} - \mathbf{L} - \mathbf{L}^T$ so that, from equation (11.108), $1 - 2\alpha > 0$.

Hence, from inequality (11.109), $2/\omega \leq 1$ so that either $\omega < 0$ or $\omega \geq 2$. Thus if $0 < \omega < 2$, $|\lambda| < 1$ and SOR converges. The result that if $\omega < 0$ or $\omega > 2$ then $|\lambda| > 1$ can be proved similarly.

Another special case arises when \mathbf{A} has *property A* and is *consistently ordered*.

Definition 14. The matrix $\mathbf{A} = \mathbf{I} - \mathbf{L} - \mathbf{U}$ is said to have property A if there exists a permutation matrix \mathbf{P} such that

$$\mathbf{P}^T \mathbf{A} \mathbf{P} = \begin{bmatrix} \mathbf{I} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{I} \end{bmatrix}. \quad (11.110)$$

If, in addition, the eigenvalues of

$$\mathbf{J}(\alpha) \equiv \alpha \mathbf{L} + \alpha^{-1} \mathbf{U},$$

where $\alpha \neq 0$ is an arbitrary scalar, are independent of α then the matrix is said to be consistently ordered.

Matrices possessing property A are not necessarily consistently ordered but any such matrix may be re-ordered to achieve this by identical row-column permutations. Indeed the ordering of $\mathbf{P}^T \mathbf{A} \mathbf{P}$ as defined by equation (11.110) is consistent. Since $\mathbf{A} = \mathbf{I} - \mathbf{L} - \mathbf{U}$ we have

$$\alpha \mathbf{L} + \alpha^{-1} \mathbf{U} = - \begin{bmatrix} \mathbf{O} & \alpha^{-1} \mathbf{A}_{12} \\ \alpha \mathbf{A}_{21} & \mathbf{O} \end{bmatrix}$$

and

$$\begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \alpha \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{O} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \alpha^{-1} \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{O} & \alpha^{-1} \mathbf{A}_{12} \\ \alpha \mathbf{A}_{21} & \mathbf{O} \end{bmatrix}.$$

Thus for any matrix having the form of equation (11.110), the matrix $\mathbf{L} + \mathbf{U}$ may be transformed to $\alpha \mathbf{L} + \alpha^{-1} \mathbf{U}$ by a simple diagonal similarity transformation. Since these transformations preserve eigenvalues this implies that such matrices are consistently ordered. Essentially the same proof of the invariance of the eigenvalues, using somewhat more elaborate diagonal matrices, holds for other consistent orderings [43]. Now if the eigenvalues of $\mathbf{J}(\alpha)$ are independent of α , the eigenvalues of $\lambda^{\frac{1}{2}} \mathbf{L} + \lambda^{-\frac{1}{2}} \mathbf{U}$ are equal to those of $\mathbf{L} + \mathbf{U}$, the Jacobi operator. If these are denoted by μ we have, from equation (11.107),

$$\lambda + \omega - 1 = \mu \omega \lambda^{\frac{1}{2}}, \quad (11.111)$$

one of the most famous equations in the theory of linear iterative methods. It was first given in the form $(\lambda + \omega - 1)^2 = \mu^2 \omega^2 \lambda$ by David Young [259] in 1954.

11.6.1. SSOR

In the method of SOR the equations are “relaxed” in the order that their coefficients appear in the matrix \mathbf{A} , starting with the first and ending with the last. An

alternative version of this scheme is the relaxation of the equations in the reverse order. If \mathbf{A} is given by equation (11.86) then it is straightforward to show that the change in the approximate solution due to this operation is given by

$$(\mathbf{I} - \omega\mathbf{U})\mathbf{x}_{i+2} = [(1 - \omega)\mathbf{I} + \omega\mathbf{L}]\mathbf{x}_{i+1} + \omega\mathbf{b} \quad (11.112)$$

which is simply equation (11.105) with \mathbf{L} and \mathbf{U} interchanged and i replaced by $i+1$. In the method of SSOR, forward and reverse steps alternate so that if \mathbf{x}_{i+1} were computed by equation (11.105) then \mathbf{x}_{i+2} would be computed by equation (11.112). Premultiplying this latter equation by $(\mathbf{I} - \omega\mathbf{L})$ gives, from equation (11.105),

$$\begin{aligned} & (\mathbf{I} - \omega\mathbf{L})(\mathbf{I} - \omega\mathbf{U})\mathbf{x}_{i+2} = \\ & [(1 - \omega)\mathbf{I} + \omega\mathbf{L}][(1 - \omega)\mathbf{I} + \omega\mathbf{U}]\mathbf{x}_i + \omega(2 - \omega)\mathbf{b}. \end{aligned}$$

Comparison with equation (11.97) shows that this is just the method of successive approximation applied to the equation $\mathbf{Ax} = \mathbf{b}$ with the splitting defined by

$$\mathbf{M} = \frac{(\mathbf{I} - \omega\mathbf{L})(\mathbf{I} - \omega\mathbf{U})}{\omega(2 - \omega)} \quad (11.113)$$

and

$$\mathbf{N} = \frac{[(1 - \omega)\mathbf{I} + \omega\mathbf{L}][(1 - \omega)\mathbf{I} + \omega\mathbf{U}]}{\omega(2 - \omega)}.$$

Since for symmetric systems ($\mathbf{U} = \mathbf{L}^T$) SSOR often works well, and since the desirable characteristics of \mathbf{M} when used in successive approximations are the same as those required of a preconditioner, it is not surprising that the above value of \mathbf{M} has been frequently used as a preconditioner for CG.

11.7. ILU preconditioning

It is always possible in principle to solve any system of linear equations $\mathbf{Ax} = \mathbf{b}$ by Gaussian elimination or triangular decomposition in which the coefficient matrix \mathbf{A} is expressed, either implicitly or explicitly, as the product \mathbf{LU} of lower and upper triangular factors. The difficulty that arises when this technique is applied to large sparse systems is that generally the factors have far more nonzero elements than the original matrix. This means that more storage space is needed in the computer and that extra time is needed to process the extra nonzeros. Indeed it was for these reasons that iterative algorithms and methods of CG-type were developed in the first place. However LU decomposition is such a reliable algorithm and since, as we have seen, preconditioners should resemble \mathbf{A} as much as possible it was natural to use the product of triangular factors as a preconditioner. Since \mathbf{A} is large and sparse, in order to reduce the workload it was natural to require that these factors should also be sparse. There is, though, one caveat. ILU preconditioners are difficult to implement efficiently on vector and parallel machines since both the construction

of the factors and their use within a step of an iterative method are inherently sequential operations.

ILU factorisations were originally developed for solving finite difference approximations to elliptic partial differential equations (see Buleev [53], 1960, Varga [245], 1960 and Oliphant [194], 1961 and [195], 1962). These approximations were based on the familiar five-point discretisation of $\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2}$ on a rectangular grid (see e.g. [134]) and gave rise to matrices that were usually symmetric and positive definite with the nonzero elements appearing in certain well-defined patterns. In view of the importance of the original problems it was scarcely surprising that the early incomplete factorisations exploited this specialised structure and that the factors thus obtained were also highly structured. Indeed for some problems it is even possible to express the condition number of the preconditioned coefficient matrix in terms of h , the mesh constant of the original finite difference approximation of $\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2}$ (see [138] and section 11.2 of [134]). While this work is undoubtedly extremely important and leads to significant algorithmic improvements we regard it as being too restricted to a specific type of problem for further discussion here.

The first use of approximate triangular factors as conjugate gradient preconditioners was due to Meijerink and van der Vorst ([183], 1977) who also generalized the earlier work to matrices with arbitrary sparsity patterns. To calculate the approximate factors for such matrices it is necessary to know:

- (1) how to choose the pattern of zero/nonzero elements for both \mathbf{L} and \mathbf{U} , and
- (2) how to compute the values of the nonzero elements.

A simple, if not always effective, solution to both these problems is to choose the pattern of zeros arbitrarily (save that the diagonal elements of both factors must always be non-zero) and to compute the values of the non-zero elements according to the following strategy:

Let $\mathbf{L} = [l_{ij}]$ and $\mathbf{U} = [u_{ij}]$, and let \mathcal{Z} denote the set of ordered pairs of integers (i, j) for which l_{ij} ($i > j$) and u_{ij} ($i < j$) are required to be zero. We refer to \mathcal{Z} as a *valid* zero set for $\mathbf{A} \in \mathbb{R}^{n \times n}$ if $1 \leq i, j \leq n$ and $i \neq j$. Define a residual matrix $\mathbf{R} = [r_{ij}]$ by

$$\mathbf{R} = \mathbf{LU} - \mathbf{A}. \quad (11.114)$$

Then the nonzero elements of \mathbf{L} and \mathbf{U} are chosen so that $r_{ij} = 0$ for $(i, j) \notin \mathcal{Z}$. This expression of an approximation to \mathbf{A} as the product of sparse factors is known as *incomplete LU (ILU) factorisation*, and its mechanics are examined in more detail in the following section. To see that it represents a valid computational option it is probably easiest to write the equation defining \mathbf{R} as

$$\mathbf{LU} = \mathbf{A} + \mathbf{R} \quad (11.115)$$

and note that if this equation is satisfied by the matrices concerned then, since \mathbf{L} and \mathbf{U} are triangular, it is also satisfied by their leading principal submatrices. If we denote the leading principal submatrices of order k by the subscript k and apply the equation to submatrices of order $k+1$ we have, with obvious notation,

$$\begin{bmatrix} \mathbf{L}_k & \mathbf{0} \\ \mathbf{p}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_k & \mathbf{v} \\ \mathbf{0}^T & \omega \end{bmatrix} = \begin{bmatrix} \mathbf{A}_k & \mathbf{b} \\ \mathbf{c}^T & \alpha \end{bmatrix} + \begin{bmatrix} \mathbf{R}_k & \mathbf{s} \\ \mathbf{q}^T & 0 \end{bmatrix}. \quad (11.116)$$

Thus

$$\mathbf{L}_k \mathbf{v} = \mathbf{b} + \mathbf{s} \quad (11.117)$$

and we compute \mathbf{v} by first setting its required elements to zero and then computing the remainder in the normal way by forward substitution, beginning with the first unknown element and finishing with the last. Thus, if v_i is not assigned to be zero, it is computed by

$$v_i = b_i - \sum_{j=1}^{i-1} l_{ij} v_j. \quad (11.118)$$

This implies, from equation (11.117), that $s_i = 0$ so that, if $(i, j) \notin \mathcal{Z}$, $r_{ij} = 0$. The vector \mathbf{p}^T is computed in precisely the same way and ω is given by $\omega = \alpha - \mathbf{p}^T \mathbf{v}$. Provided that no value of ω is zero (and this cannot always be guaranteed even if the leading principal submatrices of \mathbf{A} are all nonsingular) we can obtain an incomplete LU decomposition. If $\mathbf{A} = [a_{ij}]$ and \mathcal{Z} consists of the (i, j) pairs for which $a_{ij} = 0$ then we have the decomposition that is referred to as ILU(0), the reason for the zero becoming apparent later. This particular choice is the “no fill-in” option since the total number of significant nonzero elements of \mathbf{L} and \mathbf{U} is equal to that of \mathbf{A} provided that no diagonal element of \mathbf{A} is zero.

11.7.1. Incomplete Cholesky (IC) preconditioning

If \mathbf{A} is symmetric and positive definite it is sometimes possible to carry out the incomplete factorisation

$$\mathbf{L} \mathbf{L}^T = \mathbf{A} + \mathbf{R} \quad (11.119)$$

where $\mathbf{L} \in \mathbb{R}^{n \times n}$ has a given sparsity pattern and may be computed analogously to the \mathbf{L} of equation (11.115). This is known as *incomplete Cholesky (IC) factorisation*. The existence of such factorisations, though, is not guaranteed in real terms even if \mathbf{A} is positive definite. A further condition is needed.

One such condition is that \mathbf{A} is an M-matrix. Meijerink and van der Vorst [183] showed that ILU factorisation is possible for *any* M-matrix \mathbf{A} and *any* valid set of zero elements defined by \mathcal{Z} . They further showed that if \mathbf{A} is a symmetric M-matrix then IC factorisation is similarly always possible. Moreover if $\mathbf{M} = \mathbf{L}\mathbf{U}$ and $\mathbf{N} = \mathbf{R}$ so that equation (11.115) defines a matrix splitting in the sense of Definition 8, then this splitting is convergent. These results were extended to H-matrices by Manteuffel [182]. Thus if \mathbf{A} is an M-matrix or an H-matrix, not only is incomplete factorisation always possible but this factorisation will provide a workable preconditioner. Since many important applications derived from partial differential equations give rise to just such matrices, the results of Meijerink, van der Vorst and Manteuffel are of very considerable practical and theoretical importance.

Despite its simplicity, ILU(0) has had some success in solving practical problems. Numerical experiments reported in [183] compared IC(0) (referred to therein as ICCG(0)) with unpreconditioned conjugate gradients, the strongly implicit procedure (*SIP*) of Stone [228] and a successive line over-relaxation method. The two test problems considered arose from five-point approximations to a second-order self-adjoint elliptic partial differential equation over a square region, and had dimensions 992 and 1849 respectively. The coefficient matrices $\mathbf{A} = [a_{ij}]$ were symmetric positive definite, diagonally dominant and block tridiagonal, having the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{T} & -\mathbf{I} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{O} \\ -\mathbf{I} & \mathbf{T} & -\mathbf{I} & \cdots & \mathbf{O} & \mathbf{O} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \cdots & -\mathbf{I} & \mathbf{T} \end{bmatrix} \quad (11.120)$$

where $\mathbf{I}, \mathbf{T} \in \mathbb{R}^{m \times m}$ and \mathbf{T} is itself tridiagonal. The amount of work required by each method was expressed in terms of “equivalent iterations”.

For the first test example, IC(0) needed 45 equivalent iterations to reduce the true residual norm to 10^{-10} while unpreconditioned CG only reduced the residual norm to just below 10^{-2} after 50 equivalents. For the second example, IC(0) needed 45 equivalents to reduce the residual norm to 10^{-7} whereas the other methods failed to reduce the residual norm to less than 10^{-2} even after some 60 equivalents. Thus for these examples the simple IC(0) preconditioner made all the difference between getting, or not getting, a solution. The numerical comparisons only refer to the iterative phase of the algorithms. The fixed costs of computing the preconditioners were not given on the grounds that they would be amortized if many similar problems were to be solved. They represent in any case only a comparatively small part of the total cost of a single calculation when carried out on serial machines.

A variant of the symmetric form of this algorithm, due to Manteuffel [182], is the *shifted IC (SIC) algorithm*. The matrix \mathbf{A} has to be symmetric and is assumed here for simplicity to be symmetrically scaled so that it satisfies

$$\mathbf{A} = \mathbf{I} - \mathbf{L} - \mathbf{L}^T. \quad (11.121)$$

Define the matrices $\mathbf{B} = [b_{ij}]$ and $\overline{\mathbf{A}}$ by $\mathbf{B} = \mathbf{L} + \mathbf{L}^T$ and

$$\overline{\mathbf{A}} = \mathbf{I} - \left(\frac{1}{1 + \alpha} \right) \mathbf{B} \quad (11.122)$$

for some scalar parameter α . Clearly, if $(1 + \alpha) > \rho(\mathbf{B})$ then $\overline{\mathbf{A}}$ is positive definite and equally clearly if $(1 + \alpha) > \rho(|\mathbf{B}|)$, where $|\mathbf{B}| = [|b_{ij}|]$, then $\overline{\mathbf{A}}$ is an H-matrix. Assume that this latter is the case. It was then shown by Manteuffel that for any valid zero-set \mathcal{Z} it is possible to find a unit lower triangular factor $\overline{\mathbf{L}}$ and positive definite diagonal matrix \mathbf{D} such that

$$\overline{\mathbf{A}} = \overline{\mathbf{L}} \mathbf{D} \overline{\mathbf{L}}^T - \overline{\mathbf{R}} \quad (11.123)$$

where $\overline{\mathbf{L}} = [\bar{l}_{ij}]$, $\bar{l}_{ij} = 0$ both for $i < j$ and for $(i, j) \in \mathcal{Z}$, $i \neq j$, and $\overline{\mathbf{R}} = [\bar{r}_{ij}]$, $\bar{r}_{ij} = 0$ for $(i, j) \notin \mathcal{Z}$. Now equations (11.121) and (11.122) yield

$$\mathbf{A} = \bar{\mathbf{A}} - \left(\frac{\alpha}{1 + \alpha} \right) \mathbf{B}$$

so that, from equation (11.123),

$$\mathbf{A} = \bar{\mathbf{L}} \mathbf{D} \bar{\mathbf{L}}^T - \mathbf{R} \quad (11.124)$$

where

$$\mathbf{R} = \bar{\mathbf{R}} + \left(\frac{\alpha}{1 + \alpha} \right) \mathbf{B}.$$

Equation (11.124) defines a splitting of \mathbf{A} and the matrix $\bar{\mathbf{L}} \mathbf{D} \bar{\mathbf{L}}^T$ may be used as a preconditioner for \mathbf{A} provided that $\|\mathbf{R}\|$ is not too large.

Manteuffel tested the algorithm on various problems with \mathbf{A} symmetric and (effectively) scaled to satisfy equation (11.121). He then compared CG preconditioned by SIC with straight CG. Typically, using SIC, the time spent in the iterative phase of the algorithm was reduced by about half.

Another IC preconditioner was proposed by Ajiz and Jennings [3]. Their chosen method of dropping yields, in equation (11.119), a matrix \mathbf{R} that is positive semi-definite even though the original matrix is neither an M- nor an H-matrix. This preconditioner is thus always stable and it can be shown that if \mathbf{S} is as defined by equation (11.146) then $\|\mathbf{S}\| < 1$ and the corresponding splitting $\mathbf{A} \equiv (\mathbf{A} + \mathbf{R}) - \mathbf{R}$ is convergent. Despite this and the method's popularity for certain types of problem, it can sometimes give a preconditioner that requires far more iterations than other versions of ICT (see [23]). However if reliability is of paramount importance, this method is clearly a strong contender.

11.7.2. DCR

In the above algorithm the modification of the matrix \mathbf{A} is carried out dynamically during the factorisation itself. It is possible though to do this at the outset, one approach being the *diagonally-compensated reduction* method of Axelsson and Koltolilina [14]. In one version of this, positive off-diagonal elements are set to zero and at the same time added to the corresponding diagonal elements, an operation that preserves positive definiteness. The resulting matrix is positive definite with non-positive off-diagonal elements and is hence a Stieltjes matrix from which, when performing IC decomposition, off-diagonal elements may be dropped with impunity. The method works well for moderately difficult problems but becomes less effective as the difficulty increases (see [23] and the references therein cited).

11.7.3. ILU(p)

The no-fill ILU and IC preconditioners are very simple to implement, cheap to compute and quite effective for the type of problems for which they were originally proposed. However for more difficult problems, i.e. for the highly nonsymmetric and

indefinite matrices arising in computational fluid dynamics and similar applications, the no-fill factorizations result in too crude an approximation of \mathbf{A} . More sophisticated preconditioners are needed which allow the incomplete factors to have some fill-in. If the full triangular factors \mathbf{L} and \mathbf{U} are determined for a sparse matrix it is often found that not only is there a considerable amount of fill-in when computing the factors but that many of these introduced nonzero elements are quite small, so small in fact that their individual calculation is not cost-effective. We now we describe a method due to Watts [254] of estimating the magnitude of the elements of \mathbf{L} and \mathbf{U} and discarding those that are smaller than a certain perceived level. This is done essentially by approximating each element of \mathbf{A} , \mathbf{L} and \mathbf{U} by ε^j , where $0 < \varepsilon < 1$ and j is an integer known as the *level of fill*. The level of fill is essentially a measure of the order of magnitude of an element (expressed in numbers of base ε) and is clearly logarithmic in nature. The quantity ε is notional and does not require a numerical value. Interestingly, Watts used the term “order of magnitude” rather than “level-of-fill”, and referred to the calculation of the approximate Cholesky factors as “truncated direct elimination”, admirably precise and descriptive terms. We, however, follow the conventions and notation used by Saad [108] as these are probably more familiar than the original terminology.

The values of the level of fill for the elements of \mathbf{L} and \mathbf{U} are determined by applying to \mathbf{A} a symbolic variant of Gaussian elimination which uses a simplified arithmetic, and we first establish the rules for this arithmetic. Denote the level of fill of α by $lev(\alpha)$. If we regard two elements having the same level of fill as having the same order of magnitude, their sum or difference may also be thought of as having the same order of magnitude so that, if $lev(\alpha) = lev(\beta)$,

$$lev(\alpha \pm \beta) = lev(\alpha) = lev(\beta).$$

Clearly this ignores the possibility of order-of-magnitude reductions by cancellation. If α and β have different levels of fill then since $|\varepsilon| < 1$ the element having the higher level may be regarded as being negligible compared with the other so that

$$lev(\alpha \pm \beta) = \min(lev(\alpha), lev(\beta)). \quad (11.125)$$

Since this equation includes the previous one as a special case it may be taken as the definition of addition and subtraction. The rules for multiplication and division follow immediately from the identification of the matrix elements with ε^j and are $lev(\alpha * \beta) = lev(\alpha) + lev(\beta)$ and

$$lev(\alpha / \beta) = lev(\alpha) - lev(\beta). \quad (11.126)$$

These, then, are the rules of the game. To see how it is played we first need to consider the numerical algorithm on which it is based. If $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$ and $a_{ij} = a(i, j)$ then this algorithm is:

The simple triangularisation (ST) algorithm

- (1) for $k = 1 : n - 1$ do

```

(a) for  $i = k + 1 : n$  do
    (i) for  $j = k + 1 : n$  do
        (A)  $a(i, j) := a(i, j) - a(i, k) * a(k, j) / a(k, k);$ 
    (ii) end
(b) end
(2) end simple triangularisation algorithm.

```

This algorithm is clearly related to Gaussian elimination and is, in fact, a “symmetrized” form of that algorithm in which rows and columns are treated identically (in Gaussian elimination the strictly lower triangular elements of the final matrix are replaced by zeros and in LU decomposition by the appropriate multipliers). This identical treatment is important since failure to implement it could lead to inconsistencies when the numerical operations of the algorithm are replaced by symbolic ones.

To see what the ST algorithm does, denote the final computed matrix by $\mathbf{M} = [m_{ij}]$ and let $\mathbf{M} = \mathbf{D} + \mathbf{B} + \mathbf{C}$ where \mathbf{D} is diagonal and \mathbf{B} and \mathbf{C} are respectively strictly lower and strictly upper triangular. A simple comparison indicates that $\mathbf{D} + \mathbf{C}$ is precisely the upper triangular matrix \mathbf{U} that would be computed by Gaussian elimination. Thus (see e.g. [129] or [258]) $\mathbf{A} = \mathbf{LU}$ where \mathbf{L} is some unit lower triangular matrix. Now since the ST algorithm is symmetric (even if the matrices to which it is applied are not) it is straightforward to show (imagine applying the ST algorithm to \mathbf{A}^T) that $\mathbf{D} + \mathbf{B}$ is some lower triangular matrix \mathbf{L}_1 where $\mathbf{A} = \mathbf{L}_1 \mathbf{U}_1$ and \mathbf{U}_1 is upper *unit* triangular. Thus $\mathbf{D} + \mathbf{M} = \mathbf{U} + \mathbf{L}_1$ and $\mathbf{L}_1 \mathbf{U}_1 = (\mathbf{L}_1 \mathbf{D}^{-1})(\mathbf{D} \mathbf{U}_1) = \mathbf{LU}$. Now LU decomposition is unique and it follows from this that, since both \mathbf{L} and $\mathbf{L}_1 \mathbf{D}^{-1}$ are unit lower triangular, $\mathbf{L}_1 \mathbf{D}^{-1} = \mathbf{L}$. Thus $\mathbf{D} + \mathbf{M} = \mathbf{U} + \mathbf{LD}$ or

$$\mathbf{M} = (\mathbf{L} - \mathbf{I}) \mathbf{D} + \mathbf{U} \quad (11.127)$$

where \mathbf{L} and \mathbf{U} are the factors of \mathbf{A} that would be obtained by LU decomposition. The off-diagonal elements of \mathbf{M} comprise essentially the “strictly triangular” parts of the triangular factors. If we knew their orders of magnitude before carrying out incomplete LU decomposition we could choose \mathcal{Z} to eliminate the smaller ones and obtain a simplified approximate factorisation.

To illustrate this by a simple example, if

$$\mathbf{A} = \begin{bmatrix} 8 & 0 & 4 & 4 \\ 4 & 8 & 4 & 0 \\ 0 & 4 & 8 & 0 \\ 4 & 4 & 0 & 8 \end{bmatrix} \quad \text{then} \quad \mathbf{M} = \begin{bmatrix} 8 & 0 & 4 & 4 \\ 4 & 8 & 2 & -2 \\ 0 & 4 & 7 & 1 \\ 4 & 4 & -3 & \frac{52}{7} \end{bmatrix}$$

as may readily be verified by applying the ST algorithm to \mathbf{A} . It then follows from the above discussion that

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & -\frac{3}{7} & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 8 & 0 & 4 & 4 \\ 0 & 8 & 2 & -2 \\ 0 & 0 & 7 & 1 \\ 0 & 0 & 0 & \frac{52}{7} \end{bmatrix} \quad (11.128)$$

from which it is simple to confirm that $\mathbf{LU} = \mathbf{A}$.

Now had we known the approximate values of m_{ij} before carrying out the LU decomposition we might, since m_{12} , m_{31} and m_{34} are the smallest elements of \mathbf{M} by a considerable margin, have decided to set $u_{12} = l_{31} = u_{34} = 0$ and performed an incomplete decomposition instead. A more cavalier approach would in addition set $u_{23} = u_{24} = 0$ since $|m_{23}| = |m_{24}| = 2$, and a more vigorous approach still would have set $l_{43} = 0$ as well. The index pairs corresponding to the elements required to be zero would be included in Z and the incomplete factors calculated to satisfy equation (11.115). Now all these choices would be available if we knew in advance the orders of magnitude of the elements of \mathbf{L} and \mathbf{U} , and we now describe how the approximate orders of magnitude of these elements may be computed.

The symbolic algorithm to compute levels of fill is obtained by simply replacing $a(i, j)$ by $lev(a(i, j))$ (written as lev_{ij}) in the simple triangularisation algorithm and replacing the arithmetic operations $+$, $-$, $*$ and $/$ by their symbolic equivalents as defined by equations (11.125) - (11.126). To initiate the procedure we assign “levels of fill” to the elements of the original matrix \mathbf{A} . This is done simply by setting the levels to be:

- (1) zero for the diagonal elements of \mathbf{A}
- (2) unity for the nonzero off-diagonal elements, and
- (3) infinity for the zero off-diagonal elements.

Note that these choices are only valid if \mathbf{A} shows at least some signs of diagonal dominance since they imply that the diagonal elements of \mathbf{A} are larger than the off-diagonal ones by at least an order of magnitude, i.e. by ε^{-1} . Once the original values have been assigned the algorithm is as follows:

The “Level of fill” algorithm

- (1) for $k = 1 : n - 1$ do
 - (a) for $i = k + 1 : n$ do
 - (i) for $j = k + 1 : n$ do
 - (A) $lev_{ij} := \min(lev_{ij}, lev_{ik} + lev_{kj} - lev_{kk})$;
 - (ii) end
 - (b) end
 - (2) end “Level of fill” algorithm.

We now consider some details of this symbolic calculation. In the original assignments we set the diagonal levels to be zero so that the first inner iteration of the algorithm becomes

- (a) for $i = 2 : n$ do

```

i. for  $j = 2 : n$  do
    A.  $lev_{ij} = \min(lev_{ij}, lev_{i1} + lev_{1j})$ 
ii. end
(b) end

```

Since the off-diagonal levels have been set to values that are at least unity this implies that after one inner iteration the diagonal levels are unchanged and that the off-diagonal levels cannot be less than either two or their original values, whichever is the smaller. Thus the unit levels assigned to the off-diagonal nonzeros are also unchanged and the general pattern, that of zeros on the diagonal, ones for the original off-diagonal non-zeros and “greater than ones” for the original off-diagonal zeros has been retained. Applying these arguments recursively to the results obtained after successive inner iterations leads to the conclusion that, at the termination of the symbolic calculation,

- (1) The diagonal levels remain zero
- (2) The off-diagonal levels corresponding to $a_{ij} \neq 0$ remain unity, and
- (3) The off-diagonal levels corresponding to $a_{ij} = 0$ are at least two.

Since each of these levels represents an “order of magnitude”, with the magnitude of the element decreasing as the level increases, we can use the results of this symbolic calculation to determine which elements of \mathbf{L} and \mathbf{U} should be zero.

Remark 3. The level-of-fill algorithm as presented above is a formal algorithm intended to illustrate its basic structure and its mathematical underpinning. It is not intended to be used as it stands as the basis of a computer program. In practice we would note that Step 1.(a)i. implies that lev_{ij} can only change if both lev_{ik} and lev_{kj} are finite, and even then change is not guaranteed. Thus it is only necessary to perform the detailed comparisons implicit in Step 1.(a)i. for elements appearing in those rows i and columns j for which lev_{ik} and lev_{kj} are both finite. This means that effectively when performing Step 1.(a). all rows and columns associated with infinite values of lev_{ik} and lev_{kj} may be ignored. Since for large sparse matrices (at least in the early stage of the algorithm) the vast majority of the elements $a(i, j)$ are zero (and hence the corresponding values of lev_{ij} are infinite) this results in a significant reduction in work when compared with the formal algorithm.

Returning to our example and denoting the symbolic forms of the matrices by the subscript s gives

$$\mathbf{A}_s = \begin{bmatrix} 0 & \infty & 1 & 1 \\ 1 & 0 & 1 & \infty \\ \infty & 1 & 0 & \infty \\ 1 & 1 & \infty & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{M}_s = \begin{bmatrix} 0 & \infty & 1 & 1 \\ 1 & 0 & 1 & 2 \\ \infty & 1 & 0 & 3 \\ 1 & 1 & 2 & 0 \end{bmatrix}.$$

We see immediately that the smaller elements of \mathbf{M} , (m_{12} , m_{31} and m_{34}), correspond to the highest levels of fill, (∞ , ∞ and 3), as would be expected from the theory. The next levels correspond to m_{24} and m_{43} which are somewhat larger

while level one relates to the original non-zero off-diagonal elements. The only discrepancy occurs with m_{23} and m_{24} for which $|m_{23}| = |m_{24}| = 2$ but for which $\text{lev}_{23} = 1$ and $\text{lev}_{24} = 2$. The reason for this is that $|m_{23}|$ is artificially small due to cancellation which is, of course, ignored when computing levels of fill. Thus even for this trivial example which is not particularly diagonally dominant the level-of-fill calculations give a reasonable estimate of the magnitude of the elements of the complete factors and thus indicate which could perhaps be neglected when deciding how the incomplete factors should be structured. Despite this algorithm being based on order-of-magnitude considerations, its outcome is determined solely by the zero/nonzero structure of the matrix \mathbf{A} . Incomplete LU-decompositions based on this and similar algorithms are therefore referred to as *factorisations by position*.

We are now in a position to define $\text{ILU}(p)$, where p is some “cut-off” level of fill. It is defined to be the incomplete factorisation determined by the set \mathcal{Z} , where

$$\mathcal{Z} = \{(i, j) \mid \text{lev}_{ij} > p\} \quad (11.129)$$

and where the approximate factors \mathbf{L} and \mathbf{U} are computed using equation (11.115). Note that if we choose $p = 1$ the pattern of zeros given by the above algorithm is precisely the “no fill-in” option since unit levels correspond to the disposition of zero/nonzero elements in the original matrix \mathbf{A} . This, however, causes a slight problem. We would like, for cosmetic reasons, $\text{ILU}(0)$ rather than $\text{ILU}(1)$ to correspond to the “no fill-in” option, and this can easily be achieved by reducing the initial assigned levels by unity so that the triplet $(0, 1, \infty)$ becomes $(-1, 0, \infty)$. It is then simple to show from equation (11.126) that Step 1.(a)i. of the symbolic version of the algorithm becomes

$$\text{lev}_{ij} = \min(\text{lev}_{ij}, \text{lev}_{ik} + \text{lev}_{kj} + 1)$$

and that all levels computed by the algorithm are simply reduced by one. With this change, $\text{ILU}(0)$ as defined by equation (11.129) corresponds to no fill-in. Note that reducing the level of fill of all elements by the same amount does not alter their relative orders-of-magnitude.

To conclude our example, if we select the “no fill-in” option we obtain

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 8 & 0 & 4 & 4 \\ 0 & 8 & 2 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}.$$

The elements u_{12} , u_{24} , l_{31} , u_{34} and l_{43} have been set to zero and of these, u_{12} and l_{31} would have been zero in any case (see equation (11.128)). The matrix \mathbf{R} is null except that $r_{24} = 2$ and $r_{43} = 3$. The element r_{34} , which in general would not have been expected to be zero, is fortuitously so in this case.

Once the sparsity pattern (the set \mathcal{Z}) has been determined the triangular decomposition proper is carried out. This is usually done by a form of Gaussian elimination

based on a variant of the simple triangularisation algorithm. Algorithm ST may be written:

```

for  $k = 1 : n - 1$  do
    for  $i = k + 1 : n$  do
        modify row( $i$ ) using row( $k$ )
    end
end

```

and the variant is

```

for  $i = 2 : n$  do
    for  $k = 1 : i - 1$  do
        modify row( $i$ ) using row( $k$ )
    end
end.

```

This is referred to as the IKJ variant from the order of nesting of the for-loops. The results obtained by the two versions of the algorithm are identical even to the effects of rounding. This implies that any reason for preferring one to the other is not numerical but purely computational and depends on the way the data (the sparse matrix) is structured. The elements of the triangular factors are merely computed in a different order (to be convinced of this, replace the line “modify row(i) etc.” by “print(k), print(i)”, run the programs and compare the results). Note that in both cases row(k) is not used to modify other rows until it has itself undergone its full complement of modifications. The IKJ version is, according to Saad ([217], page 272), that most commonly used when dealing with matrices stored as a row-contiguous data structures.

Consider now the IKJ variant of the ST algorithm but where, for computational reasons, row(i) of \mathbf{A} has been stored as a work vector $\mathbf{w} = [w_j]$. This is the

ST algorithm, IKJ variant

- (1) for $i = 2 : n$ do
 - (a) for $j = 1 : n$ do $w(j) = a(i, j)$
 - (b) end setting up the work vector
 - (c) for $k = 1 : i - 1$ do
 - (i) for $j = k + 1 : n$ do $w(j) = w(j) - w(k) * a(k, j) / a(k, k)$
 - (ii) end
 - (d) end
 - (e) for $j = 1 : n$ do $a(i, j) = w(j)$
 - (f) end updating row(i)
- (2) end

Now this algorithm computes the matrices \mathbf{U} and $(\mathbf{L} - \mathbf{I})\mathbf{D}$ (see discussion and notation on page 237) so in order to compute the off-diagonal elements of $(\mathbf{L} - \mathbf{I})$

it is necessary to replace $m(i,j)$ by $m(i,j)/m(j,j)$ for $i > j$. This may be achieved by dividing the first $(i-1)$ elements of $w(j)$ by $a(j,j)$ just before carrying out Step 1(e). However since Step 1(c)i only changes elements $k+1$ to n of the work vector w , $w(k)$ may be replaced by $w(k)/a(k,k)$ before this step of the algorithm, and if this is done the expression $w(k)/a(k,k)$ in Step 1(c)i must be replaced by $w(k)$. With these changes Steps 1(c) and 1(d) of the algorithm become

- (c) for $k = 1 : i-1$ do
 - i. $w(k) = w(k)/a(k,k)$
 - ii. for $j = k+1 : n$ do $w(j) = w(j) - w(k) * a(k,j)$
 - iii. end
- (d) end

If we now add the conditional statements to obtain the full version of the algorithm we finally obtain:

The basic ILU algorithm (IKJ version)

- (1) for $i = 2 : n$ do
 - (a) for $j = 1 : n$ do
 - (i) if $(i,j) \notin \mathcal{Z}$ then $w(j) = a(i,j)$ else $w(j) = 0$
 - (ii) endif
 - (b) end setting up the work vector
 - (c) for $k = 1 : i-1$ do
 - (i) if $(i,k) \notin \mathcal{Z}$ then
 - (A) $w(k) = w(k)/a(k,k)$
 - (B) for $j = k+1 : n$ do
 - (C) if $(i,j) \notin \mathcal{Z}$ then $w(j) = w(j) - w(k) * a(k,j)$
 - (D) endif
 - (E) end j -th modification of work vector
 - (ii) endif
 - (d) end modifications of work vector
 - (e) for $j = 1 : n$ do $a(i,j) = w(j)$
 - (f) end modification of row(i)
 - (2) end basic ILU algorithm.

Watts carried out no comparisons with IC(0) or straight CG when testing his algorithm but did compare it with SIP and with a line over-relaxation scheme. The tests were performed on twelve problems derived from petroleum reservoir pressure calculations, five of them two-dimensional and the remaining seven three-dimensional. IC(3) was used on the 2-D problems and IC(5) on the 3-D ones. They showed both IC(3) and IC(5) to be reliable for the problems to which they were applied and both had an advantage over SIP in not requiring parameters to be provided. For 2-D problems, IC(3) was generally at least as fast as SIP, and occasionally much faster. For 3-D problems, SIP was normally superior but could be spectacularly slow. The over-relaxation algorithm performed badly in all tests

and was not regarded as a serious competitor. These assessments were more than reinforced by three years of subsequent routine use.

Meijerink and van der Vorst [183] also made comparisons between IC(0) and an algorithm they referred to as ICCG(3), essentially our IC(3). For the matrices defined by equation (11.120), IC(3) allowed fill-in such that if \mathbf{M} denotes the “matrix” resulting from the procedure (in reality the superimposed triangular factors) then

$$\mathbf{M} = \begin{bmatrix} \mathbf{Q} & \mathbf{L} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{O} \\ \mathbf{L}^T & \mathbf{Q} & \mathbf{L} & \cdots & \mathbf{O} & \mathbf{O} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \cdots & \mathbf{L}^T & \mathbf{Q} \end{bmatrix}$$

where \mathbf{Q} is symmetric and quidiagonal and $\mathbf{L} = [l_{ij}]$ is simultaneously lower triangular and tridiagonal, i.e. $l_{ij} = 0$ for $j < i - 2$ and $j > i$. Thus \mathbf{M} has six more diagonals than in the original coefficient matrix, three on each side of the main diagonal. For the first test example, IC(3) reduced the Euclidean norm of the true residual to 10^{-10} in 25 equivalent iterations whereas IC(0) needed 45 equivalents to obtain the same reduction. For the second test example, IC(3) reduced the residual norm to 10^{-7} in 35 equivalents whereas IC(0) needed 45.

The implication of the tests is that better convergence may be obtained by increasing the level-of-fill but as always with preconditioned CG, the cost of computing the preconditioners could well offset any reduction in the number of iterations. The matter is not clear-cut, and only becomes so when in the absence of preconditioning the iteration fails to converge. This is unlikely to happen if a symmetric positive definite problem is solved by CG but can occur when indefinite or non-symmetric problems are solved by GMRes(m) (see e.g. [67] or [215]).

Incomplete factorization is also applicable to block-matrices where \mathbf{A} has been partitioned into submatrices \mathbf{A}_{ij} . If the order of each submatrix is small, the cost of computing the exact inverse of the pivot blocks is acceptable. Alternatively the algorithm can be generalized to permit approximations of these matrix inverses. Concus, Golub and Meurant [76] describe a preconditioner for block tridiagonal matrices derived from the discretisation of boundary value problems arising from elliptic partial differential equations. Sparse approximate matrix inverses are used to generate incomplete block Cholesky factors. Other block approaches have been considered by Axelsson and his co-workers [10], [12], [13], [15], Meurant [184] and Underwood [236].

11.7.4. Threshold variants (ILUT)

Although the ILU(p) method can result in a satisfactory preconditioner if conditions are favourable, it is based on an assumption of diagonal dominance that is not valid for all matrices. In particular it is not valid for indefinite and nonsymmetric matrices. For these it is often not possible to determine *a priori* which elements of the triangular factors should be zero and other methods of deciding the sparsity structure must be employed. These methods can be based on computing the ac-

tual values for l_{ij} or u_{ij} and setting them to zero if they are less than a particular threshold. The rules for doing this are known as *dropping rules* and are based on so-called *dropping criteria*. They generally yield more accurate factorizations with the same amount of fill-in than level-of-fill methods (even for some diagonally dominant M-matrices, see [67]). A possible, if not particularly satisfactory, algorithm of this type could be based on equations (11.117) and (11.118). When the elements v_i (which are of course elements of \mathbf{U}) are computed they could be monitored and replaced by zero if thought appropriate. The problem with this approach is that although the sum in equation (11.118) could be replaced by a sparse sum since the zero values of l_{ij} would already be known, it could still be necessary to examine every element of \mathbf{v} . This in effect means that every off-diagonal element of both \mathbf{L} and \mathbf{U} would need to be checked for size even if it were obviously zero, not a very efficient procedure.

The use of dropping rules highlights the importance of implementation since a dropping rule is often specific to a particular variant of an algorithm. Thus two versions of ILU(p) might give the same results for the same zero set \mathcal{Z} but threshold versions could differ because the zero sets would not be the same for each version. In what follows, therefore, we sometimes discuss more than one version of the same algorithm, referring to them collectively as ILUT algorithms.

Another property of the ILUT algorithm is that unless additional measures are taken, the total amount of storage needed is not known beforehand. It is therefore common in such algorithms to limit the number of nonzero elements in each row of the final matrix to m (say), retaining only the m largest elements and setting the others to zero regardless of their magnitude. We shall refer to such methods as *limited fill-in methods*.

One of the first threshold methods was devised by Munksgaard [190] who proposed what is, in effect, a dynamic level-of-fill algorithm. It is related to an earlier one by Gustafsson [138] which was specific to problems derived from partial differential equations, and Munksgaard's version is intended for use on general symmetric positive definite matrices. It is, in fact, another variation on the simple triangularisation algorithm (see page 236). Its principal step is Step 1.(a).A of that algorithm which may be written

$$a(i,j) = a(i,j) - \delta(i,j)$$

where the correction $\delta(i,j)$ is given by¹⁷

$$\delta(i,j) = a(i,k) * a(k,j) / a(k,k).$$

Munksgaard, however, carries out this step if at least one of the following two dropping conditions is *not* satisfied:

- (1) $a(i,j) = 0$, and
- (2) $|\delta(i,j)|$ is less than some lower threshold.

Thus only if both $a(i,j)$ is zero and $|\delta(i,j)|$ is “small” is $a(i,j)$ left equal to zero, i.e. no additional fill-in is introduced. Also if either $a(i,k) = 0$ or $a(k,j) = 0$

¹⁷The asterisk denotes multiplication.

then $\delta(i, j) = 0$ and again no change to $a(i, j)$ occurs. This means in effect that when carrying out Step 1.(a)i. of simple triangularisation only a fraction of the rows and columns of the matrix need be considered, i.e. those for which $a(i, k)$ and $a(k, j)$ are both nonzero. This is completely analogous to the practical version of the level-of-fill algorithm (see Remark 3, above) with similar programming implications. These, then, are the basic ideas behind the algorithm, and we now consider some refinements added by Munksgaard to give the finished version.

The first concerns the decision whether the correction $\delta(i, j)$ is large enough to change a zero element to a nonzero one (i.e. to introduce fill-in), and it is so deemed for this purpose if

$$|\delta(i, j)| > c\sqrt{a(i, i) * a(j, j)}$$

where c , $0 \leq c \leq 1$, is some arbitrary constant which is subsequently referred to as the *relative drop tolerance*. Since replacing $\delta(i, j)$ by zero is only considered if $a(i, j) = 0$ this is equivalent to replacing a zero value of $a(i, j)$ by a nonzero one, $\bar{a}(i, j)$ say, if $|\bar{a}(i, j)|$ satisfies the same inequality.

The second refinement is implemented if $a(i, j)$ is *not* changed, i.e. if neither of the two dropping conditions is satisfied. In this case Munksgaard puts

$$a(i, i) = a(i, i) - \delta(i, j) \quad \text{and} \quad a(j, j) = a(j, j) - \delta(i, j),$$

a process that has been referred to as *diagonal compensation*. The motivation for this is to ensure that the row-sums of the original matrix \mathbf{A} are equal to those of the product of the factors generated by the algorithm and was first proposed by Gustafsson [138]. This implies that if $\mathbf{e}^T = [1, 1, \dots, 1]$ then

$$\mathbf{R}\mathbf{e} = \mathbf{0} \tag{11.130}$$

where \mathbf{R} is defined by equation (11.114). For the reason for this and the proof that it is achieved by the proposed algorithmic modification, see [190] or [217].

Now any reduction of a diagonal element of a symmetric positive definite matrix will generally exacerbate its condition number and can make the matrix indefinite. To avoid this the diagonal elements $a(i, i)$ of the original matrix are scaled by a factor greater than one. Even before the algorithm proper gets under way they are replaced by $(1 + \varsigma) * a(i, i)$, where ς is a second positive arbitrary constant. However ς should not be chosen to be too large since this reduces the accuracy by which $\mathbf{L}\mathbf{L}^T$ approximates \mathbf{A} , but if it is too small it is possible that during the course of the algorithm a pivot $a(k, k)$ also becomes too small and the factorisation becomes numerically unstable. Thus if, when it is about to be used as a pivot,

$$a(k, k) \leq u * M \tag{11.131}$$

where u is yet another arbitrary constant, it is replaced by M where

$$M = \max_{k+1 \leq j \leq n} |a(k, j)|$$

(if $a(k, k) < 0$ and $M = 0$ then $a(k, k)$ is set equal to unity). If inequality (11.131) is not satisfied then the algorithm is allowed to continue normally. If pivots satisfying

these criteria are used the growth in size of elements of \mathbf{A} is limited to at most $(1 + u^{-1})$ at each stage. The value of u in [190] was chosen to be 0.01.

A final refinement of the algorithm is pivoting, introduced not to improve numerical stability but to exploit maximally the sparseness of \mathbf{A} . Before each elimination step (Step 1.(a) of the ST algorithm) the matrix is subjected to (symmetry preserving) row and column interchanges to minimise the number of nonzero elements in row k and hence reduce the potential fill-in. Bundling all those modifications together results in:

Munksgaard's algorithm ICT (variant)

1. Choose the constants c , u and ς
2. for $k = 1 : n$ do $a(k, k) = (1 + \varsigma) * a(k, k)$
3. end
4. for $k = 1 : n - 1$ do
 5. interchange row and column k with row and column r where
 $k < r \leq n$ and the number of nonzero elements $a(r, j)$,
 $j = k, k + 1, \dots, n$ is minimal
 6. compute $M = \max_{j \in [k+1, n]} |a(k, j)|$
 7. if $a(k, k) \leq u * M$ then $a(k, k) = M$ unless $M = 0$ in which
 case $a(k, k) = 1$
8. endif
9. for $i = k + 1 : n$ do
 10. $a(i, k) = a(i, k) / a(k, k)$
 11. for $j = k + 1 : n$ do
 12. compute $s = a(i, k) * a(k, j)$
 13. if $a(i, j) \neq 0$ or $|s| > c * \sqrt{a(i, i) * a(j, j)}$ then
 14. $a(i, j) = a(i, j) - s$ else
 15. begin $a(i, i) = a(i, i) - s$
 16. $a(j, j) = a(j, j) - s$
 17. end
 18. endif
 19. end
 20. end
21. end Munksgaard's algorithm ICT (variant)

Notes. In practice this algorithm would be modified to make it more efficient. In Step 11, i would replace $k + 1$ to exploit the symmetry of \mathbf{A} . This necessitates further changes and in [190] two extra arrays were introduced, both to effect these changes and to clarify the exposition. Another refinement is that Steps 12–18 would be omitted unless both $a(i, k)$ and $a(k, j)$ are nonzero. This is the basic form of the algorithm. Munksgaard also proposed a further procedure which changes c so as to effectively limit possible fill-in.

To check the effectiveness of his new method, Munksgaard [190] compared the following algorithms:

- (1) CG with a symmetric scaling that sets all diagonal entries to 1;

- (2) preconditioned CG where the preconditioning matrix is computed by IC with three relative drop tolerances ($c = 1, 10^{-2}, 0$) corresponding to incomplete factorisation with no fill-in, incomplete factorisation with fill-in greater than 10^{-2} and complete factorisation;
- (3) a direct method (the Yale sparse matrix code).

Fourteen test problems were used, each with multiple right-hand sides. The main conclusion seems to be that no one method is best for all the problems. That the Munksgaard algorithm is not uniformly successful may be due to the errors incurred in the approximations when modifying the factors to avoid numerical instability (Steps 2 and 7).

The extension of the ideas of Munksgaard to general ILU decomposition was carried out by various authors, most notably by Zlatev [262]. The idea was taken up by Saad [215] whose algorithm ILUT is based on a variant of the basic ILU algorithm (see page 242). ILUT differs from this algorithm by omitting Steps 7 and 13, replacing Steps 2 – 3 by $w(j) = a(i, j)$, and zeroing both $w(k)$ in Step 8 and $w(k) * a(k, j)$ in Step 10 if these quantities satisfy the appropriate dropping criteria. Note that monitoring $a(i, k)/a(k, k)$ (i.e. $w(k)$) but not $a(k, j)/a(k, k)$ introduces a lack of symmetry not present in the Munksgaard algorithm for which neither $a(i, k)/a(k, k)$ nor $a(k, j)/a(k, k)$ is monitored. The final algorithm becomes:

Algorithm ILUT (variant)

1. for $i = 2 : n$ do
2. for $j = 1 : n$ do $w(j) = a(i, j)$
3. end setting up the work vector
4. for $k = 1 : i - 1$ do
5. compute $w(k) = w(k)/a(k, k)$ but set it to zero if the dropping condition is satisfied
6. for $j = k + 1 : n$ do
7. compute $s = w(k) * a(k, j)$ but set it to zero if the dropping condition is satisfied
8. $w(j) = w(j) - s$
9. end ($j - k$)-th modification of work vector
10. end all modifications of work vector
11. for $j = 1 : n$ do $a(i, j) = w(j)$
12. end modification of row(i)
13. end ILUT (variant).

Note: Step 2 is a “sparse copying” operation and is assumed to set $w(j) = 0$ when appropriate. In the version of ILUT described in [215], fill-in limitation is incorporated. Thus after Step 10, the number of non-zero elements of the vector \mathbf{w} is checked and the smaller “dual threshold” ones zeroed if there are more than k , say, in either the “L-part” or the “U-part” of row(i). This version is usually referred to as ILUT(τ, k), where τ is the drop tolerance.

Numerical experiments using ILUT-preconditioned GMRes(m) for $m = 10$ or $m = 20$ and applied to problems from the Harwell-Boeing collection [94] were reported by Saad in [215], but the presented conclusions were mainly of a qualitative nature. They amounted to:

- (1) A good ILUT factorisation (i.e. large number of nonzeros permitted in each row and a small drop tolerance) is preferable to a poor one, for which GMRes(m) may not converge
- (2) If the accuracy of the factorisation is progressively increased there eventually comes a point beyond which GMRes(m) does converge
- (3) This point is reached suddenly, and there is another point beyond which any further increases of accuracy are not cost-effective.

The one reported disadvantage of ILUT was the difficulty of obtaining an optimal implementation on high-performance computers.

Another version [67] of ILUT is based on triangular decomposition. Let \mathbf{A}_i denote the matrix comprising the first i rows of \mathbf{A} . Then, with obvious notation,

$$\mathbf{A}_{i+1} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{c}^T & \mathbf{d}^T \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{11} & \mathbf{0} \\ \mathbf{p}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{0}^T & \mathbf{v}^T \end{bmatrix}$$

where $\mathbf{A}_{11}, \mathbf{U}_{11} \in \mathbb{R}^{i \times i}$. Thus \mathbf{p} and \mathbf{v} may be computed from

$$\mathbf{U}_{11}^T \mathbf{p} = \mathbf{c} \quad \text{and} \quad \mathbf{v} = \mathbf{d} - \mathbf{U}_{12}^T \mathbf{p},$$

and numerical dropping may be performed in two ways:

- (1) drop small entries in \mathbf{p} and \mathbf{v} after these vectors have been computed, or
- (2) drop small entries in \mathbf{p} during the solution of $\mathbf{U}_{11}^T \mathbf{p} = \mathbf{c}$. This type of numerical dropping apparently introduces less error into the factorization than the first strategy.

As in ILUT, fill-in limitation is used to limit the storage requirements so an additional parameter is included which specifies the maximum number of nonzeros in \mathbf{p} and \mathbf{v} . If the number of computed nonzeros exceeds this, only the largest are retained.

One of the practical difficulties associated with ILUT is the need to specify both the threshold(s) and the fill-in limit(s). Although the larger the number of arbitrary parameters, the greater the opportunity for “tuning” it is also the case that the more parameters that are needed, the further the algorithm is from the “black-box” ideal where no information at all is required of the user beyond that needed to specify the problem. In an attempt to seek a compromise between these opposing requirements Jones and Plassmann [166], [165] proposed a scheme of IC preconditioning where the *number* of the non-zero elements of the i -th row of the approximate lower triangular factor \mathbf{L} is determined to be the same that would be obtained by IC(0), i.e. the number of non-zero elements a_{ij} for $j \leq i$. The *positions* of the selected non-zeroes are then determined by their values. This strategy often works rather well although it can be slow for difficult problems (see [23]).

An obvious extension of this idea is to increase the number of non-zeroes assigned to each row of \mathbf{L} by the same fixed amount, p say [178]. This provides a precon-

ditioner where storage requirements can still be predicted and which only requires single parameter p to be specified. Even this could perhaps be dispensed with as for many cases a value of five appears to be quite adequate [178].

11.7.5. Imposing symmetry

In the ILUT algorithms previously discussed the number of nonzero elements in each row may be limited, but no such constraint applies to columns. It is possible that even if \mathbf{A} is *structurally symmetric* ($a_{ji} = 0$ if and only if $a_{ij} = 0$), the product \mathbf{LU} is not, and intuitively this may be regarded as a weakness of the algorithm. The condition of structural symmetry can be imposed on \mathbf{LU} by reverting to the original LU-decomposition as defined by equation (11.116). This incomplete form of LU decomposition was discussed by Ortega [199], p. 397 and by Chow and Saad [68].

Let \mathbf{A}_i be the i -th leading principal submatrix of \mathbf{A} and assume we have a decomposition $\mathbf{A}_i = \mathbf{L}_i \mathbf{D}_i \mathbf{U}_i$ where \mathbf{L}_i and \mathbf{U}_i are unit triangular matrices. Equation (11.116) becomes

$$\begin{bmatrix} \mathbf{A}_i & \mathbf{b} \\ \mathbf{c}^T & \alpha \end{bmatrix} = \begin{bmatrix} \mathbf{L}_i & \mathbf{0} \\ \mathbf{p}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{D}_i & \mathbf{0} \\ \mathbf{0}^T & \delta \end{bmatrix} \begin{bmatrix} \mathbf{U}_i & \mathbf{v} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

so that \mathbf{v} and \mathbf{p} may be computed by solving

$$\mathbf{L}_i \mathbf{D}_i \mathbf{v} = \mathbf{b} \quad (11.132)$$

and

$$\mathbf{U}_i^T \mathbf{D}_i \mathbf{p} = \mathbf{c}. \quad (11.133)$$

The scalar δ is computed by

$$\delta = \alpha - \mathbf{p}^T \mathbf{D}_i \mathbf{v}.$$

In order for \mathbf{v} and \mathbf{p} to have the same sparsity pattern, equations (11.132) and (11.133) are solved simultaneously with numerical dropping. Corresponding entries in \mathbf{v} and \mathbf{p} are both kept or both dropped to maximize $|\mathbf{p}^T \mathbf{D}_i \mathbf{v}|$. Two advantages [67] of the bordered form of factorization are

- (1) if all elements of \mathbf{b} and \mathbf{c} are nonzero, fill-in onto the diagonal is guaranteed, and
- (2) a running condition estimate $\|(\mathbf{L}_i \mathbf{U}_i)^{-1}\|_\infty$ can be monitored since \mathbf{L}_i and \mathbf{U}_i are available at the end of step i .

Other variants of the ILU algorithm have been suggested by van der Ploeg *et al* [237], who consider a submatrix form based on a combination of preconditioned CG and multigrid methods. Although this is more expensive than row-based methods like the basic ILU algorithm, it is also more flexible and allows symmetry to be imposed.

11.7.6. Tismenetsky's method

A more recent algorithm for ILU decomposition is that due to Tismenetsky [234]. Let

$$\mathbf{A} = \mathbf{LDU} \quad (11.134)$$

where \mathbf{D} is diagonal and \mathbf{L} and \mathbf{U} are unit lower and unit upper triangular respectively. Tismenetsky defines matrices \mathbf{L}_i and \mathbf{U}_i by

$$\mathbf{L}_i = \mathbf{I} + (\mathbf{L} - \mathbf{I}) \mathbf{e}_i \mathbf{e}_i^T \quad \text{and} \quad \mathbf{U}_i = \mathbf{I} + \mathbf{e}_i \mathbf{e}_i^T (\mathbf{U} - \mathbf{I})$$

where \mathbf{e}_i denotes the i -th column of the unit matrix. Since $\mathbf{L} - \mathbf{I}$ is strictly lower triangular we have, for $j \leq i$, $\mathbf{e}_j^T (\mathbf{L} - \mathbf{I}) \mathbf{e}_i = 0$ so that

$$\mathbf{L}_i^{-1} = \mathbf{I} - (\mathbf{L} - \mathbf{I}) \mathbf{e}_i \mathbf{e}_i^T \quad (11.135)$$

and

$$\mathbf{L}_j \mathbf{L}_i = \mathbf{I} + (\mathbf{L} - \mathbf{I}) (\mathbf{e}_j \mathbf{e}_j^T + \mathbf{e}_i \mathbf{e}_i^T).$$

Simple induction based on the second of these equations shows that

$$\mathbf{L}_1 \mathbf{L}_2 \dots \mathbf{L}_n = \mathbf{I} + (\mathbf{L} - \mathbf{I}) = \mathbf{L} \quad (11.136)$$

with a similar equation for \mathbf{U} . Define now the matrices $\bar{\mathbf{L}}_i$ and \mathbf{A}_i by $\bar{\mathbf{L}}_i = \mathbf{L}_1 \mathbf{L}_2 \dots \mathbf{L}_i$ and $\mathbf{A}_i = \bar{\mathbf{L}}_i^{-1} \mathbf{A} \bar{\mathbf{U}}_i^{-1}$ so that, if $\mathbf{A}_0 = \mathbf{A}$,

$$\mathbf{A}_i = \mathbf{L}_i^{-1} \mathbf{A}_{i-1} \mathbf{U}_i^{-1}, \quad i = 1, 2, \dots, n. \quad (11.137)$$

From equations (11.134) and (11.136), $\mathbf{A}_n = \mathbf{D} = \text{diag}(d_i)$ while equations (11.135) and (11.137) imply that

$$\mathbf{A}_i = (\mathbf{I} - (\mathbf{L} - \mathbf{I}) \mathbf{e}_i \mathbf{e}_i^T) \mathbf{A}_{i-1} (\mathbf{I} - \mathbf{e}_i \mathbf{e}_i^T (\mathbf{U} - \mathbf{I})). \quad (11.138)$$

Now the i -th column of \mathbf{A}_i is given by $\mathbf{A}_i \mathbf{e}_i$ so that

$$\mathbf{A}_i \mathbf{e}_i = (\mathbf{I} - (\mathbf{L} - \mathbf{I}) \mathbf{e}_i \mathbf{e}_i^T) \mathbf{A}_{i-1} \mathbf{e}_i. \quad (11.139)$$

We want to choose the i -th column of \mathbf{L} so that $\mathbf{A}_i \mathbf{e}_i$ forms the i -th column of the diagonal matrix \mathbf{D} . Substituting $\mathbf{e}_i d_i$ for $\mathbf{A}_i \mathbf{e}_i$ in equation (11.139) gives

$$\mathbf{e}_i d_i = \mathbf{A}_{i-1} \mathbf{e}_i - (\mathbf{L} - \mathbf{I}) \mathbf{e}_i \mathbf{e}_i^T \mathbf{A}_{i-1} \mathbf{e}_i \quad (11.140)$$

which on pre-multiplication by \mathbf{e}_i^T yields $d_i = \mathbf{e}_i^T \mathbf{A}_{i-1} \mathbf{e}_i$. Substituting this expression for d_i back into equation (11.140) then gives

$$\mathbf{L} \mathbf{e}_i = \mathbf{A}_{i-1} \mathbf{e}_i d_i^{-1} \quad (11.141)$$

so that the i -th column of \mathbf{L} is just the i -th column of \mathbf{A}_i divided by its own i -th element. The general appearance of \mathbf{A}_i may be inferred from the following example for which $i = 2$ and $n = 5$:

$$\mathbf{A}_2 = \begin{bmatrix} d_1 & 0 & * & 0 & 0 \\ 0 & d_2 & * & 0 & 0 \\ 0 & 0 & \alpha_{33} & \alpha_{34} & \alpha_{35} \\ 0 & 0 & \alpha_{43} & \alpha_{44} & \alpha_{45} \\ 0 & 0 & \alpha_{53} & \alpha_{54} & \alpha_{55} \end{bmatrix} \quad (11.142)$$

where the asterisks represent zeroes for complete decomposition. If we substitute the value of \mathbf{Le}_i obtained by equation (11.141) back into equation (11.138) we get

$$\mathbf{A}_i = \mathbf{A}_{i-1} - \frac{\mathbf{A}_{i-1}\mathbf{e}_i\mathbf{e}_i^T\mathbf{A}_{i-1}}{d_i} + \mathbf{e}_i d_i \mathbf{e}_i^T. \quad (11.143)$$

To see how the incomplete version is obtained from the above, denote by a “hat” the incomplete counterparts of \mathbf{A} , \mathbf{L} and d_i and suppose that $\widehat{\mathbf{A}}_{i-1}$ has already been computed. Let (by analogy with equation (11.141))

$$\widehat{\mathbf{Le}}_i = (\widehat{\mathbf{A}}_{i-1}\mathbf{e}_i - \mathbf{f}_i) \widehat{d}_i^{-1} \quad \text{and} \quad \mathbf{e}_i^T \widehat{\mathbf{U}} = \widehat{d}_i^{-1} (\mathbf{e}_i^T \widehat{\mathbf{A}}_{i-1} - \mathbf{g}_i^T)$$

where $\widehat{d}_i = \mathbf{e}_i^T \widehat{\mathbf{A}}_{i-1} \mathbf{e}_i$ and the vectors \mathbf{f}_i and \mathbf{g}_i^T consist of those elements of $\widehat{\mathbf{A}}_{i-1}\mathbf{e}_i$ and $\mathbf{e}_i^T \widehat{\mathbf{A}}_{i-1}$ to be excluded from the incomplete factors $\widehat{\mathbf{L}}$ and $\widehat{\mathbf{U}}$. Define \mathbf{u}_i and \mathbf{v}_i by

$$\mathbf{u}_i = \widehat{\mathbf{A}}_{i-1}\mathbf{e}_i - \mathbf{f}_i \quad \text{and} \quad \mathbf{v}_i^T = \mathbf{e}_i^T \widehat{\mathbf{A}}_{i-1} - \mathbf{g}_i^T$$

and note that in the “hatted” version of equation (11.142) some of the elements denoted by asterisks, which are zero for complete decomposition, may be non-zero and must be included in \mathbf{f}_i and \mathbf{g}_i . This guarantees that \mathbf{u}_i and \mathbf{v}_i^T have zeroes in the correct positions and that the remaining elements of these vectors are the corresponding elements of $\widehat{\mathbf{A}}_{i-1}\mathbf{e}_i$ and $\mathbf{e}_i^T \widehat{\mathbf{A}}_{i-1}$ respectively. The matrix $\widetilde{\mathbf{A}}_i$ is then given by

$$\widetilde{\mathbf{A}}_i = \widehat{\mathbf{A}}_{i-1} - (\mathbf{u}_i \mathbf{v}_i^T + \mathbf{u}_i \mathbf{g}_i^T + \mathbf{f}_i \mathbf{v}_i^T) \widehat{d}_i^{-1} \quad (11.144)$$

and the updated matrix $\widehat{\mathbf{A}}_i$ is obtained from $\widetilde{\mathbf{A}}_i$ by putting its i -th diagonal element equal to d_i .

The recursion (11.144) differs from (11.143) in lacking two terms. The first of these, $\mathbf{f}_i \widehat{d}_i^{-1} \mathbf{g}_i^T$, represents the dropping error while the other, $\mathbf{e}_i \widehat{d}_i^{-1} \mathbf{e}_i^T$, is just a single diagonal element whose omission is over-ridden by the change from $\widetilde{\mathbf{A}}_i$ to $\widehat{\mathbf{A}}_i$. Note that although the discarded elements are omitted from the triangular factors, they are not totally excluded from $\widehat{\mathbf{A}}_i$.

In practice the algorithm gives one of the most effective ILU preconditioners in terms of iteration speed. It also results in triangular factors that are guaranteed to be real and nonsingular for *any* symmetric positive definite matrix \mathbf{A} . This is in

sharp contrast with IC preconditioning (see page 233) which can fail if \mathbf{A} is not an M-matrix or an H-matrix. Its downside is that it requires more time and storage to set up the factors in the first place. Indeed for some problems, the storage required to compute the preconditioner approaches that needed for a complete factorization of the original problem if the latter's inherent fill-in has been reduced by a suitable re-ordering [32]. Other variants of the algorithm have been considered by Kaporin [167] and Suarjana and Law [230].

11.7.7. Numerical considerations

In preconditioned Krylov methods there are numerical implications over and above those discussed in Chapter 9, but there have been comparatively few papers devoted to this aspect of the subject. One of these is by Chow and Saad [67]. The present section is largely based on that paper, although we usually offer our own interpretation of the observed algorithmic behaviour reported therein.

Incomplete LU factorization is based on computing triangular matrices \mathbf{L} and \mathbf{U} such that $\mathbf{LU} = \mathbf{A} + \mathbf{R}$ (equation (11.115)) where \mathbf{R} is a “residual” matrix whose properties are determined by the sparsity patterns imposed upon the triangular factors. For good preconditioning:

- (1) \mathbf{LU} should be a “good” approximation to \mathbf{A} (i.e. $\|\mathbf{R}\|$ should be “small” in some sense) so that if all calculations are carried out in *exact arithmetic throughout* the iterative phase of the algorithm converges rapidly, and
- (2) \mathbf{L} and \mathbf{U} should not be excessively ill-conditioned so that
 - (a) they themselves may be computed accurately, i.e. if the computed factors $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ satisfy $\bar{\mathbf{L}}\bar{\mathbf{U}} - \mathbf{LU} = \mathbf{E}$ then $\|\mathbf{E}\|$ should also be “small”, and
 - (b) if conditions 1 and 2(a) are satisfied, the algorithm is not sabotaged by the inaccurate calculation of the preconditioned residuals during its iterative phase.

In order to determine whether or not $\|\mathbf{R}\|$ is small we need to look at the use to which it will be put. From the vector form of equation (3.21), ignoring subscripts, we see that we have to calculate \mathbf{Kf} where, for ILU preconditioning, $\mathbf{K} = (\mathbf{LU})^{-1}$. If we put $\mathbf{G} = \mathbf{A}$ we have, from equations (1.3) to (1.8), $\mathbf{f} = \mathbf{r} = \mathbf{Ae}$ so that the vector to be calculated is $(\mathbf{LU})^{-1} \mathbf{Ae}$ or, from equation (11.115), $(\mathbf{A} + \mathbf{R})^{-1} \mathbf{Ae}$. Denote this vector by \mathbf{p} . The “ideal” preconditioner \mathbf{K}_I is simply \mathbf{A}^{-1} so since $\mathbf{f} = \mathbf{Ae}$ the ideal correction \mathbf{p}_I is simply the error \mathbf{e} . To assess the quality of $(\mathbf{LU})^{-1}$ as a *preconditioner* it would therefore seem reasonable to compute something like $\sigma = \|\mathbf{p} - \mathbf{p}_I\| / \|\mathbf{p}_I\|$, which is an expression of the relative error introduced by performing (in exact arithmetic) incomplete as opposed to complete factorisation. If σ were small (less than 0.1 say) then we might expect that PCG would converge rapidly. Now from the above discussion and since $\mathbf{e} = \mathbf{p}_I$,

$$\mathbf{p} - \mathbf{p}_I = \mathbf{Sp}_I \tag{11.145}$$

where

$$\mathbf{S} = (\mathbf{A} + \mathbf{R})^{-1} \mathbf{A} - \mathbf{I}. \tag{11.146}$$

Thus, from equation (11.145), $\sigma \leq \|\mathbf{S}\|$ and since equation (11.146) may be rearranged to give

$$\mathbf{S} = -(\mathbf{I} + \mathbf{A}^{-1}\mathbf{R})^{-1}\mathbf{A}^{-1}\mathbf{R}$$

we have, if $\|\mathbf{A}^{-1}\mathbf{R}\| < 1$,

$$\sigma \leq \frac{\|\mathbf{A}^{-1}\mathbf{R}\|}{1 - \|\mathbf{A}^{-1}\mathbf{R}\|}. \quad (11.147)$$

If $\|\mathbf{A}^{-1}\mathbf{R}\|$ approaches unity it is therefore possible for the preconditioner $(\mathbf{L}\mathbf{U})^{-1}$ to be very poor. It follows from this discussion that for our purposes the phrase “ \mathbf{R} is small” means that $\|\mathbf{A}^{-1}\mathbf{R}\| \ll 1$, for which a sufficient condition is that $\|\mathbf{R}\| \ll 1/\|\mathbf{A}^{-1}\|$. Thus an excessive value of $\|\mathbf{S}\|$ can occur even when $\|\mathbf{R}\|$ is small if $\|\mathbf{A}^{-1}\|$ is sufficiently large. This possible discrepancy has led some authors to regard \mathbf{R} as a measure of the *accuracy* of an ILU decomposition and \mathbf{S} as a measure of its *stability*.

To examine the *numerical* instabilities associated with ILU we return to equation (11.115) which states that $\mathbf{LU} = \mathbf{A} + \mathbf{R}$. If we denote by $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ the computed values of \mathbf{L} and \mathbf{U} we would want $\bar{\mathbf{L}}\bar{\mathbf{U}}$ to be a good approximation of \mathbf{LU} , and this is most likely to be achieved if both \mathbf{L} and \mathbf{U} are well-conditioned. Another reason for wanting both \mathbf{L} and \mathbf{U} to be well-conditioned is that every step of the final iterative phase of the algorithm requires the calculation of the vector of preconditioned residuals $\mathbf{p} = \mathbf{K}\mathbf{f}$. For ILU preconditioning, $\mathbf{p} = (\bar{\mathbf{L}}\bar{\mathbf{U}})^{-1}\mathbf{f}$ and is computed by solving $\bar{\mathbf{L}}\mathbf{y} = \mathbf{f}$ and $\bar{\mathbf{U}}\mathbf{p} = \mathbf{y}$ by forward- and back-substitution respectively. Again, if $\bar{\mathbf{y}}$ and $\bar{\mathbf{p}}$ denote computed values, $\|\bar{\mathbf{y}} - \mathbf{y}\|$ and $\|\bar{\mathbf{p}} - \mathbf{p}\|$ are unlikely to be small if $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ are ill-conditioned. These effects were first observed by van der Vorst [238] and analysed by Elman [104].

We now consider the factors governing the conditioning of \mathbf{L} and \mathbf{U} . If κ_r denotes the condition number of the leading principal submatrix of $\mathbf{A} + \mathbf{R}$ of order r ($r = n$ gives $\mathbf{A} + \mathbf{R}$ itself) then [44]

$$k(\mathbf{L})k(\mathbf{U}) \geq \max_{1 \leq r \leq n} \kappa_r.$$

Let r_m denote that value of r for which κ_r is a maximum. If $r_m = n$ then the matrix itself provides a lower bound for $k(\mathbf{L})k(\mathbf{U})$ and unless this product is considerably greater than κ_n there is not a great deal that can be done except to reduce the condition number of $\mathbf{A} + \mathbf{R}$ by effectively changing \mathbf{R} . One way of doing this, if \mathbf{A} is symmetric positive definite, is by stabilisation (see below). If $r_m \neq n$ then normally at least one pivot is small and providing that there is no symmetry to be preserved, pivoting can be used. On the other hand if it is important to preserve symmetry or if pivoting does not work, it is again possible to try stabilisation.

Chow and Saad [67] defined three numerical indicators and used them to classify the types of failure observed in their tests on preconditioned GMRes. Four types of failure were identified and in slightly modified form are:

- (1) *Zero pivots.* Occur when computing $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$. The majority were “structurally zero”, i.e. not caused by cancellation, and mainly occurred in the absence of pivoting. The afflicted matrices tended to have irregular structures, frequently with many zero diagonal elements. The basic cause of this type of breakdown is the (often structural) singularity of a leading principal submatrix of $\mathbf{A} + \mathbf{R}$.
- (2) *Inaccuracy due to very small pivots.* Usually occurs in the absence of pivoting and is due to singularity or near-singularity of a leading principal submatrix of $\mathbf{A} + \mathbf{R}$. If this type of breakdown occurs when using pivoting then either the pivoting strategy is ineffective or $\mathbf{A} + \mathbf{R}$ is badly-conditioned.
- (3) *Unstable triangular solutions* (without very small pivots). This is an unusual type of behaviour since numerical instability in triangular decomposition is normally associated with small pivots. It is discussed more fully below.
- (4) *Inaccuracy due to dropping.* This problem is not strictly numerical since it can occur even when using exact arithmetic. It is due to $\|\mathbf{S}\|$ being excessive.

Two techniques were suggested to overcome the first two types of failure. The first of these is *stabilisation* in which the pivots (essentially the diagonal elements of $\bar{\mathbf{U}}$ if $\bar{\mathbf{L}}$ is unit lower triangular) are artificially bumped-up to some higher value in order to make the factorisation viable or to improve the condition number of the triangular factors. This is satisfactory [67] if \mathbf{A} is symmetric or nearly so, even if it is indefinite, but does not work well with very unstructured matrices. It is an integral component of the algorithms of Manteuffel [182] and Munksgaard [190] and has also been explored in [170], [210], [216] and [238], and in [161] for complete factorisation. Note that stabilisation is yet another “trade-off” technique since increasing the pivots, while reducing the condition numbers of \mathbf{L} and \mathbf{U} , increases $\|\mathbf{R}\|$. Overall, progressively increasing pivot size tends to yield a rapid initial improvement followed by a slow deterioration [182], [216].

The second technique for avoiding problems due to small or zero pivots is good old-fashioned *pivoting* which gives rise to the ILUTP version of ILUT. Pivoting is particularly suitable for very unstructured matrices with many structurally zero pivots. However probably the most common way of representing a sparse matrix is by some structure that links the elements of each row together, as is done in the compressed sparse row format (see e.g. [108], p.85). With these representations, checking the elements of a given column for size is inefficient whereas it is comparatively simple to determine the largest element of a particular row. For this reason sparse pivoting algorithms usually rely on column interchanges. Note though that although column interchanges will always yield a nonzero pivot when applied to LU decomposition, this is not the case when applied to incomplete decomposition.

Chow and Saad observe that the third type of problem, unstable triangular solutions, is rare for complete (but not incomplete) factorisation and is the most difficult of the four types of failure to overcome. While not being able to provide a watertight explanation of what happens we can consider a simple example that might give some sort of insight. Define \mathbf{A}_1 and \mathbf{A}_2 by

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 10 & 0 \\ 10 & 101 & 10 \\ 0 & 10 & 101 \end{bmatrix} \quad \text{and} \quad \mathbf{A}_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1.01 & 0.1 \\ 0 & 0.1 & 1.01 \end{bmatrix}.$$

Despite its modest dimensions and innocuous appearance, \mathbf{A}_1 is quite ill-conditioned with $k(\mathbf{A}_1) = 1.14 \times 10^6$. Nevertheless if factorised without pivoting by either LU or Cholesky decomposition, all three pivots are unity. However in practice it is unlikely that such a matrix would be subjected to triangular decomposition without some form of prior scaling to bring the rowsums more into agreement. One possibility would be to scale both the second and third rows and columns (in order to preserve symmetry) by one-tenth to give \mathbf{A}_2 . The condition number is still large ($k(\mathbf{A}_2) = 4.08 \times 10^4$) but considerably smaller than that of \mathbf{A}_1 . However when \mathbf{A}_2 is factorised the diagonal elements of the upper triangular factor are 1, 0.1 and 0.1 (Cholesky) or 1, 0.1 and -0.001 (LU, where the negative pivot results from a row interchange). Thus the effect of scaling is to make the ill-conditioning show up in the size of the pivots, especially for LU decomposition. Since scaling is normally applied we would generally expect, if this example is typical, that ill-conditioning would reveal itself as strongly unequal pivots, and it usually does. Although this is just an illustrative example it might be the case that pivots of similar sizes are associated with unequal scaling, and that this type of scaling can result from carrying out certain kinds of incomplete LU factorisation, but this is only conjecture. In practice Chow and Saad state that severe ill-conditioning of \mathbf{L} and \mathbf{U} where the pivots are of comparable size can be ameliorated by thresholding the pivots using very large thresholds, but this can be done only at the expense of accuracy. See their paper [67] for a fuller discussion.

The fourth cause of failure, inaccuracy due to dropping, is comparatively simple to overcome. The remedy is to permit increased fill-in to reduce $\|\mathbf{R}\|$. Of the six failures in the tests of Chow and Saad that the authors attributed to this defect (see below), five were cured by using higher levels of fill. However they suggested that increasing the level of fill for failures due to other causes was unlikely to give any improvement.

Chow and Saad carried out a series of numerical tests using a set of 36 test problems selected from the Harwell-Boeing, UMFPACK and SPARSKIT collections whose dimensions varied between $n = 363$ and $n = 38744$. None of these problems could be solved by GMRes using ILU(0) as a preconditioner. All the matrices were scaled so that their columns had unit Euclidean norms and then scaled again so that their rows had unit Euclidean norms. Without this scaling there were too many zero pivots when computing the preconditioners. The single iterative method used was right-preconditioned GMRes(50). The iterations began with a zero initial guess and were regarded as a success if the true residual norm was reduced by eight orders of magnitude in fewer than 500 iterations. Table 11 indicates the number of problems that failed due to zero pivots (zp), small pivots (sp), unstable triangular solutions (uts) and inaccuracy due to dropping (id) for three different preconditioners. The final column gives the number of successes. The preconditioners were ILU(0) and two versions of ILUTP, with a relatively large (30) maximum permitted number

of non-zeroes in each row of \mathbf{L} and each row of \mathbf{U} . Pivoting took place if the off-diagonal element a_{ij} of largest modulus satisfied $|a_{ij}| > |a_{ii}|/\tau$, where τ is an arbitrary constant. The two values chosen were $\tau = 1$ (normal pivoting) and $\tau = 0.01$ (pivoting only if $|a_{ii}|$ is relatively quite small).

Table 11

Failure type	zp	sp	uts	id	ok
ILU(0)	19	5	6	6	0
ILUTP($\tau = 1$)	2	1	8	3	22
ILUTP($\tau = 0.01$)	5	1	11	1	18

A brief glance at the table indicates that the threshold methods with pivoting were, for the problems selected, far superior to the unpivoted level-based version but it is not clear from the table if this is due to moving to a threshold-based method or to the introduction of pivoting. The authors note though that they did not include in the table the results for ILUTP($\tau = 0$) (no pivoting) as these were “extremely poor”. This strongly suggests that the principal reason for the observed improvement was the use of pivoting, a conclusion not contradicted by the last two rows of the table.

A subset of 13 structurally symmetric problems was then selected from the original 36, the table analogous to Table 11 for these problems being

Table 12

Failure type	zp	sp	uts	id	ok
ILU(0)	5	2	3	3	0
ILUTP($\tau = 1$)	1	0	2	1	9
ILUTP($\tau = 0.01$)	2	0	4	0	7

These structurally symmetric problems exhibit the same general characteristics as those in the larger collection. The matrices were then re-ordered to ensure that there would be no structurally zero pivots (possible for these particular matrices). Despite this, only ten problems were then solved by a stabilised version of ILUT compared to the nine solved by ILUTP($\tau = 1$) applied to the original (un-reordered) systems. ILU(1), on the other hand, solved all 13 problems in their re-ordered form. It would therefore appear that for problems with non-symmetric structure, pivoting could be more effective than stabilisation although for a more definitive conclusion a greater number of examples would be needed.

11.7.7.1. $ILU(p)$ versus $ILUT$

To sum up the various difference between $ILU(p)$ and $ILUT$:

$ILUT$ with fill-in limitation

- generally yields more accurate factors than $ILU(p)$
- is more prone to unstable triangular solution since the off-diagonal elements are generally larger, the largest elements in \mathbf{L} and \mathbf{U} being systematically retained

- propagates large and inaccurate entries that may have been computed via a small and inaccurate pivot (this does not happen when a level-of-fill rule is used).
- limits the fill-in in each row but not in each column. This may produce a very nonsymmetric preconditioner even for structurally symmetric problems. Symmetry may be imposed by using the bordered form of factorization.

The principal drawback of ILU(p) factorization is the amount of labour required to establish the required sparsity pattern. See [67] for further details.

11.7.8. The impact of re-ordering

Since pivoting can have such a profound impact on ILU preconditioning (see Numerical considerations, page 252) and since, for implementational reasons, pivoting is usually restricted to column interchanges it was natural that another way of improving the quality of preconditioners involving row interchanges would be sought. One such way is *re-ordering*, where $\mathbf{Ax} = \mathbf{b}$ is replaced by $\{\mathbf{PAx} = \mathbf{Pb}\}$ or by $(\mathbf{PAP}^T)\mathbf{Px} = \mathbf{Pb}$ if it is desired} to maintain symmetry. These changes are made once-for-all before the start of the algorithm proper. There are three principal reasons for re-ordering. These are:

- the reduction of the fill-in incurred during triangular decomposition
- the improvement of the rate of convergence of the subsequent iterative phase of the algorithm, and
- rendering more suitable for parallel computation the triangular factors obtained by ILU decomposition.

The first of these reasons is equally applicable to *complete* LU decomposition and several algorithms have been motivated by this. Two of the best-known ones are the bandwidth-reduction algorithm of Cuthill and McKee [81] (together with the reverse variation due to George [127]) and *minimum degree ordering* of Tinney and Walker [233] with modifications by Liu [179]. Although the objective of both these schemes is the same, the approaches adopted and results obtained are entirely different.

The Cuthill and McKee algorithm is based on the observation that fill-in during LU decomposition without pivoting is confined to positions within the band so that a smaller bandwidth reduces the potential for fill-in.

Definition 15. A matrix $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$ is said to be a band matrix, or to be banded, if $a_{ij} \neq 0$ only if $i + m_L \leq j \leq i + m_R$ for some constants m_L and m_R . The bandwidth is defined to be $m_R - m_L + 1$. Generally $m_L m_R \leq 0$ implying that the band includes the main diagonal.

A well-known example of a band matrix is a tri-diagonal matrix which has a bandwidth of three and for which $m_L = -1$ and $m_R = 1$.

Equations derived from highly-structured problems and ordered naturally can often have a simple structure and a low bandwidth (row ordering for the model problem is a case in point). For other problems with a less regular structure it

is much more difficult to obtain a low bandwidth, and for these some form of algorithm based on the matrix itself is needed. Cuthill and McKee set out to reduce the bandwidth by carrying out a sequence of *symmetric permutations*¹⁸. Initially rows and columns of \mathbf{A} are permuted so that if $\mathbf{A}^{(1)} = [a_{ij}^{(1)}]$ denotes the matrix obtained after the first permutation, the n_1 (say) nonzero elements of the first row of $\mathbf{A}^{(1)}$ occupy the positions $a_{11}^{(1)}, a_{12}^{(1)}, \dots, a_{1,n_1}^{(1)}$, (it is assumed that $a_{11} \neq 0$ so that the first row and column of \mathbf{A} can remain *in situ*). Then if any elements $a_{ij}^{(1)}$ for $2 \leq i \leq n_1$ and $n_1 + 1 \leq j \leq n$ are nonzero the columns containing these elements, together with the corresponding rows, are permuted to become columns $n_1 + 1, n_1 + 2, \dots, n_2$ (say) of $\mathbf{A}^{(2)}$. This process is then repeated for the nonzero elements of $\mathbf{A}^{(2)}$ in columns $n_2 + 1$ to n , and so on, and eventually yields a block lower-triangular matrix. If \mathbf{A} is *structurally symmetric*¹⁹ the end result is a block tridiagonal matrix for which the triangular factors will be block bi-diagonal.

Minimum degree ordering is obtained by simulating an actual triangular decomposition and symmetrically permuting the rows and columns of \mathbf{A} so that at every step of the algorithm the fill-in due to that step is minimised. Since fill-in is independent of the numerical values of the elements of \mathbf{A} (ignoring possible cancellation) this simulation may be performed not on \mathbf{A} itself but with a “mask” of \mathbf{A} which has ones wherever \mathbf{A} has nonzero elements. The procedure is reminiscent of, and simpler than, the determination of level-of-fill (see page 233). The philosophy underlying this process is one of obtaining the greatest immediate improvement (as in greedy algorithms or the method of steepest descent) and does not yield the overall optimal solution. It does, though, generally give less fill-in than Cuthill and McKee.

Both these algorithms were originally developed to simplify *complete* LU decomposition and it was by no means clear that they would be suitable for ILU preconditioning. Indeed some authors have suggested otherwise, probably on the basis of insufficient evidence. On the other hand Duff and Meurant [97] tested no fewer than 17 different ordering strategies on a raft of symmetric positive definite problems using four different incomplete Cholesky preconditioners. Their tables show that a group of orderings that included row, Cuthill and McKee and reverse Cuthill and McKee (RCM), see [127], consistently resulted in virtually the same minimum or near-minimum number of iterations for all the problems in the representative set selected for discussion. However, the red-black and minimum degree orderings did comparatively badly. For one problem they needed more than five times the number of iterations required by the best methods to obtain a solution and for another problem, where the diagonal elements of the matrix had been modified in order to satisfy equation (11.130), they failed to converge at all. Perhaps the most important conclusion of Duff and Meurant was that the quality of the preconditioner depends on $\|\mathbf{R}\|$ (see equation (11.115)) and not, as had been suggested, on the number of its

¹⁸A symmetric permutation is one where identical permutations are carried out to both rows and columns of the matrix.

¹⁹A structurally symmetric matrix $\mathbf{A} = [a_{ij}]$ is one for which $a_{ji} = 0$ if and only if $a_{ij} = 0$.

non-zero elements. This is in accordance with intuition and has been theoretically justified under certain conditions by Axelsson and Eijkhout [13]. In particular the minimum degree ordering generally gave few non-zero elements but these few were often large enough to severely damage the algorithm's convergence properties, a characteristic also observed by Benzi *et al* [28] for other types of matrix. However, it does not hold for strongly nonsymmetric matrices (see below). Another useful conclusion was the observation that whereas for most orderings the reduction in the number of iterations when going from IC(0) to IC(1) did not quite compensate for the extra work per iteration (extra because with IC(1) there are more non-zero matrix entries to take part in the calculation), the opposite was true for a threshold version of IC. It was also noted that the best orderings for reducing the number of iterations were amongst the poorest for exploiting parallelism, yet another example of the trade-offs that challenge numerical analysts.

Another ordering algorithm that was developed specifically for use with ILU preconditioners is the method of *minimum discarded fill (MDF)* of D'Azevedo *et al* [86]. It is essentially a straightforward ILU(p) algorithm (see above) but with pivoting based on the *numerical* value of the discarded elements. Thus if the interchanges required by pivoting could be carried out before the start of the calculation, the results obtained by standard ILU(p) would be the same as those obtained by the proposed algorithm. This is underlined by D'Azevedo *et al* who regard their algorithm as an ordering rather than as a pivoting one.

The algorithm itself is based on the identity

$$\mathbf{A}_{k-1} \equiv \mathbf{L}_{k-1} \mathbf{M}_{k-1} \mathbf{U}_{k-1} + \mathbf{R}_{k-1} \quad (11.148)$$

where

$$\begin{aligned} \mathbf{A}_{k-1} &\equiv \begin{bmatrix} \delta_k & \mathbf{a}_k^T \\ \mathbf{b}_k & \mathbf{B}^{(k)} \end{bmatrix}, & \mathbf{M}_{k-1} &\equiv \begin{bmatrix} \delta_k & \mathbf{0}^T \\ \mathbf{0} & \mathbf{A}_k \end{bmatrix}, \\ \mathbf{L}_{k-1} &\equiv \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{b}_k \delta_k^{-1} & \mathbf{I} \end{bmatrix}, & \mathbf{U}_{k-1} &\equiv \begin{bmatrix} 1 & \delta_k^{-1} \mathbf{a}_k^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \end{aligned}$$

and

$$\mathbf{R}_{k-1} \equiv \begin{bmatrix} \mathbf{0} & \mathbf{0}^T \\ \mathbf{0} & \mathbf{F}^{(k)} \end{bmatrix}.$$

The matrix \mathbf{A}_{k-1} is a submatrix of a partially-factorised version of \mathbf{A} and $\mathbf{F}^{(k)}$ is defined below. The matrices $\mathbf{C}^{(k)}$ and \mathbf{A}_k are given by

$$\mathbf{C}^{(k)} \equiv \mathbf{b}_k \delta_k^{-1} \mathbf{a}_k^T$$

and

$$\mathbf{A}_k \equiv \mathbf{B}^{(k)} - \mathbf{C}^{(k)} - \mathbf{F}^{(k)}, \quad (11.149)$$

where $\mathbf{B}^{(k)}$, \mathbf{A}_k , $\mathbf{C}^{(k)}$, $\mathbf{F}^{(k)} \in \mathbb{R}^{(n-k) \times (n-k)}$.

Suppose now that $\mathbf{A}_1 = \mathbf{A}$ and that equation (11.148) holds for $k = 1, 2, \dots, n-1$. Then, since \mathbf{L}_k and \mathbf{U}_k are unit lower and unit upper triangular matrices respectively we obtain, after some manipulation,

$$\mathbf{A} = \mathbf{LDU} + \mathbf{R}$$

where $\mathbf{D} = \text{diag}(\delta_i)$. The matrices \mathbf{L} and \mathbf{U} are unit lower and upper triangular and \mathbf{R} is obtained from the matrices \mathbf{F}_k , $k = 1, 2, \dots, n$. If these are null then we have a conventional LDU factorisation of \mathbf{A} . If not and they are chosen appropriately we will have an ILU decomposition of \mathbf{A} . The algorithm hinges on this choice. If $\mathbf{F}^{(k)} = \begin{bmatrix} f_{ij}^{(k)} \end{bmatrix}$, etc., then D'Azevedo *et al* put $f_{ij}^{(k)} = 0$ if $b_{ij}^{(k)} \neq 0$ and, if $b_{ij}^{(k)} = 0$,

$$f_{ij}^{(k)} = \begin{cases} -c_{ij}^{(k)} & \text{if } \text{lev}(c_{ij}^{(k)}) > p \\ 0 & \text{otherwise.} \end{cases}$$

Thus if $b_{ij}^{(k)} = 0$ and $\text{lev}(c_{ij}^{(k)}) > p$ then, from equation (11.149), element (i, j) of \mathbf{A}_k is also zero and any potential fill-in has been suppressed. The resulting discrepancy has been transferred to \mathbf{F}_k .

The matrices \mathbf{F}_k now come into their own. Assume for simplicity that \mathbf{A} is symmetric positive definite. Since $\mathbf{A}_{k-1} \in \mathbb{R}^{(n+1-k) \times (n+1-k)}$ there are, if symmetric row/column pivoting is permitted, $(n+1-k)$ choices of pivot, each with its own \mathbf{F}_k . The one chosen is that which minimises $\|\mathbf{F}_k\|_F$. Much of [86] deals with the mechanism of computing these matrices and the associated levels of fill, for which the original paper should be consulted.

In a recent (1999) study, Benzi, Szyld and van Duin [28] considered the effects of re-ordering on a selection of non-symmetric matrices (some quite strongly non-symmetric) but which were structurally symmetric or nearly so. The problems attempted fell naturally into two groups. The first, G1, consisted of essentially one basic convection-diffusion problem whose difficulty and degree of non-symmetry could be controlled by a parameter. Ten values of this parameter were chosen to give a wide range of typical behaviours. The second group, G2, consisted of seven unrelated problems whose difficulty ranged from “moderate” to “extreme”. These were, in general, more ill-conditioned and had more complicated structures than those in the first set. The authors tested all possible combinations of the solvers GMRes(20), BiCGStab and TFQMR with the preconditioners ILU(0), ILU(1), ILUT($10^{-2}, 5$) and ILUT($10^{-3}, 10$) combined with minimum degree, Cuthill and McKee and reverse Cuthill and McKee re-orderings in addition to the “normal” ordering in terms of which the problems were originally presented. These were applied to all 17 test problems, a total of 816 tests in all. They also tested the minimum discarded fill (MDF) algorithm [85], [84] (effective but generally too expensive to be practical) and, for group G1, re-orderings based on one-way and nested dissection (similar results to those given by minimum degree re-ordering).

In addition to the number of iterations required to reduce the residual norm to 10^{-4} for G1 and 10^{-9} for G2 (a welcome acknowledgment that for more difficult problems *more*, and not *less*, precision of the residuals is required to obtain the

same accuracy of solution), values of $\|\mathbf{R}\|_F$ and $\left\| \mathbf{I} - \mathbf{A} (\overline{\mathbf{L}}\mathbf{U})^{-1} \right\|_F$ were quoted for the problems in group G1. The authors presented a very detailed analysis of their findings, of which the main points are:

- the more difficult the problem, the greater the effect of re-ordering
- the original (natural) orderings were not particularly good. This was in contrast to the results obtained by Duff and Meurant [97] for symmetric problems
- the more fill-in, the greater the likelihood of subsequent convergence (or the greater speed of that convergence). Thus ILUT($10^{-3}, 10$) is more successful than ILUT($10^{-2}, 5$) which in turn is better than ILU(1) and ILU(0)
- there is, in general, a far better correlation between the speed of convergence and $\left\| \mathbf{I} - \mathbf{A} (\overline{\mathbf{L}}\mathbf{U})^{-1} \right\|_F$ than there is between the speed of convergence and $\|\mathbf{R}\|_F$. This is less true for comparatively easy problems
- overall the best (fastest and most reliable) re-ordering is reverse Cuthill and McKee. However there were some problems that could only be solved using minimum degree re-ordering. In particular the combination

$$\text{minimum degree}/\text{ILUT}(10^{-3}, 10)$$

with either BiCGStab or TFQMR²⁰ were the *only* algorithms that could solve all 17 problems within the maximum permitted number (500) of matrix-vector products.

The need for re-ordering prior to carrying out ILU is most marked for very unstructured matrices with perhaps many zeroes on the diagonal. With re-ordering it is hoped to reduce or even eliminate the need for interchanges during ILUTP and to give, if possible, a matrix with better numerical properties, e.g. more nearly diagonally dominant. If we were to replace re-ordering by a general matrix operation, both these ends could be achieved simply by multiplying $\mathbf{Ax} = \mathbf{b}$ by \mathbf{A}^T but this would create unwanted fill-in and square the condition number of the matrix. Another possibility is to multiply the equations by \mathbf{A}_1^T where \mathbf{A}_1 is some skeletal approximation to \mathbf{A} . This is an alternative way of looking at some recent algorithms that were differently motivated (see below) and which have been developed since the publication of [28]. For example, \mathbf{A}_1 could be chosen so that each of its columns had just one non-zero element equal both in value and position (in the column) to the element of largest absolute value in the corresponding column of \mathbf{A} . Then $\mathbf{A}_1^T \mathbf{A}$ would have positive diagonal elements and would be in a sense more symmetric than \mathbf{A} , and some move in the direction of diagonal dominance would have been made. If it so happened that each *row* of the matrix \mathbf{A}_1 thus computed also had just one non-zero element then \mathbf{A}_1 would be a scaled permutation matrix and would be nonsingular. If some rows of \mathbf{A}_1 had no non-zero elements a different strategy would have to be used in its construction, either by moving some non-zeroes to more favourable locations or (perhaps) by adding further non-zeroes to selected columns. If chosen appropriately this could improve the numerical properties of $\mathbf{A}_1^T \mathbf{A}$ at the expense of introducing some fill-in.

²⁰Both algorithms, be it noted, derived from CGS.

The idea of pre-multiplying $\mathbf{Ax} = \mathbf{b}$ by a scaled permutation matrix was due to Olschowka and Neumaier [196] and has been developed by Duff and Koster [95], [96]. Additional refinements include fill-in reduction by one of the standard routines (RCM or minimum degree), equivalent to pre- and post-multiplication by permutation matrices \mathbf{P}^T and \mathbf{P} respectively, and column scaling (post-multiplication by a diagonal matrix \mathbf{D}). The matrices \mathbf{A}_1 and \mathbf{D} were not chosen according to the above criterion but with a view to maximising the product of the absolute values of the diagonal elements of the resulting matrix $\mathbf{P}^T \mathbf{A}_1^T \mathbf{ADP}$. The final equations to be solved were

$$\mathbf{P}^T \mathbf{A}_1^T \mathbf{ADPy} = \mathbf{P}^T \mathbf{A}_1^T \mathbf{b} \quad \text{and} \quad \mathbf{x} = \mathbf{DPy}.$$

The method was tested by Benzi *et al* [25] and in [23] Benzi reviewed a selection of five of the problems tested. Of these five, none could be solved by ILUT (because of zero pivots, see page 254) and only three by ILUTP. All could be solved easily by the above method followed by ILUT, a combination that represents one of the more promising recent developments in the field of ILU preconditioning.

11.8. Methods for parallel computers

Preconditioning based on incomplete LU decomposition is both effective and robust, its only drawback being that since it is inherently sequential it does not work too well on vector or parallel computers. It is not therefore surprising that other, more suitable, methods have been developed and this section sets out to describe some of the more important. It relies heavily for its choice of subject matter, content and nomenclature on the comparative study of Benzi and Tůma [31].

We divide the algorithms considered here into the following five groups:

- (1) Sparse approximate inverse matrix (AIM) methods. In these methods a sparse approximation \mathbf{K} to \mathbf{A}^{-1} is sought, either by minimising $\|\mathbf{I} - \mathbf{AK}\|_F$ for a given sparsity structure of \mathbf{K} as in SpAI or by the use of a truncated dropping version of Richardson's iteration as in MR (we use the term *dropping* to indicate an algorithm that has been modified to replace by zero some of its normally-computed results in order to satisfy some level-of-fill criterion or drop tolerance).
- (2) Sparse approximate inverse factor (AIF) methods. Here sparse approximations $\widehat{\mathbf{L}}$ and $\widehat{\mathbf{U}}$ of \mathbf{L}^{-1} and \mathbf{U}^{-1} are computed, where $\mathbf{A} = \mathbf{LDU}$, \mathbf{L} and \mathbf{U} are unit triangular and \mathbf{D} is diagonal (AIB and AInv) or where $\mathbf{A} = \mathbf{LL}^T$ (FSAI). The methods used vary from dropping versions of LU decomposition (AIB, AInv) to one that minimises $\|\mathbf{I} - \widehat{\mathbf{L}}\widehat{\mathbf{L}}^T\|_F$ (FSAI).
- (3) Polynomial preconditioners (PP). This group consists of a somewhat mixed collection of algorithms and includes a method of Johnson *et al* (referred to by Benzi and Tůma as JP - Jacobi polynomial) together with some truncated

Neumann methods (TNSGS and TNIC) based on the symmetric Gauss-Seidel and incomplete Cholesky algorithms respectively.

- (4) Approximate sparse inverse of approximate sparse factor (AIAF) methods. In these methods sparse approximate triangular factors $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ are computed and their approximate sparse inverses $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ are determined using a selection of the other methods described in this chapter.
- (5) Other methods. This group consists of various methods, most based on partial differential equations.

We begin with a discussion of the first of these categories.

11.8.1. The AIM methods ($SpAI$, MR and MRP)

It might be thought odd to approximate the *inverse* of a sparse matrix \mathbf{A} by another sparse matrix since the inverse of a sparse matrix is generally dense [93], but it is often the case that many of the elements of \mathbf{A}^{-1} are relatively small and may be ignored (though this does not necessarily guarantee a good preconditioner). Once the “large” elements have been identified it is a comparatively simple matter to compute approximations to them.

Define then the residual matrix $\mathbf{R} \in \mathbb{R}^{n \times n}$ by

$$\mathbf{R} = \mathbf{I} - \mathbf{A}\mathbf{K} \quad (11.150)$$

and assume that the sparsity pattern of \mathbf{K} has somehow been determined. Denote by \mathbf{r}_i the i th column of \mathbf{R} . Then $\mathbf{r}_i = \mathbf{e}_i - \mathbf{A}\mathbf{K}\mathbf{e}_i$, where $\mathbf{K}\mathbf{e}_i$, the i th column of \mathbf{K} , is required to have a given sparsity pattern. This may be imposed by putting

$$\mathbf{K}\mathbf{e}_i = \mathbf{S}_i\mathbf{x}$$

where \mathbf{S}_i is the matrix consisting of those columns of the unit matrix corresponding to the permitted nonzero elements²¹. For example, if all but the first, fifth and seventh elements of $\mathbf{K}\mathbf{e}_i$ are required to be zero then $\mathbf{S}_i = [\mathbf{e}_1, \mathbf{e}_5, \mathbf{e}_7]$. The vector $\mathbf{S}_i\mathbf{x}$ will have zero in every position for which $\mathbf{K}\mathbf{e}_i$ is required to be zero and the remaining elements, i.e. the elements of \mathbf{x} , may be determined by solving for \mathbf{x} the overdetermined system

$$\mathbf{r}_i = \mathbf{e}_i - \mathbf{A}\mathbf{S}_i\mathbf{x} \quad (11.151)$$

according to some suitable criterion that “minimises” \mathbf{r}_i . One such criterion is the requirement that $\mathbf{S}_i^T \mathbf{r}_i = \mathbf{0}$ and \mathbf{x} is then obtained by solving

$$\mathbf{S}_i^T \mathbf{A} \mathbf{S}_i \mathbf{x} = \mathbf{S}_i^T \mathbf{e}_i. \quad (11.152)$$

Another criterion for choosing \mathbf{x} is the minimisation of $\|\mathbf{r}_i\|$, and the required value of \mathbf{x} is then the solution of

$$\mathbf{S}_i^T \mathbf{A}^T \mathbf{A} \mathbf{S}_i \mathbf{x} = \mathbf{S}_i^T \mathbf{A}^T \mathbf{e}_i. \quad (11.153)$$

²¹Strictly the not necessarily zero elements.

If carried out for $i = 1, 2, \dots, n$, this process minimises $\|\mathbf{R}\|_F$ for a given sparsity pattern of \mathbf{K} since $\|\mathbf{R}\|_F^2 = \sum_{i=1}^n \|\mathbf{r}_i\|^2$ and the minimisation of each $\|\mathbf{r}_i\|$ involves only the i th column of \mathbf{K} . If the number of columns of \mathbf{S}_i (i.e. the number of nonzero elements of $\mathbf{K}\mathbf{e}_i$) is small and $\mathbf{S}_i^T \mathbf{A}^T \mathbf{A} \mathbf{S}_i$ is nonsingular for all i then the matrix \mathbf{K} can be computed column by column by solving, generally by QR decomposition, a series of least-squares problems based on equation (11.151).

For SpAI, $\|\mathbf{I} - \mathbf{A}\mathbf{K}\|_F$ is minimised initially for a given sparsity pattern of \mathbf{K} (i.e. set of matrices \mathbf{S}_i). This, however, is only the beginning and the idea was extended to enable the dynamic determination of \mathbf{S}_i . Let $\mathbf{AS}_i = \mathbf{A}_j$, where the subscript j now denotes the number of columns of \mathbf{A}_j . Equation (11.151) with the subscript on \mathbf{r}_i omitted becomes

$$\mathbf{r} = \mathbf{e}_i - \mathbf{A}_j \mathbf{x} \quad (11.154)$$

and the value of \mathbf{x} that minimises $\|\mathbf{r}\|$, \mathbf{x}_j say, is the solution of

$$\mathbf{A}_j^T \mathbf{A}_j \mathbf{x} = \mathbf{A}_j^T \mathbf{e}_i. \quad (11.155)$$

Substituting \mathbf{x}_j in equation (11.154) gives

$$\mathbf{r}_j = \mathbf{P}_j \mathbf{e}_i \quad (11.156)$$

where

$$\mathbf{P}_j = \mathbf{I} - \mathbf{A}_j (\mathbf{A}_j^T \mathbf{A}_j)^{-1} \mathbf{A}_j^T \quad (11.157)$$

and \mathbf{r}_j now denotes the minimised residual corresponding to \mathbf{A}_j . The problem now is how to augment \mathbf{A}_j . Grote and Huckle [137] did this by inserting a non-zero element θ into a hitherto zero k th position of the sparse i th column of \mathbf{K} . Since \mathbf{x}_j is assumed to have already been computed this gives $\mathbf{x}(\theta) \equiv \mathbf{x}_j + \mathbf{e}_k \theta$, and the corresponding residual norm $\|\mathbf{r}(\theta)\|$ may then be minimised as a function of the single variable θ . Equation (11.154) gives $\mathbf{r}(\theta) \equiv \mathbf{e}_i - \mathbf{A}_j \mathbf{x}_j - \mathbf{a}_k \theta \equiv \mathbf{r}_j - \mathbf{a}_k \theta$ and $\|\mathbf{r}(\theta)\|$ is minimised by $\theta = \mathbf{a}_k^T \mathbf{r}_j / \mathbf{a}_k^T \mathbf{a}_k$. The associated residual is given by $(\mathbf{I} - \mathbf{a}_k (\mathbf{a}_k^T \mathbf{a}_k)^{-1} \mathbf{a}_k^T) \mathbf{r}_j$ and if this is denoted by \mathbf{r}_{jk} then

$$\|\mathbf{r}_{jk}\|^2 = \|\mathbf{r}_j\|^2 - \left(\frac{\mathbf{a}_k^T \mathbf{r}_j}{\|\mathbf{a}_k\|} \right)^2. \quad (11.158)$$

The quantity $\mathbf{a}_k^T \mathbf{r}_j / \|\mathbf{a}_k\|$ may be computed for various vectors \mathbf{a}_k , and those giving the smallest $\|\mathbf{r}_{jk}\|^2$ chosen as the columns of \mathbf{A} to be appended to \mathbf{A}_j . We note from equation (11.154) that since \mathbf{A} is sparse it follows that both \mathbf{r}_j and \mathbf{a}_k are also sparse, and it is thus likely that $\mathbf{a}_k^T \mathbf{r}_j = 0$ for many values of k . However $\mathbf{a}_k^T \mathbf{r}_j$ cannot be zero for all k . If it were it would imply that $\mathbf{A}^T \mathbf{r}_j = \mathbf{0}$ since, from equations (11.156) and (11.157), $\mathbf{A}_j^T \mathbf{r}_j = \mathbf{0}$. This in turn would imply, since \mathbf{A} is nonsingular, that $\mathbf{r}_j = \mathbf{0}$, contrary to hypothesis. For a more thorough discussion of the algorithm and for various computational refinements, see the original papers.

We conclude our analysis of SpAI by noting that \mathbf{x}_{jk} does not normally minimise $\|\mathbf{r}\|$ over all the columns of \mathbf{A}_{j+1} where

$$\mathbf{A}_{j+1} = [\mathbf{A}_j \ \mathbf{a}_k]. \quad (11.159)$$

To determine what does, define $\mathbf{U}_j \in \mathbb{R}^{j \times j}$ to be an upper triangular matrix such that if

$$\mathbf{V}_j = \mathbf{A}_j \mathbf{U}_j \quad (11.160)$$

then the columns of \mathbf{V}_j are mutually orthonormal (Gram-Schmidt orthogonalisation). This is always possible since \mathbf{A}_j has rank j . Substituting $\mathbf{V}_j \mathbf{U}_j^{-1}$ for \mathbf{A}_j in equation (11.157) then gives

$$\mathbf{P}_j = \mathbf{I} - \mathbf{V}_j \mathbf{V}_j^T. \quad (11.161)$$

Equations (11.160) with $j+1$ replacing j and (11.159) yield

$$\mathbf{v}_{j+1} = \mathbf{A}_j \mathbf{u}_{j+1} + \mathbf{a}_k \mu_k \quad (11.162)$$

for some vector \mathbf{u}_{j+1} and scalar μ_k . But since, from the orthonormality of the columns of \mathbf{V}_{j+1} and equations (11.161) and (11.157), $\mathbf{P}_j \mathbf{v}_{j+1} = \mathbf{v}_{j+1}$ and $\mathbf{P}_j \mathbf{A}_j = \mathbf{O}$, premultiplication of equation (11.162) by \mathbf{P}_j gives

$$\mathbf{v}_{j+1} = \mathbf{P}_j \mathbf{a}_k / \|\mathbf{P}_j \mathbf{a}_k\| \quad (11.163)$$

where μ_k has been chosen to normalise \mathbf{v}_{j+1} . Now from the orthogonality of the columns of \mathbf{V}_{j+1} and equation (11.161), $\mathbf{P}_{j+1} = (\mathbf{I} - \mathbf{v}_{j+1} \mathbf{v}_{j+1}^T) \mathbf{P}_j$ and it follows from equation (11.156) that $\mathbf{r}_{j+1} = (\mathbf{I} - \mathbf{v}_{j+1} \mathbf{v}_{j+1}^T) \mathbf{r}_j$. Thus $\|\mathbf{r}_{j+1}\|^2 = \|\mathbf{r}_j\|^2 - (\mathbf{v}_{j+1}^T \mathbf{r}_j)^2$ or, from equation (11.163),

$$\|\mathbf{r}_{j+1}\|^2 = \|\mathbf{r}_j\|^2 - \left(\frac{\mathbf{a}_k^T \mathbf{P}_j \mathbf{r}_j}{\|\mathbf{P}_j \mathbf{a}_k\|} \right)^2.$$

Finally, from the idempotency of \mathbf{P}_j and equation (11.156), $\mathbf{P}_j \mathbf{r}_j = \mathbf{r}_j$ so that

$$\|\mathbf{r}_{j+1}\|^2 = \|\mathbf{r}_j\|^2 - \left(\frac{\mathbf{a}_k^T \mathbf{r}_j}{\|\mathbf{P}_j \mathbf{a}_k\|} \right)^2. \quad (11.164)$$

Thus provided that $\mathbf{P}_j \mathbf{a}_k$ can be computed reasonably cheaply for a selection of vectors \mathbf{a}_k it is possible to use this equation rather than equation (11.158) to choose the vectors to be adjoined to \mathbf{A}_j to give \mathbf{A}_{j+1} .

The idea of minimising $\|\mathbf{I} - \mathbf{AK}\|_F$ for some fixed structure of \mathbf{K} was first proposed by Benson [19] in 1973. Other early contributions were made by Benson and Frederickson [20], Benson *et al* [21] and Frederickson [114]. Dynamically modifying \mathbf{S}_i by the addition of further columns up to a given limit or until $\|\mathbf{e}_i - \mathbf{AS}_i \mathbf{x}\| < \epsilon$ was first proposed by Cosgrove *et al* [78]. This modification also forms the basis of the method of Grote and Huckle [137] who used equation (11.158) to determine the columns to be adjoined to \mathbf{A}_j and gave the method the name SpAI (Sparse

Approximate Inverse). More recently Gould and Scott [130] examined techniques based on equation (11.164) and showed that the apparently excessive linear algebra costs can be sidestepped.

The MR method of Chow and Saad [69] is based on determining an approximate *i*th column of \mathbf{K} by partially solving $\mathbf{Ax} = \mathbf{e}_i$ using a dropping version of Richardson's method (see [209] and [35], page 101). An earlier version [66] used GMRes as the iterative solver. The approximate solution thus obtained is then chosen to be the approximate *i*th column of \mathbf{K} . Given an initial approximate solution \mathbf{x}_1 , Richardson's method computes (with the usual notation)

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \mathbf{r}_j \left(\frac{\mathbf{r}_j^T \mathbf{A}^T \mathbf{r}_j}{\mathbf{r}_j^T \mathbf{A}^T \mathbf{A} \mathbf{r}_j} \right)$$

from which we deduce that

$$\mathbf{r}_{j+1} = \left(\mathbf{I} - \frac{\mathbf{A} \mathbf{r}_j \mathbf{r}_j^T \mathbf{A}^T}{\mathbf{r}_j^T \mathbf{A}^T \mathbf{A} \mathbf{r}_j} \right) \mathbf{r}_j.$$

Since the matrix transforming \mathbf{r}_j to \mathbf{r}_{j+1} is an orthogonal projector we have immediately $\|\mathbf{r}_{j+1}\| \leq \|\mathbf{r}_j\|$ with strict inequality if \mathbf{A} is positive real (see p. 31) since in this case $\mathbf{r}_j^T \mathbf{A}^T \mathbf{r}_j > 0$. The vectors \mathbf{x}_j computed by the Richardson iteration are then subjected to dropping with fill-in limitation. For further details see [69].

A variation of this algorithm is the “self-preconditioned” version, also due to Chow and Saad [69], and referred to in [31] as MRP. Since MR seeks to obtain an approximation \mathbf{M} to \mathbf{A}^{-1} using an iterative algorithm, Chow and Saad proposed employing the most recent approximation to \mathbf{M} in this iteration as well. However the need to provide a large number of user-defined parameters makes this version difficult to use and the computed matrix \mathbf{M} is not necessarily either positive definite or even symmetric, restricting its area of application. Moreover the performance of the algorithm is not exceptional and its preconditioner is more costly to compute than that of MR [31].

Other AIM methods have been proposed to compute approximate sparse inverses of \mathbf{A} . If \mathbf{K}_0 denotes a particular sparse inverse of \mathbf{A} and we define $\mathbf{K}(\alpha)$ by $\mathbf{K}(\alpha) \equiv \mathbf{K}_0 + \alpha \mathbf{S}$ for some matrix \mathbf{S} then α may be chosen to minimise $\|\mathbf{I} - \mathbf{A} \mathbf{K}(\alpha)\|_F^2$, and since each column of $\mathbf{I} - \mathbf{A} \mathbf{K}(\alpha)$ may be minimised independently (c.f. SpAI) some straightforward algebra yields

$$\alpha_{\min} = \frac{\sum_{j=1}^n \mathbf{s}_j^T \mathbf{A}^T (\mathbf{e}_j - \mathbf{A} \mathbf{k}_j)}{\|\mathbf{A} \mathbf{S}\|_F^2}$$

where \mathbf{e}_j , \mathbf{s}_j and \mathbf{k}_j denote the *j*th columns of \mathbf{I} , \mathbf{S} and \mathbf{K}_0 . The two choices of \mathbf{S} suggested in [69] are (essentially) \mathbf{R} and $\mathbf{A}^T \mathbf{R}$.

11.8.2. The AIF methods (FSAI, AIB and AInv)

The algorithm FSAI [172], [173] sets out to compute an approximate sparse inverse $\widehat{\mathbf{L}}$ of the Cholesky factor \mathbf{L} of \mathbf{A} , which for this method is assumed to be symmetric positive definite, by minimising $\|\mathbf{I} - \widehat{\mathbf{L}}\mathbf{L}\|_F$ for some given sparsity pattern of $\widehat{\mathbf{L}}$. Often this is chosen to be the same as that of \mathbf{A} . Since $\|\mathbf{I} - \widehat{\mathbf{L}}\mathbf{L}\|_F = \|\mathbf{I} - \mathbf{L}^T \widehat{\mathbf{L}}^T\|_F$, all that is necessary is to substitute \mathbf{L}^T for \mathbf{A} in equation (11.153) to obtain the approximate *i*th row of $\widehat{\mathbf{L}}$, and since $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ this gives

$$\mathbf{S}_i^T \mathbf{A} \mathbf{S}_i \mathbf{x} = \mathbf{S}_i^T \mathbf{L} \mathbf{e}_i.$$

If \mathbf{x}_i denotes the solution of this equation then the required sparse *i*th row of $\widehat{\mathbf{L}}$ is simply $\mathbf{x}_i^T \mathbf{S}_i^T$. Now the columns of \mathbf{S}_i correspond to the permitted non-zero elements of $\mathbf{e}_i^T \widehat{\mathbf{L}}$. Since $\widehat{\mathbf{L}} = [\widehat{l}_{ij}]$ is, by definition, lower triangular this means that these elements \widehat{l}_{ij} are nonzero only if $j \leq i$. The columns of \mathbf{S}_i can therefore be chosen only from the first *i* columns of the unit matrix and since \mathbf{L} is also lower triangular, it follows that the vector $\mathbf{S}_i^T \mathbf{L} \mathbf{e}_i$ will have only one nonzero element l_{ii} where $\mathbf{L} = [l_{ij}]$ (we assume that \mathbf{e}_i is included among the columns of \mathbf{S}_i). It is thus necessary to know only the diagonal elements of the Cholesky factor \mathbf{L} in order to obtain a sparse approximation of its inverse using the above technique.

If these are not available, \mathbf{x}_i may be computed by solving

$$\mathbf{S}_i^T \mathbf{A} \mathbf{S}_i \mathbf{x}_i = \mathbf{S}_i^T \mathbf{e}_i$$

(see equation (11.152)) and defining δ_i by $\delta_i = \mathbf{x}_i^T \mathbf{S}_i^T \mathbf{A} \mathbf{S}_i \mathbf{x}_i$. The *i*-th row of $\widehat{\mathbf{L}}$, $\mathbf{e}_i^T \widehat{\mathbf{L}}$, is then given approximately by

$$\mathbf{e}_i^T \widehat{\mathbf{L}} = \delta_i^{-\frac{1}{2}} \mathbf{x}_i^T \mathbf{S}_i^T.$$

11.8.2.1. AIB

Methods AIB and AInv are both related to the LU decomposition of \mathbf{A} where $\mathbf{A} \in \mathbb{R}^{n \times n}$ and is nonsingular. Suppose that such a decomposition exists so that $\mathbf{A} = \mathbf{LDU}$, where both \mathbf{L} and \mathbf{U} are unit triangular and \mathbf{D} is diagonal. If $\mathbf{Z} = \mathbf{U}^{-1}$ and $\mathbf{W}^T = \mathbf{L}^{-1}$,

$$\mathbf{W}^T \mathbf{A} \mathbf{Z} = \mathbf{D} \tag{11.165}$$

and the columns of \mathbf{W} and \mathbf{Z} are biconjugate with respect to \mathbf{A} . Equation (11.165) gives

$$\mathbf{A}^{-1} = \mathbf{Z} \mathbf{D}^{-1} \mathbf{W}^T \tag{11.166}$$

so if \mathbf{D} , \mathbf{W} and \mathbf{Z} can be determined we can compute \mathbf{A}^{-1} . If accurate sparse approximations to \mathbf{W} and \mathbf{Z} can be found we can obtain a sparse approximation

to \mathbf{A}^{-1} which may be used as a preconditioner. We now describe two ways of doing this, starting with the AIB method of Chow and Saad.

This method is the more straightforward of the two and is based on the identity

$$\begin{bmatrix} \mathbf{W}_k^T & \mathbf{0} \\ \mathbf{w}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A}_k & \mathbf{b} \\ \mathbf{c}^T & \alpha \end{bmatrix} \begin{bmatrix} \mathbf{Z}_k & \mathbf{z} \\ \mathbf{0} & 1 \end{bmatrix} \equiv \\ \begin{bmatrix} \mathbf{W}_k^T \mathbf{A}_k \mathbf{Z}_k & \mathbf{W}_k^T (\mathbf{A}_k \mathbf{z} + \mathbf{b}) \\ (\mathbf{w}^T \mathbf{A}_k + \mathbf{c}^T) \mathbf{Z}_k & \mathbf{w}^T \mathbf{A}_k \mathbf{z} + \mathbf{w}^T \mathbf{b} + \mathbf{c}^T \mathbf{z} + \alpha \end{bmatrix} \quad (11.167)$$

where \mathbf{W}_k , etc., denotes the leading principal submatrix of order k of \mathbf{W} , etc. We show that if

$$\mathbf{W}_k^T \mathbf{A}_k \mathbf{Z}_k = \mathbf{D}_k \quad (11.168)$$

for some diagonal matrix \mathbf{D}_k then \mathbf{w} , \mathbf{z} and δ may be determined such that equation (11.168) is satisfied with k replaced by $k+1$.

Equating the right-hand side of equation (11.167) to \mathbf{D}_{k+1} , where

$$\mathbf{D}_{k+1} = \begin{bmatrix} \mathbf{D}_k & \mathbf{0} \\ \mathbf{0}^T & \delta \end{bmatrix}.$$

gives equation (11.168) which is true by hypothesis and, assuming \mathbf{W}_k and \mathbf{Z}_k to be nonsingular, $\mathbf{A}_k \mathbf{z} = -\mathbf{b}$ and $\mathbf{w}^T \mathbf{A}_k = -\mathbf{c}^T$. These equations may be easily solved for \mathbf{z} and \mathbf{w} since, from equation (11.168), $\mathbf{A}_k^{-1} = \mathbf{Z}_k \mathbf{D}_k^{-1} \mathbf{W}_k^T$ so that

$$\mathbf{z} = -\mathbf{Z}_k \mathbf{D}_k^{-1} \mathbf{W}_k^T \mathbf{b} \quad (11.169)$$

with a similar equation for \mathbf{w} . The scalar δ is then given by

$$\delta = \mathbf{w}^T \mathbf{A}_k \mathbf{z} + \mathbf{w}^T \mathbf{b} + \mathbf{c}^T \mathbf{z} + \alpha. \quad (11.170)$$

From equation (11.167) we see that if \mathbf{Z}_k is unit upper triangular then so is \mathbf{Z}_{k+1} . The basic algorithm is implemented by putting $\mathbf{W}_1 = \mathbf{Z}_1 = 1$ and $\mathbf{D}_1 = a_{11}$, then computing \mathbf{W}_{k+1} , \mathbf{Z}_{k+1} and \mathbf{D}_{k+1} for $k = 1, 2, \dots, n-1$, from equations (11.169) and (11.170). Simple induction then shows that both $\mathbf{W} = \mathbf{W}_n$ and $\mathbf{Z} = \mathbf{Z}_n$ are unit upper triangular.

11.8.2.2. AInv and SAInv

The method AInv of Benzi and Tuma is based on a “Biconjugation Algorithm” which consists in fact of two separate, distinct and independent algorithms rolled into one. Let \mathbf{e}_i , $i = 1, 2, \dots, n$, denote the i th column of the unit matrix and \mathbf{a}_i^T the i th row of \mathbf{A} . One of the two components of the biconjugation algorithm, that which computes $\mathbf{Z} = [\mathbf{z}_1 \ \mathbf{z}_2 \ \dots \ \mathbf{z}_n]$, may be written as the following formal algorithm:

Primal conjugation algorithm

- (1) Put $\mathbf{Q}_0 = \mathbf{I}$

- (2) for $i = 1 : n$ do
- (3) $\mathbf{z}_i = \mathbf{Q}_{i-1} \mathbf{e}_i$
- (4) $\mathbf{Q}_i = \left(\mathbf{I} - \mathbf{z}_i (\mathbf{a}_i^T \mathbf{z}_i)^{-1} \mathbf{a}_i^T \right) \mathbf{Q}_{i-1}$
- (5) end loop
- (6) end Primal conjugation algorithm.

Trivially, from Step 4, $\mathbf{a}_i^T \mathbf{Q}_i = \mathbf{0}^T$ so that by implication $\mathbf{a}_{i-1}^T \mathbf{Q}_{i-1} = \mathbf{0}^T$ and, from Step 3, $\mathbf{a}_{i-1}^T \mathbf{z}_i = 0$. A simple induction argument then extends this to $\mathbf{a}_j^T \mathbf{z}_i = 0$ for $j < i$ so that, in the absence of breakdown,

$$\mathbf{A}\mathbf{Z} = \mathbf{L} \quad (11.171)$$

for some lower triangular matrix \mathbf{L} . If, in Step 3, \mathbf{Q}_{i-1} is expressed as the product of the factors $\left(\mathbf{I} - \mathbf{z}_j (\mathbf{a}_j^T \mathbf{z}_j)^{-1} \mathbf{a}_j^T \right)$ it is straightforward to show that

$$\mathbf{z}_i = \mathbf{e}_i - \sum_{j=1}^{i-1} \mathbf{z}_j u_{ji}, \quad i = 1, 2, \dots, n,$$

for some scalars u_{ji} . Written as a matrix equation this becomes $\mathbf{I} = \mathbf{ZU}$ where $\mathbf{U} = [u_{ji}]$ is some unit upper triangular matrix. Thus $\mathbf{Z} = \mathbf{U}^{-1}$ and is also unit upper triangular so that we have, from equation (11.171), performed LU-decomposition using a sequence of projections.

The dual conjugation algorithm is the same as the primal except that \mathbf{A}^T and \mathbf{W} replace \mathbf{A} and \mathbf{Z} . By analogy with equation (11.171)

$$\mathbf{A}^T \mathbf{W} = \tilde{\mathbf{L}} \quad (11.172)$$

for some lower triangular matrix $\tilde{\mathbf{L}}$ and $\mathbf{W} = \tilde{\mathbf{U}}^{-1}$ for some unit upper triangular matrix $\tilde{\mathbf{U}}$. Hence \mathbf{W} is also unit upper triangular. Since $\mathbf{Z} = \mathbf{U}^{-1}$ equations (11.171) and (11.172) yield

$$\mathbf{W}^T \mathbf{A} \mathbf{Z} = \tilde{\mathbf{U}}^{-T} \mathbf{L} = \tilde{\mathbf{L}}^T \mathbf{U}^{-1} = \mathbf{D} \text{ (say)} \quad (11.173)$$

and it follows from the triangularities of \mathbf{L} , \mathbf{U} , $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$ that \mathbf{D} is simultaneously both lower and upper triangular and hence diagonal. Since both \mathbf{U} and $\tilde{\mathbf{U}}$ are unit triangular, equation (11.173) also implies that the diagonal elements of $\tilde{\mathbf{L}}$ are equal to those of both \mathbf{L} and \mathbf{D} . Moreover, since both \mathbf{W} and \mathbf{Z} are unit upper triangular, the factorisation achieved by the algorithm is the same as that obtained by AIB. Equation (11.173) is commonly written

$$\mathbf{A}^{-1} = \mathbf{ZD}^{-1} \mathbf{W}^T. \quad (11.174)$$

The primal conjugation algorithm above is essentially the basic ABS algorithm, implicit LU version [1].

The above descriptions of AIB and AI_{Inv} give only the algebraic foundations of the methods. In practice the vectors \mathbf{w} and \mathbf{z} computed by both methods are subject to dropping and, in the case of AIB, to restriction of the number of non-zero elements

(this is apparently not necessary for AInv). The differences in the performance of the two methods may be attributed to differences in the dropping procedures.

If \mathbf{A} is symmetric positive definite a “stabilised” form of AInv (SAInv) can be obtained by replacing Step 4 of the primal conjugation algorithm by

$$4. \quad \mathbf{Q}_i = \left(\mathbf{I} - \mathbf{z}_i (\mathbf{z}_i^T \mathbf{A} \mathbf{z}_i)^{-1} \mathbf{z}_i^T \mathbf{A} \right) \mathbf{Q}_{i-1}.$$

This algorithm is self-dual and the same argument used to derive equation (11.174) gives

$$\mathbf{A}^{-1} = \mathbf{Z} \mathbf{D}^{-1} \mathbf{Z}^T.$$

Despite Step 4 being different, both the primal conjugation algorithm and its self-dual equivalent generate (in the absence of dropping and using exact arithmetic) identical sequences of projection matrices $\{\mathbf{Q}_i\}$ and hence identical sequences $\{\mathbf{z}_i\}$. Moreover $\mathbf{z}_i^T \mathbf{A} \mathbf{z}_i = \mathbf{a}_i^T \mathbf{z}_i$ so that if $\mathbf{z}_i = [\zeta_j]$ and if, for some j satisfying $1 \leq j \leq i-1$, $|\zeta_j| \gg 1$, the primal conjugation algorithm might prove to be the more stable of the two. When dropping is taken into account, though, things change. The divisor $\mathbf{z}_i^T \mathbf{A} \mathbf{z}_i$ can never be zero for positive definite \mathbf{A} if $\mathbf{z}_i \neq \mathbf{0}$ even though $\mathbf{a}_i^T \mathbf{z}_i$ could well be. Tests indicate that SAInv is a reliable algorithm, and the method has been used to solve some highly ill-conditioned sets of symmetric linear equations [23].

Block versions of both AInv and SAInv have been proposed and may be obtained by simply interpreting \mathbf{a}_i and \mathbf{z}_i as groups of columns of \mathbf{A}^T and \mathbf{Z} respectively instead of as single columns as for the basic algorithms. All that is required is that both \mathbf{a}_i and \mathbf{z}_i represent the same number of columns (and not necessarily the same as \mathbf{a}_j and \mathbf{z}_j , $j \neq i$) in order that $\mathbf{a}_i^T \mathbf{z}_i$ should be square. Judging by the published results, the AInv type of factorisation (both scalar and block) seems to be particularly suited to the construction of sparse factorised inverses.

AIB was proposed by Chow and Saad [66] while AInv was introduced as a method for preconditioning symmetric positive definite systems by Benzi [22] and Benzi *et al* [27]. It was subsequently extended to non-symmetric matrices by Benzi and Tůma [30]. The stabilised version, SAInv, was described in [24] and [32] with further reports on its performance in practice in [26] and [171].

11.8.3. The PP methods (JP and the TN methods)

The JP (Jacobi polynomial) method as implemented in [31] is based on the least-squares minimisation of a residual polynomial $p_r(\lambda) \in Q_r$ given by expression (11.82) with the weight function being given by equation (11.83). In the implementation of Benzi and Tůma the eigenvalues λ_1 and λ_n were estimated using Gershgorin discs and the product of a matrix polynomial and a vector was carried out using Horner’s scheme. This may be derived straightforwardly from equation (11.69).

The preconditioners for the truncated Neumann (TN) versions of the symmetric successive over-relaxation and symmetric Gauss-Seidel algorithms are based on equation (11.113) where \mathbf{M} is regarded as an approximation to \mathbf{A} . Gustafsson and

Lindskog [139] modify this slightly and, for a symmetric system ($\mathbf{U} = \mathbf{L}^T$), essentially define \mathbf{M} by

$$\mathbf{M} = \frac{(\mathbf{I} - \omega\mathbf{L})(\mathbf{I} - \omega\mathbf{L}^T)}{\omega} \quad (11.175)$$

omitting the factor $(2 - \omega)$. Note that in the present discussion the matrices \mathbf{L} and \mathbf{U} are not the triangular factors of \mathbf{A} but are defined by equation (11.86). From equation (11.175)

$$\mathbf{K} = \mathbf{M}^{-1} = \omega (\mathbf{I} - \omega\mathbf{L}^T)^{-1} (\mathbf{I} - \omega\mathbf{L})^{-1}$$

and could be used as it stands as a preconditioner as the triangular matrices are simple to invert. The authors, however, replace the inverse matrices by truncated power series to obtain a completely parallelizable algorithm. Since $\mathbf{L}^n = \mathbf{O}$, $(\mathbf{I} - \omega\mathbf{L})^{-1} \equiv \sum_{j=0}^{n-1} (\omega\mathbf{L})^j$ and this is then approximated by $\sum_{j=0}^{p-1} (\omega\mathbf{L})^j$ where, in the tests reported in [139], p varied between two and six.

It must be remembered that this algorithm involves two approximations. First \mathbf{M} is only an approximation to \mathbf{A} and second, the truncated series only approximates $(\mathbf{I} - \omega\mathbf{L})^{-1}$. For most symmetric positive definite matrices \mathbf{A} the optimal value of ω (that giving the most rapid asymptotic convergence of SSOR) lies in $(1, 2)$ but for $\omega > 1$ the truncated power series might not be such a good approximation to $(\mathbf{I} - \omega\mathbf{L})^{-1}$. There is also the question of the determination of ω , especially for problems not of simple Dirichlet type. It may be better to set $\omega = 1$ (the TNSGS variant) as was done by Benzi and Tuma in the tests reported in [31].

11.8.4. The AIAF methods

These methods consists of various combinations of IC(p), ICT, ILU(p) and ILUT which are used to compute first the approximate sparse factors $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$ of \mathbf{A} and secondly the approximate sparse inverses $\hat{\mathbf{L}}$ and $\hat{\mathbf{U}}$ of $\bar{\mathbf{L}}$ and $\bar{\mathbf{U}}$. See e.g. [4], [83], [231], [232] and [244]. In view of the assertions in [31] that these methods are not competitive either for symmetric positive definite or for more general problems, we refer interested readers either to [31] or to the individual references cited above.

11.8.5. Other methods

In this book we have concentrated on those aspects of Krylov theory that are independent both of any special features of the problems requiring solution (apart from size and sparsity), and also of the nature of the machinery available for performing the calculations. However the increasing size of the problems together with the increasing availability of parallel computers has given added impetus to the development of methods devised to exploit any idiosyncrasy of both problem and hardware. We have already outlined methods like SpAI and AInv that are specifically intended for use with parallel computers and in this section we mention briefly

some other methods whose development was stimulated by their derivation from partial differential equations.

We have stated on various occasions that essentially sequential methods like ILU are inherently unsuitable for parallel computers and this is generally true. However for certain well-structured problems it is possible to re-order the variables to permit more effective use of parallel computers. A simple example of this is the “red-black” ordering derived from the paradigm of solving $\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = f(x, y)$ over a plane rectangular region (the so-called *model problem*). This is done by superimposing a rectangular grid on the region and replacing the Laplacian operator by a 5-point finite difference approximation. This approximation is applied at each mesh intersection (*mesh-point*, *grid-point* or *node*) and consists of a simple linear equation linking the approximate value of φ at the mesh-point to those of its four nearest neighbours, i.e. the points “north”, “south”, “east” and “west” of the central point. Thus each mesh-point has its own approximate value of φ and its own linking equation. Suppose now that the mesh-points are ordered from 1 to n , where n is the total number of mesh-points and the actual ordering is arbitrary. If now φ_i denotes the approximate value of φ at the point i and the corresponding linking equation is assigned to be the i -th equation of the set, then depending on how the mesh-points are ordered the matrix of coefficients \mathbf{A} assumes a particular characteristic form. In particular, since the coefficient of φ_i is (for the model problem) always non-zero, this choice of assignment guarantees that the diagonal coefficients of \mathbf{A} are all non-zero regardless of the ordering of the points. If, for example, the points in each mesh-row are numbered sequentially from the left and the rows are taken in order starting from the top we get “row ordering” which results in \mathbf{A} being tridiagonal but with an extra diagonal on each side of, and separated some distance from, the main tridiagonal core (c.f. equation (11.120)). Another ordering is based on the observation that the mesh-points of the model problem can be divided into two sets (“red” and “black”) such that if the central point of each 5-point approximation is in one set then the four peripheral points are in the other. If the points are first ordered into sets and each set is then “row-ordered”, the resulting matrix \mathbf{A} has the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{D}_2 \end{bmatrix}$$

where \mathbf{D}_1 and \mathbf{D}_2 are diagonal. Red-black ordering is one of the more favourable orderings for parallel and vector computing, and can be generalized to more colours (especially for three-dimensional problems). Its principal disadvantage is that if ILU or IC preconditioning is then applied (and the effective implementation of this is the whole point of the ordering) the convergence of the subsequent Krylov iteration can be impaired (see [97], and [207] for the multi-colouring analogue). It is possible though, for certain problems, to find orderings that are both parallel-computer friendly and for which ILU or IC give similar convergence rates to the ones that they give when applied to the un-ordered system. The earliest of these were the *van der Vorst orderings* [240] and were applied to simple 2-dimensional problems on regular grids similar to the model problem defined above. These methods have been

developed by Hysom, Pothen, Magolu monga Made and van der Vorst [157], [158], [181] who have shown that it is possible to exploit parallelism while at the same time preserving the original Krylov rates of convergence. This is done by restructuring (see footnote, page 211) the coefficient matrix \mathbf{A} into a form suitable for parallel ILU decomposition and is accomplished by using graph-theoretic techniques. A graph is constructed with n nodes and with an *arc* (or *edge*) between nodes i and j if and only if $a_{ij} \neq 0$. It is then analysed using the algorithm described in [169] or similar, and the equations are re-ordered to give a new matrix $\mathbf{A}_1 = \mathbf{P}^T \mathbf{A} \mathbf{P}$ where \mathbf{P} is the permutation matrix effectively determined by the graph-analysis algorithm. For more information on the relationships between matrices and graphs, see e.g. [246].

According to Benzi [23] these techniques have (up to 2002) been applied only to fairly simple and regular problems and their performance on more complicated (and more realistic) systems is yet to be determined. Despite this though they are regarded as one of the more promising recent developments in the field of preconditioning. Other orderings designed specifically for use with IC or ILU, or otherwise stimulated by the need to improve parallel implementation, include those described in [58], [62], [73], [84], [85], [90], [91], [98], [100], [105], [177], [192], [200], [207] and [229].

For problems involving finite difference approximations derived from partial differential equations the algebraic equations to be solved have regularities that may be exploited over and above those used to obtain good parallel orderings. The coarseness or otherwise of the superimposed grid clearly affects the accuracy of the finite-difference approximation. A coarser grid results in greater truncation error but fewer grid-points, hence fewer linear equations and generally more rapid convergence of a Krylov solver. Moreover there are relationships between the coarser and finer solutions that may be exploited either to develop new methods of solution or new preconditioners. These *multigrid methods* and *multilevel methods* have been described by Hackbusch [145], [147] and more recently by Bramble [34], Briggs *et al* [41] and Trottenberg *et al* [235].

The early multigrid methods were specific to the partial differential equations from which they were derived and to the structures being modelled. They could in no way be regarded as general-purpose algorithms. These early methods are referred to in [23] as *geometric multigrid methods*. However, as the subject developed, such methods became more general and more able to solve wider classes of problems, with the user becoming less involved in the implementational details. Milestones along this particular route were the *black-box multigrid* method of Dendy [88] and the *algebraic multigrid* (AMG) method of Ruge and Stüben [211]. Another facet of this trend towards generality was the realisation that some of the techniques deployed by one class of methods (multigrid) could be used with advantage by the other (Krylov), and *vice versa*. For further details, see [23] and the references cited therein.

The synthesis between the multigrid (and multilevel) methods and the Krylov methods goes further than the mere exchange of techniques. Recently methods like AMG and ILU have both been interpreted as approximate Schur complement methods [16], [17], [82], [247] and these interpretations may perhaps be the first

steps in the creation of a more comprehensive theory of ILU preconditioning. See also Axelsson [11] for a general description of Schur complement methods. The motivation behind the attempts to merge these two theories is the construction of algorithms that combine the stability of the multilevel methods with the speed of convergence of the preconditioned Krylov methods. See [23] for further references on this theme. However the ever-increasing size of problems requiring solution is such that it is perhaps not possible to achieve the ideal of a truly algebraic preconditioner, i.e. one that is derived wholly from the original matrix of coefficients and which makes no concessions to the physical origins of the problem. For really big problems, any help that may be forthcoming from any other aspect of the problem is often needed to obtain a solution.

Other methods derived from partial differential equations include those based on domain decomposition. These methods, like the multigrid methods, are derived from the geometry of the original problem and feature the decomposition of the original domain into a set of subdomains which may be dealt with in parallel. If these are solved approximately using ILU then we have a “parallel ILU” strategy [18].

Finally we consider algorithms based on *blocking*, i.e. partitioning. These techniques were originally applied to dense matrices and the block sizes were determined by the architecture of the computers used to solve the problem. It is much more difficult to apply them to sparse problems, more particularly when the latter are unstructured. Block methods applied to sparse systems work best when the original coefficient matrix can be partitioned in such a way that a partition is either dense or null. The matrix is then a sparse matrix but with (hopefully dense) blocks replacing the scalar elements. The block algorithm is then a simple block analogue of the scalar one.

If the original matrix does not partition readily into blocks it is either because it is genuinely unstructured (as might be the case if it were derived from a problem involving a power cable network whose sparseness depends on random geographical features) or because the original equations are badly ordered. This could occur if a matrix were derived from a finite element problem where the variables appropriate to a particular element were not ordered successively. In the second case an appropriate re-ordering would result in a matrix much closer to the ideal. To effect this re-ordering, graph-based algorithms may again be employed. The graph compression algorithm of Ashcraft [8], [218] has been used to advantage for certain types of problem [26] as has reverse Cuthill and McKee. Other schemes include the parameterized block ordering algorithm (PABLO) (see [63], [197], and other references cited in [23]). Software for these algorithms is available and described in [65].

11.8.5.1. General survey

Before discussing the relative merits of the preconditioners described above, a few general comments are in order. The utility of a preconditioner derives as much from its ease of use as its effectiveness in reducing computing time. This in turn is

influenced by the number of parameters required of the user. Thus ILU(0) requires no extra information, ILU(p) only the value of p but ILUT requires one (or perhaps two) drop tolerances together with the maximum number of nonzero elements per row. While an increased number of parameters often leads to a greater “tuning” capability and perhaps a very good performance, this may be difficult to achieve in practice. The AIAF algorithms in particular seem to suffer from this problem as choices have to be made not only in computing the final approximate inverses but also in computing the original sparse factors. To make matters worse, the best values for obtaining the second approximations are often far removed from those for computing the first ones [31]. On the other hand, ICT and ILUT algorithms require two or three arbitrary parameters but despite this are not too difficult to construct. A good rule-of-thumb for these methods seems to be to choose the parameters so that the number of non-zeroes in the sparse approximate factors roughly equals that in the original matrix.

Two more obvious attributes that we might expect are that expensive-to-calculate preconditioners should be both robust (i.e. should fail on comparatively few problems) and efficient (i.e. should result in rapidly-converging Krylov iterations). To a certain extent these expectations were met by the algorithms in Benzi and Tůma’s tests, but not perhaps to the extent that one would wish. If we consider all thirty test problems, AInv failed on three and AIB on four, while FSAI failed on one of the ten symmetric positive definite problems to which it was restricted. SpAI (restricted to the twenty general problems) also failed on three. Thus all four of these preconditioners had a failure rate of about 10% overall, satisfactory but hardly spectacular. However it must be borne in mind that none of these algorithms as tested had benefited from the re-ordering strategy of Olschowka and Neumaier. For general problems TNSGS failed on three occasions out of twenty (it also failed three times out of ten for symmetric positive definite problems, probably ruling it out as a serious contender in that particular arena despite it being quite efficient when it did work). These methods, though, were comprehensively out-performed by ICT, ILU(0) and ILUT which, in their respective domains, notched up not a single failure. Even DS, hardly the most glamorous of preconditioners, failed on only three out of thirty examples while two of the AIAF algorithms, one in each category, managed two failures each. Note that of the two successful AIAF methods, that for symmetric positive definite problems computed $\bar{\mathbf{L}}$ using ICT and imposed the same sparsity structure upon $\hat{\mathbf{L}}$ while the other used an ILUT/ILUT approach. Other methods tested were either less robust than the above or were significantly more expensive to compute.

Thus if all we are concerned with is robustness (and it must be the most important consideration) the clear choice is ICT for symmetric positive definite problems and either ILU(0) or ILUT for the more general ones. The catch here though is that ICT and ILU are inherently sequential though this drawback may be mitigated to some extent by the use of van der Vorst and other orderings. The other methods described in this section were specifically designed to avoid the shortcomings associated with back substitution. In this they have succeeded but with a 10% failure rate it would appear that, in the quest for a robust preconditioner that is also well-adapted to

vector and parallel computers, there is still some way to go.

Since the publication of [31], on which much of the above was based, other methods have come to the fore. One of the more important of these is that of Olschowka and Neumaier (see page 262 above). This method combines scaling and permutation and can be applied to any linear system. The resulting equations can then be solved using any of the other preconditioned algorithms described above. The method works particularly well when paired with ILU but has also improved the properties of SpAI [23].

When it comes to assessing the relative efficiencies of the various preconditioners, matters are somewhat more complicated. Features like the nature of \mathbf{A} (symmetric positive definite or general), the number of right-hand sides and the type of computer enter the discussion making comparative assessment more difficult. In order to simplify matters we first consider problems with large numbers of right-hand sides, which in most cases permits us to ignore the initial costs of setting up the preconditioner and allows us to concentrate on the rate of convergence of the resulting iteration. The cases where it does not give us this latitude are those involving the AIAF methods. Despite some of these methods producing good (but not outstanding) iterative performances their costs, together with the implementational difficulties of setting them up, are so great that the conclusion in [31] was that for both symmetric positive definite and also for more general problems they were just not competitive. We shall, therefore, ignore them in the subsequent discussion.

The algorithms requiring the fewest iterations are, once again, ICT and ILU, closely followed by (in order) AInv, SpAI and AIB. The TN methods needed on average more iterations than AIB but fewer than JP and MR. Now for most of these methods it may be assumed that a smaller number of iterations implies faster convergence but this depends very much upon the machine architecture. On serial machines, ILU is almost always faster than AIF or AIM but this is often not the case for parallel or vector computers. However this may change if orderings more suited to parallel computers are used.

Thus for problems with several right-hand sides which are not likely to suffer breakdown, the AIF methods are among the most satisfactory as are the more recent variants of ILU. AInv is probably the best of the AIF methods for general problems and, on account of the marginally superior robustness reported in [31], AIB for symmetric positive definite ones. The older IC and ILU methods are slower but even more robust while SpAI is also a competitor. Other methods are either less robust or slower, and would probably not be regarded as “first-choice” methods. Their importance lies in providing alternatives for problems whose idiosyncrasies make the first-choice methods unsuitable. Examples of this were given in [64] and [216] where MR was reported as solving problems that were too difficult for the ILU methods.

For solving problems with few right-hand sides, the costs of setting-up the preconditioners weigh heavily against the AIF and AIM methods. In the search for efficiency we are left only with DS and the PP methods to compete against IC and ILU. According to [31], these fail to do when solving symmetric positive definite problems, leaving ICT as the top method but with DS and TNSGS as pos-

sible alternatives despite being less robust. For more general problems the fastest methods are TNILUT and TNSGS with TNILU(0) also recording a strong performance. It was further noted that the TN versions of ILU(0) and ILUT were, on the whole, faster than plain ILU(0) and ILUT on vector processors, indicating that the gain in speed due to vectorisation more than outweighed the degradation of the preconditioner caused by replacing it by a TN approximation.

Finally, if we look at the most robust methods there is one notable omission, IC(0). This algorithm was designed for problems for which the assumptions underlying the level-of-fill method might be expected to be valid but it failed in three of the ten symmetric positive definite examples. This was probably due to stability problems and the consequent need to modify the diagonal elements of $\bar{\mathbf{L}}$ (see Munksgaard's algorithm, page 246). These failures contrasted vividly with the success of ILU(0) which intuition would suggest should be less robust. Clearly the presence of sparsity requires a re-appraisal of some of the old certainties.

11.9. In conclusion

With so many papers describing so many different preconditioners being published over the last few years it has not been possible to do justice to them all in the space of a single chapter. We have therefore tried to select those that are most effective in practice as well as others that are of historical importance or which, although superseded, form the basis of the current methods-of-choice. We have included unsuccessful methods if based on some original ideas (like the AIAF methods) but have excluded others (like the incomplete Gauss-Jordan methods [77, 256]) in which two well-known techniques are unsuccessfully coupled. Among other techniques not discussed here are those based on incomplete QR factorization [160]. Here the incomplete orthogonal decompositions of \mathbf{A} are obtained by modifying either the Gram-Schmidt or the Givens rotation method. These methods can also be applied to the CGNR method since if $\mathbf{A} \simeq \mathbf{Q}\mathbf{U}$ then $\mathbf{A}^T\mathbf{A} \simeq \mathbf{U}^T\mathbf{U}$, and a study of the different choices of preconditioners that have been applied to CGNR appears in [29]. In [214] the idea of computing an incomplete LQ factorisation of \mathbf{A} as a preconditioner for the CGNE method was introduced while the Compressed Incomplete Modified Gram-Schmidt preconditioner was discussed in [253]. Finally for the most recent and authoritative survey of preconditioners and preconditioning known to the authors, see Benzi [23].

As the tests reported in Chapter 10 (above) indicate, the Krylov methods applied to the raw equations as derived by the scientists, engineers or other practitioners are altogether too slow and unreliable for normal use. Their performances, though, can be improved dramatically by preconditioning. Preconditioning must therefore be regarded not merely as an optional extra, to be called upon at the whim of users as and when the fancy takes them but as an integral part of the finished algorithm. It can be achieved by the insertion of a single entity (\mathbf{K}) in the equations defining the unpreconditioned algorithm, scarcely affects the difficulty of establishing the algorithm's elementaty properties and can easily be included in any introductory

description of the basic CG method. Not only does it permit the more efficient solution of practical problems but it enables the theory to be presented in a more balanced and elegant fashion. Thus any discussion of the Krylov methods that omits preconditioning is essentially incomplete.

It will be clear from the discussion in this chapter that we do not yet have a “universal preconditioner” that is optimal in all computing environments. For serial machines, methods like ICT and ILU give the fastest and most robust preconditioners but the older versions of these techniques are not well suited to vector and parallel machines. Since some modern problems are enormous and demand parallel computation the older ILU methods are gradually being phased out. Unfortunately those methods that are so suited tend to be less robust than ICT or ILU. However the AIM and AIF methods are almost as reliable and, if they do converge, do so considerably more rapidly than their more robust competitors. These methods, though, do have a serious inherent weakness. Any method that computes sparse approximations of either \mathbf{A}^{-1} or its triangular factors is assuming implicitly that such sparse approximations exist. They often do, especially for highly-structured matrices [87], but are unlikely to work well with matrices whose inverses consist of elements all of which are of roughly the same size. For such matrices it seems likely that methods based on sparse approximate factors of \mathbf{A} , which is itself sparse, will be more effective but with the disadvantage of requiring serial techniques for their exploitation.

In their tests Benzi and Tůma used QMRCGS (TFQMR), BiCGStab and GMRes(20) as their Krylov solvers but they reported only the results obtained by BiCGStab. These were regarded as being marginally superior to those of the other two methods. The tests reported in Chapter 10 (above) confirm this, with BiCGStab being superior to QMRCGS on grounds of robustness and to GMRes(20) on grounds of efficiency, especially if the latter is measured in terms of BiCG equivalents (but not for arc130, ever the maverick!). The claim, though, that convergence rates depend more on the preconditioner than the Krylov method was probably only made because Benzi and Tůma never tested the Hegedüs methods or LSQR. As the tests in Chapter 10 (above) indicate, there is a marked difference between the performances of methods where \mathbf{KG} is block-diagonal (BiCG and the related Sonneveld algorithms) and those where it is skew block-diagonal (HG, BiCR and LSQR). However if the Hegedüs Galerkin method HG is regarded as an alternatively-preconditioned version of BiCG (and since the only difference between the two algorithms is the preconditioning matrix \mathbf{K} , why should it not be?) then their claim stands, though perhaps not quite in the way that they originally intended.

Finally we noted above that the introduction of the preconditioning matrix \mathbf{K} permits the theory to be developed in a more satisfying way as it gives more “balance” to some of the equations. We have already seen this since if $\mathbf{K} = \mathbf{K}^T$, equation (3.8) gives $\mathbf{F}_j^T \mathbf{K} \mathbf{F}_k = \mathbf{O}$, $j \neq k$, which precisely mirrors equation (1.40). In our final chapter we enlarge on this and extend the idea of duality a little further.

Chapter 12

Duality

‘Beauty is truth, truth beauty’ - that is all
Ye know on earth, and all ye need to know.

John Keats, Ode on a Grecian Urn

In the previous chapters we have tried to present a simple and unified theory of conjugate gradient methods for solving general systems of linear equations. Following Broyden [47] the theory has been expressed in terms of two matrices \mathbf{G} and \mathbf{K} rather than the three matrices used by Manteuffel and his co-workers in their taxonomy and related studies [7], [106], or the three matrices employed by Tyrtishnikov and Voevodin [249], [250], [251]. The precise relationship between the matrices \mathbf{A} , \mathbf{B} and \mathbf{C} of Manteuffel *et al.*, the (different) matrices \mathbf{A} , \mathbf{B} and \mathbf{C} of Tyrtishnikov and Voevodin, and our own matrices \mathbf{G} and \mathbf{K} is given in the final section of Chapter 3 (above).

The decision to use these particular two matrices as our “primitives” (i.e. matrices in terms of which all others are expressed) was determined by their basic properties. A sufficient (but not necessary) condition for the recurrences of both the HS and Lanczos versions of a particular algorithm to be short is that both \mathbf{G} and \mathbf{K} are symmetric. A sufficient condition for the same algorithm to be breakdown-free is that they are both positive definite. Moreover, of the two types of breakdown, crashing is associated with the indefiniteness of \mathbf{G} and stagnation with that of \mathbf{K} .

A further unification of the theory became possible when it was realised [45] that methods like QMR and BiCG could be expressed as particular examples of the block-CG algorithm. This not only enabled these and similar methods to be included in the same overall theory with \mathbf{G} and \mathbf{K} playing the same rôles as before but it also caused a re-appraisal of the nature of \mathbf{G} . For the earlier, simple methods that generated vector sequences, \mathbf{G} was normally assumed to be positive definite in order to guarantee stability but for the compound algorithms (see page 48), \mathbf{G} is often by its very nature indefinite so that certain algorithms, e.g. BiCG, are inherently prone to breakdown. The indefiniteness of \mathbf{G} also has certain linguistic implications. We have referred to \mathbf{K} as the *preconditioning matrix* since preconditioning is what it does, and if its inverse is normally given that name we

hope that our departure from convention may be regarded as venial. The matrix \mathbf{G} , on the other hand, has sometimes been referred to as the *inner-product matrix*, a name that is quite inappropriate for an indefinite matrix. This matrix has also been referred to as the *Hessian* since it is the Hessian associated with the energy norm of the error, but this term is not in general use. Now it may be thought that this discussion of nomenclature is a little pedantic, but it is just possible that the use of the term *inner-product matrix* has actually acted as an impediment to the development of the theory by linguistically pre-empting the possibility of an indefinite \mathbf{G} . We therefore accept the possibility of an indefinite \mathbf{G} so that we have two matrices whose symmetry determines the number of terms in the recurrences for $\{\mathbf{P}_i\}$ and whose definiteness guarantees the absence of breakdown. We might note here that we have always (except for CGW which is a special case) assumed \mathbf{G} to be symmetric and have always had short recurrences for $\{\mathbf{F}_i\}$, so that under these conditions the brevity or otherwise of the sequence $\{\mathbf{P}_i\}$ depends only on the symmetry of \mathbf{K} . It is tempting to speculate on whether the symmetry of \mathbf{G} is sufficient for the brevity of the \mathbf{F}_i -recurrences while that of \mathbf{K} is sufficient for the brevity of the \mathbf{P}_i -recurrences, but even if that is not the case the possibility that \mathbf{G} and \mathbf{K} are somehow dual is beginning to emerge.

The idea that \mathbf{G} and \mathbf{K} are in some sense dual is strengthened by the property that if both \mathbf{G} and \mathbf{K} are symmetric then the matrices \mathbf{P}_i are mutually \mathbf{G} -conjugate while the matrices \mathbf{F}_i are mutually \mathbf{K} -conjugate. It would appear that if \mathbf{G} and \mathbf{K} are dual then \mathbf{F}_i and \mathbf{P}_i are also dual, and if this is so then they would be expected to have similar properties. To a certain extent they do. Equations (3.20) and (3.21), repeated here for convenience, are

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G}\mathbf{P}_i\mathbf{D}_i^{-1}\mathbf{C}_i \quad (12.1)$$

and

$$\mathbf{P}_{i+1} = \mathbf{K}\mathbf{F}_{i+1} + \mathbf{P}_i\mathbf{C}_i^{-1}\mathbf{C}_{i+1} \quad (12.2)$$

but although these equations have similarities they are not obviously "dual" and it is difficult to see how they could be made so. Other features too argue against duality. The matrices \mathbf{P}_i may be normalised by replacing equation (12.2) by equation (3.18), where the normalising matrices \mathbf{B}_j , $j = i, i+1$ are arbitrary apart from being nonsingular, and are generally chosen to improve the numerical properties of \mathbf{P}_j . The matrices \mathbf{F}_i , on the other hand, cannot be normalised as they are defined to be the residuals $\mathbf{G}\mathbf{X}_i - \mathbf{H}$ and are thus sacrosanct. Our search for duality therefore seems to have reached an *impasse*. The main argument, though, against duality is that there is no obvious dual system of equations that intuition suggests should be central to any such theory.

We now consider the possibility of constructing such a system. Should we wish to do so the clear choice for the matrix of coefficients has to be \mathbf{K} but there is no obvious candidate for the matrix of right-hand sides. We therefore suggest that an arbitrary matrix \mathbf{C} be chosen for these right-hand sides, where \mathbf{C} is quite distinct from any matrix \mathbf{C} or \mathbf{C}_i appearing elsewhere, so that the dual system of equations that we wish to solve becomes

$$\mathbf{K}\mathbf{Y} = \mathbf{C}. \quad (12.3)$$

We now look at the possibility of solving these equations by generating a sequence of matrices $\{\mathbf{P}_i\}$, mutually conjugate with respect to \mathbf{G} , by a scaled version of GODir and calculating from this sequence a further sequence of approximate solutions $\{\mathbf{Y}_i\}$ of equation (12.3), where

$$\mathbf{P}_i = \mathbf{K}\mathbf{Y}_i - \mathbf{C}. \quad (12.4)$$

Now the unscaled form of the algorithm GODir is defined on page 32. For the scaled form the generating matrices \mathbf{W}_i are given by equation (2.6) with $\mathbf{B} = \mathbf{KG}$, where \mathbf{G} is symmetric but \mathbf{K} is only assumed to be nonsingular, so that

$$\mathbf{W}_{i+1} = \mathbf{KGP}_i \mathbf{H}_{i+1,i}^{-1}. \quad (12.5)$$

Since

$$\mathbf{P}_{i+1} = \mathbf{Q}_i^T \mathbf{W}_{i+1} \quad (12.6)$$

where (equation (1.50))

$$\mathbf{Q}_i = \mathbf{I} - \sum_{j=1}^i \mathbf{GP}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \quad (12.7)$$

it follows from equations (12.5) - (12.7) that, since \mathbf{D}_j is symmetric for all j ,

$$\mathbf{P}_{i+1} \mathbf{H}_{i+1,i} = \mathbf{KGP}_i - \sum_{j=1}^i \mathbf{P}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{GKGP}_i. \quad (12.8)$$

Define now, for simplicity, the matrices \mathbf{H}_{ji} by

$$\mathbf{H}_{ji} = \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{GKGP}_i, \quad 1 \leq j \leq i. \quad (12.9)$$

Equation (12.8) then becomes

$$\mathbf{KGP}_i = \sum_{j=1}^{i+1} \mathbf{P}_j \mathbf{H}_{ji}$$

and if, in the right-hand side of this equation, we substitute $\mathbf{KY}_j - \mathbf{C}$ for \mathbf{P}_j and pre-multiply the whole by \mathbf{K}^{-1} we obtain

$$\mathbf{Y}_{i+1} \mathbf{H}_{i+1,i} = \mathbf{GP}_i - \sum_{j=1}^i \mathbf{Y}_j \mathbf{H}_{ji} + \mathbf{K}^{-1} \mathbf{C} \sum_{j=1}^{i+1} \mathbf{H}_{ji} \quad (12.10)$$

as the equation that we will use to compute \mathbf{Y}_{i+1} . Now if we examine this equation we see that everything is known except \mathbf{Y}_{i+1} so in principle it can be computed in terms of known quantities, but in order to do so we would have to know $\mathbf{K}^{-1} \mathbf{C}$ which is the very solution we are seeking. Once again we appear to have reached an

impasse. However since $\mathbf{H}_{i+1,i}$ is an arbitrary matrix it can be chosen not to impose a normalisation condition like $\mathbf{P}_{i+1}^T \mathbf{P}_{i+1} = \mathbf{I}$ but to ensure that $\sum_{j=1}^{i+1} \mathbf{H}_{ji} = \mathbf{O}$. Thus if we put

$$\mathbf{H}_{i+1,i} = -\sum_{j=1}^i \mathbf{H}_{ji},$$

equation (12.10) becomes

$$\mathbf{Y}_{i+1} = \left(\mathbf{G}\mathbf{P}_i - \sum_{j=1}^i \mathbf{Y}_j \mathbf{H}_{ji} \right) \mathbf{H}_{i+1,i}^{-1}. \quad (12.11)$$

Since all terms on the right-hand side are known and $\mathbf{H}_{i+1,i}$ is of low order (it is frequently a scalar and would be so if vector sequences were being generated), \mathbf{Y}_{i+1} may be readily computed.

Assume now, for simplicity, that vector sequences are being computed. Equation (12.11) becomes

$$\mathbf{y}_{i+1} = \left(\mathbf{G}\mathbf{p}_i - \sum_{j=1}^i \mathbf{y}_j h_{ji} \right) h_{i+1,i}^{-1}$$

and the residuals \mathbf{p}_i are conjugate with respect to \mathbf{G} as they have been implicitly computed by GODir. Hence if \mathbf{G} is positive definite then for some $s \leq n + 1$, $\mathbf{p}_s = \mathbf{0}$ since at most n vectors of order n can be mutually conjugate with respect to a positive definite matrix. Since $\mathbf{p}_s = \mathbf{K}\mathbf{y}_s - \mathbf{c}$ from equation (12.4) this implies that \mathbf{y}_s is the solution of $\mathbf{K}\mathbf{y} = \mathbf{c}$. Moreover as the residuals \mathbf{p}_i are conjugate with respect to \mathbf{G} , if we put $\mathbf{G} = \mathbf{I}$ they are mutually orthogonal. It is not surprising therefore that this form of the algorithm is known as OrthoRes [260].

We now consider the possibility of solving $\mathbf{K}\mathbf{Y} = \mathbf{C}$ using a procedure based on GOMin rather than GODir. As before we compute \mathbf{P}_i using equation (12.6) to ensure conjugacy but we now take \mathbf{W}_i to be $\mathbf{K}\mathbf{F}_i \mathbf{U}_{ii}^{-1}$, where \mathbf{U}_{ii} is our normalisation matrix, nonsingular but otherwise arbitrary. Again if \mathbf{Q}_{i-1} is defined by equation (12.7) we get, by analogy with equation (12.8),

$$\mathbf{P}_i \mathbf{U}_{ii} = \mathbf{K}\mathbf{F}_i - \sum_{j=1}^{i-1} \mathbf{P}_j \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{G} \mathbf{K}\mathbf{F}_i \quad (12.12)$$

and if we define the matrices \mathbf{U}_{ji} by

$$\mathbf{U}_{ji} = \mathbf{D}_j^{-1} \mathbf{P}_j^T \mathbf{G} \mathbf{K}\mathbf{F}_i, \quad 1 \leq j \leq i-1,$$

this equation becomes

$$\mathbf{P}_i \mathbf{U}_{ii} = \mathbf{K}\mathbf{F}_i - \sum_{j=1}^{i-1} \mathbf{P}_j \mathbf{U}_{ji}.$$

Substituting $\mathbf{K}\mathbf{Y}_j - \mathbf{C}$ for \mathbf{P}_j as before then yields

$$\mathbf{Y}_i \mathbf{U}_{ii} = \mathbf{F}_i - \sum_{j=1}^{i-1} \mathbf{Y}_j \mathbf{U}_{ji} + \mathbf{K}^{-1} \mathbf{C} \sum_{j=1}^i \mathbf{U}_{ji}$$

which becomes, if we choose the arbitrary scaling matrix \mathbf{U}_{ii} by $\mathbf{U}_{ii} = -\sum_{j=1}^{i-1} \mathbf{U}_{ji}$,

$$\mathbf{Y}_i \mathbf{U}_{ii} = \mathbf{F}_i - \sum_{j=1}^{i-1} \mathbf{Y}_j \mathbf{U}_{ji}. \quad (12.13)$$

Again we have an equation where all quantities but \mathbf{Y}_i are known so if \mathbf{U}_{ii} is non-singular then \mathbf{Y}_i can be computed and we now have the HS analogue of OrthoRes. Rather than pursue this analogy, though, we strike out in a different direction and consider what happens to equation (12.13) when \mathbf{K} is symmetric.

If \mathbf{K} is symmetric, the scaled form of the HS version of block CG is given by equations (3.17) and (3.18) which for convenience we reproduce here:

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G}\mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{B}_i^T \mathbf{C}_i \quad (12.14)$$

and

$$\mathbf{P}_{i+1} = (\mathbf{K}\mathbf{F}_{i+1} + \mathbf{P}_i \mathbf{B}_i^{-1} \mathbf{C}_i^{-1} \mathbf{C}_{i+1}) \mathbf{B}_{i+1}. \quad (12.15)$$

We would like these two equations to have the same form if possible and one way of achieving this is to choose the arbitrary scaling matrices \mathbf{B}_i and \mathbf{B}_{i+1} to satisfy $\mathbf{C}_i \mathbf{B}_i = \mathbf{C}_{i+1} \mathbf{B}_{i+1}$. This suggests that we require that

$$\mathbf{C}_i \mathbf{B}_i = \mathbf{M} \quad (12.16)$$

for some constant matrix \mathbf{M} and for all values of i . Now since \mathbf{K} is symmetric by hypothesis and \mathbf{C}_i is defined by $\mathbf{C}_i = \mathbf{F}_i^T \mathbf{K} \mathbf{F}_i$ it follows that \mathbf{C}_i too is symmetric. Substituting from equation (12.16) into equations (12.14) and (12.15) and exploiting the symmetry of \mathbf{C}_i then gives

$$\mathbf{P}_{i+1} = \mathbf{P}_i + \mathbf{K}\mathbf{F}_{i+1} \mathbf{C}_{i+1}^{-1} \mathbf{M}. \quad (12.17)$$

and

$$\mathbf{F}_{i+1} = \mathbf{F}_i - \mathbf{G}\mathbf{P}_i \mathbf{D}_i^{-1} \mathbf{M}^T \quad (12.18)$$

and these two equations demonstrate, by their symmetries and their similarities, the duality inherent in the conjugate gradient methods. The duality is even more marked in the vector forms of the equations. In these variants the matrices \mathbf{C}_i , \mathbf{D}_i and \mathbf{M} become the scalars $\gamma_i = \mathbf{f}_i^T \mathbf{K} \mathbf{f}_i$, $\delta_i = \mathbf{p}_i^T \mathbf{G} \mathbf{p}_i$ and κ (say) and equations (12.18) and (12.17) become respectively

$$\mathbf{f}_{i+1} = \mathbf{f}_i - \mathbf{G} \mathbf{p}_i \left(\frac{\kappa}{\delta_i} \right) \quad (12.19)$$

and

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{Kf}_{i+1} \left(\frac{\kappa}{\gamma_{i+1}} \right), \quad (12.20)$$

revealing a structure of great simplicity and beauty. They form the basis of algorithms for solving both $\mathbf{Gx} = \mathbf{h}$ and $\mathbf{Ky} = \mathbf{c}$ and were first given in this form by Broyden and Foschi [51].

12.1. Interpretations

Equations (12.19) and (12.20) can be used to solve both the primal system $\mathbf{Gx} = \mathbf{h}$ and the dual system $\mathbf{Ky} = \mathbf{c}$ but it is confusing to have a single algorithm to solve two different problems. It is perhaps better to derive from the primal equations a set of dual equations that may be used to construct an alternative algorithm for solving the primal problem. To do so it suffices to interchange \mathbf{G} and \mathbf{K} , \mathbf{f} and \mathbf{p} and δ and γ in equations (12.19) and (12.20) and this gives the fundamental equations of the dual algorithm to be

$$\mathbf{p}_{i+1} = \mathbf{p}_i - \mathbf{Kf}_i \left(\frac{\kappa}{\gamma_i} \right) \quad (12.21)$$

and

$$\mathbf{f}_{i+1} = \mathbf{f}_i + \mathbf{Gp}_{i+1} \left(\frac{\kappa}{\delta_{i+1}} \right) \quad (12.22)$$

while from equation (12.22) we obtain, since $\mathbf{f}_i = \mathbf{Gx}_i - \mathbf{h}$,

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{p}_{i+1} \left(\frac{\kappa}{\delta_{i+1}} \right). \quad (12.23)$$

We note that if we have already computed \mathbf{f}_i and \mathbf{p}_i then to compute \mathbf{f}_{i+1} and \mathbf{p}_{i+1} by the primal algorithm we first compute \mathbf{f}_{i+1} and then \mathbf{p}_{i+1} , but in the dual algorithm the order of calculation is reversed.

The simplicity of equations (12.19) and (12.20) suggests other possibilities. If we write them as

$$\mathbf{p}_i = \mathbf{p}_{i+1} - \mathbf{Kf}_{i+1} \left(\frac{\kappa}{\gamma_{i+1}} \right) \quad (12.24)$$

and

$$\mathbf{f}_i = \mathbf{f}_{i+1} + \mathbf{Gp}_i \left(\frac{\kappa}{\delta_i} \right) \quad (12.25)$$

we see that it is possible, given \mathbf{p}_{i+1} and \mathbf{f}_{i+1} , to compute \mathbf{p}_i and \mathbf{f}_i . The sequences may thus be generated in reverse order, opening up the possibility of reverse forms of the CG methods, but just as in the case of the dual methods the reverse forms

are difficult to envisage, if only because decreasing subscripts opens the possibility of their becoming negative. However, all we have to do to overcome this is to interchange the subscripts i and $i + 1$ in equations (12.24) and (12.25). The subscripts will now increase and the equations we obtain are, precisely, equations (12.21) and (12.22). The dual algorithm and the reverse algorithm are identical!

The reverse algorithms were first discussed by Hegedüs [149] - [152] and the dual algorithms, and their identification with the reverse algorithms, by Broyden and Foschi [51]. Apart from OrthoRes, which has been with us since 1980, these methods are virtually unknown. The major difference between them and their primal counterparts is probably their stability, the dual algorithm stagnating whereas the corresponding primal algorithm would crash, and vice-versa. The block forms of equations (12.21) - (12.23), or equivalently the dual forms of equations (12.17) and (12.18), are

$$\mathbf{P}_{i+1} = \mathbf{P}_i - \mathbf{K}\mathbf{F}_i\mathbf{C}_i^{-1}\mathbf{M}^T, \quad (12.26)$$

$$\mathbf{F}_{i+1} = \mathbf{F}_i + \mathbf{G}\mathbf{P}_{i+1}\mathbf{D}_{i+1}^{-1}\mathbf{M} \quad (12.27)$$

and

$$\mathbf{X}_{i+1} = \mathbf{X}_i + \mathbf{P}_{i+1}\mathbf{D}_{i+1}^{-1}\mathbf{M} \quad (12.28)$$

and just in the same way that particular values of \mathbf{G} and \mathbf{K} were substituted in equations (3.19) - (3.21) to give the HS algorithms of Chapter 3, so could these same values be substituted in equations (12.26) - (12.28) to give the corresponding dual algorithms. This, at the time of writing, is almost totally uncharted territory, the authors knowing of no references apart from those already cited together with a few references specific to OrthoRes. The properties of these methods must therefore be the subject of another chapter to be written at some time in the future, and are beyond the scope of the present volume.

This page intentionally left blank

Appendix A

Reduction of upper Hessenberg matrix to upper triangular form

We first consider the 2×2 case and derive the orthogonal matrix that transforms $[\alpha \ \beta]^T$ to $\begin{bmatrix} \sqrt{\alpha^2 + \beta^2} & 0 \\ \beta & 0 \end{bmatrix}^T$, where α and β are arbitrary real scalars. Define \mathbf{P}_r by

$$\mathbf{P}_r = \begin{bmatrix} c_r & -s_r \\ s_r & c_r \end{bmatrix}. \quad (\text{A.1})$$

It is now straightforward to show that if

$$c_r = \frac{\alpha}{\sqrt{\alpha^2 + \beta^2}},$$

and

$$s_r = \frac{-\beta}{\sqrt{\alpha^2 + \beta^2}}$$

(so that c_r and s_r may be identified with the cosine and sine of some angle) then \mathbf{P}_r is orthogonal and

$$\mathbf{P}_r \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \sqrt{\alpha^2 + \beta^2} \\ 0 \end{bmatrix}.$$

Let now $\tilde{\mathbf{H}}_{i+1} \in \mathbb{R}^{(i+1) \times i}$ be upper Hessenberg, and assume that after $r-1$ plane rotations it has the form \mathbf{K}_r , where

$$\mathbf{K}_r = \begin{bmatrix} \mathbf{U}_r \\ \mathbf{H}_{i-r+1} \end{bmatrix}, \quad (\text{A.2})$$

\mathbf{U}_r consists of the *first* r rows of some upper triangular matrix and \mathbf{H}_{i-r+1} consists of the *last* $(i - r + 1)$ rows of $\tilde{\mathbf{H}}_{i+1}$. This partition is clearly possible for $r = 1$. Define now the orthogonal matrix \mathbf{S}_r by

$$\mathbf{S}_r = \begin{bmatrix} \mathbf{I}_{r-1} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{P}_r & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{I}_{i-r} \end{bmatrix} \quad (\text{A.3})$$

where the subscripts on the identities (but not the matrix \mathbf{P}_r) denote their orders. Clearly if we premultiply \mathbf{K}_r by \mathbf{S}_r we leave the first $(r - 1)$ rows of \mathbf{U}_r and the last $(i - r)$ rows of \mathbf{H}_{i-r+1} unchanged, but replace the last row of \mathbf{U}_r and the first row of \mathbf{H}_{i-r+1} by linear combinations of these same two rows. *Thus the r -th plane rotation replaces rows r and $r + 1$ of \mathbf{K}_r by linear combinations of those same two rows while leaving all other rows unchanged, yielding \mathbf{K}_{r+1} .* Now if \mathbf{P}_r is chosen appropriately, \mathbf{K}_{r+1} admits to the same partitioning as \mathbf{K}_r and if this process is repeated $(i - 1)$ times we obtain a matrix of upper triangular form.

To see that \mathbf{K}_{r+1} may be so partitioned it is perhaps easier to consider a specific example, and the one chosen is where $i = 7$ and $r = 4$. Let

$$\mathbf{K}_4 = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \alpha & \times & 0 \\ \beta & \times & \times \\ \times & \times & \times \\ \times & \times & \\ \times & & \end{bmatrix}$$

where \times denotes a generic non-zero and a blank denotes a zero (in this particular example the original upper Hessenberg matrix is tridiagonal). Premultiplication by \mathbf{S}_4 replaces rows four and five by linear combinations of these two rows and if \mathbf{P}_r is as defined above, α is replaced by $\sqrt{\alpha^2 + \beta^2}$ and β is replaced by zero. The element denoted by 0 is replaced by some non-zero element with all the other zeroes remaining unchanged. It is then obvious that \mathbf{K}_5 may be partitioned according to equation (A.2) with \mathbf{U}_5 and \mathbf{H}_2 having the appropriate forms. Moreover it follows from the above example that a tridiagonal matrix is transformed into a tridiagonal upper triangular matrix with only the principal and two adjacent super-diagonals being non-null. More generally, if \mathbf{H}_{i+1} is a general band matrix the resulting upper-triangular matrix has the same structure as $\tilde{\mathbf{H}}_{i+1}$ but with the band shifted one column to the right.

Let now $\bar{\mathbf{H}}_{i-1}$ denote the leading principal submatrix of order $i - 1$ of some upper Hessenberg matrix and let $\tilde{\mathbf{H}}_i$ denote the submatrix consisting of the first i rows and $i - 1$ columns of the same matrix so that

$$\tilde{\mathbf{H}}_{i+1} = \begin{bmatrix} \bar{\mathbf{H}}_{i-1} & \mathbf{h}_i \\ h_{i,i-1}\mathbf{e}_{i-1}^T & h_{ii} \\ \mathbf{0}^T & h_{i+1,i} \end{bmatrix}.$$

for some vector \mathbf{h}_i and scalars $h_{i,i-1}$, h_{ii} and $h_{i+1,i}$. It follows from the above discussion that if we perform $(i-2)$ plane rotations on $\tilde{\mathbf{H}}_{i+1}$ we obtain a matrix of the form

$$\mathbf{K}_{i-1} = \begin{bmatrix} \overline{\mathbf{U}}_{i-1} & \overline{\mathbf{u}}_i \\ h_{i,i-1}\mathbf{e}_{i-1}^T & h_{ii} \\ \mathbf{0}^T & h_{i+1,i} \end{bmatrix}$$

where $\overline{\mathbf{U}}_{i-1}$ is upper triangular with its last diagonal element equal to $\overline{u}_{i-1,i-1}$ say. The matrix $\overline{\mathbf{U}}_{i-1}$ will thus be the matrix $\overline{\mathbf{H}}_{i-1}$ transformed to upper triangular form. If we now perform a further plane rotation (the $(i-1)th$), this affects only rows $i-1$ and i of \mathbf{K}_{i-1} , that is it affects only the last row of $\overline{\mathbf{U}}_{i-1}$ and element of $\overline{\mathbf{u}}_i$ together with the row denoted by $[h_{i,i-1}\mathbf{e}_{i-1}^T \ h_{ii}]$, and the first partition of this will be nullified. Thus this rotation yields a matrix of the form

$$\mathbf{K}_i = \begin{bmatrix} \tilde{\mathbf{U}}_{i-1} & \tilde{\mathbf{u}}_i \\ \mathbf{0}^T & \tilde{u}_{ii} \\ \mathbf{0}^T & h_{i+1,i} \end{bmatrix}$$

where $\tilde{\mathbf{U}}_{i-1}$ is upper triangular and is identical to $\overline{\mathbf{U}}_{i-1}$ except that its last diagonal element is different and equal to $\tilde{u}_{i-1,i-1}$ say, where

$$\tilde{u}_{i-1,i-1} = \sqrt{\tilde{u}_{i-1,i-1}^2 + h_{i,i-1}^2}.$$

A further rotation, involving rows i and $i+1$, then gives

$$\mathbf{K}_{i+1} = \begin{bmatrix} \tilde{\mathbf{U}}_{i-1} & \tilde{\mathbf{u}}_i \\ \mathbf{0}^T & \tilde{u}_{ii} \\ \mathbf{0}^T & 0 \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{U}}_i \\ \mathbf{0}^T \end{bmatrix}$$

where

$$\tilde{\mathbf{U}}_i = \begin{bmatrix} \tilde{\mathbf{U}}_{i-1} & \tilde{\mathbf{u}}_i \\ \mathbf{0}^T & \tilde{u}_{ii} \end{bmatrix} \quad (\text{A.4})$$

and $\tilde{u}_{ii} = \sqrt{\tilde{u}_{ii}^2 + h_{i+1,i}^2}$. Any additional rotations involve only rows $i+1$ and higher so will leave the matrix $\tilde{\mathbf{U}}_i$ intact. It follows from this that the sequence $\{\overline{\mathbf{U}}_i\}$ is not a nested sequence since $\overline{\mathbf{U}}_i$ is not a submatrix of $\overline{\mathbf{U}}_{i+1}$, but that the sequence $\{\tilde{\mathbf{U}}_i\}$ is a nested sequence. Note that $\overline{\mathbf{U}}_i$ is singular if and only if $\overline{\mathbf{H}}_i$ is singular but $\tilde{\mathbf{U}}_i$ is singular if and only if $\tilde{\mathbf{H}}_{i+1}$ is rank-deficient, and this is true regardless of whether $\overline{\mathbf{H}}_i$ is singular or not.

The calculation of the matrices $\{\tilde{\mathbf{U}}_i\}$ is usually associated with carrying out the identical orthogonal transformations on some given vector since we often want the least-squares solution of

$$\tilde{\mathbf{H}}_{i+1}\mathbf{z} + \mathbf{c}_{i+1} = \mathbf{0} \quad (\text{A.5})$$

for some given vector \mathbf{c}_{i+1} (see e.g. the section FOM and GMRes, page 2.6 above). Let then, analogously to $\tilde{\mathbf{H}}_{i+1}$, the original vector \mathbf{c}_{i+1} be defined by $\mathbf{c}_{i+1} = [\gamma_i] \in \mathbb{R}^{i+1}$ and, after $i-2$, $i-1$ and i transformations respectively, denote it by

$$\begin{bmatrix} \bar{\mathbf{c}}_{i-1} \\ \gamma_i \\ \gamma_{i+1} \end{bmatrix}, \quad \begin{bmatrix} \tilde{\mathbf{c}}_{i-1} \\ \bar{\gamma}_i \\ \gamma_{i+1} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \tilde{\mathbf{c}}_{i-1} \\ \bar{\gamma}_i \\ \bar{\gamma}_{i+1} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{c}}_i \\ \bar{\gamma}_{i+1} \end{bmatrix}.$$

Here, after $i-2$ rotations, every element of \mathbf{c}_{i-1} will have been changed but not γ_i or γ_{i+1} . The next rotation changes γ_i to $\bar{\gamma}_i$ and updates $\bar{\mathbf{c}}_{i-1}$ to $\tilde{\mathbf{c}}_{i-1}$ by changing its last element. This vector then remains constant thereafter. The $i-th$ rotation changes $\bar{\gamma}_i$ to $\tilde{\gamma}_i$ (which suffers no further change) and γ_{i+1} to $\bar{\gamma}_{i+1}$. Thus equation (A.5) will have been transformed after i rotations into

$$\begin{bmatrix} \tilde{\mathbf{U}}_i \\ \mathbf{0}^T \end{bmatrix} \mathbf{z} + \begin{bmatrix} \tilde{\mathbf{c}}_i \\ \bar{\gamma}_{i+1} \end{bmatrix} = \mathbf{0}. \quad (\text{A.6})$$

The least-squares solution of this is simply $\tilde{\mathbf{U}}_i^{-1}\tilde{\mathbf{c}}_i$ and the norm of the residual at the solution is $|\bar{\gamma}_{i+1}|$. The detailed calculation of the sequence $\{\tilde{\mathbf{U}}_i^{-1}\tilde{\mathbf{c}}_i\}$ is described in Chapter 6.

It is also useful to have some idea of the structure of \mathbf{Q}_{i+1} , where $\mathbf{Q}_{i+1} \in \mathbb{R}^{(i+1) \times (i+1)}$ is the orthogonal matrix that represents the product of the first i rotations. Since \mathbf{Q}_{i+1} has an extra row and column over \mathbf{Q}_i , in order to obtain its structure we first have to bring \mathbf{Q}_i up to size and we do this by first constructing the matrix $\mathbf{P}_{i+1} \in \mathbb{R}^{(i+1) \times (i+1)}$, where

$$\mathbf{P}_{i+1} = \begin{bmatrix} \mathbf{Q}_i & \mathbf{z}_1 \\ \mathbf{z}_2^T & 1 \end{bmatrix}$$

and where \mathbf{z}_1 and \mathbf{z}_2 are both null (!). To obtain \mathbf{Q}_{i+1} from \mathbf{P}_{i+1} we apply to the latter the $i-th$ plane rotation. If \mathbf{p}_j^T denotes the $j-th$ row of \mathbf{P}_{i+1} with \mathbf{q}_j^T similarly defined this implies, from equation (A.1), that $\mathbf{q}_i^T = c_i \mathbf{p}_i^T - s_i \mathbf{p}_{i+1}^T$ and $\mathbf{q}_{i+1}^T = s_i \mathbf{p}_i^T + c_i \mathbf{p}_{i+1}^T$. Since, however, \mathbf{z}_1 and \mathbf{z}_2 are both null this leads to the following rules:

- Replace the last row of \mathbf{Q}_i by c_i times itself
- Replace \mathbf{z}_2^T by s_i times the last row of \mathbf{Q}_i
- Replace the last zero of \mathbf{z}_1 and the unit element by $-s_i$ and c_i respectively.

From this we deduce immediately that \mathbf{Q}_{i+1} is lower Hessenberg, and its general structure may be inferred from the following example constructed according to the above rules:

$$\mathbf{Q}_5 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 & 0 \\ c_2 s_1 & c_2 c_1 & -s_2 & 0 & 0 \\ c_3 s_2 s_1 & c_3 s_2 c_1 & c_3 c_2 & -s_3 & 0 \\ c_4 s_3 s_2 s_1 & c_4 s_3 s_2 c_1 & c_4 s_3 c_2 & c_4 c_3 & -s_4 \\ s_4 s_3 s_2 s_1 & s_4 s_3 s_2 c_1 & s_4 s_3 c_2 & s_4 c_3 & c_4 \end{bmatrix}$$

from which we deduce

$$\mathbf{e}_i^T \mathbf{Q}_i \mathbf{e}_1 = \prod_{j=1}^{i-1} s_j. \quad (\text{A.7})$$

This gives the norm reduction over i steps of GMRes. The above results are important for understanding the relationships between FOM, GMRes, LSQR and MinRes, and for deriving the implementation of the QMR technique.

This page intentionally left blank

Appendix B

Schur complements

Let

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \quad (\text{B.1})$$

where \mathbf{A}_{12} (say) is square and nonsingular. Then the Schur complement of \mathbf{A}_{12} in \mathbf{A} , \mathbf{S}_{21} say, is defined by

$$\mathbf{S}_{21} = \mathbf{A}_{21} - \mathbf{A}_{22}\mathbf{A}_{12}^{-1}\mathbf{A}_{11}. \quad (\text{B.2})$$

It is possible to define the Schur complement of any of the four submatrices \mathbf{A}_{ij} in \mathbf{A} provided that the submatrix is square and nonsingular, and although it is more common to find the Schur complements of \mathbf{A}_{11} or \mathbf{A}_{22} we give as our paradigm that of \mathbf{A}_{12} to emphasise that these nonsymmetric versions are possible. The matrix \mathbf{A} is normally square, although this is not essential for the definition of the complement, but if it is we can prove the following theorem

Theorem 38. Let \mathbf{A} and \mathbf{S}_{21} satisfy equations (B.1) and (B.2), where \mathbf{A}_{12} is assumed to be square and nonsingular and \mathbf{A} is assumed to be square. Thus \mathbf{A} is singular if and only if \mathbf{S}_{21} is singular.

Proof. Let \mathbf{A} be singular. Then there exists a vector $\mathbf{x} \neq \mathbf{0}$ such that $\mathbf{Ax} = \mathbf{0}$ or, from equation (B.1) and with obvious notation, $\mathbf{A}_{11}\mathbf{x}_1 + \mathbf{A}_{12}\mathbf{x}_2 = \mathbf{0}$ and $\mathbf{A}_{21}\mathbf{x}_1 + \mathbf{A}_{22}\mathbf{x}_2 = \mathbf{0}$. Thus $\mathbf{x}_2 = -\mathbf{A}_{12}^{-1}\mathbf{A}_{11}\mathbf{x}_1$ and $\mathbf{S}_{21}\mathbf{x}_1 = \mathbf{0}$. Now $\mathbf{x}_1 \neq \mathbf{0}$ since if otherwise $\mathbf{x}_2 = \mathbf{0}$ implying $\mathbf{x} = \mathbf{0}$ giving a contradiction so that \mathbf{S}_{21} is singular. Conversely, if \mathbf{S}_{21} is singular there exists an $\mathbf{x}_1 \neq \mathbf{0}$ such that $\mathbf{S}_{21}\mathbf{x}_1 = \mathbf{0}$ or, from equation (B.2), $(\mathbf{A}_{21} - \mathbf{A}_{22}\mathbf{A}_{12}^{-1}\mathbf{A}_{11})\mathbf{x}_1 = \mathbf{0}$, and if we define \mathbf{x}_2 by $\mathbf{x}_2 = -\mathbf{A}_{12}^{-1}\mathbf{A}_{11}\mathbf{x}_1$ it follows from this equation and the previous one that $\mathbf{Ax} = \mathbf{0}$ where $\mathbf{x} \neq \mathbf{0}$ since $\mathbf{x}_1 \neq \mathbf{0}$. Thus \mathbf{A} is singular proving the theorem. ■

The importance of Schur complements in the study of conjugate gradient methods stems from their relationship to projection matrices. Let $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times r}$ where $n > r$ and let $\mathbf{V}^T\mathbf{U}$ be nonsingular. Then $\mathbf{Q} = \mathbf{I} - \mathbf{U}(\mathbf{V}^T\mathbf{U})^{-1}\mathbf{V}^T$ is an oblique projector since it is idempotent. But \mathbf{Q} is also the Schur complement of $\mathbf{V}^T\mathbf{U}$ in \mathbf{M} , where

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} \\ \mathbf{V}^T \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{U} \end{bmatrix},$$

a result that is useful in determining the causes of breakdown in the CG methods.

Finally if \mathbf{A} satisfies equation (B.1) and \mathbf{A}_{11} is square and nonsingular then it is trivial to verify that if \mathbf{A} is nonsingular then

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{A}_{11}^{-1} + \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{S}_{22}^{-1}\mathbf{A}_{21}\mathbf{A}_{11}^{-1} - \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{S}_{22}^{-1} \\ -\mathbf{S}_{22}^{-1}\mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{S}_{22}^{-1} \end{bmatrix} \quad (\text{B.3})$$

where $\mathbf{S}_{22} = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}$. We infer from this that if \mathbf{A}_{11} and \mathbf{A} are both nonsingular then its lower right-hand submatrix of \mathbf{A}^{-1} is also nonsingular since it is the inverse of the Schur complement of \mathbf{A}_{11} in \mathbf{A} which is itself nonsingular from Theorem 38. An alternative form of equation (B.3) is

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{A}_{11}^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{bmatrix} + \begin{bmatrix} \mathbf{A}_{11}^{-1}\mathbf{A}_{12} \\ -\mathbf{I} \end{bmatrix} \mathbf{S}_{22}^{-1} \begin{bmatrix} \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & -\mathbf{I} \end{bmatrix}$$

Appendix C

The Jordan Form

A theorem by Jordan (see e.g. [155]) states that for any $\mathbf{A} \in \mathbb{C}^{n \times n}$ there exists a nonsingular matrix \mathbf{X} such that

$$\mathbf{X}^{-1} \mathbf{A} \mathbf{X} = \mathbf{J} \quad (\text{C.1})$$

where \mathbf{J} is block diagonal with diagonal blocks of the form

$$\mathbf{J}_k = \begin{bmatrix} \lambda_k & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_k & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_k & 1 \\ 0 & 0 & 0 & \cdots & 0 & \lambda_k \end{bmatrix}.$$

These matrices \mathbf{J}_k are called *Jordan blocks* and \mathbf{J} is often referred to as the *direct sum* of such blocks. If two or more Jordan blocks have the same value of λ_k on the principal diagonal a simple permutation of the columns of \mathbf{X} can ensure that, within \mathbf{J} , all such blocks are adjacent to each other. \mathbf{J} can thus be expressed as the direct sum of p matrices \mathbf{U}_i , where each \mathbf{U}_i is either a Jordan block or is itself the direct sum of such blocks with identical diagonal elements. Since the various values of λ_k are the eigenvalues of the matrix, p denotes the number of distinct eigenvalues of \mathbf{A} .

For our purposes this elegant structure has a degree of overkill. Our more modest requirements, for which the detailed internal structure outlined above is sufficient but by no means necessary, are:

- (1) $\mathbf{U}_i - \lambda_j \mathbf{I}$ is singular if and only if $j = i$, and
- (2) $\mathbf{U}_i - \lambda_i \mathbf{I}$ is strictly upper triangular so that there exists a smallest positive integer k_i such that $(\mathbf{U}_i - \lambda_i \mathbf{I})^{k_i} = \mathbf{O}$.

Assume now that $\mathbf{U}_i \in \mathbb{C}^{m_i \times m_i}$ so that $n = \sum_{i=1}^p m_i$. Then, trivially,

$$(\mathbf{U}_i - \lambda_i \mathbf{I})^{m_i} = \mathbf{O}$$

implying that $k_i \leq m_i$. In fact it is equally simple to deduce that k_i is the dimension of the largest Jordan block occurring in \mathbf{U}_i . The extreme cases occur when \mathbf{U}_i

consists of a single Jordan block when $k_i = m_i$, or alternatively when all the Jordan blocks occurring in \mathbf{U}_i have dimension unity so that for each such block, $\mathbf{J}_k = \lambda_i$. In this case $\mathbf{U}_i = \lambda_i \mathbf{I}$ so that $\mathbf{U}_i - \lambda_i \mathbf{I}$ is itself null and $k_i = 1$. This is the commonest case (it occurs always if \mathbf{A} is real and symmetric) but if \mathbf{A} is non-symmetric all available configurations are possible.

Let now

$$\mathbf{X} = [\mathbf{X}_1 \ \mathbf{X}_2 \ \cdots \ \mathbf{X}_p]$$

where $\mathbf{X}_i \in \mathbb{C}^{n \times m_i}$. Define \mathbf{Y} by $\mathbf{Y}^H = \mathbf{X}^{-1}$, where the superscript H denotes transposition of the complex conjugate (the Hermitian transpose), and similarly partition \mathbf{Y} . Equation (C.1) then becomes

$$\mathbf{A} = \mathbf{X} \mathbf{J} \mathbf{Y}^H = \sum_{i=1}^p \mathbf{X}_i \mathbf{U}_i \mathbf{Y}_i^H. \quad (\text{C.2})$$

Since $\mathbf{Y}^H \mathbf{X} = \mathbf{I}$ the matrices \mathbf{X}_i and \mathbf{Y}_i satisfy $\mathbf{Y}_i^H \mathbf{X}_j = \mathbf{O}$ if $i \neq j$ and $\mathbf{Y}_i^H \mathbf{X}_i = \mathbf{I}$. Thus $\mathbf{X}_i \mathbf{Y}_i^H$ is an oblique projector, orthogonal if $\mathbf{Y} = \mathbf{X}$. It follows trivially from equation (C.2) that, for any positive integer k ,

$$\mathbf{A}^k \equiv \sum_{i=1}^p \mathbf{X}_i \mathbf{U}_i^k \mathbf{Y}_i^H$$

and a simple extension of this argument indicates that if $p_r(\mathbf{A})$ denotes any polynomial of degree r in \mathbf{A} then

$$p_r(\mathbf{A}) \equiv \sum_{i=1}^p \mathbf{X}_i p_r(\mathbf{U}_i) \mathbf{Y}_i^H. \quad (\text{C.3})$$

This equation is used in Chapter 4 to analyse the convergence of GMRes.

Appendix D

Chebychev polynomials

We give here a brief account of the Chebychev polynomials sufficient to underpin the analyses in Chapter 4 and Chapter 11. Define the Chebychev polynomial of the first kind of degree k , $T_k(x)$, by $T_0(x) \equiv 1$, $T_1(x) \equiv x$ and, for $k \geq 1$,

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x). \quad (\text{D.1})$$

This is not perhaps the usual definition but it has the virtue of establishing from the outset that $T_k(x)$ is indeed a polynomial of degree k in x . We show, that if $T_k(x)$ is so defined, then

$$T_k(x) \equiv \frac{1}{2} \left[\left(x + \sqrt{x^2 - 1} \right)^k + \left(x - \sqrt{x^2 - 1} \right)^k \right] \quad (\text{D.2})$$

but before we do this we examine some of the quantities involved. We note that both $x + \sqrt{x^2 - 1}$ and $x - \sqrt{x^2 - 1}$ are just the sums of x plus a square root (positive or negative) of $x^2 - 1$ and so are in a sense dual. Moreover since

$$\left(x + \sqrt{x^2 - 1} \right) \left(x - \sqrt{x^2 - 1} \right) \equiv 1$$

we can write equation (D.2) as

$$T_k(x) \equiv \frac{1}{2} (z^k + z^{-k}) \quad (\text{D.3})$$

where

$$z \equiv x + \sqrt{x^2 - 1}, \quad (\text{D.4})$$

although equation (D.3) would be equally valid if we chose the negative square root in equation (D.4). Also, from equation (D.4),

$$\frac{1}{2}z^2 \equiv xz - \frac{1}{2} \quad \text{and} \quad \frac{1}{2}z^{-2} \equiv xz^{-1} - \frac{1}{2} \quad (\text{D.5})$$

and with these identities established the derivation of equation (D.2) is straightforward.

We note first that this equation is trivially true for $k = 0$ and $k = 1$ and proceed by induction. Assume that equation (D.3) is true for $k - 1$ and for k . Then, from equation (D.1),

$$\begin{aligned} T_{k+1}(x) &\equiv x(z^k + z^{-k}) - \frac{1}{2}(z^{k-1} + z^{-(k-1)}) \\ &\equiv \left(xz - \frac{1}{2}\right)z^{k-1} + \left(xz^{-1} - \frac{1}{2}\right)z^{-(k-1)} \end{aligned}$$

and it follows immediately from equation (D.5) that this equation is simply equation (D.3) with k replaced by $k + 1$. Since equation (D.3) is valid for $k = 0$ and $k = 1$ it is thus valid for any positive integer k . We note, too, that all the above equations are valid even if $|x| < 1$ and z is thus complex even if we assume, as we do, that x is real.

We can now use equations (D.3) and (D.4) to deduce some of the elementary properties of the Chebychev polynomials. We first note, from these two equations, that for $x \geq 1$, $T_k(x) \geq 1$ and for $x \leq -1$, $T_k(x) \geq 1$ for k even and $T_k(x) \leq -1$ for k odd. Thus $T_k(x) \neq 0$ for $|x| \geq 1$ so all the zeroes of $T_k(x)$ must either lie in $(-1, 1)$ or be complex. We now show the former to be the case.

For $|x| < 1$ equation (D.3) may be written

$$T_k(x) = \frac{1}{2} \left[(x + i\sqrt{1-x^2})^k + (x - i\sqrt{1-x^2})^k \right]. \quad (\text{D.6})$$

Furthermore, since $-1 < x < 1$, there exists a unique value of θ , $0 < \theta < \pi$, such that $x = \cos \theta$ and for this value of θ , $\sin^2 \theta = 1 - x^2$ or $\sin \theta = \sqrt{1 - x^2}$, the positive value being assigned since by hypothesis $0 < \theta < \pi$. Thus, from equation (D.6),

$$\begin{aligned} T_k &\equiv \frac{1}{2} \left[(\cos \theta + i \sin \theta)^k + (\cos \theta - i \sin \theta)^k \right] \\ &= \cos(k\theta) \end{aligned} \quad (\text{D.7})$$

by de Moivre's theorem. Hence, for $-1 < x < 1$,

$$T_k(x) \equiv \cos(k \cos^{-1} x). \quad (\text{D.8})$$

Equation (D.7) gives us the remaining essential information pertaining to $T_k(x)$ for $-1 < x < 1$ since for this range of x , $0 < \theta < \pi$. In this interval the function $\cos k\theta$ has exactly k zeroes at $\theta = \frac{\pi}{2k}, \frac{3\pi}{2k}, \dots, \frac{(2k-1)\pi}{2k}$ and $|\cos \theta|$ achieves its maximum value of unity ($k + 1$) times at $\theta = 0, \frac{\pi}{k}, \frac{2\pi}{k}, \dots, \pi$.

Appendix E

The companion matrix

The *companion matrix* \mathbf{C} of any polynomial

$$p_n(\lambda) \equiv \lambda^n + a_{n-1}\lambda^{n-1} + \dots + a_0 \quad (\text{E.1})$$

is defined to be the matrix

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \dots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & -a_{n-2} \\ 0 & 0 & \dots & 1 & -a_{n-1} \end{bmatrix}.$$

From the point of view of proving Theorem 20 it suffices to show that the eigenvalues λ_i of \mathbf{C} satisfy $p_n(\lambda_i) = 0$. To do this we note that the eigenvalues of \mathbf{C} are also those of \mathbf{C}^T and consider the equation $\mathbf{C}^T \mathbf{x} = \mathbf{x}\lambda$. If $\mathbf{x} = [x_i]$, equating the first $n-1$ elements of $\mathbf{C}^T \mathbf{x}$ and $\mathbf{x}\lambda$ gives

$$x_{i+1} = \lambda x_i, \quad i = 1, 2, \dots, n-1$$

while equating the final elements gives

$$-\sum_{i=1}^n \alpha_{i-1} x_i = \lambda x_n. \quad (\text{E.2})$$

This implies that $x_1 \neq 0$ (since if $x_1 = 0$ then $\mathbf{x} = \mathbf{0}$ and cannot be an eigenvector), and that $x_i = \lambda^{i-1} x_1$, $1 \leq i \leq n$. Substituting these values for x_i and x_n in equation (E.2) then gives

$$-\sum_{i=1}^n \alpha_{i-1} \lambda^{i-1} x_1 = \lambda^n x_1$$

or, from equation (E.1) and since $x_1 \neq 0$, $p_n(\lambda) = 0$. Thus any eigenvalue λ_i of \mathbf{C} must satisfy $p_n(\lambda_i) = 0$. Conversely, if $p_n(\lambda) = 0$, then $\mathbf{x} = [1, \lambda, \dots, \lambda^{n-1}]$ is an

eigenvector of \mathbf{C}^T corresponding to the eigenvalue λ . It is therefore simple in principle to construct a matrix having the form required by equation (4.49) and having the prescribed eigenvalues λ_i merely by putting $p_n(\lambda) \equiv (\lambda - \lambda_1)(\lambda - \lambda_2) \dots (\lambda - \lambda_n)$.

Appendix F

The algorithms

In this appendix we give details of the principal algorithms used in our tests. Since these are closely related to the flopcounts, approximate values of these are given in this section as well in the columns headed by $O(n)$ etc.

We assume the Hessenberg matrix to be $\mathbf{H} = [h_{ij}]$. The vectors \mathbf{z}_j and $\tilde{\mathbf{z}}_j$ are working vectors. The first three procedures are subroutines used by the other procedures.

$OTinit(k)$
$\delta_1 = [\ \mathbf{r}_1\ _2] \mathbf{z}_k = \tilde{\mathbf{z}}_k = \mathbf{0} s_k = 0 c_k = t_{kk} = 1$

$OT(p, q, r)$	$O(1)$
$t_{pq} = s_p * \tilde{h}_{qq}$	$\tilde{h}_{qq} = c_p * \tilde{h}_{qq}$
$s_q = \frac{h_{rq}}{t_{qq}}$	$d_q = c_q * \delta_q$
$\eta_p = \frac{t_{pq}}{t_{pp}}$	$\eta_p = \frac{t_{pq}}{t_{pp}}$
$t_{qq} = \sqrt{\tilde{h}_{qq}^2 + \tilde{h}_{rq}^2}$	$c_q = \frac{\tilde{h}_{qq}}{t_{qq}}$
$\delta_r = s_q * \delta_q$	$\gamma_q = \frac{d_q}{t_{qq}}$
	12

$OTplus(j, k, \mathbf{s}_j, \mathbf{y}_j)$	$O(n)$	$O(1)$
$OT(j - k, j, j + k)$		12
$\mathbf{z}_j = \mathbf{s}_j - \mathbf{z}_{j-k} \eta_{j-k}$	2	
$\tilde{\mathbf{z}}_j = \mathbf{y}_j - \tilde{\mathbf{z}}_{j-k} \eta_{j-k}$	2	
$\mathbf{x}_{j+k} = \mathbf{x}_j - \mathbf{z}_j \gamma_j$	2	
$\mathbf{r}_{j+k}^{QMR} = \mathbf{r}_j^{QMR} - \tilde{\mathbf{z}}_j \gamma_j$	2	

Algorithm GMRes(m)

GMRes(m)	$O(N)$	$O(n)$	$O(1)$	
$\mathbf{r}_i = \mathbf{Ax}_i - \mathbf{b}$		2	1	
$\nu_i = \ \mathbf{r}_i\ _2$			2	1
$\tilde{\mathbf{p}}_1 = \mathbf{r}_i / \nu_i$			1	
for $1 \leq j \leq m$				
$\mathbf{v}_j = \mathbf{A}\tilde{\mathbf{p}}_j$	2			
$\mathbf{h}_j = \tilde{\mathbf{P}}_j^T \mathbf{v}_j$		2j		
$\mathbf{p}_{j+1} = \mathbf{v}_j - \tilde{\mathbf{P}}_j \mathbf{h}_j$		2j		
$\eta_{j+1} = \ \mathbf{p}_{j+1}\ _2$		2	1	
$\tilde{\mathbf{p}}_{j+1} = \mathbf{p}_{j+1} / \eta_{j+1}$		1		
$OTgmr(j)$			8j	
$\mathbf{z}_j = \tilde{\mathbf{p}}_j - \mathbf{Z}_{j-1} \mathbf{u}_{j-1}$		2j	j	
$\mathbf{x}_{i+j} = \mathbf{x}_{i+j-1} - \mathbf{z}_j (d_j / \omega_j)$				
$\nu_{i+j} = s_j * \nu_{i+j-1}$			1	
$\rho_{i+j} = \nu_{i+j} / \nu_1$			1	
end for				

$OTgmr(j)$
$\hat{u} = h_1$
if $j > 1$
for $k = 1 : (j - 1)$
$u_k = c_k * \hat{u} + s_k * h_{k+1}$
$\hat{u} = -s_k * \hat{u} + c_k * h_{k+1}$
end for
end if
$\omega = \sqrt{\hat{u}^2 + \eta_{j+1}^2}$
$c_j = \hat{u} / \omega$
$s_j = \eta_{j+1} / \omega$
$d_j = c_j * \hat{\delta}$
$\hat{\delta} = -s_j * \hat{\delta}$

\mathbf{u}_{j-1} is the $(j-1)$ -dimensional vector whose components u_k , $1 \leq k \leq j-1$, are computed in $OTgmr(j)$.

The total flop count per group of m iterations is

$$(2m + 2)N + (3m^2 + 6m + 4)n + \left(\frac{9}{2}m^2 + \frac{15}{2}m + 1\right)$$

Algorithm BiCG

BiCG		$O(N)$	$O(n)$	$O(1)$
$\mathbf{v}_1 = \mathbf{u}_1 = \mathbf{s}_1 = \mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$		2	1	
$\nu_1 = \ \mathbf{r}_1\ _2$			2	1
$\rho_1 = \mathbf{s}_1^T \mathbf{r}_1$			2	
$\mathbf{w}_1 = \mathbf{A}\mathbf{u}_1$		2		
for $i = 1, 2, \dots$				
$\sigma_i = \mathbf{v}_i^T \mathbf{w}_i$		2		
$\alpha_i = \rho_i / \sigma_i$				1
$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \alpha_i$		2		
$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \alpha_i$		2		
$\ \mathbf{r}_{i+1}\ _2 / \nu_1$		2	2	
$\mathbf{y}_i = \mathbf{A}^T \mathbf{v}_i$	2			
$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \alpha_i$		2		
$\rho_{i+1} = \mathbf{s}_{i+1}^T \mathbf{r}_{i+1}$		2		
$\beta_{i+1} = \rho_{i+1} / \rho_i$				1
$\mathbf{u}_{i+1} = \mathbf{r}_{i+1} + \mathbf{u}_i \beta_{i+1}$		2		
$\mathbf{v}_{i+1} = \mathbf{s}_{i+1} + \mathbf{v}_i \beta_{i+1}$		2		
$\mathbf{w}_{i+1} = \mathbf{A}\mathbf{u}_{i+1}$	2			
end for				

Flop count for a single iteration $i > 1$: $4N + 16n + 4$

Algorithm QMRBiCG

QMRBiCG		$O(N)$	$O(n)$	$O(1)$
$\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$		2	1	
$\mathbf{v}_1 = \mathbf{u}_1 = \mathbf{s}_1 = \mathbf{r}_1^{QMR} = \mathbf{r}_1$				
$\nu_1 = \ \mathbf{r}_1\ _2$			2	1
$\rho_1 = \mathbf{s}_1^T \mathbf{r}_1$			2	
$\mathbf{w}_1 = \mathbf{Au}_1$		2		
$OTinit(0)$				
for $i = 1, 2, \dots$				
$\sigma_i = \mathbf{v}_i^T \mathbf{w}_i$		2		
$\alpha_i = \rho_i / \sigma_i$			1	
$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \alpha_i$		2		
$\nu_{i+1} = \ \mathbf{r}_{i+1}\ _2$		2	1	
ν_{i+1} / ν_1			1	
$\tilde{h}_{ii} = \nu_i / \alpha_i$			1	
$\tilde{h}_{i+1,i} = -\nu_{i+1} / \alpha_i$			1	
$OTplus(i, 1, \mathbf{u}_i, \mathbf{w}_i)$		8	12	
$\mathbf{y}_i = \mathbf{A}^T \mathbf{v}_i$		2		
$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \alpha_i$		2		
$\rho_{i+1} = \mathbf{s}_{i+1}^T \mathbf{r}_{i+1}$		2		
$\beta_{i+1} = \rho_{i+1} / \rho_i$			1	
$\mathbf{u}_{i+1} = \mathbf{r}_{i+1} + \mathbf{u}_i \beta_{i+1}$		2		
$\mathbf{v}_{i+1} = \mathbf{s}_{i+1} + \mathbf{v}_i \beta_{i+1}$			2	
$\mathbf{w}_{i+1} = \mathbf{A} \mathbf{u}_{i+1}$		2		
end for				

Flop count for a single iteration $i > 1$: $4N + 22n + 18$

Algorithm CGS

CGS	$O(N)$	$O(n)$	$O(1)$
$\hat{\mathbf{r}}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$	2	1	
$\hat{\mathbf{u}}_1 = \hat{\mathbf{q}}_1 = \mathbf{s}_1 = \hat{\mathbf{r}}_1$			
$\nu_1 = \ \hat{\mathbf{r}}_1\ _2$		2	1
$\rho_1 = \mathbf{s}_1^T \hat{\mathbf{r}}_1$		2	
$\hat{\mathbf{w}}_1 = \mathbf{A}\hat{\mathbf{u}}_1$	2		
for $i = 1, 2, \dots$			
$\sigma_i = \mathbf{s}_1^T \hat{\mathbf{w}}_i$		2	
$\alpha_i = \rho_i / \sigma_i$			1
$\hat{\mathbf{p}}_{i+1} = \hat{\mathbf{q}}_i - \hat{\mathbf{w}}_i \alpha_i$		2	
$\mathbf{y}_i = \hat{\mathbf{q}}_i + \hat{\mathbf{p}}_{i+1}$		1	
$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - \mathbf{y}_i \alpha_i$		2	
$\mathbf{z}_i = \mathbf{A}\mathbf{y}_i$	2		
$\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{r}}_i - \mathbf{z}_i \alpha_i$		2	
$\ \hat{\mathbf{r}}_{i+1}\ _2 / \nu_1$		2	2
$\rho_{i+1} = \mathbf{s}_1^T \hat{\mathbf{r}}_{i+1}$		2	
$\beta_{i+1} = \rho_{i+1} / \rho_i$			1
$\hat{\mathbf{q}}_{i+1} = \hat{\mathbf{r}}_{i+1} + \hat{\mathbf{p}}_{i+1} \beta_{i+1}$		2	
$\hat{\mathbf{u}}_{i+1} = \hat{\mathbf{q}}_{i+1} + (\hat{\mathbf{p}}_{i+1} + \hat{\mathbf{u}}_i \beta_{i+1}) \beta_{i+1}$		4	
$\hat{\mathbf{w}}_{i+1} = \mathbf{A}\hat{\mathbf{u}}_{i+1}$	2		
end for			

Flop count for a single iteration $i > 1$: $4N + 19n + 4$

Algorithm QMRCGS

QMRCGS	N	n	1
$\hat{\mathbf{r}}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$	2	1	
$\hat{\mathbf{q}}_1 = \mathbf{s}_1 = \mathbf{r}_1^{QMR} = \hat{\mathbf{r}}_1$			
$\nu_1 = \ \hat{\mathbf{r}}_1\ _2$		2	1
$\rho_1 = \mathbf{s}_1^T \hat{\mathbf{r}}_1$			2
$\hat{\mathbf{w}}_1 = \hat{\mathbf{z}}_1 = \mathbf{A}\hat{\mathbf{q}}_1$	2		
$OTinit(\frac{1}{2})$			
for $i = 1, 2, \dots$			
$\sigma_i = \mathbf{s}_1^T \hat{\mathbf{w}}_i$	2		
$\alpha_i = \rho_i / \sigma_i$		1	
$\hat{\mathbf{r}}_{i+\frac{1}{2}} = \hat{\mathbf{r}}_i - \hat{\mathbf{z}}_i \alpha_i$	2		
$\nu_{i+\frac{1}{2}} = \ \hat{\mathbf{r}}_{i+\frac{1}{2}}\ _2$	2	1	
$h_{ii} = \nu_i / \alpha_i$		1	
$h_{i+\frac{1}{2}, i} = -\nu_{i+\frac{1}{2}} / \alpha_i$		1	
$OTplus(i, \frac{1}{2}, \hat{\mathbf{q}}_i, \hat{\mathbf{z}}_i)$	8	12	
$\hat{\mathbf{p}}_{i+1} = \hat{\mathbf{q}}_i - \hat{\mathbf{w}}_i \alpha_i$	2		
$\hat{\mathbf{y}}_{i+1} = \mathbf{A}\hat{\mathbf{p}}_{i+1}$	2		
$\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{r}}_{i+\frac{1}{2}} - \hat{\mathbf{y}}_{i+1} \alpha_i$	2		
$\nu_{i+1} = \ \hat{\mathbf{r}}_{i+1}\ _2$	2	1	
$h_{i+\frac{1}{2}, i+\frac{1}{2}} = -h_{i+\frac{1}{2}, i}$			
$h_{i+1, i+\frac{1}{2}} = -\nu_{i+1} / \alpha_i$		1	
$OTplus(i + \frac{1}{2}, \frac{1}{2}, \hat{\mathbf{p}}_{i+1}, \hat{\mathbf{y}}_{i+1})$	8	12	
$\ \mathbf{r}_{i+1}^{QMR}\ _2 / \nu_1$	2	2	
$\rho_{i+1} = \mathbf{s}_1^T \hat{\mathbf{r}}_{i+1}$	2		
$\beta_{i+1} = \rho_{i+1} / \rho_i$		1	
$\hat{\mathbf{q}}_{i+1} = \hat{\mathbf{r}}_{i+1} + \hat{\mathbf{p}}_{i+1} \beta_{i+1}$	2		
$\hat{\mathbf{z}}_{i+1} = \mathbf{A}\hat{\mathbf{q}}_{i+1}$	2		
$\hat{\mathbf{w}}_{i+1} = \hat{\mathbf{z}}_{i+1} + (\hat{\mathbf{y}}_{i+1} + \hat{\mathbf{w}}_i \beta_{i+1}) \beta_{i+1}$	4		
end for			

Flop count for a single iteration $i > 1$: $4N + 38n + 33$

Algorithm BiCGStab

BiCGStab		$O(N)$	$O(n)$	$O(1)$
$\hat{\mathbf{r}}_1 = \mathbf{A}\hat{\mathbf{x}}_1 - \mathbf{b}$		2	1	
$\nu_1 = \ \hat{\mathbf{r}}_1\ _2$			2	1
$\hat{\mathbf{u}}_1 = \mathbf{s}_1 = \hat{\mathbf{r}}_1$				
$\rho_1 = \mathbf{s}_1^T \hat{\mathbf{r}}_1$			2	
$\hat{\mathbf{w}}_1 = \mathbf{A}\hat{\mathbf{u}}_1$		2		
for $i = 1, 2, \dots$				
$\sigma_i = \mathbf{s}_1^T \hat{\mathbf{w}}_i$		2		
$\alpha_i = \rho_i / \sigma_i$				1
$\hat{\mathbf{z}}_i = \hat{\mathbf{r}}_i - \hat{\mathbf{w}}_i \alpha_i$		2		
$\hat{\mathbf{q}}_i = \mathbf{A}\hat{\mathbf{z}}_i$		2		
$\omega_i = (\hat{\mathbf{z}}_i^T \hat{\mathbf{q}}_i) / (\hat{\mathbf{q}}_i^T \hat{\mathbf{q}}_i)$		4	1	
$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i - \hat{\mathbf{u}}_i \alpha_i - \hat{\mathbf{z}}_i \omega_i$		4		
$\hat{\mathbf{r}}_{i+1} = \hat{\mathbf{z}}_i - \hat{\mathbf{q}}_i \omega_i$		2		
$\ \hat{\mathbf{r}}_{i+1}\ _2 / \nu_1$		2	2	
$\rho_{i+1} = \mathbf{s}_1^T \hat{\mathbf{r}}_{i+1}$		2		
$\beta_{i+1} = (\alpha_i \rho_{i+1}) / (\omega_i \rho_i)$				3
$\hat{\mathbf{u}}_{i+1} = \hat{\mathbf{r}}_{i+1} + (\hat{\mathbf{u}}_i - \hat{\mathbf{w}}_i \omega_i) \beta_{i+1}$			4	
$\hat{\mathbf{w}}_{i+1} = \mathbf{A}\hat{\mathbf{u}}_{i+1}$		2		
end for				

Flop count for a single iteration $i > 1$: $4N + 22n + 7$

Algorithm QMRBiCGStab

QMRBiCGStab		$O(N)$	$O(n)$	$O(1)$
$\mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$				
$\widehat{\mathbf{u}}_1 = \widehat{\mathbf{r}}_1 = \mathbf{s}_1 = \mathbf{r}_1$		2	1	
$\nu_1 = \ \widehat{\mathbf{r}}_1\ _2$			2	1
$\rho_1 = \mathbf{s}_1^T \widehat{\mathbf{r}}_1$			2	
$\widehat{\mathbf{w}}_1 = \mathbf{A}\widehat{\mathbf{u}}_1$		2		
$OTinit(\frac{1}{2})$				
for $i = 1, 2, \dots$				
$\sigma_i = \mathbf{s}_1^T \widehat{\mathbf{w}}_i$		2		
$\alpha_i = \rho_i / \sigma_i$				1
$\widehat{\mathbf{z}}_i = \widehat{\mathbf{r}}_i - \widehat{\mathbf{w}}_i \alpha_i$		2		
$\mu_i = \ \widehat{\mathbf{z}}_i\ _2$		2	1	
$h_{ii} = \nu_i / \alpha_i$				1
$\tilde{h}_{i+\frac{1}{2}, i} = -\mu_i / \alpha_i$				1
$OTplus(i + \frac{1}{2}, i - \frac{1}{2}, i, \widehat{\mathbf{u}}_i, \widehat{\mathbf{w}}_i)$		8	12	
$\widehat{\mathbf{q}}_i = \mathbf{A}\widehat{\mathbf{z}}_i$		2		
$\omega_i = (\widehat{\mathbf{z}}_i^T \widehat{\mathbf{q}}_i) / (\widehat{\mathbf{q}}_i^T \widehat{\mathbf{q}}_i)$		4	1	
$\widehat{\mathbf{r}}_{i+1} = \widehat{\mathbf{z}}_i - \widehat{\mathbf{q}}_i \omega_i$		2		
$\nu_{i+1} = \ \widehat{\mathbf{r}}_{i+1}\ _2$		2	1	
$h_{i+\frac{1}{2}, i+\frac{1}{2}} = \mu_i / \omega_i$				1
$h_{i+1, i+\frac{1}{2}} = -\nu_{i+1} / \omega_i$				1
$OTplus(i + 1, i, i + \frac{1}{2}, \widehat{\mathbf{z}}_i, \widehat{\mathbf{q}}_i)$		8	12	
$\left\ \mathbf{r}_{i+1}^{QMR} \right\ _2 / \nu_1$		2	2	
$\rho_{i+1} = \mathbf{s}_1^T \widehat{\mathbf{r}}_{i+1}$		2		
$\beta_{i+1} = (\alpha_i \rho_{i+1}) / (\omega_i \rho_i)$				3
$\widehat{\mathbf{u}}_{i+1} = \widehat{\mathbf{r}}_{i+1} + (\widehat{\mathbf{u}}_i - \widehat{\mathbf{w}}_i \omega_i) \beta_{i+1}$		4		
$\widehat{\mathbf{w}}_{i+1} = \mathbf{A}\widehat{\mathbf{u}}_{i+1}$		2		
end for				

Flop count for a single iteration $i > 1$: $4N + 38n + 37$

Algorithm LSQR

LSQR	$O(N)$	$O(n)$	$O(1)$
$\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$	2	1	
$\nu = \nu_1 = \ \mathbf{r}_1\ _2$		2	1
$\tilde{\mathbf{s}}_1 = \mathbf{r}_1 / \nu_1$		1	
$OTinit(0)$			
$\mathbf{v}_0 = \mathbf{0}$ and $\tilde{h}_{10} = 0$			
for $i = 1, 2, \dots$			
$\mathbf{v}_i = \mathbf{A}^T \tilde{\mathbf{s}}_i - \tilde{\mathbf{v}}_{i-1} \tilde{h}_{i,i-1}$	2	2	
$\tilde{h}_{ii} = \ \mathbf{v}_i\ _2$		2	1
$\tilde{\mathbf{v}}_i = \mathbf{v}_i / \tilde{h}_{ii}$		1	
$\mathbf{s}_{i+1} = \mathbf{A} \tilde{\mathbf{v}}_i - \tilde{\mathbf{s}}_i \tilde{h}_{ii}$	2	2	
$\tilde{h}_{i+1,i} = \ \mathbf{s}_{i+1}\ _2$		2	1
$\tilde{\mathbf{s}}_{i+1} = \mathbf{s}_{i+1} / h_{i+1,i}$		1	
$OT(i+1, i-1, i)$			12
$\mathbf{z}_i = \tilde{\mathbf{v}}_i - \mathbf{z}_{i-1} \eta_{i-1}$		2	
$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{z}_i \gamma_i$		2	
$\nu = s_i \times \nu$			1
$\rho = \nu / v_1$			1
end for			

Flop count for a single iteration $i > 1$: $4N + 14n + 16$

Algorithm HG2

HG2		$O(N)$	$O(n)$	$O(1)$
$\mathbf{r}_1 = \mathbf{Ax}_1 - \mathbf{b}$		2	1	
$\mathbf{v}_1 = \mathbf{u}_1 = \mathbf{s}_1 = \mathbf{r}_1$				
$\sigma_1 = \rho_1 = \mathbf{r}_1^T \mathbf{r}_1$			2	
$\nu_1 = \sqrt{\rho_1}$				1
$\mathbf{w}_1 = \mathbf{Au}_1$			2	
for $i = 1, 2, \dots$				
	$\tau_i = \mathbf{v}_i^T \mathbf{w}_i$		2	
	$\alpha_i = \rho_i / \tau_i$			1
	$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \alpha_i$		2	
	$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \alpha_i$		2	
	$\rho_{i+1} = \mathbf{r}_{i+1}^T \mathbf{r}_{i+1}$		2	
	$\nu_{i+1} = \sqrt{\rho_{i+1}}$			1
	ν_{i+1} / ν_1			1
	$\mathbf{y}_i = \mathbf{A}^T \mathbf{v}_i$		2	
	$\beta_i = \sigma_i / \tau_i$			1
	$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \beta_i$		2	
	$\sigma_{i+1} = \mathbf{s}_{i+1}^T \mathbf{s}_{i+1}$		2	
	$\gamma_i = \sigma_{i+1} / \sigma_i$			1
	$\mathbf{u}_{i+1} = \mathbf{s}_{i+1} + \mathbf{u}_i \gamma_i$		2	
	$\mathbf{w}_{i+1} = \mathbf{A} \mathbf{u}_{i+1}$		2	
	$\delta_i = \nu_{i+1} / \nu_i$			1
	$\mathbf{v}_{i+1} = \mathbf{r}_{i+1} + \mathbf{v}_i \delta_i$		2	
end for				

Flop count for a single iteration $i > 1$: $4N + 16n + 6$

Algorithm BiCR

BiCR		$O(N)$	$O(n)$	$O(1)$
$\mathbf{y}_0 = \mathbf{0}$ and $\gamma_0 = 0$				
$\mathbf{u}_1 = \mathbf{s}_1 = \mathbf{r}_1 = \mathbf{A}\mathbf{x}_1 - \mathbf{b}$		2	1	
$\nu_1 = \ \mathbf{r}_1\ _2$			2	1
$\mathbf{q}_1 = \mathbf{A}^T \mathbf{r}_1$		2		
$\sigma_1 = \mathbf{q}_1^T \mathbf{s}_1$			2	
$\mathbf{w}_1 = \mathbf{A}\mathbf{u}_1$		2		
for $i = 1, 2, \dots$				
$\alpha_i = \sigma_i / (\mathbf{w}_i^T \mathbf{w}_i)$		2	1	
$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{u}_i \alpha_i$		2		
$\mathbf{r}_{i+1} = \mathbf{r}_i - \mathbf{w}_i \alpha_i$		2		
$\ \mathbf{r}_{i+1}\ _2 / \nu_1$		2	2	
$\mathbf{y}_i = \mathbf{q}_i + \mathbf{y}_{i-1} \gamma_{i-1}$		2		
$\beta_i = \sigma_i / (\mathbf{y}_i^T \mathbf{y}_i)$		2	1	
$\mathbf{s}_{i+1} = \mathbf{s}_i - \mathbf{y}_i \beta_i$		2		
$\mathbf{q}_{i+1} = \mathbf{A}^T \mathbf{r}_{i+1}$	2			
$\sigma_{i+1} = \mathbf{q}_{i+1}^T \mathbf{s}_{i+1}$		2		
$\gamma_i = \sigma_{i+1} / \sigma_i$			1	
$\mathbf{u}_{i+1} = \mathbf{s}_{i+1} + \mathbf{u}_i \gamma_i$		2		
$\mathbf{w}_{i+1} = \mathbf{A}\mathbf{u}_{i+1}$	2			
end for				

Flop count for a single iteration $i > 1$: $4N + 18n + 5$

This page intentionally left blank

Appendix G

Guide to the graphs

The graphs illustrating aspects of the performances of the algorithms discussed in the text are to be found in the accompanying CD-ROM. They are stored as encapsulated postscript (*.eps) files. To view these files on Windows, obtain Ghostscript and GSview from <<http://www.cs.wisc.edu/~ghost/>>.

The graphs are grouped as follows:

Graphs	Descriptions
1 - 12a	Comparisons of MATLAB and "in house" versions of GMRes(m) and QMRBiCG
13 - 24	Comparisons of Lanczos and HS versions of BiCG and BiCR
25 - 28a	Various discrepancy comparisons
29 - 34	Comparisons of three versions of QMR
35 - 58	Effects of scaling
59 - 64	Comparisons of BiCG, QMRBiCG, HG and BiCR
65 - 70	Comparisons of BiCG, CGS, QMRCGS, BiCGStab and QMRBiCGStab
71 - 76	Comparisons of BiCG and GMRes(m), $m = 10, 20$ and 30
77 - 88	Comparisons of BiCG, QMRBiCG, GMRes(20) and BiCGStab.
89 - 95	Failures solving e05r0500 (in this and the next five groups, only results of the "in house" versions are given)
96 - 104	Failures solving mahindas
105 - 108	Failures solving orsirr1
109 - 110	Failures solving orsreg1
111 - 119	Failures solving shl400
120 - 122a	Failures solving watt2
123 - 125	Chebychev polynomials

In the graphs the following abbreviations are used:

RRR	relative recursive residuals
RTR	relative true residuals
diff	discrepancy as defined by equation (9.41)

References

- [1] J. Abaffy and E. Spedicato. *ABS Projection Algorithms*. Ellis Horwood, Chichester, 1989.
- [2] R. L. Adler and T. J. Rivlin. Ergodic and mixing properties of Chebyshev polynomials. *Proc. Amer. Math. Soc.*, 15:794–796, 1964.
- [3] M. A. Ajiz and A. Jennings. A robust incomplete Choleski-conjugate gradient algorithm. *Internat. J. Numer. Methods Engrg.*, 20:949–966, 1984.
- [4] F. Alvarado and H. Dağ. Sparsified and incomplete sparse factored inverse preconditioners. In *Proceedings of the 1992 Copper Mountain Conference on Iterative Methods, Vol. I (April 9–14, 1992)*, 1992.
- [5] M. Arioli, V. Pták, and Z. Strakoš. Krylov sequences of maximal length and convergence of GMRES. *BIT*, 38:636–643, 1998.
- [6] W. E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.
- [7] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal.*, 27:1542–1568, 1990.
- [8] C. Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM J. Sci. Comput.*, 16:1404–1411, 1995.
- [9] O. Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT*, 25:166–187, 1985.
- [10] O. Axelsson. A general incomplete block-matrix factorization method. *Linear Algebra Appl.*, 74:179–190, 1986.
- [11] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, Cambridge, 1994.
- [12] O. Axelsson, S. Brinkkemper, and V. P. Il'in. On some versions of incomplete block matrix factorization iterative methods. *Linear Algebra Appl.*, 58:3–15, 1984.
- [13] O. Axelsson and V. Eijkhout. Vectorizable preconditioners for elliptic difference equations in three space dimensions. *J. Comput. Appl. Math.*, 27:299–321, 1991.
- [14] O. Axelsson and L. Yu. Kolotilina. Diagonally compensated reduction and related preconditioning methods. *Numer. Linear Algebra Appl.*, 1:155–177, 1994.
- [15] O. Axelsson and B. Polman. On approximate factorization methods for block matrices suitable for vector and parallel processors. *Linear Algebra Appl.*, 77:3–26, 1986.
- [16] O. Axelsson and P. S. Vassilevski. Algebraic multilevel preconditioning methods. I. *Numer. Math.*, 56:157–177, 1989.
- [17] O. Axelsson and P. S. Vassilevski. Algebraic multilevel preconditioning methods, II. *SIAM J. Numer. Anal.*, 27:1569–1590, 1990.
- [18] S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang. *PETSc User's Manual*. Argonne National Laboratory, Argonne, IL, 2002.
- [19] M. W. Benson. Iterative solution of large scale linear systems. Master's thesis, Lakehead University, Thunder Bay, Ontario, 1973.
- [20] M. W. Benson and P. O. Frederickson. Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems. *Util. Math.*, 22:127–140, 1982.
- [21] M. W. Benson, J. Krettmann, and M. Wright. Parallel algorithms for the solution of certain large sparse linear systems. *Int. J. Comput. Math.*, 16:245–260, 1984.

- [22] M. Benzi. *A Direct Row-Projection Method for Sparse Linear Systems*. PhD thesis, Department of Mathematics, North Carolina State University, Raleigh, (NC), 1993.
- [23] M. Benzi. Preconditioning techniques for large linear systems: a survey. *J. Comput. Phys.*, 182:418–477, 2002.
- [24] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22:1318–1332, 2000.
- [25] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comput.*, 22:1333–1353, 2000.
- [26] M. Benzi, R. Kouhia, and M. Tůma. Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics. *Computer Methods in Applied Mechanics and Engineering*, 190:6533–6554, 2001.
- [27] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17:1135–1149, 1996.
- [28] M. Benzi, D. B. Szyld, and A. van Duin. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM J. Sci. Comput.*, 20:1652–1670, 1999.
- [29] M. Benzi and M. Tůma. A comparison of some preconditioning techniques for general sparse matrices. In P. Vassilevski and S. Margenov, editors, *Iterative Methods in Linear Algebra II*, pages 191–203. IMACS Series in Computational and Applied Mathematics, Piscataway, NJ, 1996.
- [30] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19:968–994, 1998.
- [31] M. Benzi and M. Tůma. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math.*, 30:305–340, 1999.
- [32] M. Benzi and M. Tůma. A robust incomplete factorization preconditioner for positive definite matrices. *Numer. Linear Algebra Appl.*, 10:385–400, 2003.
- [33] Å. Björck. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra Appl.*, pages 297–316, 1994.
- [34] J. H. Bramble. *Multigrid Methods*. Longman Scientific & Technical, Harlow, UK, 1993.
- [35] C. Brezinski. *Projection Methods for Systems of Equations*. Studies in Computational Mathematics 7. North Holland, 1997.
- [36] C. Brezinski, M. R. Zaglia, and H. Sadok. Avoiding breakdown and near-breakdown in Lanczos type algorithms. *Numer. Algorithms*, 1:261–284, 1991.
- [37] C. Brezinski, M. R. Zaglia, and H. Sadok. Addendum to “Avoiding breakdown and near-breakdown in Lanczos type algorithms”. *Numer. Algorithms*, 2:133–136, 1992.
- [38] C. Brezinski, M. R. Zaglia, and H. Sadok. A breakdown-free Lanczos-type algorithm for solving linear systems. *Numer. Math.*, 63:29–38, 1992.
- [39] C. Brezinski, M. R. Zaglia, and H. Sadok. New look-ahead Lanczos-type algorithms for linear systems. *Numer. Math.*, 83:53–85, 1999.
- [40] C. Brezinski, M. R. Zaglia, and H. Sadok. A review of formal orthogonality in Lanczos-based methods. *J. Computational and Applied Math.*, 140:81–98, 2002.
- [41] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial. Second Edition*. SIAM, Philadelphia, PA, 2000.
- [42] P. N. Brown. A theoretical comparison of the Arnoldi and GMRES algorithms. *SIAM J. Sci. Stat. Comput.*, 12:58–78, 1991.
- [43] C. G. Broyden. Some generalisations of the theory of successive over-relaxation. *Numer. Math.*, pages 269–284, 1964.
- [44] C. G. Broyden. Some condition-number bounds for the Gaussian elimination process. *IMA J. Appl. Math.*, 12:273–286, 1973.
- [45] C. G. Broyden. A note on the block conjugate gradient algorithm of O’Leary. *Optim. Methods Softw.*, 5:347–350, 1995.
- [46] C. G. Broyden. A breakdown of the block-CG method. *Optim. Methods Softw.*, 7:41–55, 1996.
- [47] C. G. Broyden. A new taxonomy of conjugate gradient methods. *Comput. Math. Appl.*, 31(4/5):7–17, 1996.

- [48] C. G. Broyden. The Gram-Schmidt method - a hierarchy of algorithms. In A. Sydow, editor, *15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Volume 2, Numerical Mathematics*, pages 545–550, Berlin, 1997. Wissenschaft und Technik Verlag.
- [49] C. G. Broyden. Look-ahead block-CG algorithms. In G. W. Althaus and E. Spedicato, editors, *Algorithms for Large Scale Linear Algebraic Systems*, pages 197–215. NATO Advanced Study Institute, Kluwer Academic Publishers, 1998.
- [50] C. G. Broyden and M. A. Boschetti. A comparison of three basic conjugate direction methods. *Numer. Linear Algebra Appl.*, 3(6):473–489, 1996.
- [51] C. G. Broyden and P. Foschi. Duality in conjugate-gradient methods. *Numer. Algorithms*, 2(2):113–128, 1999.
- [52] C. G. Broyden and M.-T. Vespucci. On the convergence of Krylov linear equation solvers. *Optim. Methods Softw.*, 16:113–129, 2001.
- [53] N. I. Buleev. A numerical method for solving two- and three-dimensional diffusion equations. *Sb. Math.*, 51:227–238, 1960.
- [54] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 8:639–655, 1971.
- [55] S. L. Campbell, I. C. F. Ipsen, C. T. Kelley, and C. D. Meyer. GMRES and the minimal polynomial. *BIT*, 36(4):664–675, 1996.
- [56] L. Cesari. Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive. *Atti Accad. Naz. Lincei Cl. Sci. Fis. Mat. Natur. Rend. Lincei*, 25:422–428, 1937.
- [57] T. F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, and C. H. Tong. QMRCGSTAB: A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems. *SIAM J. Sci. Comput.*, 15(2):338–347, 1994.
- [58] T. F. Chan, C.-C. J. Kuo, and C. Tong. Parallel elliptic preconditioners: Fourier analysis and performance on the Connection Machine. *Comput. Phys. Comm.*, 53:237–252, 1989.
- [59] T. F. Chan, L. D. Pillis, and H. A. van der Vorst. A transpose-free squared Lanczos algorithm and application to solving nonsymmetric linear systems. UCLA technical report CAM 91-17, University of California, Los Angeles, October 1991.
- [60] R. Chandra. *Conjugate gradient methods for partial differential equations*. PhD thesis, Yale University, 1978. Res. Rep. 129.
- [61] R. Chandra, S. C. Eisenstadt, and M. H. Schultz. The modified conjugate residual method for partial differential equations. In R. Vichnevetsky, editor, *Advances in Computer Methods for Partial Differential Equations*, pages 13–19. IMACS, 1977.
- [62] M. P. Chernesky. On the preconditioned Krylov subspace methods for discrete convection-diffusion problems. *Numer. Methods Partial Differential Equations*, 13:321–330, 1997.
- [63] H. Choi and D. B. Szyld. Application of threshold partitioning of sparse matrices to Markov chains. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium, IPDS'96*, pages 158–165, Los Alamitos, CA, 1996. IEEE Computer Soc. Press.
- [64] E. Chow. *Robust Preconditioning for Sparse Linear Systems*. PhD thesis, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1997.
- [65] E. Chow and M. A. Heroux. An object-oriented framework for block preconditioning. *ACM Trans. Math. Software*, 24:159–183, 1998.
- [66] E. Chow and Y. Saad. Approximate inverse preconditioners for general sparse matrices. Technical Report UMSI 94-101, University of Minnesota Supercomputer Institute, Minneapolis, MN, 1994.
- [67] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86:387–414, 1997.
- [68] E. Chow and Y. Saad. ILUS: An incomplete LU preconditioner in sparse skyline format. *Internat. J. Numer. Methods Fluids*, 25:739–748, 1997.
- [69] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.*, 19:995–1023, 1998.
- [70] A. T. Chronopoulos. s-step iterative methods for (non)symmetric (in)definite linear systems. *SIAM J. Numer. Anal.*, 28(6):1776–1789, December 1991.

- [71] A. T. Chronopoulos and C. W. Gear. s-step iterative methods for symmetric linear systems. *J. Comput. Appl. Math.*, 25:153–168, 1989.
- [72] M. T. Chu, R. E. Funderlic, and G. H. Golub. A rank-one reduction formula and its applications to matrix factorizations. *SIAM Rev.*, 37:512–530, 1995.
- [73] S. S. Clift and W.-P. Tang. Weighted graph based ordering techniques for preconditioned conjugate gradient methods. *BIT*, 35:30–47, 1995.
- [74] L. Collatz. Aufgaben monotoner Art. *Arch. Math.*, 3:366–376, 1952.
- [75] P. Concus and G. E. Golub. A generalized conjugate gradient method for nonsymmetric systems of linear equations. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences and Engineering, Lecture Notes in Economics and Mathematical Systems, Vol. 134*, pages 56–65. Springer-Verlag, Berlin, 1976.
- [76] P. Concus, G. H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Stat. Comput.*, 6:309–332, 1985.
- [77] R. Cook. A reformulation of preconditioned conjugate gradients suitable for a local memory multiprocessor. In R. Beauwens and P. d. Groen, editors, *Iterative Methods in Linear Algebra*, pages 313–322. IMACS, North-Holland, Amsterdam, 1992.
- [78] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank. Approximate inverse preconditioning for sparse linear systems. *Int. J. Comput. Math.*, 44:91–110, 1992.
- [79] E. J. Craig. The n-step iteration procedure. *J. Math. Phys.*, 34:64–73, 1955.
- [80] J. Cullum and A. Greenbaum. Relations between Galerkin and norm-minimizing methods for solving linear systems. *SIAM J. Matrix Anal. Appl.*, 17(2):223–247, 1996.
- [81] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th National Conference of the ACM*, pages 157–172, New Jersey, 1969. Brandon Press.
- [82] W. Dahmen and L. Elsner. Algebraic multigrid methods and the Schur complement. In *Robust MultiGrid Methods (Kiel 1988)*, volume 23 of *Notes in Numerical Fluid Mechanics*, pages 58–68. Vieweg, Braunschweig, 1989.
- [83] H. Daḡ. *Iterative Methods and Parallel Computation for Power Systems*. PhD thesis, Department of Electrical Engineering, University of Wisconsin, Madison, WI, 1995.
- [84] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. An automatic ordering method for incomplete factorization iterative solvers. In *Proceedings of the 1991 SPE Reservoir Simulation Symposium, Anaheim, CA*, Richardson, TX, 1991. Society of Petroleum Engineers.
- [85] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM J. Matrix Anal. Appl.*, 13:944–961, 1992.
- [86] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Towards a cost effective ILU preconditioner with high level fill. *BIT*, 32:442–463, 1992.
- [87] S. Demko, W. F. Moss, and P. W. Smith. Decay rates for inverses of band matrices. *Math. Comp.*, 43:491–499, 1984.
- [88] J. E. Dendy. Black box multigrid. *J. Comput. Phys.*, 48:366–386, 1980.
- [89] J. E. Dennis and K. Turner. Generalized conjugate directions. *Linear Algebra Appl.*, 88–89:187–209, 1987.
- [90] S. Doi. On parallelism and convergence of incomplete LU factorizations. *Appl. Numer. Math.*, 7:417–436, 1991.
- [91] S. Doi and A. Lichnewsky. A graph-theory approach for analyzing the effects of ordering on ILU preconditioning. Technical Report 1452, INRIA, 1991.
- [92] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue. Approximating the inverse of a matrix for use in iterative algorithms on vector processors. *Computing*, 22:257–268, 1979.
- [93] I. S. Duff, A. M. Erisman, C. W. Gear, and J. K. Reid. Sparsity structure and Gaussian elimination. *SIGNUM Newsletter*, 23:2–9, 1988.
- [94] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15:1–14, 1989.

- [95] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20:889–901, 1999.
- [96] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22:973–996, 2001.
- [97] I. S. Duff and G. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989.
- [98] L. C. Dutto. The effect of ordering on preconditioned GMRES algorithms for solving the compressible Navier-Stokes equations. *Internat. J. Numer. Methods Engrg.*, 36:457–497, 1993.
- [99] E. Egervary. On rank-diminishing operations and their applications to the solution of linear equations. *Z. Angew. Math. Phys.*, 11:376–386, 1960.
- [100] V. Eijkhout. Analysis of parallel incomplete point factorizations. *Linear Algebra Appl.*, 154/156:723–740, 1991.
- [101] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 20:345–357, 1983.
- [102] T. S. Eliot. The naming of cats. In *Old Possum's Book of Practical Cats*. Faber and Faber Limited, London WC1N 3AU, 1939.
- [103] H. C. Elman. *Iterative Methods for Large, Sparse, Nonsymmetric Systems of Linear Equations*. PhD thesis, Yale University, 1982.
- [104] H. C. Elman. A stability analysis of incomplete LU factorizations. *Math. Comp.*, 47:191–217, 1986.
- [105] H. C. Elman and E. Agron. Ordering techniques for the preconditioned conjugate gradient method on parallel computers. *Comput. Phys. Comm.*, 53:253–269, 1989.
- [106] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21:352–362, 1984.
- [107] B. Fischer. *Polynomial Based Iteration Methods for Symmetric Linear Systems*. John Wiley and B. G. Teubner, New York, Stuttgart, 1996.
- [108] B. Fischer and R. W. Freund. On adaptive weighted polynomial preconditioning for Hermitian positive definite matrices. *SIAM J. Sci. Comput.*, 15(2):408–426, 1994.
- [109] D. A. Flanders and G. Shortly. Numerical determination of fundamental modes. *J. Appl. Phys.*, 21:1326–1332, 1950.
- [110] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. A. Watson, editor, *Numerical Analysis - Dundee 1975*, pages 73–89, Berlin-Heidelberg, 1976. Springer. Lecture Notes in Mathematics, volume 506.
- [111] G. E. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Englewood Cliffs, N.J., 1967.
- [112] L. Fox, H. D. Huskey, and J. H. Wilkinson. Notes on the solution of algebraic linear simultaneous equations. *Quart. J. Mech. Appl. Math.*, 1:149–173, 1948.
- [113] S. Frankel. Convergence rates of iterative treatments of partial differential equations. *MTAC*, 4:65–75, 1950.
- [114] P. O. Frederickson. Fast approximate inversion of large sparse linear systems. Technical Report 7, Lakehead University, Thunder Bay, Ontario, 1975.
- [115] R. W. Freund. On conjugate gradient type methods and polynomial preconditioners for a class of complex non-Hermitian matrices. *Numer. Math.*, 57:285–312, 1990.
- [116] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14(2):470–482, March 1993.
- [117] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput.*, 14(1):137–158, 1993.
- [118] R. W. Freund and N. M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.
- [119] R. W. Freund and N. M. Nachtigal. An implementation of the QMR method based on coupled two-term recurrences. *SIAM J. Sci. Comput.*, 15:313–337, 1994.
- [120] R. W. Freund and N. M. Nachtigal. Software for simplified Lanczos and QMR algorithms. *Applied Num. Math.*, 19:319–341, 1995.

- [121] R. W. Freund and T. Szeto. A transpose-free quasi-minimal residual squared algorithm for non-Hermitian linear systems. In R. Vichnevetsky, D. Knight, and G. Richter, editors, *Advances in Computer Methods for Differential Equations - VII*, pages 258–264. IMACS, 1992.
- [122] R. W. Freund and H. Zha. Simplifications of the nonsymmetric Lanczos process and a new algorithm for Hermitian linear systems, Numerical Analysis Manuscript. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1995.
- [123] V. M. Fridman. The method of minimum iterations with minimum errors for a system of linear algebraic equations with a symmetric matrix. *USSR Comput. Math. and Math. Phys.*, 2:362–363, 1963.
- [124] G. Frobenius. Über Matrizen aus nicht negativen Elementen. *S.-B. Preuss. Akad. Wiss.*, pages 456–477, 1912.
- [125] F. R. Gantmacher. *The Theory of Matrices*, volume 1. Chelsea, New York, 1977.
- [126] C. W. Gear and Y. Saad. Iterative solution of linear equations in ODE codes. *SIAM J. Sci. Stat. Comput.*, 4:583–601, 1983.
- [127] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [128] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numer. Anal. Ser. B*, 2(2):205–244, 1965.
- [129] G. H. Golub and C. F. van Loan. *Matrix Computations. Third Edition*. Johns Hopkins University Press, Baltimore and London, 1996.
- [130] N. I. M. Gould and J. A. Scott. Sparse approximate-inverse preconditioners using norm-minimization techniques. *SIAM J. Sci. Comp.*, 19:605–625, 1998.
- [131] J. P. Gram. Über die Entwicklung reeller Funktionen in Reihen mittelst der Metode der kleinsten Quadrate. *J. Reine Angew. Math.*, 94:41–73, 1883. First published in Danish as the author's doctoral dissertation.
- [132] P. R. Graves-Morris and A. Salam. Avoiding breakdown in Van der Vorst's method. *Numer. Algorithms*, 21:205–223, 1999.
- [133] A. Greenbaum. Estimating the attainable accuracy of recursively computed residual methods. *SIAM J. Matrix Anal. Appl.*, 18:535–551, 1997.
- [134] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia, 1997.
- [135] A. Greenbaum, V. Pták, and Z. Strakoš. Any nonincreasing convergence curve is possible for GMRES. *SIAM J. Matrix Anal. Appl.*, 17(3):465–469, July 1996.
- [136] A. Greenbaum and Z. Strakoš. Matrices that generate the same Krylov varieties. In G. Golub, editor, *Recent Advances in Iterative Methods*, pages 95–119. Springer, Berlin, 1994. IMA Volumes in Maths and its Applications.
- [137] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18:838–853, 1997.
- [138] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.
- [139] I. Gustafsson and G. Lindsberg. Completely parallelizable preconditioning methods. *Numer. Linear Algebra Appl.*, 2:447–465, 1995.
- [140] M. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, part 1. *SIAM J. Matrix Anal. Appl.*, 13:594–639, 1992.
- [141] M. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, part 2. *SIAM J. Matrix Anal. Appl.*, 15:15–58, 1994.
- [142] M. H. Gutknecht and M. Rosložnik. By how much can residual minimization accelerate the convergence of orthogonal residual methods? *Numer. Algorithms*, 27:189–213, 2001.
- [143] M. H. Gutknecht and M. Rosložnik. Residual smoothing techniques: do they improve the limiting accuracy of iterative solvers? *BIT*, 41(1):86–114, 2001.
- [144] M. H. Gutknecht and Z. Strakoš. Accuracy of two three-term and three two-term recurrences for Krylov space solvers. *SIAM J. Matrix Anal. Appl.*, 22(1):213–229, 2000.
- [145] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer-Verlag, Berlin, 1985.
- [146] W. Hackbusch. A parallel conjugate gradient method. *J. Num. Lin. Alg. Applics.*, 1(2):133–147, 1992.

- [147] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer-Verlag, New York, 1994.
- [148] C. J. Hegedüs. Private communication.
- [149] C. J. Hegedüs. Generating conjugate directions for arbitrary matrices by matrix equations. Technical Report KFKI-1990-36/M, Hungarian Academy of Sciences, Central Research Institute for Physics, Budapest, 1990.
- [150] C. J. Hegedüs. Generating conjugate directions for arbitrary matrices by matrix equations - I. *Comput. Math. Appl.*, 21(1):71–85, 1991.
- [151] C. J. Hegedüs. Generating conjugate directions for arbitrary matrices by matrix equations - II. *Comput. Math. Appl.*, 21(1):87–94, 1991.
- [152] C. J. Hegedüs. Generation of conjugate directions for arbitrary matrices and solution of linear systems. In E. Spedicato and M. T. Vespucci, editors, *Computer Algorithms for Solving Linear Algebraic Equations: The State of the Art*. University of Bergamo, Bergamo, 1991. NATO Advanced Study Institute, contributed papers, University of Bergamo research report.
- [153] J. V. Heist, J. Jacobs, and J. Scherders. Kleinste-kwadraten problemen. Technical report, Dep. Mathematics Rep., Eindhoven University of Technology, Eindhoven, The Netherlands, August 1976.
- [154] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bureau of Standards*, 49:409–436, 1952.
- [155] R. A. Horn and C. A. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.
- [156] A. S. Householder. *The Theory of Matrices in Numerical Analysis*. Blaisdell Publishing Company, New York, 1964.
- [157] D. Hysom. *New Sequential and Scalable Parallel Algorithms for Incomplete Factor Preconditioning*. PhD thesis, Department of Computer Science, Old Dominion University, Norfolk, VA, 2001.
- [158] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM J. Sci. Comput.*, 22:2194–2215, 2001.
- [159] IBM spokesperson. Private communication, e-mail, 6th October, 2001.
- [160] A. Jennings and M. A. Ajiz. Incomplete methods for solving $ATAx = b$. *SIAM J. Sci. Stat. Comput.*, 5:978–987, 1984.
- [161] A. Jennings and G. M. Malik. Partial elimination. *IMA J. Appl. Math.*, 20:307–316, 1977.
- [162] O. G. Johnson, C. A. Micchelli, and G. Paul. Polynomial preconditioners for conjugate gradient calculations. Technical Report RC-9208, IBM T. J. Watson Research Center, Yorktown Heights, N.Y., January 1982.
- [163] O. G. Johnson, C. A. Micchelli, and G. Paul. Polynomial preconditioning for conjugate gradient calculations. *SIAM J. Numer. Anal.*, 20:362–376, 1983.
- [164] P. Joly. Présentation de synthèse des méthodes de gradient conjugué. *Mathematical Modelling and Numerical Analysis*, 20(4):639–665, 1986.
- [165] M. T. Jones and P. E. Plassmann. Algorithm 740: Fortran subroutines to compute improved incomplete Cholesky factorization. *ACM Trans. Math. Software*, 21:18, 1995.
- [166] M. T. Jones and P. E. Plassmann. An improved incomplete Cholesky factorization. *ACM Trans. Math. Software*, 21:5–17, 1995.
- [167] I. E. Kaporin. High quality preconditioning of a general symmetric positive definite matrix based on its $U^T U + U^T R + R^T U$ decomposition. *Numer. Linear Algebra Appl.*, 5:483–509, 1998.
- [168] S. Karlin and L. S. Shapley. Geometry of moment spaces. Memoirs of the American Mathematical Society, 12, Amer. Math. Soc., 1953.
- [169] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, 1998.
- [170] D. S. Kershaw. The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations. *J. Comput. Phys.*, 26:43–65, 1978.
- [171] S. A. Kharchenko, L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin. A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form. *Numer. Linear Algebra Appl.*, 8:165–179, 2001.

- [172] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditioning I. Theory. *SIAM J. Matrix Anal. Appl.*, 14:45–58, 1993.
- [173] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditioning II. Solution of 3D FE systems on massively parallel computers. *Int. J. High Speed Comput.*, 7:191–215, 1995.
- [174] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, 45:255–282, 1950.
- [175] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bureau of Standards*, 49:33–53, 1952.
- [176] C. Lanczos. Chebyshev polynomials in the solution of large-scale linear systems. In *Proceedings of the Association of Computing Machinery, Toronto*, pages 124–133, Washington, D.C., 1953. Sauls Lithograph Co.
- [177] H. P. Langtangen. Conjugate gradient methods and ILU preconditioning of non-symmetric matrix systems with arbitrary sparsity patterns. *Internat. J. Numer. Methods Fluids*, 9:213–233, 1989.
- [178] C.-J. Lin and J. J. Moré. Incomplete Cholesky factorizations with limited memory. *SIAM J. Sci. Comput.*, 21:24–45, 1999.
- [179] J. W. H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.
- [180] D. G. Luenberger. Hyperbolic pairs in the method of conjugate gradients. *SIAM J. Appl. Math.*, 17:1263–1267, 1969.
- [181] M. Magolu monga Made and H. A. van der Vorst. Parallel incomplete factorizations with pseudo-overlapping subdomains. *Parallel Comput.*, 27:989–1008, 2001.
- [182] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34:473–497, 1980.
- [183] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31:148–162, 1977.
- [184] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT*, 24:623–633, 1984.
- [185] G. Meurant. *Computer Solution of Large Linear Systems*. North-Holland, Amsterdam, 1999.
- [186] J. Morris. A successive approximation process for solving simultaneous linear equations. *A. R. C. R. and M. No. 1711*, 1936. Report published by the Aeronautical Research Council, UK.
- [187] J. Morris. An escalator process for the solution of linear simultaneous equations. *Philos. Mag.*, VII Ser. 37:106–120, 1946.
- [188] J. Morris and J. W. Head. Lagrangian frequency equations, an “escalator” method for numerical solution. *Aircraft Engin.*, 14:312–316, 1942.
- [189] J. Morris and J. W. Head. The “escalator” process for the solution of Lagrangian frequency equations. *Philos. Mag.*, VII Ser. 35:735–759, 1944.
- [190] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM Trans. Math. Software*, 6:206–219, 1980.
- [191] A. A. Nikishin and A. Y. Yeremin. Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, 1: General iterative scheme. *SIAM J. Matrix Anal. Appl.*, 16(4):1135–1153, October 1995.
- [192] Y. Notay. Ordering methods for approximate factorization preconditioning. Technical Report IT/IF 14-11, Université Libre de Bruxelles, 1993.
- [193] D. P. O’Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra Appl.*, 29:293–322, 1980.
- [194] T. A. Oliphant. An implicit numerical method for solving two-dimensional time-dependent diffusion problems. *Quart. Appl. Math.*, 19:221–229, 1961.
- [195] T. A. Oliphant. An extrapolation process for solving linear systems. *Quart. Appl. Math.*, 20:257–265, 1962.
- [196] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra Appl.*, 240:131–151, 1996.

- [197] J. O'Neil and D. B. Szyld. A block ordering method for sparse matrices. *SIAM J. Sci. Stat. Comput.*, 11:811–823, 1990.
- [198] J. M. Ortega. *Matrix Theory - A second course*. Plenum Press, New York, 1987.
- [199] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press, New York, 1988.
- [200] J. M. Ortega. Orderings for conjugate gradient preconditionings. *SIAM J. Optim.*, 1:565–582, 1991.
- [201] C. C. Paige. Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *J. Inst. Math. Appl.*, 18:341–349, 1976.
- [202] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.
- [203] C. C. Paige and M. A. Saunders. An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 6(1):43–71, 1982.
- [204] B. N. Parlett. Reduction to tridiagonal form and minimal realizations. *SIAM J. Matrix Anal. Appl.*, 13:567–593, 1992.
- [205] B. N. Parlett and H. C. Chen. Use of indefinite pencils for computing damped natural modes. *Linear Algebra Appl.*, 140:53–88, 1990.
- [206] O. Perron. Zur Theorie der Matrizen. *Math. Ann.*, 64:248–263, 1907.
- [207] E. L. Poole and J. M. Ortega. Multicolor ICCG methods for vector computers. *SIAM J. Numer. Anal.*, 24:1394–1418, 1987.
- [208] J. K. Reid. On the method of conjugate gradients for the solution of large sparse linear equations. In J. K. Reid, editor, *Large Sparse Sets of Linear Equations*, pages 231–254. Academic Press, New York, 1971.
- [209] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with application to the stress in a masonry dam. *Philos. Trans. Roy. Soc. London, Ser. A*, pages 307–357, 1910.
- [210] Y. Robert. Regular incomplete factorizations of real positive definite matrices. *Linear Algebra Appl.*, 48:105–117, 1982.
- [211] J. W. Ruge and K. Stüben. Algebraic multigrid. In S. F. McCormick, editor, *Multigrid Methods*, pages 73–130. SIAM, Philadelphia, PA, 1987.
- [212] A. Ruhe. Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices. *Math. Comp.*, 33(146):680–687, 1979.
- [213] H. Rutishauser. Theory of gradient methods. In M. Engeli, T. Ginsburg, H. Rutishauser, and E. Stiefel, editors, *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems*, pages 24–49. Birkhäuser, 1959.
- [214] Y. Saad. Preconditioning techniques for nonsymmetric and indefinite linear systems. *J. Comput. Appl. Math.*, 24:89–105, 1988.
- [215] Y. Saad. ILUT: A dual threshold incomplete LU factorization. *Numer. Linear Algebra Appl.*, 1:387–402, 1994.
- [216] Y. Saad. Preconditioned Krylov subspace methods for CFD applications. In W. Habashi, editor, *Solution Techniques for Large-Scale CFD Problems*, pages 139–158. Wiley, New York, 1995.
- [217] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, Mass., 1996.
- [218] Y. Saad. Finding exact and approximate block structures for ILU preconditioning. *SIAM J. Sci. Comput.*, 24:1107–1123, 2003.
- [219] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, July 1986.
- [220] E. Schmidt. *Math. Ann.*, 63:433–476, 1907.
- [221] W. Schönauer. *Scientific Computing on Vector Computers*. North-Holland, Amsterdam, New York, Oxford, Tokyo, 1987.
- [222] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10:36–52, 1989.
- [223] R. V. Southwell. *Relaxation methods in engineering science*. Oxford Clarendon Press, 1940.

- [224] G. W. Stewart. Conjugate direction methods for solving systems of linear equations. *Numer. Math.*, 21:285–297, 1973.
- [225] E. L. Stiefel. Relaxationsmethoden bester Strategie zur Lösung linearer Gleichungssysteme. *Comment. Math. Helv.*, 29:157–179, 1955.
- [226] J. Stoer. Solution of large linear systems of equations by conjugate gradient type methods. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming - The State of the Art*, pages 540–565, Berlin, 1983. Springer.
- [227] J. Stoer and R. W. Freund. On the solution of large indefinite systems of linear equations by conjugate gradient algorithms. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences and Engineering V*, pages 35–53, Amsterdam, 1982. North-Holland.
- [228] H. L. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM J. Numer. Anal.*, 5:530–558, 1968.
- [229] S. A. Stotland and J. M. Ortega. Orderings for parallel conjugate gradient preconditioners. *SIAM J. Sci. Stat. Comput.*, 18:854–868, 1997.
- [230] M. Suarjana and K. H. Law. A robust incomplete factorization based on value and space constraints. *Internat. J. Numer. Methods Engrg.*, 38:1703–1719, 1995.
- [231] R. Suda. Large scale circuit analysis by preconditioned relaxation methods. In *Proceedings of PCG'94*, pages 189–205. Keio University, Japan, 1994.
- [232] R. Suda. *New Iterative Linear Solvers for Parallel Circuit Simulation*. PhD thesis, Department of Information Sciences, University of Tokio, 1996.
- [233] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE*, 55:1801–1809, 1967.
- [234] M. Tismenetsky. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra Appl.*, 154–156:331–353, 1991.
- [235] U. Trottenberg, C. W. Osterlee, and A. Schüller. *Multigrid*. Academic Press, London, 2000.
- [236] R. R. Underwood. An approximate factorization procedure based on the block Cholesky decomposition and its use with the conjugate gradient method. Technical Report NEDO-11386, General Electric Co., Nuclear Energy Division, San José, CA, 1976.
- [237] A. van der Ploeg, E. F. F. Botta, and F. W. Wubs. Nested grids ILU-decomposition (NGILU). *J. Comput. Appl. Math.*, 66:515–526, 1996.
- [238] H. A. van der Vorst. Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems. *J. Comput. Phys.*, 44:1–19, 1981.
- [239] H. A. van der Vorst. A vectorizable variant of some ICCG methods. *SIAM J. Sci. Stat. Comput.*, 3:350–356, 1982.
- [240] H. A. van der Vorst. Large tridiagonal and block tridiagonal linear systems on vector and parallel computers. *Parallel Comput.*, 5:45–54, 1987.
- [241] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly-convergent variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.
- [242] H. A. van der Vorst and G. L. G. Sleipen. Iterative Bi-CG type methods and implementational aspects. In G. W. Althaus and E. Spedicato, editors, *Algorithms for Large Scale Linear Algebraic Systems*, pages 217–253. NATO Advanced Study Institute, Kluwer Academic Publishers, 1996.
- [243] H. A. van der Vorst and C. Vuik. The superlinear convergence behaviour of GMRES. *J. Comput. Appl. Math.*, 48:327–341, 1993.
- [244] A. C. N. van Duin and H. Wijshoff. Scalable parallel preconditioning with the sparse approximate inverse of triangular systems. Technical report, Computer Science Department, University of Leiden, Leiden, The Netherlands, 1996.
- [245] R. S. Varga. Factorizations and normalized iterative methods. In R. E. Langer, editor, *Boundary Problems in Differential Equations*, pages 121–142. University of Wisconsin Press, Madison (WI), 1960.
- [246] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1962.
- [247] P. S. Vassilevski. A block-factorization (algebraic) formulation of multigrid and Schwarz methods. *East-West J. Numer. Math.*, 6:65–79, 1998.

- [248] P. K. W. Vinsome. Orthomin, an iterative method for solving sparse sets of linear equations. In *Proc. Fourth Symposium on Reservoir Simulation, Society of Petroleum Engineers of AIME*, pages 149–159, 1976.
- [249] V. V. Voevodin. On methods of conjugate directions. *J. of Computational Math. and Math. Phys.*, 19(5):1313–1317, 1979. In Russian.
- [250] V. V. Voevodin. The problem of generalization of the CG method is now closed. *J. of Computational Math. and Math. Phys.*, 23(2):477–479, 1983. In Russian.
- [251] V. V. Voevodin and E. Tytyshnikov. On a generalization of the method of conjugate directions. In V. V. Voevodin, editor, *Numerical Methods of Algebra*, pages 3–9. Moscow University Press, Moscow, 1981. In Russian.
- [252] R. von Mises and H. Geiringer. Praktische Verfahren der Gleichungsauflösung. *Zeitschrift für angewandte Mathematik und Mechanik*, 9:58–77 and 152–164, 1929.
- [253] X. Wang, K. A. Gallivan, and R. Bramley. CIMGS: An incomplete orthogonal factorization preconditioner. *SIAM J. Sci. Comput.*, 18:516–535.
- [254] J. W. Watts III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Society of Petroleum Engineers Journal*, 21:345–353, 1981.
- [255] R. Weiss. *Convergence Behaviour of Generalized Conjugate Gradient Methods*. PhD thesis, University of Karlsruhe, 1990.
- [256] R. Weiss. *Parameter-Free Iterative Linear Solvers*, volume 97 of *Mathematical research*. Akademie Verlag, Berlin, 1996.
- [257] O. Widlund. A Lanczos method for a class of nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 15:801–812, 1978.
- [258] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965.
- [259] D. M. Young. Iterative methods for solving partial differential equations of elliptic type. *Trans. Amer. Math. Soc.*, 76:92–111, 1954.
- [260] D. M. Young and K. C. Jea. Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra Appl.*, 34:159–194, 1980.
- [261] L. Zhou and H. F. Walker. Residual smoothing techniques for iterative methods. *SIAM J. Sci. Comput.*, 15:297–312, 1994.
- [262] Z. Zlatev. *Computational Methods for General Sparse Matrices*. Kluwer, Dordrecht, 1991.

This page intentionally left blank

Index

- algorithm
Arnoldi's original, 28
BiCG, 17, 45, 52, 99, 125, 170, 173, 180, 200, 278
BiCG, transpose-free version, 114
BiCGL, 44, 53, 99, 170
BiCGStab, 17, 109, 115, 132, 173, 278
BiCR, 17, 40, 45, 60, 170, 173, 180, 278
BiCRL, 17, 61, 173
BICG, 17
block-CG, 43, 44, 153
CG, 17, 45, 51
CGNE, 17, 64
CGNR, 17, 59
CGS, 17, 102, 105, 114, 132, 170, 173
CGW, 17, 55
compound, 48
CR, 17, 40, 45, 57, 149
FOM, 17, 35, 36, 40
GCR, 17, 32, 40
generalised Arnoldi, 29
generalised Gram-Schmidt, 23, 27
GMRes, 17, 35, 37, 40, 91, 117, 173, 278
GMRes(m), 173
GODir, 17, 32, 43, 46, 133
GOMin, 17, 32, 43, 46, 133
GORes, 17
HG, 17, 45, 54, 173, 180, 200, 278
HGL, 17, 54, 173
level of fill, 238
LSQR, 17, 67, 117, 278
MCR, 17, 40, 58
MinRes, 35, 199
MRS, 126, 131
MRSM, 130
MRZ, 17, 44
OD, 17, 62, 149
OrthoDir, 17, 32, 40
OrthoMin, 17
OrthoRes, 17, 171, 282
HS analogue of, 283
- QMR, 17, 44, 69, 102, 117, 125, 126, 149, 171, 173
QMR (HS version), 120
QMR (symmetric), 122
QMRCBiCG, 173, 180
QMRCGStab, 17, 114, 131
QMRS, 114
Remez, 217
SIC, 234
simple, 48
ST, 236
ST, IKJ variant, 241
StOD, 17, 63
SymmLQ, 17, 65, 117, 199
TFQMR, 17, 113, 115, 131, 278
- algorithms
alternative forms of, 45
crashing of, 49
dual, 284
equivalent, 84
implementation of, 117
minimum-residual, 57, 61
preconditioned, 207
restarting of, 39
reverse, 284
Sonneveld, 115, 278
stability of, 83
stagnation of, 50
truncation of, 40
- bandwidth, 257
block conjugacy, 19
block methods
GMRes, 162
of Chronopoulos and Gear, 158
of Hackbusch, 157
VBCG, 158
- blocking, 274
breakdown
artificial incurable, 150
causes of, 24
incurable, 27, 134
serious, 24, 27, 133

- compressed sparse row format, 254
- convergence, 193
 - general theory of, 87
 - of CG, 90
 - of CGNE, 90
 - of CGNR, 90
 - of CR, 90
 - of GMRes, 92
- diagonal compensation, 245
- displacement vectors, 7
- distribution function, 217
- dropping, 262
- dropping criteria, 244
- dropping rules, 244
- dual system, 280
 - solving by GODir, 281
 - solving by GOMin, 282
- duality, 279
 - inherent in CG, 283
- energy norm, 3
- equivalent BiCG iterations, 176
- equivalent system, 4
- error vector, 3
- Euclidean norm, 3
- factorisation by position, 240
- fill-in, 19
- Galerkin condition, 6, 12, 35
- GODir
 - avoidance of breakdown, 78
- GOMin
 - avoidance of breakdown, 80
 - avoidance of stagnation, 82
- Harwell-Boeing collection, 176, 248, 255
- IKJ variant, 241
- ILUTP, 254, 255
- Jacobi operator, 227
- Krylov, 16
- Krylov aspects, 145
- Krylov sequence, 77
 - level of fill, 236
 - look-ahead, 188
 - look-ahead algorithms
 - BiCG, 143
 - block HS, 139
 - block Lanczos, 138
 - CG, 142, 149
 - GODir, 145
 - GOMin, 146
 - HG, 144
 - HGL, 144
 - MRZ, 143
 - QMR, 144
- look-ahead versions, 27
- MATLAB, 173
- matrices
 - B-adjoint, 74
 - B-normal, 74
 - B-normal(s), 75
 - B-self-adjoint, 74
 - band, 257
 - conjugate, 13
 - consistently ordered, 230
 - convergent, 221
 - diagonally dominant, 220
 - H-matrices, 226
 - idempotent, 6
 - inner-product, 75
 - irreducible, 219
 - irreducibly diagonally dominant, 221
 - J-Hermitian, 124
 - J-symmetric, 124
 - Krylov, 77
 - M-matrices, 226
 - monotone, 223
 - non-negative, 219
 - normal, 92
 - persymmetric, 124
 - positive, 219
 - positive real, 31
 - preconditioned system, 75
 - preconditioning, 75, 197
 - projection, 6
 - sparse, 1
 - square roots of, 4
 - Stieltjes, 226
 - system, 75
 - Toeplitz, 125
- matrix
 - normality, 176
 - symmetry, 177
- matrix equations, 12
- matrix scaling
 - absolute, 180
 - Euclidean, 180
- matrix splitting, 223
- method
 - Arnoldi's, 28
 - escalator, 19
 - Gauss-Seidel, 225, 228
 - Gram-Schmidt, 22
 - Jacobi's, 222, 226

- of hyperbolic pairs, 142
- of minimised iterations, 19
- of simultaneous displacements, 222
- of steepest descent, 7
- of successive displacements, 229
- of von Mises, 208
- original Lanczos, 47
- SOR, 228
- SSOR, 271
- strongly implicit, 234
- methods
 - block, 151
 - conjugate direction, 13
 - conjugate gradient, 13
 - for parallel computers, 262
 - Galerkin, 51, 186, 195
 - Hegedus, 202, 278
 - HS, 16, 45, 183
 - hybrid, 207
 - Lanczos, 16, 44, 64, 183
 - limited fill-in, 244
 - look-ahead, 133
 - minimum residual, 186
 - multigrid, 273
 - multilevel, 273
 - norm-reducing, 3
 - projection, 10
 - Schur complement, 273
 - transpose-free, 105
- model problem, 272
- multigrid
 - algebraic, 273
 - black-box, 273
 - geometric, 273
- nested sequence, 117
- numerical considerations, 163, 252
- oblique projector, 7
- ordering/s
 - Cuthill and McKee, 257
 - MDF, 259
 - minimum degree, 257
 - RCM, 258
 - van der Vorst, 272
- orthogonal projector, 6
- outliers, 94, 194
- partial solution, 134
- peaks and plateaus, 39
- Perron-Frobenius theorem, 220
- polynomials
 - Chebychev, 88, 207, 216
 - Jacobi, 216
 - minimal, 80
- residual, 92
- preconditioned algorithms
 - PBiCR, 205
 - PCR, 205
- preconditioner, 197
 - Legendre, 218
- preconditioners
 - (S)SOR, 226
 - AIAF, 263, 271, 275
 - AIB, 262, 267, 275
 - AIF, 262, 267, 276
 - AIM, 262, 266, 276
 - Alnv, 262, 268
 - Cesari's, 208
 - Chebychev, 212
 - DCR, 235
 - DS, 275
 - FSAI, 262, 267, 275
 - IC, 212, 225, 233
 - ICT, 246, 275
 - ILU, 231, 232
 - ILU(0), 233, 256, 260, 275
 - ILU(p), 235, 256, 275
 - ILU, IKJ version, 242
 - ILUT, 243, 247, 256, 260, 275
 - ILUT, variant, 248
 - ILUTP, 256
 - JP, 262, 270
 - MR, 262, 266
 - MRP, 266
 - Neumann, 210
 - of Ajiz and Jennings, 235
 - polynomial, 215
 - SAInv, 268
 - SpAI, 262, 264, 275
 - SSOR, 230
 - Tismenetsky's, 250
 - TN, 270
 - TNIC, 263
 - TNILU(0), 277
 - TNILUT, 277
 - TNSGS, 263, 275
 - preconditioning, 189, 193
 - general polynomial, 208
 - ILU
 - accuracy of, 253
 - pivoting in, 254
 - stabilisation of, 254
 - stability of, 253
 - types of failure, 253
 - left, 194
 - polynomial, 207
 - right, 194

split, 195
preface, viii
projector, 6
property A, 230
QMR technique, 8, 117, 131
re-ordering, 257
recurrences
 existence of short, 70
 long, 21, 35
 restarted, 35
 short, 35, 43
 truncated, 35
relaxation factor, 228
relaxation step, 227
residual smoothing, 187
residual vector, 3
residuals
 computed, 164
 true, 164
Ritz-Galerkin condition, 6
Ritz-value, 94, 98
Schur complements, 83, 273
sophisticated linear algebra, 189
sparse systems, 1
SPARSKIT, 255
splitting, 223
 convergent, 223
 regular, 224
stabilisation, 254
stability
 of GODir, 83
 of GOMin, 83
structurally symmetric, 258
symmetric permutation, 258
symmetry, imposition of, 249
systems
 multiple, 151
 single, 157
termination, 193
 finite, 15
 GODir, finite, 32
 GOMin, finite, 34
 regular, 24, 26
 types of, 27
UMFPACK, 255