Modified Cholesky Decomposition and Applications

McSweeney, Thomas

2017

MIMS EPrint: **2017.45**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

MODIFIED CHOLESKY DECOMPOSITION AND APPLICATIONS

A dissertation submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Science and Engineering

2017

Thomas McSweeney
School of Mathematics

Contents

Word count 30,489

List of Tables

List of Figures

Abstract

The modified Cholesky decomposition is one of the standard tools in various areas of mathematics for dealing with symmetric indefinite matrices that are required to be positive definite. We survey the literature and determine which of the existing modified Cholesky algorithms is most suitable for inclusion in the Numerical Algorithms Group (NAG) software library, focussing in particular on the algorithms of Gill, Murray and Wright, Schnabel and Eskow, Cheng and Higham, and Moré and Sorensen. In order to make this determination we consider how best to take advantage of modern computer architectures and existing numerical software. We create an efficient implementation of the chosen algorithm and perform extensive numerical testing to ensure that it works as intended. We then discuss various applications of the modified Cholesky decomposition and show how the new implementation can be used for some of these. In particular, significant attention is devoted to describing how the modified Cholesky decomposition can be used to compute an upper bound on the distance to the nearest correlation matrix.

Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual Property Statement

i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the dissertation, for example graphs and tables ("Reproductions"), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Dissertation restriction declarations deposited in the University Library, The University Library's regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University's Guidance on Presentation of Dissertations.

Acknowledgements

I would like to thank my academic supervisor Nick Higham for all his help and guidance. Many thanks are of course also due to all the staff at NAG's Manchester office, especially my industrial supervisor Craig Lucas.

Chapter 1

Introduction

1.1 The Cholesky factorization

An $n \times n$ symmetric matrix A is called *positive definite* if $x^T Ax > 0$ for all nonzero vectors $x \in \mathbb{R}^n$ [36, p. 196]. It is *positive semidefinite* if the inequality is not strict. The concepts of *negative definite* and *negative semidefinite* are defined analogously. We say the matrix is *indefinite* if none of the previous definitions hold.

The definiteness of a matrix is one of its most revealing attributes, with many equivalent definitions and mathematical implications. For example, a matrix is positive definite if, and only if, all of its eigenvalues are positive [36, p. 196]. It is positive semidefinite if we also allow the possibility of some of its eigenvalues being exactly zero. The sign of the eigenvalues of a negative definite or semidefinite matrix should again be clear by analogy. A matrix is indefinite if it has both positive and negative eigenvalues.

Positive definiteness confers many desirable properties upon a matrix, leading Higham to remark in [36, p. 196] that *"symmetric positive definiteness is one of the highest accolades to which a matrix can aspire."* In particular, A is positive definite if, and only if, it has a unique factorization of the form $A = LL^T$, where L is a lower triangular matrix with strictly positive diagonal entries. This is known as the *Cholesky decomposition* (or *factorization*) and its discovery dates back to the early part of the twentieth century [38]. A standard algorithm for computing it also dates to the same period, and can be described as follows.[1]

[1] There are several minor variants of this algorithm; this is known as the *jik* or "sdot" form [36,

Algorithm 1.1: Computes the Cholesky factorization $A = LL^T$ of a symmetric positive definite matrix A

Input : Symmetric positive definite matrix A
Output: Lower triangular matrix L with positive diagonal entries

for $j = 1, \ldots, n$ **do**
  $\ell_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} \ell_{jk}^2 \right)^{1/2}$
  for $i = j + 1, \ldots, n$ **do**
    $\ell_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk} \right) / \ell_{jj}$
  end
end

When applied to an $n \times n$ matrix, Algorithm 1.1 costs about $n^3/3$ flops [38], where a *flop* (short for *floating point operation*) is one of the fundamental arithmetical operations $+, -, /$, or $*$. This can prove to be extremely efficient for many applications. For example, when A is positive definite we can use a Cholesky factorization to solve linear systems of equations $Ax = b$: we find the factorization $A = LL^T$, then we solve the systems $Ly = b$ for y and $L^T x = y$ for x. The cost of solving the two triangular systems is only $O(n^2)$ flops, so therefore the overall cost to highest order terms is just the cost of the factorization, $n^3/3$. When it is applicable, this can often be the most efficient method for solving systems of linear equations and is therefore used widely for that purpose [28, Chapter 4].

Cholesky factorization is considered to be amongst the most numerically stable (see Appendix A) of all matrix factorizations [35, 75]. This stability follows from the fact that the elements of L are bounded relative to those of A since

$$\sum_{k=1}^{i} \ell_{ik}^2 = a_{ii} \implies \ell_{ij} \leq a_{ii}.$$

With this it can be shown that if $\hat{x}$ is the computed solution to the linear system $Ax = b$ found using the Cholesky factorization as described previously, then $\hat{x}$ is in fact the exact solution to the system $(A+E)\hat{x} = b$, where $\|E\|_2 \leq c_n u \|A\|_2$ [28, p. 147]. Here, c_n is a small constant that depends on n, u is the unit roundoff (see Appendix A) and $\|\cdot\|_2$ is the matrix 2-norm (see Appendix B). Further, Wilkerson showed in [75] that if $\kappa_2(A) d_n u < 1$, where d_n is another constant depending on n and $\kappa_2(A)$ is

p. 197]. They all begin by equating $A = LL^T$ elementwise and then solving the resulting equations; the order in which this is done determines the order of i, j and k.

the 2-norm condition number of A (see Appendix B), then the Cholesky factorization always runs to completion.

Other than solving positive definite systems of linear equations, the Cholesky factorization is used in many other contexts. A simple extension of the previous example would be solving the normal equations for any nonsingular system of linear equations. More broadly, we can use the Cholesky factorization in almost any area in which positive definite matrices occur. To name just a few examples, the Cholesky factorization is used in the Monte Carlo method [66], Kalman filtering [46] and many different areas of optimization [16].

An alternative way to express the Cholesky factorization of A is $A = R^T R$, where R is upper triangular with positive diagonal entries; in this case, R is simply L^T. Less obviously, the Cholesky decomposition of a symmetric positive definite matrix A can also be expressed as $A = LDL^T$, where L is a unit lower triangular matrix and D is a diagonal matrix with strictly positive diagonal entries. The two factorizations are in fact equivalent: if we are given $A = LDL^T$ then we have $A = (LD^{\frac{1}{2}})(LD^{\frac{1}{2}})^T =: \tilde{L}\tilde{L}^T$. Alternatively, if we are given $A = \tilde{L}\tilde{L}^T$, then if we define a diagonal matrix D such that $d_{ii}^{1/2} = \ell_{ii}$ for all $i = 1, \ldots, n$, we can find a unit lower triangular matrix L such that $A = LDL^T$ by solving the system $\tilde{L} = LD^{1/2}$.

Algorithm 1.2 describes one method of computing the LDL^T variant of the Cholesky factorization of a symmetric positive definite matrix A[2]. We can see that we do not need to take any square roots in the algorithm. This eliminates one of the immediate potential issues arising from Algorithm 1.1 that we have thus far avoided: what if the expression we wish to find the square root of is negative? However, we still have a problem if any of the d_{jj} are zero. In fact, it can be shown that one of the many equivalent definitions of positive definiteness is that the expression we take the square root of in Algorithm 1.1 is always strictly positive [36, p. 196]. Similarly, the d_{jj} in Algorithm 1.2 are all positive if the matrix A is positive definite. Indeed, this is actually often the preferred way to check if a given matrix actually is positive definite: we attempt a Cholesky factorization and if it fails, we know that it is not [36, p. 210]. This once more emphasises the potential efficiency of the Cholesky factorization relative to other methods.

[2]Again, there are minor variants.

Algorithm 1.2: Computes the Cholesky factorization $A = LDL^T$ of a symmetric positive definite matrix A

Input : Symmetric positive definite matrix A
Output: Unit lower triangular matrix L, positive diagonal matrix D

for $j = 1, \ldots, n$ **do**
  $d_{jj} = a_{jj} - \sum_{k=1}^{j-1} \ell_{jk}^2 d_{kk}$
  for $i = j + 1, \ldots, n$ **do**
    $\ell_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} \ell_{jk} d_{kk} \right) / d_{jj}$
  end
end

Although the LL^T and LDL^T factorizations are equivalent, there can be circumstances in which one is preferred to the other. For example, the method of solving systems of linear equations using the LDL^T variant is very similar to the LL^T method already described but is actually more efficient when solving tridiagonal systems of equations [36, p. 197]. Computing square roots can also be expensive relative to more fundamental mathematical operations in some computing environments. If it is necessary to use a machine for which this disparity is extreme, the LDL^T variant of the Cholesky factorization would again be preferred, especially if efficiency is of the utmost importance.

A matrix H with complex entries is *Hermitian* if it is equal to its *conjugate transpose*, the matrix H^* formed by taking the transpose of H and then replacing all its elements by their respective complex conjugates. We mention in passing that the Cholesky factorization can be naturally extended to all complex Hermitian positive definite matrices if we allow the factorization to include the conjugate transpose of the triangular matrix rather than just its transpose [28, p. 147]. Many of the results in this dissertation may be likewise extended; however, the reader should be aware that others may not and we shall restrict ourselves entirely to matrices with real entries.

1.2 The modified Cholesky factorization

Due to the efficiency and stability of the Cholesky factorization, in practice situations arise in which we wish to use one but the matrix in question is not positive definite; see Chapters 7 and 8 for examples. If the matrix A is positive semidefinite, then we can

always find a factorization of the form $A = LDL^T$, but D may have zero elements on the diagonal [35, 38] and the factorization is not unique [36, p. 201]. We can however find a permutation matrix P such that PAP^T has a unique factorization of the form LDL^T, with

$$D = \begin{bmatrix} D_1 & 0 \\ 0 & 0 \end{bmatrix},$$

where D_1 is a square diagonal matrix with positive diagonal elements. The dimension of D_1 is r, where r is the rank of A, so this factorization is useful for its rank-revealing property [36, p. 202]. This could be considered to be sufficient to say that we can extend the Cholesky factorization to all positive semidefinite matrices as well, however there are contexts in which this factorization is not considered adequate [74, 76]. In any event, if the matrix A is neither positive definite nor semidefinite then we have no such results and it is not clear how we should proceed.

Several *modified* Cholesky algorithms exist that aim to compensate for this lack of positive definiteness. The basic idea is that we perturb A (i.e., add a matrix E to it) to make it positive definite and then find a Cholesky factorization of this perturbed matrix instead. The challenge is to do this in such a way that the perturbed matrix remains pertinent to the original application.

In practice, we usually need to perform pivoting on the matrix being factorized to ensure numerical stability, so we actually find the factorization

$$P(A + E)P^T = LDL^T,$$

where P is a permutation matrix instead. However, this does not alter the theory in any appreciable way [12]. We discuss the particular pivoting strategies employed by the existing modified Cholesky algorithms in turn as we discuss them in depth in Chapters 2 and 3.

Clearly, for the perturbed matrix $A+E$ to be of any practical use, the perturbation made—and therefore the matrix E—must be in some mathematical sense "small". The natural way to define this is in terms of the norm of E: we desire that this is not much larger than it needs to be in order to ensure that $A + E$ is positive definite (and in particular, if A is actually positive definite then we should like it to be zero). It is not clear that any particular norm is naturally better suited for this than the others. We

will most frequently use the norms described in Appendix B, as they are the norms most commonly used throughout numerical linear algebra [28, p. 55].

Given a symmetric matrix A and a real number $\delta \geq 0$, a perturbation matrix whose norm is the minimal distance from A to the set of symmetric matrices with minimum eigenvalue δ can be readily computed for the 2- and Frobenius norms [12]. The most obvious approach to finding E then would be to simply undertake this computation, for the chosen norm, with δ as some acceptably small tolerance for the minimum eigenvalue of $A+E$. The problem is that finding these minimal perturbations requires computing the eigenvalues of A, which is an $O(n^3)$ operation that is more expensive than the standard Cholesky factorization itself. As explained in the previous section, one of the most common reasons for wishing to use a Cholesky factorization for a matrix that is not positive definite is its efficiency relative to other methods and we should therefore like to preserve this as far as possible. Hence we desire that any prospective modified Cholesky algorithm is no more expensive than the standard Cholesky algorithm, or at least not significantly more so.

While the modified Cholesky factorization of a matrix may be of mathematical interest in itself, we shall see in later chapters that in practice we very often want to actually use the perturbed matrix $A + E$ in computations. Therefore, the condition number of the matrix $A + E$ is important here. We wish for it to be relatively small and hence for the matrix itself to be well conditioned. At the very least, we do not want the perturbed matrix to be so ill conditioned that it would adversely affect the accuracy of further computations.

Schnabel and Eskow in [70] codified all the objectives we have mentioned into a concise list, which is repeated with some minor revisions by Cheng and Higham in [12] and Fang and O'Leary in [21]. We give the variant of Cheng and Higham here, for reasons that we shall elaborate on forthwith.

1. If A is "sufficiently positive definite," then E should be zero.

2. If A is indefinite then $\|E\|$ should be relatively small, i.e., $\|E\|$ should not be much larger than $\min\{\|\Delta A\| : A + \Delta A \text{ is positive semidefinite}\}$, for some appropriate norm.

3. The matrix $A + E$ should be reasonably well conditioned.

4. The cost of the algorithm should not be prohibitive: it should be at most the same as the standard Cholesky algorithm, to highest order terms (i.e., $n^3/3 + O(n^2)$ flops).

We shall refer to these four objectives by number throughout this dissertation.

Precisely how we define "sufficiently" positive definite may depend on the algorithm we are using and the problem to which we are applying it [21]. Naively, we would simply say a matrix is positive definite if all of its eigenvalues are greater than zero (and therefore, in practice, we would generally consider a matrix to be positive definite if all of its eigenvalues exceed the unit roundoff). However, this is not necessarily the most practical choice and we shall see in the following two chapters that the existing modified Cholesky algorithms often use different metrics to determine if a matrix is to be regarded as positive definite or not.

Schnabel and Eskow in [70] and [71] state the second objective as $\|E\|_\infty$ being not much greater than the magnitude of the most negative eigenvalue of A. For any given matrix, this quantity is the distance to the nearest positive semidefinite matrix in the 2-norm [21] (and therefore also the ∞-norm for a diagonal matrix E). We shall see in Chapter 2 that many of the existing modified Cholesky algorithms restrict themselves to diagonal E; however, others do not (see Chapter 3) and therefore the more general formulation is preferred here.

It is not immediately clear how we should quantify "reasonably" in the statement of Objective 3, or even if we can. At the very least we would hope that we can somehow bound the condition number of $A + E$ relative to A. There are applications in which we wish to preserve the condition number of the matrix as much as possible when we perturb it, even if it is extremely ill conditioned [71]. Theoretical bounds on the conditioning of the perturbed matrix are known for many of the existing algorithms and will be discussed in the coming chapters.

The success of many of the current modified Cholesky algorithms in meeting these objectives has been considered, both in theory and through numerical experimentation [12, 21, 70, 71]. Several of them have also been applied to real-world problems [68, 77], allowing us to analyse their practical performance.

In addition to the stated objectives, we obviously also wish that a modified Cholesky algorithm adheres to the more general aims of any numerical algorithm, chief among

them numerical stability.

With regards to practical implementation in particular, it is surely also wise to consider how easily an algorithm can be adapted to modern computer architectures and integrated into existing software libraries.

The reader should be aware that—confusingly—the LDL^T variant of the standard Cholesky factorization for positive definite matrices is occasionally referred to as the modified Cholesky factorization in some sources; for example, [29, p. 295].

1.3 Structure and aims of this dissertation

This dissertation is sponsored by the Numerical Algorithms Group Ltd., who shall generally be referred to by their preferred acronym of 'NAG' for the rest of this dissertation. Their desire is for a survey of the extant modified Cholesky algorithms to be conducted and that the most suitable one be implemented in the NAG software library. Chapters 2 and 3 are the result of the aforementioned survey, which has been undertaken by the author. The existing algorithms can be broadly divided into two categories that approach the problem of an indefinite A from opposite directions. The first is those that work by making a strictly diagonal perturbation to A in order to make it positive definite. These are the focus of Chapter 2. The other category is those that work by first finding another factorization of A and then perturbing the factors in order to construct a positive definite matrix $\tilde{A} = A + E$ that is (in some well-defined sense) close to the original matrix. In Chapter 3, we discuss suitable factorizations and the modified Cholesky algorithms that employ them.

In Chapter 4, we collate all of the existing theory and data in order to compare the relative performance of many of the algorithms that have been discussed and ultimately decide which one is most suitable for inclusion in the NAG Library. As the NAG Library is a commercial product, intended to be run on a wide variety of machines and used for many different purposes, it is important that the implementation of the chosen algorithm be as robust and efficient as possible. Chapter 5 records the details of how we implemented the algorithm with these additional objectives in mind and Chapter 6 the results of the numerical testing that we performed to ensure that it works as intended.

The inspiration for NAG's decision to incorporate the modified Cholesky algorithm into their library was a recent paper of Higham and Strabić in which the modified Cholesky factorization is used to compute an upper bound on the distance from a symmetric indefinite matrix to the *nearest correlation matrix* [40]. This term will be defined in Chapter 7, where we discuss the problem in greater depth and show how the new NAG modified Cholesky algorithm can be used to compute this distance efficiently. Other applications of the modified Cholesky factorization are then briefly described in Chapter 8. We conclude with some final remarks in Chapter 9.

1.4 Computing environment

All numerical experiments in this dissertation were performed on a machine with two octa-core Intel Sandy Bridge processors (Xeon E5-2670 2.60 GHz; see here [44] for more detailed specifications). We used Intel's multithreaded Math Kernel Library (MKL) [43] for optimized BLAS (see Appendix C); the number of cores used to achieve individual results will always be stated explicitly before they are presented.

From Chapter 5 onwards we used a bespoke build of the NAG Library Toolbox [51] for MATLAB R2016b incorporating the new `F01MDF` routine. Although the routine is written in Fortran 90, we accessed MATLAB via a MEX interface, enabling us to take advantage of its excellent built-in linear algebra and plotting functions. Note that the Toolbox uses the alternative name `F01MD` for the routine but we will generally use the former name as that is the name of the routine in the NAG Library kernel.

Data from numerical experiments performed by other sources may occasionally be presented. These will be clearly identified and referenced as thoroughly as possible.

Chapter 2

Diagonal Perturbation Algorithms

The first numerically stable modified Cholesky algorithm was introduced by Gill and Murray in the early 1970s [25] and refined by Gill, Murray and Wright in 1981 [27, Chapter 4]. For the sake of brevity, we shall refer to this algorithm as the GMW algorithm. Another modified Cholesky algorithm was proposed by Schnabel and Eskow in 1990 [70] and revised by the same authors in 1999 [71]. We shall refer to this as the SE algorithm, or SE90 and SE99 if it is necessary to distinguish between the two.

Motivated by practical problems involving indefinite Hessian matrices in optimization (see Chapter 8), both the GMW and SE algorithms deal with an $n \times n$ symmetric indefinite matrix A by essentially proceeding as an ordinary Cholesky factorization but adding a sequence of numbers e_j, $j = 1, \ldots, n$ to the diagonal of A such that the d_{jj} calculated by Algorithm 1.2 (or the ℓ_{jj} calculated by Algorithm 1.1) are always strictly positive. Effectively, they find the Cholesky factorization of the positive definite matrix $A + E$, where $E = \text{diag}(e_j)$.

The key to both algorithms is choosing the numbers e_j. The naive approach would be to simply always choose the smallest $e_j \geq 0$ such that d_{jj} or ℓ_{jj} is positive. This would certainly satisfy Objectives 1 and 4. However, in general Objective 2 is not satisfied [21, 71] and there is no reason to believe it would be with this approach. Precisely how the numbers are actually chosen is one of the key differences between the two algorithms—although there are others—and will be described in the following two sections.

2.1 The Gill, Murray and Wright (GMW) algorithm

In order to describe the GMW algorithm it suffices to describe one iteration. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Define $A_1 = A$. At the jth stage of the factorization, the submatrix of A still to be factorized has the form

$$A_j = \begin{bmatrix} \alpha_j & b_j^T \\ b_j & \bar{A}_j \end{bmatrix}, \tag{2.1}$$

where $\alpha_j \in \mathbb{R}$, $b_j \in \mathbb{R}^{(n-j)}$ and $\bar{A}_j \in \mathbb{R}^{(n-j)\times(n-j)}$. In practice we normally pivot on the matrix A_j at this stage: we find the largest entry in magnitude along the main diagonal and then interchange rows and columns so this entry is in the top left corner (i.e., it is α_j). Pivoting is not theoretically necessary for the algorithm to work but empirical data suggests that it usually improves its performance [70].

Let $\delta \geq 0$ be an input tolerance. Then we determine the numbers e_j for each $j = 1, \ldots, n$ such that

$$\alpha_j + e_j = \max\left\{\delta, |\alpha_j|, \frac{\|b_j\|_\infty^2}{\beta^2}\right\}, \tag{2.2}$$

where $\beta > 0$ is a constant. Usually, the tolerance δ is set to the unit roundoff, u [27, Chapter 4]. The $|\alpha_j|$ term is included in an attempt to ensure that if $\alpha_j < 0$, then e_j is bounded below by $-2\alpha_j$ [71].

Having found e_j, we can now update the factorization according to

$$d_{jj} = \alpha_j + e_j, \quad \ell_{ij} = \frac{(b_j)_i}{d_{ii}}, \quad i = j+1, \ldots, n,$$

and calculate the next submatrix iterate A_{j+1} by

$$A_{j+1} = \bar{A}_j - \frac{b_j b_j^T}{\alpha_j + e_j}. \tag{2.3}$$

The constant β is included in an attempt to achieve Objectives 1 and 2 (i.e., ensuring that E is zero when A is positive definite and $\|E\|$ is small otherwise). To see how it is chosen, let us define the following two quantities:

$$\zeta = \max_{\substack{1\leq i,j\leq n \\ i\neq j}} |a_{ij}|, \qquad \text{the maximum off-diagonal entry of } A,$$

$$\gamma = \max_{1\leq i\leq n} |a_{ii}|, \qquad \text{the maximum diagonal entry of } A.$$

Then we can establish the bound [12, 21],

$$\|E\|_2 \leq \left(\frac{\xi}{\beta} + (n-1)\beta\right)^2 + 2\Big(\gamma + (n-1)\beta^2\Big) + \delta. \tag{2.4}$$

Again, note that because we only consider diagonal E, $\|E\|_2 = \|E\|_1 = \|E\|_\infty$. So from (2.2), the smallest possible bound we can have over all $\beta > 0$ is

$$\|E\|_\infty \leq 2\xi(\sqrt{n^2-1} + n - 1) + 2\gamma + \delta, \tag{2.5}$$

which is achieved when $\beta^2 = \frac{\xi}{\sqrt{n^2-1}}$ [27, Chapter 4]. However, choosing this value of β can perturb positive definite A, so steps must be taken to ensure that we avoid this in order to achieve Objective 1. Gill, Murray and Wright did this by showing that if $\beta^2 \geq \gamma$, then E is zero. So if we calculate β using

$$\beta^2 = \max\left\{\gamma, \frac{\xi}{\sqrt{n^2-1}}, u\right\}, \tag{2.6}$$

then Objective 1 should be satisfied [27, Chapter 4]. Substituting the possible values of β from (2.6) into the bound (2.4), we see that in any case, $\|E\|_\infty \leq K$, where K is a constant of $O(n^2)$.

The GMW algorithm very closely follows the standard Cholesky algorithm but with the additional cost of finding each of the e_j using (2.6). To highest order, the cost of calculating β from (2.6) is just the cost of finding ζ, which requires at most n^2 comparisons. Similarly, the cost of computing each of the $\|b_j\|_\infty$ requires at most n comparisons, and therefore finding all of them is also an $O(n^2)$ operation. Overall then, to highest order terms, the cost of the GMW algorithm is the same as that of a standard Cholesky algorithm ($n^3/3$ flops) and Objective 4 is achieved.

Objective 3 is a little trickier. A bound that is exponential in n for the worst case condition number of the matrix $A + E$ can be found [21],

$$\kappa_2(A+E) = O\left(n^3\left(\frac{\xi+\gamma}{\delta}\right)^n\right).$$

This is of limited practical use, but it has been found in applications that $\kappa_2(A+E)$ is generally much smaller than this [21, 70].

No explicit analysis of the GMW algorithm's numerical stability was presented in [27, Chapter 4]. However, Cheng and Higham in [12] make the following simple argument demonstrating that it is in fact backward stable (and therefore numerically

stable). Given a matrix A, use the GMW algorithm to compute the factorization $P(A+E)P^T = LDL^T$. Since $P(A+E)P^T$ is positive definite, when we apply the GMW algorithm to it, we get the same factorization $P(A+E)P^T = LDL^T$. This is just a standard Cholesky factorization, which is backward stable. Therefore, the GMW algorithm itself is also backward stable.

From its inception, the GMW algorithm gained widespread use, particularly in unconstrained optimization problems, and is generally considered to have performed very well [68, 70].

2.2 The Schnabel and Eskow (SE) algorithm

2.2.1 The SE90 algorithm

If $\gamma > \frac{\xi}{\sqrt{n^2-1}}$, then the upper bound on the norm of the perturbation matrix E computed by the GMW algorithm is actually

$$\|E\|_\infty \leq (n^2+1)\gamma + 2(n-1)\xi + \frac{\xi^2}{\gamma} + \delta, \tag{2.7}$$

which is looser than the bound (2.4). Schnabel and Eskow argued that this was likely to be the case in practice, and the desire to improve this bound motivated the creation of their alternative modified Cholesky algorithm [70]. The algorithm proceeds much as the GMW algorithm does—like a standard Cholesky factorization with the additional step of finding the numbers e_j—but instead uses a lemma based on Gershgorin's theorem to find the diagonal perturbations [70].

Theorem 2.1 (Gershgorin's theorem) *The eigenvalues of $A \in \mathbb{C}^{n\times n}$ are contained in the union of the discs C_i, which are defined for $i = 1, \ldots, n$ by*

$$C_i = \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \right\}.$$

We have given the more general formulation of the theorem for complex matrices, although as ever here we only consider matrices with real entries. Further, since we are concerned with symmetric A and the eigenvalues of a symmetric matrix are all

real, the discs C_i are in fact intervals R_i of the real line defined by

$$R_i = \left[a_{ii} - \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, a_{ii} + \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \right].$$

The most important implication of the theorem from our perspective is that if we define the numbers e_i, $i = 1, \ldots, n$ such that

$$e_i \geq -a_{ii} + \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|,$$

then $A+E$ has no negative eigenvalues and so is at least positive semidefinite. [21, 70].

Note that for the moment, we will focus on making the perturbed matrix $A + E$ positive semidefinite rather than strictly positive definite. Steps are taken later in the algorithm to ensure that the matrix $A + E$ is in fact strictly positive definite [70].

The next lemma follows from Theorem 2.1 [21].

Lemma 2.2 *Suppose $A \in \mathbb{R}^{n \times n}$ has the form*

$$A = \begin{bmatrix} \alpha & b^T \\ b & \bar{A} \end{bmatrix},$$

where $\alpha \in \mathbb{R}$, $b \in \mathbb{R}^{n-1}$ and $\bar{A} \in \mathbb{R}^{(n-1)\times(n-1)}$. Let $\delta \geq \max\{0, \|b\|_1 - \alpha\}$ and define $\tilde{A} = \bar{A} - bb^T/(\alpha + \delta)$. Then $C_i(\tilde{A}) \subseteq C_{i+1}(A)$ for $i = 1, \ldots, n-1$.

A proof is omitted here but one can be found in [70]. The relevance of Lemma 2.2 to a modified Cholesky algorithm is made clear by the next theorem.

Theorem 2.3 *Suppose that at each iteration of the modified Cholesky factorization as described in the previous section, the remaining submatrix A_j still to be factorized is as in (2.1) and the next iterate A_{j+1} is given by (2.3). Let $e_j = \max\{0, \|b_j\|_1 - \alpha_j\}$ and $E = \text{diag}(e_j)$. Then $A + E$ is positive semidefinite and*

$$\|E\|_\infty \leq \gamma + (n-1)\xi, \tag{2.8}$$

where ξ is the maximum (in magnitude) off-diagonal element of A and γ the corresponding diagonal maximum of A. Furthermore, if any diagonal pivoting stategy is employed at each iteration, then the bound (2.8) remains true.

Again, a proof of this theorem is omitted but can be found in [70].

In actuality, the SE algorithm does not compute the e_j as in Theorem 2.3. Instead they are calculated from

$$e_j = \max\{0, \|b_j\|_1 - \alpha_j, e_{j-1}\}, \tag{2.9}$$

This modification is to ensure that the e_j are a nondecreasing sequence. The motivation for this is that if $\max\{0, \|b_j\|_1 - \alpha_j\} < e_{j-1}$ then using (2.9) does not change $\|E\|_\infty$ at this stage of the factorization but could result in subsequent e_j being smaller and therefore may potentially reduce the value of $\|E\|_\infty$ [70]. It should be noted however that in some applications it has been found that this step is not always necessary and may even impair performance [68].

The SE algorithm uses the following quantity to inform its pivoting strategy. At the jth stage of the factorization, we can define for each row k of the submatrix $\tilde{A}_j$

$$G_k = \tilde{a}_{kk} - \sum_{\substack{i=1 \\ i \neq k}}^{n} |\tilde{a}_{ik}|, \tag{2.10}$$

which is the lower bound of the Gershgorin interval R_k for that row. Rather than selecting the diagonal element with the greatest absolute value as pivot, as in the GMW algorithm, we choose the pivot to be the diagonal element from the row such that G_k is maximized over all $k = j+1, \ldots, n$. The analysis justifying this choice of pivoting strategy is elucidated in [70]. Usually finding all such pivots throughout the entire factorization would incur a cost of $O(n^3)$ flops, however Schnabel and Eskow were able to show that, through judicious use of Lemma 2.2, it is possible for this to be done in just $2n^2$ flops [70].

The cost of finding the e_j using (2.9) is about $n^2/2$ flops, so the total additional cost of the SE algorithm relative to the standard Cholesky algorithm is only about $5n^2/2$ flops, and therefore Objective 4 is achieved.

The other major difference between the SE and GMW algorithms specifies when a matrix is to be considered to be positive definite and how the algorithm avoids perturbing the matrix if that is the case. The algorithm is divided into two distinct phases: in the first, a standard Cholesky factorization with GMW-like diagonal pivoting (i.e., choosing the largest element in magnitude as the pivot) is performed. However, as soon as it is detected that the next iteration of the standard algorithm would lead to any

one of the diagonal elements in the next submatrix becoming nonpositive (and therefore A not being positive definite), the second phase is initiated and the alternative pivoting strategy described previously is applied until the factorization is complete.

In practice, the algorithm actually moves to the second phase when continuing the first would lead to one or more of the diagonal elements in the next iterate submatrix becoming smaller than a tolerance δ. Schnabel and Eskow suggest that this be chosen as $\delta = \tau\gamma$, where γ is (as defined previously) the largest diagonal element of A (in magnitude) and τ is a constant for which they suggest a value of $u^{1/3}$, where u is the unit roundoff.

It is in the second phase that the algorithm makes sure that the perturbed matrix $A+E$ is strictly positive definite, rather than merely positive semidefinite. By actually finding e_j from

$$e_j = \max\{0, -\alpha_j + \max\{\|b_j\|_1, e_{j-1}\}\}, \tag{2.11}$$

rather than (2.9), the SE algorithm ensures that $A + E$ is strictly positive definite, although this does have the unwanted side effect of slightly increasing the bound (2.8) on $\|E\|_\infty$ to

$$\|E\|_\infty \leq \gamma + (n-1)\xi + \tau\gamma. \tag{2.12}$$

Empirically, Schnabel and Eskow found that by calculating the eigenvalues of the final 2×2 submatrix A_{n-1} they could obtain smaller values of e_{n-1} and e_n (and therefore $\|E\|_\infty$) than using the previous strategy and therefore also incorporated this into the algorithm [70]. However, in practice, particularly for large-scale problems, this may not be worthwhile [68].

A consequence of setting the tolerance level to the suggested value of $\tau\gamma$ is that the SE algorithm will perturb a matrix if its condition number is greater than $1/\tau$, even if would otherwise qualify as positive definite.

For those matrices that are perturbed, the following bound on the condition number of $A + E$ can be established [21],

$$\kappa_2(A+E) = O\left(n^3 4^n \left(\frac{\xi+\gamma}{\delta}\right)\right). \tag{2.13}$$

As with the GMW algorithm, this is exponential in n and therefore would seem to be of little interest (Schnabel and Eskow do not even explicitly state it in [70]), but—also

as in the GMW algorithm—in practice it has been found that the condition number of the perturbed matrix is generally much smaller than this. In numerical experiments, Schnabel and Eskow believed they observed a de facto upper bound of about $1/\tau$ [70] and so suggested that Objective 3 is usually met.

With regards to Objective 1, Fang and O'Leary in [21] conclude that no perturbation is made (i.e., $E = 0$) if

$$\lambda_{\min}(A) \geq \frac{1}{2}n(n+1)\delta.$$

Note that this means the conditions under which perturbations are actually made may vary considerably from the GMW algorithm, particularly if n is large.

Schnabel and Eskow conclude their consideration of the success of their algorithm in meeting Objective 2 with the following theorem.

Theorem 2.4 *Suppose we apply the SE algorithm to a symmetric matrix $A \in \mathbb{R}^{n\times n}$, and it stays in phase one for $N \geq 0$ iterations. Let ϕ be the maximum magnitude of the bounds G_k (defined as in (2.10)) of A_{N+1}. If $N = 0$ (i.e., A is sufficiently positive definite), then $E = 0$. Otherwise, E is a diagonal matrix with nonzero entries along the diagonal, and*

$$\|E\|_\infty \leq \phi + \frac{2\tau}{1-\tau}(\phi + \gamma). \tag{2.14}$$

Further, if $N = 0$, then

$$\phi \leq \gamma + (n-1)\xi,$$

and if $N > 0$, then

$$\phi \leq (n - N + 1)(\gamma + \xi).$$

We can see that, no matter what the value of N, we have $\|E\|_\infty = O(n)$ and the SE algorithm achieves Schnabel and Eskow's stated aim of improving upon the bound (2.4) achieved by the GMW algorithm.

The same argument of Cheng and Higham used to show that the GMW algorithm is numerically stable also applies to the SE algorithm [12]. Also like the GMW algorithm, the SE algorithm has been widely used since its publication and is generally considered to have been successful [71].

A block version (see Appendix D) of the SE algorithm was implemented by Daydé in [14].

2.2.2 The SE99 algorithm

Although the SE algorithm has a tighter theoretical bound on $\|E\|$ than the GMW algorithm, it soon became apparent that sometimes the perturbation matrix E that was produced was unacceptably large[1]. In particular, this often appeared to be the case when the matrix A to be factorized was the sum of a large (in norm) positive semidefinite matrix and another, smaller indefinite matrix. With the goal of correcting this issue, the algorithm was revised by the original authors in [71].

The main change to the algorithm is determining exactly when we move into the second phase of the two-phase strategy. Select $\mu \in (0, 1]$[2] and let

$$A_j = \begin{bmatrix} \alpha_j & b_j^T \\ b_j & \bar{A}_j \end{bmatrix} \quad \text{and} \quad B_j = \bar{A}_j - \frac{b_j b_j^T}{\alpha_j},$$

where A_j is the submatrix still to be factorized at the jth stage of the algorithm, after diagonal pivoting has been performed. Then we begin phase two unless both of the inequalities

$$(A_j)_{ii} \geq -\mu\alpha_j \quad \text{and} \quad (B_j)_{ii} \geq -\mu\gamma$$

hold for all valid i, where γ is the greatest diagonal element of A, as before. The effect of imposing both of these conditions is that the algorithm tends to remain in the first phase for longer than the previous version [71].

Several other minor changes are made to the algorithm as a consequence of this modification; for example, the constant τ is now replaced by another constant $\tilde{\tau}$ with a suggested value of $u^{2/3}$, where u is the unit roundoff. An unfortunate result of these changes is that the bound (2.14) is loosened slightly; the quantity ϕ must be replaced by Φ, where Φ is defined by

$$\begin{aligned} \Phi &\leq (n - (k+1))((1+\mu)\gamma + \zeta), && \text{for } N > 0, \\ \Phi &\leq \gamma + (n-1)\zeta, && \text{for } N = 0, \end{aligned}$$

and N is the number of steps that the algorithm completed in the first phase, as before. However, this only increases the bound (2.14) by a factor of at most 1.1 [71], so Objective 2 is still achieved. The cost of the algorithm remains the same as

[1] In fact, an example was even remarked upon in Schnabel and Eskow's paper introducing their original algorithm.

[2] $\mu = 0.1$ is suggested to be an empirically good value [71].

before. The bound (2.13) also remains the same, although different behaviour has been observed in practice: the new algorithm seems to often produce considerably larger condition numbers for some matrices than the old one. Schnabel and Eskow note that since these matrices are usually extremely ill conditioned in the first place, it may in fact be advantageous for the perturbed matrices to retain this property [71].

After computational testing, Schnabel and Eskow ultimately concluded that they were successful in resolving the motivating problem; we will consider this further in Chapter 4.

2.3 Variants of the GMW and SE algorithms

Fang and O'Leary propose a variant of the SE algorithm and two variants of the GMW algorithm in [21] and we will follow their naming conventions as we describe them here.

The GMW-I algorithm incorporates the two-phase strategy of the SE algorithms into the GMW[3]. The first phase is identical to that of the SE99 algorithm, but once the second is initiated, the algorithm proceeds as in the GMW. The advantage of this compared to the original algorithm is that the bound on $\|E\|_2$ is now reduced to $O(n)$, as in the SE algorithms (although it is still larger than them, as noted in [70]). The tolerance δ is suggested to be set as machine precision u and $\mu = 0.75$ is stated to be an empirically good value for that constant, which is defined as in section 2.2.2. Experimentally, Fang and O'Leary also found that by pivoting on the maximum element rather than the maximum element in magnitude, they could reduce the condition number of the perturbed matrix and therefore included this change into the algorithm.

The other GMW variant proposed by Fang and O'Leary also takes inspiration from the SE algorithms in order to reduce the bound on the $\|E\|$ to $O(n)$. The GMW-II algorithm again incorporates the two-phase strategy and the modified diagonal pivoting method as used in the GMW-I algorithm, but also attempts to ensure that the diagonal perturbations e_j, $j = 1, \ldots, n$, are a nondecreasing sequence by choosing them from

$$\alpha_j + e_j = \max\left\{\delta, \alpha_j + e_{j-1}, \frac{\|b_j\|_\infty^2}{\beta^2}\right\}, \tag{2.15}$$

[3]This wasn't a new idea and had actually been considered by Schnabel and Eskow in [70].

rather than (2.2), where α_j, b_j and β are all as defined in section 2.1. It is suggested to take the same value for the tolerance δ as in the SE99 algorithm, $\delta = u^{2/3}\eta$.

Fang and O'Leary call their modified SE algorithm the SE-I algorithm. Three major changes are made to the SE99 algorithm as described in section 2.2.2. We shall only describe one in detail here; a full description of the other changes can be found in [21]. Let N be as in section 2.2, the number of steps the algorithm takes in the first stage. Then instead of (2.11), we find e_j from

$$e_j = \max\{0, -2\alpha_j, -\alpha_j + \max\{\|b_j\|_1, \tilde{\tau}\eta\}\}, \tag{2.16}$$

for $j = N+1, \ldots, n-2$, where $\tilde{\tau}$ is as defined in section 2.2.2. The other changes to the algorithm concern how the final 2×2 submatrix is dealt with and what to do if the algorithm moves into the second phase in the last iteration. The effect of these changes is to roughly half the upper bound (2.14) on the norm of E established for the SE99 algorithm.

Fang and O'Leary performed extensive numerical experimentation with all three of their variant algorithms to determine how well they achieve the four primary objectives of a modified Cholesky algorithm and we discuss the results of this in Chapter 4.

2.4 Others

Wright proposes an algorithm that he refers to as a modified Cholesky algorithm in [76]. Despite the name, the algorithm is intended only for symmetric positive definite matrices that arise in the context of interior-point algorithms in linear programming but which are so ill conditioned that the standard Cholesky factorization can encounter problems during pivoting, so is of little interest to us since we are concerned with potentially indefinite matrices. It may be possible to extend the idea to indefinite matrices but it is not immediately clear how that would be done.

An algorithm for finding what they call the Unconventional Modified Cholesky (UMC) factorization is introduced by Schlick and Xie in [77]. The context is a truncated-Newton method (see Chapter 8) for large-scale optimization problems in chemical applications; sometimes, preconditioners (see section 8.2) derived from the problem are not positive definite when they (ideally) should be. The authors had

experience using the GMW and SE90 algorithms, as well as the modified Cholesky algorithm of Cheng and Higham (see section 3.1.2), but found that all three were capable of modifying the indefinite matrix by too large an amount with resultant unacceptable computational costs. Their method is designed to minimize these costs as much as possible. However, as with the method of Wright, there is a critical issue: it is possible for the perturbed matrix to be indefinite. This proves to be acceptable for their purposes, but clearly not for ours.

The UMC algorithm does have a good record of performance: it is used in TNPACK, a Fortran package for unconstrained optimization problems, with a particular emphasis on those arising from molecular applications in chemistry [67, 69], and TNPACK itself was included in the more widely used molecular simulation program CHARMM[4] [11, 78]. Therefore, it may be worthwhile for it to be investigated whether it is possible to alter the algorithm for more general use as a modified Cholesky factorization method.

[4]Chemistry at HARvard Macromolecular Mechanics.

Chapter 3

Indefinite Factorization Algorithms

It has already been mentioned that the LDL^T factorization of an $n \times n$ symmetric indefinite matrix A may not exist. Perhaps the simplest example of this is the 2×2 matrix $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$. However, if we allow D to be *block* diagonal, with blocks of order 1 or 2, then a factorization of the form $PAP^T = LDL^T$, where P is a permutation matrix, can always be found [6, 12]. This is one of the most useful factorizations of a symmetric indefinite matrix [36, p. 214] and has long been implemented in LINPACK [17], LAPACK [2] and (more recently) as a built-in function in MATLAB [31].

The basic idea behind the algorithms discussed in the first section of this chapter is to compute the symmetric indefinite factorization $PAP^T = LDL^T$, perturb D to make it positive definite and then use this perturbed D to construct a positive definite matrix near the original matrix A[1]. This approach appears to have first been proposed, at least in a rigorous way, by Moré and Sorensen in the late 1970s [47]. However, because of the limitations of the pivoting strategies available at the time to compute the indefinite LDL^T factorization, it was uncertain that their algorithm satisfied Objectives 1–4[2]. Cheng and Higham's 1998 modified Cholesky algorithm attempts to remedy this by utilising a pivoting strategy discovered in the interim, so-called "rook" pivoting [6, 12].

Throughout this chapter (and beyond), we will refer to the algorithm of Cheng and Higham as the CH algorithm and that of Moré and Sorensen as the MS algorithm. We shall focus exclusively on these two examples of their class of modified Cholesky

[1] Cheng and Higham do note in [12] that because D is block diagonal, this is not technically a Cholesky factorization. However, since all the diagonal blocks of D are positive definite, they conclude that it can be justly viewed as one.

[2] In fact, their algorithm predates any explicit formulation of the objectives and was created for a different purpose, but the analysis is the same.

algorithms as they are by far the most prominent [21].

Another useful symmetric matrix factorization is the tridiagonal factorization LTL^T, where T is a tridiagonal matrix. Although not nearly as common in the modified Cholesky literature as the LDL^T factorization, Fang and O'Leary in [21] propose variants of both the CH and MS algorithms that incorporate it. We shall briefly consider this factorization and its application to the modified Cholesky factorization in the final section of this chapter.

3.1 Block diagonal LDL^T factorization

The process of computing an LDL^T factorization of a nonzero symmetric matrix, where D is block diagonal, is described succinctly by Higham in [36, pp. 214–215], and we follow his exposition closely here. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Then we can find a permutation matrix P_1 and an integer $s = 1$ or 2 such that

$$P_1 A P_1^T = \begin{array}{c} \\ s \\ n-s \end{array} \begin{array}{c} \begin{array}{cc} s & n-s \end{array} \\ \begin{bmatrix} E & C^T \\ C & B \end{bmatrix} \end{array},$$

where E is nonsingular. With this P_1, we then compute the factorization

$$P_1 A P_1^T = \begin{bmatrix} I_s & 0 \\ CE^{-1} & I_{n-s} \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & B - CE^{-1}C^T \end{bmatrix} \begin{bmatrix} I_s & E^{-1}C^T \\ 0 & I_{n-s} \end{bmatrix}.$$

This process is then repeated recursively on the $(n-s)\times(n-s)$ submatrix $\tilde{A} = B - CE^{-1}C^T$ (called the *Schur complement*). We continue for k steps until we arrive at the factorization $PAP^T = LDL^T$, where $P = P_1 P_2 \dots P_k$.

The cost of the factorization is the same as that of the standard Cholesky factorization, $n^3/3$ flops. However, we also need to account for the pivoting strategy we employ in order to find the permutation matrices P_i.

3.1.1 Pivoting strategies

There were two obvious pivoting strategies available to Moré and Sorensen: Bunch-Parlett ("complete") [10] and Bunch-Kaufman("partial") [9] pivoting. Both strategies are numerically stable and were widely used, with the Bunch-Kaufman strategy in

particular being used for the LDL^T factorization in both LINPACK and LAPACK since soon after its inception [6]. However, both were problematic in the context of a modified Cholesky algorithm. In order to determine the pivot at each stage of the factorization, the Bunch-Parlett strategy requires searching the entire remaining submatrix $\tilde{A}$, which can cost up to $n^3/6$ total comparisons for an $n \times n$ matrix, and therefore any modified Cholesky algorithm incorporating it would fail to achieve Objective 4. Alternatively, the Bunch-Kaufman pivoting strategy only searches one column at a time, thus requiring at most $O(n^2)$ comparisons and satisfying Objective 4 even in the worst possible case. However, although the algorithm is actually numerically stable, the elements of L are not bounded relative to those of A. The problem with this in the context of a modified Cholesky algorithm is best illustrated by the following example, cited both by Higham in [36, p. 219] and Fang and O'Leary in [21].

Let $\epsilon > 0$. Then the symmetric matrix

$$A = \begin{bmatrix} 0 & \epsilon & 0 \\ \epsilon & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

has the LDL^T factorization

$$A = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 1/\epsilon & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \epsilon & \\ \epsilon & 0 & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1/\epsilon \\ & 1 & 0 \\ & & 1 \end{bmatrix},$$

when we use the Bunch-Kaufman pivoting strategy. We can see that the elements of L are not bounded as $\epsilon \to \infty$. This causes problems, in particular, because the elements of the perturbation matrix E that we obtain from either of the two modified Cholesky algorithms that we describe in the following section are also unbounded (and therefore so is its norm).

Two alternative pivoting strategies were proposed by Ashcraft, Grimes and Lewis in [6]. One is a variant of Bunch-Kaufman and the other a variant of Bunch-Parlett. Both are numerically stable [6]. The bounded Bunch-Kaufman or "rook"[3] pivoting strategy is described as follows. At the first stage of the LDL^T factorization, the

[3]The origin of the name should hopefully be clear to those familiar with the game of chess once we come to describe the strategy.

pivot—either 1×1 or 2×2—is chosen according to the following algorithm [12] and the others are later chosen likewise.

Algorithm 3.1: Selects the pivot for the first stage of the symmetric indefinite LDL^T factorization of a symmetric matrix $A \in \mathbb{R}^{n \times n}$ according to the rook pivoting strategy of Ashcraft, Grimes and Lewis

```
Let α = (1 + √17)/8 (≈ 0.64)
Let ω_1 be the maximum magnitude of any subdiagonal element in column 1
if |a_11| ≥ α ω_1 then
    use a_11 as a 1 × 1 pivot
else
    Let i = 1 and ω_i = ω_1
    while no pivot has been chosen do
        Let r be the row index of the first subdiagonal entry of maximum
          magnitude in column i
        ω_r = maximum magnitude of any off-diagonal entry in column r
        if |a_rr| ≥ α ω_r then
            use a_rr as a 1 × 1 pivot (i.e., s = 1 and P_1 swaps rows and columns 1
              and r)
        else if ω_i = ω_r then
            use [a_ii a_ri; a_ri a_rr] as a 2 × 2 pivot (i.e., s = 2 and P_1 swaps rows and
              columns 1 and i, and 2 and r)
        else
            Let i = r and ω_i = ω_r
    end
```

On the surface, the use and choice of the constant α seems obscure, but it is included in an attempt to bound the growth of the elements of L (and therefore prevent the kind of problem illustrated by the example above). Suppose that at any stage of the factorization the pivot has been chosen and the row and column interchanges have been performed. If the pivot was a 1×1 pivot, then we have

$$\tilde{a}_{ij} = b_{ij} - c_{i1}\frac{1}{e_{11}}c_{1j} \implies |\tilde{a}_{ij}| \leq \omega_1 + \frac{\omega_1^2}{\omega_r} \leq \big(1 + \frac{1}{\alpha}\big)\omega_1.$$

If however, a 2×2 pivot was chosen then the (i, j) element of the submatrix is given by

$$\tilde{a}_{ij} = b_{ij} - \begin{bmatrix} c_{i1} & c_{i2} \end{bmatrix} E^{-1} \begin{bmatrix} c_{j1} \\ c_{j2} \end{bmatrix},$$

where

$$E^{-1} = \begin{bmatrix} a_{ii} & a_{ri} \\ a_{ri} & a_{rr} \end{bmatrix}^{-1} = \frac{1}{\det(E)} \begin{bmatrix} a_{rr} & -a_{ri} \\ -a_{ri} & a_{ii} \end{bmatrix}.$$

By the symmetry of E, we have

$$\det(E) = a_{rr}a_{ii} - a_{ri}^2 = a_{rr}a_{ii} - \omega_r^2 \leq \omega_1^2 - \omega_r^2 \leq (\alpha^2 - 1)\omega_r^2.$$

If we assume $\alpha \in (0, 1)$, then we have $|\det(E)| \geq (1 - \alpha^2)\omega_r^2$. Therefore,

$$|E^{-1}| \leq \frac{1}{(1-\alpha^2)\omega_r} \begin{bmatrix} \alpha & 1 \\ 1 & \alpha \end{bmatrix}.$$

Since $|c_{ij}| \leq \omega_r$, we have

$$|\tilde{a}_{ij}| \leq \frac{2(1+\alpha)\omega_r^2}{(1-\alpha^2)\omega_r} = \left(1 + \frac{2}{1-\alpha}\right)\omega_r.$$

To determine α, we equate the maximum growth for two $s = 1$ steps with that for one $s = 2$ step:

$$\left(1 + \frac{1}{\alpha}\right)^2 = 1 + \frac{2}{1-\alpha}.$$

This reduces to the quadratic equation $4\alpha^2 - \alpha - 1 = 0$. We solve this for the positive root to find $\alpha = (1 + \sqrt{17})/8 \approx 0.64$. With this value of α, it can be shown [12] that:

1. The entries of L are bounded above by $\max\{1/(1-\alpha), 1/\alpha\} \approx 2.781$.

2. Each 2×2 pivot block D_{ii} satisfies $\kappa_2(D_{ii}) \leq (1+\alpha)/(1-\alpha) \approx 4.56$.

The bound on the elements of L has the effect of nicely bounding the norm of the matrix L itself in terms of n. Since we know that all the elements are bounded above by 2.781, we have $\|L\|_F^2 \leq n + \frac{1}{2}n(n-1)2.781^2 \leq 4n^2 - 3n$ [12]. This bounding of the elements of L is a major reason why rook pivoting may be preferred to partial pivoting in many contexts, beyond just the modified Cholesky factorization.

Other values of α may be used instead. Fang and O'Leary note in particular that $\alpha = 0.5$ leads to tighter theoretical bounds on the elements of L but in practice seems to actually perform more poorly in the context of the modified Cholesky factorization than the value derived analytically above [21].

The ω_i are a strictly increasing sequence, so the searching part of the algorithm takes at most n steps. Therefore, the total cost of the rook pivoting strategy is intermediate between partial and complete pivoting (i.e., between $O(n^2)$ and $O(n^3)$). Matrices that require searching the entire submatrix at every stage (and therefore cost

the latter) are easy to construct. For example, the following family of matrices is from [6]:

$$\begin{bmatrix} 0 & & & 2 \\ & 4 & 4 & \\ & 4 & 0 & 3 \\ 2 & & 3 & 0 \end{bmatrix}, \begin{bmatrix} 0 & & & & 2 \\ & 5 & 5 & & \\ & 5 & 0 & 4 & \\ & & 4 & 0 & 3 \\ 2 & & & 3 & 0 \end{bmatrix}, \begin{bmatrix} 0 & & & & & 2 \\ & 6 & 6 & & & \\ & 6 & 0 & 5 & & \\ & & 5 & 0 & 4 & \\ & & & 4 & 0 & 3 \\ 2 & & & 0 & 3 & 0 \end{bmatrix}, \ldots$$

The potential $O(n^3)$ cost would seem to suggest that rook pivoting is unsuitable for use in a modified Cholesky algorithm because it would fail to achieve Objective 4. However, numerical experimentation suggests that in practice, the number of comparisons required for a $k \times k$ submatrix is usually less than $5k/2$, and therefore the total number of comparisons required for the full matrix is usually only $O(n^2)$ [6]. Ashcraft, Grimes and Lewis make a probabilistic argument suggesting that the expected number of comparisons for such a submatrix is less than $ek \approx 2.718k$ [6, 12], which again suggests a practical $O(n^2)$ bound. Similarly, Foster considers rook pivoting applied to the factorization of symmetric matrices with independent identically distributed random variables from any continuous probability distribution, both theoretically and through numerical experimentation, and comes to the same conclusion in [23]. Cheng and Higham—who chose rook pivoting as the default pivoting strategy for their modified Cholesky algorithm—therefore considered it to be an acceptable choice and regarded failure to achieve Objective 4 as unlikely to occur in practice [12].

Neal and Poole present empirical evidence in [55] that rook pivoting consistently produces more accurate solutions to problems than Bunch-Kaufman pivoting, whilst generally having a similar cost in practice.

It should be noted that Ashcraft, Grimes and Lewis in [6] devote a significant amount of attention to the application of rook pivoting to sparse matrices. For many applications involving dense matrices—although not the modified Cholesky factorization because of the problem already identified—there is little incentive to choose rook pivoting over partial pivoting since the latter is cheaper. However, in certain applications involving sparse matrices, rook pivoting may preserve sparsity where partial pivoting does not and therefore may be preferred [18, section 5.2.2].

Software implementations of rook pivoting are increasingly common. It is used in the built-in `ldl` function in MATLAB [31, p. 144, 252], which is itself built on the LAPACK routine `DSYTRF_RK`; we discuss the latter further in section 4.2.

The other pivoting strategy introduced by Ashcraft, Grimes and Lewis in [6] is the "fast" Bunch-Parlett strategy. This is also intermediate in cost between complete and partial pivoting (i.e., between $O(n^2)$ and $O(n^3)$ comparisons). It is very similar to rook pivoting but has proven to be less immediately popular and therefore we will not describe it here. It may however be more efficient when implemented in block form [6, 12].

3.1.2 The Cheng and Higham (CH) algorithm

The basic approach of this algorithm has already been described at the start of this chapter: compute the factorization $PAP^T = LDL^T$ and then perturb D to make it positive definite. In order to achieve the objectives of a modified Cholesky algorithm as far as possible, Cheng and Higham recommend using rook pivoting to find the indefinite factorization of A, although they do note that similar alternatives are permitted. One of these is the fast Bunch-Parlett pivoting strategy already mentioned; they also suggest that any of those described in [19] and [20] would be suitable.

After the factorization has been found we perturb the symmetric indefinite block diagonal matrix D. To do this, we first consider the following. Let $\delta \geq 0$ and define $\mu_X(A, \delta)$ as the minimum distance from the $n \times n$ symmetric matrix A to the set of all symmetric matrices with minimum eigenvalue δ, in the X-norm, i.e.,

$$\mu_X(A, \delta) = \min\{\|\Delta A\|_X : \lambda_{\min}(A + \Delta A) \geq \delta\}.$$

Then the following theorem of Higham [12, 34] tells us the values of μ_X in both the 2-norm and the Frobenius norm and also the perturbation matrices that give them.

Theorem 3.1 *Let the symmetric matrix $A \in \mathbb{R}^{n\times n}$ have the spectral decomposition $A = Q\Lambda Q^T$, where Q is orthogonal and $\Lambda = \operatorname{diag}(\lambda_i)$. Then, for the Frobenius norm,*

$$\mu_F(A, \delta) = \left(\sum_{\lambda_i < \delta} (\delta - \lambda_i)^2 \right)^2$$

and there is a unique optimal perturbation given by

$$\Delta A = Q\mathrm{diag}(\tau_i)Q^T, \text{ where } \tau_i = \begin{cases} 0, & \lambda_i \geq \delta, \\ \delta - \lambda_i, & \lambda_i < \delta. \end{cases}$$

For the 2*-norm,*

$$\mu_2(A, \delta) = \max\{0, \delta - \lambda_{\min}(A)\},$$

and an optimal perturbation is $\Delta A = \mu_2(A, \delta)I$*. The Frobenius norm perturbation is also optimal in the* 2*-norm.*

We can now summarise the CH algorithm in two steps, given a symmetric matrix A and tolerance δ:

1. Use rook pivoting to find a symmetric indefinite factorization $PAP^T = L\tilde{D}L^T$.

2. Calculate the minimum perturbation in the Frobenius norm $\Delta\tilde{D}$ such that $\lambda_{\min}(\tilde{D} + \Delta\tilde{D}) \geq \delta$ using Theorem 3.1. Let $D = \tilde{D} + \Delta\tilde{D}$. Then we have $P(A + E)P^T = LDL^T$, where $\lambda_{\min}(A + E) \geq \delta$ (and so in particular $A + E$ is positive definite if $\delta > 0$).

Note that the algorithm does not compute E explicitly, unlike the diagonal perturbation algorithms from Chapter 2. Cheng and Higham suggest using $\delta = \sqrt{u}\,\|A\|_\infty$ for the tolerance, where u is the unit roundoff.

To make the perturbations in the second step and therefore calculate the matrix D, we simply iterate over the diagonal blocks of $\tilde{D}$. If we have a 1×1 block $\tilde{d}_i$, then we set $d_i = \max\{\delta, \tilde{d}_i\}$. If instead we have a 2×2 block $\tilde{D}_i$, then we find its spectral decomposition $\tilde{D}_i = U\mathrm{diag}(\tilde{\lambda}_1, \tilde{\lambda}_2)U^T$, where the $\tilde{\lambda}_j$ ($j = 1$ or 2) are the eigenvalues, and let $D_i = U\mathrm{diag}(\lambda_1, \lambda_2)U^T$, where $\lambda_j = \max\{\delta, \tilde{\lambda}_j\}$. The cost of making these modifications is relatively small so the overall cost of the algorithm to highest order terms is the just the cost of the first step, which we have already argued will usually satisfy Objective 4.

Assuming that the pivoting strategy used in the factorization is backward stable—and all those that have been discussed so far are [6, 9, 10]—then the CH algorithm itself (and also the MS algorithm from the next section) is also backward stable (and therefore numerically stable). In the presence of rounding error, the computed factors

L and D are the exact factors not of $P(A+E)P^T$ but of $P(A+E+F)P^T$, where

$$\|F\|_2 \leq c_n u \|A+E\|_2 , \tag{3.1}$$

$$\leq c_n u \|L\|_2 \|D\|_2 \left\|L^T\right\|_2 , \tag{3.2}$$

with c_n a modest constant depending on n and u the unit roundoff [12].

Cheng and Higham considered the success of their algorithm in achieving Objectives 1–3 in [12] and we shall summarise their results here. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix with eigenvalues $\lambda_n \leq \cdots \leq \lambda_1$. By a theorem of Ostrowski [12], we have that if $\lambda_{\min}(A) \geq 0$ (i.e., A is at least positive semidefinite), then E is zero if

$$\lambda_{\min}(A) \geq \delta \lambda_{\max}(LL^T).$$

Alternatively, if $\lambda_{\min}(A) < 0$ then we have

$$\|E\|_2 = \lambda_{\max}(E) \leq \lambda_{\max}(LL^T)\left(\delta - \frac{\lambda_{\min}(A)}{\lambda_{\min}(LL^T)}\right). \tag{3.3}$$

The following bound can be established on the condition number of the perturbed matrix $A+E$,

$$\kappa_2(A+E) \leq \kappa_2(LL^T) \max\left(1, \frac{\lambda_{\max}(A)}{\lambda_{\min}(LL^T)\delta}\right). \tag{3.4}$$

We can see that the success of the algorithm in satisfying Objectives 1–3 depends on the quantities $\lambda_{\min}(LL^T)$ and $\lambda_{\max}(LL^T)$ (and therefore $\kappa_2(LL^T)$). Ultimately, Cheng and Higham conclude that if L is well conditioned then their algorithm is guaranteed to perform well [12]; however, Schnabel and Eskow comment that if it isn't then the bound on $\|E\|$ is weak [71].

Cheng and Higham suggest that a significant advantage of their algorithm compared to others is that it can easily take advantage of any existing implementation of a symmetric indefinite LDL^T factorization [12]. From the perspective of choosing a modified Cholesky algorithm for the NAG Library, this seems to us a strong argument, particularly since the preferred rook pivoting strategy has recently been included in the library.

3.1.3 The Moré and Sorensen (MS) algorithm

Like the GMW and SE algorithms, the MS algorithm arose as a way to find descent directions from indefinite Hessian matrices in optimization (see Chapter 8) [47].

The algorithm is fundamentally very similar to the CH algorithm, in that we compute an LDL^T factorization of the indefinite matrix A and then perturb the result. As previously noted, at the time of publication, Moré and Sorensen considered only the Bunch-Kaufman and Bunch-Parlett pivoting strategies for computing the LDL^T factorization, ultimately suggesting use of the latter because of the problems already noted when the former is used in a modified Cholesky algorithm. As a consequence of this, Objective 4 would not be met and the algorithm would not be worthy of consideration here. However, the algorithm can be very easily adapted to use rook pivoting (or a similar alternative) instead and therefore remain viable [21].

The MS algorithm has the same basic two step structure as the CH algorithm described in section 3.1.2, but the perturbations made in the second step differ. We also take the same approach to calculate D as before in that we iterate over the diagonal blocks of $\tilde{D}$. However, in the MS algorithm, if we have a 1×1 block $\tilde{d}_i$, then we set $d_i = \max\{\delta, |\tilde{d}_i|\}$ and if we have a 2×2 block $\tilde{D}_i$, then we find the decomposition $\tilde{D}_i = U\text{diag}(\tilde{\lambda}_1, \tilde{\lambda}_2)U^T$, where the $\tilde{\lambda}_j$ ($j = 1$ or 2) are the eigenvalues, and set $D_i = U\text{diag}(\lambda_1, \lambda_2)U^T$, where $\lambda_j = \max\{\delta, |\tilde{\lambda}_j|\}$. Again, we see that the MS algorithm will satisfy Objective 4, if we ignore the worst case cost of rook pivoting. Note that we do not not necessarily use the same value of δ as for the CH algorithm: Moré and Sorensen suggest taking $\delta = u$ [47], where u is the unit roundoff.

It is not remarked upon by Moré and Sorensen but a connection between their algorithm and another decomposition of the indefinite matrix A can be made. For any square real matrix A, the *polar decomposition* is of the form $A = UH$, where U is an orthogonal matrix and H is symmetric positive semidefinite. This factorization always exists and is unique so long as A is not singular. Given the spectral decomposition $A = Q\text{diag}(\lambda_i)Q^T$, the factor H is given explicitly by $H = Q\text{diag}(|\lambda_i|)Q^T$. In particular, this implies that a nearest positive semidefinite matrix to A in the 2-norm is given by $(A + H)/2$. Further, a good approximation to the nearest positive semidefinite matrix to A is given by H itself [33]. When $\delta = 0$, the perturbed matrix $A + E$ produced by the MS algorithm is in fact the positive semidefinite factor H from the polar decomposition of A.

No explicit consideration of the success of the MS algorithm in meeting Objectives 1–3 is given in [47], however Fang and O'Leary perform the analysis in [21] and we

Table 3.1: Objectives 1–3 considered for the MS algorithm.

Objective 1:	$E = 0$ for $\lambda_{\min}(A) \geq \delta \left\|LL^T\right\|_2$
Objective 2:	$\|E\|_2 \leq -2\lambda_{\min}(A)\kappa_2(LL^T)$:
Objective 3	$\kappa_2(A+E) \leq \kappa_2(LL^T)^2\kappa_2(A)$

summarise their findings in Table 3.1. From the table, we see that the smallest and largest eigenvalues of LL^T are important here, as with the corresponding results for the CH algorithm. The bound for $\|E\|_2$ is approximately twice as loose as for the CH algorithm. Fang and O'Leary in [21] state that the bound on the conditioning of $A+E$ is generally superior to the CH algorithm, although our analysis in section 4.1.3 does not necessarily support this conclusion. We shall compare the practical performance of both algorithms in Chapter 4.

3.2 The LTL^T factorization

Other than the LDL^T factorization, one of the most useful factorizations of a symmetric indefinite matrix A is of the form $A = LTL^T$, where T is a tridiagonal matrix. Two methods of achieving this factorization were discovered independently in the early 1970s by Aasen [1], and Parlett and Reid [64]. Both versions are numerically stable [1, 64] but the latter's method costs about twice that of the standard Cholesky factorization and is therefore too expensive for use in a modified Cholesky algorithm. Aasen's method however only costs about the same [1] and so may be suitable.

A pleasing property of this factorization is that the absolute values of all the entries of the triangular matrix L are bounded above by one [6] which can often be desirable [36, p. 224], but the key fact from our perspective is that, if $A = LTL^T$, then A is positive definite if, and only if, T is also positive definite. Hence Fang and O'Leary in [21] propose the following variant of both the CH and MS algorithms: find the factorization $PAP^T = LTL^T$ and then apply either the CH or MS algorithms to the tridiagonal matrix T [21]. The motivation here is that since T only has at most two nonzero off-diagonal elements per row, the cost of applying either the CH or MS algorithms to it is negligible and therefore the total cost of the algorithm (to highest order terms) is just the cost of the initial factorization using Aasen's method, which is

guaranteed to be the same as the standard Cholesky factorization. This ensures that Objective 4 is met.

The introduction of the LTL^T factorization does have the effect of loosening the bounds on both $\|E\|$ and $\kappa_2(A+E)$ given in the previous two sections for both the CH and MS algorithms, although Fang and O'Leary suggest that in practice their variants generally achieve smaller values for both than the original algorithms [21]; this claim is considered in Chapter 4.

It is also possible to apply any of the diagonal perturbation algorithms from Chapter 2 to the tridiagonal matrix T instead of the CH or MS. However, since they generally meet Objective 4 anyway, there does not appear to be any particular advantage in doing so.

Given the properties of the LTL^T factorization (and Aasen's method in particular) described, practical implementations are surprisingly rare compared to the block LDL factorization. Higham in [36, p. 224] states that he is only aware of one software library containing Aasen's method and ponders why this is so, speculating that it may simply have proven easier to work with a block diagonal matrix than a tridiagonal one. Anderson and Dongarra compare Aasen's method to the Bunch-Kaufman factorization in [3], concluding after experimentation on a Cray 2 machine that the latter leads to superior performance when implemented in block form; however, it appears that the codes used have been lost [6]. Ashcraft, Grimes and Lewis believe the deciding factor between the two factorizations to be speed, although they do note that the data available to them are unclear on this point [6].

Rozložník, Shklarski and Toledo present a blocked algorithm—a modified version of Aasen's method—for computing an LTL^T factorization of a symmetric matrix in [65]. They compare the performance of an implementation of their algorithm with an existing efficient blocked version of the Bunch-Kaufman factorization and find no significant difference in performance, noting in particular that their results do not concur with those of Anderson and Dongarra already mentioned. Another blocked variant of Aasen's method is proposed in [8] and the performance of an implementation on modern (parallel) computer architectures considered in [7]; ultimately it is concluded that the algorithm is competitive with existing indefinite factorization methods.

Although Higham was writing in 2002 when he stated that he believed Aasen's

method had only been included in one software library, implementations still remain relatively uncommon. It has however recently been introduced into LAPACK [60].

Chapter 4

Selecting a Modified Cholesky Algorithm for the NAG Library

As stated in Chapter 1, one of the primary aims of this dissertation is to decide which of the existing modified Cholesky algorithms should be included in the NAG library. In this chapter, we collate all the theoretical expectations of the algorithms discussed in the previous two chapters, as well as data from numerical experimentation and practical applications, and ultimately make that decision.

The four most prominent modified Cholesky algorithms are the GMW, SE, CH and MS algorithms [21]. We shall focus our consideration on these and their variants discussed in Chapters 2 and 3. We do not believe either of the truly distinct algorithms discussed in section 2.4 merit further consideration. We will generally distinguish between the SE90 and SE99 algorithms here because the latter is a significant revision of the former by the original authors. Variants of the other algorithms will be discussed, but not distinguished so fundamentally. For example, when we refer to the "GMW" it may be a synecdoche both for the original algorithm and the GMW-I and GMW-II variants discussed in section 2.3; similarly, when we refer to the CH and MS algorithms we may mean both the originals and the LTL^T variants mentioned in section 3.2. The context should make clear which is meant and we will distinguish where necessary.

We will consider each of the four primary Objectives of a modified Cholesky algorithm in turn and compare the success of the prospective algorithms at ensuring they are met. We will then discuss any other pertinent information, before ultimately making our conclusions (and decision) in the final section.

Table 4.1: Suggested tolerances δ and the conditions for which matrices are not perturbed for the most prominent modified Cholesky algorithms.

Algorithm	**Tolerance** δ	$E = 0$ **if...**
GMW	u	$\lambda_{\min}(A) \geq \delta$
SE90	$\tau\gamma = u^{1/3}\gamma$	$\lambda_{\min}(A) \geq \frac{1}{2}n(n+1)\delta$
SE99	$\tilde{\tau}\gamma = u^{2/3}\gamma$	$\lambda_{\min}(A) \geq \frac{1}{2}n(n+1)\delta$
MS	u	$\lambda_{\min}(A) \geq \delta \left\|LL^T\right\|_2$
CH	$\sqrt{u}\,\|A\|_\infty$	$\lambda_{\min}(A) \geq \delta \left\|LL^T\right\|_2$

4.1 Achieving the objectives

4.1.1 Objective 1

Under what precise circumstances a matrix qualifies as sufficiently positive definite varies for each of the algorithms. They all include a tolerance δ used to make this determination. Table 4.1 records the suggested values for δ and the conditions under which no perturbation is made for each of the algorithms. Here, γ is the largest diagonal element of the matrix A (in magnitude), as in Chapter 2 and u is the unit roundoff.

The conditions under which the algorithms actually perturb the matrix being factorized are all clear and well-defined. The GMW is the most straightforward, only ever perturbing a matrix if it is—at least practically—positive definite. All of the other algorithms do permit the possibility of perturbing a positive definite matrix, although it could be argued that this is sometimes advantageous [70]. It has been found in applications that the different conditions for perturbation can lead to differing performance; see [68] for an example when this is the case for the GMW and SE90 algorithms. Overall, however, we believe that all of the algorithms generally achieve this objective and that there is little difference between them in this regard.

In their experiments with random distance matrix completion problems in [22], Fang and O'Leary believed that the suggested tolerance parameter $\delta = \sqrt{u}\,\|A\|_\infty$ for the CH algorithm was too small for their purpose and used the tolerance $\tau\gamma$ suggested for the SE90 algorithm instead [21].

Table 4.2: Bounds on the size of the perturbation matrix E for the most prominent modified Cholesky algorithms.

Algorithm	Upper bound
GMW	$\|E\|_\infty \leq O(n^2)$
SE90	$\|E\|_\infty \leq \phi + \frac{2\tau}{1-\tau}(\phi+\gamma)$
SE99	$\|E\|_\infty \leq \Phi + \frac{2\tau}{1-\tau}(\Phi+\gamma)$
MS	$\|E\|_2 \leq -2\lambda_{\min}(A)\kappa_2(LL^T)$
CH	$\|E\|_2 \leq \lambda_{\max}(LL^T)\left(\delta - \frac{\lambda_{\min}(A)}{\lambda_{\min}(LL^T)}\right)$

4.1.2 Objective 2

In the previous two chapters we stated the established upper bounds on the norm of the perturbation matrix E for each of the algorithms under consideration and these are collected in Table 4.2. Here, ϕ and Φ are as defined in section 2.2.1. Note also that we do not give the upper bound for the GMW explicitly here but simply state its order.

Considering just the diagonal perturbation methods, the upper bound for the GMW algorithm is approximately n^2 multiplied by the maximum element in A, whereas for the SE algorithms it is only about $2n$ times the largest element in A [68], so can be considerably smaller. The bound for the SE99 algorithm is slightly looser than the original but, as described in Chapter 2, only by a small amount. The two indefinite LDL^T factorization algorithms have bounds defined in terms of the smallest and largest eigenvalues (and therefore the conditioning) of LL^T, and the smallest eigenvalue of A; as previously noted, this has the advantage of ensuring a good bound if L is well conditioned and the downside of only establishing a weak bound if it isn't. Of the two, the bound for the CH algorithm is about half that of the MS algorithm [21]. Ultimately, we believe that all four algorithms have acceptable theoretical limits on the size of E.

After introducing their algorithm in [70], Schnabel and Eskow performed numerical experiments to compare its performance to both their expectations and the GMW algorithm. The natural quantity to consider when evaluating the success of any algorithm in achieving Objective 2 is the ratio of the norm of E to the norm of the minimal perturbation to A that makes it positive semidefinite. In particular, Schnabel

and Eskow considered $r_\infty = \|E\|_\infty / |\lambda_{\min}(A)|$, which they call the "relative maxadd", although we follow the later convention of Cheng and Higham in using r_∞. Schnabel and Eskow favoured the ∞-norm for measuring the success of this objective, although since the perturbation matrices they are considering are diagonal, the 1- and 2-norms are the same. They tested both their algorithm and the GMW on a collection of 90 test matrices of dimension $n = 25$, 50 or 75, with eigenvalues in the ranges $[-1, 10^4]$, $[-1, 1]$ and $[-10^4, -1]$; ten for each combination of dimension and eigenvalue range. Each test matrix was of the form $Q_1Q_2Q_3D(Q_1Q_2Q_3)^T$, where each Q_i is a *Householder matrix* of the form $Q_i = I - \frac{2ww^T}{w^Tw}$, where $w \in \mathbb{R}^n$ is a vector with components randomly generated from a uniform distribution in the range $[-1, 1]$ and D is a diagonal matrix with diagonal elements likewise randomly generated from a uniform distribution in the desired eigenvalue range. Note that in the $[-1, 10^4]$ case, steps were taken to ensure that at least one element of D was negative and therefore the test matrix was truly indefinite.

Schnabel and Eskow found that the their algorithm produced values of r_∞ between 1.06 and 2.5 for the 90 test matrices, compared to a range of 1.6 to 77.8 for the GMW algorithm, which they also found performed particularly poorly on matrices with eigenvalues in the range $[-1, 1]$. Comparing on a matrix-by-matrix basis for each of the matrices in the test set, the SE algorithm produced perturbation matrices 1.3 to 60.9 times smaller than the GMW (and in particular 3.5 to 60.9 times smaller for those matrices with eigenvalues in $[-1, 1]$).

Interestingly, Schnabel and Eskow found that when they performed further experimentation on 30 matrices of dimension 25 with eigenvalues in the range $[-1, 1]$ but this time ensured that at least three eigenvalues were negative, for one particular matrix the GMW achieved a significantly smaller value of $\|E\|_\infty$ than the SE algorithm. This was the only one of the 120 matrices they tested for which this was the case. They established that the problem came down to a 4×4 submatrix and created the following similar 4×4 matrix that illustrated the issues even more clearly:

$$A = \begin{bmatrix} 1890.3 & -1705.6 & -315.8 & 3000.3 \\ -1705.6 & 1538.3 & 284.9 & -2706.6 \\ -315.8 & 284.9 & 52.5 & -501.2 \\ 3000.3 & -2706.6 & -501.2 & 4760.8 \end{bmatrix}. \tag{4.1}$$

The eigenvalues of this matrix are -0.378, -0.343, -0.248 and 8242.869. The difference in size between the perturbations produced by the SE90 and GMW algorithms was stark, with the E from the latter being almost a thousand times smaller than that from the former; clearly a worrying disparity. Schnabel and Eskow deduced that the problem was the one discussed in section 2.2.1 and suggested that applications would reveal whether it was truly necessary to correct it (ultimately of course deciding that it was in [71]). Nevertheless, they conclude that overall their experiments suggested that the SE90 algorithm is generally successful in achieving a smaller E than the GMW algorithm in practice.

After revising their algorithm in [71], Schnabel and Eskow repeated their numerical experiments using the same set of test matrices as in [70]. They found that the size of the perturbation produced by the SE99 algorithm was identical to the SE90 for the matrices with eigenvalues in the ranges $[-1, 1]$ and $[-10^4, -1]$ but was usually much smaller for those matrices with eigenvalues in the range $[-1, 10^4]$, for which it also generally produced a smaller perturbation than the GMW algorithm, albeit only slightly.

Concerning the problematic matrices that motivated the revision of the algorithm, further experiments with test matrices of dimension $n = 25$, 50 or 75 with eigenvalues in the range $[-1, 10^4]$ and with either 3 or 9 negative eigenvalues, produced significantly smaller values of r_∞ than before and which were almost always smaller than the GMW as well. Schnabel and Eskow also considered another set of 33 indefinite Hessian matrices, provided by Gay, Overton and Wright, that arose from barrier methods in constrained optimization, for which the SE90 algorithm had proven problematic [24]. The dimension of these matrices was generally small, with the largest having dimension 55 and the majority being much smaller than that. For these matrices too they found the same pattern: the SE99 algorithm produced much smaller $\|E\|_\infty$ than the SE90 (by up to seven order of magnitude) and slightly smaller than the GMW algorithm. For the particular matrix (4.1), they found that the SE99 had $\|E\|_\infty = 1.76$, as opposed to 2.73 for the GMW and 2778 for the SE90 algorithm. They concluded that their new algorithm was generally more successful at achieving Objective 2 than the previous one and that it successfully corrects the known issues.

In order to determine the success of their algorithm in achieving this objective,

Cheng and Higham in [12] defined the quantities

$$r_F = \frac{\|E\|_F}{\mu_F(A,\delta)} \quad \text{and} \quad r_2 = \frac{\|E\|_2}{|\lambda_{\min}(A)|}, \tag{4.2}$$

where $\mu_F(A,\delta)$ is, as defined in Chapter 3, the minimal distance from A to the set of matrices with smallest eigenvalue greater than or equal to δ (and therefore in particular is the distance to the set of positive semidefinite matrices when $\delta = 0$) in the Frobenius norm. Note that when E is diagonal (as in the GMW and SE algorithms), $r_2 = r_\infty$.

To assess the practical performance of their new algorithm, Cheng and Higham then ran a very similar set of numerical experiments to those undertaken by Schnabel and Eskow in [70] that have previously been described. However, their test matrices were instead of the form $Q\Lambda Q$, where $\Lambda = \text{diag}(\lambda_i)$, with the λ_i being chosen from one of the same three random uniform distributions used by Schnabel and Eskow, and Q being a random orthogonal matrix generated using the `qmult` function from the Matrix Computation Toolbox of Higham [32]. Note that `qmult` can now be called from MATLAB using the built-in `gallery` function for generating test matrices [31, Chapter 6]. The matrices Cheng and Higham tested were of dimension $n = 25$, 50 or 100, and 30 different matrices were generated for each eigenvalue distribution and dimension, giving a total of 270 matrices considered. For each matrix, the quantities r_F and r_2 were calculated for the CH, GMW and SE90 algorithms. Three nonrandom matrices from the Matrix Computation Toolbox were also considered: `Clement`, `Dingdong` and `Ipjfact`. Details of these matrices can be found in [12] or [32]. Alternatively, all of these but `Dingdong` are now included as test matrices in MATLAB and can be constructed using the `gallery` function [31, Chapter 6].

Ultimately Cheng and Higham concluded that none of the three algorithms they considered was uniformly superior to the others at achieving this objective. It appeared that the SE90 algorithm generally produced smaller r_2 for matrices with eigenvalues in the ranges $[-1, 10^4]$ and $[-1, 1]$ than the others, no matter what the dimension, although for r_F the CH algorithm was generally slightly smaller for matrices with eigenvalues in the range $[-1, 1]$. The CH algorithm was also particularly successful at minimizing the ratios r_2 and r_F when the matrix was negative definite (i.e., the eigenvalues were in the range $[-10^4, -1]$), with both generally being extremely close to one. Cheng and Higham elucidate why this is to be expected as follows. Let A be

Table 4.3: Measures of the size of the perturbation made to the matrix A from (4.1) for the most prominent modified Cholesky algorithms.

	CH	**MS**	**MS**	**SE90**	**SE99**
r_F	1.3	2.7	2.7	3.7×10^3	1.8
r_2	1.7	3.3	2.7	2.8×10^3	1.8

negative definite. Then the CH algorithm computes the factorization $P(A+E)P^T = L(\delta I)L^T$. Therefore

$$\begin{aligned} r_F &= \frac{\|E\|_F}{(\sum_i(\delta-\lambda_i)^2)^{1/2}} \\ &\leq \frac{\|E\|_F}{\|A\|_F} = \frac{\left\|A-\delta\cdot P^TLL^TP\right\|_F}{\|A\|_F} \\ &\leq \frac{\|A\|_F+\delta\left\|LL^T\right\|_F}{\|A\|_F} \\ &\leq 1+\frac{(4n^2-3n)\delta}{\|A\|_F}, \end{aligned}$$

so r_F can only ever be slightly larger than 1. A similar analysis applies for r_2.

Unlike Schnabel and Eskow, and Cheng and Higham, Fang and O'Leary in [21] performed numerous numerical experiments with all of the algorithms we are considering, including the MS and the SE99, as well as the three variants of the GMW and SE algorithms discussed in section 2.3 and the LTL^T variants of the CH and MS from section 3.2. Like Cheng and Higham, they also use the ratios r_2 and r_F to measure the success of an algorithm in meeting the objective. For the matrix (4.1) identified by Schnabel and Eskow as causing difficulty for the SE90 algorithm, Table 4.3 records the values of the two ratios for the algorithms we are considering.

The indefinite Hessian matrices produced by Gay, Overton and Wright from [24] and tested by Schnabel and Eskow in [71] were also considered. For this set, the SE99 was by far the most successful of the original algorithms at producing the smallest r_2, doing so for almost two-thirds of the matrices; the CH algorithm achieved the smallest r_2 for the bulk of the others. The GMW and MS algorithms also mostly performed well, but the SE90 did not (which is to be expected since these were precisely the matrices on which it was found to struggle); however, the CH algorithm also produced perturbations an order of magnitude too large on three occasions, and once a perturbation that was two orders of magnitude larger than necessary. It should be

noted that only the values of r_2 were given, not r_F.

Fang and O'Leary also undertook numerical testing with random matrices of the form $Q\Lambda Q^T$, where Q and Λ are as before. Precisely how the random orthogonal matrices Q were generated is not described in detail, although it is stated that the method of Stewart [73] is used, which is the same algorithm used by the `qmult` function. They follow the lead of Cheng and Higham, and Schnabel and Eskow in considering the three different eigenvalue ranges $[-1, 10^4]$, $[-1, 1]$ and $[-10^4, -1]$. Multiple (usually 30) tests were performed, on different sets of matrices, with different dimensions (usually $n = 50$ or 100); for full details, see [21]. Comparing just the diagonal perturbation algorithms, they conclude that the SE99 and their variant GMW-II (described in section 2.3) algorithm are generally the most successful at minimizing E for matrices with eigenvalues in $[-1, 10^4]$ and that the SE-I algorithm is most successful for the other ranges, although the SE99 algorithm in particular also performed well.

Concerning the symmetric indefinite LDL^T algorithms, the CH usually produced a smaller E than the MS algorithm. Fang and O'Leary concluded that their variants incorporating the LTL^T factorization generally outperformed the originals in achieving this objective, however, the originals did usually also perform well, with the few exceptions already mentioned.

Data on how well most of the algorithms we are considering meet Objective 2 in practical applications are also available. Schlick and Xie consider the GMW, SE90 and CH algorithms for dealing with potentially indefinite preconditioners in a Newton-like method for large-scale chemical applications in [77]. They ultimately decide that all three generally perform well but are also capable of producing inordinately large perturbations for some problems.

It should be emphasised at this stage that it has been found in certain applications that a strictly smaller $\|E\|$ does not necessarily give superior performance overall, as acknowledged by Schnabel and Eskow in [71]. For example, Schlick compared the performance of the GMW and SE90 algorithms in a truncated-Newton minimization method used in the context of a large-scale optimization problem in computational chemistry [68]. She found that the SE90 algorithm generally produced smaller perturbation matrices E than the GMW, sometimes by a factor of up to two orders of magnitude, but that the performance of the algorithms was comparable overall.

Table 4.4: Upper bounds on $\kappa_2(A+E)$ for the most prominent modified Cholesky algorithms.

Algorithm	$\kappa_2(A+E)$
GMW	$\leq O\left(n^3\left(\frac{\xi+\gamma}{\delta}\right)^n\right)$
SE90	$\leq O\left(n^3 4^n\left(\frac{\xi+\gamma}{\delta}\right)\right)$
SE99	$\leq O\left(n^3 4^n\left(\frac{\xi+\gamma}{\delta}\right)\right)$
MS	$\leq \kappa_2(LL^T)^2\kappa_2(A)$
CH	$\leq \kappa_2(LL^T)\max\left(1, \frac{\lambda_{\max}(A)}{\lambda_{\min}(LL^T)\delta}\right).$

4.1.3 Objective 3

We collate all the relevant a priori upper bounds on $\kappa_2(A+E)$ established in the previous two chapters in Table 4.4.

The bounds for the diagonal perturbation algorithms are all exponential in n and therefore looser than we would like. The bounds for the MS and CH algorithms have the advantage of being expressed in terms of the eigenvalues of the matrices A and LL^T, and the conditioning of the latter as well. If $\lambda_{\max}(A) < \lambda_{\min}(LL^T)\delta$, then the MS algorithm bound is $\kappa_2(LL^T)\kappa_2(A)$ times the bound for the CH algorithm and therefore will generally be the larger. If this is not the case then the ratio of the bound for the MS algorithm to the bound for the CH algorithm is

$$\begin{aligned}\frac{\kappa_2(LL^T)^2\kappa_2(A)}{\kappa_2(LL^T)\frac{\lambda_{\max}(A)}{\lambda_{\min}(LL^T)\delta}} &= \frac{\kappa_2(LL^T)\kappa_2(A)\lambda_{\min}(LL^T)\delta}{\lambda_{\max}(A)} \\ &= \frac{\lambda_{\max}(LL^T)\delta}{\lambda_{\min}(A)} \\ &\geq \frac{\delta}{\lambda_{\min}(A)}.\end{aligned}$$

Therefore, we see that which of the bounds is the smaller very much depends on both δ and $\lambda_{\min}(A)$.

All of the numerical experiments undertaken by Schnabel and Eskow in [70] and [71], Cheng and Higham in [12], and Fang and O'Leary in [21] described in the previous section also considered the conditioning of the perturbed matrix $A+E$, so we will

discuss their results and conclusions without describing the details of their experiments again.

In [70], Schnabel and Eskow concluded that the SE90 and GMW algorithms both produced acceptably conditioned matrices, with the order of $\kappa_2(A+E)$ ranging from 10^1 to 10^6 for the SE90 algorithm and 10^1 to 10^8 for the GMW algorithm for all of the matrices in their test set; on a matrix-by-matrix basis, they were always within two orders of magnitude of each other. The SE90 algorithm consistently produced better conditioned matrices for matrices with eigenvalues in $[-1, 1]$ and $[-10^4, -1]$, whereas the GMW algorithm was superior for the range $[-1, 10^4]$.

Things were less clear in their follow up experimentation accompanying the introduction of the SE99 algorithm. For the 33 indefinite Hessian matrices of Gay, Overton and Wright, the condition numbers of the perturbed matrices produced by the new algorithm were often significantly higher than both the GMW and SE90, with 13 of them being about 10^9 to 10^{11}, although it is stated that the two highest values for the GMW exceeded this. Schnabel and Eskow do note that the original matrices were extremely ill conditioned and suggest it is important that this property be retained. For the 120 random matrices initially tested in [70], this pattern is also evident, with the SE99 algorithm consistently producing more ill conditioned matrices than the SE90 or GMW algorithms—sometimes by as much as five orders of magnitude—but the GMW algorithm occasionally producing matrices more ill conditioned than either of the others.

Cheng and Higham include the condition numbers $\kappa_2(A+E)$ for 90 matrices of dimension $n = 25$, 30 of each with eigenvalues in the three ranges being considered. Results for $n = 50$ and 100 were stated to be similar. They largely supported the assessment of Schnabel and Eskow from [70], finding that the SE90 algorithm generally produced better conditioned matrices than the GMW algorithm, except for those matrices with eigenvalues in the range $[-1, 10^4]$. Their own algorithm generally produced higher condition numbers than either, for matrices with eigenvalues in all three ranges.

The most extensive testing was done by Fang and O'Leary [21]. Experimentation with random matrices suggested that the MS algorithm is usually better conditioned than the CH algorithm, except for matrices with eigenvalues in $[-10^4, -1]$, for which

they were very similar. Introducing the LTL^T factorization to the CH or MS algorithms as in section 3.2 did not generally improve the conditioning of the resultant matrix. On the set of random matrices described previously, the SE99 algorithm generally produced better conditioned matrices than the CH algorithm, except for matrices with eigenvalues in the range $[-1, 10^4]$. Among the diagonal perturbation variants from section 2.3, the GMW-II algorithm was very successful at achieving a small condition number. For the matrix (4.1), the SE90 algorithm had by far the smallest condition number of all the algorithms we are considering, but the SE99 algorithm had easily the highest (of order 10^{10}). Concerning the 33 matrices of Gay, Overton and Wright, the CH algorithm consistently produced condition numbers of 10^7 or 10^8, with one being order 10^9. The others were much more variable, with the GMW and SE90 algorithms generally the best.

Specific references to the success of the algorithms at meeting Objective 3 in applications are scarce. This could suggest that in practice it often may not be as important as we have supposed.

4.1.4 Objective 4

Both of the diagonal perturbation algorithms meet this objective (see Chapter 2). In some applications the SE algorithm has proven to be the more efficient of the two [15] and in others the GMW algorithm has [68]; the difference may simply come down to the efficiency of the implementation.

The real issue here is the rook pivoting strategy employed to find the indefinite LDL^T factorization used in the CH and MS algorithms. This can add an additional cost of $O(n^3)$ comparisons for certain matrices. The analyses of Ashcraft, Grimes and Lewis [6] and Foster [23] referred to in Chapter 3 suggest that matrices of this form will be rarely encountered in practice. Cheng and Higham recorded the number of comparisons required for the rook pivoting used for each of the matrices they considered in their numerical experiments previously described. We include these results in Table 4.5. As can be seen from the table, the maximum number of comparisons required was always smaller than n^2 and the average was approximately $0.6n^2$.

A pragmatic argument for the practicality of rook pivoting is its continued popularity. For example, as already noted, it is the pivoting strategy used for the built-in

Table 4.5: Number of comparisons required for rook pivoting for the matrices in the test set used by Cheng and Higham in [12].

Dimension	25	50	100
max	523	2188	8811
mean	343.9	1432.8	5998.4

`ldl` function in MATLAB [31]. This does suggest that in many applications at least it has proven to be adequately efficient.

4.2 Other considerations

Implementations of the modified Cholesky factorization in software libraries are scarce. However, the SE90 algorithm was used as a preconditioner in the LANCELOT [13] package for nonlinear optimization problems [71].

From a software engineering perspective, the MS and CH algorithms have the advantage over the others of being extremely easy to implement once the LDL^T factorization has been found. Cheng and Higham created a MATLAB code that implements their algorithm in about twenty lines of code, which can be found at Higham's GitHub page [39], and this author did the same for the algorithm of Moré and Sorensen (provided as the function `more_sorensen` in Appendix E). Given that symmetric LDL^T factorization itself is included in almost all of the major numerical software libraries, this should be considered a major advantage for those algorithms that employ it.

Ashcraft, Grimes and Lewis provide implementations of symmetric LDL^T factorization with rook pivoting that make use of both Level 2 and Level 3 BLAS in [6]. Symmetric indefinite LDL^T factorization with rook pivoting was implemented for LAPACK 3.5.0 as the routine `DSYTRF_ROOK` and later revised for LAPACK 3.7.0 as the routine `DSYTRF_RK` [61]. The latter has been implemented as a NAG Library routine, currently not user-callable but intended to be included with the next major release of the library; we could take advantage of this were we to choose to implement either the CH or MS algorithms.

The GMW and SE algorithms are trickier—although not necessarily difficult in

and of themselves—to translate into effective code. However, Daydé implements a block version of the SE90 algorithm that uses Level 3 BLAS to improve efficiency in [14].

The structure of the MS and CH algorithms also lends itself well to parallel computing. Once the indefinite factorization has been computed, we iterate over the block diagonal matrix D, modifying each 1×1 or 2×2 block in turn; the modifications are all independent, so therefore they can be performed in parallel. The SE and GMW algorithms also have some scope for parallelism, but not to quite the same extent.

A major disadvantage of the MS and CH algorithms compared to the SE and GMW algorithms is that they do not explicitly compute the perturbation matrix E. It can easily be constructed from the computed matrices L and D, but this requires at least one dense matrix multiplication ($LD \cdot L^T$), thus incurring an additional $O(n^3)$ cost. For some applications that require E explicitly, this could well be too expensive.

In terms of storage, for certain large-scale problems the SE algorithm has occasionally been preferred to the GMW because it does not require the full matrix initially at once, but only the diagonal[1]. Note that both the MS and CH algorithms also require the full matrix before they can begin.

So far, we have not distinguished between dense and sparse indefinite matrices. However, it has been found in some applications that modified Cholesky algorithms may exhibit different behaviour depending on the sparsity of the matrix to which they are applied [68]. The CH and MS algorithms have the advantage of being able to utilise the efficient implementation of the symmetric LDL^T factorization with rook pivoting for sparse matrices written by Ashcraft, Grimes and Lewis [6, 12].

It has been found that which modified Cholesky algorithm is most suitable can be highly dependent on the application for which it is used and it is difficult to know beforehand which it will prove to be [68]. It may therefore be wise to consider likely applications of the algorithm before deciding which should be chosen. In particular, for the nearest correlation matrix application that prompted the decision to include the modified Cholesky factorization in the NAG Library, Higham and Strabić conclude after numerical experimentation with the GMW, SE90, SE99 and CH algorithms that

[1] Recall from section 2.1 that the GMW algorithm needs to first calculate ζ, the maximum magnitude off-diagonal element of the matrix, before beginning the factorization.

the CH algorithm is generally the best in this regard [40]. We repeat their experiments with the MS algorithm in Chapter 7 and confirm that this remains so.

In certain contexts matrices with particular ranges of eigenvalues may be more or less likely to occur than others; for example, Fang and O'Leary consider negative definite matrices unlikely to occur in some optimization applications and matrices that are close to positive definite to be particularly common, which would suggest algorithms that perform better in this range would be preferred for those applications. In Chapter 7 we discuss an application of the modified Cholesky factorization for which the matrices in question have often been erroneously believed to be positive definite and so are therefore usually close to it.

4.3 Conclusion

All of the algorithms we are considering are numerically stable (see Chapters 2 and 3), so there is little to distinguish them in that regard. However, the CH and MS algorithms have the advantage over the diagonal perturbation algorithms in that the nature of their upper bounds on the norm of the perturbation matrix and the condition number of the perturbed matrix allow us to estimate the quality of the factorization a priori; it is usually unclear beforehand if the GMW and SE algorithms will perform well [12]. Therefore, we believe that with regards to theoretical expectations of achieving Objectives 1–3 in particular the CH and MS algorithms have the edge over the GMW and SE.

Our conclusions from consideration of the numerical experiments and practical applications detailed in the previous section largely concur with those of Cheng and Higham in [12]: none of the algorithms is uniformly superior to the others and all are capable of experiencing difficulties in some contexts. The SE algorithms—particularly the SE99 and SE-I of Fang and O'Leary—do appear overall to be the best at meeting Objective 2 and ensuring the perturbation made is as small as possible, but the CH algorithm in particular was usually competitive. The conditioning of the perturbed matrix for the MS and CH algorithms was generally more consistent—albeit perhaps consistently high—than for the GMW and SE algorithms, which varied unpredictably.

We make a broader remark at this juncture. There does not appear to be any

particular reason why it is necessary or advantageous to restrict ourselves to diagonal perturbations, in the general case. There may well be applications for which we do not wish to perturb the rest of the matrix, but there is no cause to believe that this will usually be the case.

Since it is difficult to conclude that any one of the algorithms is superior to the others, we believe that ultimately the deciding argument must be the pragmatic software engineering one. Although the SE and GMW algorithms are not difficult to code in and of themselves, the CH and MS algorithms are extremely simple to implement once an LDL^T factorization with rook pivoting has been found—and this is now in the NAG Library. We therefore decided that we should implement either the CH or MS algorithms. Since we also believe that the underlying theory for the CH algorithm is more well-developed than for the MS algorithm, we ultimately chose to implement the former.

Thus far in this section we have not discussed the major potential problem with the indefinite factorization algorithms: the potential $O(n^3)$ comparisons required for the rook pivoting strategy. However, we are satisfied from the analysis and numerical experimentation detailed in the previous sections that this is very unlikely to occur in practice. Of course, we could ensure that the cost is only $O(n^2)$ by incorporating the LTL^T factorization of Aasen, as suggested by Fang and O'Leary, whose numerical experimentation also suggests that this is not harmful and may occasionally even be beneficial. However, we do not believe that there is enough justification to do this by default—particularly since efficient implementations of Aasen's method are relatively rare—but that it should be explored as a possible solution for recalcitrant matrices or those for which there is reason to believe excessive cost is likely. There may well be situations in which we wish to further bound the entries of L, in which case it may also be regarded as a possibility.

Chapter 5

Implementing the Cheng-Higham Algorithm for the NAG Library

In order to fully understand what is required of our implementation of the Cheng-Higham modified Cholesky algorithm, we must first give a little background on NAG itself.

5.1 NAG and the NAG Library

Founded as an academic collaboration between several British universities in the 1970s, NAG[1] is now a software company with expertise in high-performance computing (HPC) and mathematical software. They are most well-known for the NAG Library, a collection of over 1700 routines for solving mathematical problems. This is sold as a commercial product, although NAG itself is a non-profit organisation: all proceeds are re-invested in academic research and further development of its work [54]. The NAG Library is widely-used both in academia and industry [48]. The salient points here are that any code contained in the NAG library must be both extremely *robust*, able to handle unexpected input or errors without breaking down, and *portable*, capable of being run on a wide variety of machines with correspondingly large differences in architectures.

These aims are reflected in the NAG Library itself. The NAG kernel is written in

[1] Originally, this stood for "*Nottingham* Algorithms Group", as the project was initially based at the University of Nottingham.

either Fortran or C, but several *wrappers* are built on top of it that allow interoperability with many other languages, such as Python or Java. Our implementation of the Cheng-Higham modified Cholesky algorithm is written in Fortran 95.

Of course, we also wish for our implementation to be as stable and accurate as possible, as with any numerical algorithm. Efficiency is also highly desirable. NAG prides itself on the efficiency of its software and this would obviously also be of great importance to users of the library. It is important however to ensure that this is not achieved at the expense of either accuracy or stability.

5.2 The `F01MDF` routine

It is intended that our new modified Cholesky factorization routine will be included in the Mark 27 release of the library, which should be available commercially from 2018. The name chosen for the routine was `F01MDF`. The first three characters identify the section of the library in which it is held: chapter `F` contains linear algebra routines and section `F01` those which deal with "matrix operations," a broad category that includes routines for matrix inversion, matrix factorizations and matrix functions, as well as more fundamental matrix operations like transposition.

5.2.1 Documentation

As with any software library, it is vital that any routine included in the NAG Library be as well-documented as possible in order for users to understand how to effectively use it. In fact, writing the accompanying documentation is often the first step of the development cycle for any new routine in the NAG Library and this was also true for `F01MDF`. Working this way is useful because it forces the developer to consider before beginning exactly what users want from the routine and how it will be delivered. The documentation for the `F01MDF` routine will be included in the user manual for the Mark 27 release of the NAG library; the Mark 26 manual can be viewed here [50].

5.2.2 Arguments

The arguments of the the Cheng-Higham modified Cholesky algorithm as described in section 3.1.2 are simple: it takes a symmetric matrix A and a tolerance δ as input and

calculates L and D. Two of the desired inputs (A and δ) and outputs (L and D) of our routine are therefore clear. We do not necessarily need to compute the permutation matrix P explicitly but since it is required to fully describe the factorization, it is an obvious third output. The practical question is, how do we accept (or return) these inputs (or outputs)?

The first thing to consider is that the matrix being factorized is symmetric. So in terms of storage, it is cheaper to simply work with either the upper or lower triangular part of A; for this reason, it is also entirely possible that the user has only actually stored one of these. A standard input argument therefore in many NAG routines that handle symmetric matrices is `uplo`, a character type defining which part of the matrix we store and reference: `L` for lower, or `U` for upper, corresponding to the lower or upper triangular part of the matrix. The other half of the matrix is neither referenced nor stored. We therefore adopted `uplo` as an input for `F01MDF`. The matrix A itself is stored in the two-dimensional array `a`. Note that we do not verify that `a` is actually symmetric: this is assumed.

The tolerance δ is more straightforward. The corresponding input variable `delta` is simply a real data type. It was considered whether to allow the possibility of a user not entering a value for `delta` and using the value $\delta = \sqrt{u}\,\|A\|_\infty$ suggested by Cheng and Higham as a default in that case, but this was ultimately rejected as unnecessary.

Although it may be possible to calculate the dimension n of the matrix A directly from the input array `a`, in practice it is simpler to also make this an input. The routine therefore takes an integer input `n` which stores the value of n. Whenever we use arrays to store the entries of matrices, the *leading dimension* of the array is important. Roughly speaking, this is an (integer) increment used to find the starting point of the next column (or row) of the matrix, depending on whether the array is stored in memory in column-major (or row-major) form. It is a standard input argument used in many NAG linear algebra routines and we therefore include it here as the variable `lda`. Note that Fortran stores arrays in column-major order and so this is the NAG standard for all of its Fortran library routines. A fuller explanation of the leading dimension and its importance in matrix computations can be found for example here [45] and a considerably more detailed one in the LAPACK user guide [2].

In order to compute the indefinite LDL^T factorization, we use the LAPACK routine `DSYTRF_RK`, which we shall discuss further in section 5.2.3. Using this mandates some of the arguments of our routine. For the sake of economy, rather than using new arrays to store L and D explicitly, `DSYTRF_RK` overwrites the entries of `a` in such a way that the user can construct them if they desire. The main diagonal of `a` is overwritten by the main diagonal of the matrix D. Since D is block diagonal, the off-diagonal elements also need to be stored—but, as it is symmetric, we actually only need to store either the subdiagonal or superdiagonal elements. Since there is only ever a nonzero off-diagonal element when we have a 2×2 block, we only strictly need an array of length $n/2$ to store them. However, `DSYTRF_RK` outputs the entire sub/superdiagonal, including the zero elements, as the one-dimensional, length `n` array `offdiag`[2]; the whole sub/superdiagonal is stored because doing otherwise makes the code unnecessarily convoluted.

If `uplo == 'L'`, then L is effectively stored in the strictly lower triangular part of `a`; we say "effectively" because since it has unit diagonal, it is not actually necessary to store this. Similarly, if `uplo == 'U'`, then the triangular matrix is effectively stored in the strictly upper triangular part of `a`. Note however that this factor is not L or L^T but a distinct upper triangular matrix; this is discussed in detail in section 5.2.3.

It is unnecessary to calculate the permutation matrix P explicitly. Instead we store the permutation information in the integer array `ipiv`. This is another output of the `DSYTRF_RK` routine used to compute the LDL^T factorization. Details of how the array actually stores the permutation information can be found in the documentation for `DSYTRF_RK` [61] and will be included in the documentation for the `F01MDF` routine when it is released. A final argument is also required by the routine, `ifail`. This is used as part of the standard error handling procedure in many NAG Library routines. Details can be found in [49].

A full description of all the arguments of the `F01MDF` routine will be available with the documentation in the user manual for the Mark 27 release of the NAG Library, although we include a brief summary here in the form of Table 5.1. The documentation

[2]Of course, the sub/superdiagonal of an order n matrix is of length $n - 1$. However, `offdiag` requires the extra element to accommodate another consequence of the choice of `uplo`: when `uplo == 'U'`, the sub/superdiagonal is stored in `offdiag(2)` to `offdiag(n)`; if `uplo == 'L'` it is stored in `offdiag(1)` to `offdiag(n - 1)`.

Table 5.1: Arguments of the F01MDF NAG Library routine.

Argument	Description
uplo	INPUT: Specifies which half of the matrix we store/reference
n	INPUT: The dimension of A
a	INPUT: The matrix A; OUTPUT: L and D
lda	INPUT: Leading dimension of a
offdiag	OUTPUT: Sub/superdiagonal of D
ipiv	INPUT/OUTPUT: Permutation information
delta	INPUT: Tolerance level δ
ifail	OUTPUT: Used in error handling

will also include code for an example program showing how to explicitly construct the matrices L, D and P, and use them to find the perturbation matrix E.

5.2.3 Computing the indefinite factorization

NAG has worked extensively on the LAPACK project for many years and sections F07 and F08 of the NAG Library contain routines implemented from LAPACK codes. As mentioned in section 4.2, one of these is an implementation of DSYTRF_RK for LDL^T factorization with rook pivoting [61], allowing us to make use of it. The DSYTRF_RK routine depends on two other auxiliary LAPACK routines, DSTYTRF2_RK [59] and DLASYF_RK [57]. Both of these actually perform LDL^T factorization with rook pivoting, however the latter is a blocked version and the former unblocked. After determining the block size b, DSYTRF_RK uses the blocked routine to factorize k blocks of b columns (called a *panel*) simultaneously, where $k = \text{floor}(n/b)$. The remaining $n - kb$ columns that have not been factorized are then handled sequentially using the unblocked routine. This is a standard approach utilised in many LAPACK routines. The optimal block size b is determined by calling the LAPACK function ILAENV [62]. This is a sophisticated routine used for this purpose that generally performs well, although for optimal performance it is recommended to adapt it to the particular machine on which it is being run [56].

As ever in software development when existing code is being adapted for use in a new routine, constraints and possibilities that had not previously been considered arose from our use of DSYTRF_RK to compute the indefinite factorization in F01MDF.

Most of these were very minor (for example, the argument `ipiv` had to be an input and an output rather than just an output to ensure compatibility with the LAPACK code). However, the most notable possibility that arose concerned another, related factorization of the matrix.

If `uplo == U`, then `DSYTRF_RK` computes not $PAP^T = LDL^T$ but $PAP^T = U\tilde{D}U^T$, where U is unit upper triangular. This is also true for the `DSYTRF` routine (which uses *partial*, instead of rook, pivoting). As previously noted, this U is not just the transpose of L but it is closely related: if we have the modified Cholesky factorization $PAP^T = LDL^T$ and Π is the permutation matrix that reverses rows by pre-multiplication, then we also have

$$\begin{aligned}\Pi^T PAP^T \Pi &= \Pi^T LDL^T \Pi \\ &= (\Pi^T L \Pi)(\Pi^T D \Pi)(\Pi^T L^T \Pi) \\ &= U(\Pi^T D \Pi)U^T \\ &= U\tilde{D}U^T,\end{aligned}$$

where U is upper triangular and $\tilde{D}$ is block diagonal.

We have not considered this factorization thus far for the simple reason that it would not be called a Cholesky factorization. However, the vast majority of the analysis from section 3.1.2 still applies, but with U instead of L (and $\tilde{D}$ instead of D). It may not technically be a modified Cholesky factorization, but can be used for almost all the same purposes. Precisely why this option was included in the LAPACK code is unclear, although we believe it may be to avoid awkward permutations. Since we wish to make use of the LAPACK routine and it is very much possible that the user has stored the upper triangular part of the symmetric matrix anyway, rather than the lower, it was reasonable for us to include this option for `F01MDF` as well.

5.2.4 Making the perturbations

Once we compute the indefinite factorization, perturbing the factors to form the modified Cholesky factorization is very simple, so will not be described in any depth here. We essentially just follow the description given in section 3.1.2 and iterate over the blocks of D, modifying them when necessary. There are some practical complications because the elements of D are stored in two separate locations rather than a single

matrix but the code is still both short and simple. When we encounter a 2×2 block, we explicitly construct it as an 2×2 array and then use the LAPACK routine DSYEVD [58] to determine its eigenvalues and eigenvectors. These are then used to compute the entries of the perturbed block and the elements of D are overwritten accordingly.

It is important to note that the F01MDF routine does not currently use any parallel code at this stage, although the method is very well-suited to parallel computing, as the modifications to the blocks of D are all independent of one another. However, it is intended that the code will be parallelized before its release with the Mark 27 version of the NAG Library.

Chapter 6

Testing the F01MDF Routine

To test our new F01MDF routine, we ran a series of numerical experiments. Note that this experimentation is distinct from the more extensive suite of testing that all routines must undergo before inclusion in the NAG Library, which should take place later this year, after the submission of this dissertation. The focus of that testing may also be slightly different than ours, placing more of an emphasis on software engineering concerns like erroneous user input, whereas we will focus on more mathematical aspects.

Fundamentally, we want to make sure that our routine matches the theoretical expectations of the Cheng-Higham algorithm as detailed in section 3.1.2. We are also of course in a sense looking for things that we do *not* expect: this is the best way to find any unforeseen problems or difficulties that may occur. Note that this dissertation will be referenced in the documentation for the F01MDF routine when it is eventually released with the NAG Library so testing also serves the further purpose of generating a body of data that may inform decisions made by future users of the routine. Our numerical experiments may for example suggest that the F01MDF routine generally obeys tighter bounds in practice than the theoretical ones established in Chapter 3, which would be eminently useful for users.

Our testing in this chapter focuses on those aspects of the routine that will be of most interest to users. First and foremost, they will want to know that the routine is accurate; this is the focus of section 6.2. They will want to know when it will perform well and when it will not; to this end, we consider how the size of the perturbation matrix E and the conditioning of $A + E$ differ for a very wide range of matrices in

sections 6.3 and 6.4 respectively. Information regarding the efficiency of the routine will surely also be useful and therefore in section 6.5, we investigate whether matrices that require $O(n^3)$ comparisons for the rook pivoting are as rare as the analysis suggests and also compare the efficiency of `F01MDF` to that of an existing NAG Library routine.

All testing in this chapter was done in the computing environment described in section 1.4, using a single computational core. We expect the results in sections 6.2–6.4 to be very similar once the routine is parallelized, as that should have no effect on the properties being considered. However results in section 6.5 concerning function timings are likely to differ when the computations are performed in parallel; this will be emphasised again before any such results are presented. The `F01MDF` routine was always run with `uplo == 'L'` and `delta == sqrt(eps) * norm(A, 'fro')`. Note that the MATLAB parameter `eps` is actually twice the unit roundoff u and the norm of A that we used was the Frobenius norm so this value of `delta` differs from that suggested for δ by Cheng and Higham in [12]. However, we found that this was generally a good value for δ in practice.

6.1 Test matrices

Ultimately, we want to perform our numerical experiments with matrices similar to those that the `F01MDF` routine may actually be used for in applications. To this end, it is important that we consider as wide a range of matrices as possible, with different eigenvalue distributions and dimensions: it could be that certain kinds of matrices are more likely to occur in a particular application than others and the user may therefore wish to know how the routine performs for those only. In section 6.1.1 we describe the method that was used for generating random matrices used in our experiments throughout this chapter and in section 6.1.2 we describe one set of matrices from real-world applications that were also considered.

6.1.1 Generating random matrices

In order to create random matrices for our numerical experiments, we follow the lead of Cheng and Higham in [12] and use matrices of the form $A = Q\Lambda Q^T$, where Q is a

random orthogonal matrix generated using the `qmult` function from the Matrix Computation Toolbox [32] and $\Lambda = \text{diag}(\lambda_i)$, with the λ_i from a chosen random uniform distribution. This method of construction is preferred because we can easily control the eigenvalues of A by specifying the range of the random uniform distribution. Throughout this chapter, when we refer to "random" matrices we will mean matrices constructed in the preceding manner.

6.1.2 Matrices from applications

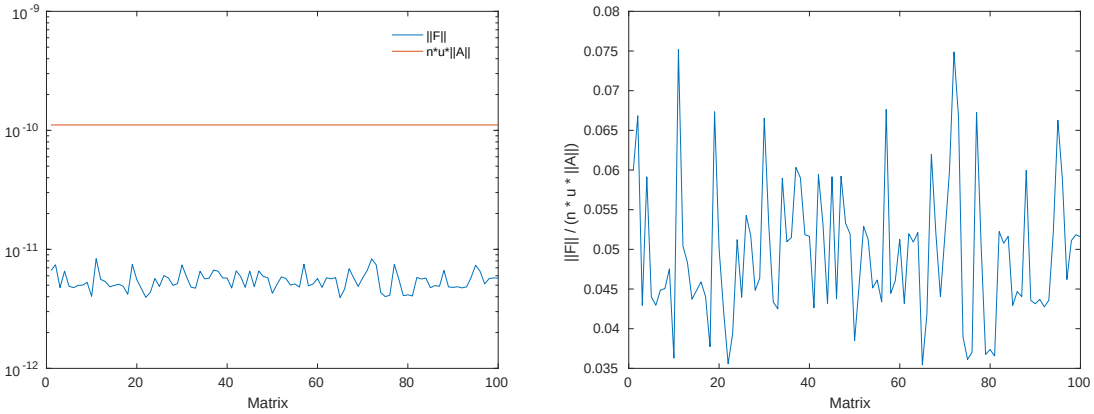
With the application of finding the distance to the nearest correlation matrix described in Chapter 7 in mind, we consider the set of 13 invalid correlation matrices from real-world applications provided at this GitHub repository by Higham and Strabić [41]. The definition of a correlation matrix—and how an invalid one may arise—will be explained in Chapter 7 but here it suffices to say that these matrices are all symmetric with unit diagonal and also indefinite. They differ quite widely in order, from 3 to 3250. A full description of each matrix can be found in the `readme` file at the repository.

Note that these matrices may not be typical of those that occur in other applications. In particular, they generally have relatively few negative eigenvalues and these tend to be much smaller in magnitude than the largest positive eigenvalues. Most (although not all) also have all of their elements bounded above by one.

6.2 Accuracy

The important theoretical result with regards to the accuracy of the Cheng-Higham modified Cholesky algorithm is the bound (3.1), so that will inform the discussion in this section. We can calculate $\|A + E\|_2$ so the only unknown on the right-hand side of (3.1) is the small function of n, c_n. In this section, we investigate if we can make some rough estimate of its size, or at least find an approximate upper bound for it. This will allow us to gauge how accurate the routine generally is in practice.

The difficulty is that in general we cannot precisely predict the size of E a priori: we have the upper bound (3.3) but that is all. So for an indefinite matrix we can't effectively distinguish between the perturbation matrix E and the error matrix F. What we can do however is test known positive definite matrices. Mathematically,



(a) Norm of F

(b) Ratio $\|F\|_2 / nu\,\|A\|_2$

Figure 6.1: Measures of the error matrix F for 100 random matrices of order $n = 100$, with eigenvalues in $[1, 10^4]$ and maximum eigenvalue fixed as 10^4.

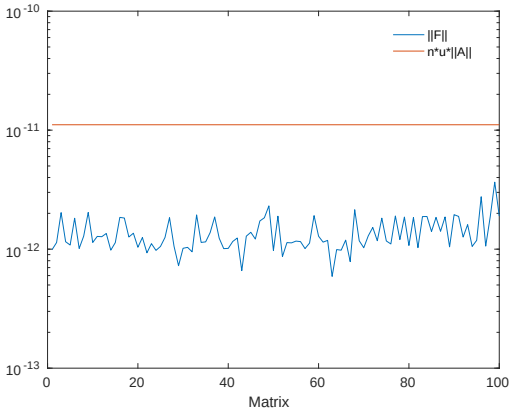
we know that in this case the algorithm returns $E = 0$, so the bound on F becomes $\|F\|_2 \leq c_n u\,\|A\|_2 = c_n u \lambda_{\max}(A)$.

Figure 6.1a plots the 2-norm of F for each of 100 random positive definite matrices of order $n = 100$ with eigenvalues in the range $[1, 10^4]$ and largest eigenvalue fixed as 10^4. We also plot the line $y = nu\,\|A\|_2$ for comparison. Figure 6.1b explicitly shows the ratio $\|F\|_2 / nu\,\|A\|_2$ for these matrices. We see that $\|F\|_2$ is generally only a small fraction of $nu\,\|A\|_2$.

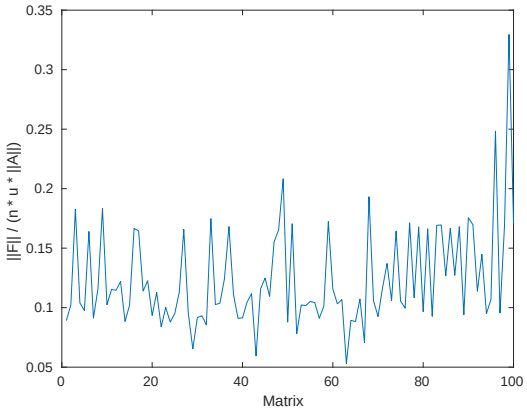
Figures 6.2a and 6.2b show the results when we repeat the previous experiments but with matrices of order $n = 10$. In this case we can see that the ratio $\|F\|_2 / nu\,\|A\|_2$ is larger than before but still appears to be bounded above by a small fraction (approximately 0.4) of n. The relative bounds observed for the two different values of n also suggests that c_n is not just a simple linear function of n.

Figures 6.3a and 6.3b plot the ratio for 100 random matrices of order 10 when we increase the maximum eigenvalue by two and eight orders of magnitude, respectively; this loosens the bound on $\|F\|_2$ correspondingly. Note that although the absolute size of the errors is larger than we have previously seen, the ratio $\|F\|_2 / nu\,\|A\|_2$ still appears to be bounded above by about 0.4.

Alternatively, Figures 6.4a and 6.4b are the result of keeping the maximum eigenvalue fixed as 10^4 but increasing the dimension of the matrix to $n = 500$ and $n = 1000$ respectively. We observe the same behaviour as in previous experiments with regards to the ratio $\|F\|_2 / nu\,\|A\|_2$.

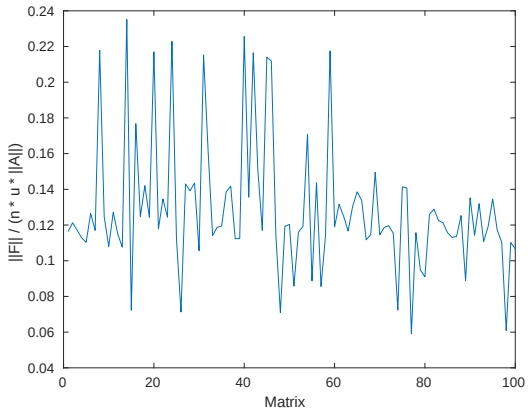


(a) Norm of F

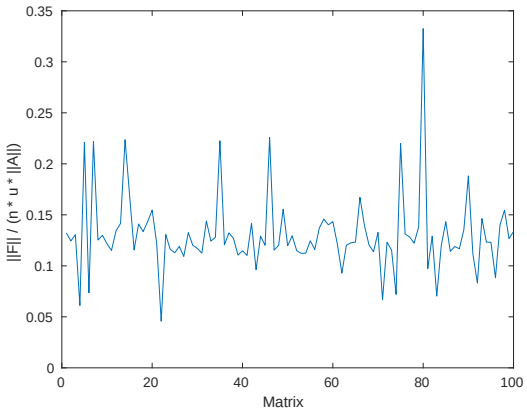


(b) Ratio $\|F\|_2 / nu \|A\|_2$

Figure 6.2: Measures of the error matrix F for 100 random matrices of order $n = 10$, with eigenvalues in $[1, 10^4]$, and maximum eigenvalue fixed as 10^4.

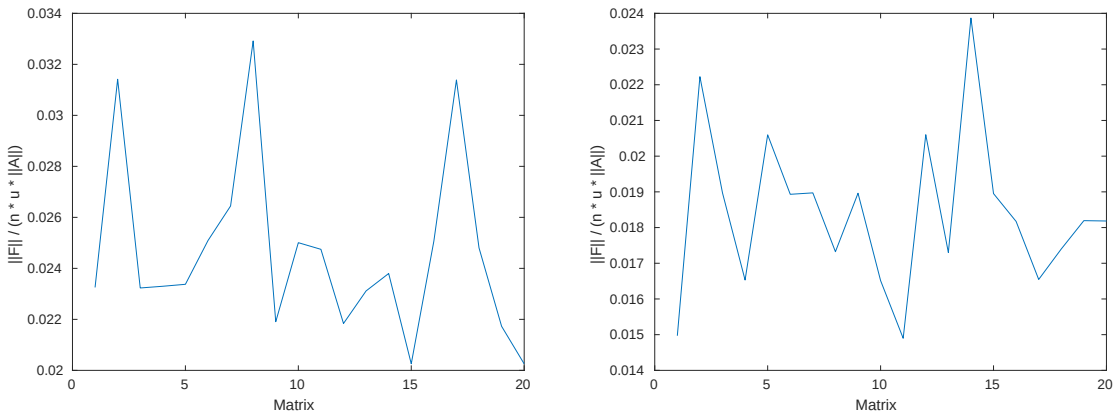


(a) $k = 6$



(b) $k = 12$

Figure 6.3: The ratio $\|F\|_2 / nu \|A\|_2$ for 100 random matrices of order $n = 10$ with eigenvalues in $[1, 10^k]$ and maximum eigenvalue fixed as 10^k.



(a) $n = 500$ (b) $n = 1000$

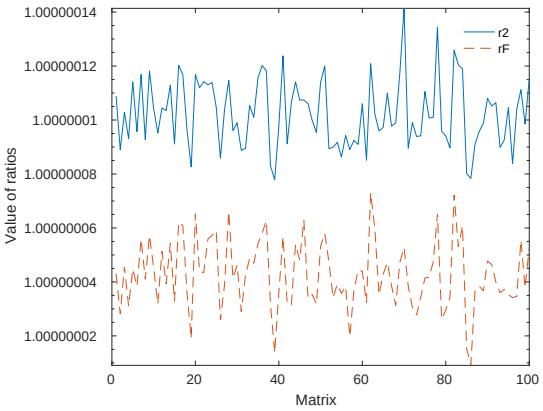
Figure 6.4: The ratio $\|F\|_2 / nu \|A\|_2$ for 20 random matrices of order n with eigenvalues in $[1, 10^4]$ and maximum eigenvalue fixed as 10^4.

To sum up, our experiments suggest that the error in using the `F01MDF` routine is perhaps bounded above by about $0.4nu \|A\|_2$ in practice, although much more experiments would be required before concluding this is generally the case. Certainly, throughout all of our testing, we never observed a matrix for which the error exceeded this bound. Further evidence in support of this (or any other) de facto bound would allow us to speak with greater confidence about the size of E when the matrix is actually perturbed by the routine (i.e., A is indefinite), particularly if we also have any information about the maximum eigenvalue of A.

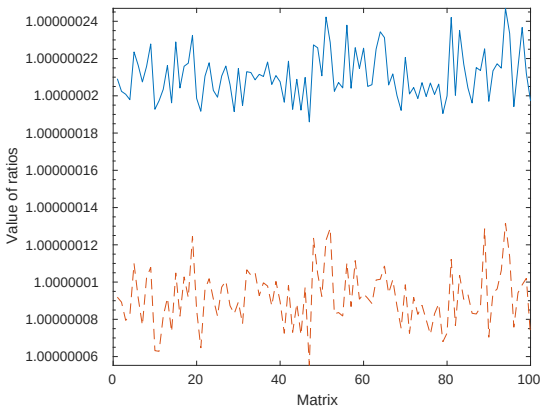
6.3 Measuring the perturbation

In this section, we establish some results for the size of the perturbation matrix E produced by the `F01MDF` routine. For random marices, we emulate the approach of Schnabel and Eskow and Cheng and Higham in considering matrices with three different eigensystems: negative definite, indefinite and "slightly" indefinite (i.e., with only a few, relatively small, negative eigenvalues). We distinguish the last grouping from other indefinite matrices because they are particularly common in certain applications (for example, that detailed in Chapter 7). We also consider the set of 13 invalid correlation matrices from Higham and Strabić described in section 6.1.2.

To evaluate the performance of the `F01MDF` routine, we make use of the ratios r_2 and r_F defined in (4.2), as these are the natural way to measure success here. The



(a) $n = 25$



(b) $n = 100$

Figure 6.5: The ratios r_2 and r_F plotted for 100 random matrices of order n with eigenvalues in $[-10^4, -1]$ and minimum eigenvalue fixed as -10^4.

practical value of the bound (3.3) will also be considered for the invalid correlation matrices in particular. We perform little comparative testing in this section as we are mostly interested in establishing base results for future users of the routine, but we will comment on the results obtained.

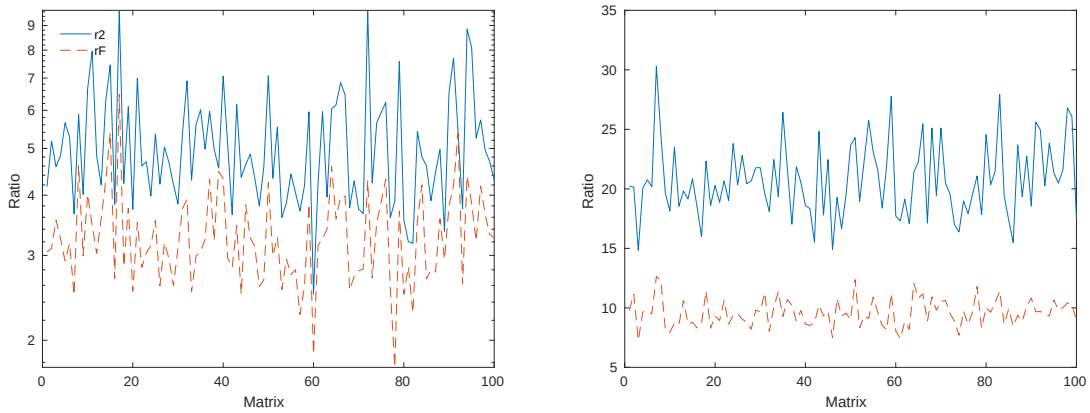
6.3.1 Negative definite random matrices

Figures 6.5a and 6.5b show the ratios r_F and r_2 for 100 random matrices with eigenvalues in the range $[-10^4, -1]$ (i.e., negative definite), of order $n = 25$ and $n = 100$ respectively. In both cases, we fixed the minimum eigenvalue of all the matrices at -10^4 in order to ensure that each had the same upper bound on the error (see section 6.2).

We see that our results support the analysis from section 4.1.2 and the previous results obtained by Cheng and Higham and Fang and O'Leary: the perturbation matrix E is usually very nearly optimal when A is negative definite. Hence we expect the routine to perform particularly well in this regard for applications in which negative definite matrices may occur.

6.3.2 Indefinite random matrices

In this section we consider only those matrices with an fairly even mix of positive and negative eigenvalues, of similar magnitudes; "slightly" indefinite matrices with only a



(a) $n = 25$ (b) $n = 100$

Figure 6.6: The ratios r_2 and r_F plotted for 100 random matrices of order n with eigenvalues in $[-1, 1]$.

few small negative eigenvalues are considered in section 6.3.3.

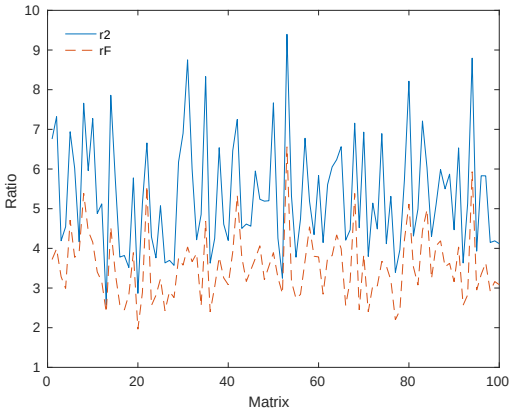
We first considered random matrices with eigenvalues in the range $[-1, 1]$. Note that we did not fix the largest eigenvalue as in the previous section. Figures 6.6a and 6.6b show the size of the perturbation matrix E produced for 100 random matrices of order $n = 25$ and $n = 100$, respectively. Figures 6.7a and 6.7b display the results when we repeat the experiment for random matrices with eigenvalues in the range $[-10^4, 10^4]$. We see that altering the eigenvalue range has no appreciable effect on the size of the perturbation matrix produced.

The ratios r_2 and r_F appear to generally be bounded above by about $n/2$, where n is the order of the matrix, and r_F in particular is usually considerably smaller. However, further testing would be required to establish whether this is in fact typical.

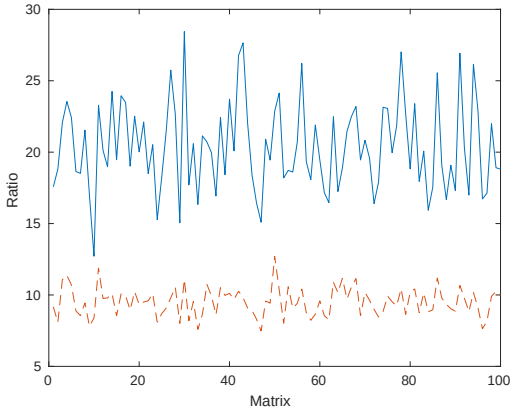
6.3.3 Slightly indefinite random matrices

Matrices with a relatively small number of negative eigenvalues are likely to occur in many applications. Indeed, there are situations in which the user may not even be aware that the matrix they have is not positive definite (an application in which this can often occur is discussed in Chapter 7). Hence we devote special attention to this type of indefinite matrix in this section.

In Figures 6.8a and 6.8b we show the ratios r_F and r_2 for 100 random matrices of order $n = 25$ and $n = 100$, respectively, with eigenvalues in the range $[-1, 10^4]$.

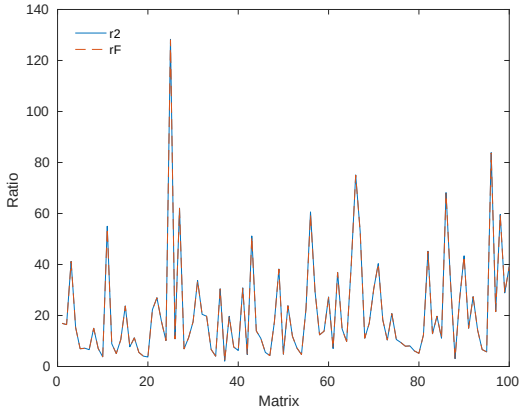


(a) $n = 25$

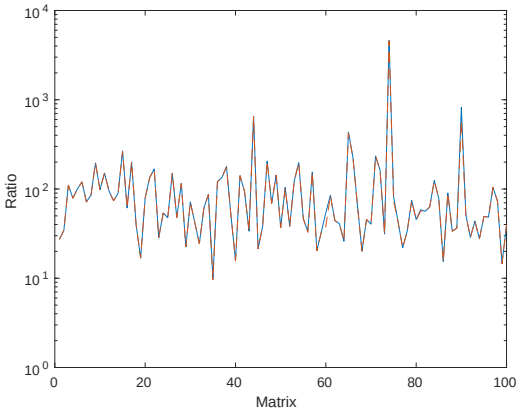


(b) $n = 100$

Figure 6.7: The ratios r_2 and r_F plotted for 100 random matrices of order n with eigenvalues in $[-10^4, 10^4]$.



(a) $n = 25$

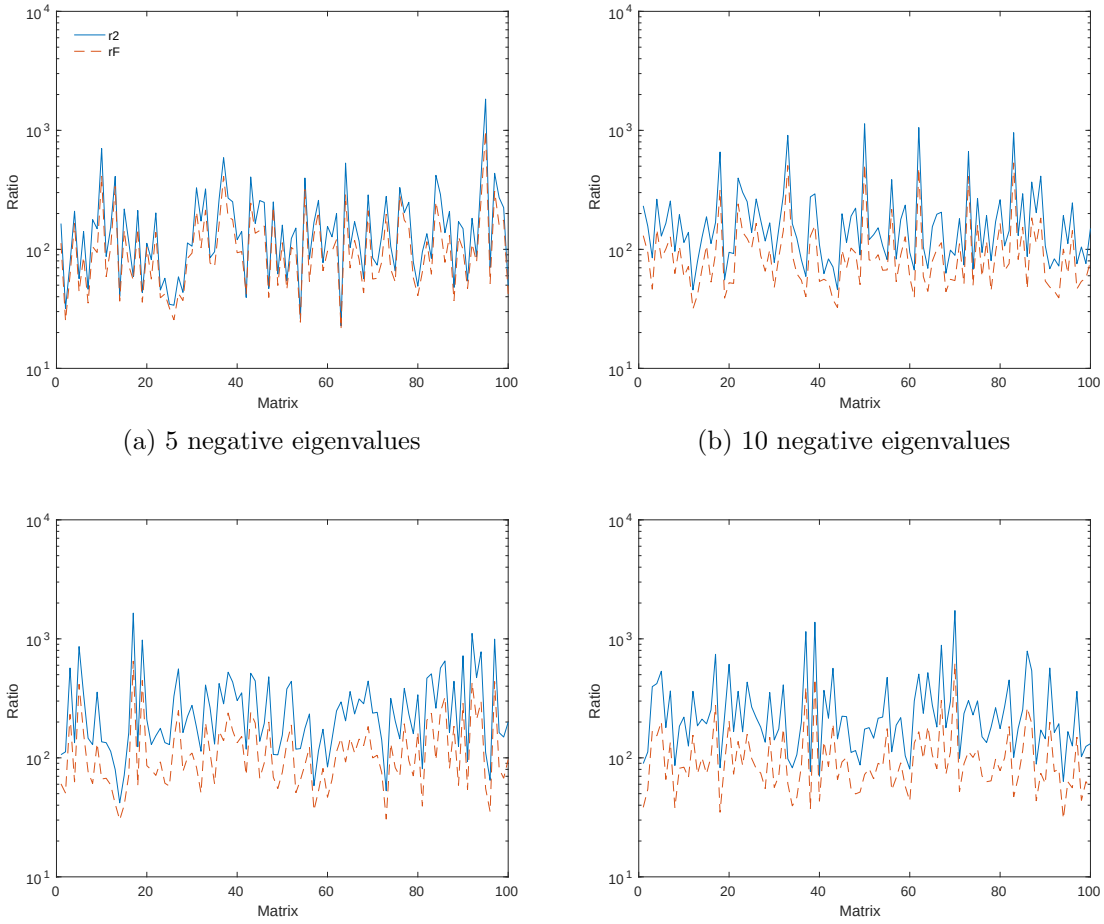


(b) $n = 100$

Figure 6.8: The ratios r_2 and r_F plotted for 100 random matrices of order n with eigenvalues in $[-1, 10^4]$ and at least one negative eigenvalue.

Note that we ensured that at least one eigenvalue was negative and therefore the matrix was truly indefinite. We observe that both ratios are considerably higher than the corresponding results achieved in section 6.3.2, by as much as three orders of magnitude in the worst case. We also see that, unlike in the previous section, the two ratios are effectively indistinguishable.

We next considered the effect of varying the number of negative eigenvalues of the input matrix. Figures 6.9a–6.9d show the effect that increasing the number of negative eigenvalues has on the ratios r_F and r_2, for 100 random matrices of order $n = 100$ and eigenvalues in the range $[-1, 10^4]$. Other than the difference between



(a) 5 negative eigenvalues

(b) 10 negative eigenvalues

(c) 15 negative eigenvalues

(d) 20 negative eigenvalues

Figure 6.9: The ratios r_2 and r_F for 100 random matrices of order $n = 100$ and eigenvalues in $[-1, 10^4]$, for differing numbers of negative eigenvalues.

r_F and r_2 increasing with the number of eigenvalues, we observe few other differences between the plots. Mathematically, there is no reason to assume that there would be but we considered this to still be worth establishing, particularly since it was an investigation of this kind that led Schnabel and Eskow to discover the matrices that created difficulties for their original algorithm [70].

6.3.4 Correlation matrix data set

Table 6.1 records the ratios r_F and r_2 for the invalid correlation matrices from Higham and Strabić, as well as the norm of the perturbation matrix E itself and the bound (3.3). We observe roughly the same behaviour with regards to the ratios r_F and r_2 that we have already remarked upon in previous sections. We also see that although

Table 6.1: Measures of the perturbation matrix E computed for 13 invalid correlation matrices, with the upper bound (3.3) included for comparison.

Matrix	Order	r_2	r_F	$\|E\|_2$	Bound (3.3)
bccd16	3250	5.68e1	5.07e1	1.56e3	2.72e7
beyu11	12	5.09	5.09	4.43e-2	1.09
bhwi01	5	4.40	4.40	5.61e-1	4.94
cor1399	1399	4.70	3.65	3.97e1	5.54e7
cor3120	3120	1.24e3	6.41e2	8.89e2	1.95e8
fing97	7	2.08	2.08	7.94e-2	8.69e-1
high02	3	2.41	2.41	1	5.77
mmb13	6	1.05	1.05	2.26e1	4.88e3
tec03	4	4.17	4.17	1.15e-1	7.57e-1
tyda99r1	8	4.28	3.83	4.33	6.82e1
tyda99r2	8	3.55	3.51	2.02	3.98e1
tyda99r3	8	4.49	4.13	2.25	3.20e1
usgs13	94	5.49e1	5.11e1	2.55	3.71e2

the norm of the perturbation matrix is always within the bound (3.3), the bound itself may not be accurate enough to be practical, often being fairly tight but also being six orders of magnitude too large in the worst case examples of `cor1399` and `cor3120`.

6.4 Conditioning of the perturbed matrix

As in the previous section, our primary objective here is simply to obtain a body of data to act as a reference for future users of the `F01MDF` routine, although we will also remark on interesting or unexpected results. To that end, we investigated the conditioning of the perturbed matrix $A+E$ for the three different varieties of matrices considered in section 6.3 (i.e., negative definite, indefinite and "slightly" indefinite), as well as the set of invalid correlation matrices of Higham and Strabić.

6.4.1 Random matrices

Figures 6.10a–6.10f plot the condition number of the perturbed matrix $A+E$ against the condition number of the original matrix A for 100 random matrices with eigenvalues in each of the ranges $[-10^4, -1]$, $[-1, 1]$ and $[-1, 10^4]$, and of orders $n = 25$ or $n = 100$. We see that the two are very close in the negative definite case but otherwise the condition number of $A+E$ is considerably higher than the condition number of the

original matrix A. We further investigate the relationship between the two in Figures 6.11a–6.11d, with the results suggesting that there is little to no relationship between the conditioning of the indefinite matrix A and the perturbed matrix $A + E$, which is in fact entirely what we would expect given the nature of the bound (3.4).

Figures 6.12a and 6.12b show the effect of changing the variable `delta` (which represents the tolerance level δ) used in the `F01MDF` routine. We see that, unlike the conditioning of A, this does seem to have an effect on the conditioning of $A + E$, with the condition number generally decreasing by about an order of magnitude as `delta` increases correspondingly. Again, this is entirely to be expected. As described in section 3.1.2, the Cheng-Higham algorithm attempts to ensure that the minimum eigenvalue of the perturbed matrix is approximately δ and, as changing δ has no effect on the maximum eigenvalue, we would therefore expect the condition number of the perturbed matrix to decrease as δ increases, in an approximately linear fashion.

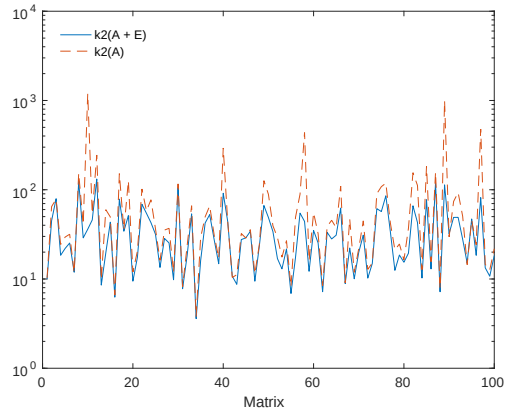
6.4.2 Correlation matrix data set

Table 6.2 records the condition numbers of the 13 invalid correlation matrices in the set provided by Higham and Strabić, as well as the condition number of the perturbed matrix $A + E$ produced by the `F01MDF` routine and the upper bound (3.4). We again observe that $\kappa_2(A)$ has little to no effect on $\kappa_2(A + E)$. The practicality of the upper bound is unclear: it is often within one or two orders of magnitude of the true condition number of $A + E$, but in the worst case example (`cor3120`) it is eight orders of magnitude too large.

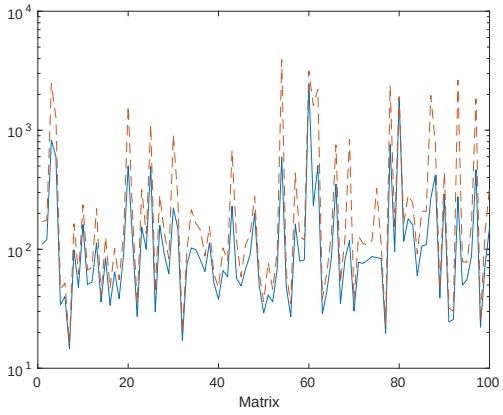
6.5 Efficiency

The major concern with the Cheng-Higham modified Cholesky algorithm in this regard is the possibility that $O(n^3)$ comparisons may be required for the rook pivoting strategy used to compute the indefinite factorization, so in section 6.5.1 we investigate how likely such matrices are to occur in practice. In section 6.5.2 we then compare the efficiency of the `F01MDF` routine with the existing NAG Library routine for standard Cholesky factorization of a positive definite matrix.

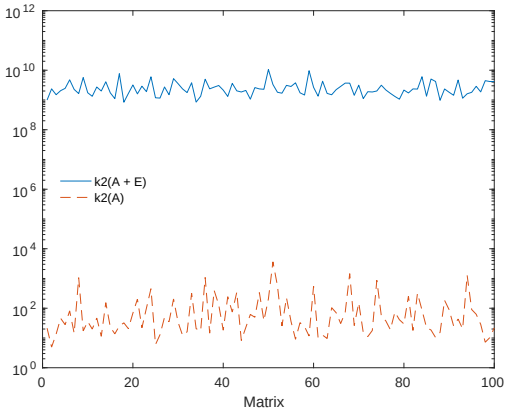
We reiterate here that as the routine is not yet parallelized, we restricted ourselves



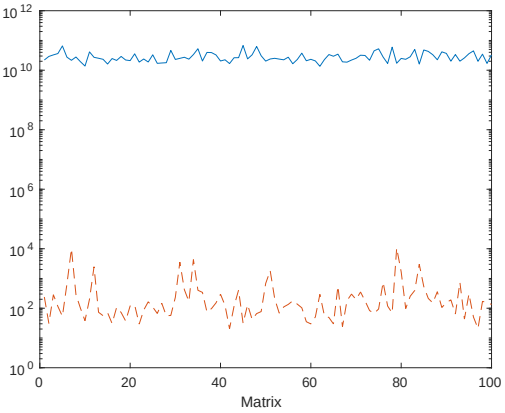
(a) $n = 25$, negative definite



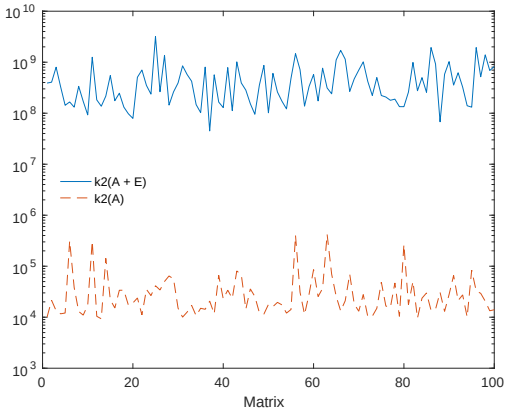
(b) $n = 100$, negative definite



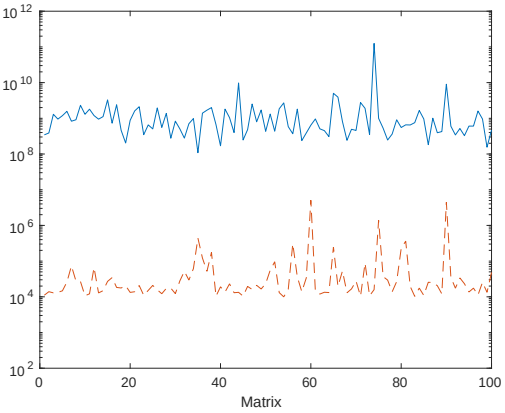
(c) $n = 25$, indefinite



(d) $n = 100$, indefinite

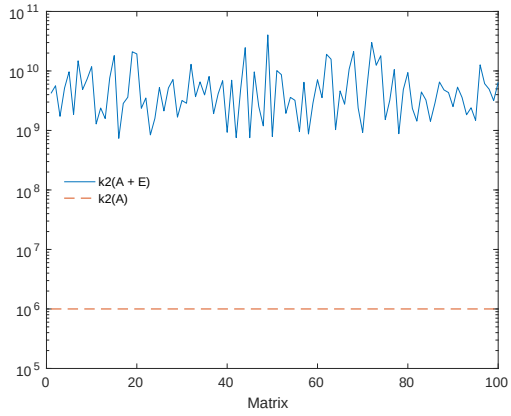


(e) $n = 25$, slightly indefinite

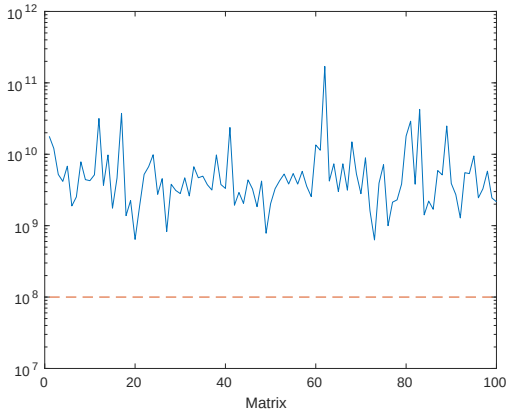


(f) $n = 100$, slightly indefinite

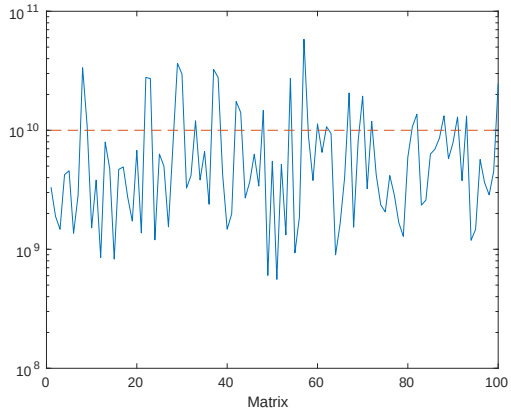
Figure 6.10: Condition numbers $\kappa_2(A)$ and $\kappa_2(A + E)$ for random matrices of different degrees of definiteness. Here, "negative definite" means eigenvalues in the range $[-10^4, -1]$, "indefinite" $[-1, 1]$ and "slightly indefinite" $[-1, 10^4]$.



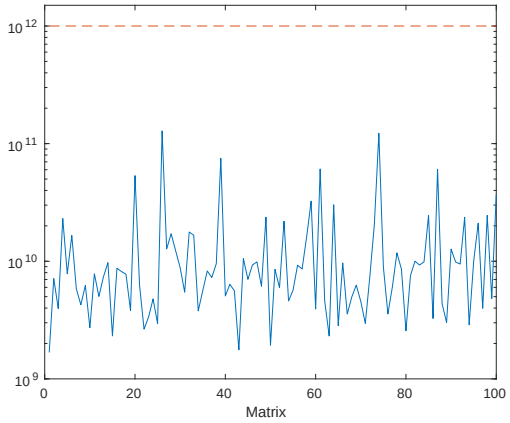
(a) $k = 6$



(b) $k = 8$

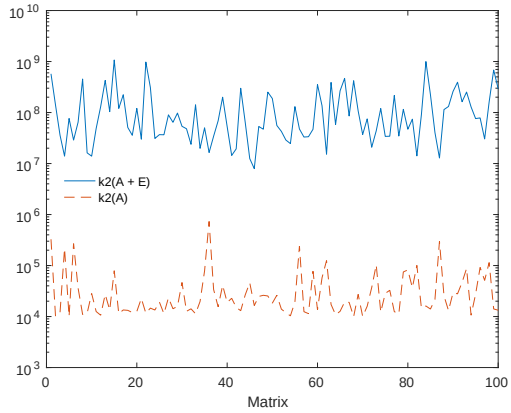


(c) $k = 10$

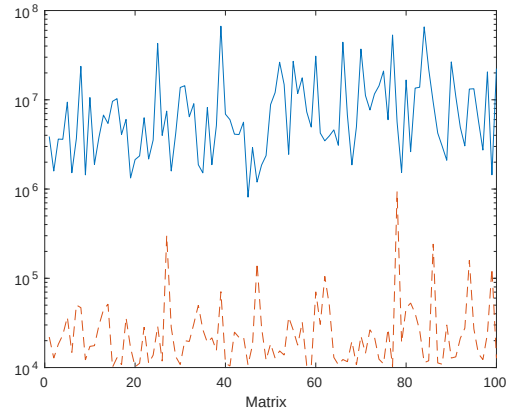


(d) $k = 12$

Figure 6.11: $\kappa_2(A)$ and $\kappa_2(A + E)$ for 100 random matrices of order $n = 100$ with eigenvalues in $[-1, 10^4]$ and $\kappa_2(A)$ fixed at 10^k, for different values of k.



(a) `delta == 0.01`



(b) `delta == 0.1`

Figure 6.12: $\kappa_2(A)$ and $\kappa_2(A + E)$ for 100 random matrices of order $n = 100$, with eigenvalues in the range $[-1, 10^4]$, for two different values of `delta`.

Table 6.2: Condition number for 13 invalid correlation matrices.

Matrix	Order	$\kappa_2(A)$	$\kappa_2(A+E)$	Bound (3.4)
bccd16	3250	4.10e3	1.48e10	1.80e16
beyu11	12	7.39e2	3.17e8	1.26e11
bhwi01	5	2.34e1	3.78e8	1.80e10
cor1399	1399	3.65e19	2.38e11	1.55e18
cor3120	3120	5.45e5	1.01e13	1.38e21
fing97	7	9.36e1	1.41e8	6.54e9
high02	3	5.83	2.28e8	3.18e9
mmb13	6	1.45e17	2.17e8	4.78e9
tec03	4	1.05e2	2.84e8	9.77e9
tyda99r1	8	1.26e1	3.98e8	2.32e10
tyda99r2	8	1.98e1	4.27e8	4.54e10
tyda99r3	8	1.51e1	4.08e8	2.99e10
usgs13	94	1.17e3	1.04e10	1.20e14

to a serial environment and performed all calculations on a single computational core. Once parallel code has been incorporated, this testing should ideally be repeated. As the routine is so well-suited to parallel computing, we do not expect the comparative results presented in section 6.5.2 to differ appreciably.

6.5.1 Rook pivoting

Making the modified Cholesky perturbations to the block diagonal matrix D is an $O(n)$ operation so the bulk of the cost of `F01MDF` comes from computing the indefinite factorization. As described in section 5.2.3, this is done with the LAPACK routine `DSYTRF_RK`. All of the pivoting is also done by that routine so in principle we could simply test the efficiency of that rather than `F01MDF`, but we believe it is wise to consider the routine as a whole in case there are any unforeseen problems elsewhere.

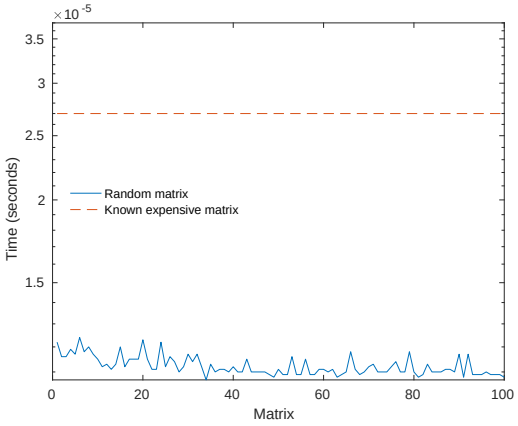
Higham in [36, p. 228] provides MATLAB code for generating matrices of an input order n that require $O(n^3)$ comparisons for the rook pivoting and we adapted this as the `expensive_matrix` function included in Appendix E. Figures 6.13a, 6.13b, 6.14a and 6.14b show the time taken for the `F01MDF` routine for random matrices of order $n = 10$ or $n = 100$ with eigenvalues in the ranges $[-1, 10^4]$ or $[-1, 1]$; negative definite matrices with eigenvalues in $[-10^4, -1]$ were very similar to the former. Also included in each figure is the average time taken for 10 tests of the `F01MDF` routine for a matrix

of the same order generated using the `expensive_matrix` function.

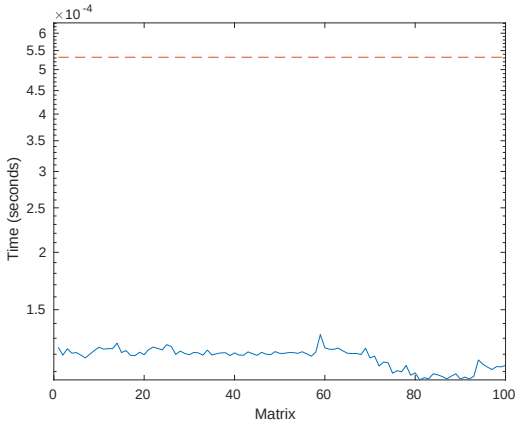
For the matrices with eigenvalues in $[-1, 10^4]$, the time taken for the known expensive matrix was always considerably higher than for random matrices of the same dimension and this was true for the random matrices with eigenvalues in $[-1, 1]$ of order $n = 100$ as well. For those matrices with eigenvalues in the latter range and of order $n = 10$, the timings were closer. The code used to generate the figures was run several times to ensure that the results were consistent and we include Figure 6.14a as that was a typical example for matrices of that type, but Figure 6.15 was the output of another iteration of the generating code.

We see from Figure 6.15 that for one of the random matrices, the time taken exceeds that of the known expensive matrix. This was the only matrix in our testing for which this was the case, although it can also be seen from Figure 6.15 that there was another matrix in that set that came fairly close. As the matrices used to generate the figures were not stored in this case, we were unable to confirm that the matrix which did exceed the time taken for the known expensive matrix also required $O(n^3)$ comparisons. However even if it did, the time taken should have only been approximately the same, when in fact it was almost double. This suggests that the problem lies elsewhere in the routine. However, since this was only observed for one matrix amongst approximately 1400 (of that order) tested overall and was not apparent in our later attempts to duplicate the result, we suspect the issue may be a glitch due to our computing environment. We do however believe it should be noted as it may be wise to pursue this as an avenue of investigation if any future users report similar issues. Note also that for larger matrices with eigenvalues in the same range, every iteration of the generating code produced a plot very similar to Figure 6.14b.

Table 6.3 records the results when we repeat the experiments for the invalid correlation matrices from Higham and Strabić. They largely concur with our findings for random matrices. Note that again we see that the disparity between the two timings is most pronounced for the largest matrices, with `F01MDF` being 270 times faster for `bccd16` than a matrix of the same order (3250) that requires $O(n^3)$ comparisons for the pivoting.

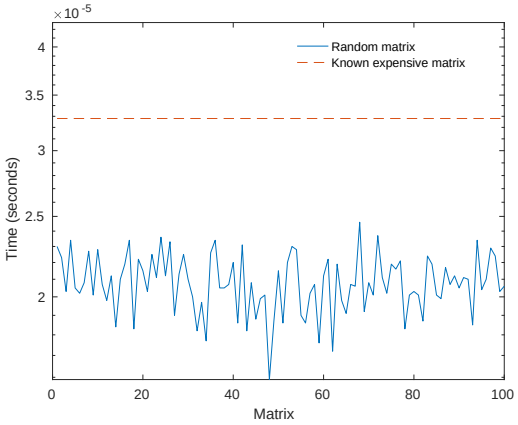


(a) $n = 10$

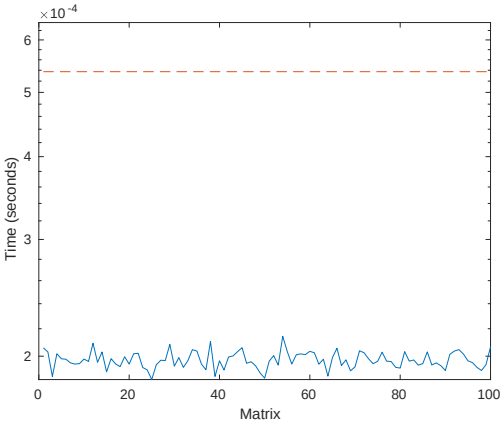


(b) $n = 100$

Figure 6.13: Timings for `F01MDF` for 100 random matrices of order n with eigenvalues in $[-1, 10^4]$. The time taken for a known expensive matrix of the same order is included for comparison.



(a) $n = 10$



(b) $n = 100$

Figure 6.14: Timings for `F01MDF` for 100 random matrices of order n with eigenvalues in $[-1, 1]$, with the time taken for a known expensive matrix of the same order included for comparison.

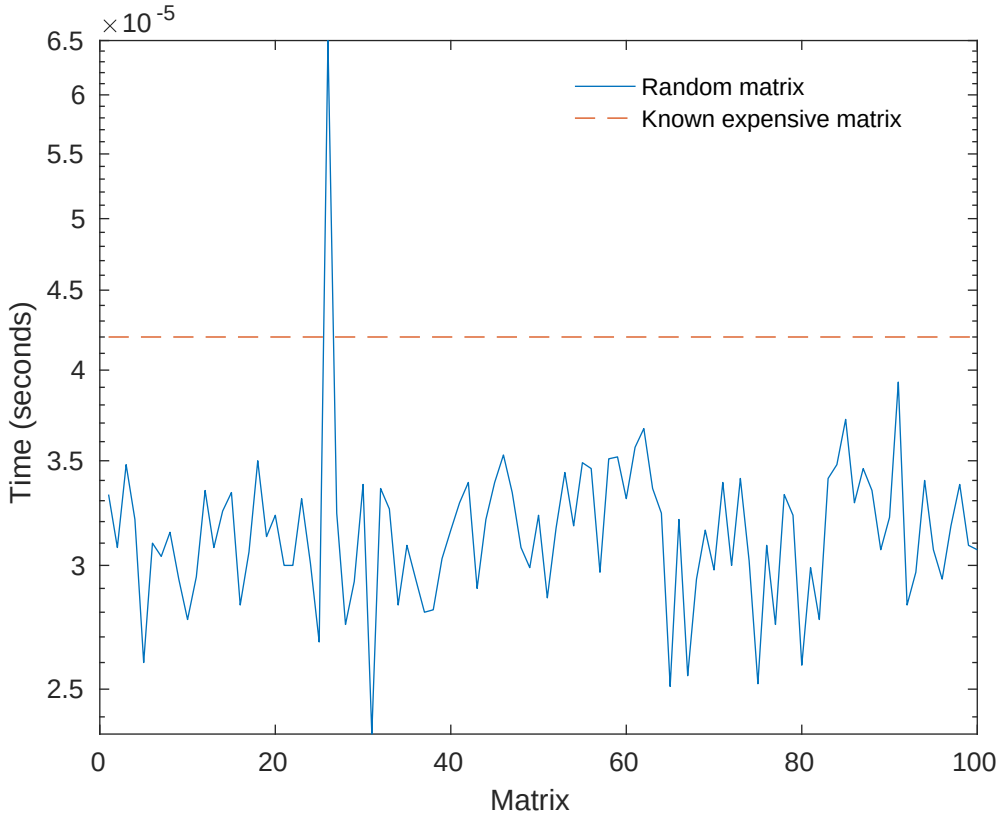


Figure 6.15: Timings for `F01MDF` for 100 random matrices of order $n = 10$ with eigenvalues in $[-1, 1]$. This plot is the output of the only one of 14 iterations of the generating code for which the time taken for a random matrix exceeded the known expensive matrix.

Table 6.3: Timings for `F01MDF` for 13 invalid correlation matrices (ICMs) and a known expensive matrix (KEM) of the same order, with the ratio between the two included for clarity.

Matrix	Order	Time (seconds)	KEM Time (seconds)	KEM/ICM
bccd16	3250	7.88e-1	2.13e2	270.3
beyu11	12	3.14e-5	6.56e-4	20.9
bhwi01	5	2.43e-5	4.80e-5	1.98
cor1399	1399	9.38e-2	1.49e1	158.8
cor3120	3120	8.09e-1	1.85e2	228.7
fing97	7	2.23e-5	9.10e-5	4.08
high02	3	1.19e-5	4.40e-5	3.70
mmb13	6	1.68e-5	3.66e-5	2.18
tec03	4	1.87e-5	3.31e-5	1.77
tyda99r1	8	1.93e-5	3.54e-5	1.83
tyda99r2	8	2.02e-5	3.88e-5	1.92
tyda99r3	8	2.09e-5	3.51e-5	1.68
usgs13	94	1.68e-4	3.04e-3	18.1

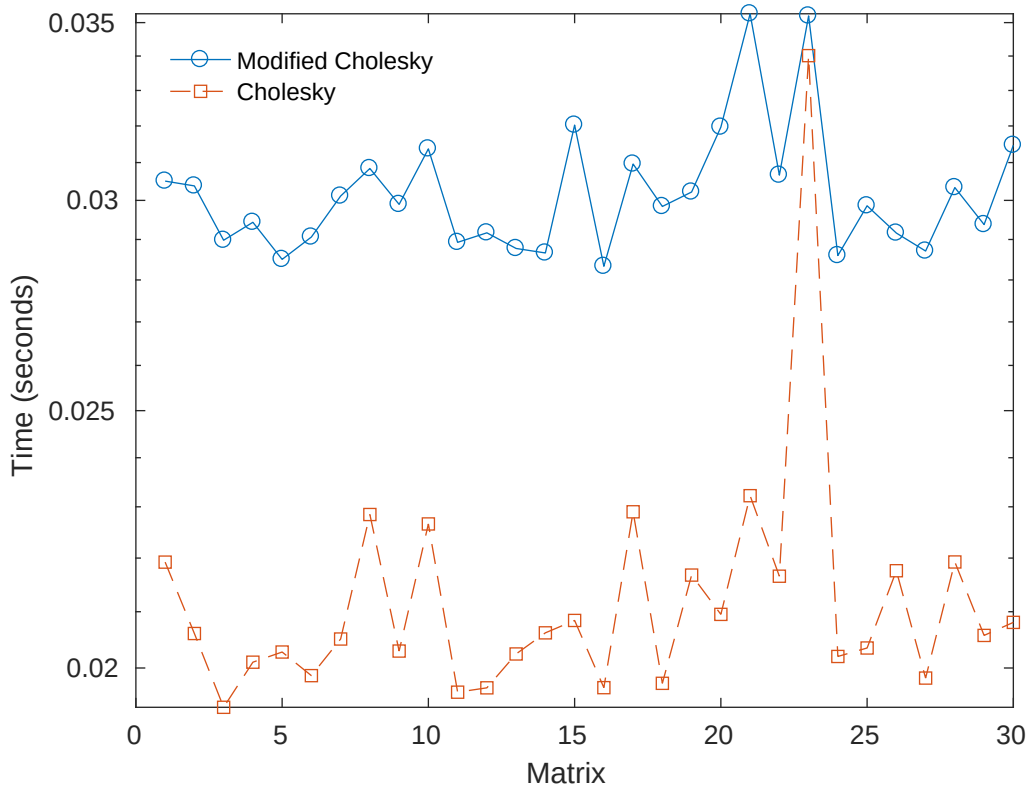


Figure 6.16: Timings for `F01MDF` for 30 random matrices (of order $n = 1000$ with eigenvalues in $[-1, 10^4]$) and `F07FD` for 30 random positive definite matrices of the same order.

6.5.2 Comparison with the standard Cholesky factorization

One of the primary objectives for any modified Cholesky algorithm is that it should not be significantly more expensive than the standard Cholesky factorization. To consider this for the `F01MDF` routine, we made use of the NAG Toolbox for MATLAB routine `F07FD` [52] which computes the standard Cholesky factorization of a positive definite matrix. Figure 6.16 shows the time taken for the `F01MDF` routine for 30 random indefinite matrices of order $n = 1000$ with eigenvalues in $[-1, 10^4]$; other indefinite eigenvalue ranges produced similar results. Also shown is the time that the `F07FD` routine takes to factorize a positive definite matrix of the same order. We see that on average `F01MDF` is about 30% more expensive than `F07FD`, which translates to roughly 0.06 seconds in runtime. Given that the modified Cholesky factorization has an additional cost beyond that of the standard Cholesky factorization of $O(n^2)$ flops and $n^2 = 10^6$ in this case, we consider this to be reasonable.

Chapter 7

Bounds on the Distance to the Nearest Correlation Matrix

7.1 The nearest correlation matrix problem

In statistical modelling, a *correlation matrix* is often constructed to express the correlation coefficients between a set of two or more random variables, where the (i, j) entry of the matrix is the correlation coefficient between the variables x_i and x_j. It is clear that such a matrix must be symmetric with unit diagonal; it is also positive semidefinite, although this is not as immediately obvious [72, p. 24–25]. The ubiquity of correlation matrices in statistical modelling is such that the name has been adopted in linear algebra to describe any real symmetric positive semidefinite matrix.

In many applications, a matrix intended to represent the correlation coefficients between a set of variables which should therefore be a correlation matrix in fact isn't, most often because it is not actually positive semidefinite. There are many possible ways this may happen but it is normally because of missing sample data being extrapolated or matrix entries being replaced, for reasons that may be either necessary or unavoidable. A specific example of a situation in which this can occur is stress testing in finance, which often requires overwriting the elements of a matrix representing the correlation between various stocks. Several more examples are listed in [37] and [40].

In such a situation, we often want to find the nearest correlation matrix to the one we actually have, to act as the "true" matrix in further computations. This has long been of interest in certain areas, particularly in the finance industry. Algorithms for

computing the nearest correlation matrix exist and the NAG Library contains several routines that implement many of them. Unfortunately, even the best of the algorithms is still fairly expensive: for a matrix of order n, computing the nearest correlation matrix costs at least $70n^3/3$ flops using the most efficient method currently available [40].

Given the relatively high cost of actually computing the nearest correlation matrix to a given matrix, it would be useful to know the actual distance a priori, or at least good bounds for it. This could help inform the decision of whether or not it is necessary to revisit the construction of the matrix, or if the current matrix will suffice for certain applications.

For a matrix A, we define the distance to the nearest correlation matrix A_{NCM} by

$$d_{\text{corr}}(A) = \|A - A_{NCM}\|_F .$$

Finding upper and lower bounds for $d_{\text{corr}}(A)$ without finding A_{NCM} itself has attracted relatively little interest, however Higham and Strabić catalogue all of the known bounds and derive several new ones in [40]. In particular, they show how an upper bound on d_{corr} can be constructed using the modified Cholesky factorization.

7.2 Computing an upper bound with the modified Cholesky factorization

Suppose $A \in \mathbb{R}^{n\times n}$ is a symmetric, possibly indefinite matrix, with positive diagonal elements. Compute a modified Cholesky factorization of A,

$$A + E = P^T L D L^T P,$$

and let $\Lambda = \text{diag}(A + E)$. Then

$$d_{\text{corr}}(A) \leq \left\|A - \Lambda^{-1/2}(A + E)\Lambda^{-1/2}\right\|_F . \tag{7.1}$$

No proof is given here, but the basic idea is simple: the perturbed matrix $A + E$ produced by the modified Cholesky factorization is symmetric positive semidefinite, so by scaling to ensure it retains those properties whilst also having unit diagonal, we

can construct a correlation matrix that is (hopefully) close enough to A to act as a useful upper bound on the distance to the nearest correlation matrix.

The restriction that A must have positive diagonal elements is unlikely to be a hindrance in practice, since in this application we are generally considering matrices that are correlation matrices in every aspect apart from their definiteness, so we expect them to have unit diagonal.

For the Cheng-Higham and Moré-Sorensen modified Cholesky algorithms, the cost of computing the bound (7.1) is $2n^3/3$ flops, including the step of forming $A + E$ explicitly, which requires at least one dense matrix multiplication. This cost would therefore be smaller for modified Cholesky algorithms such as the SE or GMW that return E explicitly. Higham and Strabić compared the accuracy of the bound computed for the GMW, SE90, SE99 and CH algorithms on a set of invalid correlation matrices and concluded that the CH algorithm was generally more accurate than the others. We perform similar experiments in section 7.4 that suggest the CH algorithm is consistently more accurate than the MS algorithm as well. The precise reason for the difference in performance between the algorithms is unclear, however.

In their numerical experiments, Higham and Strabić found that bound (7.1) was generally within at most two orders of magnitude of $d_{\text{corr}}(A)$ and suggest that this will usually be adequate for practical applications.

7.3 Other bounds

7.3.1 Upper bounds

Two other methods for computing upper bounds for d_{corr} were found to be useful by Higham and Strabić. Both are based on spectral information, although they have different costs. The first is based on the idea of shrinking from Higham, Strabić and Šego [42], and is defined by the following theorem.

Theorem 7.1 *Let $A \in \mathbb{R}^{n\times n}$ be symmetric with unit diagonal and smallest eigenvalue $\lambda_n < 0$. Then*

$$d_{corr}(A) \leq \frac{|\lambda_n|}{1 + |\lambda_n|} \|A - I\|_F . \tag{7.2}$$

The cost of computing the bound (7.2) is $4n^3/3$ flops, twice that of the bound (7.1) computed using the modified Cholesky factorization. In their numerical experiments, Higham and Strabić found that the accuracy of both bounds was generally very similar.

The other upper bound Higham and Strabić found to be useful requires the nearest symmetric positive semidefinite matrix to A in the Frobenius norm, usually denoted A_+. From, for example, Theorem 3.1, we can see that if A has the spectral decomposition $A = Q\Lambda Q^T$, where Q is orthogonal and $\Lambda = \text{diag}(\lambda_i)$, then A_+ can be explicitly computed by

$$A_+ = Q\text{diag}(\max(\lambda_i, 0))Q^T. \tag{7.3}$$

We can now state the following theorem which gives another bound on $d_{\text{corr}}(A)$.

Theorem 7.2 *Let $A \in \mathbb{R}^{n\times n}$ be symmetric with positive diagonal elements. Then*

$$d_{corr}(A) \leq \left\| A - \tilde{A}_+ \right\|_F, \tag{7.4}$$

where $\tilde{A}_+ = D^{-1/2}A_+D^{-1/2}$, with $D = \text{diag}((A_+)_{ii})$.

No proof is given here but one can be found in [40]. Higham and Strabić found this to generally be the most accurate of the upper bounds they considered, always being within a factor of 4 of d_{corr} for all the matrices in their test set. However, the cost of calculating bound (7.4) is $17n^3/3$ flops, significantly more expensive than the upper bound obtained using the modified Cholesky factorization (or indeed shrinking).

7.3.2 Lower bounds

Three lower bounds for $d_{\text{corr}}(A)$ are given by Higham and Strabić in [40], however they conclude that only one of these is accurate enough to be of any practical use. Let A_+ be defined as in (7.3). Then we have

$$\|A - A_+\|_F \leq d_{\text{corr}}(A). \tag{7.5}$$

Note that, in particular, we also have

$$\|A - A_+\|_F = \left(\sum_{\lambda_i<0} \lambda_i^2 \right). \tag{7.6}$$

The cost of computing (7.5) using (7.6) is $4n^3/3$ flops. Higham and Strabić found that the bound (7.5) was always within a factor of 2.4 of d_{corr} for all the matrices in their test set.

7.4 Using the new implementation

To analyse the performance of the new `F01MDF` routine at computing the bound (7.1) and its relative efficiency compared to actually finding the nearest correlation matrix, we considered the set of 13 invalid correlation matrices from Higham and Strabić that we also made use of in Chapter 6; full details can be found at [41].

All experiments in this section were performed in the computing environment described in section 1.4, on a single core. As noted in Chapter 5, the `F01MDF` routine does not currently incorporate any parallel code, so we have restricted ourselves to a serial environment here. However, it is intended that the routine will be parallelized in the immediate future. Once this is done, we expect to see broadly similar results to those described here, although this should be investigated when possible.

To compute the bound (7.1), we first used the `F01MDF` routine to find the modified Cholesky factorization of the input matrix and then processed the outputs to compute the perturbed matrix $A+E$ and calculate the bound (7.1). The second step was done using the MATLAB function `ncm_upper`, included in Appendix E. As it is not currently intended for this code to be incorporated into the NAG Library, we view it as simply proof-of-concept: efforts were made to make it relatively efficient but it can surely still be optimized. As with `F01MDF`, `ncm_upper` also does not currently incorporate any parallel code. For all experiments utilising the `F01MDF` routine, we set `delta == sqrt(eps) * norm(A, 'fro')` and `uplo == 'L'`.

Table 7.1 records the computed bound (7.1) for each of the matrices in the set using both the Cheng-Higham and Moré-Sorensen modified Cholesky algorithms. We follow the convention of Higham and Strabić in denoting these by $\|A - CH\|_F$ and $\|A - MS\|_F$ respectively. The latter bound was computed using the MATLAB function `more_sorensen`—which implements the Moré-Sorensen modified Cholesky algorithm (see Appendix E)—run using the tolerance `delta == eps`.

Also included in Table 7.1 for comparison is the true distance to the nearest correlation matrix $d_{\text{corr}}(A)$. This was computed by using the NAG Toolbox for MATLAB routine `G02AA` [53], run with default parameters, to actually find the nearest correlation matrix. This routine is very well regarded and implements the most efficient algorithm currently known for computing the nearest correlation matrix to an input

Table 7.1: Bounds computed using the Cheng-Higham and Moré-Sorensen algorithms and the actual values of d_{corr}(A) for 13 invalid correlation matrices.

Matrix	Order	$\|A - MS\|_F$	$\|A - CH\|_F$	$d_{\text{corr}}(A)$
bccd16	3250	9.19e2	6.91e2	2.91e1
beyu11	12	1.21e-1	6.21e-2	9.60e-3
bhwi01	5	7.11e-1	4.30e-1	1.51e-1
cor1399	1399	7.37e1	4.52e1	2.10e1
cor3120	3120	5.25e2	4.40e2	5.44
fing97	7	1.76e-1	9.24e-2	4.91e-2
high02	3	8.45e-1	5.86e-1	5.28e-1
mmb13	6	3.27e1	3.04e1	3.03e1
tec03	4	9.94e-2	5.19e-2	3.74e-2
tyda99r1	8	3.17	2.36	1.40
tyda99r2	8	2.59	1.71	7.75e-1
tyda99r3	8	1.55	1.09	6.72e-1
usgs13	94	2.97	1.92	5.51e-2

matrix.

We see from Table 7.1 that the CH algorithm consistently achieves a more accurate bound than the MS algorithm for the matrices in our test set. We also observe that the bound (7.1) is always within at most two orders of magnitude of d_{corr} and usually tighter than this.

In Table 7.2, we give the average timings over 5 tests for `F01MDF`, `ncm_upper` and `G02AA`. We see that the bulk of the time taken for the `ncm_upper` function is spent actually constructing $A + E$, rather than computing the modified Cholesky factorization using the `F01MDF` routine. With the exception of `tec03`, `ncm_upper` is always faster than `G02AA`, and this is much more pronounced for the largest matrices. Table 7.3 displays the relative efficiency of the three functions for the three largest matrices in the set. We see that `ncm_upper` is between about 12 and 17 times as fast as `G02AA` for these matrices. Of course, unless computing the bound is considerably cheaper than actually computing the nearest correlation matrix, it is not worthwhile, so the open question is whether this is efficient enough for most applications. We reiterate here that `ncm_upper` is intended largely as proof-of-concept code: we believe that it can be optimized so that it is on average at least 25 times as efficient as `G02AA` for large matrices.

When this project was first proposed, it was debated whether or not to include

Table 7.2: Timings of three routines for 13 invalid correlation matrices.

Matrix	Order	F01MDF (seconds)	ncm_upper (seconds)	G02AA (seconds)
bccd16	3250	0.90	4.52	54.72
beyu11	12	5.06e-4	1.54e-3	1.56e-3
bhwi01	5	5.73e-4	9.97e-4	1.92e-3
cor1399	1399	0.17	0.64	9.93
cor3120	3120	1.05	4.80	82.41
fing97	7	3.48e-4	9.07e-4	2.28e-3
high02	3	2.84e-4	7.89e-4	3.62e-3
mmb13	6	3.15e-4	1.10e-3	3.80e-3
tec03	4	9.94e-3	5.19e-2	3.74e-2
tyda99r1	8	7.05e-4	9.38e-4	2.56e-3
tyda99r2	8	4.26e-4	1.23e-3	1.40e-3
tyda99r3	8	4.89 e-4	9.32e-4	1.39e-3
usgs13	94	2.27e-3	5.78e-3	7.59e-3

Table 7.3: Efficiency of F01MDF and ncm_upper relative to G02AA for the largest invalid correlation matrices.

Matrix	Order	G02AA / F01MDF	G02AA / ncm_upper
bccd16	3250	60.8	12.1
cor1399	1399	58.4	15.5
cor3120	3120	78.5	17.1

another routine in the NAG Library that makes use of F01MDF to calculate the bound (7.1) efficiently. This was dismissed because processing the output of F01MDF to compute (7.1) was considered too simple to justify an entire routine. However, it may well prove to be the case that users prefer an efficient black box routine for this rather than having to write their own code. This decision should perhaps be re-evaluated after the first release of the library including the F01MDF routine.

Chapter 8

Other Applications of the Modified Cholesky Factorization

In this chapter we discuss applications of the modified Cholesky factorization other than that discussed in the Chapter 7. We will be brief here but reference will be made to other sources that provide much more in-depth information.

8.1 Finding directions of negative descent

This is often regarded as the primary application of modified Cholesky algorithms [21] and motivated the creation of the algorithms of Gill, Murray and Wright [27], Schnabel and Eskow [70], and Moré and Sorensen [47] discussed in Chapters 2 and 3.

Suppose we wish to minimize a function $f(x)$, where $x \in \mathbb{R}^n$. Then a common and effective way to do this is to start at any given point $x_0 \in \mathbb{R}^n$ and generate a series of iterates x_i such that $f(x_i)$ is a decreasing sequence (and therefore should eventually converge to the minimum). One way to do this is to define successive iterates by

$$x_{i+1} = x_i + \alpha_i p_i,$$

where α_i is a constant known as the *step length* and $p_i \in \mathbb{R}^n$ is a direction vector along which the function f is decreasing[1].

If $H_f(x_i)$ is the Hessian matrix of $f(x)$ evaluated at x_i and is positive definite, then

[1]The step length is of no further relevance for our discussion but is actually extremely important in practice.

we can always find such a p_i by solving the system

$$H_f(x_i)p_i = -\nabla f(x_i), \tag{8.1}$$

where ∇f is the gradient of f. This is known as Newton's method and, when it is applicable, can be an extremely efficient way to minimize the function $f(x)$, since it converges to the minimum at a quadratic rate. Given that $H_f(x_i)$ is positive definite, the most efficient way to solve the system (8.1) itself may be to use the Cholesky factorization (see Chapter 1).

However, if the Hessian matrix $H_f(x_i)$ is not positive definite, then we obviously cannot use a Cholesky factorization and, further, solving (8.1) does not even necessarily give a descent direction [30]. The basic idea motivating the applications in this section is that if we instead use the *modified* Cholesky factorization to find $\tilde{H}_f(x_i) = H_f(x_i) + \Delta H_f(x_i)$, where $\Delta H_f(x_i)$ is chosen to make $\tilde{H}_f(x_i)$ positive definite (i.e., it is E in our notation from previous chapters), then if we solve

$$\tilde{H}_f(x_i)\tilde{p}_i = -\nabla f(x_i),$$

for $\tilde{p}_i$, we may have a descent direction for $f(x)$. Ensuring that it actually *is* a descent direction is slightly more complicated than our simplified description suggests but can nevertheless be done [47].

Many variants of Newton's method exist and it is possible that the modified Cholesky factorization may be applicable in some manner to many of them as well. More broadly, the modified Cholesky factorization is useful in any other application in which we wish to find a negative descent direction from an indefinite matrix, such as interior-point algorithms for linear programming [76].

We will remark here that the use of modified Cholesky factorization in this context does not appear to be as popular as it once was and other approaches such as trust region methods [63, Chapter 4] or quasi-Newton [63, Chapter 6] methods that do not require fixing the Hessian have become more prominent in recent years (although there may still be situations in which they are unsuitable and the modified Cholesky approach is preferred [4]). Of course, it could well be the case that this trend was in part due to the dearth of software implementations of the modified Cholesky factorization and that our routine may therefore lead to more interest in this approach.

8.2 Preconditioners

A *preconditioner* for a matrix A is another matrix Q chosen so that $Q^{-1}A$ is more suitable for use in a numerical method than the original matrix A. For example, if A is extremely ill conditioned, then rather than solving the linear system $Ax = b$, we may choose a preconditioner Q such $Q^{-1}A$ is better conditioned than A and solve the system $Q^{-1}Ax = Q^{-1}b$ instead.

In certain applications, it can be the case that we need to construct a positive definite preconditioner, or that a derived preconditioner is not positive definite when it should be; there are many ways and contexts in which this can occur so we will not describe them here but more information can be found at the references given. The use of the modified Cholesky factorization to construct a preconditioner in a Newton-like method for large-scale problems in computational chemistry is discussed here [68]. More general use for large-scale optimization problems is considered here [14]. Use as a preconditioner in a conjugate gradient method is discussed in [26] and was implemented in the LANCELOT software package [13].

8.3 Other uses

A *covariance matrix* is very similar to a correlation matrix but differs in that the (i, j) element of the matrix represents the covariance between the random variables x_i and x_j, rather than the correlation [72, p. 24]. Use of the modified Cholesky factorization to estimate the covariance matrix for a mixture of two normal distributions is proposed here [74]. It is unclear to us however if the method is truly successful.

Schnabel and Eskow suggest that the ∞-norm of the perturbation matrix E produced by their modified Cholesky algorithm (or its later revision) could be used as a low-cost way of estimating the smallest eigenvalue of A in trust region methods in optimization [70]. However, we are uncertain whether it has ever actually been used for this purpose in any applications.

Chapter 9

Concluding Remarks

Overall, we believe that the Cheng-Higham modified Cholesky algorithm was the right choice for the NAG Library and that the `F01MDF` routine is an efficient implementation. However, it should be regarded as imperative that the routine is parallelized before it is included in any commercial release of the NAG Library, in order to better take advantage of modern computer architectures.

For the the nearest correlation matrix problem discussed in Chapter 7 in particular, time will tell whether using the `F01MDF` routine to compute an upper bound proves to be both accurate and efficient enough to be of use in practical applications. The decision of whether to include the calculation of the bound as an actual NAG Library routine should also be re-evaluated pending user feedback after the first release incorporating the `F01MDF` routine: it may well prove to be the case that users prefer to have a NAG Library routine for doing this, to alleviate the need to create an efficient code of their own.

The effect that incorporating Aasen's LTL^T factorization into the `F01MDF` routine (as suggested by Fang and O'Leary and detailed in section 3.2) has on the accuracy of the upper bound (7.1) should also be investigated, especially if NAG do ultimately decide to include a routine that computes the bound in the library. We had concluded that there was no advantage in doing this in general but—as noted in section 4.2—using different modified Cholesky algorithms generally leads to different values for the bound (7.1). As we are uncertain as to the precise cause of this behaviour, it is entirely possible that the LTL^T variant algorithm could improve the accuracy of the bound (although it should also be emphasised here that there is also no reason to assume a

priori that it actually will). If this does prove to be the case and NAG also decide to create a new routine to compute the upper bound, then the possibility of incorporating the LTL^T factorization into the new routine should be considered.

Appendix A

Numerical Stability and Computer Arithmetic

Suppose we have an an approximation $\hat{y}$ to the true solution y of the problem $y = f(x)$. The *forward error* of an algorithm is the error (both absolute and relative) in using $\hat{y}$ to approximate y. The *backward error* is the ratio $|\Delta x|/|x|$, where Δx is the smallest perturbation we can make to x such that $\hat{y} = f(x + \Delta x)$. If we define an acceptably small tolerance ϵ for the forward error and a similar acceptable tolerance η for the backward error of a problem then an algorithm is *numerically stable* if a result of the form

$$\hat{y} + \Delta y = f(x + \Delta x), \qquad |\Delta y| \leq \epsilon |y|, |\Delta x| \leq \eta |x|,$$

holds for all x. An algorithm is *backward stable* if the backward error of the solution $\hat{y}$ that it computes is bounded by one. In particular, a backward stable algorithm is also numerically stable [36, p. 7]. Numerical stability is in general an extremely desirable property for an algorithm to possess.

The most common way modern machines represent real numbers—and therefore perform calculations with them—is through a *floating point* number system. However, the machine can only accurately represent a small subset of the real numbers; these are called floating point numbers. Each nonzero floating point number $\hat{x}$ is of the form

$$\hat{x} = \pm m \times \beta^{e-t},$$

where m satisfying $0 \leq m \leq \beta^t - 1$ is called the *mantissa*, β is the *base* (almost invariably 2), t is the *precision* and e is the *exponent*. The number system itself is

characterised by the precision and the base, as well as the minimum and maximum values of the exponent.

The quantity *machine epsilon*, which is often denoted by ϵ_M, is the distance from 1.0 in particular to the next largest floating point number [36, p. 37], i.e., $\epsilon_M = \beta^{1-t}$. A related quantity is the *unit roundoff*. This is the maximum relative error in using a floating point number $\hat{x}$ to represent a real number x; it is usually denoted by u and is given by $u = \frac{1}{2}\beta^{1-t} = \epsilon_M/2$. The unit roundoff is an extremely important quantity in rounding error analysis and appears frequently throughout this dissertation.

See [36, Chapter 2] for a much more detailed treatment of the numerical issues inherent in computer arithmetic.

Appendix B

Matrix Norms and the Condition Number

Given a vector norm $\|\cdot\| : \mathbb{R} \to \mathbb{R}$, we can define a *subordinate* matrix norm $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ by

$$\|A\| = \max_{x\neq 0} \frac{\|Ax\|}{\|x\|}.$$

The most important subordinate norms in numerical linear algebra are the *matrix p-norms*, which are subordinate to the vector p-norms and therefore defined by

$$\|A\|_p = \max_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

We most frequently make use of the matrix 1-, 2- and ∞-norms, for which the following results allowing them to be practically computed for a matrix $A \in \mathbb{R}^{n\times n}$ can be established.

$$\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}|, \quad \text{the maximum column sum,}$$

$$\|A\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}|, \quad \text{the maximum row sum,}$$

$$\|A\|_2 = (\lambda_{\max}(A^T A))^{1/2},$$

where $\lambda_{\max}(B)$ denotes the largest eigenvalue of the matrix B.

In addition to these norms, we also make use of the Frobenius norm $\|\cdot\|_F : \mathbb{R}^{m\times n} \to \mathbb{R}$, which is not a subordinate norm but is instead defined by

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2} = (\text{trace}(A^T A)).$$

The *condition number* $\kappa(A)$ of a matrix $A \in \mathbb{R}^{n\times n}$ is defined with respect to a norm by $\kappa(A) = \|A\| \cdot \|A^{-1}\|$. If A is singular then its condition number is defined to be infinite. For any subordinate norm, the condition number obeys the bound $\kappa(A) \geq 1$ and for the Frobenius norm it can be shown that $\kappa_F(A) \geq n^{1/2}$.

The condition number of a square matrix A is an extremely important quantity when describing the sensitivity of the linear system $Ax = b$ to small changes in the data. Assume that $Ax = b$ and $(A + \Delta A)(x + \Delta x) = b + \Delta b$, and suppose that $\kappa(A)\gamma < 1$, where $\|\Delta A\| \leq \gamma \|A\|$ and $\|\Delta b\| \leq \gamma \|b\|$. Then we have

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{2\kappa(A)\gamma}{1 - \kappa(A)\gamma}.$$

Essentially, this means that the relative error in x is bounded by the condition number of A multiplied by the relative errors in A and b [28, p. 81].

A problem is *well conditioned* if small perturbations in the data make correspondingly small changes to the solution; otherwise, we say it is *ill conditioned* (so the condition number $\kappa(A)$ is a measure of the conditioning of $Ax = b$). In linear algebra, we say that a *matrix* is well conditioned if it has a relatively small condition number; otherwise we say it is ill conditioned. Precisely how we define "small" may depend on the problem but we usually seek to minimize the condition number of any matrix that we use in computations.

We will most commonly use the 2-norm condition number κ_2 in this dissertation. A real matrix A is *normal* if $A^TA = AA^T$. In particular, symmetric matrices are clearly normal. For all normal matrices, we have

$$\kappa_2(A) = \frac{|\lambda_{\max}(A)|}{|\lambda_{\min}(A)|}.$$

Appendix C

Prominent Linear Algebra Libraries

We make frequent reference to the following linear algebra software libraries throughout this dissertation, so we feel a brief description of each may be useful for the reader.

BLAS[1] is not actually a software library but a specification for extremely efficient, portable routines that perform basic linear algebra operations. Level 1 BLAS perform scalar, vector and vector-vector operations; Level 2, matrix-vector operations; and Level 3, matrix-matrix operations. They are the standard building blocks for these operations in linear algebra software libraries.

LINPACK[2] is a library of Fortran subroutines for solving linear equations and linear least-squares problems. Created for use on supercomputers in the 1970s [17], it is no longer as popular as it once was and has now been largely superseded by LAPACK.

LAPACK[3] is also a Fortran library but with a wider focus than LINPACK, containing subroutines for solving many different numerical linear algebra problems. It was originally conceived as a project to improve upon existing software libraries by taking better advantage of the memory hierarchies on modern machines. LAPACK routines do this by utilising block matrix operations (see Appendix D) to a large extent. They are designed to use calls to the BLAS as extensively as possible, especially Level 3 BLAS.

[1] Basic Linear Algebra Subprograms.
[2] LINear algebra PACKage.
[3] Linear Algebra PACKage.

Appendix D

Blocking and Parallelism

A *blocked* algorithm is one that performs operations on blocks of a matrix rather than individual elements, rows or columns. Blocked algorithms are mathematically identical to unblocked ones but try to take advantage of modern computer memory hierarchies to be considerably more efficient in practice. This is done by maximizing the reuse of data in the faster levels of memory: by moving a block of a matrix into cache, it can be used repeatedly with no further look-up costs. In particular, blocked matrix multiplication can be much more efficient than the unblocked version and hence blocked algorithms generally make heavy use of it. Indeed, Golub and Van Loan in [28, Chapter 1] state that "by a blocked algorithm we essentially mean one that is rich in matrix multiplication."

Parallel computing is the use of multiple computer resources simultaneously to solve a single problem. Computer architectures that allow parallel computing have become ubiquitous in recent years and are now the industry standard for high performance machines [5]. It has thus become increasingly important that numerical algorithms and software adapt to this paradigm and that scope for parallelism should be considered when designing or evaluating an algorithm.

Appendix E

Code

This appendix contains MATLAB code for three short functions that we made use of in this dissertation.

E.1 expensive_matrix

```
function A = expensive_matrix(n)
%expensive_matrix Computes a matrix that requires O(n^3)
   comparisons for LDL' factorization with rook pivoting.
% A = expensive_matrix(n) returns an n-by-n matrix A for
   which the rook (bounded Bunch-Kaufman) pivoting
% strategy of Ashcraft, Grimes and Lewis requires O(n^3)
   comparisons to determine the pivots.

% This code is adapted from code given in:
% N. J. Higham. Accuracy and Stability of Numerical
   Algorithms. Second edition, Society for Industrial and
% Applied Mathematicians, Philadelphia,
% PA, USA, 2002. xxx+680pp. ISBN 0-89871-521-0 [page 228].

% Author: Thomas McSweeney, 2017.

A = zeros(n);
```

```
A(n, 1) = 2;
for i = 2:n-1
    A(i + 1, i) = n - i + 2;
end
A = A + A';
A(2, 2) = n;
end
```

E.2 more_sorensen

```
function [L, DMC, P, D] = more_sorensen(A,delta)
%more_sorensen  More and Sorensen modified Cholesky
   algorithm based on LDL' factorization.
%   [L D,P,D0] = more_sorensen(A,delta) computes the
   modified Cholesky factorization P*(A + E)*P' = L*D*L',
%   where P is a permutation matrix, L is unit lower
   triangular, and D is block diagonal and positive
%   definite with 1-by-1 and 2-by-2 diagonal blocks.  Thus
   A+E is symmetric positive definite, but E is
%   not explicitly computed.  Also returned is a block
   diagonal D0 such that P*A*P' = L*D0*L'.  If A is
%   sufficiently positive definite then E = 0 and D = D0.
   The algorithm sets the smallest eigenvalue of D
%   to the tolerance delta, which defaults to eps.
%   The LDL' factorization is computed using a symmetric
   form of rook pivoting proposed by Ashcraft, Grimes
%   and Lewis.

%   This code is a modification of an existing code of
   Cheng and Higham, altered to use the modified
%   Cholesky algorithm of More and Sorensen rather than
   that of Cheng and Higham; the original code is
```

```
%   available here: https://github.com/higham/modified-
    cholesky.

%   Reference:
%   J. J. More and D. C. Sorensen. On the use of directions
    of negative curvature in a modified Newton
%   method. Mathematical Programming, 16(1):1-20, 1979.

%   Authors: Bobby Cheng and Nick Higham, 1996; revised
    2015.
%   Modified by Thomas McSweeney, 2017.

if ~ishermitian(A), error('Must supply symmetric matrix.'),
    end
if nargin < 2, delta = eps; end

n = max(size(A));

[L,D,p] = ldl(A,'vector');
DMC = eye(n);

% (More and Sorensen) modified Cholesky perturbations.
k = 1;
while k <= n

      if k == n || D(k,k+1) == 0 % 1-by-1 block

         if abs(D(k,k)) <= delta
            DMC(k,k) = delta;
         else
            DMC(k,k) = abs(D(k,k));
         end
```

```
            k = k+1;

        else % 2-by-2 block

            E = D(k:k+1,k:k+1);
            [U,T] = eig(E);
            T = abs(T);
            for ii = 1:2
                if T(ii,ii) <= delta
                   T(ii,ii) = delta;
                end
            end
            temp = U*T*U';
            DMC(k:k+1,k:k+1) = (temp + temp')/2;  % Ensure
               symmetric.
            k = k + 2;
        end
end
if nargout >= 3, P = eye(n); P = P(p,:);
end
```

E.3 ncm_upper

```
function [upper] = ncm_upper(A)
%ncm_upper Upper bound on the distance to the nearest
   correlation matrix.
%   [upper] = ncm_upper(A) computes an upper bound for the
   distance to the nearest correlation matrix
%   to A, using the modified Cholesky factorization of
   Cheng and Higham in the manner suggested by
%   Higham and Strabic. We make use of the new f01md
   routine from the NAG Library Toolbox for MATLAB
```

```
%    in order to compute the factorization.

%    Reference:
%    N. J. Higham and N. Strabic. Bounds for the distance to
     the nearest correlation matrix.
%    SIAM J. Matrix Anal. Appl.,37(3):1088-1102, 2016. doi:
    10.1137/15M1052007.

%    Author: Thomas McSweeney, 2017.

if ~ishermitian(A), error('The matrix must be symmetric.'),
     end
if any(diag(A) <= 0), error('The matrix must have strictly
    positive diagonal entries.'), end

n = length(A);
delta = sqrt(eps)*norm(A,'fro');
uplo = 'L';

original = A; % Copy A before we overwrite it, for use
    later.

% Compute the modified Cholesky factorization using f01md.
[A, offdiag, ipiv, ifail] = f01md(uplo, A, delta);

if uplo == 'L'
     offset = 0;
     M = tril(A) - diag(diag(A)) + eye(n);
else
     offset = 1;
     M = triu(A) - diag(diag(A)) + eye(n);
end
```

```
% M = L or U.

% Apply the permutations directly to M, i.e. M = P'*L or M
   = P'*U.
for i = n: -1:1
    if ipiv(i) < 0
        M([i -ipiv(i)], :) = M([-ipiv(i) i], :);
    else
        M([i ipiv(i)], :) = M([ipiv(i) i], :);
    end
end

T = M'; % T = L'P or U'P (depending on uplo).

% Do the block diagonal matrix multiplication M*D = P'*L*D
   or M = P'*U*D.
j = 1;
while j < n
    if (ipiv(j)< 0) && (ipiv(j + 1) < 0)
        dummy = M(:, j);
        M(:, j) = dummy * A(j, j) + M(:, j + 1) * offdiag(j
            + offset);
        M(:, j + 1) = dummy * offdiag(j + offset) + M(:, j
           + 1) * A(j + 1, j + 1);
        j = j + 2;
    else
        M(:, j) = M(:, j) * A(j, j);
        j = j + 1;
    end
end
if (ipiv(n) > 0)
    M(:, n) = M(:, n) * A(n, n);
```

```
end

% Form A + E = A_pert explictly.
A_pert = M * T;

% Scale A + E to make it a correlation matrix.
dg = sqrt(diag(A_pert));

for i = 1:n
    A_pert(:,i) = A_pert(:,i) ./ dg(i);
    A_pert(i,:) = A_pert(i,:) ./ dg(i);
end

% Compute the bound.
upper = norm(original - A_pert, 'fro');
end
```

Bibliography

[1] Jan Ole Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT Numerical Mathematics*, 11(3):233–242, 1971.

[2] Edward Anderson, Zhaojun Bai, Christian Bischof, L. Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1999.

[3] Edward Anderson and Jack J. Dongarra. Evaluating block algorithm variants in LAPACK. In *Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing, Chicago, Illinois, USA, December 11-13, 1989*, 1989, pages 3–8.

[4] Pierre Apkarian, Dominikus Noll, and Hoang Duong Tuan. Fixed-order H_∞ control design via a partially augmented Lagrangian method. *International Journal of Robust and Nonlinear Control*, 13(12):1137–1148, 2003.

[5] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. The landscape of parallel computing research: A view from Berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[6] Cleve Ashcraft, Roger G. Grimes, and John G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM Journal on Matrix Analysis and Applications*, 20(2):513–561, 1998.

[7] Grey Ballard, Dulceneia Becker, James Demmel, Jack Dongarra, Alex Druinsky, Inon Peled, Oded Schwartz, Sivan Toledo, and Ichitaro Yamazaki. Implementing a

blocked Aasen's algorithm with a dynamic scheduler on multicore architectures. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, IEEE, 2013, pages 895–907.

[8] Grey Ballard, Dulceneia Becker, James Demmel, Jack Dongarra, Alex Druinsky, Inon Peled, Oded Schwartz, Sivan Toledo, and Ichitaro Yamazaki. Communication-avoiding symmetric-indefinite factorization. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1364–1406, 2014.

[9] James R. Bunch and Linda Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, pages 163–179, 1977.

[10] James R. Bunch and Beresford N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8 (4):639–655, 1971.

[11] CHARMM. Chemistry at Harvard Macromolecular Mechanics. [Online; accessed 08-August-2017].

[12] Sheung Hun Cheng and Nicholas J. Higham. A modified Cholesky algorithm based on a symmetric indefinite factorization. *SIAM J. Matrix Anal. Appl.*, 19 (4):1097–1110, 1998.

[13] Andrew R. Conn, G. I. M. Gould, and Philippe L. Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*. Springer Science & Business Media, 2013.

[14] Michel J. Daydé. A block version of the Eskow-Schnabel modified Cholesky factorization. Rapport de recherche RT/APO/95/8, Institut National Polytechnique de Toulouse, Toulouse, France, 1996.

[15] Michel J. Daydé, Jean-Yves L'Excellent, and Nicholas I.M. Gould. Element-by-element preconditioners for large partially separable optimization problems. *SIAM Journal on Scientific Computing*, 18(6):1767–1787, 1997.

[16] John E. Dennis Jr and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, 1996.

[17] Jack J. Dongarra, Cleve B. Moler, James R. Bunch, and Gilbert W. Stewart. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1979.

[18] Iain S. Duff, Albert M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 2017.

[19] Iain S. Duff, Nick I.M. Gould, John K. Reid, Jennifer A. Scott, and Kathryn Turner. The factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis*, 11(2):181–204, 1991.

[20] Iain S. Duff, John K. Reid, N. Munksgaard, and Hans B. Nielsen. Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. *IMA Journal of Applied Mathematics*, 23(2):235–250, 1979.

[21] Haw-ren Fang and Dianne P. O'Leary. Modified Cholesky algorithms: a catalog with new approaches. *Mathematical Programming*, 115(2):319–349, 2008.

[22] Haw-ren Fang and Dianne P. O'Leary. Euclidean distance matrix completion problems. *Optimization Methods and Software*, 27(4-5):695–717, 2012.

[23] Leslie V. Foster. The growth factor and efficiency of Gaussian elimination with rook pivoting. *Journal of Computational and Applied Mathematics*, 86(1):177–194, 1997.

[24] David M. Gay, Michael L. Overton, and Margaret H. Wright. A primal-dual interior method for nonconvex nonlinear programming. In *Advances in Nonlinear Programming*, Springer, 1998, pages 31–56.

[25] Philip E. Gill and Walter Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Mathematical Programming*, 7(1):311–350, 1974.

[26] Philip E. Gill, Walter Murray, Dulce B. Ponceleón, and Michael A. Saunders. Preconditioners for indefinite systems arising in optimization. *SIAM journal on matrix analysis and applications*, 13(1):292–311, 1992.

[27] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1981.

[28] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Third edition, Johns Hopkins, Baltimore, 1996.

[29] M. S. Grewal and A. P. Andrews. *Kalman Filtering: Theory and Practice with MATLAB*. Wiley - IEEE. Wiley, 2015. ISBN 9781118984963.

[30] Raphael T. Haftka and Zafer Gürdal. *Elements of Structural Optimization*. Springer Science & Business Media, 2012. 128 pp.

[31] Desmond J. Higham and Nicholas J. Higham. *MATLAB Guide*. Third edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2017. xxvi+476 pp. ISBN 978-1-61197-465-2.

[32] Nicholas J. Higham. The Matrix Computation Toolbox. `http://www.maths.manchester.ac.uk/~higham/mctoolbox`.

[33] Nicholas J. Higham. Computing the polar decomposition—with applications. *SIAM J. Sci. Statist. Comput.*, 7(4):1160–1174, 1986.

[34] Nicholas J. Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear Algebra Appl.*, 103:103–118, 1988.

[35] Nicholas J. Higham. Analysis of the Cholesky decomposition of a semi-definite matrix. In *Reliable Numerical Computation*, M. G. Cox and S. J. Hammarling, editors, Oxford University Press, 1990, pages 161–185.

[36] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0.

[37] Nicholas J. Higham. Computing the nearest correlation matrix—A problem from finance. *IMA J. Numer. Anal.*, 22(3):329–343, 2002.

[38] Nicholas J. Higham. Cholesky factorization. *WIREs Comp. Stat.*, 1(2):251–254, 2009.

[39] Nicholas J. Higham and Sheung Hun Cheung. Modified Cholesky factorization. [Online; accessed 16-August-2017].

[40] Nicholas J. Higham and Nataša Strabić. Bounds for the distance to the nearest correlation matrix. *SIAM J. Matrix Anal. Appl.*, 37(3):1088–1102, 2016.

[41] Nicholas J. Higham and Nataša Strabić. Collection of invalid correlation matrices, 2016. [Online; accessed 17-August-2017].

[42] Nicholas J. Higham, Nataša Strabić, and Vedran Šego. Restoring definiteness via shrinking, with an application to correlation matrices with a fixed block. *SIAM Rev.*, 58(2):245–263, 2016.

[43] Intel. Intel Math Kernel Library. [Online; accessed 16-August-2017].

[44] Intel. Intel Xeon Processor E5-2670. [Online; accessed 16-August-2017].

[45] Intel. Matrix layout for LAPACK routines. [Online; accessed 14-August-2017].

[46] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *AeroSense'97*, International Society for Optics and Photonics, 1997, pages 182–193.

[47] Jorge J. Moré and Danny C. Sorensen. On the use of directions of negative curvature in a modified Newton method. *Mathematical Programming*, 16(1):1–20, 1979.

[48] NAG. Case studies. [Online; accessed 16-August-2017].

[49] NAG. How to use the NAG library and documentation. [Online; accessed 14-August-2017].

[50] NAG. NAG Library Manual, Mark 26. [Online; accessed 14-August-2017].

[51] NAG. The NAG Toolbox for MATLAB. [Online; accessed 24-August-2017].

[52] NAG. NAG Toolbox for MATLAB, User Manual, Mark 25, F07FD. [Online; accessed 07-September-2017].

[53] NAG. NAG Toolbox for MATLAB, User Manual, Mark 25, G02AA. [Online; accessed 29-August-2017].

[54] NAG. Who we are and what we do. [Online; accessed 16-August-2017].

[55] Larry Neal and George Poole. A geometric analysis of Gaussian elimination II. *Linear Algebra and its Applications*, 173:239–264, 1992.

[56] Netlib. Determining the block size for block algorithms. [Online; accessed 06-September-2017].

[57] Netlib. LAPACK 3.7.1 DLASYF_RK. [Online; accessed 16-August-2017].

[58] Netlib. LAPACK 3.7.1 DSYEVD. [Online; accessed 16-August-2017].

[59] Netlib. LAPACK 3.7.1 DSYTRF2_RK. [Online; accessed 16-August-2017].

[60] Netlib. LAPACK 3.7.1 DSYTRF_AA. [Online; accessed 25-July-2017].

[61] Netlib. LAPACK 3.7.1 DSYTRF_RK. [Online; accessed 25-July-2017].

[62] Netlib. LAPACK 3.7.1 ILAENV. [Online; accessed 06-September-2017].

[63] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. ISBN 9780387303031.

[64] B. Parlett and J. Reid. On the solution of a system of linear equations whose matrix is symmetric but not definite. *BIT Numerical Mathematics*, 10(3):386–397, 1970.

[65] Miroslav Rozložník, Gil Shklarski, and Sivan Toledo. Partitioned triangular tridiagonalization. *ACM Transactions on Mathematical Software (TOMS)*, 37(4):38, 2011.

[66] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method*. Third edition, John Wiley & Sons, 2016.

[67] Tamar Schlick. TNPACK. [Online; accessed 08-August-2017].

[68] Tamar Schlick. Modified Cholesky factorizations for sparse preconditioners. *SIAM Journal on Scientific Computing*, 14(2):424–445, 1993.

[69] Tamar Schlick and Aaron Fogelson. TNPACK – A truncated Newton minimization package for large-scale problems: I. Algorithm and usage. *ACM Transactions on Mathematical Software (TOMS)*, 18(1):46–70, 1992.

[70] Robert B. Schnabel and Elizabeth Eskow. A new modified Cholesky factorization. *SIAM Journal on Scientific and Statistical Computing*, 11(6):1136–1158, 1990.

[71] Robert B. Schnabel and Elizabeth Eskow. A revised modified Cholesky factorization algorithm. *SIAM Journal on Optimization*, 9(4):1135–1148, 1999.

[72] James R. Schott. *Matrix Analysis for Statistics*. John Wiley & Sons, 2016.

[73] Gilbert W. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, 17 (3):403–409, 1980.

[74] Jin Wang and Chunlei Liu. Generating multivariate mixture of normal distributions using a modified Cholesky decomposition. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, IEEE, 2006, pages 342–347.

[75] J. H. Wilkinson. A priori error analysis of algebraic processes. In *Proc. International Congress of Mathematicians, Moscow 1966*, I.G. Petrovsky, editor, Mir Publishers, Moscow, 1968, pages 629–640.

[76] Stephen J. Wright. Modified Cholesky factorizations in interior-point algorithms for linear programming. *SIAM Journal on Optimization*, 9(4):1159–1191, 1999.

[77] Dexuan Xie and Tamar Schlick. Efficient implementation of the truncated-Newton algorithm for large-scale chemistry applications. *SIAM Journal on Optimization*, 10(1):132–154, 1999.

[78] Dexuan Xie and Tamar Schlick. Remark on algorithm 702– the updated truncated Newton minimization package. *ACM Transactions on Mathematical Software (TOMS)*, 25(1):108–122, 1999.