THE NUMERICAL SOLUTION OF QUADRATIC MATRIX EQUATIONS

by

KHALID MAHMOOD

Submitted in partial fulfilment of the requirements
for the award of

Doctor of Philosophy of the Loughborough University of Technology

October 1990

ABSTRACT

Methods for computing an efficient and accurate numerical solution of the real monic unilateral quadratic matrix equation,

$$X^2 + PX + Q = 0$$

are few. They are not guaranteed to work on all problems. One of the methods performs a sequence of Newton iterations until convergence occurs whilst another is a matrix analogy of the scalar polynomial algorithm. The former fails from a poor starting point and the latter fails if no dominant solution exists. A recent approach, the Elimination method, is analysed and shown to work on problems for which other methods fail. The method requires the coefficients of the characteristic polynomial of a matrix to be computed and to this end a comparative numerical analysis of a number of methods for computing the coefficients is performed. A new minimisation approach for solving the quadratic matrix equation is proposed and shown to compare very favourably with existing methods.

A special case of the quadratic matrix equation is the matrix square root problem, where $P = 0$. There have been a number of method proposed for it's solution, the more successful ones being based upon Newton iterations or the Schur factorisation. The Elimination method is used as a basis for generating three methods for solving the matrix square root problem. By means of a numerical analysis and results it is shown that for small order problems the Elimination methods compare favourably with the existing methods.

The algebraic Riccati equation of stochastic and optimal control is,

$$A^T X + XA - XBR^{-1}B^T X + H = 0$$

where the solution of interest is the symmetric non-negative definite one. The current methods are based on Newton iterations or the determination of the invariant subspace of the associated Hamiltonian matrix. A new method based on a reformulation of Newton's method is presented. The method reduces the work involved at each iteration by introducing a Schur factorisation and a sparse linear system solver. Numerical results suggest that it may compare favourably with well-established methods.

Central to the numerical issues are the discussions on conditioning, stability and accuracy. For a method to yield accurate results, the problem must be well-conditioned and the method that solves the problem must be stable-consequently discussions on conditioning and stability feature heavily in this thesis.

The units of measure we use to compare the speed of the methods are the operations count and the Central Processor Unit (CPU) time. We show how the CPU time accurately reflects the amount of work done by an algorithm and that the operations counts of the algorithms correspond with the respective CPU times.

ACKNOWLEDGEMENTS

Dedicated to my family for their support and encouragement.

I am especially thankful to my supervisor Professor C. Storey for his guidance and advice.

I thank both Professor C. Storey and Dr G. Evans for help in revising my thesis.

I greatly appreciate the efforts of Miss Helen Sherwood and Miss Louise Howard for the speedy and professional typing.

CONTENTS

		Page
ABSTRACT		i
ACKNOWLEDGEMENTS		ii
CONTENTS		iii
CHAPTER 1	INTRODUCTION	1
1.1	Outline of the chapters	1
1.2	Background theory in numerical matrix algebra	3
1.3	Conditioning and Stability	12
1.4	Numerical consideration of relevant problems	21
1.5	Systems of non-linear equations	31
1.6	Some theory on quadratic matrix equations	37
CHAPTER 2	PERTURBATION ANALYSIS	44
2.1	The derivative of $F(X)$	45
2.2	Conditioning of the problem	48
2.3	Some examples of perturbed problems	55
CHAPTER 3	COMPUTING THE CHARACTERISTIC POLYNOMIAL OF A MATRIX	61
3.1	Introduction	61
3.2	Condition of the problem	62
3.3	LeVerriers method	63
3.4	Stable LeVerriers method	65
3.5	Danilevski's method and an extension	67
3.6	Block Frobenius method	70
3.7	Krylov's method	73
3.8	Characteristic polynomial of matrices with distinct eigenvalues	77
3.9	Conclusions	84

CHAPTER 4	THE ELIMINATION METHOD WITH APPLICATIONS	86
4.1	Introduction	86
4.2	The Elimination method	87
4.3	The quadratic matrix equation	99
4.4	Elimination method for the square root problem	108
4.5	Applications to matrix square root solutions	112
CHAPTER 5	CURRENT METHODS FOR QUADRATIC MATRIX EQUATIONS	123
5.1	Introduction	124
5.2	Methods for the algebraic Riccati equation	126
5.3	The unilateral quadratic matrix equation	138
5.4	The matrix square root problem	144
5.5	Minimisation of the constituent equations	149
CHAPTER 6	PRACTICAL METHODS FOR THE QUADRATIC MATRIX EQUATIONS BASED ON NEWTON'S METHOD	155
6.1	Introduction	155
6.2	Newton's method applied to the algebraic Riccati equation	156
6.3	Towards a globally convergent Newton method for the unilateral quadratic matrix equation	168
CHAPTER 7	EXAMPLES, RESULTS, COMPARISONS	181
7.1	Introduction	181
7.2	Methods for computing the characteristic polynomial	185
7.3	Methods for solving the quadratic matrix equation	189
7.4	Methods for the matrix square problem	199
7.5	Methods for the algebraic Riccati equation	204
CHAPTER 8	CONCLUSIONS	207
REFERENCES		
APPENDICES		

Acknowledgement and comment on the error analysis

A fundamental point concerning the error analysis has been pointed out by the external examiner, Dr. Nicholls. The floating point analysis as presented in Chapter 1, section 1.3, results in the inequality

$$|fl(w^T x) - w^T x| \leq nu|w^T x| \tag{1}$$

which follows correctly from the assumption that

$$fl(x_1 - x_2) = (x_1 - x_2)(1 + \epsilon). \tag{2}$$

Indeed, this is the approach employed by Wilkinson(1965), and depends on the above floating point form being valid, and in ϵ being small. Clearly there is a flaw in this assumption if cancellation occurs in the subtraction of $x_1 - x_2$ when ϵ may not now be small in equation (2). Hence the analysis presented is only valid in the stable case when the Wilkinson assumption is valid. If cancellation occurs, a more sensible assumption is

$$fl(x_1 - x_2) = x_1 - x_2 + \epsilon \tag{3}$$

which no longer links ϵ to the size of the resulting difference. With this assumption the inequality in equation (1) becomes

$$|fl(w^T x) - w^T x| \leq nu|w^T||x| \tag{4}$$

where cancellation in the inner product $w^T x$ will no longer invalidate the result.

Hence the analyses presented may under-estimate the error in the cases when

$$|w^T||x| >> |w^T x| \tag{5}$$

and in particular in the elimination method (pp 96,97 and 111), the errors would be of order $n^2 u\|A\|^n$.

In practice, instabilities in the algorithms are carefully monitored, and with modern computer arithmetic, it is possible to carry sufficient numbers of significant digits to absorb some ill-conditioning. A run at a lower precision will then give a measure of the loss of accuracy. In this work the practical loss of accuracy was always at a non-fatal level, and demonstrates that upper bounds on errors, though they give absolute certainty of the result, may prove over restrictive.

CHAPTER 1 - INTRODUCTION

SECTION 1.1: Outline of the Chapters

Chapter 1 begins with an overview of relevant topics in numerical matrix algebra. The discussion does not go into any detail nor does it presume an in-depth knowledge of the subject on the part of the reader, aiming more to present fundamental and derived results necessary to achieve a greater understanding of matrix equations within the context of this thesis. The chapter continues with a general discussion on the importance of estimating the condition of problems and determining the stability of the methods used to solve those problems. Section 1.4 applies these discussions specifically to certain problems relevant to the thesis. Section 1.5 introduces definitions and summarises results that will be used throughout the thesis. These concern the solution of a system of non-linear equations by using a sum of squares minimisation technique. Section 1.6 gives a summary of some of the available theory on quadratic matrix equations.

Chapter 2 uses the discussions and results from Chapter 1 to derive bounds pertaining to the conditioning of the quadratic matrix equations. Central to this analysis as well as to other analyses throughout the thesis is the derivative of $F(X)$. Therefore Chapter 2 begins with a discussion on the existence of this derivative and its inverse.

Chapter 3 discusses a number of methods for computing the characteristic polynomial of a matrix. The discussions include the operations count and storage requirements for implementing the methods on a computer. There is a new look at the stability of Krylov's method and at an interpolation technique for computing the characteristic polynomial of a matrix possessing distinct eigenvalues.

Chapter 4 discusses the Elimination method and shows how it may be used to derive new algorithms for computing the solution of the quadratic matrix equation and the square root problem. These discussions include original work on the stability of the algorithm and the operations count and storage requirements for their implementation.

Chapter 5 summarises the current methods for solving the matrix equations. These discussions include the stability of the methods and a brief comparison of the operations count and algorithmic features of the methods. Section 5.5 shows how globally convergent minimisation methods can be used to solve the quadratic matrix equations by redefining the problem as a system of non-linear equations.

Chapter 6 derives new methods for the solution of the matrix equations. The method for the algebraic Riccati equation is based on the minimisation approach and makes use of the sparse nature of the Jacobian matrix. The method performs matrix factorisations in obtaining an efficient and stable algorithm. The method for the unilateral quadratic matrix equation is also based on the minimisation approach and uses Newton iterations towards providing a stable, efficient and globally convergent algorithm.

Chapter 7 compares the methods discussed in the thesis by showing how they perform on particular problems. The results obtained pertaining to the accuracy of the solutions and the computational efficiency of the algorithms are discussed and related back to the earlier analyses on the conditioning of the problem and the stability of the methods.

Chapter 8 concludes the thesis by reflecting on the objectives of the thesis and how far this work has gone towards meeting these. There are concluding comments on the thesis contents and recommendations for further areas of research.

The Appendices give listing of new computer subroutines relevant to this work and a list of the test matrices used in Chapter 7.

SECTION 1.2: Background Theory in Numerical Matrix Algebra

This section summarises those aspects of matrix theory and numerical matrix algebra relevant to the remainder of the work as well as providing a reference for notations used. For a detailed discussion refer to [Wilkinson], [Stewart, 1], [Golub & Van Loan], [Lancaster & Tismenetsky].

Note that there is no loss of generality in focussing attention on the real space. All definitions and algorithms have obvious analogs in the complex space.

Vectors and Matrices

$\mathbf{R}^n$ denotes the real vector space with n components, such that

$$x \in \mathbf{R}^n \Leftrightarrow x = (x_i) = \{x_1, x_2, \ldots, x_n\}^T$$

where $\Leftrightarrow$ denotes a two-way implication.

Similarly, $\mathbf{R}^{m\times n}$ denotes the space of m-by-n real matrices, such that

$$A \in \mathbf{R}^{m\times n} \Leftrightarrow A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \cdot & \cdot & \ldots & \cdot \\ \cdot & \cdot & \ldots & \cdot \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

where $a_{ij} \in \mathbf{R}$.

Where a capital letter denotes a matrix, the corresponding lower case letter with subscript ij refers to the $(i,j)^{\text{th}}$ component. In the case of a vector it will be clear from the context of the passage whether, for example x, refers to a vector or a scalar.

Some basic matrix manipulations are,

addition	$C = A + B$,	$c_{ij} = a_{ij} + b_{ij}$,	$\mathbf{R}^{m\times n} + \mathbf{R}^{m\times n} \rightarrow \mathbf{R}^{m\times n}$
scalar multiplication	$C = \alpha A$,	$c_{ij} = \alpha a_{ij}$,	$\mathbf{R} \times \mathbf{R}^{m\times n} \rightarrow \mathbf{R}^{m\times n}$
vector multiplication	$y = Ax$,	$y_i = \sum_{k=1}^{n} a_{ik}x_k$,	$\mathbf{R}^{m\times n} \times \mathbf{R}^{n} \rightarrow \mathbf{R}^{m}$
matrix multiplication	$C = AB$,	$c_{ij} = \sum_{i=1}^{n} a_{ik}b_{kj}$,	$\mathbf{R}^{m\times n} \times \mathbf{R}^{n\times p} \rightarrow \mathbf{R}^{m\times p}$
matrix transposition	$C = A^T$,	$c_{ij} = a_{ji}$,	$\mathbf{R}^{m\times n} \rightarrow \mathbf{R}^{n\times m}$

A matrix A is square if $A \in \mathbf{R}^{n\times n}$.

The n-by-n identity matrix has unit entries along its diagonal with zero entries everywhere else and is denoted by I_n, or I where the context is clear.

A matrix in $\mathbf{R}^{m\times n}$ is,

zero	if	$a_{ij} = 0$	for all	i, j
diagonal	if	$a_{ij} = 0$	for all	$\lvert i-j\rvert > 1$
upper triangular	if	$a_{ij} = 0$	for all	$i > j$
upper hessenberg	if	$a_{ij} = 0$	for all	$i > j+1$
strictly upper triangular	if	$a_{ij} = 0$	for all	$i > j-1$

Some important types of square matrices are,

symmetric	if	$A^T = A$
positive definite	if	$x^T Ax > 0,\ x \neq 0 \in \mathbf{R}^n$ then $A > 0$
non-negative definite	if	$x^T Ax \geq 0,\ x \in \mathbf{R}^n$ then $A \geq 0$
orthogonal	if	$A^T A = I$
nilpotent	if	$A^k = 0$ for some k
idempotent	if	$A^2 = A$
diagonally dominant	if	$\lvert a_{ii}\rvert > \sum_{j \neq i} \lvert a_{ij}\rvert$ for all i

A matrix is sparse if it has relatively few non-zero entries. If $A, B \in \mathbf{R}^{n \times n}$ satisfy $AB = I$ then B is the inverse of A and is denoted by A^{-1}. If A^{-1} exists then A is said to be non-singular, otherwise A is singular.

If $A, B \in \mathbf{R}^{n \times n}$, then A commutes with B if $AB = BA$.

The vector e_k defined by $\{0, 0, \ldots, 1, 0, \ldots, 0\}$ is in $\mathbf{R}^n$ unless otherwise indicated and the '1' is in position k.

A is a permutation matrix if $A = \{e_{j_1}, e_{j_2}, \ldots, e_{j_n}\}$ where $\{j_1, j_2, \ldots, j_n\}$ is a permutation of $(1, 2, \ldots, n)$.

Let $A \in \mathbf{R}^{n \times n}$, then the determinant of A, denoted by $\det(A)$, may be defined by

$$\det(A) = \sum_j (-1)^{t(j)}\ a_{1j_1}\ a_{2j_2} \ldots a_{nj_n}$$

where $t(j)$ is the number of inversions in the permutation $j = \{j_1, j_2, \ldots, j_n\}$ and j varies over all n permutations of $1, 2, \ldots, n$.

Alternatively,

$$\det(A) = \sum_{j=1}^{n} (-1)^{j+1} a_{1j} \det(A_{1j})$$

where the minor A_{1j} is an $(n-1)$-by-$(n-1)$ matrix obtained by deleting the first row and j^{th} column of A. $\det(A) \neq 0$ implies A is non-singular and vice-versa. The rank of a matrix $A \in \mathbf{R}^{m \times n}$ is the order of the largest non-singular minor of A and is denoted by $R(A)$. The trace of a square matrix A is the sum of the elements on the diagonal of that matrix and is denoted by $tr(A)$.

If $A \in \mathbf{R}^{m \times n}$ and $k \leq m,\ \ell \leq n$ then any k rows and ℓ columns of A determine a k-by-ℓ submatrix or partition of A.

The elementary row (column) operations are,

1. interchange two rows (columns) of a matrix
2. multiply all elements of a row by some non-zero number
3. multiply any row (column) of a matrix by a non-zero number and add it to any other row (column) of the matrix.

The matrices that effect any of these operations are called elementary matrices.

An inner product of $x, y \in \mathbf{R}^n$ is given by,

$$x^T y = \sum_{i=1}^{n} x_i y_i = y^T x$$

A set of vectors $\{a_1, a_2, \ldots, a_n\}$ is linearly independent if

$$\sum_{i=1}^{n} \alpha_i a_i = 0 \Leftrightarrow \alpha_1 = \alpha_2 = \ldots = \alpha_n = 0$$

for scalars α_j. Otherwise a non-trivial combination of $a_1, a_2, \ldots, a_n$ is zero and the set $\{a_1, a_2, \ldots, a_n\}$ is said to be linearly dependent.

A set of vectors $\{x_1, x_2, \ldots, x_n\}$ with each $x_i \in \mathbf{R}^m$ is orthogonal if

$$x_i^T x_j = 0 \quad \text{for} \quad i \neq j$$

A system of n linear equations in n unknowns given by

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1, 2, \ldots, n$$

may be written as $Ax = b$, $\quad A \in \mathbf{R}^{n \times n}$ and $x, b \in \mathbf{R}^n$.

Let the augmented matrix be $B = (A, b)$, then $Ax = b$ possesses,

1. a unique solution if and only if $R(A) = R(B) = n$
2. an infinite number of solutions if and only if $R(A) = R(B) < n$
3. no solution if and only if $R(A) < R(B)$

The homogeneous system $Ax = 0$ possesses a non-zero solution if and only if $R(A) < n$.

The Eigenvalue Problem and The Characteristic Polynomial

The eigenvalue problem is one of determining those λ for which

$$Ax = \lambda x \qquad \text{where } A \in \mathbf{R}^{n \times n}, x \in \mathbf{R}^n, \lambda \in \mathbf{R}$$

has a non-trivial (or non-zero) solution. Writing this as $(A - \lambda I)x = 0$, it follows from the assertions above that a non-zero solution exists if and only if the matrix $(A - \lambda I)$ is singular, that is $\det(A - \lambda I) = 0$. The determinant may be expanded as follows,

$$a_0 + a_1 \lambda + a_2 \lambda^2 + \ldots + (-1)^n \lambda^n = 0$$

This is the characteristic equation of A and the expression on the left hand side is the characteristic polynomial of A. The n roots of the characteristic polynomial, $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A. Corresponding to any eigenvalue, there exists at least one non-trivial solution x satisfying $Ax = \lambda x$. This is an eigenvector corresponding to that eigenvalue.

The matrix A and its transpose A^T possess the same eigenvalues but different eigenvectors. If A possesses distinct λ_i then the associated x_i are linearly independent. In this case, the n sets of equations $Ax_i = \lambda_i x_i$ may be written as $AX = X$ diag (λ_i). Since the n columns of X are in fact the eigenvectors x_i and they are linearly independent, then X is non-singular and

$$X^{-1}AX = \text{ diag } (\lambda_i)$$

such that there exists a similarity transformation X which reduces A to diagonal form. Two matrices, A and B, are said to be similar if there exists a non-singular matrix P such that $A \doteq P^{-1}BP$.

If A has non-distinct λ_i, the number of occurrences of each distinct eigenvalue is called the algebraic multiplicity of that eigenvalue. In this case the existence of a set of n independent eigenvectors of A is dependent on the elementary divisors of A. If these are linear, then the eigenvectors are linearly independent.

If A has k eigenvalues of multiplicities $m_1, m_2, \ldots, m_k$ such that

$$m_1 + m_2 + \ldots + m_k = n$$

then there exists a similarity transformation H such that $H^{-1}AH$ is in Jordan form,

$$H^{-1}AH = \text{ diag } (J_{p_1}(\lambda_1), \ldots, J_{p_r}(\lambda_1), J_{q_1}(\lambda_2), \ldots, J_{q_s}(\lambda_1), \ldots, J_{u_1}(\lambda_k), \ldots, J_{u_v}(\lambda_k))$$

where

$$p_1 + p_2 + \ldots + p_r = m_1$$

$$q_1 + q_2 + \ldots + q_s = m_2$$

$$\vdots \qquad\qquad \vdots$$

$$u_1 + u_2 + \ldots + u_v = m_k$$

and each $J_r(\lambda)$ is an r-by-r Jordan block,

$$J_r(\lambda) = \begin{bmatrix} \lambda & 1 & 0 & \ldots & \ldots & 0 & 0 \\ 0 & \lambda & 1 & \ldots & \ldots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \ldots & \ldots & \ldots & \lambda & 1 \\ 0 & 0 & \ldots & \ldots & \ldots & 0 & \lambda \end{bmatrix}$$

The number of independent eigenvectors of A is equal to the number of Jordan blocks in the Jordan form. If a matrix has at least one Jordan block of order greater than unity then it has one or more non-linear elementary divisors and fewer than n independent eigenvectors. Such a matrix is called defective. Non-defective matrices may be reduced to diagonal form by unitary (complex orthogonal) transformations.

An eigenvalue is simple if it has only one eigenvector associated with it. A matrix A is normal if it satisfies the commutativity relationship $AA^T = A^TA$. A matrix is called non-derogatory if it has only one Jordan block and therefore only one eigenvector associated with each distinct eigenvalue. Otherwise it is derogatory.

The companion matrix of the characteristic polynomial of A is given by,

$$C = \begin{bmatrix} 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & 0 & 1 & \dots & \dots & 0 \\ \vdots & & & & & \vdots \\ -a_n & -a_{n-1} & -a_{n-2} & \dots & \dots & -a_1 \end{bmatrix}$$

The matrices A and C have the same characteristic polynomial. Also, A is similar to C if A is non-derogatory such that $H^{-1}CH = A$. If A is derogatory, it may be transformed to a direct sum of Frobenius matrices, such that

$$H^{-1}AH = \text{ diag } (F_1, F_2, \dots, F_k)$$

where each F is a non-derogatory matrix which may be transformed into its companion form by a similarity transformation.

A polynomial f, annihilates A if $f(A) = 0$. The unique monic polynomial of least degree which annihilates A is the minimum polynomial of A. Non-derogatory matrices have the same minimum and characteristic polynomials.

The Cayley-Hamilton Theorem states that every matrix $A \in \mathbb{R}^{n \times n}$ satisfies its own characteristic equation, i.e.

$$A^n + a_1A^{n-1} + a_2A^{n-2} + \dots + a_nI = 0$$

Kronecker Products

If $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$, then the Kronecker product of A and B, denoted by $A \otimes B$ is defined by the following partitioned matrix,

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & & & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix}$$

Some useful results concerning Kronecker products now follow,

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

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

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

$$(A \otimes B)^T = A^T \otimes B^T$$

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

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

$$(A \otimes B) = (A \otimes I)(I \otimes B)$$

$$\det(A \otimes B) = (\det(A))^n (\det(B))^n$$

$$\mathrm{tr}(A \otimes B) = \mathrm{tr}(A)\mathrm{tr}(B)$$

$$R(A \otimes B) = R(A)R(B)$$

If $\lambda(A) = \lambda_i$ and $\lambda(B) = \mu_i$ then

$$\lambda(A \otimes B) = \lambda_i \mu_i$$

and

$$\lambda((A \otimes I) + (I \otimes B)) = \lambda_i + \mu_i$$

where A, B, C are matrices of appropriate dimension.

If $\mathrm{vec}(Z) \in \mathbf{R}^{mn}$ is a vector made up of the elements of a matrix $Z \in \mathbf{R}^{m \times n}$ taken a row at a time, then the following results hold for $A \in \mathbf{R}^{m \times n}$, $B \in \mathbf{R}^{p \times q}$ and $X \in \mathbf{R}^{n \times p}$

$$\mathrm{vec}(AX) = (\overset{A \otimes I_p}{\cancel{I_n \otimes A}})\ \mathrm{vec}(X)$$

$$\mathrm{vec}(XB) = (\overset{I_n \otimes B^T}{\cancel{B^T \otimes I_m}})\ \mathrm{vec}(X)$$

therefore, if $m=n, p=q$,

$$\mathrm{vec}(AX + XB) = ((\overset{A \otimes I_p}{\cancel{I_n \otimes A}}) + (\overset{I_n \otimes B^T}{\cancel{B^T \otimes I_m}}))\ \mathrm{vec}(X)$$

One important application of Kronecker products is in the study of linear matrix equations. By using the above notation the Sylvester equation,

$$AX + XB = C$$

may be represented as

$$((\overset{A \otimes I_p}{\cancel{I_n \otimes A}}) + (\overset{I_n \otimes B^T}{\cancel{B^T \otimes I_n}}))\ \mathrm{vec}(X) = \mathrm{vec}(C)$$

Similarly, the Lyapunov matrix equation,

$$A^T X + XA = -C \quad \text{where } C = C^T \geq 0 \text{ , } p=n$$

may be represented as

$$((I_n \otimes A^T) + (A^T \otimes I_n))\ \mathrm{vec}(X) = -\mathrm{vec}(C)$$

(Lancaster & Tismenetsky) discuss the algebraic aspects of linear equations with respect to Kronecker product theory.

Norms

In any analysis of matrix methods it is necessary to be able to assess the 'size' of a vector or matrix. This is done by defining a function called the norm.

The vector norm on $\mathbf{R}^n$ is a function $f : \mathbf{R}^n \to R$ with the following properties,

(1) $f(x) \geq 0$ for every $x \in \mathbf{R}^n$, with equality if and only if $x = 0$

(2) $f(x+y) \leq f(x) + f(y)$ for all $x, y \in \mathbf{R}^n$

(3) $f(\alpha x) = |\alpha| f(x)$ where $\alpha \in \mathbf{R}, x \in \mathbf{R}^n$

Such a function is denoted by $||x||$ with subscripts to distinguish between various norms. The p-norms or the Holder norms are defined by,

$$||x||_p = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{\frac{1}{p}}, \qquad p \geq 1$$

of which

$$||x||_1 = |x_1| + |x_2| + \ldots + |x_n| = \sum_{i=1}^{n} |x_i|$$

$$||x||_2 = (|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}}$$

and

$$||x||_\infty = \max_i |x_i|$$

are the most commonly used.

Unless otherwise indicated, the 2-norm will be used in the remainder of this work since it is invariant under orthogonal transformations. This is since if $U^T U = I$ then

$$||Ux||_2^2 = x^T U^T U x = x^T x = ||x||_2^2$$

Similar definitions exist for matrix norms,

(1) $f(A) \geq 0$ for all $A \in \mathbf{R}^{m \times n}$ with equality if and only if $A = 0$

(2) $f(A+B) \leq f(A) + f(B)$ for all $A, B \in \mathbf{R}^{m \times n}$

(3) $f(\alpha A) = |\alpha| f(A)$ for all $\alpha \in \mathbf{R}, A \in \mathbf{R}^{m \times n}$

(4) $f(AB) \leq f(A)f(B)$ for all $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$

The p-norms are defined as follows,

$$||A||_p = \sup_{x\neq 0} \frac{||A_x||_p}{||x||_p}$$

and in particular,

$$||A||_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$

$$||A||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

$$||A||_2 = \max_i (\sigma_i)^{\frac{1}{2}}$$

where σ_i are the singular values of A, i.e. the eigenvalues of $A^T A$.

An important norm, and one that will be used throughout unless otherwise indicated, is the Euclidean one,

$$||A||_E = \left[\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right]^{\frac{1}{2}}$$

More generally, for any vector norms $||.||_\alpha \in \mathbb{R}^n$ and $||.||_\beta \in \mathbb{R}^m$,

$$||Ax||_\beta \leq ||A||_{\alpha,\beta}||x||_\alpha$$

where $||A||_{\alpha\beta}$ is said to be subordinate to the vector norms $||.||_\alpha$ and $||.||_\beta$, and defined by,

$$||A||_{\alpha,\beta} = \sup_{x\neq 0} \frac{||Ax||_\beta}{||x||_\alpha} = \sup_{||x||_\alpha=1} ||Ax||_\beta$$

Gerschgorin's Theorem

Every root of the complex matrix $A \in \mathbb{R}^{n\times n}$ lies in at least one of the discs with centre a_{ii} and radii

$$r_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|$$

Schur's Theorem

If A is a complex n-by-n matrix and $\lambda_1, \lambda_2, \ldots, \lambda_n$ are the eigenvalues of A, then

$$\sum_{i=1}^{n} |\lambda_i|^2 \leq ||A||_E^2$$

$$\sum_{i=1}^{n} |\mathrm{Re}(\lambda_i)|^2 \leq ||B||_E^2$$

$$\sum_{i=1}^{n} |\mathrm{Im}(\lambda_i)|^2 \leq ||C||_E^2$$

where $B = \frac{1}{2}(A + A^H)$ and $C = \frac{1}{2}(A - A^H)$

Hirsch's Theorem

With the above notation, if $\rho = \max |a_{ij}|$, $\sigma = \max |b_{ij}|$, $\tau = \max |c_{ij}|$, then

$$|\lambda| \leq n\rho, \ |\mathrm{Re}(\lambda)| \leq n\sigma, \ |\mathrm{Im}(\lambda)| \leq n\tau$$

$$|\det(A)| \leq n^{\frac{n}{2}} \rho^n$$

If A and $A^T A$ have eigenvalues λ_i amd μ_i respectively, then

$$\sum_{i=1}^{n} |\lambda_i|^2 \leq \sum_{i=1}^{n} |\mu_i|^2$$

Matrix Algorithms

In the subsequent chapters it is necessary to describe new algorithms for the implementation of some of the methods discussed in this thesis. These algorithms are not given in any formalised language but in one which is precise enough to convey the important algorithmic concepts but informal enough to permit the suppression of cumbersome details.

Operation counts are used to measure the amount of work involved in an algorithm. One count is equivalent to doing a floating point add, a floating point multiplication and a little subscripting. For example the following step is equivalent to one operation count,

$$s := s + a_{ik} b_{kj}$$

Two identities that are used to determine operation counts are,

$$\sum_{p=1}^{q} p = \frac{q}{2}(q+1)$$

$$\sum_{p=1}^{q} p^2 = \frac{q^3}{3} + \frac{q^2}{2} + \frac{q}{6}$$

It is important to note, however, that this means of quantifying work is crude since it ignores looping, code jumping, subroutine calls, subscripting, paging and numerous other activities that go on during program execution. One such important activity has always been the handling of and the requirements for storage. The degree of inefficiency that this represents is dependant on a number of factors including the programming language, efficiency of the code, the storage management system, the computer used and the size of the problem. These days though, mass storage systems are readily available at relatively little cost and the problem of storage is no longer of great significance. However, for completeness, the storage requirements for each algorithm are included.

A much more accurate measure of work involved in the processing of an algorithm is the Central Processor (CPU) time, which will be discussed and used in Chapter 7.

SECTION 1.3: Conditioning and Stability

1.3.1. Introduction

Consider the problem of computing the value of a function $f(x)$. The accuracy attainable by any algorithm is limited since in practice only an approximation to x is known. The inaccuracies in the data x may be a result of two processes. Firstly, x may be determined directly from physical measurements and therefore subject to the errors inherent in all observations. The second source of inaccuracy is in the storage of the data in a digital computing machine which can only handle a finite number of digits. The data may be defined exactly by a mathematical formulae or generated internally. Either way, the exact representation of the data may require a greater number of digits than the machine may be able to hold. This leads to the rounding off of the excess digits giving rise to an element whose representation in the machine is only an approximation to the exact element.

Consequently, if x^* is an approximation to x then an algorithm can at best calculate only $f(x^*)$. If $f(x^*)$ is 'near' $f(x)$ then the problem of determining $f(x)$ is said to be well-conditioned, otherwise it is ill-conditioned. The requirement for 'nearness' in this sense is that $||f(x^*) - f(x)||$ be small with respect to $||f(x)||$ for any appropriate norm. The process that is used to determine the conditioning of a problem is Perturbation Analysis.

Notice that the conditioning of a problem is independent of the method used to compute a solution. A separate analysis for the algorithm is required. Essentially an algorithm may be regarded as a black box that takes a problem and after a number of inexact operations returns what purports to be a solution. The operations are inexact either because of inaccuracies in the given data upon which the calculations are based, as discussed above, or because of inaccuracies introduced in the subsequent analysis of that data.

Hence, if the algorithm yields a computed solution f^* as an approximation to f then if $f^*(x)$ is 'near' $f(x^*)$, the algorithm is said to be stable, otherwise unstable. The process that is used to investigate the stability of a problem is Error Analysis.

When the problem is well-conditioned **and** the algorithm is stable, then the results produced will be accurate. In this case $f(x)$ is 'near' $f^*(x)$. Otherwise there can be no guarantee of accuracy in the computed solution.

In the analysis of an algorithm one must be aware that the algorithm may be required to solve a problem that may be ill-conditioned.

There now follows a discussion of perturbation analysis and error analysis and their respective roles in the investigation of conditioning and stability.

1.3.2. Perturbation Analysis

The purpose of carrying out a perturbation analysis of a problem is to determine the degree to which any perturbations or inaccuracies in the data may affect the solution. Any small inaccuracies giving rise to large perturbations in the solution will imply that

the problem is ill-conditioned. The degree of smallness of the quantities is measured in relation to their magnitudes in the unperturbed state.

As mentioned earlier, the sources of the inaccuracy are either inherent in the data or are due to storage limitations. Although these causes cannot be overcome, the latter can be minimised by using a computing machine with extended precision or using multiple length arithmetic. Even so, minimising the inaccuracies in the original data of a problem that is very ill-conditioned will be of little consequence since even very small inaccuracies may cause massive perturbations in the solution.

Therefore the importance of a perturbation analysis is not in determining an accurate bound for $||f(x^*) - f(x)|| \ / \ ||f(x)||$ but in that it will indicate whether a problem can possibly be solved to within a reasonable accuracy and also where any possible sources of ill-conditioning are likely to occur.

It must be noted that a perturbation analysis of a problem is independent of the method employed to find a solution so that a well-conditioned problem does not automatically imply that a method will produce an accurate solution.

There now follow results from linear analysis useful in the perturbation analysis of a problem:

If $||.||$ denotes any matrix norm for which $||I|| = 1$ and if $||M|| < 1$ then $(I + M)^{-1}$ exists,

$$(I + M)^{-1} = I - M + M^2 - \ \ldots$$

and

$$||(I + M)^{-1}|| \leq \frac{1}{1 - ||M||}$$

[Lancaster & Tismenetsky] give a proof for this result.

If $||M|| < 1$ and $||I|| = 1$, then

$$||I - (I - M)^{-1}|| \leq \frac{||M||}{1 - ||M||}$$

If A is non-singular and $E \in \mathbf{R}^{n \times n}$, then [Stewart],

$$(A + E)^{-1} = A^{-1} - A^{-1}EA^{-1} + 0(||E||^2)$$

Condition Number

It is useful to have some scalar that reflects the change in the solution of a computing problem with respect to small perturbations in the initial data. Such a scalar is called the condition number and is specific to a particular computing problem - Section 1.4 gives the condition numbers for some common problems.

1.3.3. Error Analysis

Error analysis is concerned with the resolution of the round-off errors that arise as a consequence of using floating point arithmetic. Floating point numbers are those numbers, known exactly, which are rounded to r digits according to the storage capability of the computer. The number of digits in a floating point number is known as the precision of the number and on most computers, this is fixed. Many computers also have the ability to manipulate floating point numbers with about twice the usual precision. These are known as double precision numbers and although computations involving them reduce the effect of rounding errors there is an increase in computer time and the storage required for the calculations.

Computers with floating point hardware are provided with a set of instructions for manipulating floating point numbers. These instructions mimic the operations of addition, subtraction, multiplication and division. However, these operations cannot be performed exactly and rounding errors result such that in an extensive calculation there is a real possibility that rounding errors will accumulate and contaminate the solution. It is therefore desirable that any proposed algorithm be analysed to show that rounding error will not affect the results unduly.

There are two main forms of error analysis in common use. These are known as forward and backward analysis and their general principles may be described as follows:

Forward Analysis

In this case, the computation in question is regarded as being described by a number of mathematical equations. In each equation some new quantity x say, is defined in terms of previously computed quantities $a_1, a_2, \ldots, a_n$, say, where some of these may be initial data. The mathematical equation may be written in the form,

$$x = g(a_1, a_2, \ldots, a_n) = g(a_i)$$

The determination of x from the a_i must involve only the fundamental arithmetic operations. Now due to rounding errors made in the calculations, the computed value of x will be different from that obtained if $g(a_i)$ were evaluated exactly. Forward error analysis denotes the computed value by $\bar{x}$ and attempts to obtain a bound for $|\bar{x} - g(a_i)|$. An essential feature of the analysis then is a comparison of $\bar{x}$ with x.

Backward Analysis

This type of analysis is not concerned with the differences between the computed values and the true values at each step. Instead at each stage, it is shown that the computed value obtained by interpreting

$$x = g(a_1, a_2, \ldots, a_n) = g(a_i)$$

is exactly equal to

$$x \equiv g(a_1 + \epsilon_1, a_2 + \epsilon_2, \ldots, a_n + \epsilon_n)$$

for some values of the ϵ_i and to give bounds for these ϵ_i.

It is clear that both types of analysis are informative and preference of one over the other depends on the context in which it is used. In this thesis both types will be used.

We now turn our attention to the way in which rounding errors manifest themselves in floating point computations.

Rounding Errors Due to Floating Point Computation

On a computing machine with a precision of t binary digits, each number x is represented by an ordered pair a and b such that $x = 2^b a$ where b is an integer, positive or negative, and a is a number satisfying

$$\frac{1}{2} \leq |a| \leq 1$$

b is known as the exponent and a as the fractional part. Denote the operands by x_1 and x_2 where

$$x_1 = 2^{b_1} a_1 \quad \text{and} \quad x_2 = 2^{b_2} a_2$$

Firstly consider the operation of addition. Then suppose x_1 is the number with the greatest modulus and compute the integer $b_1 - b_2$. If

(i) $b_1 - b_2 > t$, then x_2 is too small to have any effect as far as the first t significant digits of the sum are concerned so that

$$fl(x_1 + x_2) \equiv x_1$$

where fl denotes floating point.

(ii) $b_1 - b_2 \leq t$, then a_2 is divided by $2^{b_1 - b_2}$ by shifting it $b_1 - b_2$ places to the right. The sum $a_1 + 2^{b_2 - b_1} a_2$ is then calculated exactly and requires less than $2t + 1$ digits for its presentation. This sum is then multiplied by the appropriate power of 2 using a left shift or a right shift so that the resulting number lies in the range permitted for the fractional part (mantissa) of a floating point number, and the exponent (index) b_1 is adjusted to deal with this shift. Finally this $2t$-digit mantissa is rounded to t digits.

In this way if the normalised sum is exactly $2^{b_3} a_3$, then the modulus of the error is bounded by $2^{b_3} . \frac{1}{2} 2^{-t}$ and the modulus of the exact sum lies between $\frac{1}{2} 2^{b_3}$ and 2^{b_3} so that

$$fl(x_1 + x_2) = (x_1 + x_2)(1 + \epsilon)$$

$$\text{where} |\epsilon| \leq 2^{-t}$$

With respect to multiplication, the exponents b_1 and b_2 are added together to give b_3 and the exact $2t$-digit product of a_1 and a_2 is computed, satisfying

$$\frac{1}{4} \leq |a_1 a_2| \leq 1$$

with normalisation, if necessary.

The resulting $2t$-digit product is rounded to give the t-digit mantissa of the computed product,

$$fl(x_1 x_2) \equiv x_1 x_2(1+\epsilon) \qquad |\epsilon| \leq 2^{-t}$$

If either x_1 or x_2 are zero then the computed product is zero.

In division, a denominator of zero is not permissible. Consider x_1/x_2. If x_1 is zero then $fl(x_1/x_2) = 0$, otherwise the exponent b_2 is subtracted from b_1 to give b_3. a_1 is placed in the t most significant digits of the double length accumulator and zero in the t least significant digits. If $|a_1| > |a_2|$ then the number in the accumulator is shifted one place to the right and b_3 is increased by one. The number in the accumulator is then divided by a_2 to give a correctly rounded t-digit quotient,

$$fl\left[\frac{x_1}{x_2}\right] \equiv \frac{x_1}{x_2}(1+\epsilon), \qquad |\epsilon| \leq 2^{-t}$$

The floating point analysis may be extended to give bounds for the rounding errors resulting from computations involving operations on matrices.

Let u denote the unit machine round-off. It may be represented by the largest number for which the computation $1 + u = 1$ is valid. Now if A, B are square matrices of order n and $u = 2^{-t}$ then their computed sum denoted by $\hat{C}$ may be written as

$$\begin{aligned}
\hat{C} &= fl(a_{ij} + b_{ij}) \\
&= (a_{ij} + b_{ij})(1 + e_{ij}), \qquad |e_{ij}| \leq u \\
&= A + B + \Delta C \\
\text{with } ||\Delta C|| &\leq ||(A+B)E||, \qquad E = (e_{ij}) \\
&\leq 2^{-t}||A + B||
\end{aligned}$$

The error bounds associated with multiplication of matrices are a little more difficult to ascertain and are related to the problem of computing the inner product.

Absolute and Relative Errors

The size of the error involved can be measured in two senses. If $y = x_1 \square x_2$, $\square$ is some operation and the computed approximation to y is $\hat{y}$,

$$\hat{y} = fl(x_1 \square x_2) = (x_1 \square x_2)(1+\epsilon), \qquad |\epsilon| \leq 2^{-t}$$

then the Absolute Error in y is,

$$|\hat{y} - y| = (x_1 \square x_2)\epsilon$$

and the Relative Error in y is,

$$\left|\frac{\hat{y} - y}{y}\right| = |\epsilon| \leq 2^{-t}$$

One significant consequence of finite precision arithmetic is the cancellation error induced as a result of taking the difference between two nearly equal numbers.

Consider the following problem on a computer with precision of four decimal places.

$$y = x_2 - x_1, \qquad \text{where } x_1 = 0.12325$$

$$x_2 = 0.12344$$

Since the number of decimal places in x_1 and x_2 is greater than the precision, the computer rounds them so that

$$\hat{y} = fl(x_2 - x_1) = 0.1234 - 0.1233 = 0.0001$$

It is clear that in exact arithmetic, $y = 0.00019$ so that the absolute error is $|\hat{y} - y| = 0.\overset{0}{\hat{0}}009$ and the relative error is $\left|\frac{\hat{y}-y}{y}\right| = 0.4737$.

Depending on the precision of the computer, computing the difference between two nearly equal number may lead to unacceptably large relative errors.

Computing the Inner-product

Let

$$y_n = \sum_{j=1}^{n} w_i x_i = w^T x$$

where w_i, x_i are standard floating point numbers.
Then

$$\hat{y}_n = fl(w_1 x_1 + \ldots + w_n x_n)$$

Define

$$\hat{t}_k = fl(w_k x_k)$$

and

$$\hat{y}_1 = \hat{t}_1, \hat{y}_k = fl(\hat{y}_{k-1} + \hat{t}_k)$$

These yield the following expressions

$$\hat{t}_k = w_k x_k(1 + \epsilon), \qquad |\epsilon| \leq 2^{-t}$$

$$\hat{y}_k = (y_{k-1} + t_k)(1 + \delta), \qquad |\delta| \leq 2^{-t}$$

Therefore

$$\hat{y}_n = w_1 x_1(1+e_1) + \ldots + w_n x_n(1+e_n)$$

$$= \sum_{k=1}^{n} w_k x_k(1+e_k)$$

where

$$1+e_1 = (1+\epsilon)(1+\delta)^{n-1}$$

$$1+e_k = (1+\epsilon)(1+\delta)^{n-k+1}$$

$$k = 2, \ldots, n$$

As a means of leading to this, consider the problem of extended additions and multiplications.

Extended Addition

$$y = \sum_{i=1}^{n} x_i, \qquad x_i \text{ are standard floating point numbers.}$$

In fact $\hat{y} = fl(x_1 + x_2 + \ldots + x_n)$

If $fl(x_i + x_j) = (x_i + x_j)(1+\epsilon)$ for all $i, j, \quad |\epsilon| \leq u$

then

$$\hat{y} = [((x_1 + x_2)(1+\epsilon) + x_3)(1+\epsilon) + \ldots + x_n](1+\epsilon)$$
$$= (x_1 + x_2)(1+\epsilon)^{n-1} + x_3(1+\epsilon)^{n-2} + \ldots + x_n(1+\epsilon)$$

Since $(1+\epsilon)^r = 1 + r\epsilon + 0(\epsilon^2)$ and ignoring terms of $0(\epsilon^2)$,

$$\hat{y} = (x_1 + x_2)(1 + (n-1)\epsilon) + x_3(1 + (n-2)\epsilon) + \ldots + x_n(1+\epsilon)$$

Taking norms

$$|\hat{y} - y| \leq |E|$$

where

$$|E| \leq (n-1)u \sum_{i=1}^{n} |x_i|$$

Extended Multiplication

$$y = \prod_{i=1}^{n} x_i$$

$$\hat{y} = fl(x_1 x_2 \ldots x_n)$$

$$\hat{y} = x_1 x_2(1+\epsilon)x_3(1+\epsilon)\ldots x_n(1+\epsilon) \qquad |\epsilon| \leq u$$

$$= x_1 x_2 \ldots x_n(1+\epsilon)^{n-1}$$

$$\hat{y} = y + y(n-1)\epsilon + 0(\epsilon^2)$$

Taking norms

$$|\hat{y} - y| \leq |y|(n-1)u$$

$$\frac{|\hat{y} - y|}{|y|} \leq (n-1)u$$

Hence

$$(1-2^{-t})^n \leq 1 + e_1 \leq (1+2^{-t})^n$$

$$(1-2^{-t})^{n-k+2} \leq 1 + e_k \leq (1+2^{-t})^{n-k+2}$$

$$k = 2, \ldots, n$$

and it can be shown [Wilkinson] that inequalities of the form

$$(1-2^{-t})^r \leq 1 + e \leq (1+2^{-t})^r$$

may be replaced by the simpler inequality

$$|e| \leq ru \text{ where } u = (1.06)2^{-t}$$

so that

$$|e_1| \leq nu \text{ and } |e_k| \leq (n-k+2)u$$

implies

$$|fl(w^T x) - w^T x| \leq nu|w^T x|.$$

Now consider the product C of two matrices A, B of order n,

$$c_{ij} = fl\left(\sum_{k=1}^{n} a_{ik} b_{kj}\right)$$

$$= \sum_{k=1}^{n} a_{ik} b_{kj}(1 + n\epsilon_{ij}) \qquad |\epsilon_{ij}| \leq u$$

and from the result obtained above it follows that since the determination of an element of C involves the computation of an inner product, then

$$\hat{C} = AB + \Delta C$$

where

$$||\Delta C|| \leq nu||C|| \leq nu||AB||$$

$$\leq nu||A||||B||$$

Similarly if we require to compute

$$y = Ax \qquad \text{where } x \in \mathbf{R}^n, y \in \mathbf{R}^m,\ A \in \mathbf{R}^{m \times n}$$

then

$$y_i = fl\left(\sum_{k=1}^{n} a_{ik}x_k\right), \qquad i = 1, \ldots, m$$

and

$$\hat{y} = Ax + \Delta y \qquad \text{where } ||\Delta y|| \leq nu||A|||x|$$

In the course of such analysis, unless otherwise indicated, the Euclidean norm will be used to estimate the size of a matrix, since

(i) $|||A||| = ||A||$

(ii) it is invariant under orthogonal transformations

(iii) it is easy to compute.

SECTION 1.4 : Numerical Consideration of Relevant Problems

A number of problems are common to many of the algorithms described in this thesis. Consequently it is appropriate to place them at one point of reference. This section states each of the problems and discusses the respective conditioning. An outline of the error analysis of their method of solution is also given.

1.4.1 Solution of Linear Equations

The problem is one of solving $Ax = b$ for non-singular A. Suppose that perturbations exist in the coefficient matrices A and b such that an exactly computed solution $\hat{x}$ satisfies

$$(A + E)(x + \delta x) = b + f$$

Eliminating $Ax = b$ and re-arranging gives

$$A(I + A^{-1}E)\delta x = f - Ex$$

now assume that $||A^{-1}||\,||E|| \leq k < 1$ and $||I|| = 1$, then using a result from the previous section implies that $(I + A^{-1}E)^{-1}$ exists and that $||I + A^{-1}E|| < (1 - k)^{-1}$. Thus

$$\delta x = (I + A^{-1}E)^{-1}A^{-1}f - (I + A^{-1}E)^{-1}A^{-1}Ex$$

and

$$||\delta x|| \leq \frac{||A^{-1}||}{1-k}||f|| + \frac{k}{1-k}||x|| \tag{1.1}$$

since $||b|| \leq ||A||\,||x||$ and hence $\frac{1}{||x||} \leq \frac{||A||}{||b||}$ with $b \neq 0$,

$$\frac{||\delta x||}{||x||} \leq \frac{||A||\,||A^{-1}||}{1-k}\frac{||\delta x||}{||x||} + \frac{k}{1-k}$$

Now since $k = ||A^{-1}||\,||\delta A|| = C(A)\frac{||\delta A||}{||A||}$ where

$$C(A) = ||A||\,||A^{-1}|| \tag{1.2}$$

(1.1) may be written as

$$\frac{||\delta x||}{||x||} \leq \frac{C(A)}{1 - (C(A)||E||(||A||)}\left(\frac{||f||}{||b||} + \frac{||E||}{||A||}\right). \tag{1.3}$$

This gives an upper bound for the relative perturbation of x in terms of the relative perturbations of b and A and the so called condition number, $C(A)$. In particular, the condition number is the dominant feature in this expression.

Note that $C(A)$ may be considered as a magnification constant with respect to this problem and if it is large then the problem is ill-conditioned regardless of the algorithm used to compute x.

From (1.2), the problem of estimating $C(A)$ is one of estimating $||A^{-1}||$ since $||A||$ is easy to determine - [Cline et al] suggest calculating an x satisfying $A^T x = b$ for a specially constructed vector b, then solving $Ay = x$ and using $||y||_1/||x||_1$ as an estimate for $||A^{-1}||_1$.

The algorithm that is used to solve the problem when A is square, dense and unstructured is the method of Gaussian elimination with partial pivoting. A complete treatment of this algorithm is given in [Golub & Van Loan].

A combination of Gauss and elementary transformations M can be found such that

$$M_{n-1}P_{n-1}\dots M_1P_1A = U$$

where U is upper triangular and P are permutation matrices. The original problem is then equivalent to the problem of solving the upper triangular system

$$Ux = (M_{n-1}P_{n-1}\dots M_1P_1)\boldsymbol{x}$$

This is solved by back-substitution.

The following algorithm, in two stages, determines $x \in \mathbb{R}^n$, for $k = 1, \dots, n-1$.

Find $|a_{pk}| = \max|a_{ik}|$ for $i \geq k$ and interchange rows p and k. If $a_{kk} = 0$ then quit else

$t_k = p$

$r_j = a_{kj} \qquad j = k+1, \dots, n$

For $i = k+1, \dots, n$

$s = \dfrac{a_{ik}}{a_{kk}}$

$a_{ik} = s$

For $j = k+1, \dots, n$

$a_{ij} = a_{ij} - sr_j$

$b_i = b_i - sb_k$

For $i = n, \dots, 1$

$x_i = b_i$

For $j = i+1, \dots, n$

$x_i = x_i - a_{ij}x_j$

$x_i = \dfrac{x_i}{a_{ii}}$

The permutations $P_1, \ldots, P_{n-1}$ are represented by the integer vector $(t_1, \ldots, t_{n-1})$ and P_k is obtained by interchanging row k and row t_k of I_n.

From the discussion in the previous section a test of an element being zero is,

$$|a_{kk}| \leq \text{TOL}$$

where TOL reflects the computer's precision and the effects of any errors in a_{kk} resulting from changes in that element.

The operations count for this algorithm is determined by the identities of Section 2. The algorithm requires

$$\frac{n^3}{3} + n^2 + 0(n) \quad \text{operations}$$

where $0(n)$ implies 'order of n'.

If no errors occur during the Gaussian Elimination process other than those in storing A and b, then the computed solution $\hat{x}$ would satisfy

$$(A + E)\hat{x} = b + e$$

where $||E||_\infty \leq u||A||_\infty$ and $||e||_\infty \leq u||b||_\infty$.

However, in practice, Gaussian Elimination does give rise to rounding errors and the computed solution $\hat{x}$ satisfies

$$(A + E)\hat{x} = b$$

where

$$||E||_\infty \leq u8n^3\rho||A||_\infty + 0(u^2) \tag{1.4}$$

The growth factor ρ measures how large the numbers become during the elimination process. In practice ρ is modestly sized (e.g. $\rho = 10$). Therefore Gaussian Elimination (with partial pivoting) is stable. The storage requirement for this algorithm is $n^2 + 2n$ if b is required to be kept or $n^2 + n$ if b is not required since x can then be stored in b.

1.4.2 The Eigenvalue Problem.

The problem is one of determining the n roots of the characteristic polynomial of A,

$$\begin{aligned} f(\lambda) &= \det(A - \lambda I) \\ &= (\lambda - \lambda_1)(\lambda - \lambda_2)\ldots(\lambda - \lambda_n)(-1)^n \end{aligned}$$

where $A \in \mathbf{R}^{n\times n}$, λ_i are the eigenvalues of A.

Firstly consider the case of a non-defective matrix. In this case there exists a non-singular similarity transformation X, that reduces A to a diagonal form,

$$X^{-1}AX = \text{ diag } (\lambda_i) \qquad X \in \mathbf{R}^{n\times n}$$

If A is perturbed then the approximate $\hat{\lambda}_i$ satisfy ,

$$\hat{X}^{-1}(A+E)\hat{X} = \text{ diag } (\hat{\lambda}_i)$$

Then the eigenvalues of $A+E$ satisfy [Stewart],

$$\min_{\lambda \in \lambda(A)} |\hat{\lambda}_i - \lambda_i| \leq ||X^{-1}||_p ||X||_p ||E|| = k(X)||E|| \tag{1.5}$$

$k(X)$ is called the spectral condition number for A with respect to the eigenvalue problem.

For the case of a general matrix A, there exists an orthogonal transformation $Q \in \mathbf{R}^{n \times n}$, such that

$$Q^T A Q = D + N$$

where

$$D = \text{ diag } (\lambda_1, \ldots, \lambda_n)$$

with real λ_i and N is strictly upper triangular. For complex λ_i the transformation is a unitary one.

This is the Schur decomposition of A. If A is perturbed to $A+E$ and μ_i are the eigenvalues of this perturbed matrix, and p is the smallest positive integer such that $N^p = 0$, then [Golub + Van Loan],

$$\min_{\lambda \in \lambda(A)} |\lambda - \mu| \leq \max\{\theta, \theta^{\frac{1}{p}}\}$$

where

$$\theta = ||E||_2 \sum_{k=0}^{p-1} ||N||_2^k$$

In the case of a normal matrix A, there exists an orthogonal transformation such that

$$Q^T A Q = D = \text{ diag } (\lambda_1, \ldots, \lambda_n)$$

The eigenvalues of the perturbed matrix $A+E$ are such that

$$\min_{\lambda \in \lambda(A)} |\lambda - \mu| \leq ||Q^T||\,||Q||\,||E||$$

and since $||Q^T||\,||Q|| = 1$,

$$\min_{\lambda \in \lambda(A)} |\lambda - \mu| \leq ||E|| \tag{1.6}$$

The first two cases indicate that for non-normal matrices, if $k_p(X)$ or $||N||_2^{p-1}$ is large then the eigenvalues of A may be sensitive to small changes in the elements of A i.e. ill-conditioning.

However, for normal matrices, the absolute error in the eigenvalues is of the same order as the perturbations in A, implying well-conditioning.

The conditions above, reflect the sensitivity of the spectrum (set) of eigenvalues rather than the sensitivity of particular eigenvalues. Now suppose that λ is a simple eigenvalue of A and that x and y satisfy

$$Ax = \lambda x \text{ and } y^T A = \lambda y^T$$

with

$$||x||_2 = 1 \text{ and } ||y||_2 = 1.$$

Then if perturbations of order ϵ are made in A, an eigenvalue λ may be perturbed by an amount $\epsilon/s(\lambda)$ where

$$s(\lambda) = |y^T x| \tag{1.7}$$

Therefore if $s(\lambda)$ is small then λ is regarded as being ill-conditioned and $\frac{1}{s(\lambda)}$) is referred to as the condition of the eigenvalue λ.

Note that x and y are normalised right and left eigenvectors of A associated with λ and are unique if λ is simple.

In the perturbation analysis for defective eigenvalues, Ostrowskis' theorem on the continuity of the eigenvalues is very useful:

Let $A, B, \in \mathbf{R}^{n\times n}$ be matrices with elements that satisfy

$$|a_{ij}| < 1 \text{ and } |b_{ij}| < 1$$

If μ is an eigenvalue of $(A + \epsilon B)$ and λ is an eigenvalue of A then,

$$|\mu - \lambda| < (n+2)(n^2\epsilon)^{\frac{1}{n}} \tag{1.8}$$

It may be shown that if λ is a defective eigenvalue of A the$^{n}_{\lambda}$ perturbations of order ϵ in A give rise to perturbations of order $\epsilon^{\frac{1}{n}}$ in λ. A detailed study of the perturbations of eigenvalues is given in [Wilkinson].

The practical QR algorithm is used to solve the eigenvalue problem. The algorithm makes use of the following definition and algorithm.

If $v \neq 0 \in \mathbf{R}^n$, a matrix $P \in \mathbf{R}^{n\times n}$ defined by

$$P = I - 2vv^T/v^T v$$

is called a Householder matrix.

Given $A \in \mathbf{R}^{n\times n}$ the following algorithm overwrites A with $H = U^T AU$ where H is upper Hessenberg and $U = P_1, P_2, \ldots, P_{n-2}$ is a product of Householder matrices,

For $k = 1, \ldots, n-2$,

determine a Householder matrix $P_k \in \mathbf{R}^{(n-k)\times(n-k)}$ such that

$$\bar{P}_k \begin{bmatrix} a_{k+1,k} \\ \vdots \\ a_{nk} \end{bmatrix} = \begin{bmatrix} x \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$A = P_k^T A P_k \quad$ where $P_k = \text{ diag } (I_k, \bar{P}_k)$

Next k.

A detailed algorithm is given in [Smith et al].

As an efficiency note, P_k can be stored in factored form below the subdiagonal of A. The operations count is $\frac{5}{3}n^3$. If U is required then the operations count is $\frac{8}{3}n^3$. An analysis of the round-off errors in this algorithm reveals that the computed Hessenberg matrix $\hat{H}$ satisfies

$$\hat{H} = Q^T(A + E)Q, \qquad Q^T Q = I \tag{1.9}$$

where $||E||_E \leq cn^2 u||A||_E$, c is a small constant. Therefore this algorithm is clearly a stable one. H is said to be unreduced if it has no zero subdiagonal entries.

The development of the QR algorithm is based on the Real Schur Decomposition: If $A \in \mathbf{R}^{n\times n}$, then there exists an orthogonal $Q \in \mathbf{R}^{n\times n}$ such that,

$$Q^T A Q = \begin{bmatrix} R_{11} & R_{12} & \ldots & R_{1m} \\ 0 & R_{21} & \ldots & R_{2m} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & R_{mm} \end{bmatrix}$$

where each $R_{ii} \in \mathbf{R}^{1\times 1}$ or $R_{ii} \in \mathbf{R}^{2\times 2}$, the latter having complex conjugate eigenvalues.

The first step of the algorithm computes unreduced upper Hessenberg matrices,

$$U_0^T A U_0 = H$$

The next step is an iterative one and is based on the double-shift QR technique of Francis (QR step) with shifts determined by the bottom 2×2 matrix. [Golub + Van Loan]. This step also includes a trace on the sub-diagonal elements. Effectively, the upper Hessenberg matrix is reduced to Schur form,

$$U_1^T A U_1 = T$$

that is

$$Q^T A Q = T \qquad \text{where } Q = U_0 U_1$$

The operations count for the whole algorithm in about $8n^3$. If Q is required then the operations count is $15n^3$.

Since the QR algorithm is an orthogonal matrix technique, its round-off properties are favourable. In fact, the computed real Schur form $\hat{T}$ is orthogonally similar to a matrix near to A,

$$Q^T(A+E)Q = \hat{T}, \qquad ||E||_2 \leq u||A||_2 \tag{1.10}$$

and the computed $\hat{Q}$ satisfies,

$$\hat{Q}^T\hat{Q} = I + F \qquad \text{where } ||F||_2 \leq u$$

Eigenvector determination

Now consider the problem of determining the eigenvectors x

$$Ax = \lambda x$$

when A possesses distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. The computed eigenvalues are assumed to be the exact eigenvalues of a matrix near to A. If y is a right eigenvector of A then the computed $\hat{x}_k$ satisfies,

$$||\hat{x}_k - x_k|| \leq u||\sum_{\substack{i=1 \\ i\neq k}}^{n} \left\{ \frac{y_i^T F x_k}{(\lambda_k - \lambda_i) y_i^T x_i} \right\} x_i| + 0(u^2) \tag{1.11}$$

where the computed $\hat{\lambda}$ satisfy

$$(A+E)\hat{x} = \hat{\lambda}\hat{x} \qquad \text{and } ||E|| \leq u||A||$$

Therefore the sensitivity of x_k is dependent on the eigenvalue sensitivity and the separation of λ_k from the other eigenvalues. That is, if λ_k is ill-conditioned or λ_k is near any other eigenvalue then x_k will be ill-conditioned.

If λ is a nondefective, repeated eigenvalue then there are an infinite number of possible eigenvector bases for the associated invariant subspace.

The following iterative algorithm, Inverse Iteration, computes an eigenvector x_i corresponding to an eigenvalue λ_i of A:

(i) Compute the Hessenberg decomposition $U^TAU = H$

(ii) For $i = 1, 2, \ldots, n$

For $k = 1, 2, \ldots$

Solve $(H - \lambda_i I)x_i^{(k)} = x_i^{(k-1)}$

Normalise: $x_i^{(k)} = \frac{x_i^{(k)}}{||x_i^{(k)}||_\infty}$

with $x_i^{(0)}$ being the unit vector $(1, 1, \ldots, 1)^T$.

A suitable stopping criterion is to quit when the residual

$$r^{(k)} = (H - \lambda_i I)x_i^{(k)}$$

is such that

$$||r^{(k)}||_\infty \leq u||H||_\infty||x_i^{(k)}||_\infty$$

(iii) Compute $x_i = Ux_i^{(p)}, \qquad p = \max(k)$

The operations count is $\frac{5}{3}n^3 + rn^2 + n^3 + 0(n^2)$ where r is the sum of iterations for each eigenvector. In practice $r \approx 3n$ so that an overall estimate for the operations count is $8n^3$.

From the error analysis point of view this algorithm uses the Hessenberg decomposition and Gaussian Elimination which are both stable algorithms. Hence for a well-conditioned eigenvalue the process of inverse iteration is stable.

The QR algorithm requires $n^2 + n$ storage locations to compute the eigenvalues. If the eigenvectors are also required then an exact n^2 storage locations are required.

1.4.3 Linear Matrix Equations

In the course of the many algorithms treated in this thesis, it is necessary to determine the numerical solution of the Sylvester equation

$$AX + XB = C, \qquad C, X \in \mathbb{R}^{m\times n},\ A \in \mathbb{R}^{m\times m},\ B \in \mathbb{R}^{n\times n} \tag{1.12}$$

and the Liapunov matrix equation

$$A^TX + XA = -C, \qquad A, X, Q \in \mathbb{R}^{n\times n},\ C = C^T \geq 0 \tag{1.13}$$

These equations also occur in various applications and so are important in their own right. Some applications of (1.12) are in the solution of certain boundary value problems and o.d.e. systems and in the analysis of beam gridworks [Bickley & McNamee], [Dou], [Lasalle & Lefschetz]. Variations of the type (1.13) occur in stability theory, construction of Luenberger observers, design of optimal control systems, [Barnett & Storey], [Luenberger], [Levine & Athans].

[Gantmacher], [Barnett & Storey], [Lancaster & Tismentesky] study the theoretical and algebraic aspects of the solution of these equations. We are more interested in the numerical conditioning of the equations and the stability of their numerical method of solution.

Firstly consider problem (1.12). Suppose that there exist perturbations in A, B and C so that a solution $\hat{X}$, computed by an 'exact' algorithm satisfies

$$(A + E)\hat{X} + \hat{X}(B + F) = C + G$$

where $||E|| \leq u||A||$, $||F|| \leq u||B||$, $||G|| \leq u||C||$. From the earlier discussion on Kronecker products, (1.12) may be represented as a linear equation

$$Px = c \qquad \text{where } P = (I_n \otimes A) + (B^T \otimes I_m)$$

and the linear transformation

$$\phi(X) = AX + XB$$

is non-singular if A and $-B$ have no eigenvalues in common.

Then

$$||\phi^{-1}|| = \left[\min_{X \in \mathbf{R}^{m \times n}} \frac{||\phi(X)||}{||X||}\right]^{-1} = ||P^{-1}|| \tag{1.14}$$

and the solution $\hat{X}$ satisfies, [Golub, Nash & Van Loan]

$$\frac{||X - \hat{X}||}{||X||} \leq 4u\{||A|| + ||B||\}||\phi^{-1}|| \tag{1.15}$$

This inequality implies that if P^{-1} is well-conditioned then the problem is well-conditioned.

Currently the best numerical technique for the solution of (1.12) [Golub, Nash & Van Loan] uses the Hessenberg and Schur reductions, discussed earlier, in the following way.

(i) Reduce A to upper Hessenberg form by using Householder's method,

$$U^T A U = H \qquad U^T U = I$$

(ii) Reduce B to lower Schur form by using the QR method,

$$V^T B V = S \qquad V^T V = I$$

(iii) Update the right-hand side

$$U^T C V = T$$

so that the original problem is equivalent to solving

(iv) $HY + YS = T$

by back substitution, where

(v) $X = UYV^T$

If $p = \max(m, n)$ then the operations count for this algorithm is $20p^3$ and the storage locations required is $6p^2$.

Applying the error analysis of the previous section pertaining to orthogonal matrices and that of Section 1.3 shows that the computed $\hat{X}$ satisfies,

$$\frac{||X - \hat{X}||}{||X||} \leq cu||\phi^{-1}||(||A|| + ||B||) + 0(u^2) \tag{1.16}$$

where c is a small constant.

This bound is essentially the same as that in (1.15) obtained for an exact algorithm. Therefore, this 'Hessenberg-Schur' algorithm is a stable technique for solving (1.12).

Now consider the problem (1.13). If A and C are perturbed slightly, $\hat{X}$ satisfies

$$(A+E)\hat{X} + \hat{X}(A^T + F) = C + G$$

and the perturbation in X is bounded by

$$\frac{||X - \hat{X}||}{||X||} \leq 8u||A||\,||\phi^{-1}|| \tag{1.17}$$

where $||\phi^{-1}||$ is defined by (1.14).

The following steps effect the numerical solution of (1.13), [Bartels & Stewart],

(i) Reduce A to lower Schur form

$$U^T A U = S$$

(ii) Update the right-hand side

$$U^T C U = T$$

so that the original problem is equivalent to solving

(iii) $S^T Y + Y S = T$

by a back substitution algorithm, where

(iv) $X = UYU^T$

The operations count for the algorithm is $19n^3$ and the storage locations required is $5n^2$.

The bound on the error in the solution is similar to (1.16);

$$\frac{||X - \hat{X}||}{||X||} \leq cu||\phi^{-1}||\,||A|| + 0(u^2) \tag{1.18}$$

where c is a small constant.

This implies that the Bartels-Stewart algorithm is a stable technique for solving (1.13).

SECTION 1.5: Systems of Non-linear Equations

The problem is that given a continuous function $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ made up of n component functions $f_i(x), \mathbb{R}^n \rightarrow \mathbb{R}$, non-linear in the n unknowns x_i, it is required to find some $x^* \in \mathbb{R}^n$ such that each component function $f_i(x)$ vanishes at x^* for $i = 1, \ldots, n$, that is $F(x^*) = 0$.

This problem may be solved by applying Newton's method for systems of non-linear equations, iteratively from a starting guess $x_0 \in \mathbb{R}^n$, such that at each iteration k, we solve

$$J(x^{(k)})p^{(k)} = -F(x^{(k)}) \tag{1.19}$$

and update with

$$x^{(k+1)} = x^{(k)} + p^{(k)} \tag{1.20}$$

where the Jacobian, $J(x^{(k)})$ is the matrix of first partial derivatives of F at $x^{(k)}$,

$$J(x^{(k)}) = \frac{\partial f_i}{\partial x_j^{(k)}}, \qquad i, j = 1, \ldots, n \tag{1.21}$$

The iterates (1.20) are dependent on a 'good' initial guess $x^{(0)}$ and on the non-singularity of J at x^*. It is known [Dennis & Schnabel] that the iterates have a quadratic convergence property from a good starting point but the convergence is not always global. To remedy this, a global strategy must be used.

One such strategy is to transform this into a sum of squares problem and minimise the resultant function; that is, minimise the scalar function

$$fc = \frac{1}{2} \sum_{i=1}^{n} fi^2 \tag{1.22}$$

Newton's method for unconstrained minimisation of an arbitrary function f from a starting guess $x^{(0)} \in \mathbb{R}^n$, solves

$$\nabla^2 f(x^{(k)})p^{(k)} = -\nabla f(x^{(k)}) \tag{1.23}$$

and updates

$$x^{(k+1)} = x^{(k)} + p^{(k)}$$

where the gradient $\nabla f(x^{(k)})$ is the vector of first partial derivatives of f at $x^{(k)}$,

$$\nabla f(x^{(k)}) = \frac{\partial f}{\partial x_i^{(k)}} \qquad i = 1, \ldots, n \tag{1.24}$$

and the Hessian, $\nabla^2 f(x^{(k)})$ is the matrix of second partial derivatives of f at $x^{(k)}$,

$$\nabla^2 f(x^{(k)}) = \frac{\partial^2 f}{\partial x_i^{(k)} \partial x_j^{(k)}} \qquad i, j = 1, \ldots, n \tag{1.25}$$

With respect to the function fc in (1.22), the expressions in (1.24) and (1.25) may be defined in terms of f and J, as follows,

$$[\nabla f(x)]_j = \frac{\partial fc}{\partial x_j}, \qquad j = 1, \ldots, n$$

$$= \sum_{i=1}^{n} \frac{1}{2} \frac{\partial f_i^2}{\partial x_j}$$

$$= \sum_{i=1}^{n} fi \frac{\partial f_i}{\partial x_j}$$

from (1.21)

$$= \sum_{i=1}^{n} fi J_{ij}$$

$$\therefore \quad \nabla f(x) = J^T F \tag{1.26}$$

Also

$$[\nabla^2 f(x)]_{ij} = \sum_{k=1}^{n} \frac{\partial}{\partial x_i}(f_k J_{kj})$$

$$= \sum_{k=1}^{n} \frac{\partial f_k}{\partial x_i} J_{kj} + \sum_{k=1}^{n} f_k \frac{\partial J_{kj}}{\partial x_i}$$

$$= \sum_{k=1}^{n} J_{ki} J_{kj} + \sum_{k=1}^{n} f_k \frac{\partial}{\partial x_i} \left(\frac{\partial f_k}{\partial x_j} \right)$$

$$= [J^T J]_{ij} + \sum_{k=1}^{n} f_k \frac{\partial^2 f_k}{\partial x_i \partial x_j} \tag{1.27}$$

We can now state that starting from an initial $x^{(0)}$, the iterates

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)} \tag{1.28}$$

where the direction of search $p^{(k)}$ solves

$$A^{(k)} p^{(k)} = -J^T(x^{(k)}) F(x^{(k)}), \tag{1.29}$$

converge to a x^* that minimises (1.22).

The scalar $\alpha^{(k)}$ is chosen so as to approximately minimise $fc(x^{(k+1)})$ with respect to $\alpha^{(k)}$ and $A^{(k)}$ is a matrix characteristic of the particular Newton method.

One such $A^{(k)}$ is $\nabla^2 f(x^{(k)})$, the Hessian of fc at $x^{(k)}$. This is typically a modified Newton method and it is well-known [Ortega & Rheinboldt] that,

(i) it is locally convergent irrespective of where the starting point is

(ii) it has quadratic local convergence,

$$\lim_{k\to\infty} \frac{||x^{(k+1)} - x^*||}{||x^{(k)} - x^*||^2} \text{ is finite}$$

A necessary condition is that every $\nabla^2 f(x^{(k)})$ must be non-singular. It has been argued that the determination of the Jacobian and the Hessian at each iteration is undesirable, particularly for large problems and since $fc(x^*)$ is sufficiently small then neglecting the second term in (1.29) would not significantly affect the local convergence properties of this method.

This argument gives rise to the Gauss-Newton method which sets $A = J^T J$. This has quadratic local convergence if

$$\lambda_i(J(x^*)^T J(x^*)) < k\, fc(x^*)^{\frac{1}{2}} \quad \text{for } k > 0$$

In this case (1.29) may be written as

$$J(x^{(k)})p^{(k)} = -F(x^{(k)}) \tag{1.30}$$

The iterations may give rise to a near-singular Jacobian implying ill-conditioning with respect to the problem (1.30). However, this may be overcome by adding a scalar matrix $H^{(k)}I$ to $A^{(k)}$, the problem of choosing $H^{(k)}$ not being a difficult one [Brown & Dennis].

It is perhaps best to use a combination of these methods, taking advantage of their respective 'nice' properties, as follows.

From a starting point $x^{(0)}$, generate a sequence of iterates (1.28) intended to converge to a local minimum of (1.22) where the direction of search depends upon the reduction in the sum of squares obtained during the last iteration. If the sum of squares was sufficiently reduced then the Gauss-Newton direction is used, otherwise the Newton direction.

This is designed to ensure that steady progress is made, whatever the starting point and to have the rapid ultimate convergence of Newtons method.

The steps in the algorithm are as follows,

(i) Select $x^{(0)} \in \mathbb{R}^n$, the initial estimates to x^*

(ii) Determine whether the Gauss-Newton or the Newton iteration is to be used

Gauss-Newton

(iii) Determine the jacobian $J(x^{(k)})$

(iv) If $J(x^{(k)})$ is singular add a scalar matrix $\mu^{(k)}I$ to $J^T J$ and solve

$$(J^T J + \mu^{(k)} I)p^{(k)} = -J^T F \quad \text{for } p^{(k)} \tag{1.31}$$

otherwise solve

$$Jp^{(k)} = -F \quad \text{for } p^{(k)} \tag{1.32}$$

Newton

(iii) Determine the Jacobian $J(x^{(k)})$, the gradient $\nabla f(x^{(k)})$ and the Hessian $\nabla^2 f(x^{(k)})$

(iv) If $\nabla^2 f(x^{(k)})$ is not positive definite add a scalar matrix $\mu^{(k)} I$ such that $(\nabla^2 f(x^{(k)}) + \mu^{(k)} I)$ is positive definite and solve

$$(\nabla^2 f(x^{(k)}) + \mu^{(k)} I)p^{(k)} = -J^T F \quad \text{for } p^{(k)} \tag{1.33}$$

otherwise solve

$$\nabla^2 f(x^{(k)})p^{(k)} = -J^T F \quad \text{for } p^{(k)} \tag{1.34}$$

(v) Minimise $fc(x^{(k)} + \alpha^{(k)} p^{(k)})$ with respect to $\alpha^{(k)} \in \mathbb{R}$

(vi) Update, $x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$

(vii) If convergence criterion is not met, go to (ii)).

At step (iv), Gaussian Elimination may be used to solve (1.32). For (1.31), (1.33) and (1.34) a more efficient technique is available. Since $J^T J$ and $\nabla^2 f(x)$ are symmetric and positive definite, the matrices in (1.31), (1.33) and (1.34) are also symmetric and positive definite. As such, they may be reduced to LL^T where L is a lower triangular matrix with positive entries on the diagonal. This is known as the Cholesky decomposition and gives rise to an algorithm for solving the linear system which uses half the number of operations used by the Gaussian Elimination algorithm [Wilkinson]. It is known that this algorithm is stable and gives rise to accurate solutions when the problem is well-conditioned.

Step (v) involves a line search; i.e. given an initial $a^{(r)}$ and a direction $p^{(r)}$, minimise the function $fc(x^{(r)} + \alpha^{(r)} p^{(r)})$ with respect to $\alpha^{(r)}$. Powell developed a quadratic interpolation method [Powell] specifically for this problem which may be summarised:

(i) Choose a step length $h|p^{(r)}|$

(ii) Evaluate $fc(x^{(r)}), fc(x^{(r)} + hp^{(r)})$

(iii) If

$$fc(x^{(r)}) < fc(x^{(r)} + hp^{(r)}), \text{ evaluate } fc(x^{(r)} - hp^{(r)})$$

otherwise evaluate $fc(x^{(r)} + 2hp^{(r)})$

Values are now known at 3 points on the line $x^{(r)} + \lambda p^{(r)}$.

(iv) Determine the turning point $\alpha^{(r)} = \alpha_m^{(r)}$ of the quadratic function $y(\alpha^{(r)})$ fitted through these three points - denoted as a, b, e - using the following formula,

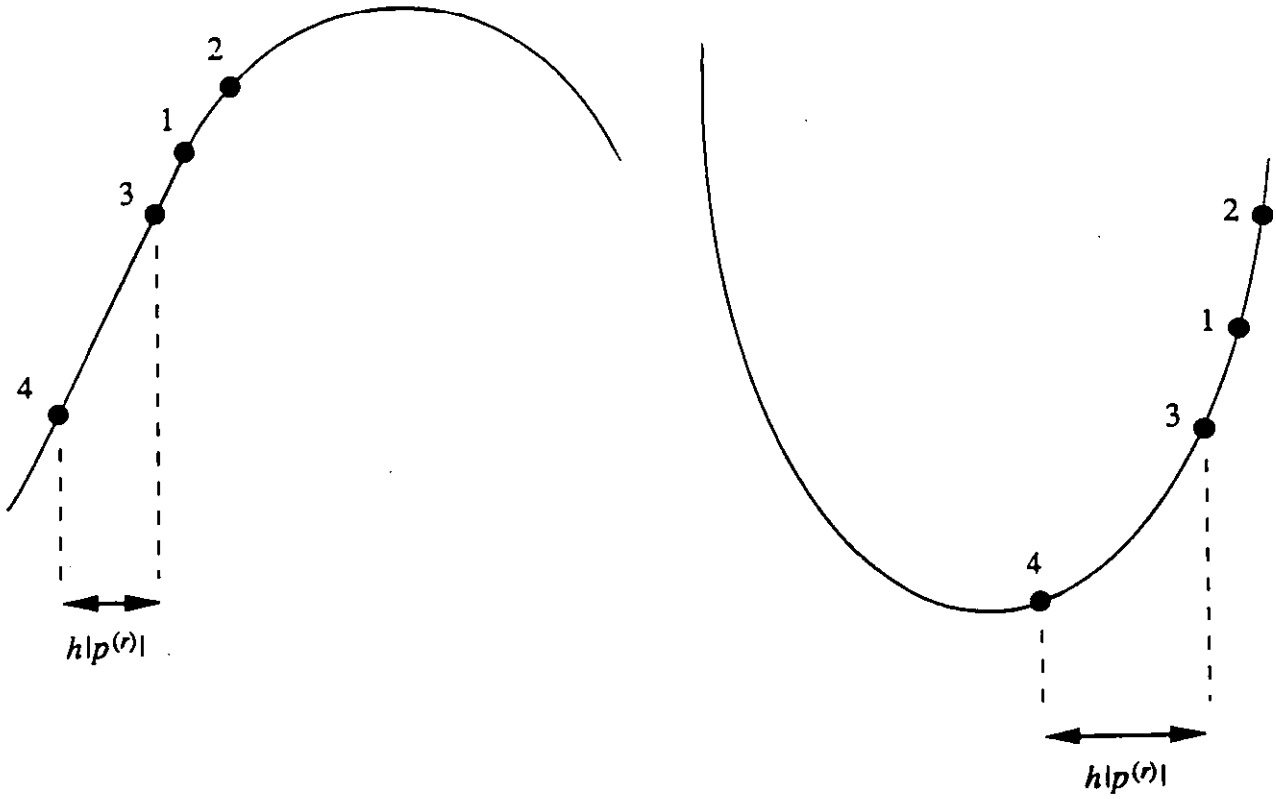
$$\alpha_m^{(r)} = \frac{1}{2}\left(\frac{(b^2 - e^2)fc_a + (e^2 - a^2)fc_b + (a^2 - b^2)fc_e}{(b - c)fc_a + (c - a)fc_b + (a - b)fc_e}\right)$$

and test for a minimum by using,

$$\frac{(b - e)fc_a + (e - a)fc_b + (a - b)fc_e}{(a - b)(b - e)(e - a)} < 0$$

(v) If:

(a) the point $\lambda = \lambda_m$ corresponds to a maximum of $y(\alpha^{(r)})$ or if it corresponds to a minimum which is at a greater distance then $k|p^{(r)}|$ (where h is prescribed) from the nearest of the three points, proceed as follows. Discard the point which is furthest from the turning point and obtain a new current direction in which the function decreases; this step is taken from the point furthest from (nearest to) the turning point when the turning point corresponds to a maximum (minimum).



or (b) If the point $\alpha^{(r)} = \alpha_m^{(r)}$ corresponds to a minimum of $y(\alpha^{(r)})$ and if it is within a small prescribed distance $\epsilon|p^{(r)}|$ of the nearest point $\alpha^{(r)} = \beta$ say, of the three current points, then take

$$\min\{fc(x^{(r)} + \alpha_m^{(r)} p), \quad fc(x^{(r)} + \beta p)\}$$

as the required minimum value of $fc(x^{(r)})$

or (c) If the point $\alpha^{(r)} = \alpha_m^{(r)}$ corresponds to a minimum of $y(\alpha^{(r)})$ to which neither (a) or (b) applies i.e. if it is not further than $h|p^{(r)}|$ from the nearest of the three current points but not within $\epsilon|p^{(r)}|$ of it, discard the point with the highest function value and replace it by $\alpha^{(r)} = \alpha_m^{(r)}$.

Goto (iv)

Using exact arithmetic, a necessary condition for convergence is $\nabla f(x) = 0$. However, in a finite precision environment that uses floating point arithmetic, this condition must be revised to test that

$$||\nabla f(x^{(k)}|| \leq TOL$$

holds, where TOL is some small tolerance.

This global strategy has one significant but unavoidable short-coming. This may arise when the function fc has a local minimiser which is not a root of $F(x)$. The algorithm may then converge to this point if the iterations are started at an $x^{(0)}$ near to this point. In this case all one can do is to restart the algorithm from a different $x^{(0)}$.

A discussion on the operations count for this algorithm, at this stage, is largely not informative, since the formation of F, J and $\nabla^2 f(x)$ contribute significantly to the count. It is problem specific and will be discussed whenever this algorithm is used for a particular problem. It suffices to say that the line-search algorithm, the test for convergence and the determination of μ are processes of $0(n^2)$ and that the solution of the linear system, by a combination of Cholesky method and Gaussian Elimination, takes less than $\frac{7}{6}n^3$ operations.

SECTION 1.6. Some Theory on Quadratic Matrix Equations.

The real monic unilateral quadratic matrix equation

$$X^2 + PX + Q = 0 \tag{1.35}$$

has coefficient matrices P and Q both real, square and of order n. The matrices that solve (1.35) are referred to by X and are also real, square and of order n. The equation (1.35) is a general one in the sense that for arbitrary P and Q we have very little information pertaining to the matrices P, Q and X. Nothing is known about their definiteness or their symmetry, we do not know how many solutions exist generally, how their localization manifests itself, if at all and if a solution is found, it is not known how to eliminate it and generate further solutions.

As a consequence, practical methods for solving (1.35) have been iterative ones requiring an initial approximation to the solution matrix, usually by guesswork. One such method, known as the 'worst possible algorithm' [Ingraham] and discussed at length in Chapter 5, expresses the matrix equation as a set of n^2 quadratic scalar equations which are then solved by some appropriate technique. The disadvantage of this method, apart from the considerable amount of work involved in solving a set of n^2 non-linear equations, is that the matrical properties of the elements in the equation are lost. Nevertheless, there are instances where no other available method can determine a solution for (1.35) in which case the 'worst possible algorithm' would be the best and only option. A number of exact, non-iterative method have been proposed and these are discussed in [McDonald]. However, these are workable only for low order problems since the degree of algebraic complexity involved when dealing with high order matrices does not lend itself to the development of practical algorithms.

However, a practical non-iterative approach to determine certain solution matrices may be developed by considering the associated lambda-matrix problem defined by

$$\lambda^m I + A_1\lambda^{m-1} + A_2\lambda^{m-2} + \ldots + A_m = 0 \tag{1.36}$$

which frequently occurs in the study of differential equations [Lancaster], [Gohberg, Lancaster & Rodman]. A special case is the quadratic one such that

$$\lambda^2 I + A_1\lambda + A_2 = 0 \tag{1.37}$$

Put $A_1 = P$ and $A_2 = Q$ and (1.37) may be written as the quadratic eigenvalue problem

$$(\lambda^2 I + P\lambda + Q)x = 0$$

or

$$F(\lambda I)x = 0 \tag{1.38}$$

with

$$X^2 + PX + Q = F(X)$$

where it is required to find all scalars λ and non-zero vectors x that satisfy (1.38). λ and x are referred to as the latent roots and latent vectors respectively. Now (1.38) possesses a non-zero x provided

$$|\lambda^2 I + P\lambda + Q| = 0 \tag{1.39}$$

The problem of determining the solutions of (1.35) is very closely related to the problem of determining the roots of the polynomial in (1.39) because every characteristic root of a solution of (1.35) is a root of (1.39) [MacDuffee],

$$\begin{aligned} 0 &= |\lambda^2 I + P\lambda + Q| \\ &= |\lambda^2 I + P\lambda - X^2 - PX| \\ &= |\lambda I + P + X|\,|\lambda I - X| \end{aligned}$$

The latent roots of $F(\lambda)$ are the union of the characteristic roots of X and those of $(-P - X)$.

As a way to determine the roots of $F(\lambda)$, consider the Block Companion matrix associated with (1.36)

$$A = \begin{pmatrix} 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & & \vdots \\ \vdots & & & & \vdots \\ \vdots & & & & I \\ -A_m & -A_{m-1} & -A_{m-2} & & -A_1 \end{pmatrix}$$

The equivalent matrix for the quadratic case is

$$A = \begin{pmatrix} 0 & I \\ -Q & -P \end{pmatrix} \tag{1.40}$$

which will be used frequently when considering the Elimination Method.

The eigenvalues of this $2n \times 2n$ real matrix are the latents roots of $F(\lambda)$ since

$$\begin{aligned} |A - \lambda I| &= \begin{vmatrix} -\lambda I & I \\ -Q & -P - \lambda I \end{vmatrix} \\ &= |\lambda^2 I + P\lambda + Q| \end{aligned}$$

The relationship between the eigenvectors of A and the latents vectors of $F(\lambda)$ is not so obvious. Let μ_i be a latent root of $F(\lambda)$ and y_i the corresponding latent vector, then

$$(\mu_i^2 I + P\mu_i + Q)y_i = 0 \tag{1.41}$$

Now μ_i is also an eigenvalue of A and if z_i is the corresponding eigenvector, then we have

$$Az_i = \mu_i z_i \tag{1.42}$$

Since z_i is a vector of length $2n$, it may be partitioned into 2 n-vectors.

$$z_i = \{z_i^{(1)}, z_i^{(2)}\}^T \tag{1.43}$$

using (1.40), (1.42) and (1.43)

$$z_i^{(2)} = \mu_i z_i^{(1)} \tag{1.44}$$

$$-Qz_i^{(1)} - Pz_i^{(2)} = \mu_i z_i^{(2)} \tag{1.45}$$

substitute (1.44) into (1.45)

$$\mu_i^2 z_i^{(1)} + P\mu_i z_i^{(1)} + Qz_i^{(1)} = 0$$

$$(\mu_i^2 I + P\mu_i + Q)z_i^{(1)} = 0 \tag{1.46}$$

comparing (1.46) with (1.41)

$$z_i^{(1)} = y_i$$

and from (1.44)

$$z_i^{(2)} = \mu_i y_i \tag{1.47}$$

such that

$$z_i = \{y_i, \mu_i y_i\}^T \tag{1.48}$$

If the solution of (1.35) has linearly independent eigenvalues then it may be diagonalised by elementary operations U say such that

$$U^{-1}XU = \Lambda = \text{ diag } (\lambda_1, \ldots, \lambda_n)$$

such that

$$X = U\Lambda U^{-1} \tag{1.49}$$

Substituting into $F(X) = 0$ gives

$$(U\Lambda^2 U^{-1} + PU\Lambda U^{-1} + Q) = 0$$

$$(U\Lambda^2 + PU\Lambda + QU)U^{-1} = 0 \tag{1.50}$$

Comparing (1.50) with (1.41), it is clear that the columns of U are the latent vectors corresponding to the latent roots of $F(\lambda)$. From (1.48)

$$U = [z_1^{(1)}, z_2^{(1)}, \ldots, z_n^{(1)}] \tag{1.51}$$

Therefore, for diagonalisable X a solution to (1.35) is given by (1.49) where U is defined by (1.51), the z_i being determined by solving the eigenvalue problem (1.38).

For non-diagonalisable matrices X a transformation of the type in (1.49) still exists but in this case the diagonal matrix Λ is replaced by J, the Jordan canonical form. If X has r distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_r$ of multiplicities $m_1, m_2, \ldots, m_r$ such that $\Sigma m_i = n$ $(i = 1, \ldots, r)$ then J consists of simple Jordan submatrices along the diagonal with all other elements equal to zero. A simple Jordan submatrix of order k, associated with λ_i, is defined as

$$\begin{pmatrix} \lambda_i & 1 & & & \\ & \lambda_i & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda_i & 1 \\ & & & & \lambda_i \end{pmatrix}$$

Now, to determine a solution of (1.35) it is necessary to determine U from

$$UJ^2 + PUJ + QU = 0 \tag{1.52}$$

where J has n of the latent roots of $F(\lambda)$ as its eigenvalues. From the numerical point of view (1.52) is difficult to solve without multiplying out the matrices to obtain a system of n^2 linear equations in the elements of U. However, this results in loss of information due to the breaking of the structure.

Before giving an example to solve (1.35) for diagonalisable X, it will be helpful to give the following definition.

Definition

A set of 2 solutions of $F(X) = 0$ is a complete set of solutions if the set of $2n$ eigenvalues of the 2 solutions is the same, counting multiplicities, as the set of $2n$ latent roots $F(\lambda)$ [Dennis, Traub & Weber, 1].

Example

Consider solving

$$F(X) = X^2 + \begin{pmatrix} -1 & -6 \\ 2 & -9 \end{pmatrix} X + \begin{pmatrix} 0 & 12 \\ -2 & 14 \end{pmatrix}$$

The block companion matrix A is

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -12 & 1 & 6 \\ 2 & -14 & -2 & 9 \end{pmatrix}$$

The eigenvalues of A are 1, 2, 3, 4, which are also the latent roots of $F(\lambda)$ where

$$F(\lambda) = \begin{pmatrix} \lambda^2 - \lambda & -6\lambda + 12 \\ 2\lambda - 2 & \lambda^2 - 9\lambda + 14 \end{pmatrix}$$

The eigenvalues of $\overset{A}{\mathscr{C}}$ are $\{1, 2, 3, 4\}$ and the corresponding eigenvectors are,

$$(1,0,1,0)^T,\ (0,1,0,2)^T,\ (1,1,3,3)^T,\ (1,1,4,4)^T$$

Therefore the latent vectors of $F(\lambda)$ are,

$$(1,0)^T,\ (0,1)^T,\ (1,1)^T,(1,1)^T$$

The problem has solutions with eigenvalue pairings $(1,2)$, $(1,3)$, $(1,4)$, $(2,3)$, $(2,4)$. The pair $(3,4)$ cannot be the eigenvalues of another solution since the corresponding eigenvectors are linearly dependent. Also, this problem has 2 sets of complete solutions.

$$\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 4 & 0 \\ 2 & 2 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & -1 \\ 0 & 4 \end{pmatrix}, \begin{pmatrix} 3 & 0 \\ 1 & 2 \end{pmatrix}$$

This example illustrates the shortcoming of this method. That is, it is not known, prior to the determination of the latent vectors, which combination of the latent values of $F(\lambda I)$ are also the eigenvalues of X. This uncertainty factor is the reason why this technique is not currently used in practice.

Matrix Square Root

Putting $P = 0$ in (1.35) results in the following special case,

$$X^2 - A = 0 \tag{1.53}$$

This is the matrix square root problem, so-called since the solution X of (1.53) is the square root of A. This problem has merited considerable research in its own right since it arises in a number of applications, e.g. least squares estimation, the study of differential equations and estimation of navigation systems.

It is not always easy to establish the existence of a square root of a matrix. Consider the following matrices,

$$A_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \; A_2 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \; A_3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

A_1 has no square root, A_2 does possess a square root and A_3 has infinitely many square roots. It may be shown that if A has at least $n-1$ non-zero eigenvalues then a square root always exists, otherwise the existence of one depends upon the structure of the elementary divisors of A corresponding to the zero eigenvalues [Gantmacher], [Cross & Lancaster].

Since the eigenvalues of X are the square roots of the eigenvalues of A, the total number of square roots of a non-singular matrix with distinct eigenvalues is 2^n.

If A is derogatory, that is there is more than one Jordan block associated with an eigenvalue, then it possesses an infinite number of square roots [McDonald].

Notice that if X_1 is a square root of A then so is $X_2 = -X_1$ since $X_2^2 = (-X_1)^2 = X_1^2 = A$.

From the above discussion it is clear that unlike in (1.35), a number of relationships exist between that matrices A and X in (1.53). Consequently there has been a greater emphasis on determining efficient and stable solutions to the matrix square root problem.

Algebraic Riccati Equation

We now move onto another type of quadratic matrix equation, the algebraic Riccati equation (ARE). The ARE plays a fundamental role in the analysis, synthesis and design of estimation systems as well as in many other branches of applied mathematics; for example, in the determination of steady-state solutions of the matrix Riccati differential equation [Reid], in the theory of multiwire transmission lines [Sternberg & Kaufman] and in optimum automatic control theory [Kalman].

The ARE may be derived by considering the Linear-Quadratic-Gaussian control problem:

Let A, H be $n \times n$ matrices with H symmetric and positive semi-definite,

B be a $n \times m$ matrix,

and R be a $m \times m$ symmetric, positive definite matrix.

Define a quadratic cost functional by

$$J[x,u] = \frac{1}{2}\int_0^\infty (x^T H x + u^T R u)\, dt \tag{1.54}$$

where x, the state vector and u the ~~control~~ input vector are of length n and linked by the linear relation

$$\dot{x} = Ax + Bu \tag{1.55}$$

The L.Q.G. control problem is then:
Minimise $J[x,u]$ for x,u satisfying (1.55).

It is known [Wonham] that if the pair $[A,B]$ are stabilizable and the pair $[A,K]$ are detectable then the optimal control u^* is given by

$$u^* = -R^{-1}B^TKx$$

the closed-loop system matrix being $A - BR^{-1}B^TK$ where K is the non-negative definite solution of the equation

$$A^TK + KA - KBR^{-1}B^TK + H = 0 \tag{1.56}$$

This is the Algebraic Riccati Equation. Writing $G = BR^{-1}B^T$, it may be written as

$$A^TK + KA - KGK + H = 0 \tag{1.57}$$

There has been a great deal of research carried out in this area and this has led to a number of successful methods for solving (1.57). Much of this has been motivated by the fact that G and H are symmetric and at least positive semi-definite and that in most cases the solution of interest is the symmetric non-negative definite one. A sufficient condition for the existence of such a solution is that $[A,C]$ is stabilizable, i.e. there exists a matrix D such that $\mathrm{Re}(\lambda_i(A-CD)) < 0$ where $CC^T = G$ [Wonham].

It is interesting to note that K may be expressed in terms of the solution to (1.35) by observing the following transformation [Barnett].

Since G is positive definite and therefore non-singular, G^{-1} exists. Put $K = YG^{-1}$ to give

$$Y^2 - A^TY - YG^{-1}AG - HG = 0$$

put $Y = X + A^T$

$$X^2 + A^TA^T + XA^T + A^TX - A^TX - A^TA^T - XG^{-1}AG - A^TG^{-1}AG - HG = 0$$

or write as

$$X^2 + XP + Q = 0$$

where

$$P = A^T - G^{-1}AG$$

$$Q = -A^TG^{-1}AG - HG$$

and the solution of (1.57) is given by

$$K = (X + A^T)G^{-1} \tag{1.58}$$

The current methods for solving (1.35) for this P,Q will not compare favourably with those that solve the ARE directly since they cannot make use of the special properties of G and H and since it is not clear how the properties of X relate to the symmetric, positive definite K in (1.58).

CHAPTER 2 - PERTURBATION ANALYSIS

In this Chapter, we make use of the discussions and results of Section 1.3 to derive bounds pertaining to the sensitivity of the solution of quadratic matrix equations to small perturbations in the elements of the coefficient matrices.

Fundamental to the analysis is $F'(X)$, the derivative of $F(X)$, and its inverse $F'(X)^{-1}$. Section 2.1 discusses the existence of this derivative and shows that for each matrix equation, there is an equivalent matrical representation.

Section 2.2 presents a detailed analysis for the conditioning of each problem and derives bounds for the error in the solution of a perturbed problem. Each bound possesses the factor $||F'(X)^{-1}||$.

Section 2.3 shows how, by the way of small order examples, small perturbations in the initial data may cause massive perturbations for ill-conditioned problems. In fact, some perturbations in the solution are unbounded.

SECTION 2.1: The Derivative of $F(X)$

Consider the matrix function $F(X) \in \mathbb{R}^{n\times n}$ given by

$$F(X) = X^2 + PX + Q \tag{2.1}$$

The Fréchet derivative $F'(X)$ of $F(X)$ at $X \in \mathbb{R}^{n\times n}$ is a linear operator on $\mathbb{R}^{n\times n}$ defined by its action on a matrix, say $H \in \mathbb{R}^{n\times n}$, as follows

$$F(X+H) - F(X) = F'(X)H + 0(H^2) \tag{2.2}$$

with respect to (2.1), (2.2) gives

$$F(X+H) - F(X) = (X+H)^2 + P(X+H) + A - X^2 - PX - Q$$

such that

$$F'(X)H = (X+P)H + HX \tag{2.3}$$

This derivative operator and in particular its inverse operator will be referred to frequently in this thesis. Consequently, we require some estimates for their sizes. From the definition of an operator norm, we have that

$$||F'(X)|| = \sup_{H\neq 0} \frac{||F'(X)H||}{||H||} \tag{2.4}$$

Using the results of Section 1.2, concerning Kronecker products, (2.3) may be written as,

$$\text{vec}[F'(X)H] = [(X+P)\otimes I + I \otimes X^T]\ \text{vec}[H]$$

Taking Euclidean norms,

$$||F'(X)H||_E = ||\ \text{vec}[F'(X)H]|| = ||T\ \text{vec}[H]|| \tag{2.5}$$

where

$$T = (X+P)\otimes I + I \otimes X^T$$

using (2.4), (2.5) and the fact that $||H||_E = ||\ \text{vec}[H]||_{\not E}$,

$$||F'(X)|| = \sup_{H\neq 0} \frac{||F'(X)H||_E}{||H||_E} = \sup_{\text{vec}[H]\neq 0} \frac{||T\ \text{vec}[H]||_{\not E}}{||\ \text{vec}[H]||_{\not E}}$$

$$\therefore \quad ||F'(X)|| = ||T||_E$$

The conditions for existence and boundedness of the inverse operator are contained in the following theorem [Milne]:

Let $F'(X) \in \mathbf{R}^{n\times n}$ be an operator. If there exists a constant $m > 0$ such that

$$||F'(X)H|| \geq m||H|| \qquad \forall H \in \mathbf{R}^{n\times n} \tag{2.6}$$

then $F'(X)$ has a continuous inverse $F'(X)^{-1}$. Furthermore

$$||F'(X)^{-1}|| \leq \frac{1}{m} \tag{2.7}$$

Now assuming that T^{-1} exists and using the notation and the matrix T as above,

$$\text{vec}[H] = T^{-1}T \text{ vec}[H]$$

Taking norms

$$|| \text{ vec}[H]|| \leq ||T^{-1}|| \, ||T \text{ vec}[H]||$$

Let $m = \dfrac{1}{||T^{-1}||}$, then we have

$$m|| \text{ vec}[H]|| \leq ||T \text{ vec}[H]||$$

using (2.5)

$$m||H|| = m|| \text{ vec}[H]|| \leq || \text{ vec}[F'(X)H]|| = ||F'(X)H||$$

so that the condition (2.6) is satisfied and the norm of the inverse operator is given by

$$||F'(X)^{-1}|| \leq ||T^{-1}|| = ||[(X+P)\otimes I + I \otimes X^T]^{-1}|| \tag{2.8}$$

This result in particular, will be used frequently. The singularity of $F'(X)$ may be related to that of T [Golub, Nash & Van Loan] where they show that the operator $F'(X)$ is ~~non~~ singular if and only if the matrices $(X+P)$ and $-X$ have eigenvalues in common. Similarly $F'(X)$ is non-singular if and only if $(X+P)$ and $-X$ do not have eigenvalues in common. It follows that the conditioning of the operator norm with respect to inversion is directly related to that of matrix T.

The following analysis relates this matrix T to the Jacobian matrix of Section 1.5.

From (2.1),

$$(f_{ij}) = \sum_{k=1}^{n} x_{ik}x_{kj} + \sum_{k=1}^{n} p_{ik}x_{kj} + q_{ij}$$

Let fc, $xc \in \mathbf{R}^{n^2}$ be composed of the elements of F and X respectively taken a row at a time. The Jacobian, J of a function fc is the matrix of first partial derivatives of fc with respect to xc,

$$J = \left(\frac{\partial fc_i}{\partial xc_j}\right) = \begin{bmatrix} \dfrac{\partial f_{11}}{\partial x_{11}} & \cdots & \dfrac{\partial f_{11}}{\partial x_{1n}} & \dfrac{\partial f_{11}}{\partial x_{21}} & \cdots & \cdots & \dfrac{\partial f_{11}}{\partial x_{nn}} \\ \dfrac{\partial f_{12}}{\partial x_{11}} & \cdots & \dfrac{\partial f_{12}}{\partial x_{1n}} & \dfrac{\partial f_{12}}{\partial x_{21}} & \cdots & \cdots & \dfrac{\partial f_{12}}{\partial x_{nn}} \\ \vdots & & & & & & \vdots \\ \dfrac{\partial f_{nn}}{\partial x_{11}} & \cdots & \dfrac{\partial f_{nn}}{\partial x_{1n}} & \dfrac{\partial f_{nn}}{\partial x_{21}} & \cdots & \cdots & \dfrac{\partial f_{nn}}{\partial x_{nn}} \end{bmatrix}$$

Differentiating each element of F_{ij} and using Kronecker products, we have

$$J = (X + P) \otimes I + I \otimes X^T$$

which is exactly the matrix T above.

An analogous approach for the algebraic Riccati equation

$$A^T X + XA - XGX + \tilde{H} = 0$$

where $\tilde{H}$ is denoted so as not to confuse it with H, shows that

$$F(X + H) - F(X) = (A^T - XG)H + H(A - GX) + 0(H^2)$$

such that

$$F'(X)H = (A^T - XG)H + H(A - GX) \tag{2.9}$$

taking norms

$$\begin{aligned} ||F'(X)||_E &= \sup_{H \neq 0} \frac{||F'(X)H||}{||H||} \\ &= \sup_{H \neq 0} \frac{||\ \mathrm{vec}[F'(X)H]||}{||H||} \\ &= \sup_{H \neq 0} \frac{||\tilde{T}\ \mathrm{vec}[H]||}{||\ \mathrm{vec}[H]||} \\ &= ||\tilde{T}||_E \end{aligned}$$

Similarly $||F'(X)^{-1}|| \leq ||\tilde{T}^{-1}||$ where

$$\tilde{T} = (A^T - XG) \otimes I + I \otimes (A^T - XG) \tag{2.10}$$

This is also the expression for the Jacobian matrix.

As a result of this analysis, it is clear that the existence of the inverse operator depends upon the non-singularity of the matrix T (or $\tilde{T}$). Similarly, we can say that if T (or $\tilde{T}$) is well-conditioned then so is the derivative operator.

Moreover, for the algebraic Riccati equation, the stabilising solution satisfies $\mathrm{Re}(\lambda_i(A - GX)) < 0$ in which case $\tilde{T}$ is non-singular and the inverse operator exists.

SECTION 2.2: Conditioning of the Problem

We now examine how the presence of perturbations in the original data may affect any solution of the matrix equations.

Firstly consider the quadratic matrix equation,

$$F(X) = X^2 + PX + Q = 0 \tag{2.11}$$

Assume that P and Q are perturbed such that (2.11) becomes

$$G(X) = X^2 + (P + \Delta P)X + (Q + \Delta Q) \tag{2.12}$$

Now the computed solution, $\hat{X}$ that makes $F(X) = 0$ satisfies,

$$(X + \Delta X)^2 + (P + \Delta P)(X + \Delta X) + (Q + \Delta Q) = 0 \tag{2.13}$$

Multiplying out,

$$X^2 + X\Delta X + \Delta X X + \Delta X^2 + PX + \Delta PX + P\Delta X + \Delta P\Delta X + Q + \Delta Q = 0 \tag{2.14}$$

Re-arranging this gives,

$$X^2 + PX + Q + (X + P)\Delta X + \Delta X\, X + \Delta X^2 + \Delta PX + \Delta P\Delta X + \Delta Q = 0 \tag{2.15}$$

Using (2.3) and (2.11), (2.15) may be written as

$$F'(X)\Delta X = -(\Delta X^2 + \Delta PX + \Delta P\Delta X + \Delta Q)$$

Let $F'(X)^{-1}$ exist. Premultiply by $F'(X)^{-1}$ and take norms,

$$||\Delta X|| \leq ||F'(X)^{-1}||(||\Delta X||^2 + ||\Delta P||\,||X|| + ||\Delta P||\,||\Delta X|| + ||\Delta Q||) \tag{2.16}$$

Let $k = ||F'(X)^{-1}||\,||\Delta P||$, then

$$||\Delta X|| \leq \frac{1}{1-k}||F'(X)^{-1}||(||\Delta X||^2 + ||\Delta P||\,||X|| + ||\Delta Q||) \tag{2.17}$$

Let

$$\alpha = \frac{||F'(X)^{-1}||}{1-k}, \quad \beta = \alpha(||\Delta P||\,||X|| + ||\Delta Q||) \tag{2.18}$$

then (2.17) may be expressed as

$$-\alpha||\Delta X||^2 + ||\Delta X|| - \beta \leq 0 \tag{2.19}$$

To understand this inequality, consider the quadratic function,

$$y = -\alpha||\Delta X||^2 + ||\Delta X|| \quad - \beta$$

The gradient is,

$$\frac{dy}{d||\Delta X||} = -2\alpha||\Delta X|| + 1$$

The function y has turning points when the gradient vanishes. In this case there is only one turning point, at $||\Delta X||= \frac{1}{2\alpha}$ and since

$$\frac{d^2y}{d||\Delta X||^2} = -2\alpha < 0$$

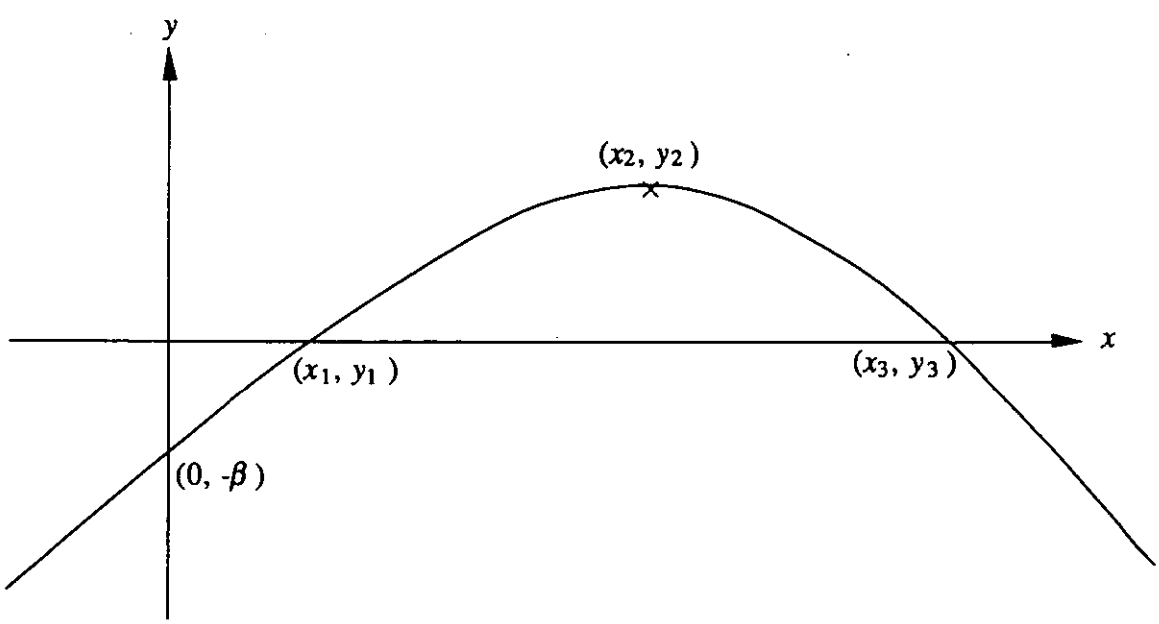
and $\alpha \gtrsim 0$, this point is a maximum. The y co-ordinate at this point is

$$y = \frac{1 - 4\alpha\beta}{4\alpha} > 0 \text{ for sufficiently small } ||\Delta P|| \text{ and } ||\Delta Q||$$

Now, the roots of y are

$$r_1 = \frac{1}{2\alpha} - \frac{\sqrt{1 - 4\alpha\beta}}{2\alpha} \quad \text{and} \quad r_2 = \frac{1}{2\alpha} + \frac{\sqrt{1 - 4\alpha\beta}}{2\alpha}$$

which for sufficiently small $||\Delta P||$ and $||\Delta Q||$ are real and positive. The curve of y may look like this,



where $(x_1, y_1) = (r_1, 0)$, $(x_2, y_2) = \left(\frac{1}{2\alpha}, \frac{1 - 4\alpha\beta}{4\alpha}\right)$, $(x_3, y_3) = (r_2, 0)$

The implication of these observations is that (2.19) holds for all $||\Delta X||$ satisfying,

$$||\Delta X|| \leq r_1 = \frac{1 - \sqrt{1 - 4\alpha\beta}}{2\alpha} \tag{2.20}$$

where α, β are given by (2.18).

Now for sufficiently small $||\Delta P||$ and $||\Delta Q||$,

$$(1 - 4\alpha\beta)^{\frac{1}{2}} \approx 1 - \frac{4\alpha\beta}{2}$$

so that (2.20) becomes

$$\begin{aligned} ||\Delta X|| \lesssim \beta &= \alpha(||\Delta P||\,||X|| + ||\Delta Q||) \\ &= \frac{||F'(X)^{-1}||}{1-k}(||\Delta P||\,||X|| + ||\Delta Q||) \end{aligned} \tag{2.21}$$

Now, from earlier, $k = ||F'(x)^{-1}||\,||\Delta P||$. ΔP is the matrix of perturbations in P and as such, $||\Delta P||$ will be of order $u||P||$. Typical, in computing machines using double precision arithmetic, $u \approx 10^{-18}$ so that we can safely say that $k < \frac{1}{2}$. Then, (2.21) may be written as

$$\frac{||\Delta X||}{||X||} \leq 2||F'(X)^{-1}||\left(||\Delta P|| + \frac{||\Delta Q||}{||X||}\right) \tag{2.22}$$

This relationship expresses the error in computing X in terms of the perturbations in the coefficient matrices. It suggests that if $F'(X)^{-1}$ exists and $F'(X)$ is well-conditioned at the solution then the computed solution is near the exact solution. If however, $F'(X)$ is ill-conditioned and therefore $||F'(X)^{-1}||$ is very large then we cannot guarantee that the computed solution is near the exact solution.

Now consider the matrix square root problem,

$$X^2 - A = 0 \tag{2.23}$$

The solution that solves the perturbed system satisfies

$$(X + \Delta X)^2 - (A + \Delta A) = 0 \tag{2.24}$$

The definition of the derivative for this problem is

$$F'(X)\Delta X = X\Delta X + \Delta X\, X \tag{2.25}$$

From (2.23) and (2.25), (2.24) may be written as,

$$F'(X)\Delta X = \Delta A - \Delta X^2$$

From the earlier discussion of Kronecker products, the derivative in (2.25) is non-singular if and only if X and $-X$ have no eigenvalues in common. In this case,

$$||\Delta X|| \leq ||F'(X)^{-1}||\,||\Delta A|| \overline{\#} ||F'(X)^{-1}||\,||\Delta X^2|| \tag{2.26}$$

This is a quadratic inequality in $||\Delta X||$. Using a similar approach to that for the quadratic matrix equation, we have that for sufficiently small $||\Delta A||$, (2.26) holds for all $||\Delta X||$ satisfying,

$$\begin{aligned} ||\Delta X|| &\leq ||F'(X)^{-1}||\,||\Delta A|| \\ \frac{||\Delta X||}{||X||} &\leq ||F'(X)^{-1}||\,\frac{||\Delta A||}{||X||} \end{aligned} \tag{2.27}$$

From (2.23), $||A|| = ||X^2|| \leq ||X||^2$

Taking roots, $||A||^{\frac{1}{2}} \leq ||X|| \Rightarrow \dfrac{1}{||X||} \leq \dfrac{1}{||A||^{\frac{1}{2}}}$

Substituting into (2.27),

$$\frac{||\Delta X||}{||X||} \leq ||F'(X)^{-1}||\,||A||^{\frac{1}{2}}\,\frac{||\Delta A||}{||A||} \tag{2.28}$$

This bound is valid when X is non-singular in which case $F'(X)$ is non-singular and the bound suggests that any relative error in X is only as large as the relative error in A multiplied by a constant. This constant say C, may be regarded as the magnification constant or condition number and is given by

$$C = ||F'(X)^{-1}||\,||A||^{\frac{1}{2}}$$

Now consider the conditioning of the Algebraic Riccati Equation,

$$A^TX - XA - XG\overset{X}{\underset{\wedge}{+}}\tilde{H} = F(X) = 0 \tag{2.29}$$

If the coefficient matrices are perturbed slightly then the computed solution satisfies

$$(A^T+\Delta A^T)(X+\Delta X)+(X+\Delta X)(A+\Delta A)-(X+\Delta X)(G+\Delta G)(X+\Delta X)+\tilde{H}+\Delta H = 0 \tag{2.30}$$

where

$$||\Delta A^T|| \leq u||A||,\ ||\Delta A|| \leq u||A||,\ ||\Delta G|| \leq u||G||,\ ||\Delta H|| \leq u||\tilde{H}|| \tag{2.31}$$

The Fréchet derivative in this case is,

$$F'(X)H = (A^T - XG)H + H(A - GX) \tag{2.32}$$

Also,

$$\hat{F}'(X)H = (A^T + \Delta A^T - X(G + \Delta G))H + H(A + \Delta A - (G + \Delta G)X) \tag{2.33}$$

Define $\Delta F'(X)H = \hat{F}'(X)H - F'(X)H$
then

$$\Delta F'(X)H = (\Delta A^T - X\Delta G)H + H(\Delta A - \Delta G X) \tag{2.34}$$

Substituting (2.29), (2.32) and (2.34) into (2.30) gives

$$F'(X)\Delta X + \Delta F'(X)\Delta X - \Delta X(G + \Delta G)\Delta X + R = 0 \tag{2.35}$$

where

$$R = \Delta A^T X + X\Delta A - X\Delta G X + \Delta H \tag{2.36}$$

Multiplying each side by $F'(X)^{-1}$ and taking norms,

$$||\Delta X|| = ||F'(X)^{-1}||\,||(\Delta X(G + \Delta G)\Delta X - \Delta F'(X)\Delta X - R)|| \tag{2.37}$$

$$||\Delta X|| \leq ||F'(X)^{-1}||(||G + \Delta G||\,||\Delta X||^2 + ||\Delta F'(X)||\,||\Delta X|| + ||R||) \tag{2.38}$$

Let

$$||F'(X)^{-1}||\,||\Delta F'(X)|| = k,$$

$$\alpha = \frac{||F'(X)^{-1}||}{1-k}||G + \Delta G||, \qquad \beta = \frac{||F'(X)^{-1}||\,||R||}{1-k}$$

then (2.38) becomes

$$-\alpha||\Delta X||^2 + ||\Delta X|| - \beta \leq 0 \tag{2.39}$$

This is a quadratic inequality in $||\Delta X||$. Using a similar analysis to that used for (2.19) yields,

$$||\Delta X|| \leq \beta = \frac{||F'(X)^{-1}||\,||R||}{1-k} \tag{2.40}$$

for sufficiently small $||\Delta A^T||$, $||A||$, $||G||$, $||\Delta H||$.

Now from (2.34) and (2.31),

$$||\Delta F'(X)|| \leq 2nu(||A|| + ||G||\,||X||)$$

since this is of order u we can assume that

$$k = ||F'(X)^{-1}||\,||\Delta F'(X)|| \leq \frac{1}{2} \tag{2.41}$$

From (2.36) and (2.41), (2.40) becomes

$$\frac{||\Delta X||}{||X||} \leq (2||F'(X)^{-1}||\left(2||\Delta A|| + ||\Delta G||\,||X|| + \frac{||\Delta H||}{||X||}\right) \tag{2.42}$$

This expression gives the relative error in computing X in terms of the absolute perturbation in the coefficient matrices and the inverse of the derivative operator. Clearly this relationship is only valid when the operator $F'(X)$ is non-singular. In fact the stabilising solution will always yield a non-singular $F'(X)$, although if it is ill-conditioned there is no guarantee that the computed solution is near the exact solution.

A slightly different approach to determining the sensitivity of the Riccati problem is given by [Byers, 1] where the term in (2.36) is represented as the sum of three linear operators such that

$$F'(X)^{-1}R = F'(X)^{-1}(\Delta H) - \theta(\Delta A) + \pi(\Delta G) \tag{2.43}$$

where

$$\theta(Z) = F'(X)^{-1}(Z^T X + XZ) \tag{2.44}$$

and

$$\pi(Z) = F'(X)^{-1}(XZX) \tag{2.45}$$

The operators θ, F' and π determine the sensitivity of X with respect to the uncertainty in the matrices, A, G and H respectively. Then the relative error in computing X is

$$\frac{||\Delta X||}{||X||} \leq \frac{4}{||X||}\left[||F'(X)^{-1}||\,||\Delta H|| + ||\theta||\,||\Delta A|| + ||\pi||\,||\Delta G||\right] \tag{2.46}$$

We can relate the bound in (2.46) to that derived in (2.42). From the definition of an operator (Euclidean) norm,

$$\begin{aligned} ||\theta|| &= \sup_{Z\neq 0} \frac{||F'(X)^{-1}(Z^T X + XZ)||}{||Z||_E} \\ &= \sup_{\mathrm{vec}(Z)\neq 0} \frac{||F'(X)^{-1}T\ \mathrm{vec}(Z)||}{||\ \mathrm{vec}(Z)||} \\ &= ||F'(X)^{-1}T|| \leq ||F'(X)^{-1}||\,||T|| \end{aligned} \tag{2.47}$$

Since

$$T = X \otimes I + I \otimes X^T \tag{2.48}$$

$$||\theta|| \leq ||F'(X)^{-1}||\,||2X|| \tag{2.49}$$

Similarly

$$||\pi|| \leq ||F'(X)^{-1}||\,||X||^2 \tag{2.50}$$

Substituting these into (2.46) gives,

$$\frac{||\Delta X||}{||X||} \leq 4||F'(X)^{-1}||\left[2||\Delta A|| + ||\Delta G||\,||\Delta X|| + \frac{||\Delta H||}{||X||}\right] \tag{2.51}$$

We observe that the bound on the right hand side of (2.46) is less than the bound on the right hand side of (2.51). Just how close the bounds are depends largely on the particular problem.

[Byers, 1] performs a comparison of the condition number obtained from (2.46),

$$K_C = \frac{||F'(X)^{-1}||\,||H|| + ||\theta||\,||A|| + ||\pi||\,||a||}{||X||}$$

with a condition number obtained from a bound similar to (2.42), [Byers, 2],

$$K_B = ||F'(X)^{-1}|| \left(2||A|| + ||G||\,||X|| + \frac{||H||}{||X||}\right)$$

and with [Arnold & Laub],

$$K_A = \frac{||F'(X)^{-1}||\,||H||}{||X||}$$

He shows that $K_A \leq K_C \leq K_B$.

[Arnold & Laub] observe that K_A tends to be too small and K_B tends to be too large. In numerical experiments [Byers, 1], K_C compares favourably with K_A and K_B.

SECTION 2.3: Some examples of perturbed problems

The difficulties encountered when solving problems with a singular derivative at a solution may be illustrated by looking at some examples.

Consider the square root problem

$$X^2 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{2.52}$$

Let a solution of (2.52) be of the form

$$X = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{2.53}$$

where $a,\ b,\ c,\ d \in \mathbf{R}$.

Substituting (2.53) into (2.52) yields a set of four non-linear equations in the four unknowns,

$$a^2 + bc = 0$$

$$(a+d)b = 0$$

$$(a+d)c = 0$$

$$bc + d^2 = 0$$

These imply that either $b = 0,\ c = 0$ or $a + d = 0$.
If $b = 0$ then $a = 0,\ d = 0$ and c is arbitrary
If $c = 0$ then $a = 0,\ d = 0$ and b is arbitrary
If $a = -d$ then b is arbitrary and $c = -a^2/b$.

Therefore there are three groups of solution that satisfy (2.52),

$$X_1 = \begin{pmatrix} 0 & \alpha \\ 0 & 0 \end{pmatrix}$$

$$X_2 = \begin{pmatrix} 0 & 0 \\ \beta & 0 \end{pmatrix}$$

$$X_3 = \begin{pmatrix} \gamma & \delta \\ \epsilon & -\gamma \end{pmatrix}$$

where $\alpha,\ \beta,\ \gamma,\ \delta$ are arbitrary and $\epsilon = -\dfrac{\gamma^2}{\delta}$.

The partial derivatives of the corresponding functions are

$$J(X_1) = \begin{pmatrix} 0 & 0 & \alpha & 0 \\ \alpha & 0 & 0 & \alpha \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha & 0 \end{pmatrix}, \text{ then } \quad ||F'(X_1)|| = ||J(X_1)||$$

$$J(X_2) = \begin{pmatrix} 0 & \beta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \beta & 0 & 0 & \beta \\ 0 & \beta & 0 & 0 \end{pmatrix}, \text{ then } \quad ||F'(X_2)|| = ||J(X_2)||$$

$$J(X_3) = \begin{pmatrix} 2\gamma & \epsilon & \delta & 0 \\ \delta & 0 & 0 & \delta \\ \epsilon & 0 & 0 & \epsilon \\ 0 & \epsilon & \delta & -2\gamma \end{pmatrix}, \text{ then } \quad ||F'(X_3)|| = ||J(X_3)||$$

These derivatives are all singular. This implies that the bound (2.28) is not defined and therefore cannot determine the effect of any perturbations in the coefficient matrix. To see this, assume that in generating the coefficient matrix a small inaccuracy occurs in the element in position (1,2) so that the problem (2.52) is now perturbed to

$$X^2 - \begin{pmatrix} 0 & \omega \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{2.54}$$

where ω is a small non-zero element.

Again assume that the solution of (2.52) is of the form (2.53) and substitute this into (2.54) giving

$$a^2 + bc = 0$$

$$(a + d)b = \omega$$

$$(a + d)c = 0$$

$$bc + d^2 = 0$$

Since $\omega > 0$, $(a + d)$ and b must be non-zero. But this is not consistent with the remaining equations and therefore a solution of (2.54) does not exist. Thus a small perturbation in the coefficient matrix has caused unbounded changes in the solution to (2.52) and hence the problem is ill-conditioned.

Now consider the following problem where not all the solutions possess singular derivatives,

$$X^2 - \begin{pmatrix} y & 0 \\ 0 & y \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{2.55}$$

where $y \geq 1.0$.

Once again assume that that solutions are of the form (2.53) and substitute into (2.55) giving

$$a^2 + bc = y$$

$$(a + d) = 0$$

$$(a + d)c = 0$$

$$bc + d^2 = y$$

Solving this set of non-linear equations gives the following set of solutions,

$$X_1 = \pm \begin{pmatrix} \sqrt{y} & 0 \\ 0 & \sqrt{y} \end{pmatrix}$$

$$X_2 = \pm \begin{pmatrix} \sqrt{y} & 0 \\ 0 & -\sqrt{y} \end{pmatrix}$$

$$X_3 = \pm \begin{pmatrix} -\sqrt{y} & \alpha \\ 0 & \sqrt{y} \end{pmatrix}$$

$$X_4 = \pm \begin{pmatrix} -\sqrt{y} & 0 \\ \beta & \sqrt{y} \end{pmatrix}$$

$$X_5 = \pm \begin{pmatrix} \delta & \gamma \\ \epsilon & -\delta \end{pmatrix}$$

where α, β, γ, δ are arbitrary and

$$\epsilon = \frac{y - \delta^2}{\gamma}$$

The derivative of the function $F(X_1)$ is,

$$J(X_1) = 2\sqrt{y} I_4$$

which is clearly non-singular since $y \geq 1$. We can now estimate $||F'(X)^{-1}||$ from the singular values,

$$J^T(X_1)J(X_1) = 4y I_4 \Rightarrow \sigma_1 = \sigma_2 = \sigma_3 = \sigma_4 = 2\sqrt{y}$$

then,

$$||F'(X_1)^{-1}|| \leq \frac{1}{\sqrt{y}} \leq 1$$

such that $F'(X_1)$ is well-conditioned.

The partial derivatives corresponding to X_2, X_3, X_4 and X_5 are all singular since for the square root problem any 2×2 solvent of the form

$$X = \begin{pmatrix} a & b \\ c & -a \end{pmatrix}$$

has the derivative

$$J(X) = \begin{pmatrix} 2a & c & b & 0 \\ b & 0 & 0 & b \\ c & 0 & 0 & c \\ 0 & c & b & -2a \end{pmatrix}$$

which is clearly singular.

Now since $F'(X_1)$ is non-singular and well bounded, (2.28) suggests that the norm of the error in X_1 due to a perturbed system will be well-bounded. Therefore, consider a non-zero inaccuracy, ω in the coefficient matrix, such that,

$$X^2 - \begin{pmatrix} y & \omega \\ 0 & y \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{2.56}$$

Substitute (2.53) into (2.56) giving

$$a^2 + bc = y$$

$$(a + d)b = \omega$$

$$(a + d)c = 0$$

$$bc + d^2 = y$$

These equations imply that $c = 0,\ a \neq -d$ and $d^2 = y,\ a^2 = y,\ b = \dfrac{\omega}{a + d}$.

Therefore the solutions of (2.56) are,

$$X = \pm \begin{pmatrix} \sqrt{y} & \omega/2\sqrt{y} \\ 0 & \sqrt{y} \end{pmatrix} \tag{2.57}$$

Now since, for X_1, $||F'(X)^{-1}||$ is known and

$$||Q|| = \frac{y}{\sqrt{2}} \quad \text{and} \quad ||\Delta Q|| = \frac{\omega}{2}$$

we would expect, from (2.28), that the relative error is bounded by,

$$\frac{||\Delta X||}{||X||} \leq \left| \frac{\omega,}{\sqrt{2}y - 2\sqrt{\omega}y} \right|$$

In fact

$$\frac{||X - X_1||}{||X||} = \left| \frac{\omega}{2y\sqrt{2y}} \right| \leq \left| \frac{\omega}{\sqrt{2}y - 2\sqrt{\omega y}} \right| \tag{2.58}$$

but observe that

$$\frac{||X - X_i||}{||X||} \geq |\sqrt{y}| \qquad \text{for } i = 2, 3, 4, 5$$

The relative error in X_1, due to the perturbation ω is less than ω and satisfies (2.28), as expected. However, the relative error in X_i for $i = 2, 3, 4, 5$ is proportional to the elements of the original coefficient matrix.

SECTION 2.4: Conclusions

This chapter derives bounds for the sensitivity of the solutions of the quadratic matrix equations, to small perturbations in their coefficient matrices. The results indicate that when the derivative of $F(X)$ is singular at a solution then the derived bounds can reveal no information regarding the conditioning of the problems. This is a direct consequence of the inverse of $F'(X)$ not being defined.

When $F'(X)$ is non-singular with a well-conditioned inverse, then $||F'(X)^{-1}||$ is reasonably sized and the computed solution is near to the exact solution.

When $F'(X)$ is non-singular with an ill-conditioned inverse then $||F'(X)^{-1}||$ is large and the computed solution is not guaranteed to be near the exact solution.

CHAPTER 3 - COMPUTING THE CHARACTERISTIC POLYNOMIAL OF A MATRIX

SECTION 3.1: Introduction

In many of the methods of Chapter 4, where we solve the quadratic matrix equation using Elimination method techniques, it is required to compute the coefficients of the characteristic polynomial of a matrix, usually within an iterative scheme. Consequently, the method used must be efficient and stable and must determine the coefficients accurately.

There have been a number of methods proposed, the most well-known and widely used one being the stable LeVerriers method. We discuss the original LeVerriers approach and show that it's instability can be overcome by considering a reformulation of the method. One of the oldest methods based on similarity transformations is Danilevski's method, which pays no attention to the stability properties of the algorithm. An extension of this method is discussed which addresses the stability aspects but which can at best be viewed as a 2-phase algorithm which uses stable orthogonal similarity transformations for the first phase and unavoidably, unstable elementary similarity transformations for the second phase. A recent approach extends the 2-phase Danilevski's method to one which stabilizes the elementary similarity transformations of the second phase by reducing the original matrix to Block Frobenius form. It has always been viewed that Krylov's method involves the solution of an ill-conditioned linear system. It is shown, firstly, how rounding errors give rise to this ill-conditioned system and secondly that there exists a certain class of matrices for which Krylov's method will yield an accurate characteristic polynomial. A new approach for matrices possessing distinct eigenvalues is discussed. It is shown that the approach involves the solution of a linear system. The matrix in this system is the well-known Vandermonde matrix which is known to be generally ill-conditioned with respect to solving the linear system. However, we show that there exist a certain class of matrices for which this interpolation method will yield a very accurate solution.

Accuracy of solutions is not solely dependent on the stability of the algorithm, but also on the sensitivity of the problem to small perturbations in its coefficients. Therefore we begin by looking at the condition of the problem.

SECTION 3.2: Condition of the Problem

Let the characteristic polynomial of a matrix $X \in \mathbb{R}^{n \times n}$ be

$$f(\lambda) = |X - \lambda I| = \lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \ldots + a_n \tag{3.1}$$

where a_i are the scalars to be determined.

Suppose that the elements of X are perturbed by a small amount ϵ, then the characteristic polynomial of $X + \epsilon X$ is [Wilkinson],

$$\hat{f}(\lambda) = |X + \epsilon X - \lambda I| = \lambda^n + \hat{a}_1\lambda^{n-1} + \hat{a}_2\lambda^{n-2} + \ldots + \hat{a}_n$$

where

$$\begin{aligned} \hat{a}_i &= a_i(1+\epsilon)^i \\ &= a_i(1+i\epsilon) + O(\epsilon^2) \end{aligned}$$

Taking norms,

$$\begin{aligned} ||\hat{a}_i - a_i|| &= ||a_i i\epsilon + O(\epsilon^2)|| \\ &\leq i||a_i||\epsilon + O(\epsilon^2) \end{aligned} \tag{3.2}$$

Since ϵ is small, we ignore terms of $O(\epsilon^2)$. Then

$$\frac{||\hat{a}_i - a_i||}{||a_i||} \leq i\epsilon \leq n\epsilon \tag{3.3}$$

This says that the relative errors in the computed coefficients of a perturbed matrix are only of the same order as the perturbations. That is, the problem is well-conditioned and that any stable method used to determine the coefficients will produce accurate solutions.

SECTION 3.3: LeVerriers Method

This method requires the traces of the powers of a matrix. Define

$$\text{tr}\,(X^k) = \sum_{i=1}^{n} x_{ii}^{(k)} = s_k \quad \text{say} \tag{3.4}$$

where $x_{ii}^{(k)}$ denotes the element of X^k in position (i,i), [Faddeev & Faddeeva].

Now since the trace of a matrix is equivalent to the sum of the eigenvalues of that matrix, we have that

$$\text{tr}\,(X^k) = \sum_{i=1}^{n} \lambda_i^k = s_k \tag{3.5}$$

The problem of determining the eigenvalues of a matrix was discussed in Section 1.4 where it was shown that the problem is generally well-conditioned with ill-conditioning for particular types of non-normal matrices. The QR method is a stable technique for determining the eigenvalues and provides accurate solutions for a well-conditioned eigenvalue problem. In fact,

$$\hat{\lambda}_i = \lambda_i(1+\epsilon) \qquad |\epsilon| \leq u \tag{3.6}$$

The algorithm for determining s_k from (3.5) is as follows.

$$\begin{aligned}
&\text{For} \quad j = 1, n \\
&\qquad t_{ij} = \lambda_j \\
&\text{For} \quad j = 1, n \\
&\qquad \text{For} \quad i = 2, n \\
&\qquad\qquad t_{ij} = t_{i-1,j} * \lambda_j \\
&\text{For} \quad i = 1, n \\
&\qquad s_i = 0.0 \\
&\qquad \text{For} \quad j = 1, n \\
&\qquad\qquad s_i = s_i + t_{ij}
\end{aligned} \tag{3.7}$$

The operations count for determining the λ_i and the s_i is approximately $8n^3$ and the storage required is $n^2 + 2n$.

Once the s_k have been found, the coefficients are determined from the following Newton formulae [Dief],

$$-ka_k = s_k + a_1 s_{k-1} + a_2 s_{k-2} + \ldots + a_{k-1} s_1 \qquad (3.8)$$

$$\text{for} \quad k = 1, \ldots, n.$$

This may be written algorithmically as,

$$a_1 = a_2 = \ldots = a_n = 0 \qquad (3.9)$$

For $k = 1, n$

For $i = 1, k-1$

$$a_k = a_k + a_i s_{k-i}$$

$$a_k = \frac{-s_k - a_k}{k}$$

The operations count is $O(n^2)$ with no additional requirements for storage.

Applying the floating point error analysis of Section (1.3) to the algorithm that effects (3.5), with computed $\hat{\lambda}_i$ (3.6), gives

$$fl(\hat{\lambda}_i^k) = \lambda_i^k (1+\epsilon)^k (1+\epsilon)^{k-1} = \lambda_i (1 + (2k-1)\epsilon) + O(\epsilon^2)$$

$$\hat{s}_k = fl(\sum \hat{\lambda}_i^k) = fl\left(\sum_{i=1}^{n} \lambda_i^k (1 + (2k-1)\epsilon) + O(\epsilon^2)\right) \qquad (3.10)$$

Re-arranging and taking norms,

$$|\hat{s}_k - s_k| \leq u(2k - 1 + n - 1)|s_k| \leq 3nu|s_k| \qquad (3.11)$$

This bound suggests that (3.7) is a stable algorithm for determining s_k. A similar treatment for algorithm (3.9) reveals that the computed characteristic polynomial possesses coefficients of a matrix 'near' to the original matrix,

$$|X(1+\delta) - \lambda I| = \hat{f}(\lambda) = \lambda^n + a_1(1+\delta)\lambda^{n-1} + a_2(1+\delta)\lambda^{n-2} + \ldots + a_n(1+\delta)$$

$$\text{where} \quad |\delta| = |4n^2\epsilon| \leq 4n^2 u \qquad (3.12)$$

This suggests that the algorithm in (3.9) is a stable technique for computing the coefficients a_k given ~~that~~ the sums s_k. However, in this analysis, we have presumed that the problem of determining the a_k from (3.8) is a well-conditioned one. In fact, this is not the case and [Wilkinson] states that it is common for severe cancellations to take place.

Therefore, since LeVerriers method requires the solution of an ill-conditioned problem, the method, in this form, must be considered unstable.

[Faddeev & Faddeeva] overcome this limitation by re-addressing the approach above and considering the traces of matrices rather than the sum of the eigenvalues to yield a modified stable LeVerriers method.

SECTION 3.4: Stable LeVerriers Method

This method determines the coefficients by successively computing the traces of certain matrices $A_1, A_2, \ldots, A_n$, as follows, [Faddeev & Faddeeva],

Let $A_1 = X$ then $a_1 = -\operatorname{tr}(A_1)$, denote $B_1 = A_1 - a_1 I$

Let $A_2 = XB_1$ then $a_2 = \frac{1}{2}\operatorname{tr}(A_2)$, denote $B - 2 = A_2 - a_2 I$

$\vdots$

Let $A_n = XB_{n-1}$ then $a_n = \frac{(-1)^n}{n}\operatorname{tr}(A_n)$, denote $B_n = A_n - a_n I$

where $a_1, a_2, \ldots, a_n$ are the coefficients of the characteristic polynomial of X.

This method is a reformulation of LeVerriers method and gives rise to the following algorithm:

let $X = (X_{ij})$ and $B = I$, $a_1 = a_2 = \ldots = a_n = 0$, sign $= -1$ then

For $p = 1, n$

For $j = 1, n$

For $i = 1, n$

$v_i = 0.0$

For $k = 1, n$

$v_i = v_i + x_{ik} b_{kj}$

For $i = 1, n$

$b_{ij} = v_i$

For $j = 1, n$

$a_p = a_p + b_{jj}$

$a_p = (a_p + \text{sign})/p$

sign $= -$sign

For $i = 1, n$

$b_{ii} = b_{ii} - a_p$

next p

The operations count is $n^4 - n^3 + O(n)$ and the storage required is $2n^2 + 2n$.

Applying the floating point error analysis to this algorithm reveals that the computed characteristic polynomial is the exact characteristic polynomial of a matrix

$$X(1+\delta) \qquad \text{where} \qquad |\delta| \leq n^2 u \tag{3.13}$$

suggesting that this method is stable and will always yield accurate coefficients.

SECTION 3.5: Danilevski's Method and an Extension

This method [Danilevski] is applicable for non-derogatory matrices since in this case X may be reduced to its companion form by a sequence of simple elementary (similarity) transformations.

If R_k is the transforming matrix at step k of the reduction,

$$X^{(k+1)} = R_k^{-1} X^{(k)} R_k, \qquad X^{(1)} = X \tag{3.14}$$

where

$$R_k = \begin{bmatrix} 1 & \dots & r_{1,k+1} & \dots & 0 \\ 0 & \dots & r_{2,k+1} & \dots & 0 \\ \vdots & & \vdots & & \\ 0 & & r_{n,k+1} & & 1 \end{bmatrix}$$

and

$$R_k^{-1} = \frac{1}{r_{k+1,k+1}} \begin{bmatrix} 1 & \dots & -r_{1,k+1} & \dots & 0 \\ 0 & \dots & -r_{2,k+1} & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & \dots & -r_{n,k+1} & \dots & 1 \end{bmatrix}$$

The companion matrix C, given by

$$C = \begin{bmatrix} 0 & 0 & \dots & a_n \\ 1 & 0 & \dots & a_{n-1} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & a_1 \end{bmatrix}$$

is such that

$$C = X^{(n)} = (R_1 R_2 \dots R_{n-1})^{-1} X R_1 R_2 \dots R_{n-1} \tag{3.15}$$

Since the characteristic polynomial is invariant under similarity transformations, C and X have the same characteristic polynomial.

The following algorithm effects the transformation in (3.14)

For $k = 1, n-1$

For $i = 1, n$

For $j = 1, n$

$$b_{ij}^{(k)} = \frac{1}{x_{i,k}^{(k)}} x_{i,j}^{(k)} \qquad \text{when} \quad i = k+1$$

$$b_{ij}^{(k)} = x_{ij}^{(k)} - x_{ik}^{(k)} b_{k+1,j}^{(k)} \qquad \text{otherwise}$$

For $i = 1, n$

For $j = 1, n$

$$x_{ij}^{(k+1)} = b_{ij}^{(k)}$$

$$x_{ij}^{(k+1)} = \sum_{\ell=1}^{n} b_{ij}^{(k)} x_{jk}^{(k)} \qquad \text{when} \quad j = k+1$$

Next k

The operations count for this algorithm is $2n^3$ and the storage required is $2n^2$.

This is the usual description of Danilevski's method and pays no attention to numerical stability [Wilkinson]. An approach possessing better stability properties, firstly reduces a general X to an upper Hessenberg matrix H.

$$H = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1,n-1} & h_{1n} \\ k_2 & h_{21} & \dots & h_{2,n-1} & h_{2n} \\ \vdots & \vdots & & \vdots & \\ 0 & 0 & & k_n & h_{nn} \end{bmatrix}$$

by using stable orthogonal transformations. Then H is reduced by elementary similarity transformations to F, where

$$F = \begin{bmatrix} 0 & 0 & \dots & \dots & x_n \\ k_2 & 0 & \dots & \dots & x_{n-1} \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & & k_n & x_1 \end{bmatrix}$$

Then finally there exists a diagonal similarity transformation D such that

$$C = D^{-1} F D \tag{3.16}$$

is in companion form.

We assume that the k_i are non-zero, since if any $k_i = 0$ then the relevant smaller Hessenberg matrices are reduced in exactly the same way.

The transformation from H to F is affected by the following algorithm:

$$
\begin{aligned}
&\text{For} \quad r = 1, n-1 \\
&\qquad \text{For} \quad i = 1, r \\
&\qquad\qquad s_{i,r+1} = \frac{h_{ir}}{k_{r+1}} \\
&\qquad\qquad \text{For} \quad j = 1, n \\
&\qquad\qquad\qquad h_{ij} = h_{ij} - s_{i,r+1} h_{r+1,j} \\
&\qquad \text{For} \quad i = 1, r \\
&\qquad\qquad h_{i,r+1} = h_{i,r+1} + k_{i+1} s_{i,r+1} \\
&\text{Next} \quad r
\end{aligned}
\tag{3.17}
$$

For the reduction in (3.16),

$$D = \text{diag}\,(1,\ k_2,\ k_2, k_3 \ldots k_n)$$

and the coefficients are given by

$$a_i = k_2 k_3 \ldots k_i x_i$$

The operations count for the whole process is about $2n^3$ and the storage requirements are $n^2 + 2n$.

The numerical processes for computing H and for the reduction in (3.16) are stable. The algorithm (3.17), however, is not so since if in a typical step, $|k_{r+1}|$ is much smaller than the $|h_{ir}|$, the multipliers $s_{i,r+1}$ will be unacceptably large and numerical instability will result. This situation will arise if X is nearly derogatory. Unfortunately it is not possible to use interchanges because this would destroy the pattern of zeros. Hence, there is no related transformation based on orthogonal matrices which would stabilize this method.

SECTION 3.6: Block Frobenius Method

Here we extend the approach in the previous section to yield a stable algorithm for the coefficients of a general matrix X. [Wang & Chen] reduce the upper Hessenberg matrix H to Block Frobenius form, as illustrated in the following example for $n = 8$,

$$B = \left[\begin{array}{ccc:cc:ccc} b_1 & b_2 & b_3 & g_1 & g_2 & h_1 & h_2 & h_3 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & e_1 & c_1 & c_2 & f_1 & f_2 & f_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & e_2 & d_1 & d_2 & d_3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right] \tag{3.18}$$

where $|e_i|$ are relatively small numbers or zero.

In this reduction, eliminations using small pivotal elements are avoided and the computation is numerically stable. Since the reduction is via elementary similarity transformations the characteristic polynomials of B and H (and X) coincide. In fact,

$$f(\lambda) = \det \begin{bmatrix} \lambda^3 - b_1\lambda^2 - b_2\lambda - b_3 & -g_1\lambda - g_2 & -h_1\lambda^2 - h_2\lambda - h_3 \\ e_1 & \lambda^2 - c_1\lambda - c_2 & -f_1\lambda^2 - f_2\lambda - f_3 \\ 0 & e_2 & \lambda^3 - d_1\lambda^2 - d_2\lambda - d_3 \end{bmatrix} \tag{3.19}$$

(3.19) is multiplied out to yield the required polynomial. The algorithm that effects this method is given in Appendix A2.13. The operations count is dependent on the form of the elementary divisors of X, but from practical experience it is estimated by $(c+1)n^3$ where c is the number of Frobenius matrices on the diagonal of B. The storage requirements for this method are $n^2 + n$.

From Section 1.3, the rounding error for the Householder reduction of a matrix to upper Hessenberg form is

$$\hat{H} = R^{-1}(X + E)R, \tag{3.20}$$

where

$$||E|| \leq cn^2 u ||X||$$

If $\hat{H}$ has well-conditioned sub-diagonal elements, then B will be the companion matrix in which case the computed $\hat{B}$ would satisfy

$$\hat{B} = S^{-1}(\hat{H} + F)S \tag{3.21}$$

where

$$||F|| \leq 8n^2u||H|| \leq 8n^2u||X||$$

However, if any of the subdiagonal elements of $\hat{H}$ is very small then the computed $\hat{B}$ would certainly satisfy (3.20) also. Hence combining (3.20) and (3.21),

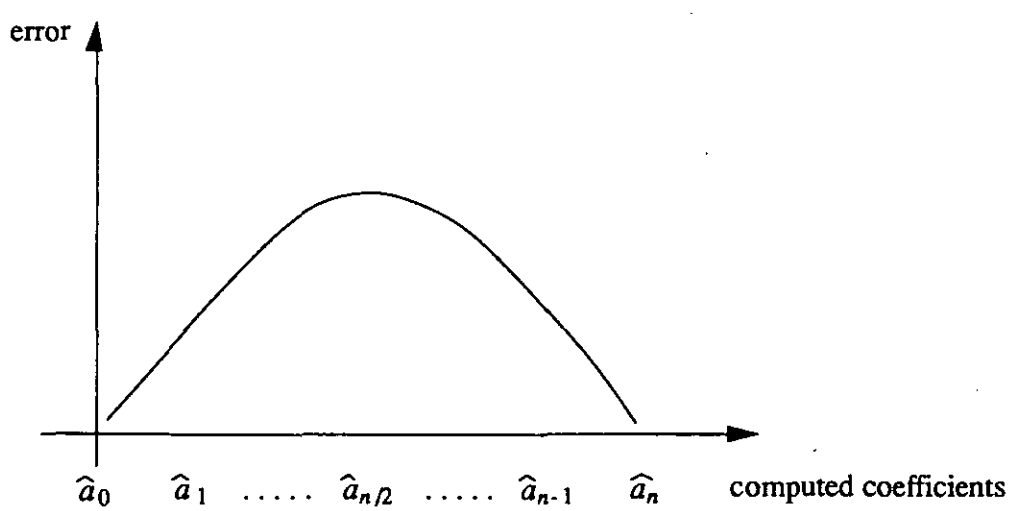
$$\hat{B} = T^{-1}(X + E)T$$

where

$$||E|| \leq (c + 8)n^2u||X||$$

If $\hat{B}$ has k Frobenius blocks, then it may be regarded as an upper Hessenberg partitioned matrix, as in the example in (3.18). This can then be regarded as a $k \times k$ upper Hessenberg polynomial matrix, as in (3.19), with a zero subdiagonal element corresponding to a derogatory X or a small subdiagonal element (corresponding to a possible source of ill-conditioning only if the element were to be used as a pivot).

The error involved in expanding the polynomial matrix is dependent on the number of Frobenius blocks. The error in the computed coefficients can be viewed as a normal distribution curve,



An upper bound for the maximum possible relative error is

$$|\Delta| \leq 2^{n/2}u \tag{3.22}$$

and this occurs when there is a maximum of int $(n/2)$ Frobenius blocks. At first sight it would seem that this bound is too big to be useful. In practice however, this bound

represents the worst possible case and is never attained. Thus for non-large order systems the error in the coefficients will be small and well-bounded. Consider the following,

$$n = 10, \qquad |\Delta| \leq n^2 u$$

$$n = 20, \qquad |\Delta| \leq 3n^2 u$$

$$n = 30, \qquad |\Delta| \leq 5n^3 u$$

$$n = 50, \qquad |\Delta| \leq n^6 u$$

Since on a double precision machine $u \approx 10^{-18}$ we are guaranteed accuracy to 11 decimal places for a matrix of order 50.

SECTION 3.7: Krylov's Method

The Cayley-Hamilton theorem states that every matrix satisfies its own characteristic polynomial,

$$f(X) = X^n + a_1 X^{n-1} + \ldots + a_n I$$

The characteristic equation is $f(X) = 0$ such that

$$a_1 X^{n-1} + a_2 X^{n-2} + \ldots + a_n I = -X^n \tag{3.23}$$

Now (3.23) represents a system of n^2 linear equations in the n unknowns $a_1,\ a_2, \ldots, a_n$. From this set, a subset of n linearly independent equations is necessary to uniquely determine the a_i.

Now consider the first column of X^{n-i} and denote it as the n-vector t_i. Equating the coefficients in the first column of (3.23) gives

$$a_1 t_1 + a_2 t_2 + \ldots + a_n t_n = -t_0$$

We may write this in a matrix form with

$$t_j = (\tau_{1j},\ \tau_{2j}, \ldots, \tau_{nj})^T \qquad \text{where} \quad j = 1, \ldots, n-1$$

and $\quad t_n = (1, 0, \ldots, 0)^T$

$$\begin{pmatrix} \tau_{11} & \tau_{12} & \cdots & \cdots & \tau_{1,n-1} & 1 \\ \tau_{21} & \tau_{22} & \cdots & \cdots & \tau_{2,n-1} & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ \tau_{n1} & \tau_{n2} & & & \tau_{n,n-1} & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = - \begin{pmatrix} t_{10} \\ t_{20} \\ \vdots \\ t_{n0} \end{pmatrix}$$

This is a linear matrix equation of the form

$$Ta = b \tag{3.24}$$

If T is non-singular then (3.24) may be solved uniquely to determine the a_i.

If the matrix T is singular then the second columns of X^{n-i} $(i = 0, n)$ are used to form T, etc.

The following new theorem concerns the existence of a non-singular matrix T.

Theorem

If the minimum polynomial of a matrix X is equal to its characteristic polynomial, i.e. X is non-derogatory, then there always exists a non-singular matrix T.

Moreover, any column of X^{n-i} may be used to form T.

Proof

Conversely, assume that T is singular. Then its rows are linearly dependent,

$$\alpha_1 t_1 + \alpha_2 t_2 + \ldots + \alpha_n t_n = 0 \tag{3.25}$$

for some $\alpha_1,\ \alpha_2, \ldots, \alpha_n$, not all zero.

Now $t_i = X^{n-i}y$ where y is some column-vector of size n of the form $(0, 0 \ldots 0, 1, 0 \ldots 0)^T$.

$$\alpha_1 X^{n-1}y + \alpha_2 X^{n-2}y + \ldots + \alpha_{n-1} Xy + \alpha_n y = 0$$

$$(\alpha_1 X^{n-1} + \alpha_2 X^{n-2} + \ldots + \alpha_n I)y = 0$$

Now since the minimum polynomial - the unique monic polynomial of least degree that annihilates X - and the characteristic polynomial of X are the same, and y is non-zero, (3.25) can be valid only if

$$\alpha_1 = \alpha_2 = \alpha_3 \ldots = \alpha_n = 0$$

which contradicts our assumption that T is singular.

Clearly the position of the 1 in the n-vector y is arbitrary.

As an aside, if X is derogatory then the minimum polynomial is not equal to the characteristic polynomial and T will be singular. In this case if Gaussian Elimination is used to solve (3.24) and rank $(T) = r$ then at the r^{th} stage of the Elimination , the element in position $t_{r+1,\ r+1}$ is zero and the unique scalars $a_1,\ a_2 \ldots a_r$ are the coefficients of the minimum polynomial of X.

The following algorithm computes the matrix $T = (\tau_{ij})$:

```
For     k = 1, n
        τ_k1 = x_k1
For     j = 2, n
        For     i = 1, n                                   (3.26)
                τ_ij = 0.0
                For     k = 1, n
                        τ_ij = τ_ij + x_ik τ_k,j-1

b_1 = - τ_1n
τ_1n = 0.0
For     k = 2, n
        b_k = -τ_kn
        τ_kn = 0.0
```

The operations count is $n^2(n-1)$. Since Gaussian Elimination requires $\frac{1}{3}n^3+n^2$ operations, the total count for Krylov's method is $\frac{4}{3}n^3$ and the storage requirements are $2n^2+2n$.

Now let t_j represent the columns of T. These are generated by the following recursion,

$$t_j = Xt_{j-1} \qquad j = 2, \ldots, n-1 \tag{3.27}$$

We try to find bounds for the rounding errors incurred in the computation of these t_j. Using the notation and results of Section 1.3,

$$\begin{aligned}
\hat{t}_j &= fl(\hat{X}\hat{t}_{j-1}) \\
&= fl(X(1+\epsilon)t_{j-1}(1+\Delta_{j-1})) \qquad\qquad |\epsilon| \leq u \\
&= X(1+\epsilon)t_{j-1}(1+\Delta_{j-1})(1+n\epsilon) \\
&= Xt_{j-1}(1+(n+1)\epsilon)(1+\Delta_{j-1}) + O(\epsilon^2) \\
&= t_j(1+\Delta_j) + O(\epsilon^2)
\end{aligned}$$

where

$$\Delta_j = \Delta_{j-1}(1+(n+1)\epsilon) + (n+1)\epsilon \tag{3.28}$$

Since $t_0 = (1, 0, \ldots, 0)^T$ is known exactly,

$$\begin{aligned}
\Delta_1 &= (n+1)\epsilon \\
\Delta_2 &= \Delta_1(1+(n+1)\epsilon) + (n+1)\epsilon = 2(n+1)\epsilon + O(\epsilon^2) \\
\Delta_3 &= \Delta_2(1+(n+1)\epsilon) + (n+1)\epsilon = 3(n+1)\epsilon + O(\epsilon^2) \\
&\vdots \\
\Delta_{n-1} &= (n^2-1)\epsilon + O(\epsilon^2) \\
\Delta_n &= n(n+1)\epsilon + O(\epsilon^2)
\end{aligned}$$

therefore $\hat{t}_j - t_j = t_j\Delta_j$ where $|\Delta_j| \leq j(n+1)u$

$$|\hat{t}_j - t_j| \leq j(n+1)u|t_j| \leq (n^2+n)u|t_j| \tag{3.29}$$

which suggests that the relative error in determining the columns of T is small and well-bounded.

Consider the absolute error in column $\hat{t}_{j+1}$

$$\begin{aligned}\hat{t}_{j+1} - t_{j+1} &= t_{j+1}(j+1)(n+1)\epsilon \qquad + O(\epsilon^2) \\ &= Xt_j(j+1)(n+1)\epsilon \qquad + O(\epsilon^2) \\ &= X^{j+1}t_0(j+1)(n+1)\epsilon \qquad + O(\epsilon^2)\end{aligned}$$

$$|\hat{t}_{j+1} - t_{j+1}| \leq ||X||^{j+1}(nj+n+j+1)u$$

This shows that the rounding error in the columns of $\hat{T}$ can become very large and therefore lead to dependencies between the columns. Consequently $\hat{T}$ may be nearly singular, possessing at least one very small eigenvalue. In this case $\hat{T}$ will be ill-conditioned and Gaussian Elimination is not guaranteed to provide an accurate solution.

However, if T is well-conditioned then we solve the system equivalent to (3.24),

$$(T + E_1)a = b \qquad ||E_1|| \leq 2n^2u||T|| \tag{3.30}$$

and Gaussian Elimination provides a solution $\hat{a}$ satisfying

$$(T + E_2)\hat{a} = b \qquad ||E_2|| \leq (8n^3\rho + 2n^2)u||T|| \tag{3.31}$$

A necessary condition for T to be well-conditioned is that X must possess well-distributed eigenvalues [Wilkinson].

SECTION 3.8: Characteristic Polynomial of Matrices with Distinct Eigenvalues

Let the characteristic equation of X be

$$f(\lambda) = \lambda^n + a_1\lambda^{n-1} + \ldots + a_n = 0 \tag{3.32}$$

Substitute λ_i, the eigenvalues of X, into (3.32),

$$\begin{aligned}
a_1\lambda_1^{n-1} + a_2\lambda_1^{n-2} + \ldots + a_n &= -\lambda_1^n \\
a_1\lambda_2^{n-1} + a_2\lambda_2^{n-2} + \ldots + a_n &= -\lambda_2^n \\
&\vdots \\
a_1\lambda_n^{n-1} + a_2\lambda_n^{n-2} + \ldots + a_n &= -\lambda_n^n
\end{aligned}$$

Let $\quad a = (a_1, a_2, \ldots, a_n)^T$

and $\quad \lambda^n = (-\lambda_1^n, -\lambda_2^n, \ldots, -\lambda_n^n)^T$

Then the above system of simultaneous equations may be written as

$$Pa = \lambda^n \tag{3.33}$$

with

$$P = \begin{bmatrix} \lambda_1^{n-1} & \lambda_1^{n-2} & \ldots & \lambda_1 & 1 \\ \vdots & & & & \\ \lambda_n^{n-1} & \lambda_n^{n-2} & \ldots & \lambda_n & 1 \end{bmatrix}$$

Since the λ_i are distinct, P will be non-singular and the unique solution of (3.33) will be the coefficients of the characteristic polynomial of X.

From earlier discussions the problem (3.33) is ill-conditioned if the condition number of P, that is $||P^{-1}||\,||P||$, is large. Two possible sources of ill-conditioning are,

(i) any two eigenvalues are nearly equal,

(ii) the eigenvalues are poorly distributed.

Rounding errors in the computed $\hat{\lambda}_i$ may cause (i) to hold.

It is known [Gautschi], that for general points λ_i the condition of P with respect to the problem (3.33) is large. An immediate implication of this is that no algorithm (whether stable or not) will accurately solve (3.33). However, it is also known [Wilkinson], that if the eigenvalues of X are well-distributed then P will be well-conditioned. (3.33) can obviously be solved by Gaussian Elimination but a much quicker technique is in existence. This is due to [Bjorck & Pereya] and is implemented as follows:

interchange rows of P such that

$$\tilde{P} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ \vdots & \vdots & & & \\ 1 & \lambda_n & \lambda_n^2 & \dots & \lambda_n^{n-1} \end{bmatrix} = V^T$$

where V is The Vandermonde matrix.
Let $b = (b_1, \dots, b_n)^T$
where $a_i = b_{n+1-i}, \qquad i = 1, \dots, n$

The problem of solving (3.33) now becomes one of solving

$$V^T b = \boldsymbol{\lambda}^n \qquad \text{for} \quad b \tag{3.34}$$

This is the 'dual problem' and is equivalent to polynomial interpolation [Bjorck & Pereya]. This follows because if

$$V^T b = \boldsymbol{\lambda}^n \quad \text{and} \quad f(\lambda) = \sum_{j=1}^{n} b_j \lambda^{j-1}$$

then

$$f(\lambda_i) = -\lambda_i^n \qquad \text{for} \quad i = 1, \dots, n$$

The first step in computing the b_j is to calculate the Newton representation,

$$f(\lambda) = \sum_{k=1}^{n} c_k \prod_{i=1}^{k-1} (\lambda - \lambda_i)$$

The constants c_k are divided differences and may be determined as follows,

$$c_i^{(1)} = -\lambda_i^n \qquad \text{for} \quad i = 1, \dots, n \tag{3.35}$$

then for $k = 1, \dots, n-1$

for $i = n, \dots, k+1$

$$c_i^{(k+1)} = (c_i^{(k)} - c_{i-1}^{(k)})/(\lambda_i - \lambda_{i-k}) \tag{3.36}$$

The next step is to generate the b_i from the c_i. Define polynomials $g_n(\lambda), \dots, g_0(\lambda)$ by the iteration

$$g_n(\lambda) = c_n$$

For $\quad k = n-1, 1$

$$g_k(\lambda) = c_k + (\lambda - \lambda_k) g_{k+1}(\lambda) \tag{3.37}$$

and observe that $g_0(\lambda) = f(\lambda)$. Writing

$$g_k(\lambda) = b_k^{(k)}\lambda^{n-k} + b_{k+1}^{(k)}\lambda^{n-k-1} + \ldots + b_n^{(k)} \tag{3.38}$$

and equating like powers of λ in the equations (3.37) and (3.38) gives the following recursion for the coefficients $b_i^{(k)}$,

$$b_n^{(n)} = c_n$$

For $k = n-1, \ldots, 1$

$$b_k^{(k)} = c_k - \lambda_k b_{k+1}^{(k+1)}$$

For $i = k+1, \ldots, n-1$

$$b_i^{(k)} = b_i^{(k+1)} - \lambda_k b_{i+1}^{(k+1)}$$

$$b_n^{(k)} = b_n^{(k+1)}$$

Consequently the coefficients $b_i = b_i^{(0)}$ can be calculated as follows:

$$b_i^{(n)} = c_i^{(n)} \qquad (i = 1, n) \tag{3.39}$$

For $k = n-1, \ldots, 1$

For $i = k, \ldots, n-1$ (3.40)

$$b_i^{(k)} = b_i^{(k+1)} - \lambda_k b_{i+1}^{(k+1)}$$

The operations count for this interpolation procedure is of the order $n^2 - n$. The majority of the work is involved in determining the eigenvalues of X. Therefore the overall operations count is $8n^3 + n^2$ and the storage requirements are $n^2 + 2n$ locations.

Error Analysis

Now we take a detailed look at the error analysis associated with implementing this algorithm on a computer using floating point arithmetic.

The eigenvalues are determined by the QR algorithm and satisfy

$$\hat{\lambda}y = (X + E)y, \qquad E = X\delta \quad \text{with} \quad |\delta| \leq u$$

Since $Xy = \lambda y$

$$\hat{\lambda}y = \lambda y + X\,\delta y = \lambda y(1 + \delta)$$

Taking norms

$$|\hat{\lambda} - \lambda| \leq u|\lambda|$$

From (3.35)

$$\hat{c}_i^{(1)} = fl(-\lambda_i^n)$$

$$= -fl\left(\prod_1^n \hat{\lambda}_i\right) = -(1+\delta)^n \; fl\left(\prod_1^n \lambda_i\right)$$

using a result from Section (1.3),

$$\hat{c}_i^{(1)} = c_i^{(1)}(1+(2n-1)\epsilon) + O(\epsilon^2) \qquad \text{where} \quad |\epsilon| \leq u$$

Now consider (3.36),

$$\hat{c}_i^{(k+1)} = fl\left[\frac{fl(\hat{c}_i^{(k)} - \hat{c}_{i-1}^{(k)})}{fl(\hat{\lambda}_i - \hat{\lambda}_{i-k})}\right]$$

$$= fl\left[\frac{fl(\hat{c}_i^{(k)} - \hat{c}_{i-1}^{(k)})}{\lambda_i - \lambda_{i-k}}\right] \quad (1+2\epsilon) + O(\epsilon^2)$$

when $k=1$, $\hat{c}_i^{(2)} = fl\left[\dfrac{fl(\hat{c}_i^{(1)} - \hat{c}_{i-1}^{(1)})}{\lambda_i - \lambda_{i-1}}\right] \quad (1+2\epsilon) + O(\epsilon^2)$

$$= fl\left[\frac{fl(c_i^{(1)} - c_{i-1}^{(1)})}{\lambda_i - \lambda_{i-1}}\right] \quad (1+2\epsilon)(1+(2n-1)\epsilon) + O(\epsilon^2)$$

$$= \left[\frac{c_i^{(1)} - c_{i-1}^{(1)}}{\lambda_i - \lambda_{i-1}}\right] \quad (1+\epsilon)(1+\epsilon)(1+2\epsilon)(1+(2n-1)\epsilon) + O(\epsilon^2)$$

$$\hat{c}_i^{(2)} = c_i^{(2)}(1+(2n+3)\epsilon) + O(\epsilon^2), \qquad |\epsilon| \leq u$$

similarly $\quad \hat{c}_i^{(3)} = c_i^{(3)}(1+(2n+7)\epsilon) + O(\epsilon^2), \qquad |\epsilon| \leq u$

$$\vdots$$

$$\hat{c}_i^{(r)} = c_i^{(r)}(1+(2n+4r-5)\epsilon) + O(\epsilon^2) \tag{3.41}$$

$$\text{for} \quad i = r, \ldots, n$$

(3.41) gives the error in computing $\hat{c}_i^{(r)}$.

The following result is useful in the error-analysis for (3.40),

$$\hat{y} = fl(\hat{x}_1 + \hat{a}\hat{x}_2) \qquad \text{where} \qquad \hat{x}_1 = x_1(1 + m_1\epsilon)$$

$$\hat{x}_2 = x_2(1 + m_2\epsilon)$$

$$\hat{a} = a(1 + \epsilon) \qquad \text{and} \quad |\epsilon| \leq u$$

Then

$$\hat{y} = fl(x_1(1 + m_1\epsilon) + a(1 + \epsilon)x_2(1 + m_2\epsilon)(1 + \epsilon))$$

$$= fl(x_1(1 + m_1\epsilon) + ax_2(1 + (m_2 + 2)\epsilon)) + O(\epsilon^2)$$

We would like m_1 and $m_2 + 2$ to be equal. This is achieved by multiplying the respective term the appropriate number of times by 1 so that

$$\hat{y} = (x_1 + ax_2)(1 + d\epsilon) + O(\epsilon^2) \tag{3.42}$$

$$\hat{y} = y(1 + d\epsilon)$$

where $d = \max\{m_1,\ m_2 + 2\}$
Now consider (3.40),
$\underline{k = n - 1}$

$$\hat{b}_{n-1}^{(n-1)} = fl\left[\hat{b}_{n-1}^{(n)} - \hat{\lambda}_{n-1}\hat{b}_n^{(n)}\right]$$

using (3.41),

$$\hat{b}_{n-1}^{(n-1)} = b_{n-1}^{(n)}(1 + (6n - 4)\epsilon) - \lambda_{n-1}b_n^{(n)}(1 + (6n - 2)\epsilon) + O(\epsilon^2)$$

from (3.42),

$$\hat{b}_{n-1}^{(n-1)} = b_{n-1}^{(n-1)}(1 + (6n - 2)\epsilon) + O(\epsilon^2)$$

$\underline{k = n - 2}$ Similarly,

$$\hat{b}_{n-2}^{(n-2)} = b_{n-2}^{(n-2)}(1 + (6n + 1)\epsilon) + O(\epsilon^2)$$

and

$$\hat{b}_{n-1}^{(n-2)} = b_{n-1}^{(n-2)}(1 + (6n - 1)\epsilon) + O(\epsilon^2)$$

Continuing in this way we obtain the following general relation,

$$\hat{b}_j^{(i)} = b_j^{(i)}(1 + p_{ij}\epsilon) + O(\epsilon^2) \tag{3.43}$$

where the p_{ij} are given as follows,

j							
1					$9n-9$	$9n-8$	
2					$9n-11$	$9n-10$	
$\vdots$					$\vdots$	$\vdots$	
$n-3$			$6n+4$		$\vdots$	$\vdots$	
$n-2$		$6n+1$	$6n+2$		$\vdots$	$\vdots$	
$n-1$	$6n-2$	$6n-1$	$6n$		$\vdots$	$\vdots$	
	$n-1$	$n-2$	$n-3$	...	2	1	i

Generally,

$$p_{ij} = 9n - 5 - i - 2j \tag{3.44}$$

The coefficients $\hat{a}_k$ are then given by

$$\hat{a}_k = \hat{b}^{(1)}_{n+1-k}, \qquad k = 1, \ldots, n$$

Using (3.42) and (3.44),

$$\hat{a}_k = b^{(1)}_{n+1-k}(1 + (7n - 8 + 2k)\epsilon) + O(\epsilon^2) \qquad |\epsilon| \leq u$$

$$\hat{a}_k - a_k = a_k \quad (7n - 8 + 2k)\epsilon + O(\epsilon^2)$$

Taking norms over all k

$$|\hat{a} - a| \leq |a| 9nu \qquad \text{for any norm}$$

so that the relative error incurred in computing the a_i using this interpolation algorithm is

$$\frac{|\hat{a} - a|}{|a|} \leq 9nu \tag{3.45}$$

which is well-bounded and very small.

In summary, if the eigenvalues of X are well-distributed then the matrix P is well-conditioned and the Interpolation algorithm will yield a very accurate solution, such that the computed coefficients of X are in fact the exact coefficients of a matrix $X + E$, near to X where E is bounded by

$$||E|| \leq 9nu||X||$$

However, if the matrix P is ill-conditioned then there is no guarantee that the method will provide an accurate solution. There are situations where this method does give accurate solutions to an ill-conditioned P [Higham, 1].

Therefore, since the overall method may give rise to an ill-conditioned matrix P, it must be considered as unstable. [Higham, 1] presents an error analysis for the Bjorck-Pereya algorithm for when the λ_i are non-negative and arranged in increasing order. It is shown that for a particular class of Vandermonde problems, the error bound obtained depends on the dimension n and on the machine precision only, being independent of the condition number of the coefficient matrix.

[Higham, 2] develops algorithms for solving (3.34) for when the points in the Vandermonde's structure are polynomials that satisfy a three term recurrence relation.

SECTION 3.9: Conclusions

It is evident from the discussions and analyses of this chapter, why the stable LeVerriers algorithm is so widely used. It is a stable algorithm which will always yield accurate solutions. The only disadvantage it appears to have is the number of arithmetic operations it must perform.

The original implementation of LeVerriers algorithm will always give rise to a potentially ill-conditioned problem and is therefore inappropriate in any circumstance.

The reduction of the original matrix, by Danilevski's method, to its upper Hessenberg form uses stable orthogonal similarity transformations. The reduction of the Hessenberg form to companion form can use only elementary similarity transformations which are not always stable. In the reduction, a small sub-diagonal pivotal element may cause drastic changes to the elements in that column. If this method is used then any good algorithm should detect this condition and terminate processing with an error code. Clearly Danilevski's method is not appropriate for derogatory matrices. However, it is not always possible to determine beforehand whether a matrix is derogatory or not, and if Danilevski's method is used then rounding errors may accumulate to produce an ill-conditioned Hessenberg form. The processing will eventually terminate with an error-code and it will appear on the surface, that a small sub-diagonal pivotal element has been detected for a non-derogatory matrix.

The Block Frobenius method is potentially a very good one in the sense that for a well-conditioned matrix it will determine the solutions accurately and very quickly. The reference to well-conditioning is related to the number of Frobenius blocks that arise on the diagonal of the reduced matrix. The method works on any type of matrix, the only disadvantage being a possible loss of accuracy in mid-ranged coefficients of the characteristic polynomial, for large order matrices.

Krylov's method has been discussed extensively in the literature and is commonly regarded as being unworkable. We have seen why this pessimism is attached to the method, firstly in that it is valid for non-derogatory matrices only and secondly, it involves the solution of a linear system in which the matrix is generally ill-conditioned. It is known, however, that this matrix is well-conditioned when the eigenvalues of the original matrix are well-distributed. In this case, Krylov's method yields accurate solutions and involves very few arithmetic operations.

The Interpolation method has a number of similarities to Krylov's method. It is valid only for those matrices possessing distinct eigenvalues and the method involves the solution of a linear system in which the matrix is a Vandermonde matrix. This matrix is generally ill-conditioned but if the original matrix has well-distributed computed eigenvalues then the Vandermonde matrix is well-conditioned. In this case, an Interpolation approach rather

than Gaussian Elimination is used to solve the linear system. This approach makes use of the structure of the Vandermonde matrix to yield very accurate solutions very rapidly.

From the discussions above, the best general purpose algorithm to provide accurate solutions for all problems is the LeVerriers method. For problems of order less than 50, the Block Frobenius is a better alternative since it will be considerably faster with very little loss of accuracy in the solutions. Solely on the basis of speed, the quickest method is Krylov's, under the conditions discussed above. From the accuracy point of view, the errors involved in the Interpolation method are very small when compared with the 'best algorithm' case in section 3.1.

CHAPTER 4 – THE ELIMINATION METHOD WITH APPLICATIONS

SECTION 4.1 : Introduction

The Elimination method is a technique that generates an expression that gives the solution X to the quadratic matrix equation, in terms of the coefficient matrices P and Q and the coefficients of the characteristic polynomial (c.c.p) of X. The expression is of the form $RX = -S$ where the matrices R and S are generated recursively.

We derive alternative explicit representations for R and S, that are related to the associated quadratic eigenvalue problem. Further analysis leads to a relationship between the conditioning of the original problem and the conditioning of matrix R. This is not an 'if and only if' condition and an example illustrates that R may be ill-conditioned even when the original problem is not.

An error analysis of the Elimination method reveals that the rounding errors generated as a result of computing R and S are small and well bounded such that the accuracy of the solution of the matrix equation $RX = -S$ is dependent on the conditioning of R.

Section 4.3 describes an iterative Elimination method based algorithm for computing the solution to the quadratic matrix equation. The problem of finding a suitable starting point to the iterations is discussed and a heuristic formula is suggested. An analysis of the stopping criterion for the iterations shows it to be truely reflective of the accuracy of the computed solution. We conclude that if R is well-conditioned at each iteration and a stable method is used to determine the c.c.p of the current X then the iterations are stable. A discussion of convergence theory as applied to the iterative algorithm is included, mainly to provide a foundation for further analysis.

Section 4.4 discusses and applies the points of section 4.2 to the matrix square root problem.

Section 4.5 describes three methods for computing the square root of a matrix, all based on the Elimination method. Method 1 applies the Elimination method iteratively as section 4.3 does for the quadratic matrix equation and it is shown that the work done by the algorithm may be reduced considerably by transforming the coefficient matrix. Method 2 determines the c.c.p of X from its eigenvalues and uses these as input to the Elimination method. Method 3 derives a relationship between the c.c.p of X and those of the coefficient matrix. The relationship yields a system of n non-linear equations which are solved by a globally convergent algorithm to give the c.c.p of X. These coefficients are then used as input to the Elimination method. For each method, there is a discussion on the stability of the algorithm and the operations count involved in their implementations.

SECTION 4.2 : The Elimination Method

Consider the monic unilateral quadratic matrix equation

$$F(X) = X^2 + PX + Q = 0 \tag{4.1}$$

where P, Q, X are square matrices of order n. Let a_i $(i = 1, \ldots, n)$ be the c.c.p. of X, satisfying

$$\tilde{f}(\lambda) = |X - \lambda I| = (\lambda^n + a_1\lambda^{n-1} + \ldots + a_n)(-1)^n$$

The Cayley-Hamilton theorem states that every square matrix satisfies its own characteristic polynomial.

Therefore we have that

$$X^n + a_1X^{n-1} + a_2X^{n-2} + \ldots + a_nI = 0 \tag{4.2}$$

Postmultiply (4.1) by X^{n-2},

$$X^n + PX^{n-1} + QX^{n-2} = 0 \tag{4.3}$$

Subtract (4.3) from (4.2) to give

$$(a_1I - P)X^{n-1} + (a_2I - Q)X^{n-2} + a_3X^{n-3} + \ldots + a_nI = 0 \tag{4.4}$$

Let

$$R_1 = a_1I - P$$

$$S_1 = a_2I - Q$$

such that (4.4) may be written as

$$R_1X^{n-1} + S_1X^{n-2} + a_3X^{n-2} + \ldots + a_nI = 0 \tag{4.5}$$

Postmultiply (4.1) by X^{n-3} and premultiply by R_1

$$R_1X^{n-1} + R_1PX^{n-2} + R_1QX^{n-3} = 0 \tag{4.6}$$

Subtracting (4.6) from (4.5) eliminates the term in X^{n-1},

$$(S_1 - R_1P)X^{n-2} + (a_3I - R_1Q)X^{n-3} + a_4X^{n-4} + \ldots + a_nI = 0 \tag{4.7}$$

Let

$$R_2 = S_1 - R_1P$$

$$S_2 = a_3I - R_1Q$$

such that (4.7) may be written as

$$R_2X^{n-2} + S_2X^{n-3} + a_4X^{n-4} + \ldots + a_nI = 0$$

Continuing in this way, at the i^{th} stage we eliminate the term in X^{n-i} between the following equations,

$$R_iX^{n-i} + S_iX^{n-(i+1)} + a_{i+2}X^{n-(i+2)} + \ldots + a_nI = 0$$

$$R_iX^{n-i} + R_iPX^{n-(i+1)} + R_iQX^{n-(i+2)} = 0$$

giving

$$R_{i+1}X^{n-(i+1)} + S_{i+1}X^{n-(i+2)} + a_{i+3}X^{n-(i+3)} + \ldots + a_nI = 0 \tag{4.8}$$

where

$$R_{i+1} = S_i - R_iP$$

$$S_{i+1} = a_{i+2}I - R_iQ$$

At the $(n-2)^{th}$ stage, we have that

$$R_{n-1}X + S_{n-1} = 0 \tag{4.9}$$

where X solves (4.1).

What this gives us is a relationship between the coefficients of the characteristic polynomial of a matrix X satisfying (4.1) and the elements of X. This is known as the Elimination method [McDonald].

An Explicit Representation

From (4.8) it follows that

$$R_{i+1} = a_{i+1}I \bar{\#} R_{i-1}Q - R_iP$$

or

$$R_i = a_iI - R_{i-1}P - R_{i-2}Q$$

The equations may be transposed and expressed in the following way,

$$\begin{bmatrix} R_{i-1}^T \\ R_i^T \end{bmatrix} = \begin{bmatrix} 0_n & I_n \\ -Q^T & -P^T \end{bmatrix} \begin{bmatrix} R_{i-2}^T \\ R_{i-1}^T \end{bmatrix} + \begin{bmatrix} 0_n \\ a_iI_n \end{bmatrix}$$

a more convenient form being

$$c_i = A^Tc_{i-1} + b_i \tag{4.10}$$

where

$$c_i = \begin{bmatrix} R_{i-1}^T \\ R_i^T \end{bmatrix} \text{ with } c_0 = \begin{bmatrix} 0_n \\ I_n \end{bmatrix}, \quad b_i = \begin{bmatrix} 0_n \\ a_i I_n \end{bmatrix} \text{ with } b_0 = \begin{bmatrix} 0_n \\ I_n \end{bmatrix}$$

(4.10) is a first order difference equation. An explicit solution is given by

$$c_i = \sum_{k=0}^{i} (A^T)^k b_{i-k}$$

Transposing both sides and substituting for c_i and b_i gives,

$$[R_{i-1} \quad R_i] = [0_n \quad I_n] \sum_{k=0}^{i} a_{i-k} A^k \tag{4.11}$$

where $A = \begin{bmatrix} 0 & -Q \\ I & -P \end{bmatrix}$ is the companion matrix of the corresponding quadratic eigenvalue problem. Therefore

$$R_{n-2} = [0_n \quad I_n] \sum_{k=0}^{n-1} a_{n-k-1} A^k \begin{bmatrix} I_n \\ 0_n \end{bmatrix} \tag{4.12}$$

and

$$R_{n-1} = [0_n \quad I_n] \sum_{k=0}^{n-1} a_{n-k-1} A^k \begin{bmatrix} 0_n \\ I_n \end{bmatrix} \tag{4.13}$$

Let

$$T \in \mathbb{R}^{2n \times 2n} = f(A) = \sum_{k=0}^{n-1} a_{n-k-1} A^k \tag{4.14}$$

and partition T as

$$T = \left[\begin{array}{c|c} T_{11} & T_{12} \\ \hline R_{n-2} & R_{n-1} \end{array}\right]$$

Let $$Z \in \mathbb{R}^{2n \times 2n} = \begin{bmatrix} 0 & -Q \\ I & X \end{bmatrix}$$

then $$Z^{-1} = \begin{bmatrix} XQ^{-1} & I \\ -Q^{-1} & 0 \end{bmatrix}$$

and $$\begin{bmatrix} 0 & -Q \\ I & -P \end{bmatrix} \begin{bmatrix} 0 & -Q \\ I & X \end{bmatrix} = \begin{bmatrix} 0 & -Q \\ I & X \end{bmatrix} \begin{bmatrix} -(X+P) & 0 \\ I & X \end{bmatrix}$$

$$AZ = Z\ B$$

$$A = Z\ B\ Z^{-1} \tag{4.15}$$

A is similar to a matrix B which has as its eigenvalues, the union of the eigenvalues of $-(X+P)$ and X. We would expect this, since from Section 1.6, A is the companion matrix of the associated quadratic eigenvalue problem. Substitute (4.15) into (4.14) to give,

$$\begin{aligned} f(A) &= \sum_{k=0}^{n-1} a_{n-k-1}\,(ZBZ^{-1})^k \\ &= \sum_{k=0}^{n-1} a_{n-k-1}\; Z\; B^k\; Z^{-1} \\ &= Z\; \sum_{k=0}^{n-1} a_{n-k-1}\; B^k\; Z^{-1} \\ &= Z\; f(B)\; Z^{-1} \end{aligned} \tag{4.16}$$

Given the similarity relationship in (4.15), (4.16) is a standard result concerning functions of matrices [Lancaster & Tismentsky]. Two other results which we use here are

$$f(B)B = Bf(B) \tag{4.17}$$

and if

$$B = \text{ diag } (B_1, B_2, \ldots, B_r)$$

then

$$f(B) = \text{ diag } (f(B_1), f(B_2), \ldots, f(B_r)) \tag{4.18}$$

(4.18) suggests that the form of $f(B)$ is,

$$\begin{bmatrix} f(-(X+P)) & 0 \\ f\alpha & f(X) \end{bmatrix} \tag{4.19}$$

for unknown $f\alpha$.

Substitute this into (4.17),

$$\begin{bmatrix} f(-(X+P)) & 0 \\ f\alpha & f(X) \end{bmatrix} \begin{bmatrix} -(X+P) & 0 \\ I & X \end{bmatrix} =$$

$$\begin{bmatrix} -(X+P) & 0 \\ I & X \end{bmatrix} \begin{bmatrix} f(-(X+P)) & 0 \\ f\alpha & f(X) \end{bmatrix}$$

Expanding both sides,

$$\begin{bmatrix} f(-(X+P))(-(X+P)) & 0 \\ -f\alpha(X+P)+f(X) & f(X)X \end{bmatrix} = \begin{bmatrix} -(X+P)f(-(X+P)) & 0 \\ f(-(X+P))+Xf\alpha & Xf(X) \end{bmatrix}$$

Comparing coefficients and using (4.17) gives

$$\begin{aligned} Xf\alpha + f\alpha(X+P) + f(-(X+P)) - f(X) &= 0 \\ Xf\alpha + f\alpha(X+P) &= f(X) - f(-(X+P)) \end{aligned} \tag{4.20}$$

Substitute (4.19) into (4.16) $f(A) =$

$$\begin{bmatrix} 0 & -Q \\ I & X \end{bmatrix} \begin{bmatrix} f(-(X+P)) & 0 \\ f\alpha & f(X) \end{bmatrix} \begin{bmatrix} XQ^{-1} & I \\ -Q^{-1} & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -Q \\ I & X \end{bmatrix} \begin{bmatrix} f(-(X+P))XQ^{-1} & f(-(X+P)) \\ f\alpha XQ^{-1} - f(X)Q^{-1} & f\alpha \end{bmatrix}$$

$$= \begin{bmatrix} -Qf\alpha XQ^{-1} + Qf(X)Q^{-1} & -Qf\alpha \\ f(-(X+P))XQ^{-1} + Xf\alpha XQ^{-1} - Xf(X)Q^{-1} & f(-(X+P)) + Xf\alpha \end{bmatrix}$$

Using (4.20)

$$= \begin{bmatrix} Q(f(X) - f\alpha X)Q^{-1} & -Qf\alpha \\ f\alpha & f(-(X+P)) + Xf\alpha \end{bmatrix} \tag{4.21}$$

From the earlier definition, $f(A) = T = \left[\begin{array}{c|c} T_{11} & T_{12} \\ \hline R_{n-2} & R_{n-1} \end{array}\right]$

$$\begin{aligned} R_{n-2} &= f\alpha \\ R_{n-1} &= f(-(X+P)) + Xf\alpha = f(X) - f\alpha(X+P) \end{aligned} \tag{4.22}$$

where $f\alpha$ is given by (4.20). (4.22) gives us an explicit expression for R_{n-1}, used in the next section.

We can show that (4.22) is the correct expression for R_{n-1} since,

$$f(X) = a_{n-1}I + a_{n-2}X + \ldots + a_1X^{n-2} + X^{n-1}$$

$$X.f(X) = a_{n-1}X + a_{n-2}X^2 + \ldots + a_1X^{n-1} + X^n$$

and since a_i are the c.c.p. of X and X satisfies its own characteristic polynomial,

$$f(X) \cdot X = -a_nI \tag{4.23}$$

substituting this into (4.22) gives

$$R_{n-1} = -a_n X^{-1} - f\alpha(X+P)$$

$$\begin{aligned} R_{n-1}X &= -a_n I - R_{n-2}(X+P)X \\ &= -a_n I - R_{n-2}(X^2 + PX) \\ &= -(a_n I - R_{n-2}Q) \\ &= -S_{n-1} \qquad \text{as required} \end{aligned}$$

Condition of R_{n-1}

We must firstly look at the effect of perturbations in the initial data, P and a_i, to the solution $f\alpha$ of (4.20). This is a Sylvester equation and the results of section 1.4.3 on the conditioning of this type of equation may be applied here. Let the error in P be ΔP with $\| \Delta P \| \leq u \| P \|$, then the solution $\hat{f}\alpha$, to (4.20) satisfies

$$\| \hat{f}\alpha - f\alpha \| = \| \Delta f\alpha \| \leq 4 \| \Delta P \| \, \| G^{-1} \| \, \| f\alpha \| \tag{4.24}$$

where G solves

$$G \text{ vec } (f\alpha) = \text{ vec } [f(X) - f(-(X+P))]$$

Let the perturbed coefficient a_i be such that

$$\hat{a}_i = a_i(1+\varepsilon) \quad , \quad |\varepsilon| \leq u.$$

Since

$$f(X) = -a_n X^{-1}$$

then

$$\hat{f}(X) = -a_n(1+\varepsilon)X^{-1} = f(X)(1+\varepsilon) \tag{4.25}$$

From (4.22)

$$\begin{aligned} \hat{R}_{n-1} &= \hat{f}(X) - \hat{f}\alpha(X+P+\Delta P) \\ &= f(X) + \varepsilon f(X) - f\alpha(X+P) - f\alpha\Delta P - \Delta f\alpha(X+P+\Delta P) \end{aligned}$$

Taking norms

$$\| \hat{R}_{n-1} - R_{n-1} \| \leq u \| f(X) \| + u \| P \| \, \| f\alpha \| + 4u \| P \| \, \| X+P \| \, \| G^{-1} \| \, \| f\alpha \|$$

now,

$$\| \text{vec } (f\alpha) \| = \| f\alpha \| \leq \| G^{-1} \| \, \| c \|$$

where

$$c = \text{vec } [\| f(X) - f(-X - P) \|]$$

Finally

$$\frac{\| \hat{R}_{n-1} - R_{n-1} \|}{\| R_{n-1} \|} \leq \frac{u|a_n| \, \| X^{-1} \| + u \| G^{-1} \| A_1 + u \| G^{-1} \|^2 A_2}{\| R_{n-1} \|} \tag{4.26}$$

and recalling our definition of the derivative of $F(X)$ in section 2.1, we observe that G and the matrix T of that section possess the same eigenvalues and

$$\|G\| = \|T\| = \|F'(X)\|.$$

Notice that as X approaches singularity, its determinant approaches zero so that at singularity (4.26) is still defined with $|a_n| = 0$.

For non-singular X, Rank $(X) = n$ and

$$\text{Rank } (R_{n-1}) = \min \{\text{Rank } (S_{n-1}), \text{ Rank } (X^{-1})\}$$

$$\therefore \text{Rank } (R_{n-1}) = \text{Rank } (a_n I - f\alpha Q) \leq n$$

with equality implying the existence of a unique matrix X, otherwise there is an infinite number of solutions of $RX = -S$ for this particular set of a_i.

Singularity may be considered as an extreme form of ill-conditioning. However ill-conditioning does not necessarily imply the existence of a small eigenvalue.

The following example illustrates the case of a singular R_{n-1} when the derivative is non-singular and well-conditioned.

$$X^2 + \begin{bmatrix} -1 & -6 \\ 2 & -9 \end{bmatrix} X + \begin{bmatrix} 0 & 12 \\ -2 & 14 \end{bmatrix} = 0$$

The eigenvalues of the associated quadratic eigenvalue problem are 1,2,3,4. If we select $\lambda = 1, 2$ as possible eigenvalues for X, then

$$| X - \lambda I | = \lambda^2 - 3\lambda + 2 \Rightarrow a_1 = -3, \; a_2 = 2$$

$$R_1 X = -S_1 \Rightarrow \begin{bmatrix} 2 & -6 \\ 2 & -6 \end{bmatrix} X = \begin{bmatrix} 2 & -12 \\ 2 & -12 \end{bmatrix}$$

Then

$$X = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \text{ and } F'(X) = \begin{pmatrix} 1 & 0 & -6 & 0 \\ 0 & 2 & 0 & -6 \\ 2 & 0 & -6 & 0 \\ 0 & 2 & 0 & -5 \end{pmatrix}$$

where R is singular and $F'(X)$ is non-singular and well-conditioned.

Error Analysis

We now investigate the effect of rounding errors in computing R_{n-1} and S_{n-1}. It is not necessary to compute all the elements of the $2n \times 2n$ matrix A^k, since if A is partitioned as

$$A^k = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \quad , \quad A_{ij}^{(k)} \in \mathbb{R}^{n \times n} \tag{4.27}$$

then A^{k+1} is

$$\begin{aligned} \begin{bmatrix} A_{11}^{(k+1)} & A_{12}^{(k+1)} \\ A_{21}^{(k+1)} & A_{22}^{(k+1)} \end{bmatrix} &= \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \begin{bmatrix} 0 & -Q \\ I & -P \end{bmatrix} \\ &= \begin{bmatrix} A_{12}^{(k)} & A_{11}^{(k)}Q - A_{12}^{(k)}P \\ A_{22}^{(k)} & A_{21}^{(k)}Q - A_{22}^{(k)}P \end{bmatrix} \end{aligned} \tag{4.28}$$

and from our definitions for R_{n-1} and R_{n-2}, (4.12) and (4.13), we require only the bottom blocks,

$$\begin{bmatrix} A_{21}^{(k+1)} & A_{22}^{(k+1)} \end{bmatrix} = \begin{bmatrix} A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \cdot A \tag{4.29}$$

and

$$\begin{bmatrix} R_{n-2} & R_{n-1} \end{bmatrix} = \sum_{0}^{n-1} a_{n-k-1} \begin{bmatrix} A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \tag{4.30}$$

We can write (4.29) as

$$B^{(k+1)} = B^{(k)} A \text{ with } B^{(0)} = \begin{bmatrix} 0_n & I_n \end{bmatrix} \tag{4.31}$$

The following algorithm determines R_{n-1} and R_{n-2} (hence S_{n-1}):

$k = 0$ until $k = n - 1$

Form $B^{(k)} = \left[A_{21}^{(k)} \quad A_{22}^{(k)} \right]$ using (4.31)

$$R_{n-2} = R_{n-2} + a_{n-k-1}\ A_{21}^{(k)}$$

$$R_{n-1} = R_{n-1} + a_{n-k-1}\ A_{22}^{(k)}$$

Next k

$$S_{n-1} = a_n I - R_{n-2} Q$$

For ease of notation the following analysis is for the transposed system $B_{K+1}^T = A^T B_K^T$ and is equivalent to that for 4.31.

Let the errors in P and Q be of machine precision u and the relative error in the coefficients of the characteristic polynomial of X be δa. Ignoring terms of $O(u^2)$,

$$\begin{aligned}
\hat{B}^{(k)} &= f\ell\ \left(\hat{A}\hat{B}^{(k-1)}\right) && |\varepsilon| \le u \\
&= A(1+\varepsilon)(B^{(k-1)} + \Delta B^{(k-1)})(1+n_1\varepsilon) && n_1 = 2n \\
&= (A + A\varepsilon + An_1\varepsilon)(B^{(k-1)} + \Delta B^{(k-1)}) + O(\varepsilon^2) \\
&= AB^{(k-1)} + AB^{(k-1)}(1+n_1)\varepsilon + A\Delta B^{(k-1)} \\
\Delta B^{(k)} &= A\Delta B^{(k-1)} + AB^{(k-1)}(1+n_1)\varepsilon
\end{aligned}$$

since $B^{(k)} = A^k B^{(0)}$

$$\begin{aligned}
\Delta B^{(k)} &= A\left(A\Delta B^{(k)} + A^{k-1}B^{(0)}(1+n_1)\varepsilon\right) + A^k B^{(0)}(a+n_1)\varepsilon \\
&= A^2\Delta B^{(k)} + 2A^k B^{(0)}(1+n_1)\varepsilon
\end{aligned}$$

Continuing in this way,

$$\begin{aligned}
\Delta B^{(k)} &= A^k B^{(0)} + kA^k B^0(1+n_1)\varepsilon \\
&= A^k B^{(0)}\,(1 + k(1+n_1)\varepsilon) = B^{(k)}(1 + k(n_1+1)\varepsilon).
\end{aligned}$$

Now form $R_i^{(k)}$,

$$\begin{aligned}
\hat{R}_i^{(k)} &= f\ell\left[a_{i-k}(1+\delta a)\ \left(B^{(k)} + \Delta B^{(k)}\right)\right] \\
&= a_{i-k}(1+\delta a)B^{(k)}(1+k(1+n_1)\varepsilon)(1+\varepsilon) \\
&= a_{i-k}(1+\delta a)B^{(k)}(1+k_1\varepsilon) + O(\varepsilon^2)
\end{aligned}$$

where $k_1 = 2nk + k + 1$.

Finally,

$$\hat{R}_i = f\ell \left\{ \sum_0^i a_{i-k}(1+\delta a)\ B^{(k)}\ (1+k_1\varepsilon) \right\} \tag{4.32}$$

this is equivalent to computing $\hat{y}$ in section 1.3.3 where

$$\hat{y} = f\ell(x_1 + x_2 + \ldots + x_i)$$

A similar analysis on (4.32) gives

$$\hat{R}_i = \sum_0^i a_{i-k}(1+\delta a)B^{(k)}(1+nk\varepsilon + (i+2)\varepsilon) + O(\varepsilon^2)$$

$$= R_i + (i+2)\varepsilon(1+\delta a)R_i + \delta a R_i + n\varepsilon(1+\delta a)\sum_0^i ka_{i-k}B^{(k)}$$

at $i = n-2$

$$\hat{R}_{n-2} - R_{n-2} = (\varepsilon n(1+\delta a) + \delta a)\, R_{n-2} + n\varepsilon(1+\delta a)\sum_0^{n-2} ka_{n-k-2}B^{(k)}$$

Taking norms,

$$\|\hat{R}_{n-2} - R_{n-2}\| \leq u\, n\, (1+|\delta a|)\ \|R_{n-2}\| + |\delta a|\ \|R_{n-2}\ \|$$

$$+ nu\,(1+|\delta a|)\sum_0^{n-2} k\ |a_{n-k-2}|\ \|B^{(k)}\|$$

$$\frac{\|\hat{R}_{n-2} - R_{n-2}\|}{\|R_{n-2}\|} \leq |\delta a| + nu + \frac{nu}{\|R_{n-2}\|}(n-1)a_{\max}\|A^k\| + 0(u^2) \tag{4.33}$$

$$\text{where } a_{\max} = \max_i\ |a_i|$$

The bound in (4.33) implies that the rounding errors induced from computing $\hat{R}_{n-2}$ are of the order u. The effect of any errors in the c.c.p of X on $\hat{R}_{n-2}$ are of the same order as the errors in the initial coefficients. An immediate consequence of this is that the algorithm used to determine the initial coefficients must be accurate. Clearly this is an important observation, particularly with regard to those iterative methods of the next two Sections, that are based on the Elimination method.

From earlier

$$\hat{S}_{n-1} = f\ell(\hat{a}_n I - \hat{R}_{n-2}\hat{Q})$$

$$= (a_n(1+\delta a)I - \hat{R}_{n-2}Q(1+(n+1)\varepsilon))(1+\varepsilon) + 0(\varepsilon^2)$$

substituting $\hat{R}_{n-2}$ gives

$$\frac{\|\hat{S}_{n-1} - S_{n-1}\|}{\|S_{n-1}\|} \leq |a_n \delta a| + u + \frac{u\, n^2}{\|S_{n-1}\|} [\|R_{n-2}\| + a_{\max}\ \|A^n\|]\ \|Q\| + O(u^2) \qquad (4.34)$$

similarly, for R_{n-1}

$$\hat{R}_{n-1} = R_{n-1} + (n+1)\,\varepsilon\,(1+\delta a)\ R_{n-1} + \delta a\, R_{n-1} + n\,\varepsilon\ (1+\delta a)\ \sum_{0}^{n-1}\ k\ a_{n-k-1} B^{(k)}$$

$$\frac{\|\hat{R}_{n-1} - R_{n-1}\|}{\|R_{n-1}\|} \leq |\delta a| + u(n+1) + \frac{u\, n^2}{\|R_{n-1}\|}\ a_{\max}\ \|A^n\| + O(u^2) \qquad (4.35)$$

The comments pertaining to (4.33) are also valid here. The final step in the Elimination method is the solution of

$$R_{n-1}\ X = -S_{n-1}.$$

However, due to rounding errors the problem becomes

$$(R_{n-1} + \Delta R_{n-1})\hat{X} = -(S_{n-1} + \Delta S_{n-1})$$

Using the Gaussian Elimination method of Section 1.4, the computed solution satisfies

$$(R_{n-1} + \Delta R_{n-1} + E)\ \hat{X} = -(S_{n-1} + \Delta S_{n-1})$$

where

$$\|E\| \leq 80\ u\ \|R_{n-1} + \Delta R_{n-1}\| \leq 80\ u\ \|R_{n-1}\| + O(u^2)$$

so that in terms of exact R_{n-1} and S_{n-1}, we solve a system 'close' to the original one,

$$(R_{n-1} + E_1)\ \hat{X} = -(S_{n-1} + E_2) \qquad (4.36)$$

where $\|E_1\| \leq \|E\| + \|\Delta\, R_{n-1}\|$

and $\|E_2\| \leq \|E\| + \|\Delta\, S_{n-1}\|$

$\|\Delta\, R_{n-1}\|$ and $\|\Delta\, S_{n-1}\|$ given by (4.35) and (4.34) respectively.

Remarks

We have shown how the Elimination method may be used to determine the solution matrix of (4.1) given the coefficients of its charcteristic polynomial. The method is closely linked with the quadratic eigenvalue problem, via the companion matrix. In fact, the matrices R_{n-1} and S_{n-1} may be determined by the c.c.p of X and powers of the companion matrix. It has been shown that the problem of determining R_{n-1} may lead to difficulties and that a relationship between the condition of R_{n-1} and of the original problem (4.1) exists. Clearly it is necessary that R_{n-1} be well-conditioned for there to be a unique accurate solution $\hat{X}$, of the linear matrix equation.

The rounding errors generated by the algorithm that computes R_{n-1} and S_{n-1} are relatively small and well-bounded. The error due to the c.c.p of X are of the same order as the initial inaccuracies.

In summary, provided that $\hat{R}_{n-1}$ is well-conditioned, the Elimination method will accurately determine X of (4.1) given the c.c.p of X.

SECTION 4.3 : The Quadratic Matrix Equation

The Elimination method may be used in the following iterative scheme for determining a solution of (4.1):

(i) Select scalars a_i $(i = 1, \ldots, n)$ as initial estimates to the c.c.p of X.

(ii) Carry out one step of the Elimination method to determine R_{n-1} and S_{n-1}.

(iii) Solve $\quad R_{n-1}\, X^{(r+1)} = -S_{n-1}$.

(iv) Compute $\quad \dfrac{\|F(X^{(r+1)}) - F(X^{(r)})\|}{\|F(X^{(r)})\|}$

If this is less than some specified tolerance, then the iterations have converged.

(v) Compute the c.c.p of $X^{(r+1)}$. Go to (ii).

We know that when the derivative of $F(X)$ is ill-conditioned, the problem (4.1) is ill-conditioned and R_{n-1} may be ill-conditioned also. The impact of this is that at step (iii), it may not be possible to solve the linear system at all and when it can be solved, the computed solution would be unreliable. The example of the previous section illustrates also that R_{n-1} may be ill-conditioned even when $F'(X)$ is not. Consequently any good algorithm should monitor and report on the condition of R_{n-1} at each step of the iterative process and terminate with an error message if necessary.

Starting Point

The algorithm requires n values as initial estimates to the c.c.p of X. It is not possible to select values that will ensure convergence and it has been shown through examples [McDonald], that convergence may occur to the same solution from totally different starting points and that initial points near to each other may not converge to the same solution, if they converge at all.

The following heuristic procedure yields a formula for determining a set of initial estimates and has been justified through numerical experience.

Any non-derogatory matrix, X say, can be transformed into its companion form C say, by similarity transformations,

$$H^{-1}XH = C = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & & & & \vdots \\ -a_n & -a_{n-1} & -a_{n-2} & \ldots & -a_1 \end{bmatrix}$$

where a_i are the c.c.p of X.

Consider

$$X^2 + PX + Q = 0$$

$$H^{-1}XHH^{-1}XH + H^{-1}PHH^{-1}XH + H^{-1}QH = 0 \tag{4.37}$$

$$C^2 + \tilde{P}C + \tilde{Q} = 0$$

where$\tilde{P} = H^{-1}PH$ and $\tilde{Q} = H^{-1}QH$. Denote $\tilde{P} = (\tilde{p}_{ij})$ and $\tilde{Q} = \tilde{q}_{ij}$ then equating the elements in the bottom row of (4.37) gives,

$$\begin{array}{ccccccccccc}
a_1a_n & - & \tilde{p}_{nn}a_n & + & \tilde{q}_{n1} & & & & & = & 0 \\
-a_n & + & a_1a_{n-1} & + & \tilde{p}_{n1} & - & \tilde{p}_{nn}a_{n-1} & + & \tilde{q}_{n2} & = & 0 \\
\vdots & & & & & & & & & & \\
-a_{n-i} & + & a_1a_{n-i+1} & + & \tilde{p}_{n,i+1} & - & \tilde{p}_{nn}a_{n-i-1} & + & \tilde{q}_{n,i+2} & = & 0 \\
\vdots & & & & & & & & & & \\
-a_2 & + & a_1^2 & + & \tilde{p}_{n,n-1} & - & \tilde{p}_{nn}a_1 & + & \tilde{q}_{nn} & = & 0.
\end{array}$$

Since we are dealing with estimates, let

$p_{ij} = \text{sign } ((p_{ij})\|P\|_E$

and

$q_{ij} = \text{sign } (q_{ij})\|Q\|_E$

then we can write the above systems of equations in terms of the c.c.p of X,

$$\begin{aligned}
a_n &= -\frac{q_{n1}}{a_1 - p_{nn}} \\
a_{n-1} &= \frac{a_n - p_{n1} - q_{n2}}{a_1 - p_{nn}} \\
&\vdots \\
a_{n-i-1} &= \frac{a_{n-i} - p_{n,i+1} - q_{n,i+2}}{a_i - p_{nn}} \\
&\vdots \\
a_2 &= \frac{a_3 - p_{n,n-2} - q_{n,n-1}}{a_1 - p_{nn}}.
\end{aligned} \tag{4.38}$$

It only remains for us to determine an estimate for a_1.

Since

$$Q = -X^2 - PX$$

$$\|Q\| \leq \|X\|^2 + \|P\| \, \|X\|$$

Let $x = \|X\|$, $p = \|P\|$, $q = \|Q\|$. Then

$$-x^2 - px + q \leq 0 \tag{4.39}$$

We have already dealt with this type of inequality in Chapter 2 and a similar analysis gives

$$x_1 \leq -\frac{p}{2} - \frac{\sqrt{p^2+4q}}{2} \quad \text{or} \quad x_2 \geq -\frac{p}{2} + \frac{\sqrt{p^2+4q}}{2}$$

Since $x = \|X\| \geq 0$, we choose x^2 to satisfy (4.39).
Then

$$a_1 = \text{trace } (X) \leq \|X\| = \frac{-p + \sqrt{p^2+4q}}{2}$$

Therefore an estimate for a_1 is given by

$$\pm\left(\frac{-p + \sqrt{p^2+4q}}{2}\right)$$

with the sign chosen such that $a_i - p_{nn} \neq 0$ and the remaining a_i are generated by the formulae in (4.38).

Accuracy of Convergence Criterion

We need to specify, in the algorithm, when we consider convergence to have occurred. It is not sufficient to insist on $F(X^{(k+1)}) = 0$ exactly since due to rounding errors, exact zero may never be obtained. The quantity that is most often compared against is the relative decrease in the function defined by,

$$\frac{\|F(X^{(r+1)}) - F(X^{(r)})\|}{\|F(X^{(r)})\|} = FNORM, \quad \text{say}$$

computed at the r^{th} iteration.
For convergence to occur, $FNORM$ must be less than some absolute tolerance value, TOL say. We would like TOL to be as small as possible. This 'smallness' is dependent on the rounding incurred from computing $FNORM$, as follows:

$$F(X^{(r+1)}) = X^{(r+1)^2} + P\,X^{(r+1)} + Q$$

Due to the storage errors, the machine representations of X, P and Q have an error of $O(u)$. On computing $F(X^{(r+1)})$, rounding errors arise due to floating point arithmetic such that the computed matrix is given by,

$$\begin{aligned}
\hat{F}(X^{(r+1)}) &= f\ell\left\{X^{(r+1)^2}(1+\varepsilon)^2 + f\ell\left[PX^{(r+1)}(1+\varepsilon)^2 + Q(1+\varepsilon)\right]\right\} \\
&= f\ell\left\{X^{(r+1)^2}(1+\varepsilon)^2 + PX^{(r+1)}(1+\varepsilon)^3(1+n\varepsilon) + Q(1+\varepsilon)^2\right\} \\
&= X^{(r+1)^2}(1+n\varepsilon)(1+\varepsilon)^3 + PX^{(r+1)}(1+\varepsilon)^4(1+n\varepsilon) + Q(1+\varepsilon)^3 \\
\hat{F}(X^{(r+1)}) &= X^{(r+1)^2}(1+(n+3)\varepsilon) + PX^{(r+1)}(1+(n+4)\varepsilon) + Q(1+3\varepsilon) + O(\varepsilon^2).
\end{aligned}$$

A similar expression exists for $\hat{F}(X^{(r)})$, such that

$$\hat{F}(X^{(r+1)}) = F(X^{(r+1)}) + \overset{F\ell}{\Delta}X^{(r+1)})$$

and

$$\hat{F}(X^{(r)}) \quad = F(X^{(r)}) \quad + \overset{F\ell}{\Delta}X^{(r)})$$

Our convergence criteria must take these errors into account, such that

$$\begin{aligned}
FNORM = \frac{\|F(X^{(r+1)}) - F(X^{(r)})\|}{\|F(X^{(r)})\|} &\leq \frac{\|\Delta F(X^{(r)}) - \Delta F(X^{(r+1)})\|}{\|F(X^{(r)})\|} \\
&\leq (n+4)\frac{\left[\|F(X^{(r)})\| + \|F(X^{(r+1)})\|\right]}{\|F(X^{(r)})\|} \leq 2(n+4)u.
\end{aligned} \tag{4.40}$$

This says that TOL must be set to greater than or equal to $2(n+4)u$ and that it will truly reflect the accuracy of the computed solution.

Accumulation of Rounding Error

In general terms, accumulation of rounding errors may contaminate the solution obtained by any algorithm, particularly an iterative one. In this particular algorithm, there are three processes within the algorithm where it is possible for a significant loss of accuracy to occur. These are in the determination of the c.c.p of the current estimate to the solution matrix, in the computation of R_{n-1} and S_{n-1} and in the solution of the linear system at step 3. We have already discussed the stability of the algorithms to effect these processes, individually. The error in determining the a_i is small, δu say and the error in the computed R_{n-1} and S_{n-1} is of the same order as δu plus some small multiple of the machine precision u. This implies that the condition of the computed R_{n-1} will not be affected by rounding errors, that is, any well-conditioned exact R_{n-1} will not appear

to be ill-conditioned, and vice versa. The accuracy of the solution of the linear system using Gaussian Elimination is dependent on the condition of R_{n-1}. For an ill-conditioned R_{n-1} the computed solution X may not be close to the exact solution and any ensuing processing will lead to erroneous results.

Therefore, if R_{n-1} is well-conditioned and a stable method for determining the c.c.p of X is used, then at each iteration the error increases by a small multiple of machine precision. If the processing continues for a large number of iterations then the errors in the computed solution become significantly large. In practice however, we would expect to terminate the processing long before such a number of iterations is reached.

Convergence

The analysis presented here attempts to apply convergence theory to the iterative algorithms as well as laying the foundations for further investigation into their convergence.

The Contraction Mapping Theorem states that:

Let G map $R^n \rightarrow R^n$. If for some norm $\|\cdot\|$, there exists $\alpha \in [0,1)$ such that

$$\|G(x) - G(y)\| \leq \alpha \|x - y\| \quad , \qquad x, y \in R^n \tag{4.41}$$

then

(i) there exists a unique $x^* \in R^n$ such that $G(x^*) = x^*$,

(ii) for any $x^{(0)} \in R$, the sequence generated by

$$x^{(r+1)} = G(x^{(r)}) \tag{4.42}$$

remain in R^n and converges q-linearly to x^* with constant α,

(iii) for any $\eta \geq \|G(x^{(0)}) - x^{(0)}\|$,

$$\|x^{(r)} - x^*\| \leq \frac{\eta \alpha^r}{1-\alpha} \quad , \qquad r = 0, 1, \ldots$$

This implies global convergence of the iterations (4.42) provided that (4.41) is satisfied.

Also it is known [Morris] that if, with respect to (4.41)

(i) $G(x^{(r)})$ is defined and continuous on a region M, and

(ii) for each $x^{(r)} \in M$, there exists a $G(x^{(r)}) \in M$ and

(iii) the Jacobian matrix J of $G(x^{(r)})$ satisfies $\|J\| < 1$,

then the sequence of iterates (4.42) converge to a solution $x^* \in M$.

There are two approaches to redefining the iterative algorithm of this section in terms of the iteration (4.42).

Firstly, at iteration r we have,

$$R_{n-1}^{(r)} \; X^{(r+1)} = -S_{n-1}^{(r)} \tag{4.43}$$

where from section 4.2,

$$R_{n-1}^{(r)} = f(X^{(r)}) - f\alpha^{(r)}(X^{(r)} + P)$$

and

$$\begin{aligned} -S_{n-1}^{(r)} &= \left[a_n^{(r)} I - R_{n-2}^{(r)}\, Q\right] \\ &= -\left[a_n^{(r)} I - f\alpha^{(r)}\, Q\right] \\ &= f(X^{(r)})\, X^{(r)} + f\alpha^{(r)}\, Q \end{aligned}$$

Using Kronecker products, (4.43) may be written as

$$\left(I \otimes R_{n-1}^{(r)}\right)\ \underline{x}^{(r+1)} = \text{vec}\ \left(-S_{n-1}^{(r)}\right)$$

where $\underline{x}^{(r+1)}$ is a n^2- vector containing the elements of $X^{(r+1)}$ taken a row at a time. Then our iteration, equivalent to that in (4.42), is

$$\underline{x}^{(r+1)} = \left[I \otimes R_{n-1}^{(r)}\right]^{-1}\ \text{vec}\ \left(-S_{n-1}^{(r)}\right) \tag{4.44}$$

Secondly, from (4.30),

$$R_{n-1}^{(r)} = \sum_{0}^{n-1}\ a_{n-k-1}^{(r)}\ A_{22}^{(k)}$$

and

$$S_{n-1}^{(r)} = a_n^{(r)} I - \sum_{0}^{n-1}\ a_{n-k-1}^{(r)}\ A_{21}^{(k)}$$

that is, $R_{n-1}^{(r)}$ and $S_{n-1}^{(r)}$ are functions of $a_i^{(r)}$ and consequently we can write $R_{n-1}^{-1}\ S_{n-1}$ as some n function say G, of $a_i^{(r)}$. Also, since a non-derogatory $X^{(r+1)}$ may be transformed into its companion form which holds the c.c.p of the matrix $X^{(r+1)}$, it may be possible to obtain an iteration,

$$a_i^{(r+1)} = G\ \left((a_i^{(r)}\right) \tag{4.45}$$

Notice that the iterates are $\in R^n$ whereas those in (4.44) are $\in R^{n^2}$.

An alternative analysis now follows:
We have that

$$F(X^*) = 0 = X^{*^2} + PX^* + Q \tag{4.46}$$

$$F(X^{(r)}) = X^{(r)^2} + PX^{(r)} + Q \tag{4.47}$$

$$F(X^{(r+1)}) = X^{(r+1)^2} + PX^{(r+1)} + Q \tag{4.48}$$

Also,

$$\left(X^{(r)} - X^*\right)^2 = X^{(r)^2} + X^{*^2} - X^{(r)}\, X^* - X^*\, X^{(r)}$$

From the definition of the Fréchet derivative $F'(X)$, this may be written as

$$\left(X^{(r)} - X^*\right)^2 = X^{(r)^2} + X^{*^2} - F'(X^{(r)})\, X^* + PX^* \tag{4.49}$$

Similarly

$$\left(X^{(r+1)} - X^{(r)}\right)^2 = X^{(r+1)^2} + X^{(r)} - F'(X^{(r)})\, X^{(r+1)} + PX^{(r+1)} \tag{4.50}$$

Adding (4.46) to (4.47),

$$F(X^{(r)}) = (X^{(r)} - X^*)^2 + F'(X^{(r)})\, X^* + PX^{(r)} + 2Q \tag{4.51}$$

Adding (4.47) to (4.48),

$$F(X^{(r)}) + F(X^{(r+1)}) = (X^{(r+1)} - X^{(r)})^2 + F'(X^{(r)})X^{(r+1)} + PX^{(r)} + 2Q \tag{4.52}$$

Subtract (4.51) from (4.52),

$$F(X^{(r+1)}) = (X^{(r+1)} - X^{(r)})^2 - (X^{(r)} - X^*)^2 + F'(X^{(r)})\, (X^{(r+1)} - X^*)$$

or

$$F'(X^{(r)})\, (X^{(r+1)} - X^*) = (X^{(r)} - X^*)^2 + F(X^{(r+1)}) - (X^{(r+1)} - X^{(r)})^2 \tag{4.53}$$

Let the inverse of $F'(X^{(r)})$ exist. Multiply both sides by $F'(X)^{-1}$ and take norms,

$$\frac{\|X^{(r+1)} - X^*\|}{\|X^{(r)} - X^*\|^2} \leq \|F'(X^{(r)})^{-1}\| + \frac{\|F'(X^{(r)})^{-1}\|}{\|X^{(r)} - X^*\|^2} \left(\|F(X^{(r+1)}) - \left(X^{(r+1)} - X^{(r)}\right)^2\|\right) \tag{4.54}$$

This implies that the convergence of any iterative algorithm for the quadratic matrix equation is dependent on the conditioning of the derivative at each iteration.

Operations Count and Storage

We now look at the implementation of the method on a computer and determine the operations count and storage requirements. Detailed algorithms may be found in the Appendices.

Step 1. Generate a starting point from the following formulae,

$$a_1 = \frac{-\|P\|_E + \sqrt{\|P\|_E^2 + 4\|Q\|_E}}{2}$$

For $i = 0,\ n-2$

$$a_{n-i} = \frac{a_{n-(i-1)} - p_{n,i} - q_{n,i+1}}{a_1 - p_{n,n}}$$

where $p_{ij} = \text{sign}\ (p_{ij})\ \|P\|_E$, $q_{ij} = \text{sign}\ (q_{ij})\ \|Q\|_E$

This step requires $2n^2 + n$ storage locations - for P, Q and a_i.

The operations count is $n^2 + n$.

Step 2.

$$R1 = R2 = 0$$

$$A21 = 0$$

$$A22 = I$$

For $k = 0, \ldots, n-1$

$$APREV = A22$$

$$A22 = -A21 * Q - A22 * P$$

$$A21 = APRAV$$

$$R2 = R2 + a_{n-k-1} * A21$$

$$R1 = R1 + a_{n-k-1} * A_{22}$$

Next k

then $R_{n-1} = R1$

$$S_{n-1} = a_n I - R2 * Q$$

Since P and Q are already stored, we require $5n^2 + n$ storage locations for this step. The operations count is $2n^4 + 2n^3$.

Step 3. Gaussian Elimination with partial pivoting solves

$$R_{n-1}X = -S_{n-1}$$

It uses $\frac{4}{3}\ n^3$ operations and does not require any additional storage locations.

Step 4. Here we compute the norm of $F(X) = (f_{ij})$.

For $i = 1, \ldots, n$

For $j = 1, \ldots, n$

$$f_{ij} = \sum_{k=1}^{n} (x_{ik} + p_{ik})\ x_{kj} + q_{ij}$$

and

$$\|F(X)\|_E = \left(\sum\sum\ f_{ij}^2\right)^{\frac{1}{2}}$$

The operations count for this step is $n^3 + n^2$.

Step 5. Chapter 3 discussed a number of techniques for computing the characteristic polynomial of a matrix. For the purposes of this problem where the matrix X has no special form we choose the method due to Wang and Chen with a modest value of 5 for the constant c. Therefore the operations count is $6n^3$ and no further storage is required.

Note that Steps 2,3,4 and 5 are repeated until the algorithm is deemed to have converged.

The total operations count is $\left(2n^4 + 11n^3\right) m$ where m is the number of iterations. The total storage locations required is $6n^2 + n$.

SECTION 4.4: Elimination Method For The Square Root Problem

In this case, we require the solution X of

$$F(X) = X^2 - A = 0 \tag{4.55}$$

for the general matrix A.

The Elimination method of Section 4.1 gives the solution of (4.55) to be the solution of

$$R_{n-1}X = -S_{n-1} \tag{4.56}$$

where

$$R_{i+1} = S_i \qquad \text{where} \quad R_i = a_1 I$$
$$S_{i+1} = a_{i+2}I + R_i A \qquad S_i = a_2 I + A$$

The explicit expression for R_{n-1} and R_{n-2} (and hence S_{n-1}) is

$$[R_{n-2}\ R_{n-1}] = [0_n\ I_n] \sum_{k=0}^{n-1} a_{n-k-1} \begin{bmatrix} 0 & A \\ I & 0 \end{bmatrix}^k \tag{4.57}$$

Alternatively, R_{n-1} and R_{n-2} may be represented as,

$$R_{n-1} = f(A) - f\alpha X \tag{4.58}$$

$$R_{n-2} = f\alpha \tag{4.59}$$

where

$$Xf\alpha + f\alpha X = f(X) - f(-X) \tag{4.60}$$

and the scalar polynomial $f(c)$ is

$$f(c) = \sum_{k=0}^{n-1} a_{n-k-1} c^k$$

From Chapter 2, the derivative of $F(X)$ is $F'(X)$ and is defined as an operator such that (4.60) may be written as,

$$F'(X)\ f\alpha = f(X) - f(-X) \tag{4.61}$$

Also if $F'(X)$ is operating on a matrix $f(X)$, then

$$F'(X)f(X) = Xf(X) + f(X)X$$

Since $f(X)\ X = X f(X)$,

$$F'(X) f(X) = 2\ f(X) X \tag{4.62}$$

From (4.58),

$$\begin{aligned} F'(X)\ R_{n-1} &= F'(X)\,[f(X) - f\alpha X] \\ &= F'(X)\ f(X) - F'(X)\ f\alpha X \end{aligned}$$

using (4.61) and (4.62),

$$\begin{aligned} F'(X)\ R_{n-1} &= 2f(X)\ X - f(X)\ X + f(-X)\ X \\ &= f(X)\ X + f(-X)\ X \end{aligned} \tag{4.63}$$

Let $F'(X)$ be non-singular. Multiply both sides of (4.36) by $F'(X)^{-1}$ and take norms,

$$\|R_{n-1}\| \le \|F'(X)^{-1}\|\ \ \|\,[f(X) + f(-X)]\ \ X\| \tag{4.64}$$

similarly

$$\|R_{n-2}\| \le \|F'(X)^{-1}\|\ \ \|f(X) - f(-X)\ \|$$

Therefore the condition of the matrices R is related to the conditioning of the original problem (4.55).

Using terminology similar to that in Section 4.2, we can write

$$\begin{bmatrix} 0 & A \\ I & 0 \end{bmatrix}^k \triangleq \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix}$$

then

$$\begin{aligned} \begin{bmatrix} 0 & A \\ I & 0 \end{bmatrix}^{k+1} &\triangleq \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix} \begin{bmatrix} 0 & A \\ I & 0 \end{bmatrix} \\ &= \begin{bmatrix} A_{12}^{(k)} & A_{11}^{(k)} A \\ A_{22}^{(k)} & A_{21}^{(k)} A \end{bmatrix} \end{aligned}$$

$$\begin{aligned} A_{21}^{(k+1)} &= A_{22}^{(k)} \\ A_{22}^{(k+1)} &= A_{21}^{(k)}\ \ A \end{aligned} \qquad \text{with} \quad A_{22}^{(0)} = I,\ A_{21}^{(0)} = 0 \tag{4.65}$$

(4.57) may be written as,

$$[R_{n-2}\ R_{n-1}] = \sum_{0}^{n-1} a_{n-k-1} \left[A_{12}^{(k)}\ A_{22}^{(k)}\right]$$

The sequences in (4.65) are alternating between the zero matrix and an increasing power of A such that

$$R_{n-2} = \sum_{0}^{\text{int}\left(\frac{n-2}{2}\right)} a_{n-(2k+2)}\ A^k = f_2(A) \quad \text{say} \tag{4.66}$$

$$R_{n-1} = \sum_{0}^{\text{int}\left(\frac{n-1}{2}\right)} a_{n-(2k+1)}\ A^k = f_1(A) \quad \text{say} \tag{4.67}$$

and

$$\begin{aligned} S_{n-1} &= a_n I + R_{n-2}\ A \\ &= \sum_{0}^{\text{int}\left(\frac{n}{2}\right)} a_{n-2k}\ A^k = f_3(A) \quad \text{say} \end{aligned} \tag{4.68}$$

It is known that the characteristic roots of $f_1(A)$ are $f_1(\lambda_i(A))$ where $\lambda_i(A)$ are the eigenvalues of A. Since the eigenvalues of A are the square roots of those of X, then for singular X the following results hold,

$$\mu_r\ (R_{n-1}) = a_{n-1}$$

$$\mu_r\ (R_{n-2}) = a_{n-2}$$

$$\mu_r\ (S_{n-1}) = a_n = 0 \qquad \text{for some } r,$$

and for general X, the eigenvalues of R_{n-1} and S_{n-1} are

$$\mu_i\ (R_{n-1}) = \sum_{k=0}^{\text{int}\left(\frac{n-1}{2}\right)} a_{n-(2k+1)}\ \lambda_i^k$$

$$\mu_i\ (S_{n-1}) = \sum_{k=0}^{\text{int}\left(\frac{n}{2}\right)} a_{n-2k}\ \lambda_i^k$$

where $\lambda_i \ \ (i = 1, \ldots, n)$ are eigenvalues of Q.

An analysis of the rounding errors from (4.67) reveals,

$$\hat{R}_{n-1} = f\ell\left\{\sum_{0}^{r_1} a_{n-(2k+1)}(1+\delta a)\ \hat{A}^k\right\} \quad , \ r_1 = \text{int}\left(\frac{n-1}{2}\right) < \frac{n}{2}.$$

Now,

$$fl(\hat{A}^2) = fl(A^2(1+\varepsilon)^2) = A^2(1+2\varepsilon)(1+n\varepsilon) + O(\varepsilon^2)$$

$$fl(\hat{A}^3) = A^3(1+3\varepsilon)(1+2n\varepsilon) + O(\varepsilon^2)$$

$$\text{generally} \quad fl(\hat{A}^k) = A^k(1+k\varepsilon)(1+(k-1)n\varepsilon) + O(\varepsilon^2)$$

then

$$fl(\hat{a}_{n-(2k+1)}\hat{A}^k) = a_{n-(2k+1)}\ (1+\delta a)\ A^k\ (1+k\varepsilon)\ (1+(k-1)n\varepsilon)\ (1+\varepsilon) + O(\varepsilon^2)$$

$$= a_{n-(2k+1)}\ A^k\ (1+\delta a)(1+(k+1+kn-n)\varepsilon) + O(\varepsilon^2)$$

then from Section (1.3.3), the result on extended addition gives

$$\hat{R}_{n-1} = \sum_0^{r_1} a_{n-(2k+1)}\ A^k\ (1+\delta a)\ (1+(k+1+kn-n)\varepsilon)\ (1+(r_1+1-k)\varepsilon)$$

$$\frac{\|\hat{R}_{n-1} - R_{n-1}\|}{\|R_{n-1}\|} \leq |\delta a| + \left(\frac{n+4}{2}\right) u(1+|\delta a|) + \frac{nu(1+|\delta a|)}{\|R_{n-1}\|} \sum_0^{r_1} k|a_{n-(2k+1)}|\|A^k\|$$

$$\frac{\|\hat{R}_{n-1} - R_{n-1}\|}{\|R_{n-1}\|} \leq |\delta a| + \left(\frac{n+4}{2}\right) u + \frac{n^2u}{\|R_{n-1}\|}\ |a_{\max}| \cdot \|A^n\| \tag{4.69}$$

Similarly, for (4.68)

$$\frac{\|\hat{S}_{n-1} - S_{n-1}\|}{\|S_{n-1}\|} \leq |\delta a| + \left(\frac{n+4}{2}\right) u + \frac{n^2u}{\|S_{n-1}\|}\ |a_{\max}| \cdot \|A^n\| \tag{4.70}$$

These bounds suggest that the error in the computed matrices is of the same order as that in the initial data. That is, (4.67) and (4.68) are stable techniques for computing R_{n-1} and S_{n-1}.

In summary, if a_i are the c.c.p of X and the problem of solving (4.55) is well-conditioned, then R_{n-1} is well-conditioned and the solution of

$$R_{n-1}\ X = -Sn - 1$$

where R_{n-1} and S_{n-1} are accurately computed, is also the solution of (4.55).

SECTION 4.5 : Applications to Matrix Square Root Equations

This section describes three Elimination method based techniques for computing the square root of a matrix. The discussions consider the conditioning of any interprocess problems, the stability of the algorithms and the operations count for each defined task of an algorithm.

Method 1

This method is similar to that in Section 4.3 for the quadratic matrix equation. The algorithmic steps are as follows:

(i) Select scalars a_i $(1 = i, \ldots, n)$ as initial estimates to the c.c.p of X.

(ii) Carry out one step of the Elimination method to determine R_{n-1} and S_{n-1}.

(iii) Solve $R_{n-1}\, X^{(r+1)} = -S_{n-1}$.

(iv) Compute $\dfrac{\|F(X^{(r+1)}) - F(X^{(r)})\|}{\|F(X^{(r)})\|}$

If less than some specified tolerance, then the iterations have converged.

(v) Compute the c.c.p of $X^{(r+1)}$. Go to (ii).

If R_{n-1} is singular at any iteration then the Gaussian Elimination method that solves the equation in step (iii) will fail. If R_{n-1} is ill-conditioned at any iteration then the solution provided by Gaussian Elimination will be wholly unreliable. Within a computer algorithm, both these possible outcomes should be monitored, and reported on occurrence.

The discussions in Section 4.3 on a suitable starting point, accuracy of convergence criterion, accumulation of rounding error and convergence of the iterations are applicable and valid here with the following modifications:

The starting point is generated by

$$a_{n-i} = \frac{a_n - (i-1) + q_{n,i+1}}{a_1}$$

where $a_1 = \|A\|_E^{\frac{1}{2}}$ and $q_{i,j} = \text{sign}\,(q_{ij})\|A\|_E$.

The bound for the convergence criterion is

$$FNORM \leq 2(n+3)u$$

The step-by-step operations count now follows:

Step 1. n^2 to determine $\|A\|$

n to determine a_i

Step 2. R_{n-1} and S_{n-1} are computed simultaneously, using (4.67) and (4.68). The operations count is $\dfrac{n^4}{2} + n^3$.

Step 3. $\frac{4}{3}n^3$ operations are used by the Gaussian Elimination algorithm.

Step 4. n^3 to determine $f_{ij}^{(r+1)}$
n^2 to determine $\|F(X^{(r+1)})\|$

Step 5. The Wang and Chen method of Chapter 3 requires approximately $6n^3$ operations.

The total operations count is around $\frac{n^4}{2}+10n^3$ per iteration. The storage requirements are $4n^2+n$ locations.

The operations count here as compared to that for other algorithms for the matrix square problem will, in due course, be shown to be unfavorable. One way to speed up this algorithm is as follows:

From Section 1.4, there exists an orthogonal matrix H, say, that transforms a general matrix A, to upper Schur form. If A is symmetric then the transformed form $\tilde{A}$, is block diagonal with Jordan blocks on the diagonal. If additionally, A has distinct eigenvalues then $\tilde{A}$ will be diagonal. In any of these cases we have,

$$\begin{aligned} X^2 - A &= 0 \\ H^{-1}X\ HH^{-1}XH - H^{-1}AH &= 0 \\ \tilde{X}^2 - \tilde{A} &= 0 \end{aligned} \tag{4.71}$$

so the problem of solving (4.55) is now of solving (4.71) for $\tilde{X}$, then $X = H\tilde{X}H^{-1}$. The operations count for when $\tilde{A}$ is upper Schur and diagonal are now given:

	Upper Schur Form	**Diagonal Form**
Form $\tilde{A}$.	$15n^3$	$15n^3$
Step 1.	n^2+n	n^2+n
Step 2.	$\frac{n^4}{12}+\frac{n^3}{4}+\frac{n^2}{6}$	$\frac{n^3}{8}+n^2$
Step 3.	$\frac{n^2}{2}+\frac{n}{2}$	n
Step 4.	n^3+n^2	$2n^2+n$
Step 5.	$4n^3$	$3n^3$
$X=H\tilde{X}H^{-1}$	$2n^3$	n^3+n^2
Total (approx)	$17n^3+m\left(\frac{n^4}{12}+\frac{21n^3}{4}\right)$	$16n^3+n^2+4n^3m$

where m is the number of iterations for convergence to occur. Clearly this represents a major saving in the iterative processing. For small order problems, this saving is offset by the large amount of work required to transform A to $\tilde{A}$ but for large order problems, the saving may prove to be significant.

Method 2.

The roots of a polynomial are very sensitive to small changes in the coefficients of the polynomial, but the reverse is generally well-conditioned.

Let the characteristic polynomial of X be

$$f(\lambda)=\lambda^n+a_1\lambda^{n-1}+a_2\lambda^{n-2}+\ldots+a_n. \tag{4.72}$$

and if $\lambda_1,\lambda_2,\ldots,\lambda_n$ are the roots of $f(\lambda)$ i.e. the eigenvalues of X, then (4.72) may be factored as

$$f(\lambda)=(\lambda-\lambda_1)\,(\lambda-\lambda_2)\,(\lambda-\lambda_3)\ldots(\lambda-\lambda_n) \tag{4.73}$$

Equating coefficients in powers of λ in (4.72) and (4.73) yields expressions which relate the coefficients a_i in terms of the elementary symmetric functions of the eigenvalues. The m^{th} elementary symmetric function is the sum Σ of the n eigenvalues $\lambda_1,\lambda_2,\ldots\lambda_n$ taken m at a time so that Σ has $n_{(m)}$ terms where [Turnbull]

$$n_{(m)}=\binom{n}{m}=\frac{n!}{m!(n-m)!} \tag{4.74}$$

The m^{th} coefficient is given by

$$a_m = (-1)^m \sum_{i_1=1}^{n-(m-1)} \sum_{i_2=i_1+1}^{n-(m-2)} \cdots \sum_{i_m=i_{m-1}+1}^{n} \lambda_{i_1}, \lambda_{i_2}, \ldots \lambda_{i_m} \tag{4.75}$$

Let μ_i be the eigenvalues of A and λ_i the eigenvalues of X. Then since

$$\mu_i(X^2) = (\lambda_i(X))^2$$

and $X^2 = A$, we have that the eigenvalues of X are the square roots of the eigenvalues of A.

$$\lambda_i = \pm \mu_i^{\frac{1}{2}} \tag{4.76}$$

The choice of signs determines the definiteness of X. If a matrix X with eigenvalues $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$ is a square root of A then so is $X_1 = -X$ with eigenvalues $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$.

If each distinct set $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$ represents a distinct solution X then there are 2^n possible solutions corresponding to 2^n possible combinations of n positive and negative signs in (4.76).

The observations above lead to the following algorithm for computing a square root of A.

(i) Compute the eigenvalues of A say $\mu_1, \mu_2, \ldots \mu_n$.

(ii) Determine the square roots of μ_i and denote as λ_i $(i = 1, n)$. These are the eigenvalues of X.

(iii) Compute the coefficients a_i, the c.c.p. of X from (4.75).

(iv) Use the Elimination method to compute X.

Error Analysis

From Section 1.4, we have that if a general square matrix A is perturbed by a matrix E of order ε then an eigenvalue μ_k may be perturbed by an amount ε/s_k where

$$s_k = |y^T x| \quad , \text{ i.e. } |\hat{\mu}_k - \mu_k| \leq \frac{\varkappa \varepsilon}{s_k} \tag{4.77}$$

and

$$Ax = \mu x \qquad\qquad y^T A - \mu y^T$$

with x, y normalised such that $\|x\|_2 = \|y\|_2 = 1$. The QR algorithm that is used to compute the eigenvalues does not generate errors that are not present in the original data, so that the errors in the computed eigenvalues at step 1 are bounded by (4.77).

At step 3, the coefficients are computed from (4.75). The results from Section 1.3 on the rounding errors analysis of extended addition and subtraction are used here.

$$\hat{a}_1 = f\ell\left(\sum_{i=1}^{n_1} \hat{\lambda}_i\right)$$

$$= \hat{\lambda}_1 (1+\varepsilon)^{n_1-1} + \sum_{i=2}^{n_1} \hat{\lambda}_i (1+\varepsilon)^{n_1-i+1}$$

$$= a_i + \sum_{i=1}^{n_1} \delta\lambda_i + \sum_{i=1}^{n_1} i\,\lambda_i\,\varepsilon - \lambda_1\varepsilon$$

$$|\hat{a}_1 - a_1| \leq u \sum_{i=1}^{n_1} \frac{1}{s_i} + (n_1 - 1)\, u \sum_{i=1}^{n_2} |\lambda_i|$$

Similarly,

$$|\hat{a}_2 - a_2| \leq u \sum_i \sum_j \left|\left(\frac{\lambda_i}{s_j} + \frac{\lambda_j}{s_i}\right)\right| + n_2 u \sum_i \sum_j |\lambda_i \lambda_j|$$

$$\leq u\, 2n_2 \frac{\max(|\lambda_k|)}{\min(s_k)} + n_2 u \sum_i \sum_j |\lambda_i \lambda_j|$$

and generally,

$$|\hat{a}_m - a_m| \leq umn_m \frac{\max|\lambda_k|}{\min(s_k)} + u(n_m + m - 2) \sum_{i_1} \cdots \sum_{i_m} |\lambda_{i_1} \lambda_{i_2} \ldots \lambda_{i_m}| \tag{4.78}$$

$$= u(n_m\alpha + (n_m + m - 2)\beta)$$

where $\alpha = m \frac{\max|\lambda_k|}{\min(s_k)}$, $\beta = \sum_{i_1} \cdots \sum_{i_m} |\lambda_{i_1}||\lambda_{i_2}| \ldots |\lambda_{i_m}|$

Hence

$$\frac{|\hat{a}_m - a_m|}{|a_m|} \leq u \frac{(n_m(\alpha+\beta) + (m-2)\beta}{|a_m|} \tag{4.79}$$

This bound suggests that if the eigenvalues of the matrix A are ill-conditioned, then the computed coefficients may be at least as large as the error in the most ill-conditioned eigenvalue multiplied by $n_m u$. The existence of n_m on its own may lead to significant errors in the computed values, for large n. To give some idea of the sizes involved we give some values of n_m for varying n. The term n_m has a greatest value at $m = \text{int}\left(\frac{n+1}{2}\right)$.

n	n_m	un_m	$u \approx 10^{-18}$
10	252	10^{-15}	
20	184756	10^{-13}	
30	1.6×10^8	10^{-16}	
40	1.4×10^{11}	10^{-7}	
50	1.2×10^{14}	10^{-4}	

It is evident that as n increases by 10, the accuracy of the coefficient diminishes by a factor of 10^{-3}.

At step 4 of the Elimination method, the computed coefficients are used to build a solution. From the previous section we observed how the accuracy of R_{n-1}, S_{n-1} and hence X is dependant on the accuracy of the coefficients. Therefore, we can say that this method will produce an accurate solution to a low order matrix square root problem that has a well-conditioned eigenvalue problem.

Now let us examine the operations count and storage requirements for this method.

The storage locations needed is $4n^2 + 2n$.

The operations count for each step is:

(i) $16n^3$ to compute the eigenvalues

(ii) n to compute the roots of the eigenvalues

(iii) to compute each coefficient requires $n_m * (m-1)$ operations

$$\text{sum} = \sum_{m=1}^{n} (m-1) \frac{n!}{m!\,(n-m)!}$$

(iv) $\frac{n^4}{2} + n^3$ for the Elimination method

$$\text{Total} \approx \sum_{m=1}^{n} (m-1)\frac{n!}{m!\,(n-m)!} + n^4 + 16n^3 + 0(n^2)$$

The factorial term in the total suggests that this method is unsatisfactory for large order problems.

Method 3

Let the characteristic polynomial of X be

$$f(\lambda) = \lambda^n + a_1\lambda^{n-1} + \ldots + a_n = |X - \lambda I| \tag{4.80}$$

and that of A be

$$f(\mu) = \mu^n + c_1\mu^{n-1} + \ldots + c_n = |A - \mu I|$$

Now since $X^2 = A$ it follows that

$$|A - w^2 I| = |X^2 - w^2 I| = |X - wI|\ |X + wI| \tag{4.81}$$

or

$$\begin{aligned} & w^{2n} + c_1^{2(n-1)} + c_2 w^{2(n-2)} + \ldots + c_{n-1} w^2 + c_n \\ & = \left(w^n + a_1 w^{n-1} + a_2 w^{n-2} + \ldots + a_{n-1}\ w + a_n\right) \\ & * \ \left(w^n - a_1 w^{n-1} + a_2 w^{n-2} + \ldots + (-1)^n a_n\right) \end{aligned} \tag{4.82}$$

All the coefficients in odd powers of w vanish on the right hand side of (4.82) giving,

$$w^{2n} + c_1 w^{2(n-1)} + \ldots + c_n = w^{2n} + \left(2a_2 - a_1^2\right) w^{2(n-1)} + \ldots + (-1)^n a_n^2 I \tag{4.83}$$

Comparing coefficients of w in (4.83) gives:

$$\begin{aligned} c_1 &= 2a_2 - a_1^2 \\ c_2 &= a_2^2 + 2a_4 - 2a_1 a_3 \\ &\vdots \\ c_{n-1} &= (-1)^{n-1}\left(a_{n-1}^2 - 2a_n a_{n-2}\right) \\ c_n &= (-1)^n a_n^2 \end{aligned}$$

Generally

$$c_i = 2\sum_{k=i}^{2i} (-1)^{2i-k}\ a_{2i-k}\ a_k + (-1)^{i-1}\ a_i^2 \tag{4.84}$$

with $a_0 = c_0 = 1$ and $i = 1, \ldots, n$

(4.84) is in fact a relationship between the c.c.p of A and the c.c.p of X, giving rise to a system of n non-linear equations in n unknowns.

This relationship yields the following algorithm for computing the square root X of a matrix A :

(i) compute the c.c.p of A

(ii) solve (4.84) for the unknowns a_i, $i = 1, \ldots, n$

(iii) use the Elimination method to determine X.

The problem of determining the a_i from (4.84) is equivalent to one of finding the zeros of,

$$f_i(a) = c_i - 2 \sum_{k=i}^{2i} (-1)^{2i-k}\, a_{2i-k} a_k + (-1)^i a_i^2 \tag{4.85}$$

Newton's method for non-linear equations may be used to determine the zeros. However, the convergence is not global. A global strategy of Section 1.5 uses a combination of Newton and Gauss-Newton iterations to determine an unconstrained minimisation of fc, where

$$fc = \frac{1}{2} \sum_{1}^{n} fi^2.$$

The update is given by

$$a^{(k+1)} = a^{(k)} - \alpha^{(k)}\, p^{(k)}$$

where

$$J^T J\, p^{(k)} = -J^T\, f^{(k)}, \quad \text{Gauss} - \text{Newton iteration} \tag{4.86}$$

or

$$H\, p^{(k)} = -J^T f, \quad \text{Newton iteration}$$

where J and H are the Jacobian and Hessian respectively, of the function fc at $a^{(k)}$.

Notice that (4.86) is equivalent to

$$J\, p^{(k)} = -f^{(k)} \tag{4.87}$$

Using (4.85), we have that

$$J = \frac{\partial f_i}{\partial a_j} = 2 \begin{bmatrix} a_1 & -1 & 0 & 0 & \dots & \dots & 0 \\ a_3 & -a_2 & a_1 & -1 & \dots & \dots & 0 \\ a_5 & -a_4 & a_3 & -a_2 & \dots & \dots & 0 \\ \vdots & & & & & & \vdots \\ 0 & & & & & & (-1)^{n-1} a_{n-2} \\ 0 & & & & & 0 & (-1)^{n-1} a_n \end{bmatrix} \tag{4.88}$$

and from (1.27), $H = J^T J + Z$, where

$$Z = \sum_{i=1}^{n} fi \, \frac{\partial^2 f_i}{\partial a_k \partial a_j} = 2 \begin{bmatrix} f_1 & 0 & f_2 & 0 & \dots & \dots \\ 0 & -f_2 & 0 & -f_3 & & \\ f_2 & 0 & f_3 & & & \\ 0 & -f_3 & & \ddots & & \\ \vdots & & & & \ddots & \\ & & & & & (-1)^{n-1} f_n \end{bmatrix} \tag{4.89}$$

A couple of interesting points arise out of these expressions. Firstly, if the solution X of the matrix square root problem is singular then its derivative $F'(X)$ is singular, since $a_n^* = 0$, and consequently the Jacobian in (4.88) is singular at the root. Therefore the addition of a scalar matrix to $J^T J$, as described in section 1.5, is necessary.

Secondly, the argument put forward in Section (1.5) implying that the local convergence of the iterates would not be greatly affected if the Gauss-Newton iteration is used rather than the Newton certainly stands up here owing to the smallness of Z, locally. From an operations count point of view, this is clearly preferable.

Algorithmically, (4.88) may be represented as

$$J = (J_{ij}) = (-1)^{j+1} 2 a_{2i-j}$$

and

$$(g_j) = J^T f = \sum_{i=1}^{n} J_{ij} f_i$$

Let

$$J^T J = U = (u_{ij}) = \sum_{k=1}^{n} J_{ki} \; J_{kj} = 4 \sum_{k=1}^{n} a_{2k-i} \; a_{2k-j}$$

Since some of the operations in forming U are redundant due to the number of zeros in J, an efficient way to express U is

$$u_{ij} = 4 \sum_{k=i_0}^{i_1} a_{2k-i} \; a_{2k-j}$$

where $j = 1, \; n$ and $i = i_0, \; i_1$ with $i_0 = \text{int} \left(\frac{j+1}{2} \right)$ and $i + 1 = \text{int} \left(\frac{n + i_0}{2} \right)$
where int represents the integer part.
Similarly

$$g_j = \sum_{i=i_0}^{i_1} 2 a_{2i-j} f_i$$

and

$$z_{ij} = \begin{cases} (-1)^{j+1} \quad \dfrac{f_{i+j}}{2} & \text{when } i+j \text{ is even} \\ 0 & \text{when } i+j \text{ is odd} \end{cases}$$

We may determine the operation count in determining J^TJ, J^Tf and H.

$$\text{Total (for } J^TJ) = \sum_{j=1}^{n} \sum_{i=i_0}^{i_1} \sum_{k=i_0}^{i_1} 1$$

$$= \sum_{j=1}^{n} \sum_{i=i_0}^{i_1} (i_1 + 1 - i_0)$$

and since i_0, i_1 are independent of i

$$= \sum_{j=1}^{n} (i_1 + 1 - i_0)^2$$

$$= \sum_{j=1}^{n} (i_1^2 + i_0^2 - 2i_0i_1 + 2i_1 - 2i_0 + 1)$$

From earlier, put $i_0 = \dfrac{j+1}{2}$ and $i_1 = \dfrac{n+i_0}{2}$, to give

$$\text{Total} = \sum_{j=1}^{n} \left[\frac{n^2}{4} + n + 1 + \frac{1}{16}(j^2 + 2j + 1) - \frac{n}{4}(j+1) - \frac{1}{2}(j+1) \right]$$

Using the identities from Chapter 1 concerning the summation of series we have,

$$\text{total} = \frac{1}{6} n^3 + \frac{1}{2} n^2 + 0(n)$$

To compute J^Tf and Z it is necessary to determine f_i for $i = 1, \ldots, n$.
f_i may more usefully be expressed as,

$$f_i = c_i - \sum_{k=i}^{2i} (-1)^{2i-k} \, a_{2i-k} a_k + (-1)^i a_i^2$$

so that the number of operations required to determine the f_i is given by,

$$\sum_{i=1}^{n} (2(2i + 1 - i) + 2)$$

$$= 4n + 2 \sum_{i=1}^{n} i = n^2 + 5n$$

To determine J^Tf requires $\displaystyle\sum_{j=1}^{n} \sum_{i=i_0}^{i_1} 3 + n^2 + 5n = \frac{7}{4} n^2 + 7n$ operations.

Since it requires one operation to determine the sign of a scalar, the operations count in determining the signs of the elements of the matrix Z is $\frac{n^2}{2} + O(n)$, so that overall, to compute the Hessian once requires about

$$\frac{1}{6}\, n^3 + 2n^2 \text{ operations}$$

The Choleski decomposition method is used to solve the linear system with an operations count of $\frac{n^3}{6}$.
Overall, the Newton iterates require

$$\frac{n^3}{6} + c_1 n^2 + \frac{n^3}{6} + 2n^2 + c_2\, \frac{3n^2}{2} + O(n)$$

operations per iteration. c_1 is the number of function calls and $c_2 = 1$ if the Hessian is required, otherwise 0.

An estimate for the operations count at each step of the square root algorithm is,

(i) $6n^3$ to compute the c.c.p of A

(ii) $\left(\frac{n^3}{3} + \frac{n^2}{2}\,(4 + 2c_1 + c_2)\right)\, m$, m is the total number of iterations

(iii) $\frac{n^4}{2} + n^3$ to compute X from the Elimination method.

Step (ii) exhibits all the usual convergence and stability properties of the Newton method, as described in Section 1.5. The known constants c_i, the inputs to the Newton iterations, are computed accurately at step (i) using the Block Frobenius method (or alternatively the Stable Le Verriers method). The a_i determined by the Newton iterations will usually be accurate since the terminating condition is that the norm of the gradient vector be less than some specified small tolerance value. Consequently, provided that the matrix R_{n-1} in the Elimination method is well-conditioned at the root, step(iii) will determine an accurate solution for the matrix square root problem.

CHAPTER 5 : CURRENT METHODS FOR QUADRATIC MATRIX EQUATIONS

This chapter summarily describes some of the more widely used methods for solving the matrix equations of interest. For each type of matrix equation, numerical methods are discussed with respect to their efficiency, stability and accuracy of solution. At the end of each section, a brief comparison of the operations count and algorithmic features of the methods is given. Notice that these comparisons are only relative. Consequently, it would be useful to have some absolute point of reference with which the efficiency of the methods can be compared with. Therefore, we discuss a globally convergent variant of the 'worst' possible algorithm where the matrix equation of order n, say, is redefined as a system of n^2 non-linear equations in the n^2 unknown elements of the solution matrix and solved by minimisation. It may be that for certain problems, no method can provide a solution in which case this variant of the 'worst algorithm' would be the only alternative.

We begin by identifying the classes of methods that have been used to solve the matrix equations.

SECTION 5.1 : Introduction

There have been greater advances towards the solutions of the Algebraic Riccati Equations than the monic unilateral quadratic matrix equation. This not surprising since the ARE arises more frequently in applications where information on the coefficient matrices and on the solution matrix is known beforehand. This is in contrast to the unilateral equation where usually nothing is known about the matrices. Clearly, when $P = 0$, we have the matrix square root problem for which a large amount of theory is available.

The currently available practical methods for solving the ARE fall into two groups, there is a possible third group which we will discuss later. These are iterative methods and those that are based on the determination of the invariant subspace of the associated Hamiltonian matrix, sometimes referred to as doubling algorithms. The classical method based on successive Newton iterations involving an induced Lyapunov equation is still the best iterative method available, and is the only one of that type to be considered here. As a consequence of advances in numerical analysis and in the production of more reliable and accurate numerical techniques, this Newton's method is continually being tuned to improve performance and stability [Hammarling].

In a similar way, the broader principles of the eigenvector method still form the basis of new methods. With the emergence of numerically stable and efficient algorithms, like the QR algorithm, variants of the eigenvector approach have arisen, particularly the Schur vectors approach. A recent method uses the properties of the Hamiltonian matrix, common to both the eigenvector method and the Schur vectors approach, to derive a variant of the QR algorithm for determining the Schur vectors.

The matrix sign function was originally designed for control problems and forms the basis of a family of algorithms for solving a variety of invariant subspace related problems. With respect to the ARE, the matrix sign function gives rise to a matrix $\hat{X}$ that is close to the solution of the ARE. $\hat{X}$ is not close enough to be considered as an accurate solution, hence iterative refinement in the form of Newton's method takes place. In this sense, this method, as mentioned earlier, may be thought of as belonging to a different class of methods, that combine the properties of both the invariant subspace and iterative approaches.

However, analysis of matrix sign function algorithms has shown them to be numerically stable only for a small class of matrices. This includes symmetric matrices and a recent method shows how non-symmetric matrix inversions in the matrix sign function based methods for the ARE can be changed into symmetric matrix inversions.

From the theoretical point of view there have been a number of proposed approaches to solving the unilateral quadratic matrix equation but those taking numerical aspects into consideration have been far fewer, due to the arbitrary nature of the coefficient and solution matrices. The two methods discussed in this chapter are the Newton Iterations method

and a generalisation of a scalar polynomial algorithm, the Matrix Polynomial algorithm. The convergence conditions for the iterative method are extensions of these of Newton's method for the scalar case. To our knowledge there does not exist any stability analysis of the Matrix Polynomial algorithm and as such we develop a relationship between the norm of the inverse of the derivative operator introduced in Section 2.1 and the stability of the algorithm.

A number of methods for computing the square root of a matrix have been proposed, some for general problems and others for specific problems. Specific types include those that are real, symmetric, positive definite, of small order etc. For all problems, advances in numerical mathematics have lead to improved algorithms, the more successful ones being based on Newton's method either directly or via the sign function and Schur vector factorisations. We discuss these along with some algorithms for special problems.

SECTION 5.2 : Methods For The Algebraic Ricatti Equation

Of interest here is the non-negative symmetric solution of the equation

$$A^T K + KA - KGK + H = 0 \tag{5.1}$$

where all matrices are square, of order n with $H = H^T$ positive semi-definite and G positive definite.

5.2.1 Eigenvector Method

This method [Potter] is based on the eigenvalue-eigenvector analysis of a $2n$ partitioned matrix M, given by

$$M = \begin{bmatrix} A^T & H \\ G & -A \end{bmatrix}.$$

Suppose K, the solution of (5.1) is such that

$$C = GK - A \tag{5.2}$$

From (5.1)

$$KC = H + A^T K. \tag{5.3}$$

Let S transform C into its Jordan canonical form J,

$$J = S^{-1} C S \tag{5.4}$$

and let

$$R = KS. \tag{5.5}$$

Using (5.2)-(5.5) to eliminate C and K yields

$$RJ = A^T R + HS$$

$$SJ = GR - AS$$

$$\begin{bmatrix} R \\ S \end{bmatrix} J = \begin{bmatrix} A^T & H \\ G & -A \end{bmatrix} \begin{bmatrix} R \\ S \end{bmatrix} = M \begin{bmatrix} R \\ S \end{bmatrix}. \tag{5.6}$$

In (5.6) J must be diagonal for if $a_1, a_2, \ldots a_n$ are the columns of $\begin{bmatrix} R \\ S \end{bmatrix}$ and J is not a diagonal matrix, then for some k

$$0 = (M - \lambda I)a_k$$

and

$$a_k = (M - \lambda I)a_{k+1}$$

where a_k is an eigenvector of M corresponding to the eigenvalue λ. If M is assumed to possess a diagonal Jordan canonical form then its minimal polynomial is a product of distinct linear factors,

$$m(x) = (x - \lambda_1)\dots(x - \lambda_p)(x - \lambda)$$

Now

$$0 = m(M)a_{k+1} = (\lambda - \lambda_1)\dots(\lambda - \lambda_p)a_k$$

and since $\lambda \neq \lambda_i$ for $i = 1, \dots, p$ we conclude that $a_k = 0$. But this is impossible since S is non-singular. It follows from the fact that J is diagonal that $a_1, \dots, a_n$ are eigenvectors of M and the solution K is given by (5.5).

The following theorems, due to Potter, concern the symmetry and definiteness of the solution K.

If a is an eigenvector of M of dimension $2n$ then let $a = \begin{bmatrix} b \\ c \end{bmatrix}$ where b and c are n-vectors.

Theorem

If H and G are hermitian, $a_1 \dots a_n$ are eigenvectors of M corresponding to eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$ and $\lambda_i \neq \lambda_j^*$ (* denotes the complex conjugate) for $1 \leq i,\ j \leq n$, and if $[c_1, \dots, c_n]$ is non-singular, then

$$[b_1, \dots, b_n]\ [c_1, \dots, c_n]^{-1} \text{ is hermitian.}$$

Theorem

Let H, G be positive semi-definite hermitian and $a_1, \dots, a_n$ be eigenvectors of M corresponding to eigenvalues $\lambda_1, \dots, \lambda_n$, then

(a) if H or G is non-singular and

$$X = [b_1, \dots, b_n]\ [c_1, \dots, c_n]^{-1} \text{ is positive definite}$$

then $\lambda_1, \dots, \lambda_n$ have positive real parts

(b) if $\lambda_1, \ldots, \lambda_n$ have positive real parts and $[c_1, \ldots, c_n]$ is non-singular then $[b_1, \ldots, b_n]\ [c_1, \ldots, c_n]^{-1}$ is positive semi-definite.

Here is a sequence of steps which would effect the eigenvector method, along with the operations count.

1. Compute the upper Hessenberg form of M such that

$$U_0^T M U_0 = H, \qquad M,\ U_0,\ H \in \mathrm{R}^{2n \times 2n}$$

The operations count is $\frac{5}{3}\ (2n)^3$.

2. Compute the Schur form of M from the upper Hessenberg form, without accumulating transformations, to determine the eigenvalues of M. Operations count is about $8(2n)^3$.

3. For each eigenvalue λ_i satisfying Re $(\lambda_i) > 0$, apply the following algorithm (inverse iteration):

For $k = 1, 2, \ldots$

Solve $(H - \lambda_i I)\ z_i^{(k)} = z_i^{(k-1)}$

Normalise: $z_i^{(k)} = \dfrac{z_i^{(k)}}{\|z_i^{(k)}\|_\infty}$

with $z_i^{(0)}$ as the unit vector.

A suitable stopping criteria might be to quit when the residual

$$r^{(k)} = (H - \lambda_i I) z_i^{(k)}$$

is such that

$$\|r^{(k)}\|_\infty \leq cu\ \|H\|_\infty\ \|z_i^{(k)}\|_\infty$$

where c is of order unity.

The operations count is about $k(2n)^2$ per eigenvalue $= 4kn^3$. From experience, it is found that an average of 3 iterations is required.

4. Set $a_i = U_0 z_i,\ i = 1, \ldots, n,\ a_i \in \mathrm{R}^{2n}$.

If $[a_1, a_2, \ldots a_n] = \begin{bmatrix} B \\ C \end{bmatrix}$ where $B,\ C \in \mathrm{R}^{n \times n}$ and $B = [b_1, b_2, \ldots, b_n]$ and $C = [c_1, c_2 \ldots, c_n]$, then the solution of (5.1) is given by X where

$$XC = B$$

The operation count for this step is

$$4n^3 + \frac{4}{3}\ n^3 = \frac{16}{3}\ n^3$$

The operation count for the entire process is about $100n^3 + O(n^2)$. The storage requirements are $10n^2$ storage locations.

From section 1.4, the methods for determining the Hessenberg form and the Schur form of a matrix are stable and accurate. Similarly, the methods for determining the eigenvectors of a matrix and for solving a linear system are also stable. However, the problem of determining the eigenvectors is dependent on the conditioning of those eigenvalues of M having Re $(\lambda) > 0$. If these eigenvalues are ill-conditioned, then the problem of determining the eigenvectors is also ill-conditioned and the overall method is considered as unstable. Having said this, in applications which require the solution of the ARE, the solution is symmetric and the eigenvalues are usually well-conditioned [Hewer & Nazaroff].

5.2.2 Schur Vectors Method

In this method [Laub], it is assumed that (A, B) is a stabilizable pair [Wonham], where $BB^T = G$ with Rank (B) = Rank (G), and (C, A) is a detectable pair [Wonham], where $C^TC = H$ with rank (C) = rank (H).

The method uses a set of Schur vectors to solve (5.1) for the unique non-negative definite solution.

Consider the Hamiltonian matrix

$$Z = \begin{bmatrix} A^T & H \\ G & -A \end{bmatrix}$$

where the above assumptions guarantee that Z has no pure imaginary values. There exists an orthogonal transformation, $U \in \mathrm{R}^{2n \times 2n}$ which puts Z into real Schur form

$$U^T Z U = \begin{bmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{bmatrix}, \qquad S_{ij} \in \mathrm{R}^{n \times n}$$

Moreover, it is possible to arrange that the real parts of the spectrum of S_{11} are negative while the real parts of the spectrum of S_{22} are positive. U is conformably partitioned into four $n \times n$ blocks,

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix}$$

The first n-vectors of U are the Schur vectors corresponding to the spectrum of S_{11}.

Theorem

U_{11} is invertible and $X = U_{21}U_{11}^{-1}$ solves (5.1) with $X = X^T$ positive definite.

Proof

See [Laub].

The steps involved in the implementation of this method along with the respective operation counts are as follows,

1. Reduce Z to Upper Hessenberg form, using Householder reductions. The operations count is $\frac{5}{3}(2n)^3$ bearing in mind that Z is $2n \times 2n$.
2. Reduce the Upper Hessenberg form to Upper Real Schur form.
   Order the blocks on the diagonal of the Schur matrix such that the eigenvalues appear in descending order of magnitude along the diagonal.
   The ordering can be incorporated into the QR algorithm [Stewart, 2], requiring $8(2n)^3 + O(n^2)$ operations.
3. Solve
   $$XU_{11} = U_{21} \qquad \text{for } X$$
   Gaussian Elimination with partial pivoting uses $\frac{4}{3}n^3$ operations to solve this matrix equation.
   An overall estimate for the entire process is $75n^3$ operations.
   With respect to storage considerations, the algorithm requires $8n^2$ storage locations.

A recent analysis [Petkov, Christov & Konstantinov] shows that in some cases this method is numerically unstable, as reflected by the following theorem.

Theorem

Define the separation between two matrices M and N say, by

$$\text{sep}\,(M,N) = \min_{\|Y\|=1} \{\|MY - YN\|_E\}$$

Then if

$$\Delta = \text{sep}\,(S_{11}, S_{22}) - 2c_1 u\|Z\| > 0$$

and

$$c_1 u\|Z\|^2(1 + c_1 u) \leq \frac{1}{4}\Delta^2$$

then the solution of (5.1) using the Schur approach satisfies

$$\frac{\|\hat{X} - X\|}{\|X\|} \leq 2c_1 u\left(1 + \frac{1}{\|X\|}\right)(2\|A\| + \|H\| + \|G\|)\,\frac{\|\hat{U}_{11}^{-1}\|}{\Delta} + c_2 u\ \text{cond}\,(\hat{U}_{11})\,\frac{\|\hat{X}\|}{\|X\|}$$

where c_1, c_2 are small constants, u is machine precision and cond $(\hat{U}_{11})$ is the condition of $\hat{U}_{11}$.

This bound suggests that the error in the computed solution $\hat{X}$ will be very large if the computed matrix $\hat{X}_M$ is ill-conditioned, or if $\|\hat{U}_{11}^{-1}\|$ is very large.

5.2.3 Hamiltonian-Schur Decomposition

This approach [Byers, 5] differs from the Schur Decomposition of Section 5.2.2 in the way it takes advantage of the Hamiltonian structure of the matrix Z.

A matrix Z is Hamiltonian if

$$JZ + Z^T J = 0$$

and symplectic if

$$Z^T JZ - J = 0$$

where

$$J = \begin{bmatrix} 0_n & I_n \\ -I_n & 0_n \end{bmatrix}$$

The approach in Section 5.2.2 computes the invariant subspace by using the QR algorithm, which does not make use of the Hamiltonian structure of Z. The symmetric solution matrix X is then obtained from the product of two non-symmetric matrices U_{21} and U_{11}^{-1}. The Hamiltonian-Schur approach computes the invariant subspace using similarity transformations with symplectic matrices, which preserve the Hamiltonian structure of Z and maintains the symmetry of an approximate solution X at each step of the variant QR algorithm [Byers, Mehrmann].

Details of the algorithmic and analytical aspects of this method are given in [Byers, 5] where it is shown to be numerically stable. Test examples indicate that this method is faster than the method in Section 5.2.2 with less storage requirements and a smaller operations count. Unfortunately, this method is not a general one in the sense that it is limited to solving those ARE's arising from a single input control system, i.e. Rank $(G) = 1$, when G is symmetric and positive semi-definite.

5.2.4 Newton's Method

This method [Kleinman] is a monotically convergent iterative technique based on the method of successive substitutions.

Remember from Chapter 1, that the optimal control that minimises the quadratic cost functional

$$J[x, u] = \frac{1}{2} \int_0^\infty (x^T Hx + u^T Ru)\, dt$$

is given by

$$u_* = -R^{-1}B^T\, Kx.$$

Now suppose that

$$u_L(x(t)) = -Lx(t)$$

is an arbitrary feedback law. If this is applied to the system

$$\dot{x} = Ax + Bu \tag{5.7}$$

the resulting cost functional is

$$J[x,\ u_L] = x^T V_L x$$

where V_L is defined as the cost matrix associated with the feedback gains L. Note that the cost matrix associated with the optimal

$$L_* = R^{-1}B^T K \quad \text{is} \quad V_L = K.$$

The cost matrix is given by

$$V_L = \int_0^\infty e^{(A-BL)^T t}\,(H + L^T R L)e^{(A-BL)t}\,dt$$

V_L is finite if and only if the closed-loop system matrix $A - BL$ has eigenvalues with negative real parts. In this case V_L is the unique positive definite solution of the linear equation

$$0 = (A - BL)^T V + V(A - BL) + H + L^T R L.$$

The method for solving (5.1) is embodied in the following theorem.

Theorem

Let $V_k,\ k = 0, 1, \ldots$ be the unique positive definite solution of the linear algebraic equation

$$0 = A_k^T V_k + V_k A_k + H + L_k^T R L_k \tag{5.8}$$

where

$$\begin{aligned} L_k &= R^{-1}B^T V_{k-1} \\ A_k &= A - BL_k \qquad\qquad \text{for } k = 1, 2\ldots \end{aligned} \tag{5.9}$$

and where L_0 is chosen such that the matrix $A_0 = A - BL_0$ has eigenvalues with negative real parts. Then

1 $\quad (k) \leq V_{k+1} \leq V_k \leq \ldots \qquad k = 0, 1, \ldots$

2 $\quad \lim_{k->\infty} V_k = (k)$

Proof

See [Kleinman].

Since the system (5.7) is completely controllable it is always possible to choose an L such that Real $\{\lambda_i(A_0)\} < 0$.

It is necessary for Real $\{\lambda_i(A_0)\} < 0$ to insure the boundedness of the cost matrix V_0, otherwise the iterations may converge to an indefinite solution of (5.1), if they converge at all. The iterative scheme embodied in (5.8) is precisely that which is obtained by applying Newton's method to solve (5.1), however Newton's method alone will not provide conditions that will insure monotonic convergence.

In addition to being monotonically convergent, the iterates V_k are also quadratically convergent.

The first step in the computation of the solution of (5.1) is to determine a matrix L_0 such that A_0 is positive definite. The following theorem provides a constructive solution [Armstrong].

Theorem

Let (A, C) be stabilizable. Then

$$D = C^T Z^+, \quad \text{with} \quad CC^T = (H)$$

is a stabilizing gain matrix where $Z = Z^T \geq 0$ satisfies

$$(A + \beta I_n)Z + Z(A + \beta I_n)^T = 2CC^T \tag{5.10}$$

$$\text{with } 0 \leq \beta \leq \|A\|$$

The superscript + denotes the matrix pseudoinverse used to accommodate the inverse and solution of non-square systems. It is defined as

$$Z^+ = (Z^T Z)^{-1} Z^T$$

The most efficient method, currently, for implementing the above theorem uses Schur reductions [Sima] and may be summarised as follows:

Let U be an orthogonal matrix such that $U^T AU$ is in Real Schur form. Denote

$$\tilde{A} = U^T AU, \quad \tilde{Z} = U^T ZU, \quad \tilde{C} = U^T C, \quad \tilde{W} = \tilde{C}\tilde{C}^T$$

Premultiplying (5.10) by U^T and U respectively, gives the following reduced Lyapunov matrix equation,

$$(\tilde{A} + \beta I_n)\tilde{Z} + \tilde{Z}(\tilde{A} + \beta I_n)^T = 2\tilde{W}$$

with

$$D = \tilde{C}^T U^T (U\tilde{Z}U^T)^+ = \tilde{C}^T \tilde{Z}^+ U^T = C^T U \tilde{Z} U^T$$

For $m \leq n$ an approximation for the operations count in determining a D by this method is $16n^3$. The locations required for storage are $5n^2$.

The next stage is to solve (5.8) and compute (5.9) repeatedly until convergence. The first is effected as shown in Section 1.4, by an orthogonal reduction of A_k to its Schur form using the QR algorithm and then back substituting to solve the transformed system and hence to find V_k. The operation count for this process is approximately $20n^3 + O(n)$ and the storage requirements are $5n^2$.

Computing (5.9) involves $5n^3$ operations and requires $6n^2$ memory locations.

Therefore, to solve (5.1) by a Newton method requires $9n^2$ memory locations, since some may be shared, and $(16 + 25m)n^3$ operations, where m is the number of iterations needed for (5.8) to converge.

[Hammarling] provides a detailed study of Newton's method and suggests how it may be used to find the Choleski factor of X without first finding X.

With respect to the stability of the method, the accuracy of the computed solution is limited by the conditioning of the original problem and the precision for the mathematical operations. Also, since by definition the algorithm is iteratively self-refining it is considered to be stable.

5.2.5 The Matrix Sign Function

A scalar sign function is defined as

$$f(\lambda_i) = \text{ sign } (\lambda_i) = \begin{cases} +1 \text{ if Real } (\lambda_i) > 0 \\ -1 \text{ if Real } (\lambda_i) < 0 \end{cases}$$

The corresponding matrix sign function of a matrix Z say, is then

$$\text{sign } (Z) = M \text{ sign } (J) M^{-1}$$

where J is the Jordan Canonical form of Z and M is the matrix of eigenvectors for the eigenvalues of J. J is such that $J = \text{ diag } (J_1, J_2, \ldots J_k)$ and J_i are the Jordan blocks for the eigenvalues λ_i of Z. If λ_i is a distinct non-repeated eigenvalue then $J_i = \lambda_i$, a scalar. If λ_i is repeated r times and there is only one independent eigenvector for λ_i, then J_i is of

dimension $r \times r$ with λ_i's on the diagonal and 1's on the superdiagonal. The number of J_i blocks in J is equal to the number of elementary divisors of $\lambda I - Z$, [Gantmacher]. Now, sign $(J) =$ diag (sign $(J_1), \ldots,$ sign (J_k)) and sign (J_i) is diagonal [Denman & Beaver]. [Roberts] shows that the sign function of a matrix can be defined in terms of a contour integral or as a result of an iterated map

$$S_{r+1} = \frac{1}{2}(S_r + S_r^{-1}) \qquad \text{with} \quad S_0 = Z, \text{ say} \tag{5.11}$$

The connection between the sign function and the ARE is given in the following analysis [Byers, 4].

Now (5.1) is equivalent to [Potter],

$$Z = \begin{bmatrix} A^T & H \\ G & -A \end{bmatrix} = \begin{bmatrix} X & -I_n \\ I_n & 0_n \end{bmatrix} \begin{bmatrix} -(A-GX) & -G \\ 0_n & (A-GX)^T \end{bmatrix} \begin{bmatrix} X & -I_n \\ I_n & 0_n \end{bmatrix}^{-1}$$

Applying the matrix sign function to this equation, and observing that $\lambda_i(A - FX) < 0$,

$$\text{sign}\,(Z) = W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} X & -I_n \\ I_n & 0_n \end{bmatrix} \begin{bmatrix} I_n & T \\ 0_n & -I_n \end{bmatrix} \begin{bmatrix} X & -I_n \\ I_n & 0_n \end{bmatrix}^{-1} \tag{5.12}$$

Since Z and sign (Z) commute, T satisfies the Lyapunov equation

$$(A - GX)T + T(A - GX)^T = 2G$$

Subtracting I_{2n} from both sides of (5.12) and multiplying out gives,

$$\begin{bmatrix} W_{11} - I_n \\ W_{21} \end{bmatrix} X = - \begin{bmatrix} W_{12} \\ W_{22} - I_n \end{bmatrix}$$

or

$$M\,X = -N \quad , \quad M, N \in \mathbb{R}^{2n \times n} \tag{5.13}$$

This is a full-rank, consistent system of $2n^2$ equations in the n^2 unknowns x_{ij} and solved by using least squares QR factorisation [Lawson & Hanson]. The solution to (5.13) is then also the solution of the Algebraic Riccati Equation.

Clearly, it is necessary to devise a numerically efficient and stable technique for determining an accurate sign function of the Hamiltonian and many convergent algorithms have been proposed, [Roberts], [Balzer], [Anderson]. A stability analysis for the matrix sign function reveals that all these algorithms are not numerically stable since they work

with a matrix that does not satisfy certain criterion [Byers, 3]. However, the matrix sign function is stable for symmetric matrices but few others. [Byers, 4] shows that by a simple reorganisation, the non-symmetric matrix may be changed into symmetric matrix inversions giving the following convergent iteration for sign $(Z) = W$,

$$\begin{aligned} W^{(0)} &= Z \\ T^{(k)} &= W^{(k)} |\det(W^{(k)})|^{-\frac{1}{2n}} \\ W^{(k-1)} &= T^{(k)} - \left[\frac{T^{(k)} - (JT^{(k)})^{-1}J}{2}\right] \end{aligned} \tag{5.14}$$

where

$$J = \begin{bmatrix} 0_n & I_n \\ -I_n & 0_n \end{bmatrix}$$

Since the desired solution X of (5.1) is symmetric, replace the solution $\hat{X}$ computed from (5.13) by $(\hat{X} + \hat{X}^T)/2$.

The problem of choosing a suitable stopping point for the iterations (5.14) is a non-trivial one and it has been observed that proposed criteria stop the iteration too early or too late and rounding errors may even prevent the criteria from being satisfied at all. For this reason, iterative refinement of the computed solution $\hat{X}$ is very necessary. Let

$$R = R(\hat{X}) = A^T\hat{X} + \hat{X}A - \hat{X}G\hat{X} + H$$

If $P = P^T \in \mathbb{R}^{n\times n}$ and P satisfies the ARE [Bierman],

$$R + (A - G\hat{X})^T P + P(A - G\hat{X}) - PGP = 0 \tag{5.15}$$

then $X = \hat{X} + P$ satisfies (5.1). Newton's method is used to solve (5.15) and rounding errors will yield a matrix $\hat{P}$, an approximation to P so that this refinement step may need repeating iteratively until convergence to within the specified tolerance is reached.

Therefore, the sign function method can be considered to be in a class of its own in that it combines the properties of both the invariant subspace determining methods and Newton's method. In fact, the iterates (5.14) in conjunction with (5.13) may be regarded simply as a way to obtain a good initial guess for Newton's method.

Analysis and test examples [Byers, 4] suggest that this sign function method with iterative refinement is a stable method for determining the solution of (5.1) and compares favorably with the Schur vector method with respect to work, storage and accuracy.

5.2.6 Discussion

We stated earlier that the Eigenvector method and the Schur vectors approach are unstable in that ill-conditioned eigenvalues may contaminate the solution computed by the Eigenvector method and an ill-conditioned matrix U_{11} will do likewise for the Schur vectors approach. However in practice, the eigenvalues of the Hamiltonian matrix arising in applications are generally well-conditioned [Hewer & Nazaroff], and in the very rare case that the matrix U_{11} is ill-conditioned, this condition can be detected and an alternative method used. This statement is backed up by the fact that the Eigenvector method and the Schur vectors method have been used extensively in real applications.

By definition, any problem that is solved by the eigenvector method can be solved more efficiently by the Schur vectors approach. Additionally, if the problem arises from a single input control system, the Hamiltonian-Schur approach would prove to be more efficient.

Newton's method is stable, yields a very accurate solution and is guaranteed to converge. A disadvantage of the method is that it requires more work than any of the methods above and in particular if the eigenvalues of the starting matrix are such that $\lambda_i(A - BL_0)$ are near zero then convergence will initially be slow.

The formulation of the sign function method discussed here is a very recent approach and all indications are that it compares favorably with the Schur vectors approach.

SECTION 5.3 : The Unilateral Quadratic Matrix Equation

Of interest here is the solution of the equation

$$X^2 + PX + Q = 0 \tag{5.16}$$

where all matrices are square and of order n.

5.3.1 Newton Iteration Method

This approach [Davis, 1] is based on the application of Newton's method to the matrix function $F(X)$,

$$F(X) = X^2 + PX + Q$$

After choosing an initial guess X_0, successive iterates are generated by the formula

$$X_{i+1} = X_i - T_i \qquad i = 0, 1, 2, \ldots \tag{5.17}$$

where T_i solves

$$F'(X)T_i = F(X_i) \tag{5.18}$$

Now, the derivative of $F(X)$ is an operator and given by

$$F'(X)H = (X + P)H + HX$$

such that (5.18) becomes

$$(X_i + P)T_i + T_iX_i = F(X_i) \qquad i = 0, 1, 2, \ldots \tag{5.19}$$

The steps of the algorithm with their respective operations count are now given [Davis, 2].

1. Choose a starting matrix X_0, [Davis, 2] proposes

$$X_0 = \left[\frac{\|P\| + \sqrt{\|P\|^2 + 4\|Q\|}}{2}\right] I$$

2. Compute

$$\begin{aligned} F(X_i) &= X_i^2 + PX_i + Q \\ &= (X_i + P)X_i + Q \end{aligned}$$

This involves n^3 operations.

3. Solve (5.19) for T_i. We showed in Section 1.4 that this may be achieved by a Hessenberg-Schur [Golub, Nash & Van Loan] approach which uses $20n^3$ operations.

4. Generate X_{i+1} from the formula (5.17). Goto 2. until convergence.

The process is terminated when $\|F(X_i)\|$, for some appropriate norm, is less than some specified tolerance.

The total operations count is $21kn^3 + O(n^2)$ where k is the number of iterations, and the storage requirements are $5n^2$ storage locations.

An error analysis of this method [Davis, 1] reveals that the algorithm does not introduce any errors that are not inherent in the problem itself. That is, the method is stable. Therefore, for a well-conditioned problem (5.16), this Newton method provides a solution whose accuracy is limited only by the condition of the original problem and by the precision of the arithmetic operations involved.

The method does have shortcomings however, in that it is not known beforehand if the starting matrix will yield a convergent sequence of iterates and if the Newton correction T_i is large then iterates will converge very slowly, if at all. [Kratz & Stickel] discuss the convergence properties of these Newton iterates. Also, in the course of the iterations a singular or ill-conditioned derivative will make the problem in (5.18) unstable. [Davis, 1] carried out experimental examples using constant well-conditioned derivative $F'(X_0)$ only to find that this approach is unreliable.

5.3.2 Matrix Polynomial Algorithms

This method [Dennis, Traub & Weber, 2] is a generalisation of an algorithm for scalar polynomials [Traub]. The algorithm is designed to determine a dominant solution of (5.16).

A matrix A dominates a matrix B if all the eigenvalues of A are greater, in modulus, than those of B. If the solution S_1 dominates S_2 then S_1 is said to be a dominant solution. The algorithm fails if no dominant solution exists.

The algorithm is in two stages:

Stage 1:

Let $T_1^{(0)} = I$ and $T_2^{(0} = 0$ then for $i = 0, 1, \ldots, \ell - 1$ for some ℓ

$$\begin{aligned} T_1^{(i+1)} &= T_2^{(i)} \quad -T_1^{(i)} P \\ T_2^{(i+1)} &= \quad -T_1^{(i)} Q \end{aligned} \tag{5.20}$$

Stage 2:

Let

$$X_0 = (T_1^{(\ell)})\,(T_1^{(\ell-1)})^{-1} \tag{5.21}$$

and generate X_{k+1} by

$$X_{k+1} = (T_1^{(\ell)} X_k + T_2^{(\ell)})\,(T_1^{(\ell-1)} X_k + T_2^{(\ell-1)})^{-1} \tag{5.22}$$

This algorithm is globally convergent in the sense that if ℓ is sufficiently large and $F(X)$ is such that

(i) it has solutions S_1, S_2

(ii) S_1 is a dominant solution

(iii) $|S_2 - S_1| \neq 0$ and $|S_2| \neq 0$

then the iterates of Stage 2 are globally convergent to a solution of (5.16).

The operations count for Stage 1 is $2\ell n^3$ and for Stage 2, $\frac{4}{3}n^3 + \frac{10}{3}kn^3$ making a total of about $\left(\frac{2\ell+10k}{3}\right)n^3$ operations where ℓ and k are the number of iterations for the first and second stages respectively. The storage requirements are $6n^2$ storage locations.

To our knowledge there is no documented analysis what follows for the Matrix Polynomial algorithm so that this is an original critique of the method.

The expressions in (5.20) may be transposed and written as

$$\begin{aligned} T_2^{(i+1)} &= -QT_1^{(i)} \\ T_1^{(i+1)} &= T_2^{(i)} - PT_1^{(i)} \end{aligned} \tag{5.23}$$

where the matrices are understood to be the transposes of those in (5.20). (5.23) may be represented as

$$\begin{bmatrix} T_2^{(i+1)} \\ T_1^{(i+1)} \end{bmatrix} = \begin{bmatrix} 0 & -Q \\ I & -P \end{bmatrix} \begin{bmatrix} T_2^{(i)} \\ T_1^{(i)} \end{bmatrix}$$

or

$$T^{(i+1)} = A\,T^{(i)} \tag{5.24}$$

where A is the companion matrix of the quadratic eigenvalue problem associated with (5.16). Corresponding to the requirements in (5.21) for the transposed equation, we have

$$X_0 = (T_1^{(\ell-1)})^{-1}\,T_1^{(\ell)}$$

or

$$T_1^{(\ell-1)}\,X_0 = T_1^{(\ell)} \tag{5.25}$$

From (5.25)

$$T^{(\ell)} = \begin{bmatrix} T_2^{(\ell)} \\ T_1^{(\ell)} \end{bmatrix} = A\ T^{(\ell-1)} = A^\ell T_0 = A^\ell \begin{bmatrix} 0 \\ I \end{bmatrix}$$

Partition A^ℓ as

$$\left[\begin{array}{c|c} A_{11}^\ell & A_{12}^\ell \\ \hline A_{21}^\ell & A_{22}^\ell \end{array}\right] \tag{5.26}$$

then from (5.27) and (5.23)

$$A_{12}^{(\ell)} = T_2^{(\ell)} = -Q\ T_1^{(\ell-1)} \tag{5.27}$$

and

$$A_{22}^{(\ell)} = T_1^{(\ell)} \tag{5.28}$$

We would like to find expressions for the A_{ij} in (5.26) and we progress in a similar way to the analysis of the Elimination method in Section (4.2). From (4.15),

$$\begin{bmatrix} 0 & -Q \\ I & -P \end{bmatrix} = \begin{bmatrix} 0 & -Q \\ I & X \end{bmatrix} \begin{bmatrix} -(X+P) & 0 \\ I & -X \end{bmatrix} \begin{bmatrix} 0 & -Q \\ I & X \end{bmatrix}^{-1}$$

or

$$A = Z\ B\ Z^{-1} \tag{5.29}$$

Let

$$f(A) = A^\ell$$

then

$$\begin{aligned} f(A) &= Z\ f(B) Z^{-1} \\ &= Z \begin{bmatrix} f(-X-P) & 0 \\ f_\beta & f(X) \end{bmatrix} Z^{-1} \end{aligned} \tag{5.30}$$

Now, $B\ f(B) = f(B)\ B$

$$\begin{bmatrix} -(X+P) & 0 \\ I & -X \end{bmatrix} \begin{bmatrix} f(-X-P) & 0 \\ f_\beta & f(X) \end{bmatrix} =$$

$$\begin{bmatrix} f(-X-P) & 0 \\ f_\beta & f(X) \end{bmatrix} \begin{bmatrix} -(-X-P) & 0 \\ I & -X \end{bmatrix}$$

Multiplying out gives,

$$X f_\beta + f_\beta\ (X+P) = f(X) - f(-X-P)$$

Since we are still dealing with transposes and using the definition of the derivative operator introduced in Section 2.1, we have

$$[F'(X)\ f_\beta]^T = f(X) - f(-X - P) \tag{5.31}$$

which will be used later.

Now multiplying out (5.30) gives an expression similar to that in (4.21),

$$f(A) = \begin{bmatrix} Q(f(X) - f_\beta X)Q^{-1} & -Qf_\beta \\ f_\beta & f(-X-P) + Xf_\beta \end{bmatrix} \tag{5.32}$$

Comparing (5.26) with (5.32) and noting that $f(A) = A^\ell$,

$$A_{12}^{(\ell)} = -Qf_\beta$$

$$A_{22}^{(\ell)} = f(-X - P) + Xf_\beta$$

From (5.27), (5.28),

$$A_{12}^{(\ell)} = -QT_1^{(\ell-1)} = -Qf_\beta$$

$$A_{22}^{(\ell)} = T_1^{(\ell)} = f(-X - P) + Xf_\beta$$

giving

$$T_1^{(\ell-1)} = f_\beta \tag{5.33}$$

Now at stage 2 of the algorithm, it is necessary to solve (5.25) and it is well known that the condition of this linear system is related to the condition of $T_1^{(\ell-1)}$ and therefore f_β, from (5.33). Multiplying both sides of (5.31) by $(F'(X)^{-1})^T$ and taking norms, gives

$$\|T_1^{(\ell-1)}\| = \|f_\beta\| \leq \|F'(X)^{-1}\|\ \ \|f(X) - f(-X - P)\| \tag{5.34}$$

This bound implies that if the original problem is ill-conditioned then $T_1^{(\ell-1)}$ may also be ill-conditioned so that the solution of (5.25) will then not be accurate.

Additionally, investigation is required into the condition of the linear system in (5.22) and into the possibility of rounding errors contaminating the computed solution.

5.3.3 Discussion

Newton's method is stable and for a well-conditioned problem, it will yield a very accurate solution. There does not seem to be any stability analysis for the Matrix Polynomial algorithm although test examples have shown it too produces very accurate solutions [Dennis, Traub & Weber, 2]. We have shown that the condition of a linear system within the algorithm is dependent on the condition of the original problem.

Each method has a significant limitation. Newton's method requires for convergence, a starting matrix that is 'close enough' to the solution matrix, and the Matrix Polynomial algorithm will work only for problems that possess a dominant solution. In practice, neither of these shortcomings can be detected before attempting to solve the problem and therefore the iterations should be terminated when divergence is assumed to be occurring.

Additionally for the Newton's method, if an ill-conditioned derivative matrix is encountered at any iteration then the method as described here cannot continue.

From the operation count point of view, the Matrix Polynomial method requires much less work per iteration.

SECTION 5.4 : The Matrix Square Root Equation

Of interest here is the square root X of the matrix A, such that

$$X^2 - A = 0 \tag{5.35}$$

5.4.1 Newton's Method

If X^r is an approximation to X then

$$X = X_r + E$$

Substitute this into (5.35) to give

$$X_r^2 + X_r E + E X_r - A = 0$$

which gives the Newton iteration

$$X_r E_r + E_r X_r = A - X_r^2 \tag{5.36}$$

or

$$F'(X_r)E_r = A - X_r^2$$

with the update

$$X_{r+1} = X_r + E_r \qquad \text{for } r = 0, 1, \ldots$$

Several methods start with an initial approximation X_0 that commutes with A and instead of (5.36) use the iteration

$$E_r X_r = \frac{1}{2}(A - X_r^2) \qquad r = 0, 1, 2, \ldots \tag{5.37}$$

so that X_r will then commute with A for $r \geq 1$. However due to instability and rounding errors in (5.37), (5.36) is the preferred iteration. Note that (5.36) is the Lyapunov Matrix Equation.

The iterations are stable and they converge quadratically to the square root of A if the initial matrix X_0 is sufficiently 'close' to the solution X and $F'(X_r)$ is non-singular at each iteration.

The operations count is $14kn^3 + O(n^2)$ operations where k is the number of iterations and the storage requirements is $3n^2$ locations.

5.4.2 Sign Function Method

The problem of computing a square root of a matrix can be stated in the form of a degenerate ARE, [Denman & Beavers],

$$A - XIX = 0$$

with

$$Z = \begin{bmatrix} 0 & A \\ I & 0 \end{bmatrix}$$

The iterated map (5.11), that defines the matrix sign function gives rise to the following iteration,

let $R_0 = A, \quad S_0 = I$

then the sequence

$$\begin{aligned} R_{k+1} &= \frac{1}{2}\,(R_k + S_k^{-1}) \\ S_{k+1} &= \frac{1}{2}\,(S_k + R_k^{-1}), \qquad k = 0, 1, 2, \ldots \end{aligned} \tag{5.38}$$

converges, such that

$$\lim_{k->\infty} R_{k+1} = X = A^{\frac{1}{2}}$$

and

$$\lim_{k->\infty} S_{k+1} = X^{-1}$$

for when A has no root that is real and negative.

The operation count is $\frac{8}{3}\,n^3$ per iteration and the storage requirements are $6n^2$.

[Higham, 4] shows through analysis and examples that the iterations are stable.

The convergence of the iterates in (5.38) may be slow if the spectrum of A is large. This leads to the following variant [Hoskins & Walton].

Let

$$R_0 = A \qquad S_0 = I$$

then

$$\begin{aligned} R_{k+1} &= \alpha_k R_k + \beta_k S_k^{-1} \\ S_{k+1} &= \alpha_k S_k + \beta_k R_k^{-1} \qquad k = 0, 1, 2, \ldots \end{aligned}$$

where

$$\alpha_k^2 = \frac{2}{b_k + B_k + 6\sqrt{b_k B_k}}$$

$$\beta_k^2 = b_k B_k \alpha_k^2$$

where

$$b_k = 1 - \epsilon_{k-1}$$

$$B_k = 1 + \epsilon_{k-1}$$

and

$$\epsilon_k = 1 - 4\alpha_k \beta_k$$

where $b_0 = \frac{1}{\|A^{-1}\|}$ and $B_0 = \|A\|$.

5.4.3 Schur Vectors Approach

This method [Björck & Hammarling] computes the Schur factorisation of A such that

$$A = QSQ^T$$

where Q is orthogonal and S is an upper triangular matrix with at most one zero diagonal element. Then the square root of A is given by

$$X = QUQ^T$$

where

$$U^2 = S$$

such that

$$s_{ij} = \sum_{k=1}^{j} u_{ik}\, u_{kj}\ , \qquad i \leq j$$

and hence

$$u_{ii} = s_{ii}^{\frac{1}{2}}\ , \qquad i = 1, 2, \ldots, n$$

and

$$u_{ij} = \frac{s_{ij} - \sum_{k=i+1}^{j-1} u_{ik} u_{kj}}{u_{ii} + u_{jj}}, \qquad i < j$$

Therefore the elements of U are computed one superdiagonal at a time starting with the leading diagonal and working upwards.

If whenever $s_{ii} = s_{jj}$, $u_{ii} = u_{jj}$ is chosen then since S has at most one zero diagonal element, it is assured that

$$u_{ii} + u_{jj} \neq 0$$

and hence u_{ij} is defined.

This method requires:

$15n^3$ operations to determine the orthogonal matrix Q,

a total of

$$2 \sum_{j=1}^{n} \sum_{i=1}^{j=1} \sum_{k=i+1}^{j-1} 1 + n = \frac{n^3}{3} - n^2 + \frac{5n}{3} \quad \text{(using results from Section 1.2)}$$

operations to obtain U,

$2n^3$ operations to compute X,

so that the overall operations count is $18n^3$ and the storage requirements are $4n^2$ locations.

[Björck & Hammarling] state that for a symmetric matrix A the Schur method is numerically stable. For general matrices the method is stable if the norm $\|X\|^2/\|A\|$ is small.

However, the method may produce an ill-conditioned square root even where a well-conditioned square root exists. Although this is an extremely rare case, [Björck & Hammarling] attempt to address this problem.

[Higham, 3] addresses and resolves a disadvantage of the Schur method, that is, if A is real and has no real eigenvalues, the Schur method necessitates complex arithmetic even if the computed square root is real. However, this technique is applicable only for real square roots that are functions in A. The following represent some of the important issues in [Higham, 1].

The matrix A has a real square root if and only if each elementary divisor of A corresponding to a real negative eigenvalue occurs an even number of times.

The square roots of A which are functions of A are 'isolated' square roots, characterised by the fact that the sum of any two of their eigenvalues is non-zero.

If A has a real negative eigenvalue, then A has no real square roots that are functions of A.

The operations count for the entire algorithm is around $17n^3$. The real Schur method is stable provided that $\|X\|^2/\|A\|$ is sufficiently small.

5.4.4 Discussion

All the methods of this section when successful, provide accurate solutions to (5.35). In particular, the iterative methods can be repeated to refine the solution. It is assumed that the problem of determining the square root is well-conditioned. The matrix sign function method fails if A posseses a real negative eigenvalue. The Newton method requires a well-conditioned derivative at each iteration and a suitable starting point. The Schur vectors method will not yield reliable solutions in the unlikely event that $\|X\|^2/\|A\|$ is large. If the matrix sign function method requires less than 8 iterations, it will compare favorably with the Schur method, otherwise it will be slower. For the Newton method to be as fast, it will require to converge in 2 iterations, which is highly unlikely.

The Schur vectors method for real square roots is useful if it is known that a matrix possesses real square roots and if the requirement is to determine a root that is a function in A. Similarly, if we required the square root of a symmetric, positive definite matrix then more efficient methods are available, e.g. [Higham, 5].

SECTION 5.5 : Minimisation Of The Constituent Equation

In this section, we look at the problem of solving

$$F(X) = 0 \tag{5.39}$$

where

$$F(X) = \begin{cases} X^2 - A \\ X^2 + PX + Q \\ A^T X + XA - XGX + H \end{cases} \tag{5.40}$$

by transforming it into a sum of squares problem and minimising the resultant scalar function.

Denote the elements of $F(X)$ by F_{ij} and form a n^2-vector, f_i, from the rows of F,

$$f_i = (F_{11}, F_{12}, \ldots, F_{1n}, F_{21}, \ldots, F_{nn})^T, \quad i = 1, n^2 \tag{5.41}$$

Similarly,

$$y_i = (x_{11}, x_{12}, \ldots, x_{1n}, x_{21} \ldots, x_{nn})^T, \quad i = 1, n^2 \tag{5.42}$$

The scalar function that we minimise is then,

$$fc = \frac{1}{2} \sum_{i=1}^{n^2} f_i^2$$

where the $\frac{1}{2}$ is included for convenience.

In Section 1.5, we discussed how Newton's method for unconstrained minimisation may be used for this problem. To use the algorithm developed in that section, we require to find the Jacobian and Hessian matrices of $\{f_i\}$ for each equation in (5.40). Section 2.1 derived expressions for the Jacobians by using Kronecker products. We begin by re-stating the algorithm of Section 1.5,

(i) Select $y^{(0)} \in I\!R^{n^2}$, the initial estimates to y^*.

(ii) Determine whether the Gauss-Newton or Newton iteration is to be used.

Gauss-Newton

(iii) Determine the Jacobian at $y^{(k)}$.

(iv) If the Jacobian is singular, add a scalar matrix $\mu^{(k)} I$ to $J^T J$ such that $J^T J + \mu^{(k)} I$ is safely non-singular, and solve

$$(J^T J + \mu^{(k)} I)\, p^{(k)} = -J^T f$$

otherwise solve

$$J^T J p^{(k)} = -J^T f \qquad \text{for } p^{(k)}$$

Newton

(iii) Determine the Jacobian, the gradient and the Hessian, at $y^{(k)}$.

(iv) If the Hessian M, is not positive definite, add a scalar matrix $\mu^{(k)} I$ such that $(M + \mu^{(k)} I)$ is positive definite and solve

$$(M + \mu^{(k)} I)\;\; p^{(k)} = -J^T f$$

otherwise solve

$$M\; p^{(k)} = -J^T f$$

(v) Minimise $fc(y^{(k)} + \alpha^{(k)} p^{(k)})$ with respect to $\alpha^{(k)} \in I\!R$.

(vi) Update, $y^{(k+1)} = y^{(k)} + \alpha^{(k)} p^{(k)}$.

(vii) If convergence criterion is not met, go to (ii).

Section 1.5 discussed steps (ii), (iv)-(vii) of this algorithm. At step (i) the initial matrix is chosen arbitrary and in fact $X^{(0)} = I$ is as good as any, such that from (5.42)

$$y_j^{(0)} = \begin{cases} 1 & \text{if } j = (i-1)n + i \quad \text{for } i = 1, \ldots, n \\ 0 & \text{otherwise} \end{cases}$$

At step (ii), if the sum of the squares was sufficiently reduced during the last iteration, then the Gauss-Newton iteration is used, otherwise the Newton iteration [Gill, Murray].

At step (iii) we require explicit expressions that give the Jacobian and Hessian matrices in terms of x_{ij}, and hence y_k, since

$$y_{(i-1)n+j} = x_{ij}$$

Quadratic Matrix Equation

From Section 2.1,

$$J = (X + P) \otimes I + I \otimes X^T \tag{5.43}$$

which is non-singular if $\lambda_i(X+P) \neq -\lambda_i(X)$

The elements of J are such that $J = J_1 + J_2$, where

$$J_1((i-1)*n+k,\ \ (j-1)*n+k) = x_{ij} + p_{ij}$$

$$J_2((k-1)*n+i,\ \ (k-1)*n+j) = x_{ji},\ i,j,k = 1,\ldots,n$$

The gradient is given by,

$$\begin{aligned} g = J^T f &= [(X+P)\otimes I + I \otimes X^T]^T f \\ &= [(X+P)^T \otimes I + I \otimes X] f \end{aligned}$$

Forming a matrix $G \in \mathbb{R}^{n\times n}$ from the elements of g and using (5.41),

$$G = (X+P)^T F + F X^T$$

Hence

$$g_{(i-1)n+j} = G_{ij} = \sum_k (x_{ki} + p_{ki}) F_{kj} + \sum_k F_{ik} x_{jk} \tag{5.44}$$

The Gauss-Newton iteration requires $J^T J$ if J is singular,

$$\begin{aligned} J^T J &= ((X+P)\otimes I + I \otimes X^T)^T ((X+P)\otimes I + I \otimes X^T) \\ &= ((X+P)^T \otimes I + I \otimes X)((X+P)\otimes I + I \otimes X^T) \\ &= (X+P)^T (X+P) \otimes I + (X+P)^T \otimes X^T + (X+P) \otimes X + I \otimes X X^T \end{aligned} \tag{5.45}$$

The Newton iteration uses the Hessian H. From (1.27)

$$\begin{aligned} M = (m_{ij}) &= J^T J + \sum_{k=1}^{n^2} f_k \frac{\partial^2 f_k}{\partial y_j \partial y_i} \\ &= J^T J + Z, \quad \text{say} \end{aligned}$$

where $Z = Z_1 + Z_1^T$ and

$$Z_1 = \left[\begin{array}{ccc|ccc|c|ccc}
F_{11} & F_{12} & F_{1n} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & 0 & F_{11} & F_{12} & F_{1n} & & & & \\
\vdots & & & \vdots & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & & E_{11} & & F_{12} \\
\hline
F_{21} & F_{22} & F_{2n} & 0 & 0 & 0 & & & & \\
0 & 0 & 0 & F_{21} & F_{22} & F_{2n} & & & & \\
\vdots & & & \vdots & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & & & & \\
\hline
\vdots & & & & & & & & & \\
F_{n1} & F_{n2} & F_{nn} & \cdots & & & \cdots & 0 & 0 & 0 \\
\vdots & & & & & & & & & \\
0 & 0 & 0 & & & & & E_{n1} & F_{n2} & F_{nn}
\end{array}\right] \tag{5.46}$$

such that

$$Z_1((k-1)*n+j,\ (i-1)*n+k) = F\,(i,j) \quad i,j,\ k = 1,\ldots,n \tag{5.47}$$

Therefore, for the quadratic matrix equation, to form:

J	requires $3n^2$	operations
$J^T f$	requires $3n^3$	operations
$J^T J$	requires $n^4 + 4n^3$	operations
f	requires n^3	operations
Z	requires $4n^2$	operations
M	requires $n^4 + 5n^3 + 4n^2$	operations

For the matrix square root problem, the Jacobian is given by

$$J = X \otimes I + I \otimes X^T$$

such that J is non-singular if $\lambda_i \neq -\lambda_j$ for all i,j where λ are the eigenvalues of X. The gradient is given by

$$g_{(i-1)n+j} = \sum_k x_{ki} F_{kj} + \sum_k F_{ik} x_{jk}$$

and

$$J^T J = X^T X \otimes I + I \otimes X X^T + X^T \otimes X^T + X \otimes X$$

The Hessian is given by

$$M = J^T J + \sum_{k=1}^{n^2} f_k \frac{\partial^2 f_k}{\partial y_j \partial y_i}$$

$$= J^T J + Z$$

where $Z = Z_1 + Z_1^T$ and Z_1 is given by (5.46).

Algebraic Riccati Equation

$$F(X) = A^T X + XA - XGX + H = 0$$

$$F_{ij} = \sum_{1}^{n} a_{ki} x_{kj} + \sum_{1}^{n} x_{ik} \left(a_{kj} - \sum_{\ell=1}^{n} g_{k\ell} x_{\ell j}\right) + h_{ij}$$

The Jacobian is given by

$$J = (A^T - XG) \otimes I + I \otimes (A - GX)^T \tag{5.48}$$

Let

$$u_{ij} = \sum_{k=1}^{n} x_{ik} g_{kj} \text{ and } v_{ij} = \sum_{k=1}^{n} g_{ik} x_{kj}$$

then $J = J_1 + J_2$, where

$$J_1((i-1)*n+k,\ (j-1)*n+k) = a_{ji} - u_{ij}$$

and

$$J_2((k-1)*n+j,\ (k-1)*n+i) = a_{ij} - v_{ij}$$

The gradient g is given by

$$g = J^T f = [(A^T - XG) \otimes I + I \otimes (A - GX)^T]^T f$$

$$= [(A - GX^T) \otimes I + I \otimes (A - GX)] f, \quad \text{since } G = G^T$$

then

$$g_{(i-1)n+j} = \sum_{k} (a_{ik} - \sum_{\ell} G_\ell x_{k\ell}) F_{kj} + \sum_{k} F_{ik} \left(a_{jk} - \sum_{\ell} G_{\ell k}\ x_{j\ell}\right)$$

Let $C = A - GX$, then

$$J^T J = CC^T \otimes I + I \otimes CC^T + C^T \otimes C + C \otimes C^T$$

The Hessian is given by

$$M = J^T J + Z$$

where Z is of the form

$$\begin{bmatrix}
g_{11}f_{11}+g_{11}f_{11}\cdots g_{11}f_{1n}+g_{n1}g_{11}\cdots\cdots g_{1n}f_{11}+g_{11}f_{n1}\cdots g_{1n}f_{1n}+g_{n1}f_{n1} \\
\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
g_{n1}f_{11}+g_{n1}f_{1n}\cdots g_{n1}f_{1n}+g_{n1}f_{1n}\cdots\cdots g_{nn}f_{11}+g_{11}f_{nn}\cdots g_{nn}f_{1n}+g_{n1}f_{nn} \\
g_{11}f_{21}+g_{12}f_{11}\cdots g_{11}f_{2n}+g_{n2}f_{11}\cdots\cdots g_{1n}f_{21}+g_{12}f_{n1}\cdots g_{1n}f_{2n}+g_{n2}f_{n1} \\
\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
g_{n1}f_{21}+g_{12}f_{1n}\cdots g_{n1}f_{2n}+g_{n2}f_{1n}\cdots\cdots g_{nn}f_{21}+g_{12}f_{nn}\cdots g_{nn}f_{2n}+g_{n2}f_{nn} \\
\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
g_{11}f_{n1}+g_{1n}f_{11}\cdots g_{11}f_{nn}+g_{nn}f_{11}\cdots\cdots g_{1n}f_{n1}+g_{1n}f_{n1}\cdots g_{n1}f_{nn}+g_{nn}f_{n1} \\
\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
g_{n1}f_{n1}+g_{1n}f_{1n}\cdots g_{n1}f_{nn}+g_{nn}f_{1n}\cdots\cdots g_{nn}f_{n1}+g_{1n}f_{nn}\cdots g_{nn}f_{nn}+g_{nn}f_{nn}
\end{bmatrix}$$

To form:

J	requires $2n^3+3n^2$	operations
J^Tf	requires $7n^3$	operations
J^TJ	requires n^4+6n^3	operations
f	requires $3n^3$	operations
Z	requires $2n^4+3n^3+4n^2$	operations
M	requires $3n^4+9n^3+4n^2$	operations

To give some idea of the magnitudes involved, we give the operations count for this algorithm when the iterations are all Gauss-Newton or all Newton. Remember from Section 1.5 that the solution of the linear system is performed by using the Choleski decomposition [Golub & Van Loan] and the other steps in the algorithm use $O(n^2)$ operations.

(per iteration)	**Gauss-Newton**	**Newton**
unilateral equation	$\frac{n^6}{6}+n^4+7n^3+O(n^2)$	$\frac{n^6}{6}+n^4+7n^3+O(n^2)$
Riccati equation	$\frac{n^6}{6}+n^4+13n^3+O(n^2)$	$\frac{n^6}{6}+3n^4+16n^3+O(n^2)$

excluding the operations required to perform the line search.

For the unilateral equation, there is no advantage in using the Gauss-Newton iterations over the Newton iterations. In fact, as $X^{(k)}$ approaches X^*, it is clear from the expression for Z in (5.46) that the Gauss-Newton and Newton iterations approach equality.

For the ARE there is a saving of $2n^4$ at each iteration in using the Gauss-Newton rather than the Newton iteration. For small order problems this is significant.

Clearly, for large order problems the term $O(n^6)$ is dominant and this is the reason why minimisation methods are not favoured as numerical algorithms, when others are available. However, leaving aside the Elimination method for the moment, we have discovered that there are cases where the methods for the unilateral quadratic matrix equation fail, in which case the minimisation approach of this section would be the only alternative.

CHAPTER 6 - PRACTICAL METHODS FOR THE QUADRATIC MATRIX EQUATIONS BASED ON NEWTON'S METHOD

SECTION 6.1: Introduction

This chapter discusses two new approaches to the solution of the matrix equations.

In the first case, we show how an efficient reformulation of Newton's method for a system of non-linear equations may be used to solve the Riccati equation,

$$F(X) = A^T X + XA - XGX + H = 0 \tag{6.1}$$

The approach is significant in that the computational labour usually associated with the Newton's method for (6.1) is significantly reduced by observing the sparse nature of the associated Jacobian matrix. Also, from Section 1.5, it is known that the Newton iterates have a local quadratic convergence property from a good starting point.

If y_i and f_i, $i = 1, n^2$ are vectors formed from the rows of X and F respectively, then the Newton iterates are,

$$y^{(k+1)} = y^{(k)} + p^{(k)}$$

where $p^{(k)}$ solves,

$$J(y^{(k)})p^{(k)} = -f^{(k)} \tag{6.2}$$

We have already shown in Section 5.5 that the Jacobian J can be determined with relatively little computational effort. The problem of solving (6.2) for $p^{(k)}$ is more difficult and is treated in the next section. In that section, an algorithm for this approach is presented along with an operations count and an error analysis.

In Section 6.3, it is shown how a matrix analogy of the sum of squares minimisation method can be used to solve the unilateral quadratic matrix equation,

$$F(X) = X^2 + PX + Q = 0 \tag{6.3}$$

We show how the restrictions imposed by the methods of Sections 5.3 and 5.5, namely,

failure to converge from an arbitrary starting point for the Newton method,

failure to converge for problems not possessing a dominant solution for the Matrix Polynomial method,

large amount of computational labour required for the minimisation method,

solution of a possibly ill-conditioned linear system,

may be overcome, towards providing an efficient, stable and globally convergent method. Section 6.3 provides an analysis and the description of the algorithm along with an operations count.

SECTION 6.2: Newton's Method for the Algebraic Riccati Equation

The Jacobian matrix associated with (6.1) was determined in Chapter 2 as being,

$$J = (A^T - XG) \otimes I + I \otimes (A - GX)^T$$

Now, in control applications, the coefficient matrices of the algebraic Riccati equation possess certain properties,

$$G = G^T > 0, \quad H = H^T \geq 0$$

and the solution usually of interest is the non-negative symmetric one, such that

$$X = X^T \geq 0$$

Under these conditions the Jacobian may be written as

$$J = (A^T - XG) \otimes I + I \otimes (A^T - XG)$$

Let

$$T = A^T - XG$$

such that

$$J = T \otimes I + I \otimes T \tag{6.4}$$

Now there always exists an orthogonal $U \in \mathbb{R}^{n \times n}$ such that

$$U^T T U = S = \begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ 0 & R_{22} & & R_{2m} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & R_{mm} \end{pmatrix} \tag{6.5}$$

where each R_{ii} is either a 1-by-1 matrix or a 2-by-2 matrix having complex conjugate eigenvalues.

Premultiply and postmultiply (6.4) by $U^T \otimes U^T$ and $U \otimes U$ respectively,

$$(U^T \otimes U^T)J(U \otimes U) = (U^T \otimes U^T)(T \otimes I)(U \otimes U) + (U^T \otimes U^T)(I \otimes T)(U \otimes U)$$

since $(A \otimes B)(C \otimes D) = AC \otimes BD$, the above equation becomes

$$(U^T \otimes U^T)J(U \otimes U) = U^T T U \otimes U^T U + U^T U \otimes U^T T U$$

and since U is orthogonal, $U^T U = I$ and from (6.5),

$$(U^T \otimes U^T)J(U \otimes U) = S \otimes I + I \otimes S \tag{6.6}$$

where the right hand side of (6.6), say $\tilde{J}$, is of the form

$$\left(\begin{array}{cccc|cccc|c|cccc}
2s_{11} & s_{12} & \cdots & s_{1n} & s_{12} & 0 & \cdots & 0 & & s_{1n} & 0 & \cdots & 0 \\
s_{21} & s_{11}+s_{22} & \cdots & s_{2n} & 0 & s_{12} & \cdots & 0 & & 0 & s_{1n} & \cdots & 0 \\
0 & s_{32} & \cdots & s_{3n} & \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\
0 & 0 & \cdots & s_{11}+s_{nn} & 0 & 0 & \cdots & s_{12} & & 0 & 0 & \cdots & s_{1n} \\
\hline
s_{21} & 0 & \cdots & 0 & s_{22}+s_{11} & s_{12} & \cdots & s_{1n} & & \vdots & \vdots & & \vdots \\
0 & s_{21} & \cdots & 0 & s_{21} & s_{22}+s_{22} & \cdots & s_{2n} & & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & & \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\
0 & 0 & \cdots & s_{21} & 0 & 0 & \cdots & s_{22}+s_{nn} & & \vdots & \vdots & & \vdots \\
\hline
\cdot & \cdot & & \cdot & & & & & \cdot & & & & \cdot \\
\cdot & \cdot & & \cdot & & & & & \cdot & & & & \cdot \\
\cdot & \cdot & & \cdot & & & & & \cdot & & & & \cdot \\
\cdot & \cdot & & \cdot & & & & & \cdot & & & & \cdot \\
\hline
\cdot & \cdot & & \cdot & & & & & s_{n,n-1} & 0 & \cdots & 0 & s_{nn}+s_{11} & s_{12} & \cdots & s_{1n} \\
\cdot & \cdot & & \cdot & & & & & 0 & s_{n,n-1} & \cdots & 0 & s_{21} & s_{nn}+s_{22} & \cdots & s_{2n} \\
\cdot & \cdot & & \cdot & & & & & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\
\cdot & \cdot & & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & 0 & \cdots & s_{n,n-1} & 0 & 0 & \cdots & s_{nn}+s_{nn}
\end{array}\right) \tag{6.7}$$

In partitioned form,

$$\tilde{J} = \begin{pmatrix}
s_{11}I + S & s_{12}I & \cdots & s_{1n}I \\
s_{21}I & s_{22}I + S & \cdots & s_{2n}I \\
0 & s_{32}I & \cdots & s_{3n}I \\
\vdots & \vdots & & \\
0 & 0 & & s_{n-1,n}I \\
0 & 0 & s_{n,n-1}I & s_{nn}I + S
\end{pmatrix} \tag{6.8}$$

where the subdiagonal elements of S satisfy;

$$\begin{aligned}
& s_{j+1,j} \neq 0 \quad \text{and} \quad s_{j+2,\,j+1} = 0 && \text{(i)} \\
\text{or} \quad & s_{j+1,j} = 0 \quad \text{and} \quad s_{j+2,\,j+1} \neq 0 && \text{(ii)} \\
\text{or} \quad & s_{j+1,j} = 0 \quad \text{and} \quad s_{j+2,\,j+1} = 0 && \text{(iii)}
\end{aligned} \tag{6.9}$$

(i) implies complex conjugate eigenvalues of T associated with the block

$$\begin{pmatrix} s_{jj} & s_{j,j+1} \\ s_{j+1,j} & s_{j+1,\,j+1} \end{pmatrix}$$

(ii) implies complex conjugate eigenvalues of T associated with the block

$$\begin{pmatrix} s_{j+1,\,j+1} & s_{j+1,j+2} \\ s_{j+2,\,j+1} & s_{j+2,\,j+2} \end{pmatrix}$$

(iii) implies T has a real eigenvalue $s_{j+1,\,j+1}$

Now, if $y \in \mathbb{R}^{n^2}$ and $f \in \mathbb{R}^{n^2}$ are formed from the rows of X and $F(X)$ respectively, the iterates

$$y^{(r+1)} = y^{(r)} + p^{(r)}$$

converge to a solution of (6.1) where $p^{(r)}$ solves

$$J^{(r)}p^{(r)} = -f^{(r)} \tag{6.10}$$

J being defined by (6.4).

Now, since the iterative algorithm is prohibitively expensive due to the solution of the linear system in (6.10) make the following transformation,

Premultiply (6.10) by $U^T \otimes U^T$,

$$(U^T \otimes U^T)J^{(r)}p^{(r)} = -(U^T \otimes U^T)f^{(r)}$$

since $(U \otimes U)(U^T \otimes U^T) = I$

$$(U^T \otimes U^T)J^{(r)}(U \otimes U)(U^T \otimes U^T)p^{(r)} = -(U^T \otimes U^T)f^{(r)}$$

Let $\tilde{p}^{(r)} = (U^T \otimes U^T)p^{(r)}$ and $\tilde{f}^{(r)} = -(U^T \otimes U^T)f^{(r)}$, and using (6.6), (6.10) becomes

$$\tilde{J}^{(r)}\tilde{p}^{(r)} = \tilde{f}^{(r)} \tag{6.11}$$

where $\tilde{J}^{(r)} = S \otimes I + I \otimes S$

To develop a strategy for solving (6.11), we observe the sparsity and the special form of the $\tilde{J}$ given by (6.8). We can use a block back substitution type of technique which is dependent on the subdiagonal elements of S. If we consider the technique to be composed of n phases, since there are n blocks, then the processing at the k^{th} phase will depend on which of the conditions in (6.9) is satisfied.

(a) **if (iii) of (6.9) is satisfied**, then we attempt to solve,

$$[s_{n-k,n-k}I + s \quad s_{n-k,n-k+1}I \quad \cdots \quad s_{n-k,n}I]\tilde{\mathbf{p}}_\gamma = \tilde{\mathbf{f}}_\gamma \tag{6.12}$$

for the first n elements of $\tilde{\mathbf{p}}_\gamma$, where

$$\tilde{\mathbf{p}}_\gamma = [\tilde{p}_{(n-k-1)n+1}, \cdots, \tilde{p}_{n^2}]^T$$

and

$$\tilde{\mathbf{f}}_\gamma = [\tilde{f}_{(n-k-1)n+1}, \cdots, \tilde{f}_{(n-k_n}]^T$$

Now since by this stage, $\tilde{p}_{(n-k)n+1}, \ldots, \tilde{p}_{n^2}$ are already known, we can update the right hand side of (6.12), algorithmically, as follows,

$$\tilde{f}_{n^2-(k+1)n+j} = \tilde{f}_{n^2-(k+1)n+j} - \sum_{i=0}^{k-1} s_{n-k,n-i}\tilde{p}_{n^2-(i+1)n+j} \qquad (6.13)$$

$$j = 1, \ldots, n$$

so that (6.12) may now be expressed as

$$[s_{n-k,n-k}I + S]\tilde{\mathbf{p}}_\delta = \tilde{\mathbf{f}}_\delta \qquad (6.14)$$

where

$$\tilde{\mathbf{p}}_\delta = [\tilde{p}_{(n-k-1)n+1}, \cdots, \tilde{p}_{(n-k)n}]$$

and $\tilde{\mathbf{f}}_\delta$ is the updated right hand side.

The matrix on the left hand side of (6.14) is an upper Schur matrix. (6.14) may be solved by back-substituting either a row at a time or two rows at a time depending on the sub-diagonal elements of S. If (iii) of (6.9) is satisfied, then we update the corresponding element on the right hand side of (6.14). At step m, for $m = n, \ldots, 1$, we have

$$\tilde{f}_{(n-k-1)n+m} = \tilde{f}_{(n-k-1)n+m} - \sum_{j=m+1}^{n} s_{mj}\tilde{p}_{(n-k-1)n+j} \qquad (6.15)$$

then

$$\tilde{p}_{(n-k-1)n+m} = \frac{\tilde{f}_{(n-k-1)n+m}}{s_{m,m} + s_{n-k,n-k}} \qquad (6.16)$$

If (iii) of (6.9) is not satisfied, then we have a system of 2 linear equations in 2 unknowns. The right hand side elements are updated using (6.15) with $m = m$ and $m = m - 1$, such that we solve,

$$\left.\begin{aligned} (s_{m-1,m-1} + s_{n-k,n-k})\tilde{p}_{(n-k-1)n+m-1} + s_{m-1,m}\tilde{p}_{(n-k-1)n+m} &= \tilde{f}_{(n-k-1)n+m-1} \\ s_{m,m-1}\tilde{p}_{(n-k-1)n+m-1} + (s_{m,m} + s_{n-k,n-k})\tilde{p}_{(n-k-1)n+m} &= \tilde{f}_{(n-k-1)n+m} \end{aligned}\right\} \qquad (6.17)$$

for the unknowns $\tilde{p}_{(n-k-1)n+m-1}$ and $\tilde{p}_{(n-k-1)n+m}$.

(b) **if (if) of (6.9) is not satisfied**, then either of conditions (i) or (ii) lead to the same processing. Let $j = n - k$, then at the k^{th} phase of the process, if $s_{j+1,j} \neq 0$, then we have

$$\begin{pmatrix} | & | & & | \\ | & | & & | \\ s_{jj}I + S & s_{j,j+i}I & \cdots & s_{j,n}I \\ s_{j+1,j}I & s_{j+1,j+1}I + S & \cdots & s_{j+1,n}I \\ | & | & & | \\ | & | & & | \end{pmatrix} \begin{pmatrix} | \\ | \\ \tilde{\mathbf{p}}_\alpha \\ \tilde{\mathbf{p}}_\beta \\ | \\ | \end{pmatrix} = \begin{pmatrix} | \\ | \\ \tilde{\mathbf{f}}_\alpha \\ \tilde{\mathbf{f}}_\beta \\ | \\ | \end{pmatrix} \tag{6.18}$$

where

$$\tilde{\mathbf{p}}_\alpha = (\tilde{p}_{(j-1)n+1}, \cdots, \tilde{p}_{(j-1)n+n})^T$$

$$\tilde{\mathbf{p}}_\beta = (p_{jn+1},\ \tilde{p}_{jn+2}, \cdots, \tilde{p}_{jn+n})^T$$

Similarly $\tilde{\mathbf{f}}_\alpha = (\tilde{f}_{(j-1)n+1}, \cdots, \tilde{f}_{(j-1)n+n})^T$

$$\tilde{\mathbf{f}}_\beta = (\tilde{f}_{jn+1},\ \tilde{f}_{jn+2}, \cdots, \tilde{f}_{jn+n})^T$$

Now since, by this stage, we already know $\tilde{p}_m$ for $m = (j+1)n+1, \ldots, n^2$, we can update the right hand side using (6.13). (6.18) now becomes

$$\begin{pmatrix} s_{jj}I + S & s_{j,j+1}I \\ s_{j+1,j}I & s_{j+1,\ j+1}I + S \end{pmatrix} \begin{pmatrix} \tilde{\mathbf{p}}_\alpha \\ \tilde{\mathbf{p}}_\beta \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{f}}_\alpha \\ \tilde{\mathbf{f}}_\beta \end{pmatrix} \tag{6.19}$$

The matrix on the left is sparse, in fact it possesses at most $n^2 + 3n$ non-zero elements from a total of $4n^2$ possible elements. (6.19) can clearly be solved by the Gaussian Elimination method. However, there exists a stable variant of the Gaussian Elimination method that takes advantage of the sparsity of the matrix, requiring fewer computations [Duff].

We continue in this way until element s_{21} has been tested. Then since $U^T = U^{-1}$

$$p^{(r)} = (U^T \otimes U^T)^{-1}\tilde{p}^{(r)} = (U \otimes U)\tilde{p}^{(r)}$$

gives the direction of search used to update $y^{(r+1)}$ and hence $X^{(r+1)}$.

The convergence is quadratic from a good starting point, which is what we would expect from a Newton method. To maintain the symmetry of the computed $X^{(r+1)}$ under round-off error, we use the following transformation

$$X^{(r+1)} = \frac{1}{2}(X^{(r+1)} + X^{(r+1)^T})$$

6.2.1 An Algorithm with Operation Counts

We now present an algorithm, with operation counts, for the non-negative definite symmetric solution of the algebraic Riccati equation. Notice that the problem of computing a Kronecker product of the form,

$$a = (U^T \otimes U^T)b$$

where

$$a, b \in \mathbf{R}^{n^2} \quad \text{and} \quad U \in \mathbf{R}^{n \times n}$$

requires an operations count of $O(n^4)$. A more efficient way of computing a, using only $O(n^3)$ operations, is to transform the problem to

$$A = U^T B U$$

where a and b are formed from the elements of A and B, taken a row at a time.

(i) Input $X^{(0)} = x_{ij}^{(0)}$, the initial estimates to the solution of (6.1).
Set $r = 0$

(ii) Form $T^{(r)} = A^T - X^{(r)}G$ (n^3 operations)

(iii) Compute

$$F(X^{(r)}) = T^{(r)}X^{(r)} + X^{(r)}A + H \qquad (2n^3 \text{ operations})$$

$$f_{(i-1)n+j} = F_{ij} \qquad i, j = 1, \ldots, n \qquad (n^2 \text{ operations})$$

(iv) Compute $S^{(r)}$, the Schur decomposition of T,

$$U^T T^{(r)} U = S^{(r)} \qquad (15n^3 \text{ operations})$$

where $U^T U = I$

(v) Compute

$$\tilde{f}^{(r)} = -(U^T \otimes U^T) f^{(r)}$$

by

$$\tilde{F}^{(r)} = -U^T F^{(r)} U \qquad (2n^3 \text{ operations})$$

(vi) $k = 0$
Start:
If $s_{n-k,\, n-k-1} <$ EPS or if $k = n - 1$
update right hand side $\tilde{f}$ using (6.13). ($2kn$ operations)
solve (6.14), using (6.15), (6.16) and (6.17). ($3n^2$ operations)
Put $k = k + 1$
If $k > n - 1$ then go to (vii).
otherwise go to Start:

Otherwise

update right hand side $\tilde{f}$ using (6.13), for $k = k$ and

$k = k + 1$. ($4kn$ operations)

solve (6.19) using the sparse variant of the Gaussian Elimination method.

The algorithm requires approximately $\frac{5\tau^2}{2n} + \tau$ operations where τ is the number of non-zero elements in the matrix. In this case $\tau < n^2 + 4n$ so that an overestimate of the operations count is $\frac{5}{2}n^3 + 22n^2 + O(n)$.

Put $k = k + 2$

If $k > n - 1$ go to (vii)

otherwise go to Start:

(vii) Compute

$$p^{(r)} = (U \otimes U)\tilde{p}^{(r)}$$

by

$$P^{(r)} = U\tilde{P}^{(r)}U^T \qquad (2n^3 \text{ operations})$$

(viii) Update X, $\quad X^{(r+1)} = X^{(r)} + P^{(r)}$

(ix) Transform to a symmetric $X^{(r+1)}$,

$$X^{(r+1)} = \frac{1}{2}(X^{(r+1)} + X^{(r+1)^T})$$

(x) Test for convergence,

if $\frac{||X^{(r+1)} - X^{(r)}||}{||X^{(r)}||} \leq TOL \qquad (2n^2 \text{ operations})$

then convergence has occurred,

otherwise go to (ii).

Excluding Step (vi), the operations count per iteration is $22n^3 + 3n^2 + O(n)$.

The operations count at Step (vi) depends on the number of complex eigenvalues of T;

for real T, the count is $n^3 - n^2$

for non-real T, the count is $\frac{5}{4}n^4 + 13n^3$ per iteration.

These values reflect a real saving in using this method over the Newton's method applied directly to the constituent equations or by the sum of squares minimisation approach of Section 5.5. This is especially so when the Jacobian possesses real eigenvalues at each iteration.

6.2.2 Error Analysis

Consider the consequences of using floating point arithmetic in the algorithm of 6.2.1. Using the results and notation of Section 1.3, we have that at the r^{th} iteration, the computed estimate of the solutions is $\hat{X}^{(r)}$, such that

$$\hat{X}^{(r)} = X^{(r)}(1 + \Delta_X)$$

performing step (ii),

$$\hat{T} = fl(\hat{A}^T - \hat{X}^{(r)}\hat{G})$$

where the coefficient matrices of the algebraic Riccati equation are such that,

$$\hat{A} = A(1+\epsilon_1), \qquad |\epsilon_1| \leq u$$

$$\hat{G} = G(1+\epsilon_2), \qquad |\epsilon_2| \leq u$$

$$\hat{H} = H(1+\epsilon_3), \qquad |\epsilon_3| \leq u$$

Then

$$\hat{T}^{(r)} = T^{(r)}(1 + \Delta_X + (n+2)\epsilon) + O(\epsilon^2), \qquad |\epsilon| \leq u$$

$$\hat{T}^{(r)} = T^{(r)}(1 + \Delta_T) \qquad + O(\epsilon^2) \tag{6.20}$$

At step (iii)

$$\hat{F} = fl(\hat{T}^{(r)}\hat{X}^{(r)} + \hat{X}^{(r)}\hat{A} + \hat{H})$$

$$\hat{F} = F(1 + \Delta_T + \Delta_X + (n+3)\epsilon) + O(\epsilon^2)$$

$$\hat{F} = F(1 + \Delta_F) \tag{6.21}$$

At step (iv), the computed $\hat{S}$ satisfies

$$U^T(\hat{T}^{(r)} + E)U = \hat{S}^{(r)}, \qquad ||E|| \leq U||\hat{T}|| \leq U||T|| \tag{6.22}$$

with

$$\hat{U}^T\hat{U} = I + F, \qquad ||F|| \leq u$$

At step (v),

$$\tilde{F}^{(r)} = fl(-\hat{U}^T\hat{F}^{(r)}\hat{U})$$

$$= \tilde{F}^{(r)}(1 + \Delta_F + 2\epsilon + 2n\epsilon) + O(\epsilon^2)$$

$$= \tilde{F}^{(r)}(1 + \Delta_{\tilde{F}}) \qquad + O(\epsilon^2) \tag{6.23}$$

Using (6.20), (6.21) and (6.23),

$$\Delta_{\tilde{F}} = 2\Delta_X + (4n+7)\epsilon \tag{6.24}$$

The processing at step (vi) depends on the values of the sub-diagonal elements of $S^{(r)}$. For each element that is considered to be zero we update $\tilde{f}$ by (6.13) and solve the triangular system in (6.14). (6.13) is essentially a matrix-vector product, such that

$$\tilde{f} = fl(\hat{\tilde{f}} - \hat{S}\hat{\tilde{p}})$$

$$= fl(\tilde{f}(1+\delta_{\tilde{f}}) - S(1+\epsilon)\tilde{p}(1+\delta_{\tilde{p}}))$$

$$\hat{\tilde{f}} = \tilde{f}(1 + \delta_{\tilde{f}} + \delta_{\tilde{p}} + 2\epsilon + n\epsilon) \tag{6.25}$$

where

$$||\delta_{\tilde{f}}|| = ||\Delta_{\tilde{F}}||$$

Next, we try to solve the triangular system in (6.14),

$$\hat{S}\tilde{p} = \hat{\tilde{f}} \tag{6.26}$$

The computed solution satisfies [Golub & Van Loan],

$$(\hat{S} + E_1)\hat{\tilde{p}} = \hat{\tilde{f}} \qquad ||E_2|| \leq nu||S|| \tag{6.27}$$

Using a result from Section 1.3.2,

$$(\hat{S} + E_1)^{-1} = \hat{S}^{-1} - \hat{S}^{-1}E_1\hat{S}^{-1} \qquad + O(||E_1||^2)$$

such that

$$\hat{\tilde{p}} = (\hat{S} + E_1)\hat{\tilde{f}} = \hat{S}^{-1}\hat{\tilde{f}} - \hat{S}^{-1}E_1\hat{S}^{-1}\tilde{f}$$

Using (6.26),

$$\hat{\tilde{p}} = \tilde{p} - \hat{S}^{-1}E_1\tilde{p}$$

Taking norms,

$$\frac{||\delta_{\tilde{p}}||}{||\tilde{p}||} = \frac{||\hat{\tilde{p}} - \tilde{p}||}{||\tilde{p}||} \leq nu||S||\,||S^{-1}|| \tag{6.28}$$

If a sub-diagonal element of $S^{(r)}$ is non-zero, then once again we update using (6.13), this time to yield 2 vectors, and then solve the sparse linear system in (6.19). Using (6.13) gives,

$$\hat{\tilde{f}}_\alpha = \tilde{f}_\alpha(1 + \delta_{\tilde{f}} + \delta_{\tilde{p}} + 2\epsilon + n\epsilon)$$

$$\hat{\tilde{f}}_\beta = \tilde{f}_\beta(1 + \delta_{\tilde{f}} + \delta_{\tilde{p}} + 2\epsilon + n\epsilon)$$

The error involved in solving the $2n \times 2n$ system in (6.19) is such that [Duff],

$$(\hat{S} + E_2)\hat{\tilde{p}} = \hat{\tilde{f}}, \qquad ||E_2|| \leq 3nu||S|| \tag{6.29}$$

and a similar analysis to that above shows that,

$$\frac{||\delta_{\tilde{p}}||}{||\tilde{p}||} = \frac{||\hat{\tilde{p}} - \tilde{p}||}{||\tilde{p}||} \leq 3nu||S||\,||S^{-1}|| \tag{6.30}$$

At step (vii),

$$\hat{P}^{(r)} = fl(\hat{U}\hat{\tilde{P}}^{(r)}\hat{U}^T)$$

$$\hat{P}^{(r)} = \hat{P}^{(r)}(1 + \Delta_{\tilde{p}} + 2\epsilon + 2n\epsilon) + O(\epsilon^2)$$

Finally at step (viii), we update X,

$$X^{(r+1)} = X^{(r)} + \hat{P}^{(r)}$$

We observe that accuracy of the update matrix $P^{(r)}$ is dependent on how accurately the solutions of the linear systems (6.27) and (6.29) can be computed. The accuracy is dependent on two things, on the accuracy of $\hat{\tilde{f}}$ and on the smallness of the bounds in (6.28) and (6.30). Now (6.28) and (6.30) are small if the matrix S is well-conditioned. This would guarantee the accuracy of a linear system with an exact right hand side. In (6.28) and (6.30), the computed and exact right hand sides, $\hat{\tilde{f}}$ and $\tilde{f}$ respectively, are such that

$$\hat{\tilde{f}} - \tilde{f} = (\delta_{\tilde{f}} + \delta_{\tilde{p}} + 2\epsilon + n\epsilon)$$

Using (6.24)

$$\hat{\tilde{f}} - \tilde{f} = \tilde{f}(2\Delta_X + \delta_{\tilde{p}} + 9\epsilon + 5n\epsilon) \tag{6.31}$$

We have already stated that $\delta_{\tilde{p}}$ is small when S is well-conditioned. Δ_X is the error in the current estimate to the solution, computed at the previous step. If the matrix S at the previous iteration was well-conditioned, then $\delta_{\tilde{p}}$ and hence Δ_X will be small.

To summarize, if the matrix S is well-conditioned at each iteration of the algorithm, then the rounding errors introduced in the algorithm are small and well-bounded. If however S is not well-conditioned, then the update matrix P may be inaccurate leading to erroneous results.

The following example serves to illustrate the method, with a well-conditioned matrix S at each iteration.

Example

Consider

$$A^TX + XA - XBR^{-1}B^TX + H = 0$$

where

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad R = I, \; H = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Choose $X^{(0)} = \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix}$ with $\mathrm{Re}(\lambda(A - GX^{(0)})) < 0$ where $G = BR^{-1}B^T = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$

Observing each step of the algorithm,

(ii) $T = A^T - X^{(0)}G = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix}$

(iii) $F(X^{(1)}) = TX^{(0)} + X^{(0)}A + H$

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

and $||F||_E = 3$.

(iv) Since T is already in Schur form, $U = I_2$ and $S = T$

(v) Since $U = I$, $\tilde{f} = f = \begin{pmatrix} 2 \\ -2 \\ 0 \\ 1 \end{pmatrix}$

$k = 0$,

(vi) Since $s_{21} \neq 0$ and $k \neq 1$, solve

$$\tilde{S}\tilde{p} = \tilde{f}$$

where

$$\tilde{S} = \begin{pmatrix} 0 & -1 & -1 & 0 \\ 1 & -1 & 0 & -1 \\ 1 & 0 & -1 & -1 \\ 0 & 1 & 1 & -2 \end{pmatrix}$$

This gives

$$\tilde{p} = (2, -2, 0, 1)^T$$

(vii) Then $\tilde{p} = p$

(viii) $x_{ij}^{(1)} = x_{ij}^{(0)} - p_{(i-1)n+j}$

$$= \begin{pmatrix} 2.5 \\ 1 \\ 1 \\ 2.5 \end{pmatrix}$$

(ix) $X^{(1)}$ is already symmetric. Repeat from (ii) until convergence.

The following table gives the solution matrix, the function norm and the eigenvalues of the closed loop system, at each iteration.

Iteration	X		Function Norm	$\lambda(T)$	
0	-1.0	1.0	3.0	-0.5	$+\sqrt{3}i$
	-1.0	1.0		-0.5	$-\sqrt{3}i$
1	2.5	1.0	3.25	-1.5	
	1.0	2.5		-2.0	
2	2.05	1.0	0.2025	-0.8	
	1.0	2.05		-1.25	
3	2.000609	1.0	0.002431316	-0.975625	
	1.0	2.000609		-1.02498	
4	2.0	1.0	0.00000001	-1.0	
	1.0	2.0		-1.0	

We observe that the convergence here is ultimately quadratic, which is what we would expect from Newton's method.

Remarks

It is known that the problem of determining a solution of (6.1) by multiplying the matrices to produce a system of n^2 non-linear equations and then using Newton's method to solve these, is computationally expensive. An attempt to overcome this difficulty has been made. The Jacobian matrix is easily determined and the ensuing linear system may be solved efficiently by observing and utilising the sparse nature of the Jacobian. When the Jacobian possesses real eigenvalues at each step of the Newton method, the overall operations count compares favourably with that for Kleinman's method. The iterations are stable and yield accurate solutions provided that the matrix S is well-conditioned.

SECTION 6.3: Towards a Globally Convergent Newton Method for the Unilateral Quadratic Matrix Equation

In this section, a new sum of squares minimisation method for the solution of the quadratic matrix equation,

$$X^2 + PX + Q = F(X) = 0 \tag{6.32}$$

is proposed.

We show how the problem of an ill-conditioned linear system during the iterations may be overcome and how the inclusion of a line search in the algorithm may guarantee global convergence. We also show that by using suitable transformations, the operations count is much smaller than that of the minimisation method of Section 5.5.

We propose the following algorithm, with analysis to follow,

(i) Select $X^{(0)} \in \mathbb{R}^{n\times n}$ as an initial estimate to X^*, the solution of (6.32).

(ii) Initialise k, $k = 0$

(iii) Compute

$$F(X^{(k)}) = (X^{(k)} + P)X^{(k)} + Q \tag{6.33}$$

(iv) Test for convergence

(v) Form $y^{(k)} \in \mathbb{R}^{n^2}$,

$$y^{(k)}_{(i-1)n+j} = X^{(k)}_{ij} \tag{6.34}$$

(vi) (a) Determine the condition of the Jacobian

(b) Compute the descent direction, $d^{(k)} \in \mathbb{R}^{n^2}$

(vii) Perform a line search - find $\alpha^{(k)}$ such that

$$fc = \frac{1}{2}\sum_{i=1}^{n^2} f_i(y^{(k)} + \alpha^{(k)}d^{(k)})^2$$

is sufficiently reduced, where f_i are formed from the elements of F.

(viii) Update current solution,

$$X^{(k+1)} = X^{(k)} + \alpha^{(k)}D^{(k)}$$

where the elements of the matrix $D^{(k)} \in \mathbb{R}^{n\times n}$ taken a row at a time, forms $d^{(k)}$.

(ix) Increment k, go to (ii).

The line search ensures that under certain conditions, steady progress towards the solution is made, irrespective of the starting matrix - this is global convergence. The necessary conditions are that the update matrix, J^TJ for the Gauss-Newton iterations, is always non-singular and bounded above. Otherwise, if J^TJ is singular (or very nearly singular) then the Gauss-Newton iterates may converge very quickly to a point that is not

even a stationary point of fc [Powell]. Alternatively, if J^TJ is very large then the direction and gradient vectors are nearly at right angles to each other and convergence is very slow.

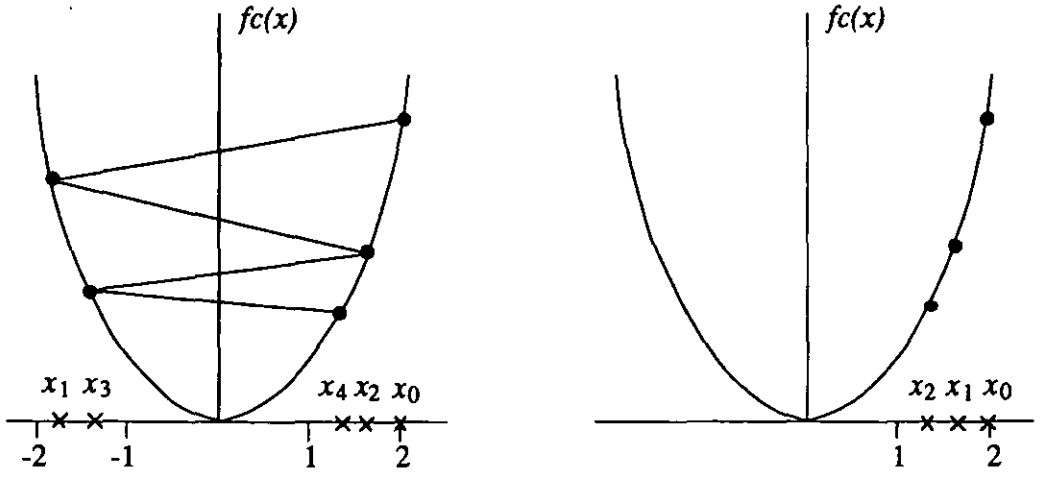
The problem of an ill-conditioned Jacobian can clearly be overcome by adding a small scalar matrix to J^TJ, as we observed in Section 5.5. However, this requires that the solution of an n^2 linear system must be computed. This is a computationally expensive task and accounts for the prohibitive nature of the minimisation technique. We address this problem in due course but firstly we take a closer look at the problem of performing a line search.

In Section 1.5, we discussed how Powell's quadratic interpolation method may be used to perform the line search. The following analysis highlights the problems that can be encountered by line search algorithms and how they may be overcome.

The problem of determining an $\alpha^{(k)}$ in the line search has, until recently, been one of choosing the $\alpha^{(k)}$ that solves the one-dimensional minimisation problem accurately. However, more careful computational testing has led to the belief that low accuracy minimisations are better in theory and in practice. The question now arises as to what the criteria for the low-accuracy minimisation should be. Consider

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$$

Clearly we require that $fc(x^{(k+1)}) < fc(x^{(k)})$. However, this does not guarantee convergence, as illustrated in the following monotonically decreasing sequences of iterates, [Dennis & Schnabel].



In the first case, we have very small decreases in fc values relative to the length of the steps. This is resolved by requiring that the average rate of decrease from $fc(x^{(k)})$ to $fc(x^{(k+1)})$ be at least some prescribed fraction of the initial rate of decrease in that direction. That is, choose an $a \in (0,1)$ and an $\alpha^{(k)}$ satisfying

$$fc(x^{(k+1)}) \leq fc(x^{(k)}) + a\nabla fc(x^{(k)})^T(x^{(k+1)} - x^{(k)})$$

where $\nabla fc(x^{(k)})$ is the gradient of fc at $x^{(k)}$.

In the second case, the steps are too small relative to the initial rate of decrease of fc. [Dennis & Schnabel] suggest the following condition that ensures sufficiently large steps, for some fixed constant $b \in (\alpha, 1)$,

$$\nabla fc(x^{(k+1)})^T(x^{(k+1)} - x^{(k)}) \geq b\nabla fc(x^{(k)})^T(x^{(k+1)} - x^{(k)})$$

Therefore the accuracy of the minimisation depends on the values of a and b.

Now we return to the problem of an ill-conditioned Jacobian.

At Step (vi) of the algorithm, we estimate the condition of the Jacobian. The iterative processing distinguishes between a well-conditioned and an ill-conditioned Jacobian as follows.

Well-conditioned Jacobian

In this case, we choose to use the Gauss-Newton direction given by the solution of,

$$J^{(k)^T} J^{(k)} d^{(k)} = -J^{(k)^T} f^{(k)} \tag{6.35}$$

where $J^{(k)}$ denotes the Jacobian at $X^{(k)}$ and $f^{(k)}$ are the elements of the function matrix $F(X^{(k)})$ taken a row at a time. (6.35) may be written as,

$$J^{(k)} d^{(k)} = -f^{(k)}$$

From (5.43),

$$((X^{(k)} + P) \otimes I + I \otimes X^{(k)^T}) d^{(k)} = -f^{(k)} \tag{6.36}$$

Using the Kronecker Product theory of Section 1.2, (6.36) may be expressed as the Sylvester equation,

$$(X^{(k)} + P)D^{(k)} + D^{(k)} X^{(k)} = -F(X^{(k)}) \tag{6.37}$$

where $d^{(k)}$ are the elements of the matrix D taken a row at at time.

The discussions in Section 1.4 on the Sylvester equation suggest that since the Jacobian is well-conditioned, the problem (6.37) is well-conditioned and the method of Section 1.4 for solving (6.37) is stable thus providing an accurate solution. Determining the direction $d^{(k)}$ (or $D^{(k)}$) by (6.37) is clearly more efficient than solving the linear system of order n^2 in (6.36).

Ill-conditioned Jacobian

In this case either of the following two options may be considered:

(a) Use the previous well-conditioned Jacobian, say $J(X_p)$. If this situation occurs at the first iteration then the iterative algorithm should be re-started from a different starting matrix.

The new direction is then given by,

$$J_p d^{(k)} = -f^{(k)} \tag{6.38}$$

or equivalently

$$(X_p + P)D^{(k)} + D^{(k)}X_p = -F(X^{(k)}) \tag{6.39}$$

Now, a direction d is a descent direction if [Dennis & Schnabel],

$$g^T d < 0 \tag{6.40}$$

where g is the gradient, in this case $g = J^T f$ as determined in Section 5.5. Therefore the direction $d^{(k)}$ of (6.38) is a descent direction if

$$-f^{(k)^T} J^{(k)} J_p^{-1} f^{(k)} < 0 \tag{6.41}$$

For arbitrary $J^{(k)}$ and J_p we cannot predict whether (6.41) holds or not. If it does hold then $d^{(k)}$ is a descent direction and the line search algorithm determines an $\alpha^{(k)}$ and hence $X^{(k+1)}$ such that the decrease of $f^{(k+1)}$ satisfies some prescribed criterion. If (6.41) does not hold, then the line search algorithm determines an $\alpha^{(k)}$ and hence $X^{(k+1)}$ that may be farther away from the solution X^* than $X^{(k)}$ is. We could then only hope that $J^{(k+1)}$ is well-conditioned and treated accordingly.

(b) Use the Steepest Descent direction $d^{(k)} = -J^{(k)^T} f^{(k)}$. This is clearly a descent direction since (6.40) is satisfied,

$$g^{(k)^T} d^{(k)} = -g^{(k)^T} J^{(k)^T} f^{(k)} = -(J^{(k)^T} f^{(k)})^T J^{(k)^T} f^{(k)} < 0 \tag{6.42}$$

The updated estimate $X^{(k+1)}$ determined via the line search algorithm is nearer to X^* than $X^{(k)}$ is to X^*. This suggests that alternative (a) above is superfluous since (6.42) is always satisfied. However, it is known [Dennis & Schnabel] that for a minimisation algorithm employing the Steepest Descent direction throughout, the convergence is only linear, and sometimes very slowly linear. In contrast, the Gauss-Newton iterations are known to have fast convergence; we will show later that they exhibit quadratic local convergence for this problem. Therefore, it may be that over the course of the minimisation method, there are found to occur a number of ill-conditioned Jacobians, most of which satisfy (6.41). This would justify using (a) over (b).

In the following analysis however, we will use (b).

6.3.1 Condition of J

The problem of determining the condition number of a matrix, say J, is not easy. [Wilkinson] shows how the Singular Value Decomposition may be used to determine the spectral condition number, $k_2(J)$ say, with

$$k_2(J) = \|J\|_2 \, \|J^{-1}\|_2 = \frac{\mu_1}{\mu_n}$$

where μ_i^2 are the singular values of J with $\mu_1^2 \geq \mu_2^2 \geq \ldots \geq \mu_n^2 \geq 0$. However, the operations count in the computations of the SVD for J is $O(n^6)$ which is clearly unacceptable.

Alternatively, steps (vi) and (vii) can be combined so that the condition of the Jacobian can be estimated in the course of solving (6.37), in the following way.

Rather than using the Hessenberg-Schur method to solve the equation (6.37), we use the Schur method [Bartels & Stewart]. The latter is slightly more expensive in computational terms, but serves our purposes nicely. If we write (6.37) as,

$$AD + DB = -H \tag{6.43}$$

where

$$A = X + P,\ B = X,\ H = F(X)$$

the Schur method reduces A to upper Schur form and B to lower Schur form,

$$U^T A U = \tilde{A} \tag{6.44}$$

$$V^T B V = \tilde{B} \tag{6.45}$$

If U, V are unitary (complex orthogonal) similarity transformations, then $\tilde{A}$ and $\tilde{B}$ will be upper and lower triangular respectively. (6.43) can then be written as

$$\tilde{A}\tilde{D} + \tilde{D}\tilde{B} = -\tilde{H} \tag{6.46}$$

where

$$\tilde{D} = U^T D V, \quad \tilde{H} = U^T F(X) V \tag{6.47}$$

Using Kronecker products, (6.46) may be written as

$$\tilde{J}\tilde{d} = -\tilde{h} \tag{6.48}$$

where

$$\tilde{J} = \tilde{A} \otimes I + I \otimes \tilde{B}^T \tag{6.49}$$

From our knowledge of Kronecker products we observe that $\tilde{J}$ is upper triangular and that

$$\begin{aligned}
(U^T \otimes V^T) J (U \otimes V) &= (U^T \otimes V^T)(A \otimes I + I \otimes B^T)(U \otimes V) \\
&= (U^T A \otimes V^T + U^T \otimes V^T B^T)(U \otimes V) \\
&= (U^T A U \otimes V^T V + U^T U \otimes V^T B^T V \\
&= (\tilde{A} \otimes I + I \otimes \tilde{B}) \\
&= \tilde{J}
\end{aligned} \tag{6.50}$$

Also $U \otimes V$ is unitary since

$$\begin{aligned}(U \otimes V)(U \otimes V)^T &= (U \otimes V)(U^T \otimes V^T) \\ &= UU^T \otimes VV^T \\ &= I \otimes I\end{aligned}$$

as required.

Hence from (6.50) and since the norm is invariant under unitary (orthogonal) transformation

$$||\tilde{J}|| = ||J|| \tag{6.51}$$

and since

$$J^{-1} = (U \otimes V)\tilde{J}^{-1}(U^T \otimes V^T)$$

we have that

$$||\tilde{J}^{-1}|| = ||J^{-1}|| \tag{6.52}$$

Therefore the problem of determining a condition number $k_\infty(J)$, (we choose ∞ norm because it is easily computed) is such that,

$$k_\infty(J) = ||J||_\infty \, ||J^{-1}||_\infty = ||\tilde{J}||_\infty \, ||\tilde{J}^{-1}||_\infty = k_\infty(\tilde{J}) \tag{6.53}$$

(6.53) states that the condition number is invariant under unitary (orthogonal) transformations.

Determining $||\tilde{J}||$ is relatively simple. To determine $||\tilde{J}^{-1}||_\infty$ [Golub & Van Loan] provide an algorithm that computes an estimate for the ∞ norm of the inverse of an upper triangular matrix. The algorithm is stable with an operations count of $\frac{5}{4}n^4$.

Summarising, at step (vi) of the minimisation algorithm we attempt to determine the descent direction by solving (6.37) for $D^{(k)}$ using the Schur method. Once the upper and lower Schur forms of $(X^{(k)} + P)$ and $X^{(k)}$ respectively are computed, we form the matrix $\tilde{J}$ and compute an estimate to the condition number of $\tilde{J}$ and hence J. If it is found that the Jacobian is well-conditioned, the Schur method is continued until completion to obtain $D^{(k)}$. Then the vector $d^{(k)}$ is formed and used in step (vii). However, if the Jacobian is ill-conditioned then we choose the Steepest Descent direction.

6.3.2 Convergence Criteria

With respect to the convergence of the iterates, there are two considerations to be made here. Firstly, the algorithm should detect that the iterations have converged and secondly to ensure that convergence is to (6.32) rather than a local minimum of the scalar function fc,

$$fc = \frac{1}{2}\sum_{i=1}^{n} f_i^2 \tag{6.54}$$

We choose to terminate the iterations when some quantity say XNORM, is less than some specified tolerance value, say TOL. The following criterion is used,

$$X\text{NORM} = \frac{||X^{(k+1)} - X^{(k)}||}{||X^{(k)}||} \leq \text{TOL} \tag{6.55}$$

To test whether (6.32) is satisfied, the following criterion is suggested [NAG],

$$F\text{NORM} = ||F(X^{(k)})|| \leq \sqrt{u} \tag{6.56}$$

where u is the unit machine round-off .

With respect to rate of convergence, it is known [Gill & Murray] that if the Hessian matrix is

$$J^T J + Z, \qquad \text{say}$$

then the Gauss-Newton steps will ultimately converge at the same rate as Newton's method, i.e. locally quadratic, provided that $||Z||$ is small compared with $||J^T J||$. From (5.46) and the fact that $F(X^*) \to 0$ as the iterates approach X^*, $||Z||$ becomes smaller such that in exact arithmetic $||Z(X^*)|| = 0$. Hence we can say that the method of this section can exhibit quadratic local convergence.

6.3.3 Operations Count

The operations performed by the algorithm are as follows:

Step (iii) n^3 to compute $F(X^{(k)})$

Step (iv) Compute the norm of $F(X^{(k+1)})$

$F\text{NORM} = \left[\sum_i \sum_j F_{ij}^{(k+1)^2}\right]^{\frac{1}{2}}$

uses $n^2 + O(n)$ operations

Step (v) n to form $y_i^{(k)}, \qquad i = 1, n^2$

Step (vi) Either,

Compute Schur forms of $(X^{(k)} + P)$ and $X^{(k)}$	$20n^3$
Form $\tilde{J}$	$3n^2$
Estimate $k_\infty(J)$	$\frac{5}{4}n^4$
Complete the solution of (6.37)	$5n^3$
Form $d_{(i-1)n+j}^{(k)} = D_{ij}^{(k)}$	n

or

Compute Schur forms of $(X^{(k)} + P)$ and $X^{(k)}$	$20n^3$

	Form $\tilde{J}$	$3n^2$
	Estimate $k_\infty(J)$	$\frac{5}{4}n^4$
	Form $d^{(k)} = -J^T f$	$3n^3$

where $\tilde{J}$ and $-J^T f$ are formed by using (5.43) and (5.44) respectively.

We see that both branches perform approximately the same number of operations. Therefore we can generalise the operations count of Step (vi) to $\frac{5}{4}n^4 + 25n^3 + 3n^2 + O(n)$

Step (vii) $3(2n^3 + n^2)$ to evaluate fc at α_1, α_2, α_3 say, in the line search algorithm and $2n^3 + n^2$ for each further function evaluation. Generalise as $r(2n^3 + n^2)$

Step (viii) n^2 to update to $X^{(k+1)}$.

The overall operations count for this algorithm is then,

$$\frac{5}{4}n^4 + (26 + 2r)n^3 + (4 + r)n^2 + O(n)$$

per iteration, where r is the average number of function evaluation performed by the line search procedure.

6.3.4 Numerical Results

The following examples illustrate how the method of this section may be used to solve various problems. Some of these problems cannot be solved by the methods of Section 5.3.

Example 1

A singular Jacobian is encountered at the first iteration

$$X^2 + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X + \begin{pmatrix} -8 & -12 \\ -18 & -26 \end{pmatrix} = 0$$

Let $$X^{(0)} = \begin{pmatrix} -2 & 0 \\ 0 & -0.5 \end{pmatrix}, \quad \text{then} \quad f^{(0)} = -(6, 12, 18, 26)^T, \quad ||F|| = 34.542,$$

$$J(X^{(0)}) = \text{ diag } (-3, -1.5, -1.5, 0),$$

and

$$\sigma(J(X^{(0)}) = 3, 1.5, 1.5, 0$$

Since the Jacobian is singular, we take the Steepest Descent direction,

$$d^{(0)} = -J^{(0)^T} f^{(0)}$$

$$d^{(0)} = -(18, 18, 27, 0)^T$$

The line search yields $\alpha = 0.1257$ such that the updated solution is,

$$X^{(1)} = \begin{pmatrix} -4.2619 & -2.2619 \\ -3.3928 & -0.5000 \end{pmatrix}$$

and

$$||F(X^{(1)})|| = 23.8532$$

so that using the Steepest Descent direction has resulted in a new estimate that is closer to the solution. Continuing in this way,

k	Singular values	α (alpha)	Solution X	$\frac{\|X^{(k+1)}-X^{(k)}\|}{\|X^{(k)}\|}$	FNORM
1	3.0 1.5	0.1257	-4.2619 -2.2619	2.2619	23.8532
	1.5 0.0		-3.3928 -0.5000		
2	10.5982 4.3060	0.4548	-2.3183 -3.5069	0.5109	16.7477
	3.7622 2.5300		-5.2604 -1.0748		
3	11.3606 6.8163	0.4252	-0.5493 -3.5723	0.4100	11.5133
	2.3931 2.1512		-5.3590 -3.2363		
4	12.2917 6.9341	0.7954	-0.1123 -2.7679	0.2969	5.3126
	2.7856 2.5000		-0.4152 -4.7567		
5	12.2625 5.2097	1.0000	-1.1069 -2.4523	0.1667	1.2601
	3.8690 3.1837		-3.6784 -4.6697		
6	11.9125 5.0916	1.0000	-1.4716 -2.2373	0.0857	0.3061
	4.7766 2.0443		-3.3560 -4.8405		
7	11.8781 5.4945	0.9999	-1.6787 -2.1472	0.0417	0.0716
	5.3122 1.0715		-3.2208 -4.8988		
8	11.8704 5.7246	1.0000	-1.7700 -2.1054	0.0188	0.0145
	5.5775 0.5683		-3.1581 -4.9277		
9	11.8671 5.8314	0.9999	-1.8007 -2.0912	0.0064	0.0017
	5.6972 0.3385		-3.1367 -4.9375		
10	11.8660 5.868	0.9998	-1.8055 -2.0890	0.00098	0.00004
	5.738 0.2596		-3.1335 -4.9389		

Example 2

The Newton and Matrix Polynomial methods of Section 5.3 work on the following problem,

$$X^2 + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X + \begin{pmatrix} -8 & -12 \\ -18 & -26 \end{pmatrix} = 0$$

with

$$X^{(0)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

In this case $||F(X^{(0)})|| = 32.8634$.

The method of this section converges to $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ in the following way,

Iteration	1	2	3	4	5	6	7
$\frac{\|\|X^{(k+1)}-X^{(k)}\|\|}{\|\|X^{(k)}\|\|}$	3.1557	0.1095	0.0538	0.0246	0.0088	0.0016	0.00004
$\|\|F(X^{(k+1)})\|\|$	1.4365	0.3552	0.0848	0.0178	0.0023	0.00007	0.000009

Example 3

In this case the norm of the starting matrix is very large compared to the solution.

$$X^2 - \begin{pmatrix} 1 & 6 \\ 2 & 9 \end{pmatrix} X + \begin{pmatrix} 0 & 12 \\ -2 & 14 \end{pmatrix} = 0$$

with

$$X^{(0)} = \begin{pmatrix} -99 & 10 \\ -2 & 14 \end{pmatrix} \text{ and } F(X^{(0)}) = 1.044 \times 10^5$$

The method converges to $\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ in the following way,

Iteration	1	2	3	4	5	6	7	8	9
$\|\|F(X^{(k)})\|\|$	2615	653.7	164.3	41.871	63.243	3.4159	0.9995	0.1424	0.0005

Example 4

The Matrix Polynomial method of Section 5.3 does not work since the problem does not possess a dominant solution.

$$X^2 - \begin{pmatrix} 1 & 6 \\ -2 & 9 \end{pmatrix} X + \begin{pmatrix} 0 & 12 \\ -2 & 14 \end{pmatrix} = 0$$

and

$$X^{(0)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } ||F(X^{(0)})|| = 8.4853$$

The method converges to $\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$ in the following way,

Iteration	1	2	3	4	5
$\|\|F(X^{(k)})\|\|$	2.0365	0.4338	0.0596	0.0023	0.000003

Example 5

The Newton method of Section 5.3 does not converge.

$$X^2 + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X + \begin{pmatrix} -8 & -12 \\ -18 & -26 \end{pmatrix} = 0$$

with

$$X^{(0)} = \begin{pmatrix} 1 & 6 \\ -5 & 1 \end{pmatrix}, \qquad ||F(X^{(0)})|| = 73.055$$

The method converges to $\begin{pmatrix} 0.8056 & 2.0889 \\ 3.1334 & 3.9390 \end{pmatrix}$ in the following way,

Iteration	1	2	3	4	5	6	7	8
$\|\|F(X^{(k)})\|\|$	16.1342	11.9821	2.7678	0.4753	0.0581	0.0117	0.0012	0.00002

Remarks

In this section, a new sum of squares minimisation method for solving the quadratic matrix equation has been proposed. The method is based on Gauss-Newton iterations and is shown to possess local quadratic convergence. The method is shown to be efficient in the sense that the operations count per iteration is $O(n^4)$ rather than $O(n^6)$ as usually required for the minimisation of the constituent equations. The term $O(n^4)$ is wholly due

to the determination of the singular values of the Jacobian. It may be that a more efficient technique for determining the condition of the Jacobian exists in which case the operations count is $O(n^3)$ per iteration and compares favourably with that for the methods in Section 5.3.

Perhaps the most significant attribute of this method is its global convergence property. The line search at each iteration ensures that progress is made towards the solution. Also, the problem of solving an ill-conditioned linear system at an iteration is overcome, by choosing the Steepest Descent direction in such cases.

The comments above are reflected in the numerical results where this method is shown to work on problems for which other methods fail.

Therefore the analysis and method of this section may contribute towards providing a stable, efficient and globally convergent algorithm for solving the general unilateral quadratic matrix equation.

CHAPTER 7 - EXAMPLES, RESULTS, COMPARISONS

SECTION 7.1: Introduction

In this chapter we use the methods of the preceding chapters to solve a number of problems in order to determine the merits of one method over another. The measures we use of 'goodness' of a method are the accuracy of the computed solution and the Central Processor Unit (CPU) time. The CPU time is the time accumulated while a particular task is in execution and is independent of external factors, for example, the number of users currently in session on the mainframe, the amount of memory resident in the machine, etc. The operations count and access rates to stored arrays are reflected in the CPU time. Immediately prior to executing a task, the program calls a function called CPU() and passes one integer parameter through it. On return, the integer contains a value representing the number of micro-seconds of CPU time that has accumulated since some arbitrary base. This base generally remains unchanged across successive CPU() calls, a non-zero return code indicating otherwise. On completion of the task, another call is made to CPU() with an integer parameter and the difference between this and the previous integer gives the accumulated CPU time.

Broadly speaking, the relative CPU times of the methods should correspond roughly to the relative operations count of the methods. The operations count, however, does not take into account the numerous other tasks the CPU must do within a program. Consequently it is important to minimise the work involved in these other tasks and reduce/re-use storage spaces wherever possible. The work involved in multiplying matrices dominates in the methods and the way in which this operation is approached can be observed by considering the following matrix multiplications,

(a)

$$C = AB, \qquad A \in \mathbf{R}^{n\times m}, \quad B \in \mathbf{R}^{m\times n}, \quad C \in \mathbf{R}^{n\times n}$$

For $i = 1, \ldots, n$

  For $j = 1, \ldots, n$

    sum $= 0.0$

    For $k = 1, \ldots, m$

      sum $=$ sum $+ a_{ik}b_{kj}$

    Next k

    $c_{ij} =$ sum

  Next j

Next i

(b)

$B = AB, \qquad A, B \in \mathbf{R}^{n \times n}, \quad \text{vec} \in \mathbf{R}^n$

For $j = 1, \ldots, n$

For $i = 1, \ldots, n$

sum $= 0.0$

For $k = 1, \ldots, n$

sum = sum $+ a_{ik} b_{kj}$

Next k

$\text{vec}_i = \text{sum}$

Next i

For $\ell = 1, \ldots, n$

$b_{\ell j} = \text{vec}_\ell$

Next ℓ

Next j

Both examples illustrate how the introduction of a scalar 'sum' reduces the amount of subscripting. Also, the second example shows how, by using an n-vector, the storage of a third $n \times n$ matrix C is unnecessary. When the problem is large, optimisations of this kind prove to be very useful.

Bearing in mind that it takes more work to perform some operations than others, great care has been taken to maintain uniformity in the program code written for the implementation of each method. Examples of some tasks common to a number of methods are:

(a) The determination of the eigenvalues of a general matrix.
(b) The computation of the upper Hessenberg form of a matrix.
(c) The computation of the real Schur form of a matrix.
(d) The computation of the companion form of a matrix.
(e) The solution of the Lyapunov and Sylvester matrix equations.
(f) The solution of a general linear system of order n.

The program code for carrying out these tasks is not included in the Appendices since it is readily available in literature [NAG], [Faddeev & Faddeeva], [Golub, Nash & Van Loan], [Bartels & Stewart], [Xinogalas et al], [Golub & Van Loan].

Three subroutines that merit greater attention and are taken from the Numerical Algorithm Group (NAG) are:

1) E04HEF - sum of squares minimisation.

This is a comprehensive modified Gauss-Newton algorithm for finding an unconstrained minimum of the sum of squares of M non-linear functions in N variables, with $M \geq N$. The routine is intended for functions which have continuous first and second derivatives but will usually work even if the derivatives have occasional discontinuities.

The subroutine requires three external functions,

(a) to calculate the vector of function values and the Jacobian matrix of first derivatives at any point.

(b) to calculate the elements of the symmetric matrix

$$B(X) = \sum_{i=1}^{M} f_i(X) G_i(X) \text{ at any point } X$$

where $G_i(X)$ is the Hessian matrix of $F_i(X)$.

For the matrix equations, f_i are the elements of the matrix $F(X)$, taken a row at a time.

(c) to monitor the minimisation process.

2) F04AXF - Sparse System Solver

This subroutine calculates the approximate solution of a set of real sparse linear equations with a single right hand side. The coefficient matrix is decomposed by subroutine F01BRF then F04AXF computes the solution by block forward or backward substitution, using simple forward and backward substitution within each diagonal block [Duff].

Subroutine F01BRF obtains the LU decomposition of a permutation of A,

$$PAQ = LU$$

where P and Q are permutation matrices, L is unit lower triangular and U is upper triangular. The routine uses a sparse variant of Gaussian Elimination and the pivotal strategy is designed to compromise between maintaining sparsity and controlling loss of accuracy through round-off.

Optionally the routine first permutes the matrix into block lower triangular form and then only decomposes the diagonal blocks. For some matrices this gives a considerable saving in storage and execution time.

3) E04ABF - Line Search

This subroutine searches for a low-accuracy minimum, in a given finite interval, of a continuous function of a single variable, using function values only. The method is based on quadratic interpolation and is intended for functions which have a continuous first derivative (although it will usually work if the derivative has occasional discontinuities).

It computes a sequence of values which tend to a minimum of the function. The sequence is terminated when the function is deemed to have been sufficiently reduced as illustrated in the discussion of the line search in Section 6.3.

The subroutines are written in FORTRAN 77 on the Honeywell computer using the Multics operating system. All computations are performed using double precision arithmetic for added precision.

SECTION 7.2: Methods for Computing the Characteristic Polynomial

From the discussion in Chapter 3, the problem of determining the coefficients of the characteristic polynomial of a matrix is well-conditioned suggesting that a stable method will yield an accurate solution for any matrix. It was shown that the stable version of LeVerriers method was the only method that could be considered as stable in this sense. The question therefore arises as to whether there is any need to consider any other method. The answer lies in the fact that the operations count for LeVerriers method is $O(n^4)$ as compared to $O(n^3)$ for the other methods of Chapter 3 which, for large order problems, may prove to be significant. Allied with the fact that the other methods provide an accurate solution for certain classes of matrices, the choice of method for a particular problem may not be straightforward.

We begin by illustrating the failings of some of the methods.

Examples of Failure

Consider the problem of determining the characteristic polynomial of

$$A = \begin{bmatrix} 0.001 & 1 & 2 & 3 \\ 0 & 0.1 & 4 & 7 \\ -1 & 10.0 & 1 & 1 \\ 1 & 0 & 4 & 100 \end{bmatrix} \tag{7.1}$$

The matrix is ill-conditioned and the characteristic polynomial, correct to 4 decimal places is,

$$f(\lambda) = \lambda^4 - 101.101\lambda^3 + 65.2011\lambda^2 + 3520.4339\lambda - 374.0104$$

Using Danilevski's method, the reduction of A to upper Hessenberg form is stable, producing small sub-diagonal elements. However, in the reduction of the Hessenberg matrix to the companion form, these small sub-diagonal elements give rise to errors that are so large that the computed coefficients are wholly inaccurate.

The Block Frobenius method however, is accurate since it recognises the small sub-diagonal elements and reduces the Hessenberg matrix to a block matrix with Frobenius matrices on its diagonal.

Consider the following matrix,

$$A = \begin{bmatrix} 12 & 11 & 10 & \dots & 2 & 1 \\ 11 & 11 & 10 & \dots & 2 & 1 \\ \vdots & & & & \vdots & \vdots \\ 2 & 2 & 2 & \dots & 2 & 1 \\ 1 & 1 & 1 & \dots & 1 & 1 \end{bmatrix} \tag{7.2}$$

The eigenvalues are not well-distributed,

$$\lambda_i = \frac{1}{2}\left[1 - \cos\left(\frac{(2i-1)\pi}{25}\right)\right]^{-1}, \qquad i = 1, \ldots, 12$$

and the exact characteristic polynomial is,

$$f(\lambda) = \lambda^{12} - 78\lambda^{11} + 1001\lambda^{10} - 5005\lambda^9 + 12870\lambda^8 - 19448\lambda^7 + 18564\lambda^6$$
$$- 11628\lambda^5 + 4845\lambda^4 - 1330\lambda^3 + 231\lambda^2 - 23\lambda + 1$$

Using Krylov's method, the linear equation that must be solved therein is ill-conditioned as a result of the eigenvalue distribution of A. Consequently, Gaussian Elimination fails to solve the linear equation and terminates with a division by zero condition, indicating the presence of a very small pivotal element.

Using the Interpolation method, the closeness of the eigenvalues of A results in relatively large rounding errors being produced. If we assume the accuracy to be determined by the norm, $|\hat{a}_i - a_i|$, where $\hat{a}_i$ and a_i are the computed and exact coefficients respectively, then for this example, $|\hat{a}_i - a_i| = 10^{-2}$. That is, a loss of accuracy has resulted as a consequence of the rounding errors.

Consider the following matrix,

$$A = \begin{bmatrix} 6 & 2 & -2 \\ -2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \tag{7.3}$$

This matrix is derogatory since the characteristic polynomial given by,

$$f(\lambda) = \lambda^3 - 10\lambda^2 + 32\lambda - 32$$

does not coincide with the minimal polynomial,

$$m(\lambda) = \lambda^2 - 6\lambda + 8$$

Consider Krylov's method. In exact arithmetic the matrices A^2 and A^3 are,

$$A^2 = \begin{bmatrix} 28 & 12 & -12 \\ -12 & 4 & 12 \\ 12 & 12 & 4 \end{bmatrix}, \qquad A^3 = \begin{bmatrix} 120 & 56 & -56 \\ -56 & 8 & 56 \\ 56 & 56 & 8 \end{bmatrix}$$

However, the error in the computed A^2 and A^3 is less than 8^2 and 8^3 times the machine precision, respectively. That is, the error is small.

Next, we form the matrices T to solve the linear system $Ta = b$ of (3.24) to determine the coefficients a_i. Taking the elements of A, A^2 and I a column at a time, gives

$$T = \begin{bmatrix} -6 & 28 & 1 \\ -2 & -12 & 0 \\ 2 & 12 & 0 \end{bmatrix}, \quad \begin{bmatrix} 2 & 12 & 0 \\ 2 & 4 & 1 \\ 2 & 12 & 0 \end{bmatrix}, \quad \begin{bmatrix} -2 & -12 & 0 \\ 2 & 12 & 0 \\ 2 & 4 & 1 \end{bmatrix}$$

The Gaussian Elimination algorithm recognises that each of these matrices is singular and terminates the processing. Notice that the eigenvalues of the matrices T are well-distributed.

Now consider the matrix

$$A = \begin{bmatrix} 100 & 2 & -2 \\ -2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \tag{7.4}$$

In this case, the matrix T_1, formed from the first column of A, A^2 and I is

$$T_1 = \begin{bmatrix} 100 & 10^4 - 8 & 1 \\ -2 & -200 & 0 \\ 2 & 200 & 0 \end{bmatrix}$$

This matrix is clearly singular. However, using Gaussian Elimination with a low tolerance value, the poor distribution of eigenvalues of T_1 contributes towards the generation of significant rounding errors. At the final stage of the Gaussian elimination process, the element in position (3,3) is very small, but not considered zero. Therefore the computed coefficients are wholly inaccurate.

This example shows that to compute the coefficients accurately, it is not sufficient to let the Gaussian elimination algorithm decide whether the initial matrix is derogatory (by detecting a non-singular matrix T) or not. It is necessary also to investigate the eigenvalue distribution of the matrix.

Operations counts and CPU times

Now we look at the efficiency of the methods with respect to operations counts and CPU times. We begin by re-stating the operations counts and storage requirements as determined in Chapter 3.

Method	Operations Count	Storage
LeVerrier	$n^4 - n^3$	$2n^2 + 2n$
Block Frobenius ($c = 5$)	$6n^3$	$n^2 + n$
Danilevski	$2n^3$	$n^2 + 2n$
Krylov	$\frac{4}{3}n^3 + n^2$	$2n^2 + 2n$
Interpolation	$8n^3 + n^2$	$n^2 + 2n$

The operations counts ignore terms of $O(n)$. An average figure of $c = 5$, corresponding to the number of Frobenius blocks, is taken for the Block Frobenius method. The storage requirements are also stated here but as stated in Chapter 1, mass storage systems are currently widely and cheaply available and, as such, storage limitations are no longer a significant concern.

As discussed in the introduction, a more realistic estimate of the efficiency of an algorithm is the CPU time taken to perform the processing. The following table gives the CPU times and the accuracy attained by each method for computing the coefficients of the characteristic polynomials of various sized matrices, given in Appendix 1.1. The times are given first, in seconds, followed by the accuracy. The machine precision is 10^{-18}. The examples were constructed so that all methods worked to their respective abilities.

n	LeVerrier	Block Frobenius	Danilevski	Krylov	Interpolation
4	0.06	0.05	0.05	0.10	0.07
	0.0	10^{-17}	10^{-17}	10^{-16}	10^{-15}
8	0.32	0.14	0.15	0.13	1.17
	10^{-16}	10^{-17}	10^{-15}	10^{-14}	10^{-15}
12	1.32	0.58	0.45	0.32	0.38
	10^{-15}	10^{-16}	10^{-15}	10^{-13}	10^{-13}
16	3.82	1.06	1.02	0.77	0.79
	10^{-14}	10^{-14}	10^{-13}	10^{-12}	10^{-11}

For matrices of small order, all methods take about the same time to compute the coefficients. For larger problems, LeVerrier's method is significantly slower than the others. This effect was predicted in Chapter 3 since the operations count for LeVerrier's method is of $O(n^4)$ whereas it is of $O(n^3)$ for the other methods. From the accuracy point of view, the LeVerrier, Block Frobenius and Danilevski methods are consistently more accurate than the other two methods.

Remarks

These results, coupled with the earlier discussions on the stability of the methods, indicate that for determining the coefficients of a general matrix accurately, LeVerrier's method should be employed. If efficiency is a significant consideration then the Block Frobenius method is much faster with a little loss in accuracy of the coefficients.

The other methods should not be ignored, however, since the problem may arise from a physical application where it is known that the matrix satisfies certain requirements concerning eigenvalue distribution, defectiveness, singularity, etc, in which case these methods may prove to be much quicker than the Block Frobenius method.

SECTION 7.3: Methods for solving the Quadratic Matrix Equation

In the examples of this section, the problems are all well-conditioned since otherwise no methods can be guaranteed to compute an accurate solution, regardless of how stable they are. Therefore, the accuracy of the computed solutions will be wholly a reflection of the stability of the method used. This is due to the fact that since the methods are iterative, a solution can be refined by further iterations but only to within the accuracy allowed by the presence of rounding errors generated by the algorithm.

Examples of Failure

We begin by studying problems for which some of the methods fail.

Consider

$$X^2 + \begin{bmatrix} -1 & -6 \\ 2 & -9 \end{bmatrix} X + \begin{bmatrix} 0 & 12 \\ -2 & 14 \end{bmatrix} = 0 \tag{7.5}$$

The eigenvalues of the associated quadratic eigenvalue problem satisfy,

$$|\lambda^2 I + P\lambda + Q| = 0$$

such that

$$\lambda = \{1, 2, 3, 4\}$$

From the discussions and example of Section 1.6, this problem has solutions with the following eigenvalue pairings,

$$(1,2),\ (1,3),\ (1,4),\ (2,3),\ (2,4)$$

The Matrix Polynomial algorithm of Section 5.3 fails to compute a solution (7.5) because the problem does not possess a dominant solution.

Using the Elimination method with $a_1 = -3,\ a_2 = 2$ gives

$$RX = -S$$

$$-\begin{bmatrix} 2 & -6 \\ 2 & -6 \end{bmatrix} X = -\begin{bmatrix} 2 & -12 \\ 2 & -12 \end{bmatrix} \tag{7.6}$$

The matrix on the left hand side of (7.6) is singular such that and (7.6) does not possess a unique solution. In fact, (7.6) has an infinite number of solutions. The Gaussian Elimination process terminates with a singularity condition.

Now consider using Davis's Newton Iterations method to solve,

$$X^2 + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X + \begin{pmatrix} -8 & -12 \\ -18 & -26 \end{pmatrix} = 0 \tag{7.7}$$

with

$$X^{(0)} = \begin{pmatrix} 1 & 6 \\ -5 & 1 \end{pmatrix}$$

After 30 iterations, the iterates are not converging. One reason may be that the starting matrix is not good enough.

Consider the same problem with a different starting matrix,

$$X^{(0)} = \begin{pmatrix} -2 & 0 \\ 0 & -0.5 \end{pmatrix}$$

From (5.19) the Newton iterate T_i solves

$$(X^{(0)} + P)T_i + T_i X^{(0)} = F(X^{(0)}) \tag{7.8}$$

Now, from Section (1.4) the problem of solving this Sylvester equation is ill-conditioned if $||J^{-1}||$ is large, where

$$J = (X^{(0)} + P) \otimes I + I \otimes X^{(0)^T}$$

In fact

$$J = \text{ diag } (-3, -1.5, -1.5, 0)$$

is singular, which is an extreme form of ill-conditioning. Therefore the Newton Iterations method in this case gives rise to an ill-conditioned problem (7.8), the solution of which may not exist. Rounding errors may cause J to be non-singular and ill-conditioned in which case the computed solution of (7.8) is inaccurate and may lie outside the problems region of convergence.

As we observed in the examples in Section 6.3, the matrix function minimisation technique of that section worked successfully on the two problems above. In fact, no well-conditioned problems have been found that cause the method to fail.

Convergence and Accuracy

Now we look at the rate of convergence of the methods.

Example 1

Consider the problem of (7.5),

$$X^2 + \begin{bmatrix} -1 & -6 \\ 2 & -9 \end{bmatrix} X + \begin{bmatrix} 0 & 12 \\ -2 & 14 \end{bmatrix} = 0 \tag{7.9}$$

As discussed earlier, this problem does not possess a dominant solution and hence the Matrix Polynomial method does not work.

Using the **Newton's Iterations method**, the starting matrix is

$$X^{(0)} = \left(\frac{||P|| + \sqrt{||P||^2 + 4||Q||}}{2} \right) I \tag{7.10}$$

$$X^{(0)} = 1.52605I$$

If convergence is not achieved in 30 iterations, the strategy is restarted from $X^{(0)} = ||Q||.I$ If this fails to converge, the iterations are restarted from $X^{(0)} = I$ [Davis]. Starting from (7.10), the Newton iterations converge to

$$X = \begin{bmatrix} 4 & 0 \\ 2 & 2 \end{bmatrix}$$

in the following way,

Iteration	FNORM	CPU time (secs)
7	10^{-5}	0.505
8	10^{-10}	0.554
9	10^{-17}	0.607

The convergence here is quadratic. Further iterations cannot improve the accuracy beyond 10^{-18}.

Using the **Elimination method**, the formulae in (4.38) provide the starting point,

$$a_i^{(0)} = (1.4566, 1.4729), \qquad i = 1, 2$$

The method converges to $X = \begin{bmatrix} 3 & 0 \\ 1 & 2 \end{bmatrix}$ in the following way,

Iteration	FNORM	CPU time (secs)
7	10^{-6}	0.602
12	10^{-12}	0.827
13	10^{-16}	1.041

Generally, the convergence here is only linear in the sense that each iteration improves accuracy by one decimal place. As with the Newton Iterations method, an accuracy of 10^{-18} cannot be improved upon.

Using the **sum of squares minimisation** approach, the starting matrix is I and the solution is $X = \begin{bmatrix} 3 & 0 \\ 1 & 2 \end{bmatrix}$,

Iteration	FNORM	CPU time (secs)
5	10^{-4}	0.811
6	10^{-8}	0.898
7	10^{-14}	1.002
8	10^{-18}	1.105

The convergence is quadratic and an accuracy of 10^{-18} is attained.

Using the **new matrix function minimisation** technique of Section 6.3 with starting matrix I, the iterates converge to $\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$, as follows

Iteration	FNORM	CPU time (secs)
4	10^{-2}	0.325
5	10^{-5}	0.407
6	10^{-6}	0.491
8	10^{-6}	0.661

Very few iterations are required to obtain a small FNORM. The maximum accuracy attainable however, is 10^{-6}.

Example 2

Consider solving the test problem [Lancaster, 2],

$$X^2 + PX + Q = 0$$

where

$$P = \begin{bmatrix} 3\alpha & -(1+\alpha^2+2\beta^2) & \alpha(1+2\beta^2) & -\beta^2(\alpha^2+\beta^2) \\ 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

and

$$Q = \begin{bmatrix} -1+2\alpha^2 & \alpha-\alpha(\alpha^2+2\beta^2) & 2\alpha^2\beta^2 & -\alpha\beta^2(\alpha^2+\beta^2) \\ 2\alpha & -(\alpha^2+2\beta^2) & 2\alpha\beta^2 & -\beta^2(\alpha^2+\beta^2) \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\beta = \alpha + 1$$

The latent roots associated with the problem are,

$$\{0,\ \pm i,\ -\alpha,\ \pm(1+\alpha)i,\ \alpha \pm (1+\alpha)i\}$$

When $\alpha = 1$, a dominant solution of this problem exists, being (to 4 significant figures)

$$X_1 = \begin{pmatrix} -1.5047 & 6.9503 & -6.1303 & 15.3257 \\ -0.6660 & -0.2705 & 0.3527 & -0.8816 \\ 0.1355 & -1.3883 & 0.7932 & -1.9829 \\ 0.2506 & -0.1639 & 0.0072 & -0.0181 \end{pmatrix}$$

Another solution is (to four decimal places),

$$X_2 = \begin{pmatrix} -0.1324 & 1.7237 & -0.1729 & 1.0443 \\ -0.6018 & -0.0253 & -0.0043 & -0.0260 \\ 0.0576 & -1.4980 & 0.0995 & -0.6042 \\ 1.1520 & -0.1034 & -0.0037 & -0.0215 \end{pmatrix}$$

The **Newton Iterations method** does not converge from any of the three starting points.

The **Matrix Polynomial method** converges to the dominant solution X_1 in 7 iterations of the first step and 3 iterations of the second step with an accuracy of 10^{-9} and CPU time of 2.480 seconds.

The **Elimination method** converges to X_1, in the following way,

Iteration	FNORM	CPU time (secs)
10	10^{-5}	2.051
15	10^{-12}	2.402
18	10^{-16}	2.721

An improvement in accuracy of one decimal point per iteration is achieved and this time an accuracy of 10^{-16} is attained.

The **sum of squares minimisation** approach converges to X_1, in the following way,

Iteration	FNORM	CPU time (secs)
7	10^{-3}	7.314
8	10^{-6}	8.422
9	10^{-12}	9.596

The convergence is quadratic and an accuracy of 10^{-15} is attained at the 10th iteration.

The matrix function minimisation approach converges to X_2 from the starting matrix I, in the following way,

Iteration	FNORM	CPU time (secs)
3	10^{-2}	0.864
4	10^{-4}	1.109
5	10^{-10}	1.431
6	10^{-15}	1.728

The convergence here is quadratic and the accuracy is 10^{-15} attained at the 6th iteration.

For this problem, a dominant solution exists so long as $\alpha > 0$. Now as $\alpha \to 0$, the functionality of the methods is affected, as follows:

The Newton Iterations method still fails to converge from any of the three starting points. This is because the starting matrices are outside the region of convergence for this problem.

The convergence rate for the Matrix Polynomial algorithm becomes considerably slower since the smallest eigenvalue of the dominant solution approaches the largest eigenvalue of the next solution.

The convergence and accuracy properties of the other three methods remain unaffected.

Example 3

In this example, there is an ill-conditioned solution.

$$X^2 + PX + Q = 0$$

$$P = \begin{pmatrix} 122 & 41 & 40 & 26 & 25 \\ 40 & 170 & 25 & 14 & 24 \\ 27 & 26 & 172 & 7 & 3 \\ 32 & 22 & 9 & 106 & 6 \\ 31 & 28 & -2 & -1 & 165 \end{pmatrix} \quad Q = \begin{pmatrix} -0.0001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1000 \end{pmatrix}$$

In fact, $$X = \begin{pmatrix} 10^{-6} & 0 & 10^{-2} & 0 & -6.218 \\ 10^{-6} & 0 & 10^{-2} & 0 & -4.24 \\ 10^{-6} & 0 & 10^{-2} & 0 & -0.6044 \\ 10^{-6} & 0 & 10^{-2} & 0 & -0.0217 \\ 10^{-6} & 0 & 10^{-2} & 0 & 48.338 \end{pmatrix}$$

with

$$\lambda(X) = \begin{pmatrix} 48.338 \\ 0.006177 \\ 10^{-7} \\ 10^{-19} \\ 10^{-23} \end{pmatrix}, \qquad \lambda(P+X) = \begin{pmatrix} 79.313 \\ 251.88 \\ 112.244 \\ 143.413 \\ 196.494 \end{pmatrix}$$

All four methods converge to this solution in the following way,

Newton Iterations method:

Iteration	FNORM	CPU time (secs)
18	10^{-4}	5.014
22	10^{-10}	5.578
25	10^{-15}	5.973

Matrix Polynomial algorithm:

3 iterations of the first step and at the second step,

Iteration	FNORM	CPU time (secs)
2	10^{-4}	1.114
3	10^{-7}	2.212
4	10^{-11}	3.320

sum of squares minimisation:

Iteration	FNORM	CPU time (secs)
11	10^{-3}	9.310
14	10^{-6}	12.301
17	10^{-13}	15.491

Elimination method:

Iteration	FNORM	CPU time (secs)
7	10^{-6}	3.301
9	10^{-9}	4.011
11	10^{-12}	4.608
14	10^{-16}	5.685

matrix function minimisation:

Iteration	FNORM	CPU time (secs)
2	10^{-2}	0.442
3	10^{-8}	0.837
4	10^{-14}	1.114
5	10^{-15}	1.442

The accuracy attained by all the methods is consistently high implying that the effect of rounding errors in their computations is relatively small. For this particular problem, the convergence for the Newton Iterations method and the sum of squares minimisation approach is no longer quadratic, probably due to the ill-conditioning of the solution matrix and the Jacobian near to the solution. Notice also, that this has not affected the matrix function minimisation technique which exhibits very fast convergence.

CPU times and operations counts

Now we look at the efficiency of the methods with respect to the operations counts and CPU times for algorithm execution, beginning with a review of the operations counts and storage requirements for each method, as determined in the previous chapters.

Method	Operations Count	Storage
Newton Iterations	$21n^3m$	$5n^2$
Matrix Polynomial	$6n^3m$	$6n^2$
Elimination Method	$(2n^4 + 6n^3)m$	$6n^2 + n$
Sum of Squares Minimisation	$(\frac{1}{3}n^6 + n^4 + 5n^3)m$	$n^4 + 4n^2$
New (Matrix Function) Minimisation	$(\frac{5}{4}n^4 + (26 + r)n^3)m$	$n^4 + 8n^2$

where m is the number of iterations and r is the number of function evaluations for the line search, at each iteration. Terms of $O(n^2)$ and $O(n)$ are ignored in the operations counts.

The methods were used to solve problems of orders 6, 8 and 11 (see Appendix 1.2). The stopping criterion for the iterations is activated when the function norm is less than 10^{-8}. The following table gives the results obtained.

	n	Elimination	Newton	Matrix Polynomial	Sum of Squares Minimisation	New Minimisation
function norm		10^{-10}	10^{-12}	10^{-8}	10^{-10}	10^{-11}
iterations	6	10	12	5 and 10	10	9
CPU time		7.0	7.2	3.9	15.2	9.1
function norm		10^{-10}	10^{-13}	10^{-9}	10^{-9}	10^{-12}
iterations	8	11	14	6 and 9	12	8
CPU time		10.0	15.2	5.4	28.2	17.9
function norm		10^{-9}	10^{-11}	10^{-9}	10^{-9}	10^{-11}
iterations	11	11	14	8 and 11	14	10
CPU time		18.9	29.3	10.6	48.0	33.5

These results indicate that the quickest method is the Matrix Polynomial one, where the execution time of the stage 2 algorithm is about twice that for the stage 1 algorithm. The next fastest is the Elimination method where most time is taken in determining the matrices R_{n-1} and S_{n-1}. The next quickest algorithm is the Newton Iterations approach followed by the New minimisation and the sum of squares minimisation methods. For the New minimisation method, the problem of determining the singular values of the Jacobian contributes over 30% to the total CPU time. Notice that although the Newton Iterations method and the New minimisation method are essentially Newton methods, the use of a line search technique in the latter significantly reduces the number of iterations performed.

It is interesting to observe, firstly that the relative difference between the CPU times for the methods corresponds to the relative differences between the operations counts for the methods and secondly that although the CPU times increase rapidly as n increases, the number of iterations required for convergence to occur does not necessarily increase.

Remarks

In practice, a dominant solution to a problem may not exist, in which case although the Matrix Polynomial algorithm is the fastest method, it will not yield a solution.

It maybe that none of the three pre-defined points used to initiate the Newton Iterations method is sufficiently 'close' to the solution for convergence to occur. Clearly other starting points can be used but there is no way to predict whether a particular point leads to convergence or not. Also, at any iteration, an ill-conditioned Sylvester equation, in the form of an ill-conditioned Jacobian, may need to be solved. In this case either the

condition is detected and the iterations terminated prematurely or the computed solution may be inaccurate and lie outside the region of convergence for the problem.

The Elimination method, however, is not reliant on a suitable starting matrix and has been shown to work on virtually all problems considered. The exceptions arise when, at any iteration, the computed matrix R_{n-1} is ill-conditioned or singular. It is not known beforehand whether this situation will occur or not and all we can do in this case is to restart the iterations from a different starting point.

The sum of squares minimisation approach is globally convergent. However, the iterations may converge to a point that is not a solution to the problem in which case the iterations are restarted from a different starting point.

The New minimisation approach also suffers from this disadvantage. The problem of an ill-conditioned Jacobian at an iteration is overcome by using the Steepest Descent direction, for that iteration only. However, if the Jacobian is singular, then the direction is no longer a descent one and the computed estimate to the solution may not be any closer to the solution than the previous estimate. Additionally, subsequent Jacobians may prove to be ill-conditioned or singular. Most importantly however, if the Jacobian is non-singular then the method is globally convergent from any starting point.

The remarks above in conjunction with the CPU time analysis, indicate that the Elimination method and the New minimisation method do not suffer from any disadvantages additional to those exhibited by the Newton Iterations method and the Matrix Polynomial algorithm. In fact, both methods possess certain 'good' convergence properties.

SECTION 7.4: Methods for the Matrix Square Root Problem

As discussed in Chapter 5, there are a number of methods in existence that compute the square root of a matrix, whether it is a general matrix or a special matrix. The purpose of this section is to compare the efficiency of the Elimination methods of Chapter 4 with those of Chapter 5.

The different Elimination methods are,

Elimination 1 is the Elimination method applied iteratively

Elimination 2 is the method based upon the symmetric functions

Elimination 3 is the method that determines the coefficients of the characteristic polynomial of X by solving a set of n linear equations.

The Matrix Square Root problem is one of solving,

$$X^2 - A = 0$$

We begin with some examples.

Selected Examples

Example 1

Consider

$$X^2 - \begin{bmatrix} 4 & 1 & -1 \\ 1 & 3 & 0 \\ -1 & 0 & 5 \end{bmatrix} = 0$$

The eigenvalues of A are $4,\ 4+\sqrt{3},\ 4-\sqrt{3}$. All methods converge to

$$X = \begin{bmatrix} 1.9667 & 0.2730 & -0.2398 \\ 0.2730 & 1.7100 & 0.0166 \\ -0.2398 & 0.0166 & 2.2223 \end{bmatrix}$$

in the following way,

Method	No. of iterations	CPU time (secs)	Function norm
Elimination 1	8	0.245	10^{-10}
Elimination 2	—	0.042	10^{-18}
Elimination 3	9 Newton iterations	0.077	10^{-10}
Newton	5	0.410	10^{-14}
Sign function	6	0.309	10^{-16}
Schur vectors	—	0.092	10^{-15}
Minimisation	6	0.511	10^{-16}

For the Newton method and the minimisation method, the initial estimate is

$$X^{(0)} = ||A||I$$

Example 2

$$X^2 - \begin{pmatrix} 1.2 \times 10^5 & 230 & 10 \\ 230 & 1.0 \times 10^3 & 1 \\ 10 & 1 & 0.5 \end{pmatrix} = 0, \qquad \lambda(X) = \begin{pmatrix} 1.2 \times 10^5 \\ 1.0 \times 10^3 \\ 0.498 \end{pmatrix}$$

Here, the eigenvalues of the coefficient matrix are very different from each other such that the condition number with respect to inversion is large.

All methods converge to

$$X = \begin{pmatrix} 346.41 & 0.6084 & -0.0288 \\ 0.6084 & 31.6169 & -0.0304 \\ -0.0288 & -0.0304 & 0.7050 \end{pmatrix}$$

in the following way,

Method	No. of iterations	CPU time (secs)	Function norm
Elimination 1	35	0.9091	10^{-1}
Elimination 1	60	1.318	10^{-10}
Elimination 2	—	0.030	10^{-14}
Elimination 3	11 Newton iterations	0.091	10^{-11}
Newton	5	0.411	10^{-14}
Sign function	7	0.329	10^{-16}
Schur vectors	—	0.120	10^{-12}
Minimisation	8	0.534	10^{-14}

The eigenvalues of the solution are,

$$\{346.411, \quad 31.616, \quad 0.706\}$$

The characteristic polynomial is,

$$f(\lambda) = \lambda^3 - 378.733\lambda^2 + 11219.016\lambda - 7732.204$$

The error analysis of Chapter 4 relating to the Elimination method and in particular the relationships (4.69) and (4.70) suggest that rounding errors may contaminate the solution if the determinant of the matrix X is large. Therefore the fact that the determinant of X is large and the problem has a spectral condition number of 2.4×10^6 may explain the slow convergence of Elimination 1. Nevertheless, as indicated in the table above, greater accuracy of solution can be obtained at the expense of many more iterations.

It is clear that the condition of the problem has not affected the convergence properties of the other methods.

Example 3

$$X^2 - \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 2 & -3 & 2 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 3 \end{bmatrix} = F(X) = 0$$

Here, the coefficient matrix is singular and the derivative, as defined in Chapter 2 is also singular. The methods converge as follows,

Method	No. of iterations	CPU time (secs)	Error norm
Elimination 1	40	1417.8	10^{-10}
Elimination 2	—	270.9	10^{-13}
Elimination 3	55 Newton iterations	1011.7	10^{-12}
Newton	100	NO CONVERGENCE	
Sign function	100	NO CONVERGENCE	
Schur vectors		22.3	10^{-14}
Minimisation	100	NO CONVERGENCE	

Clearly, the fact that the coefficient matrix is given in a Schur form reduces the work required by the Schur vectors method.

The Schur vectors method computes the square root with no problems but if the matrix had 2 zero eigenvalues then this method would have failed, as observed in Section 5.4.3.

The fact that the Jacobian matrix associated with the minimisation method is singular at the root explains why the method has failed. Consequently all those methods based on Newton iterations fail to converge.

The Jacobian matrix associated with the linear equations in Elimination 3 is singular. However, the iterations do converge, although very slowly.

This example illustrates the fact that the Elimination methods may be used to accurately compute the square roots of a matrix when other well-established methods fail.

Operations Counts and CPU times

The table below summarises the operations counts and storage requirements for each method.

Method	Operations Count	Storage Requirements
Elimination 1	$(n^4 + 7n^3 - 2n^2)m$	$5n^2 + n$
Elimination 2	$n^4 + 16n^3 + \sum_{i=1}^{n}(i-1)\frac{n!}{i!(n-i)!}$	$4n^2 + 2n$
Elimination 3	$n^4 + 2n^3 + (3n^3 + 6n^2)m$	$3n^2 + n$
Newton	$14n^3m$	$3n^2$
Sign function	$3n^3m$	$6n^2$
Schur vectors	$18n^3$	$4n^2$
Minimisation	$(\frac{1}{3}n^6 + n^4 + 5n^3)m$	$n^4 + 3n^2$

where m is the number of iterations.

The methods were used to compute the square roots of various matrices, See Appendix 1.3, with the following results,

Method, $n = 8$	No. of iterations	CPU time (secs)	Function norm
Elimination 1	10	4.1	10^{-15}
Elimination 2	—	0.6	10^{-17}
Elimination 3	9 Newton iterations	0.9	10^{-11}
Newton	7	2.2	10^{-15}
Sign function	6	1.4	10^{-15}
Schur vectors	—	0.7	10^{-11}
Minimisation	9	8.0	10^{-14}

Method, $n = 12$	No. of iterations	CPU time (secs)	Function norm
Elimination 1	9	7.2	10^{-10}
Elimination 2	—	5.1	10^{-15}
Elimination 3	12 Newton iterations	5.4	10^{-10}
Newton	11	7.8	10^{-13}
Sign function	9	3.2	10^{-13}
Schur vectors	—	2.9	10^{-10}
Minimisation	8	17.4	10^{-14}

Method, $n = 16$	No. of iterations	CPU time (secs)	Function norm
Elimination 1	11	23.3	10^{-11}
Elimination 2	—	61.1	10^{-9}
Elimination 3	15 Newton iterations	11.4	10^{-9}
Newton	12	15.6	10^{-12}
Sign function	9	6.7	10^{-13}
Schur vectors	—	4.2	10^{-8}
Minimisation	10	36.0	10^{-12}

For problems of a general order, the sign function method and the Schur vectors method provide the quickest means for computing the square roots. However, for small order problems ($n < 10$), Elimination 2 is very quick and very accurate, although for larger problems its CPU time increases significantly. The minimisation method is too slow to be of much use.

Observe that as n increases, Elimination 2, Elimination 3 and the Schur vectors methods increasingly lose accuracy. This is because they are 'exact' methods as opposed to the other iterative methods, the latter also acting as solution-refining algorithms.

From the discussion in Section 5.4, the matrix sign function method fails if the coefficient matrix possesses a real negative eigenvalue and the Newton method requires a starting matrix 'close' to the solution and a well-conditioned Jacobian at each iteration otherwise converge may be very slow or may not occur at all.

Therefore, the results and analysis of this section show that for certain problems it may be better to use the Elimination methods rather than the others, as illustrated by the behaviour of Elimination 2 for small problems and by the results of Example 2.

SECTION 7.5: Methods for the Algebraic Riccati Equation

The operations count and storage requirements for each method are given in the following table,

Method	Operations Count	Storage Requirements
Eigenvector Method	$100n^3$	$10n^2$
Newton Iterations	$(25m + 16)n^3$	$9n^2$
Schur Vectors	$75n^3$	$8n^2$
Sparse-Newton Method	$(23n^3 - 2n^2)m$	$12n^2$

where m is the number of iterations. The operations count given above, for the new Sparse-Newton method of Chapter 6, is for problems with real Jacobians at each iteration. Now consider the following example.

Example

$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} X + X \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} - X \begin{pmatrix} 0 \\ 1 \end{pmatrix} (0 \quad 1) X + \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} = 0$$

where $G = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \geq 0$ as required.

and $H = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} > 0$ as required.

Method	No. of iterations	CPU time (secs)	Function norm
Eigenvector method	—	0.3	10^{-11}
Newton Iterations	5	0.4	10^{-12}
Schur Vectors	—	0.5	10^{-10}
Sparse-Newton method	4	0.5	10^{-9}

Notice that although the Newton method and the Sparse-Newton method are based upon Newton iterations, they converge after a different number of iterations as a consequence of updating the solution matrix in different ways.

Since the algebraic Riccati equation of optimal control is always assumed to be stabilizable and observable, a non-negative definite solution always exists.

The methods were used to solve a number of problems with the following results. The coefficient matrices and the solution matrix are given in Appendix 1.4.

Method, $n = 4$	Iterations	CPU time (secs)	Function norm
Eigenvector	—	0.6	10^{-13}
Newton Iterations	5	1.8	10^{-16}
Schur Vectors	—	0.6	10^{-16}
Sparse-Newton	6	1.2	10^{-16}

Method, $n = 8$	Iterations	CPU time (secs)	Function norm
Eigenvector	—	2.7	10^{-12}
Newton Iterations	7	4.3	10^{-15}
Schur Vectors	—	3.1	10^{-14}
Sparse-Newton	8	3.9	10^{-16}

Method, $n = 12$	Iterations	CPU time (secs)	Function norm
Eigenvector	—	6.6	10^{-10}
Newton Iterations	11	13.2	10^{-15}
Schur Vectors	—	6.1	10^{-13}
Sparse-Newton	9	10.8	10^{-15}

The problems were chosen to be stabilizable and observable and well-conditioned such that all methods converged to a solution. Clearly the vector methods are much quicker than the Newton-based methods, particularly for increasing n. However, since the vector methods are non-iterative, they are less accurate than the iterative methods, which are solution-refining. The implication here is that any significant rounding errors in the vector methods may contaminate the solution.

The Schur vectors method is faster than the eigenvector method and is more stable since from Section 5.2, the eigenvectors are more likely to be ill-conditioned than the Schur vectors.

The new iterative method based on a Newton step and the solution of a sparse linear system at each iteration, converges faster than Kleinman's Newton method. This is because the update is computed by a more efficient technique, provided that the Jacobian possesses real eigenvalues at each iteration.

The conclusion is that the vectors approach, in particular the Schur vectors method, is in general the best technique for computing a solution of the algebraic Riccati equation. However, for when the associated Hamiltonian matrix possesses ill-condition eigenvectors and/or Schur vectors, then the iterative methods may prove to be the most efficient and most accurate.

No mention of the Sign Function method has been presented here since the reformulation of this method in Section 5.2 is a relatively new technique. However, all indications suggest that it compares favourably with the Schur vectors approach [Byers, 3].

CHAPTER 8: CONCLUSIONS

The initial aim of this thesis was to present an analysis of the Elimination method for the numerical solution of the general unilateral quadratic matrix equation (QME) and to compare its numerical and functional properties with those of existing methods. Progressively, the field of study widened to include two other types of quadratic matrix equation, the matrix square root and the algebraic Riccati equation (ARE). In addition to the original aims two new approaches to solving the QME and the ARE were proposed.

Central to the numerical issues were the discussions on conditioning, stability and accuracy. Conditioning analysis was performed not only on the main problems of solving the matrix equations but also on the various problems arising from within the methods. This analysis along with the stability analysis of the methods determined the accuracy of the methods.

In these days of large-order systems and the requirement for fast problem-solving algorithms, the speed of the methods was considered to be a significant factor in comparing the methods. Two units of measure were used in this thesis, the operations count and the Central Processor Unit (CPU) time.

Current methods for the solution of the QME will not always provide a solution, the Newton Iterations approach relying on a good starting point for the iterations and the Matrix Polynomial algorithm succeeding only on problems possessing a dominant solution. The Elimination method was shown to compare favourably with the Newton Iterations method with respect to accuracy and efficiency. It was shown to work on problems for which the other methods failed. Significantly, the Elimination method requires only n values as a starting point to the iterations compared with n^2 for the other methods. The analysis of the Elimination method raised some interesting points which may form the basis of further study, as follows,

conditions on the convergence of the Elimination method
accelerating the convergence from near the solutions
restarting algorithms for when the Elimination method fails

A new minimisation method was proposed. This was shown to work on problems for which other methods failed and shown to be globally convergent and to yield accurate solutions. The sum of squares minimisation approach is impractical for large problems but for when no other method will work, this approach should be used.

In this thesis, we have studied a number of methods for the numerical solution of the matrix square root problem including three new approaches based on the Elimination method. The analyses of Chapters 4 and 5 showed that all the methods possesses at least one limitation that prevents them for computing a solution and as such no one method can be guaranteed to always yield a solution. For general well-conditioned problems, the

Schur vectors approach was shown to be the quickest and the most stable of all the methods considered. However, it yielded solutions that were generally less accurate than those computed by the iterative, self-refining methods. The Elimination methods did not, in general, compare favourably against the best of the existing methods. One exception is Elimination 2 which was shown to be very fast and accurate for lower order problems. The sum of squares minimisation technique is computationally too expensive to be considered seriously although there may be case for it to be used when the coefficient matrix is such that no other method can expect to compute a solution. For problems where information pertaining to the coefficient matrix is known beforehand, particular methods may be employed.

The three existing methods for solving the ARE discussed in this thesis have all had varying levels of success. The eigenvector method has recently been superceded by the Schur vectors method because the latter is more stable and slightly faster as borne out by the experiments in Chapter 7. However, since it is an exact method rather than an iterative one, it does tend to lose accuracy for larger problems. The Newton iterations method is slower than the Schur vectors method but is more accurate. Therefore the choice of method is dependent on whether accuracy or efficiency is of greater importance. Chapter 6 proposed a new approach to solving the ARE that is based on a reformulation of Newton's method. This new approach is shown to be stable and to require fewer iterations than Newton's method to converge. However, it is more efficient than Newton's method only under certain conditions.

To summarise, this thesis has studied the Elimination method for solving the QME and the matrix square root problem. Additionally, it has proposed two new approaches to solving the QME and ARE. Finally, it has compared many methods for computing the coefficients of the characteristic polynomial of a matrix and for solving the three types of matrix equations mentioned above, basing the comparison on the functionality, stability and efficiency of the methods and the accuracy of the computed solution.

REFERENCES

ANDERSON:

'Second Order Convergent Algorithms for the Steady State Riccati Equation',
International Journal of Control, Vol. 28, pp. 295-306, 1978.

ARMSTRONG:

'An Extension of Bass' Algorithm for Stabilising Linear Continuous Constant Systems',
IEEE Transactions on Automatic Control, Vol. AC-20, pp. 629-631, 1976.

ARNOLD, LAUB:

'Generalised Eigenproblem Algorithms and Software for Algebraic Riccati Equations',
Proceedings of the IEEE, Vol. 72, No.12, pp. 1746-1754, 1984.

BALZER:

'Accelerated Convergence of the Matrix Sign Function Method of Solving Lyapunov, Riccati and other Matrix Equations',
International Journal on Control, Vol. 32, No.6, pp 1057-1078, 1980.

BARNETT:

'Matrices in Control Theory',
Van Nostrand Reinboldt, 1971.

BARNETT, STOREY:

'Matrix Methods in Stability Theory',
Nelson, 1970.

BARTELS, STEWART:

'Solution of the Matrix Equation $AX + XB = C$',
Communications of ACM, Vol. 15, No.9, Sept. 1972.

BICKLEY, McNAMEE:

'Matrix and Other Direct Methods for the Solution of Systems of Linear Difference Equations',
Philos. Trans. Roy. Soc. London, Ser. A, Vol. 252, pp. 69-131, 1960.

BIERMAN:

'Computational Aspects of the Matrix Sign Function of the ARE',
Factorised Estimation Applications, Inc. 7017, Deveron Ridge Road, Canoga Park, California 91301, 1984.

BJÖRCK, HAMMARLING:

'A Schur Method for the Square Root of a Matrix',
Linear Algebra and its Applications, Vol. 52/53, pp. 127-140, 1983.

BJÖRCK, PEREYRA:
'Solution of Vandermonde Systems of Equations',
Vol. 24, No.112, October 1970.

BROWN, DENNIS:
'Derivative-Free Analogues of the Levenberg-Marquard and Gauss Algorithms for Non-Linear Least Squares Approximation',
Numerische Mathematik, Vol. 18, pp. 289-297, 1972.

BUNSE-GERSTNER, MEHRMANN:
'A symplectic QR-like Algorithm for the Solution of the Real Algebraic Riccati Equation',
Fakultät Für Mathematik, Universität Bielefeld, Postfach 8640, 4800 Bielefeld 1, Germany.
To appear in the IEEE.

BYERS, 1:
'Numerical Condition of the Algebraic Riccati Equation',
Contemporary Mathematics, Vol 47, pp. 35-49, 1985.

BYERS, 2:
'Hamiltonian and Symplectic Algorithms for the Algebraic Riccati Equation',
Ph.D. Dissertation, Cornell University, Ithaca, New York, 1983.

BYERS, 3:
'Numerical Stability and Instability in Matrix Sign Function Based Algorithms',
Computational and Combinational Methods in Systems Theory, Elsevier Science Publishers BV. (North-Holland), pp. 185-201, 1986.

BYERS, 4:
'Solving the Algebraic Riccati Equation with the Matrix Sign Function',
Linear Algebra and its Applications, Vol. 85, pp. 267-279, 1987.

BYERS, 5:
'A Hamiltonian QR Algorithm',
SIAM Journal of Scientific and Statistical Computing, Vol. 7, No.1, pp. 212-229, January 1986.

BYERS, MEHRMANN:
'Symmetric Updating of the Solution of the Algebraic Riccati Equation',
Procedures of the 10th Symposium on Operations Research, Universitat Munchen, pp. 117-125, 1985.

CLINE et al:
'An Estimate for the Condition Number of a Matrix',
SIAM Journal of Numerical Analysis, Vol. 16, No.2, pp. 368-375, 1979.

CROSS, LANCASTER:
'Square Roots of Complex Matrices',
Linear and Multilinear Algebra, Vol. 1, pp, 289-293, 1974.

DANILEVSKI:
'On the Reduction of a General Matrix to Frobenius Form',
O cislennom resenii vekovogo uravnenija mat. sb., Vol. 2(44) pp. 169-171, 1931.

DAVIS, 1:
'Numerical Solution of a Quadratic Matrix Equation',
SIAM Journal of Scientific and Statistical Computations, Vol. 2, pp. 164-175, 1981.

DAVIS, 2:
'Algorithm 598 - An Algorithm to Compute Solvents of the Matrix Equation $AX^2 + BX + C = 0$',
ACM Transactions on Mathematical Software, Vol. 9, No.2, pp. 246-254, June 1983.

DENMAN, BEAVERS:
'The Matrix Sign Functions and Computation in Systems',
Applied Mathematics and Computation, Vol. 2, pp. 63-94, 1976.

DENNIS, SCHNABEL:
'Numerical Methods for Unconstrained Optimisation and Non-Linear Equations',
Prentice-Hall, 1983.

DENNIS, TRAUB, WEBER, 1:
'The Algebraic Theory of Matrix Polynomials',
SIAM Journal of Numerical Analysis, Vol. 13, pp. 831-845, 1976.

DENNIS, TRAUB, WEBER, 2:
'Algorithms for Solvents of Matrix Polynomials',
SIAM Journal of Numerical Analysis, Vol 15, pp. 523-533, 1978.

DIEF:
'Advanced Matrix Theory',
Abacus 1982.

DOU:
'Method of Undetermined Coefficients in Linear Differential Systems and the Matrix Equation $YA - AY = F$',
SIAM Jour. App. Math, 14, pp. 691-696, 1966.

DUFF:
'MA28 - A Set of FORTRAN Subroutines for Sparse Unsymmetric Linear Equations',
AERE report R. 8730, HMSO, 1977.

FADDEEV, FADDEEVA:
'Computational Methods of Linear Algebra',
Freeman, 1959.

FORSYTHE, MOLER:
'Computer Solution of Linear Algebraic Systems',
Prentice-Hall, 1967.

GANTMACHER:
'Matrix Theory',
Volumes I and II, Chelsea, 1959.

GAUTSCHI:
'On Inverses of Vandermonde Matrices',
Numer. Math., 24, pp. 1-12, 1975.

GILL, MURRAY:
'Algorithms for the Solution of Non-Linear Least Squares Problems',
SIAM Journal on Numerical Analysis, 15, pp. 977-992, 1978.

GOHBERG, LANCASTER, RODMAN:
'Matrix Polynomials',
Academic Press, 1982.

GOLUB, VAN LOAN:
'Matrix Computations',
John Hopkins, 1983.

GOLUB, NASH, VAN LOAN:
'A Hessenberg-Schur Method for the Problem $AX + XB = C$',
Transactions on Automatic Control, Vol. AC-24, No.6, pp. 909-913, 1979.

GRAHAM:
'Kronecker Products and Matrix Calculus',
John Wiley, 1981.

HAMMARLING:
'Newton's Method for Solving the Algebraic Riccati Equation',
NPL Report DITC 12/82, September 1982.

HEWER, NAZAROFF:
'A Survey of Numerical Methods for the Solution of Algebraic Riccati Equations',
Naval Weapons Center Report, China Lake, California.

HIGHAM, 1:
'Error Analysis of the Björck-Pereyra Algorithms for Solving Vandermonde Systems'
Numerische Mathematik, Vol. 50, pp 613-632, 1987.

HIGHAM, 2:
'Fast Solution of Vandermonde-Like Systems Involving Orthogonal Polynomials',
IMA Journal of Numerical Analysis, Vol. 8, pp. 473-486, 1988.

HIGHAM, 3:
'Computing Real Square Roots of a Real Matrix',
Linear Algebra and its Applications, Vol. 88/89, pp. 405-430, 1987.

HIGHAM, 4:
'Newton's Method for the Matrix Square Root',
Mathematics of Computation, Vol. 46, No.174, pp 537-549, April 1986.

HIGHAM, 5:
'Computing the Polar Decomposition with Applications',
SIAM Journal of Scientific and Statistical Computing, Vol. 7, No.4, pp. 1160-1174, 1986.

HOSKINS, WALTON:
'A Faster Method for Computing the Square Root of a Matrix',
IEEE Transactions on Automatic Control, Vol. AC-23, No.3, June 1978.

HOUSEHOLDER:
'The Theory of Matrices in Numerical Analysis',
Ginn-Blaisdell, 1964.

HOWLAND:
'The Sign Matrix and the Separation of Matrix Eigenvalues',
Linear Algebra and its Applications, Vol. 49, pp. 221-232, 1983.

INGRAHAM:
'Rational Methods in Matrix Equations',
Bull. Amer. Math. Soc., Vol. 47, pp. 61-70, 1941.

KALMAN:
'Contributions to the Theory of Optimal Control',
Proceedings of the Conference on O.D.E.'s, Society of Math. Mexicana, Mexico City, 1959.

KLEINMAN:
'On an Iterative Technique for Riccati Equation Computation',
IEEE Transactions on Automatic Control, Feb. 1968.

KRATZ, STICKEL:
'Numerical Solutions of Matrix Polynomial Equations by Newton's Method',
IMA Journal of Numerical Analysis, Vol. 7, pp. 255-369, 1987.

LANCASTER, 1:
'The Theory of Matrices',
Academic Press, 1983.

LANCASTER, 2:
'Lambda Matrices and Vibrating Systems',
Pergaman Press, 1966,

LASALLE, LEFSCHETZ:
'Stability by Lyapnov's Direct Method with Application',
Academic Press, New York, 1961.

LAUB:
'A Schur Method for Solving Algebraic Riccati Equations',
IEEE Transactions on Automatic Control, Vol. AC-24, No.6, Dec. 1979.

LAUB, MEYER:
'Canonical Forms for Hamiltonian and Symplectic matrices',
Celestial Mechanics, Vol. 9, pp. 213-238, 1974.

LAWSON, HANSON:
'Solving Least Squares Problems',
Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

LEVINE, ATHANS:
'On the Determination of the Optimal Constant Output Feedback Gains for Linear Multivariable Systems',
IEEE Trans. Aut. Cont., AC-15, pp. 44-48, 1970.

LUENBERGER:
'Observing the State of a Linear System',
IEEE Trans. Aut. Cont. Mil-8, pp. 74-80, 1964.

McDONALD:
'A Study of Matrix Equations',
PH.D, Thesis, Loughborough University of Technology, 1987.

MacDUFFEE:
'The Theory of Matrices',
Chelsea, 1956.

MILNE:
'Applied Functional Analysis - An Introductory Treatment',
Pitman, 1980.

MURRAY:
'Numerical Methods for Unconstrained Optimisation',
Academic Press, 1972.

NAG:
Numerical Algorithms Group Subroutines for Numerical Computations,
256 Banbury Road, Oxford OX2 7DE.

ORTEGA, RHEINBOLDT:
'Iterative Solution of Non-Linear Equations in Several Variables,
Academic Press, 1970.

PAIGE, VAN LOAN:
'A Schur Decomposition for Hamiltonian Matrices',
Linear Algebra and its Applications, Vol. 41, pp. 11-32, 1981.

PETKOV, CHRISTOV, KONSTANTINOV:
'On the Numerical Properties of the Schur Approach for Solving the Matrix Riccati Equation'
Systems and Control Letters, Vol. p, pp. 197-201, 1987.

POTTER:
'Matrix Quadratic Solution',
SIAM Journal of Applied Math., Vol, 14, No.3, May 1966.

POWELL:
'A Hybrid Method for Non-linear Equations',
in Numerical Methods for Non-linear Algebraic Equations, ed. P. Robinowitz, Gordon and Breach, pp. 87-114, 1970.

REID:
'Riccati Differential Equations',
Academic Press, 1972.

ROBERTS:
'Linear Model Reduction and Solution of the Algebraic Riccati Equation by use of the Sign Function',
International Journal of Control, Vol. 32, No.4, pp. 677-687, 1980.

SIMA:
'An Efficient Schur Method to Solve the Stabilising Problem',
IEEE Transactions on Automatic Control, Vol. AC-26, No.3, June 1981.

SMITH et al:
'EISPACK Guide',
Springer-Verlag, New York, 1974.

STERNBERG, KAUFMAN:
'Applications of the Theory of Systems of Differential Equations to Multiple Non-Uniform Transmission Lines',
Journal of Math. and Phys., Vol. 3, pp. 244-252, 1952.

STEWART, 1:
Introduction to Matrix Computations',
Academic Press, 1973.

STEWART, 2:
'Fortran Subroutines for Calculating and Ordering the Eigenvalues of a Real Upper Hessenberg Matrix',
ACM Transactions on Mathematical Software, 2, pp. 275-280, 1976.

TRAUB:
'A Class of Globally Convergent Iteration Functions for the Solution of Polynomial Equations',
Mathematics of Computation, Vol. 20, pp. 113-138, 1966.

TUEL:
'On the Transformation to (Phase-Variable) Canonical Form',
IEEE Transactions on Automatic Control, Vol, AC-11, pp. 607, 1966.

TURNBULL:
'The Theory of Equations',
Oliver and Boyd, 1939.

WALSH:
'Methods of Optimisation',
John Wiley, 1975.

WANG, CHEN:
'On the Computation of the Characteristic Polynomial of a Matrix',
IEEE Transactions on Automatic Control, Vol. AC-27 pp. 449-451, April 1982

WILKINSON:
'The Algebraic Eigenvalue Problem',
Oxford, 1965.

WONHAM:
'On a Matrix Riccati Equation of Stochastic Control',
SIAM Journal of Control, Vol. 6, pp. 681-697, 1968.

XINOGALAS, DASIGI, SINGH:
'Computational Study of Six Methods for Solving $BX + XA = C$ and $A^TX + XA = C$',
Control System Centre, UMIST, 1982.

APPENDIX 1

A1.1 : The Characteristic Polynomial

The characteristic polynomial of a matrix A is given by,

$$f(\lambda) = \lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \ldots\ldots + a_n$$

N = 4

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}$$

The eigenvalues are :

30.2886853458
3.8580574559
0.1015004840
0.8431071499

The coefficients are :

-35.0
146.0
-100.0
1.0

N = 8

$$A = \begin{bmatrix} -1 & -5 & 3 & 7 & -9 & -2 & -8 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 2 & 1 & -3 & -100 & -0.3 \\ 0 & 0 & 1 & -5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 1 & -0.1 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 1 & -0.01 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -5 \end{bmatrix}$$

The eigenvalues are :

-9.9972305317
-5.8264679893 + 0.46691640328 i
-5.8264679893 - 0.46691640328 i
-1.0370943194 + 0.22347318691 i
-1.0370943194 - 0.22347318691 i
2.6044226916
1.9946555455
-4.9847230879

The coefficients are :

24.110
214.6410
737.3540
-738.5570
-8533.8360
-10641.3360
-2969.6770
87598.2150

N = 12

$A = (a_{ij}) = 1 + j/i$

The eigenvalues are :

22.8774881521
-3.7938972379
0.6630323284 + 0.9245702233 i
0.6630323284 - 0.9245702233 i
0.0552050803 + 0.3093216119 i
0.0552050803 - 0.3093216119 i
0.5636960544 + 0.5750463850 i
0.5636960544 - 0.5750463850 i
0.5897965489 + 0.2526872626 i
0.5897965489 - 0.2526872626 i
0.6655561223
0.5073929395

The coefficients are :

-24.0
19.0
180.0
-672.0
1210.0
-1370.0
1051.0
-560.0
209.0
-55.0
9.0
-1.0

N = 16

$A = a_{ij} = n/(i + j)$

The eigenvalues are :

```
 18.4431977471
  2.7571118200
 -1.9732247968
  1.6813358119
  1.3616405514
 -1.3988888872
 -1.1303948354
 -0.7983785860
  0.7534387161
  0.6799857692
 -0.5292013529
  0.4509020252
 -0.2581613032
  0.3114454330
 -0.3508081123
  0.0
```

The coefficients are :

```
-20.0
 19.0
 190.0
-168.0
-625.0
 419.0
 905.0
-424.0
-596.0
 191.0
 179.0
-350.0
-234.0
 343.0
-512.0
  0.0
```

A1.2 : The Quadratic Matrix Equation

$$X^2 + P X + Q = 0$$

N = 6

$$P = \begin{bmatrix} -2 & -1 & 0 & 0 & -3 & 0 \\ 0 & -1 & -1 & 2 & 0 & 1 \\ 0 & 2 & 0 & 1 & 2 & 0 \\ -1 & 0 & 0 & 1 & 0 & -2 \\ 1 & 0 & 4 & 0 & 1 & -1 \\ 0 & 2 & 0 & 0 & 4 & 2 \end{bmatrix}$$

$$Q = \begin{bmatrix} -3 & -1 & 0 & 0 & -3 & 0 \\ 0 & -6 & -2 & 4 & 0 & 2 \\ 0 & -6 & -18 & 8 & 8 & 2 \\ 3 & 1 & 0 & -20 & 3 & -10 \\ -3 & -1 & -16 & 4 & -25 & -6 \\ 0 & -6 & -2 & 4 & -20 & -50 \end{bmatrix}$$

A dominant solution is,

$$X = \begin{bmatrix} 3 & 1 & 0 & 0 & 3 & 0 \\ 0 & 3 & 1 & -2 & 0 & -1 \\ 0 & 0 & 4 & -1 & -2 & 0 \\ 0 & 0 & 0 & 4 & 0 & 2 \\ 0 & 0 & 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 6 \end{bmatrix}$$

where the eigenvalues are such that,

$$\lambda(X) = \begin{bmatrix} 3 \\ 3 \\ 4 \\ 4 \\ 5 \\ 6 \end{bmatrix} \quad , \quad \lambda(P+X) = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \\ 1 \\ 2 \end{bmatrix}$$

All methods converge to this dominant solution.

N = 8

$$P = \begin{pmatrix} -3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -3 \end{pmatrix}$$

$$P = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

All methods converge to the following dominant solution

$X = \text{diag}\{1,1,1,1,1,1,1,1\} = I_g$

and $\lambda(X) > \lambda(P+X)$

N = 11

$$P = -\begin{pmatrix}
16 & 12 & 1 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
13 & 16 & 12 & 1 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\
12 & 13 & 15 & 11 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
12 & 11 & 12 & 14 & 10 & 1 & 1 & 0 & 0 & 0 & 0 \\
14 & 11 & 10 & 11 & 13 & 9 & 1 & 1 & 0 & 0 & 0 \\
11 & 16 & 10 & 9 & 10 & 12 & 8 & 1 & 1 & 0 & 0 \\
11 & 10 & 9 & 9 & 8 & 9 & 11 & 7 & 1 & 1 & 0 \\
11 & 10 & 9 & 8 & 8 & 7 & 8 & 10 & 6 & 1 & 1 \\
11 & 10 & 9 & 8 & 7 & 7 & 6 & 7 & 9 & 5 & 1 \\
11 & 10 & 9 & 8 & 7 & 6 & 6 & 5 & 6 & 8 & 3 \\
11 & 10 & 9 & 8 & 7 & 6 & 5 & 3 & 4 & 4 & 6
\end{pmatrix}$$

$$Q = \begin{pmatrix}
132 & 120 & 63 & 40 & 28 & 18 & 0 & 0 & 0 & 0 & 0 \\
121 & 110 & 81 & 24 & 0 & -6 & -10 & 0 & 0 & 0 & 0 \\
165 & 150 & 126 & 88 & 35 & 12 & 5 & 0 & 0 & 0 & 0 \\
176 & 160 & 135 & 112 & 77 & 30 & 10 & 4 & 0 & 0 & 0 \\
209 & 190 & 144 & 120 & 98 & 66 & 25 & 8 & 3 & 0 & 0 \\
242 & 220 & 198 & 128 & 105 & 84 & 55 & 20 & 6 & 2 & 0 \\
176 & 160 & 144 & 128 & 112 & 90 & 70 & 44 & 15 & 4 & 1 \\
176 & 160 & 144 & 128 & 112 & 96 & 75 & 56 & 33 & 10 & 2 \\
165 & 150 & 135 & 120 & 105 & 90 & 75 & 56 & 39 & 20 & 4 \\
143 & 130 & 117 & 104 & 91 & 78 & 65 & 52 & 36 & 22 & 8 \\
99 & 90 & 81 & 72 & 63 & 54 & 45 & 36 & 27 & 16 & 7
\end{pmatrix}$$

All methods converged to the dominant solution,

$$X = \begin{pmatrix}
11 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 10 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 10 & 9 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 10 & 9 & 8 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 10 & 9 & 8 & 7 & 6 & 0 & 0 & 0 & 0 & 0 \\
11 & 10 & 9 & 8 & 7 & 6 & 5 & 0 & 0 & 0 & 0 \\
11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 0 & 0 & 0 \\
11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 0 & 0 \\
11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 0 \\
11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1
\end{pmatrix}$$

where $\lambda(X) > \lambda(P+X)$

A1.3 : The Matrix Square Root Problem

$$X^2 - A = 0$$

N = 8

$$A = \begin{bmatrix} 22 & 21 & 9 & 9 & 21 & 21 & 6 & 6 \\ 21 & 22 & 9 & 9 & 21 & 21 & 6 & 6 \\ 7.5 & 7.5 & 13 & 9 & 7.5 & 7.5 & 9 & 9 \\ 7.5 & 7.5 & 9 & 13 & 7.5 & 7.5 & 9 & 9 \\ 21 & 21 & 6 & 6 & 22 & 21 & 9 & 9 \\ 21 & 21 & 6 & 6 & 21 & 22 & 9 & 9 \\ 7.5 & 7.5 & 9 & 9 & 7.5 & 7.5 & 13 & 9 \\ 7.5 & 7.5 & 9 & 9 & 7.5 & 7.5 & 13 & 9 \end{bmatrix}$$

A square root is,

$$X = \begin{bmatrix} 3 & 2 & 1 & 1 & 2 & 2 & 0 & 0 \\ 2 & 3 & 1 & 1 & 2 & 2 & 0 & 0 \\ 0.5 & 0.5 & 3 & 1 & 0.5 & 0.5 & 1 & 1 \\ 0.5 & 0.5 & 1 & 3 & 0.5 & 0.5 & 1 & 1 \\ 2 & 2 & 0 & 0 & 3 & 2 & 1 & 1 \\ 2 & 2 & 0 & 0 & 2 & 3 & 1 & 1 \\ 0.5 & 0.5 & 1 & 1 & 0.5 & 0.5 & 3 & 1 \\ 0.5 & 0.5 & 1 & 1 & 0.5 & 0.5 & 1 & 3 \end{bmatrix}$$

All methods converge to this solution.

N = 12

$$A = \begin{bmatrix}
3 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0 & 3 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.5 & 0 & 3 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.5 & 0 & 3 & 0 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 3 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 3
\end{bmatrix}$$

A square root is,

$$X = \begin{bmatrix}
1.738 & 0 & 1.434 & 0 & -0.556(-2) & 0 \\
0 & 1.433 & 0 & 0.230 & 0 & 0 \\
-0.143 & 0 & 1.744 & 0 & 0.143 & 0 \\
0 & -0.230 & 0 & 0.744 & 0 & 0 \\
-0.586(-2) & 0 & -0.143 & 0 & 1.744 & 0 \\
0 & -0.364(-1) & 0 & -0.339 & 0 & 0.707 \\
-0.478(-3) & 0 & -0.581(-2) & 0 & -0.143 & 0 \\
0 & 0.146(-1) & 0 & -0.803(-1) & 0 & 0.354 \\
-0.503(-4) & 0 & -0.476(-3) & 0 & -0.593(-2) & 0 \\
0 & -0.810(-2) & 0 & -0.389(-1) & 0 & 0.884(-1) \\
0.782(-5) & 0 & 0.659(-4) & 0 & 0.625(-3) & 0 \\
0 & 0.121(-2) & 0 & 0.568(-2) & 0 & 0.117(-1)
\end{bmatrix}$$

$$\begin{bmatrix}
0.478(-3) & 0 & -0.503(-4) & 0 & 0.782(-5) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
-0.588(-2) & 0 & 0.486(-3) & 0 & -0.660(-4) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0.143 & 0 & -0.593(-2) & 0 & 0.625(-3) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
-1.744 & 0 & 0.144 & 0 & -0.725(-2) & 0 \\
0 & 0.707 & 0 & 0 & 0 & 0 \\
-0.144 & 0 & 1.731 & 0 & 0.159 & 0 \\
0 & -0.354 & 0 & 0.707 & 0 & 0 \\
0.725(-2) & 0 & 0.159 & 0 & 1.405 & 0 \\
0 & 0.297(-1) & 0 & 0.205 & 0 & 1.732
\end{bmatrix}$$

All methods converge to this solution

N = 16

$$A = \begin{bmatrix}
127 & 50 & 32 & 28 & 6 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
50 & 159 & 78 & 38 & 30 & 7 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
32 & 78 & 165 & 80 & 39 & 30 & 7 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
28 & 38 & 80 & 166 & 80 & 39 & 30 & 7 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 7 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 7 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 7 & 2 & 1 & 0 & 0 & 0 & 0 \\
1 & 2 & 7 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 7 & 2 & 1 & 0 & 0 & 0 \\
0 & 1 & 2 & 7 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 7 & 2 & 1 & 0 & 0 \\
0 & 0 & 1 & 2 & 7 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 7 & 2 & 1 & 0 \\
0 & 0 & 0 & 1 & 2 & 7 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 7 & 2 & 1 \\
0 & 0 & 0 & 0 & 1 & 2 & 7 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 7 & 2 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 7 & 30 & 39 & 80 & 166 & 80 & 39 & 30 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 7 & 30 & 39 & 80 & 166 & 80 & 38 & 28 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 7 & 30 & 39 & 80 & 165 & 78 & 32 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 7 & 30 & 39 & 80 & 159 & 50 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 6 & 28 & 32 & 50 & 127
\end{bmatrix}$$

A square root is,

$$X = \begin{bmatrix}
11 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 3 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 12 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 11
\end{bmatrix}$$

All methods converge to this solution

A1.4 : The Algebraic Riccati Equation

$$A^T X + X A - X G X \Big|_H^K = 0$$

where

$$G = G^T = C C^T \geq 0, \quad H = H^T > 0$$

N = 4

$$A = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & -0.5 & 1 & 1 \\ 1 & 0 & -1.5 & 1 \\ -2 & -2 & -3 & -2 \end{bmatrix}, \quad H = \begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix}$$

$$G^T = C C^T = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 9 & 6 & 3 & 0 \\ 6 & 4 & 2 & 0 \\ 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Two solutions are,

$$X_1 = I_4$$

$$X_2 = \begin{bmatrix} -2.120 & 2.200 & 1.662 & 0.882 \\ -3.939 & 4.452 & 5.216 & 0.934 \\ -0.688 & 0.320 & 1.513 & 0.209 \\ 1.812 & -1.918 & -2.903 & 0.634 \end{bmatrix}$$

$$\lambda(X_2) = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}, \quad \lambda(X_1) = \begin{bmatrix} 0.4784 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

The eigenvector method converges to X_1
The Newton iterations method converges to X_2
The Schur vectors method converges to X_2
The new Sparse approach converges to X_1

N = 8

$$A = \begin{bmatrix} -0.5 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -2 & -1.5 & 1 & 1 & 1 & 1 \\ 0 & -1 & 0 & -1 & -1.5 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -2 & -2 & 1 & 1 \\ 0 & -1 & 0 & -1 & 0 & -1 & -2 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -2 & -1.5 \end{bmatrix}$$

$$H = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 \end{bmatrix}$$

$$C^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Two solutions are,

$$X_1 = I_8$$

$$X_2 = \begin{bmatrix} -3.53 & 2.645 & -2.277 & -1.284 & 1.639 & 1.299 & 0.331 & -0.101 \\ -12.885 & 3.364 & 4.727 & -5.305 & 5.250 & -3.862 & 0.344 & -3.103 \\ -11.607 & 2.499 & 4.864 & -4.661 & 4.687 & -3.468 & 0.353 & -2.594 \\ -17.183 & 1.553 & 8.012 & -6.585 & 7.182 & -5.202 & 0.274 & -5.010 \\ -12.907 & 1.575 & 5.582 & -5.567 & 6.349 & -3.894 & 0.253 & -3.541 \\ -14.749 & 0.178 & 8.111 & -6.880 & 6.295 & -3.501 & 0.102 & -4.930 \\ -3.599 & 0.164 & 4.665 & -3.992 & 3.663 & -2.622 & 1.066 & -2.841 \\ -6.546 & -1.344 & 5.120 & -3.508 & 2.955 & -2.043 & -0.120 & -1.964 \end{bmatrix}$$

The eigenvector method converges to X_1
The Newton iterations method converges to X_2
The Schur vectors method converges to X_2
The new Sparse approach converges to X_1

N = 12

$A = (a_{ij})$, $H = (h_{ij})$, $C = (c_i)$

where

$$a_{ij} = \begin{cases} (-1)^{i+1} \left(\frac{i-1}{2}\right) & , \text{ when } i = j \\ 1 & , \text{ when } j > i \\ 0 & , \text{ otherwise} \end{cases}$$

$$h_{ij} = \begin{cases} 1 & , \text{ when } i = j \\ 0 & , \text{ otherwise} \end{cases}$$

$$c_i = 1 \quad , \text{ for } i = 1, \ldots\ldots, n$$

Two solutions are,

$$X = I_{12}$$

$$X = \left[\begin{array}{c|c} e & \underline{f}^T \\ \hline \underline{d} & I_{11} \end{array}\right]$$

where

$e = 0.251$

$\underline{f}^T = \{ 1.071, 0.892, 0.712, 0.532, 0.352, 0.176, 0, 0, 0, 0, 0 \}$

$\underline{d} = (d_i)$, $d_i = 0$, $i = 1, \ldots\ldots, 11$

The eigenvector method converges to X_1
The Newton iterations method converges to X_2
The Schur vectors method converges to X_2
The new Sparse approach converges to X_1

APPENDIX 2

The subroutines in this Appendix relate to the Elimination method and the new methods for the solutions of the quadratic matrix equations. The calling routines and the subroutines called are given below along with their corresponding Appendix references.

subroutine	QMESOSMIN	A2.1
calls	QMEFUN	A2.2
calls	QMEHES	A2.3
calls	QMEMON	A2.4
subroutine	SARENS	A2.5
calls	SPARYSYS	A2.6
subroutine	NEWMIN	A2.7
calls	CONDEST	A2.8
calls	LINESRCH	A2.9
subroutine	INITIALCPX	A2.10
subroutine	ELIMINMETH	A2.11
subroutine	FUNNORM	A2.12
subroutine	CHARPOLY	A2.13
calls	BLOFROB	A2.14
calls	TRANSFORM	A2.15
calls	POLYDET	A2.16
calls	POLYMULT	A2.17
calls	POLYADD	A2.18
subroutine	EIGENVALUES	A2.19
subroutine	SQROOT	A2.20
subroutine	COROOT	A2.21
subroutine	CPXSOSMIN	A2.22
calls	CPXFUN	A2.23
calls	CPXHES	A2.24
calls	CPXMON	A2.25

The subroutines may be incorporated into programs in the following way :

i. Solving the quadratic matrix equation by minimising the n^2 constituent non-linear equations,

```
CALL    QMESOSMIN
```

ii. Solving the algebraic riccati equation by using the new sparse approach of Section 6.2,

```
CALL    SARENS
```

iii. Solving the quadratic matrix equation by using the new global approach of Section 6.3,

```
CALL    NEWMIN
```

iv. Solving the quadratic matrix equation and the matrix square root problem by using the Elimination method iteratively,

```
CALL    INITIALCPX

CALL    ELIMINMETH  }
CALL    FUNNORM     }   repeat until convergence
CALL    CHARPOLY    }
```

v. Solving the matrix square root problem by using the Elimination method 2,

```
CALL    EIGENVALUES
CALL    SQROOT
CALL    COROOT
CALL    ELIMINMETH
```

vi. Solving the matrix square root problem by using the Elimination method 3,

```
CALL    INITIALCPX
CALL    CPXSOSMIN
CALL    ELIMINMETH
CALL    FUNNORM
```

```
C
      SUBROUTINE QMESOSMIN(X,P,Q,N,NA,TOL,ITERMAX,ITERNO,FNORM,NF,IFAIL)
C
C     Description
C     -----------
C
C              This subroutine solves a system of n**2 non-linear
C     equations in n**2 unknown variables.
C     The function relates the known variables p(i,j) and q(i,j)
C     to the unknown variables x(i,j).
C     The matrices P, Q and X are related by the equation :
C                       X**2 + PX + Q = 0
C     The method is a comprehensive modified Gauss-Newton algorithm
C     for finding an unconstrained minimum of a sum of squares function
C     At each iteration a line search is used to ensure that convergence
C     is global (under certain conditions on the Jacobian matrix) and
C     the choice of update, a Gauss-Newton or a full Newton step, is
C     dependant on the choice of update depends on the reduction in
C     the sum of squares obtained during the last iteration.
C
C     Input Parameters
C     ----------------
C
C
C     P       - Coefficient matrix P
C     Q       - Coefficient matrix Q
C     X       - Initial estimate to the solution of the problem
C     N       - The size of the matrices P, Q, X
C     NA      - Row dimension of matrices P, Q, X in calling program
C     TOL     - Convergence criteria on the norm of the function
C               vector
C               If || f(i) || <= TOL  then method is deemed to have
C                                     converged.
C     ITERMAX - The maximum number of iterations to be executed by
C               the routine
C
C     Output Parameters
C     -----------------
C
C
C     ITERNO  - The iteration step at which the routine returns to
C               main program
C     X       - The computed estimates to the solution matrix
C     FNORM   - The Euclidean norm of the function matrix at solution
C     NF      - Specifies the number of function evaluations performed
C     IFAIL   - On output :
C               If IFAIL = 1 the method converged to within the
C                            required tolerance in ITERNO iterations.
C               If IFAIL = 2 the method has not converged to within the
C                            required tolerance in ITERMAX iterations.
C               If IFAIL = 3 the method is diverging.
C               If IFAIL = 4 the Jacobian matrix is singular at
C                            iteration ITERNO
C
```

```
C
C     Subroutine called :
C     ------------------
C
C     E04HEF - NAG routine that minimises a sum of squares scalar
C              function.
C              This subroutine requires the following parameters :
C     NN       - The dimension of the problem.
C     QMEFUN   - Subroutine that calculates the vector of values f(x)
C                and the Jacobian matrix at the current point X.
C     QMEHESS - Subroutine that calculates the Hessiam matrix at the
C                current point X.
C     QMEMON   - Subroutine that moniters the minimisation process.
C     IPRINT   - Specifies the frequency with which QMEMON is to be
C                called. QMEMON is called every IPRINT iterations.
C     ETA      - Specifies the accuracy of the line search at each
C                iteration. ETA lies between 0 and 1. The line search
C                is more accurate for small values of ETA, at a cost of
C                a greater number of calls to QMEFUN.
C     STEPMX   - An estimate of the Euclidean norm between the solution
C                and the starting point. For a general purpose algorithm
C                a large value is suggested.
C     Y        - The vector of dimension NN, the current estimate to the
C                solution.
C     FVEC     - Residual vector of dimension NN.
C     FJAC     - Jacobian at the final point.
C     LJ       - First dimension of FJAC as declared in calling program
C     S        - Contains the singular values of FJAC at the final point
C     V        - The matrix associated with the SVD, J = USV^T  of the
C                Jacobian at the final point.
C     LV       - First dimension of V as declared in calling program.
C     IW       - Integer array of dimension at least 1,used as workspace
C     LIW      - Actual length if IW.
C     W        - Real array used as workspace.
C     LW       - Actual length if W as declared in calling program.
C                LW > 8*NN + 3*NN*NN
C
      DOUBLE PRECISION X(NA,N),P(NA,N),Q(NA,N)
      DOUBLE PRECISION S(50),V(50,50),W(200),Y(50),FJAC(50,50),FVEC(50)
      DOUBLE PRECISION FNORM,ETA,TOL,STEPMX
      INTEGER IFAIL,IPRINT,LIW,LJ,LV,LW,ITERMAX,NF,ITERNO,N,NA,NN,I,J
      INTEGER IW(99)
      EXTERNAL QMEFUN,QMEHES,QMEMON
      COMMON /BLK1/P,Q
      COMMON /BLK2/N
C
```

```
      NN = N*N
      IPRINT = 1
      ETA = 0.9
      STEPMX = 100000.0
      DO 1 I=1,N
          DO 2 J=1,N
 2            Y((I-1)*N+J) = X(I,J)
 1    CONTINUE
      LJ=50
      LV=50
      LIW=99
      LW=200
      IFAIL=1
      CALL E04HEF(NN,NN,QMEFUN,QMEHES,QMEMON,IPRINT,ITERMAX,ETA,TOL,
     1STEPMX,Y,FNORM,FVEC,FJAC,LJ,S,V,LV,ITERNO,NF,IW,LIW,W,LW,IFAIL)
C
      RETURN
      END
C
```

```
      SUBROUTINE QMEFUN(IFLAG,MM,NN,XC,FVECC,FJACC,LJC,IW,LIW,W,LW)
C
C     Description
C     -----------
C
C         This subroutine computes the function vector and the
C     Jacobian matrix for the problem. The calling routine is the
C     NAG module E04HEF which itself is called from QMESOSMIN.
C     For details of input and output parameters, see QMESOSMIN.
C
      DOUBLE PRECISION FJACC(LJC,NN),FVECC(NN),W(LW),XC(NN)
      DOUBLE PRECISION P(NA,N),Q(NA,N),SUM1,SUM2
      INTEGER IFLAG,LIW,LJC,LW,MM,NN,N,S1,S2,S3,S4,S5,S6
      INTEGER IW(LIW)
      COMMON /BLK1/P,Q
      COMMON /BLK2/N,NA
C
C     Compute the function vector
C
      DO 1 I=1,N
          S1=(I-1)*N
          DO 2 J=1,N
              SUM1=0.0D0
              DO 3 K=1,N
                  DO 4 K=1,N
 4                    FJACC(I,J) = 0.0D0
 3                SUM1 = SUM1 + (XC(S1+K) + P(I,K))*XC((K-1)*N+J)
 2            FVECC(S1+J) = SUM1 + Q(I,J)
 1        CONTINUE
C
C     Compute the Jacobian matrix
C
      DO 5 I=1,N
          S1 = (I-1)*N
          DO 6 J=1,N
              S2 = (J-1)*N
              SUM1 = XC(S2+I)
              SUM2 = XC(S1+J) + P(I,J)
              DO 7 K=1,N
                  S3 = S1 + K
                  S4 = S2 + K
                  S5 = (K-1)*N
                  S6 = S5 + I
                  S5 = S5 + J
                  FJACC(S3,S4) = SUM1
 7                FJACC(S5,S6) = FJACC(S5,S6) + SUM2
 6            CONTINUE
 5        CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE QMEHES(IFLAG,MM,NN,FVECC,XC,B,LB,IW,LIW,W,LW)
C
C     Description
C     -----------
C
C         This subroutine computes the Hessian matrix of the function
C     in QMESOSMIN. The calling routine is the NAG module E04HEF
C     itself is called from QMESOSMIN.
C     For details of input and output parameters, see QMESOSMIN.
C
      DOUBLE PRECISION B(LB),FVECC(N),W(LW),XC(N),SUM,Z(500,500)
      INTEGER IFLAG,LB,LW,LIW,MM,NN,S1,N
      INTEGER IW(LIW)
      COMMON /BLK2/N,NA
C
      DO 1 I=1,N
          S1=(I-1)*N
          DO 2 J=1,N
              SUM = FVECC(S1+J)
              DO 3 K=1,N
 3                Z((K-1)*N+J,S1+K) = SUM
 2            CONTINUE
 1        CONTINUE
      DO 4 I=1,NN
          S1 = I*(I-1)/2
          DO 5 J=1,I
 5             B(S1+J) = Z(I,J) + Z(J,I)
 4        CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE QMEMON(NN,NN,XC,FVECC,FJACC,LJC,S,IGRADE,ITERNO,NF,IW,
     1LIW,W,LW)
C
C     Description
C     -----------
C
C         This subroutine moniters the minimisation process.
C     At each iteration it prints the iteration number,the number of
C     function evaluations,the norm of the residual and the current
C     estimate to the solution.
C     For details of input/output parameters refer to QMESOSMIN.
C
C     Functions called
C     ----------------
C
C     F01DEF - This functions computes the Euclidean norm of a vector
C
      DOUBLE PRECISION FJACC(LJC,NN),FVECC(NN),S(NN),W(LW),XC(NN),FNORM
      DOUBLE PRECISION X(50,50)
      INTEGER IGRADE,LIW,LJC,LW,NN,NF,ITERNO,MM
      INTEGER IW(LIW)
C
      FNORM=F01DEF(FVECC,FVECC,N)
      WRITE(*,99)ITERNO,NF,FNORM
      WRITE(*,98)
 99   FORMAT(//,14H  ITERATION: ,I2,4X,11H FUN EVALS ,I2,4X,10H     FUN NORM
     1 ,D14.6)
 98   FORMAT(/,30H CURRENT SOLUTION MATRIX IS : ,/)
      DO 1 I=1,N
          DO 2 J=1,N
              X(I,J)=XC((I-1)*N+J)
 2        CONTINUE
 1    CONTINUE
      DO 3 I=1,N
          WRITE(*,97)(X(I,J),J=1,N)
 97   FORMAT(4D10.4)
 3    CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE SARENS(X,A,G,H,N,NA,TOL,EPS,FNORM,ITERNO,ITERMAX,IFAIL)
C
C     Description
C     -----------
C
C        This subroutine computes the solution of the Algebraic Riccati
C     Equation,
C                  A'X + XA - XGX + H = F(X) = 0
C     by using the method of Chapter 6.
C     The computed solution is the symmetric, non-negative definite
C     one, i.e. the stabilising solution.
C     The method reduces the matrix (A' - XG) to upper schur form so
C     that the associated Jacobian matrix is either sparse or upper
C     triangular. A combination of back-substitution and sparse system
C     solver techniques are then used to determine the solution.
C     The algorithm is iterative and stops when either the maximum
C     number of iterations has been reached or when convergence is
C     deemed to have occurred.
C
C     Input Parameters
C     ----------------
C
C     X        - Initial estimate to the solution matrix
C     A,G,H    - Coefficient matrices
C     N        - Dimension of the coefficient and solution matrices
C     NA       - Row dimension of the matrices in the calling program
C     TOL      - Tolerance used to determine when the iterations are to
C                be terminated
C                If  DABS|F(X)| < TOL  then convergence has occurred
C     EPS      - Tolerance used to determine when an element is zero
C                If  DABS|s| < EPS then s is considered to be zero
C     ITERMAX  - Maximum number of iterations to be performed
C
C     Output Parameters
C     -----------------
C
C     X        - The solution matrix
C     FNORM    - Euclidean norm of the function matrix
C     ITERNO   - Iteration number at which the processing terminated
C     IFAIL    - Indicates the result of the processing
C                If IFAIL = 0  solution computed successfully
C                If IFAIL = 4  Solution has not converged in ITERMAX
C                              iterations
C                See SPARSYS for more IFAIL codes
C
C     Subroutines Called
C     ------------------
C
C     UPHESS   - Computes the upper hessenberg form of a matrix
C     UPTRAN   - Computes the transforming matrix for the hessenberg
C                reduction
C     SCHUR    - Computes the upper schur form of a matrix, along with
C                the transforming matrix
C     SPARSYS  - Solves AX = B, where A is a sparse matrix
C
```

```
      DOUBLE PRECISION X(NA,N),A(NA,N),G(NA,N),H(NA,N)
      DOUBLE PRECISION FNORM,TOL,EPS,SUM,SUM1,SUM2
      DOUBLE PRECISION ORT(20),U(20,20),S(20,20),T(20,20),WORK(20,20),
     1FVECT(400),PVECT(400),FVEC(400),FBKUP(400),FT(20,20),PT(20,20),
     2,ST(40,40)
      INTEGER I,J,K,L,ITERMAX,ITERNO,IFAIL,IR,NN,NNA,N,NA,IFLAG
C
      ITERNO = 0
 99   ITERNO = ITERNO + 1
C
C     Compute S = A  - XG
C
      DO 10 I=1,N
          DO 20 J=1,N
              SUM = 0.0D0
              DO 30 K =1,N
 30               SUM = SUM + X(I,K)*G(K,J)
 20           S(I,J) = A(J,I) - SUM
 10   CONTINUE
C
C     Compute the function matrix using
C                 F(X) = SX + XA
C     and determine the Euclidean norm of F
C
      FNORM=0.0D0
      DO 40 I=1,N
          DO 50 J=1,N
              SUM = 0.0D0
              DO 60 K =1,N
 60               SUM = SUM + S(I,K)*X(K,J) + X(I,K)*A(K,J)
          F(I,J) = SUM1 + H(I,J)
 50       FNORM = FNORM + F(I,J)**2
 40   CONTINUE
      FNORM = DSQRT(FNORM)
C
C     Compute the function matrix using
C                 F(X) = SX + XA
C     and determine the Euclidean norm of F
C
      IF (DABS(FNORM) .LT. TOL)
           IFAIL = 0
           GO TO 399
      ENDIF
C
C     Transform S to upper hessenburg form, accumulating the
C     transformations in U
C
      CALL UPHESS(NA,N,S,ORT)
      CALL HESSTRAN(NA,N,S,ORT,U)
C
C     Transform the upper hessenburg form to an upper schur,
C     updating the transformations in U
C
      CALL SCHUR(S,U,N,NA,EPS,JFAIL)
```

```
C     Transform the matrix F to  FT = U^TFU
C     Form the vector FVECT of size N*N, by taking the
C     elements of the matrix FT a row at a time
C
      DO 100 I=1,N
          DO 110 J=1,N
              SUM = 0.0D0
              DO 120 K=1,N
 120              SUM = SUM + F(I,K)*U(K,J)
 110      WORK(I,J) = SUM
 100  CONTINUE

      DO 130 I=1,N
          I1 = (I-1)*N
          DO 140 J=1,N
              SUM = 0.0D0
              DO 150 K=1,N
 150              SUM = SUM + U(K,I)*WORK(K,J)
          FT(I,J) = SUM
 140      FVECT(I1+J) = SUM
 130      CONTINUE
C
C         Process the Jacobian at the current iteration
C
      IR = 0
 199  CONTINUE
      IF ((DABS(S(N-IR,N-IR-1)) .LT. EPS) .OR. (IR .EQ. (N-1))) THEN
C
C     If IR = N-1 or the subdiagonal element is zero, perform
C     this bit of processing.
C
C     Update the function vector
C
          DO 160 J=1,N
              SUM = 0.0D0
              DO 170 I=0,IR-1
 170              SUM = SUM + S(N-IR,N-I)*PVECT(NN-I*N-N+J)
 160      FVECT(NN-IR*N-N+J) = SUM + FVECT(NN-IR*N-N+J)
          IFLAG = 0
          DO 180 I=N,1,-1
C
C     If the previous processing involved the solution of a
C     set of two simultaneous equations, increment I
C
              IF (IFLAG .EQ. 1) THEN
                 IFLAG = 0
                 GO TO 180
              ENDIF
```

```
C     If the sub-diagonal element is zero, then update the right
C     hand side and determine an element of the direction vector
C
            IF (DABS(S(I,I-1)) .LT. EPS) THEN
                SUM = 0.0D0
                DO 190 J=I+1,N
 190                SUM = SUM + S(I-1,J)*PVECT(NN-IR*N-N+J)
                FVECT(NN-IR*N-N+I) = SUM + FVECT(NN-IR*N-N+I)
                PVECT(NN-IR*N-N+I) = FVECT(NN-IR*N+I)/(S(I,I)
     1                              + S(N-IR,N-IR))
            ELSE

C     If the sub-diagonal element is not zero, then a set of
C     two simultaneous equations must be solved. The right hand
C     side (function vector) is updated before this is done.
C
                SUM = 0.0D0
                DO 200 J=I+1,N
                    SUM = SUM + S(I-1,J)*PVECT(NN-IR*N-N+J)
 200                SUM1= SUM1+ S(I-2,J)*PVECT(NN-IR*N-N+J)
                FVECT(NN-IR*N-N+I) = SUM + FVECT(NN-IR*N-N+I)
                FVECT(NN-IR*N-N+I-1) = SUM1+ FVECT(NN-IR*N-N+I-1)
                SUM = S(I,I-1)*FVECT(NN-IR*N-N+I-1)
                SUM1 = S(I-1,I-1)*FVECT(NN-IR*N-N+I)
                SUM2 = S(I,I-1)*S(I-1,I) - S(I,I)*S(I-1,I-1)
                PVECT(NN-IR*N-N+I) = (SUM - SUM1)/SUM2
                SUM = S(I,I-1)*FVECT(NN-IR*N-N+I-1)
                SUM1 = S(I,I-1)*S(I-1,I)*PVECT(NN-IR*N-N+I)
                SUM2 = S(I,I-1)*S(I-1,I-1)
                PVECT(NN-IR*N-N+I) = (SUM - SUM1)/SUM2
                IFLAG = 1
            ENDIF
 180    CONTINUE
C
        IR = IR + 1
        IF (IR .GT. (N-1)) GO TO 299
    ELSE
C
C     If IR < N-1 or the sub-diagonal element is non-zero then
C     this bit of processing is performed.
C
C     Update the right hand side (function vector)
C     Set up the 2N vector FBKUP holding function values
C
        DO 210 I=1,2
            DO 220 J=1,N
                SUM = 0.0D0
                DO 230 K=0,N-1
 230                SUM = SUM + S(N-(IR+1-I),N-K)*PVECT(NN-K*N-N+J)
            FVECT(NN-IR*N-I*N+J) = FVECT(NN-IR*N-I*N+J) - SUM
 220        FBKUP((I-1)*N+J) = FVECT(NN-IR*N-I*N+J)
 210    CONTINUE
```

```
C     Set up the 2N-by-2N sparse matrix
C
          DO 240 I=1,N
              I1 = (I-1)*N
              DO 250 J=1,N
                  ST(I,J) = S(I,J)
                  ST(I,I) = ST(I,I) + S(N-IR-1,N-IR-1)
                  ST(I,I+N) = S(N-IR-1,N-IR)
                  ST(J+N,J) = S(N-IR,N-IR-1)
                  ST(I+N,J+N) = S(I,J)
 250              ST(I+N,I+N) = S(N-IR,N-IR) + ST(N+I,N+I)
 240      CONTINUE
C
          NN = 2*N
          NNA       = 2*NA
          IFAIL = 0
          CALL SPARSYS(ST,FBKUP,NN,NNA,EPS,IFAIL)
          IF (IFAIL .NE. 0) GO TO 399
C
C     Form a 2N portion of the direction vector
C
          DO 260 I=1,2
              I1 = (I-1)*N
              DO 270 J=1,N
 270              PVECT(NN-IR*N-I*N+J) = FBKUP(I1+J)
 260      CONTINUE

          IR = IR + 2
          IF (IR .GT. (N-1)) GO TO 299
      ENDIF
C
      GO TO 199
C
C     Form the N-by-N direction matrix made up of the elements
C     of the direction vector PVECT
C
 299  CONTINUE
      DO 280 I=1,N
          I1 = (I-1)*N
          DO 290 J=1,N
 290      PT(I,J) = PVECT(I1+J)
 280  CONTINUE
```

```
C     Transform the direction matrix and update the current X
C
      DO 300 I=1,N
          DO 310 J=1,N
              SUM = 0.0D0
              DO 330 K=1,N
 330              SUM = SUM + PT(I,K)*U(J,K)
 320      WORK(I,J) = SUM
 310  CONTINUE
      DO 340 I=1,N
          I1 = (I-1)*N
          DO 350 J=1,N
              SUM = 0.0D0
              DO 360 K=1,N
 360              SUM = SUM + U(I,K)*WORK(K,J)
 350      S(I,J) = X(I,J) - SUM
 340  CONTINUE
C
C     Transform X to (X(i,j) + X(j,i))/2 such that it is symmetric
C
      DO 370 I=1,N
          DO 380 J=1,N
 380          X(I,J) = (S(I,J) + S(J,I))/2.0D0
 370  CONTINUE
C
      IF (ITERNO .EQ. ITERMAX)
          IFAIL = 4
          GO TO 399
      ENDIF
C
      GO TO 99
C
 399  CONTINUE
      RETURN
      END
```

```
C
      SUBROUTINE SPARSYS(S,Y,M,MA,EPS,IFAIL)
C
C     Description
C     -----------
C
C         This subroutine solves a sparse linear system using NAG the
C     NAG routines F01BRF and F04AXF.
C     F01BRF is sued to obtaim an LU-decomposition of a permutation of
C     S, PSQ = LU, where P and Q are permutation matrices, L being unit
C     lower triangular and U upper triangular. The routine uses a sparse
C     variant of Gaussian elimination, the pivotal strategy designed to
C     compromise between maintaining sparsity and controlling loss of
C     accuracy through round-off.
C     F04AXF then computes the solution by block forward and backward
C     substitution, using simple forward or backward substitution within
C     each diagonal block.
C
C     Input Parameters
C     ----------------
C
C     S      - The coefficient matrix
C     Y      - The right hand side vector
C     M      - The order of the matrix S and vector Y
C     MA     - Row dimension of S in the calling routine
C     EPS    - Tolerance to determine whether an element is to be treated
C              as zero or not
C
C     Output Parameters
C     -----------------
C
C     Y      - the soultion vector
C     IFAIL  - If IFAIL is not equal to zero on exit, then an error in
C              F01BRF has occurred. Consult the routine decsription in
C              the NAG documentation
C
C     Subroutines Called
C     ------------------
C
C     F01BRF - Decompose a matrix using Gaussian-type elimination.
C              Some of the required input parameters are as follows :
C              NZ   - specifies the number of non-zero elements in A
C              A    - contains the NZ non-zero elements of S
C              ICN  - contains column indices of the non-zero elements
C                     stored in A
C              LICN - length of vector ICN
C              IRN  - contains row indices of the non-zero elements
C                     stored in A
C              LIRN - length of vector IRN
C              U    - controls the choice of pivots
```

```
C               LBLOCK - if .TRUE. then routine F01BRY is called to pre-
C                        order the matrix to block lower triangular form
C                        before the LU decomposition is performed
C               GROW   - if .TRUE. then on exit,W(1) contains an estimate
C                        for the increase in size of the elements.
C               IW     - integer array of dimension at least 8*M
C               ABORT  - vector of logicals that determine whether the
C                        processing is terminated when particular
C                            conditions are met

C     F04AXF - Solve the transformed system bu back-substitution
C              Some of the required input parameters are as follows :
C              IKEEP - integer array, dimension at least 5*M, containing
C                      indexing information about the decomposition
C              IDISP - communicates between F01BRF and F04AXF
C              MTYPE - if = 1 then the problem is AX=Y
C
      DOUBLE PRECISION S(MA,M),Y(M),EPS,IFAIL
      DOUBLE PRECISION A(400),W(40),U,RESID
      INTEGER I,IFAIL,LICN,LIRN,N,NZ
      INTEGER ICN(400),IDISP(10),IKEEP(200),IRN(400),IW(320)
      LOGICAL GROW,BLOCK,ABORT(4)
C
      NZ=0
      DO 1 I=1,M
          DO 2 J=1,M
              IF (DABS(S(I,J)) .GT. EPS) THEN
                  A(NZ) = S(I,J)
                  IRN(NZ) = I
                  ICN(NZ) = J
                  NZ = NZ+1
              ENDIF
 2        CONTINUE
 1    CONTINUE
      LICN = 400
      LIRN = 400
      U = 0.1D0
      BLOCK = .TRUE.
      GROW = .TRUE.
      ABORT(1) = .TRUE.
      ABORT(2) = .TRUE.
      ABORT(3) = .FALSE.
      ABORT(4) = .TRUE.
      IFAIL = 0
      CALL F01BRF(N,NZ,A,LICN,IRN,LIRN,ICN,U,IKEEP,IW,W,BLOCK,GROW,ABORT
     1,IDISP,IFAIL)
      IF (IFAIL.GT.0) RETURN
      MTYPE = 1
      CALL F04AXF(N,A,LICN,ICN,IKEEP,RHS,W,MTYPE,IDISP,RES)
C
      RETURN
      END
C
```

```
      SUBROUTINE NEWMIN(X,P,Q,N,NA,TOL,EPS,FNORM,ITERNO,ITERMAX,IFAIL,
     1JCOND)
C
C     Description
C     -----------
C
C         This subroutine computes the solution of the quadratic matrix
C     equation,
C                      X**2 + PX + Q = F(X) = 0
C     by using the minimisation method of Chapter 6.
C     The method is a sum of squares minimisation technique with a line
C     search. The update at each step depends on the condition of the
C     Jacobian. If it is ill-conditioned then the Steepest Descent
C     direction is used otherwise Gauss-Newton is used. The reason why
C     the full Newton step is not used is that it would severely impact
C     the operations count and CPU time for the method. With the Gauss-
C     Newton step, the direction vector is be determined by transforming
C     the problem from one of solving a linear system of order N*N to
C     the easier problem of solving the Sylvester equation.
C     The starting point may be chosen to be the identity matrix.
C     The condition of the Jacobian is determined in the course of
C     solving the Sylvester equation by calling CONDEST, a subroutine
C     that determines the condition number of an upper trinagular
C     matrix.
C     The line search uses a quadratic interpolation technique that
C     performs a low accuracy minimisation.
C     Convergence is deemed to have occured when the function norm is
C     less than some specified tolerance, and the processing terminates.
C     The processing also terminates when a specified maximum number of
C     iterations have been reached.
C
C     Input Parameters
C     ----------------
C
C     X        - Initial estimate to the solution matrix
C     P, Q     - Coefficient matrices
C     N        - Dimension of the coefficient and solution matrices
C     NA       - Row dimension of the matrices in the calling program
C     TOL      - Tolerance used to determine when the iterations are to
C                be terminated
C                If  DABS|F(X)| < TOL  then convergence has occurred
C     JCOND    - Tolerance used to determine whether the Jacobian is
C                well-conditioned or not.
C                If  condition no. < JCOND  then it is well-conditioned
C     EPS      - Tolerance used to determine when an element is zero
C                If  DABS|s| < EPS then s is considered to be zero
C     ITERMAX - Maximum number of iterations to be performed
```

```
C     Output Parameters
C     -----------------
C
C     X        - The solution matrix
C     FNORM    - Euclidean norm of the function matrix
C     ITERNO   - Iteration number at which the processing terminated
C     IFAIL    - Indicates the result of the processing
C                If IFAIL = 0  solution computed successfully
C                If IFAIL = 3  Solution has not converged in ITERMAX
C                              iterations
C                See LINESRCH for more IFAIL codes
C
C     Subroutines Called
C     ------------------
C
C     UPHESS   - Computes the upper hessenberg form of a matrix
C     UPTRAN   - Computes the transforming matrix for the hessenberg
C                reduction
C     SCHUR    - Computes the upper schur form of a matrix and updates
C                the transforming matrix obtained from UPTRAN
C     CONDEST  - Computes the estimate to the condition number of an
C                upper triangular matrix
C     LINESRCH- Performs a line serach based on quadratic interpolation
C
      DOUBLE PRECISION X(NA,N),P(NA,N),Q(NA,N),TOL,EPS,FNORM,JCOND
      DOUBLE PRECISION      JAC(400,400),DIR(400),F(20,20),FVEC(400),
     1WORK(20,20),U(20,20),V(20,20),ORT(20),XT(20,20),XTS(20,20)
      DOUBLE PRECISION SUM,COND,ALPHA
      INTEGER I,J,K,N,NA,NN,NNA,IFAIL,JFAIL,ITERNO,MAXITER,MAXCAL
C
      NN = N*N
      NNA = NA*NA
      ITERNO = 0
 99       ITERNO = ITERNO + 1
C
C         Compute the (negative) function vector and the function norm.
C         Assign matrices X+P and the transpose of X to WORK and XT
C         respectively.
C
      FNORM=0.0D0
      DO 10 I=1,N
           I1 = (I-1)*N
           DO 20 J=1,N
                SUM=0.0D0
                DO 30 K =1,N
 30                  SUM = SUM + (X(I,K) + P(I,K))*X(K,J)
           F(I,J) = - SUM - Q(I,J)
           XT(I,J) = X(J,I)
           WORK(I,J) = X(I,J) + P(I,J)
           FVEC(I1+J) = F(I,J)
 20        FNORM = FNORM + F(I,J)**2
 10   CONTINUE
      FNORM = DSQRT(FNORM)
C
```

```
C     Test for convergence
C
      IF (DABS(FNORM) .LT. TOL)
          IFAIL = 0
          GO TO 199
      ENDIF
C
C     Transform WORK (= X + P) to upper schur form accumulating the
C     transformations in U
C
      CALL UPHESS(NA,N,WORK,ORT)
      CALL HESSTRAN(NA,N,WORK,ORT,U)
      CALL SCHUR(WORK,U,N,NA,EPS,JFAIL)
C
C     Transform XT to upper schur form, accumulate transformations in V
C
      CALL UPHESS(NA,N,XT,ORT)
      CALL HESSTRAN(NA,N,XT,ORT,V)
      CALL SCHUR(XT,V,N,NA,EPS,JFAIL)

C     Transpose the transformed matrix XT to obtain the lower
C     schur form for X.
C
      DO 30 I=1,NN
          DO 40 J=1,NN
 40           XLS(I,J) = XT(J,I)
 30   CONTINUE
C
C     Form the upper triangular Jacobian matrix
C
      DO 50 I=1,NN
          DO 60 J=1,NN
 60           JAC(I,J) = 0.0D0
 50   CONTINUE
      DO 70 I=1,N
          I1 = (I-1)*N
          DO 80 J=I,N
              J1 = (J-1)*N
              DO 90 K=1,N
                  K1 = (K-1)*N
                  JAC(I1+K,J1+K) = JAC(I1+K,J1+K) + WORK(I,J)
 90               JAC(K1+I,K1+J) = JAC(K1+I,K1+J) + XLS(J,I)
 80       CONTINUE
 70   CONTINUE
C
C     Estimate the condition of the Jacobian
C
      CALL CONDEST(JAC,NN,NNA,COND)
```

```
C     If the Jacobian is well-conditioned, solve the transformed
C     Sylvester equation by back-substitution to obtain the Gauss-
C     Newton direction
C     otherwise, form the full jacobian matrix and compute the
C     Steepest Descent direction.
C
      IF (COND .LT. JCOND) THEN
          CALL BACKSUB(WORK,U,XLS,V,F,N,NA,JFAIL)
          DO 100 I=1,N
              I1 = (I-1)*N
              DO 110 J=1,N
 110              DIR(I1+J) = F(I,J)
 100      CONTINUE
      ELSE
          DO 120 I=1,NN
              DO 130 J=1,NN
 130              JAC(I,J) = 0.0D0
 120      CONTINUE
          DO 140 I=1,N
              I1 = (I-1)*N
              DO 150 J=1,N
                  J1 = (J-1)*N
                  DO 160 K=1,N
                      K1 = (K-1)*N
                      JAC(I1+K,J1+K) = JAC(I1+K,J1+K) + X(I,J) + P(I,J)
 160                  JAC(K1+I,K1+J) = JAC(K1+I,K1+J) + X(J,I)
 150          CONTINUE
 140      CONTINUE
          DO 170 I=1,NN
             SUM = 0.0D0
             DO 180 K=1,NN
                 SUM = SUM + JAC(I,K)*FVEC(K)
             DIR(I) = SUM
 170      CONTINUE
      ENDIF
C
C     Perform a line search
C
      CALL LINESRCH(X,P,Q,DIR,ALPHA,IFAIL)
      IF (IFAIL .NE. 0) GO TO 199
C
C     Update the estimate to the solution X
C
      DO 190 I=1,N
          I1 = (I-1)*N
          DO 200 J=1,N
 200          X(I,J) = X(I,J) + ALPHA*DIR(I1+J)
 190  CONTINUE
```

```
C     If maximum number of iterations performed, set IFAIL and return
C
      IF (ITERNO .EQ. ITERMAX) THEN
          IFAIL = 3
          GO TO 199
      ENDIF
C
      GO TO 99
 199  CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE CONDEST(A,N,NA,COND)
C
C     This subroutine computes an estimate to the condition number
C     of an upper triangular matrix.
C     Ref : Matrix Computations - Golub and Van Loan - p.77.
C     The algorithm requires an operations count of 2.5*N*N.
C
C     Inputs
C     ------
C
C     A  - The upper triangular matrix
C     N  - Order of A
C     NA - Row dimension of A
C
C     Outputs
C     -------
C
C     COND - An estimate to the condition number of A
C
      DOUBLE PRECISION A(NA,N),COND
      DOUBLE PRECISION P(500),W(500),Y(500),YP(500)
      DOUBLE PRECISION YM(500),SP,SM
      INTEGER I,K
C
C     Initialise work and weights vectors
C
      DO 1 I=1,N
          W(I) = 1.0D0/A(I,I)
 1        P(I) = 0.0D0
C
      DO 2 K=N,1,-1
          YP(K) = (1.0D0 - P(K))/A(K,K)
          YM(K) = (-1.0D0 - P(K))/A(K,K)
          SUM1 = 0.0D0
          SUM2 = 0.0D0
          DO 3 I=1,K-1
              SUM1 = SUM1 + W(I)*DABS(P(I) + A(I,K)*YP(K))
 3            SUM2 = SUM2 + W(I)*DABS(P(I) + A(I,K)*YM(K))
          SP = DABS(YP(K)) + SUM1
          SM = DABS(YM(K)) + SUM2
          IF (SP .GE. SM) THEN
              Y(K) = YP(K)
          ELSE
              Y(K) = YM(K)
          ENDIF
          DO 4 I=1,K-1
 4            P(I) = P(I) + A(I,K)*Y(K)
 2    CONTINUE
C
C     Estimate infinite norm of vector Y, as an estimate to the
C     condition number.
C
      COND = 0.0D0
      DO 5 I=1,N
 5        IF (DABS(Y(I)) .GT. COND) COND = DABS(Y(I))
C
      RETURN
      END
C
```

```
      SUBROUTINE LINESRCH(X,P,Q,N,NA,DIR,NN,ALPHA,IFAIL)
C
C     Description
C     -----------
C
C        This subroutine performs a line search by using a quadratic
C     interpolation technique. It uses the NAG routine E04ABF.
C
C     Input Parameters
C     ----------------
C
C     X       - Current estimate to the solution matrix
C     P, Q - Coefficient matrices
C     N       - Dimension of the coefficient and solution matrices
C     NA      - Row dimension of the matrices in the calling program
C     DIR - Current direction vector
C     NN      - Dimension of direction vector
C
C     Output Parameters
C     -----------------
C
C     ALPHA - An estimate to the minimum
C     IFAIL - If IFAIL = 1 parameter is outside expected range
C               If IFAIL = 2 MAXCAL has been exceeded
C
C     Subroutines Called
C     ------------------
C
C     FUNCT - Calculates the value of the function at any point
C
      DOUBLE PRECISION X(NA,N),P(NA,N),Q(NA,N),DIR(NN),E1,E2,A,B,FALPHA
      INTEGER N,NA,NN,IFAIL,MAXCAL
      COMMON /BLK1/P,Q,X,DMAT,/BLK2/N
      EXTERNAL FUNCT
C
      E1 = 0.0D0
      E2 = 0.0D0
      A = 0.0D0
      B = 1.0D0
      MAXCAL = 99
      CALL E04ABF(FUNCT,E1,E2,A,B,MAXCAL,ALPHA,FALPHA,IFAIL)
      RETURN
      END
C
      SUBROUTINE FUNCT(ALPHA,FALPHA)
      DOUBLE PRECISION ALPHA,FALPHA,WORK(20,20),DIR(400)
      DOUBLE PRECISION P(20,20),Q(20,20),X(20,20),SUM
      COMMON /BLK1/P,Q,X,DIR,/BLK2/N
C
```

```
      DO 10 I=1,N
          I1 = (I-1)*N
          DO 20 J=1,N
 20           WORK(I,J) = X(I,J) + ALPHA*DIR(I1+J)
 10   CONTINUE
      FALPHA = 0.0D0
      DO 30 I=1,N
          DO 40 J=1,N
              SUM = 0.0D0
              DO 50 K=1,N
 50               SUM = SUM + (WORK(I,K) + P(I,K))*WORK(K,J)
 40       FALPHA = FALPHA + (SUM + Q(I,J))**2
 30   CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE INITIALCPX(P,Q,CPX,N,NA,EPS)
C
C     Description
C     -----------
C
C       This subroutine computes estimates to the coefficients of the
C     characteristic polynomial of X, by using a set of formulae that
C     give the coefficients in terms of the norms of matrices P and Q.
C     These values are used to initiate the iterative Elimination Method
C
C     Input Parameters
C     ----------------
C
C
C     N       - Dimension of the matrices
C     NA      - Row dimension of array in calling program
C     P, Q    - Square matrices of order N
C     EPS     - Determines whether an element can be regarded as zero
C
C     Output Parameters
C     -----------------
C
C
C     CPX - Array of size n, containing estimates for the c.c.p of X.
C
      DOUBLE PRECISION P(NA,N),Q(NA,N),CPX(N),QNORM,PNORM,EPS,DNUM,DDEN
      INTEGER N,NA
C
C     Compute the norm of matrices P and Q
C
      QNORM=0.0D0
      DO 1 I=1,N
          DO 2 J=1,N
              PNORM = PNORM + P(I,J)*P(I,J)
 2        QNORM = QNORM + Q(I,J)*Q(I,J)
 1    CONTINUE
      PNORM = DSQRT(PNORM)
      QNORM = DSQRT(QNORM)
C
C     Compute the estimates, using equations (4.38)
C
      CPX(1) = (DSQRT(PNORM**2 + 4.0D0*QNORM) - PNORM)/2.0D0
      IF (DABS(CPX(1) - P(N,N)*PNORM) .LT. EPS) THEN
          CPX(1) = -CPX(1)
      ENDIF
      DDEN =  CPX(1) - P(N,N)*PNORM
      CPX(N) = -(QNORM*Q(N,1))/DDEN
      DO 5 I=0,N-3
          DNUM = P(N,I+1)*PNORM + Q(N,I+2)*QNORM
 5        CPX(N-I-1) = (CPX(N-I) - DNUM)/DDEN
C
      RETURN
      END
C
```

```
      SUBROUTINE ELIMINMETH(X,P,Q,CPX,N,NA,EPS,IFAIL)
C
C     Description
C     -----------
C
C         This subroutine computes the matrix X by
C     solving the following matrix equation
C                     RX = - S
C     where the R and S are calculated from the Elimination method.
C
C     Input Parameters
C     ----------------
C
C
C     P, Q - The coefficient matrices (unchanged on exit from
C            routine)
C     N    - Size of the matrices P, Q, X
C     NA   - Row dimension of the matrices P, Q, X in the calling
C            routine
C     CPX  - Vector of length N containing the coefficients of
C            the characteristic polynomial of X
C     EPS  - If  | X(i,j)  | <= EPS  then  X(i,j) is considered zero
C
C     Output Parameters
C     -----------------
C
C
C     X     - an N-by-N matrix array containing the computed solution
C     IFAIL - on exit if IFAIL = 0  then the matrix R is singular
C
C     Subroutine called
C     -----------------
C
C
C     LUSOLVE - solves a system of N linear equations
C
      DOUBLE PRECISION X(NA,N),P(NA,N),Q(NA,N),CPX(N)
      DOUBLE PRECISION R1(50,50),R2(50,50),APREV(50,50)
      DOUBLE PRECISION A21(50,50),A22(50,50),EPS,SUM
      INTEGER N,NA,ITER,IPTR,I,J,K
C
C     Initialise the matrices
C
      DO 1 I=1,N
          DO 2 J=1,N
              R1(i,j) = 0.0d0
              R2(i,j) = 0.0d0
              A21(I,J) = 0.0D0
 2            A22(I,J) = 0.0D0
 1        A22(I,I) = 1.0D0
```

```
C     Begin the recursive procedure
C
      DO 99 K=0,N-1
          DO 3 I=1,N
              DO 4 J=1,N
 4                APREV(I,J) = A22(I,J)
 3        CONTINUE
          DO 5 I=1,N
              DO 6 J=1,N
                  SUM = 0.0D0
                  DO 7 K=1,N
 7                    SUM = SUM + A21(I,K)*Q(K,J) + A22(I,K)*P(K,J)
 6            X(I,J) = -SUM
 5        CONTINUE
          DO 8 I=1,N
              DO 9 J=1,N
                  A21(I,J) = APREV(I,J)
 9                A22(I,J) = X(I,J)
 8        CONTINUE
          DO 10 I=1,N
              DO 11 J=1,N
                  R2(I,J) = R2(I,J) + CPX(N-K-1)*A21(I,J)
 11               R1(I,J) = R1(I,J) + CPX(N-K-1)*A22(I,J)
 10       CONTINUE
 99   CONTINUE
C
C     End of recursive procedure
C
      DO 12 I=1,N
          DO 13 J=1,N
              SUM = 0.0D0
              DO 14 K=1,N
 14               SUM = SUM + R2(I,K)*Q(K,J)
 13           X(I,J) = CPX(N) - SUM
 12   CONTINUE
C
C     R = R1, S = X
C     Solve the linear system RX = - S, for X
C
      CALL LUSOLVE(R1,X,N,NA,EPS,IFAIL)
C
      RETURN
      END
C
```

```
      SUBROUTINE FUNNORM(F,X,P,Q,FNORM,N,NA)
C
C     Description
C     -----------
C
C                  This subroutine computes the Euclidean norm of
C     the function F defined by :
C                    F(X) = X**2 + PX + Q
C
C     Input Parameters
C     ----------------
C
C
C     P      - Matrix of size N by N
C     Q      - Matrix of size N by N
C     X      - Matrix of size N by N
C     N      - Size of the matrices P, Q, X
C     NA     - Row dimension of the matrices P, Q, X in the calling
C              routine
C
C     Output Parameters
C     -----------------
C
C
C     FNORM - The Euclidean norm of the function matrix
C     F     - On exit contains the function matrix
C
      DOUBLE PRECISION F(NA,N),X(NA,N),P(NA,N),Q(NA,N),FNORM,SUM
      INTEGER N,NAI,J,K
C
      FNORM = 0.0D0
      DO 1 I=1,N
          DO 2 J=1,N
              SUM = 0.0D0
              DO 3 K=1,N
                  SUM = SUM + (X(I,K) + P(I,K))*X(K,J)
 3            CONTINUE
              F(I,J) = SUM + Q(I,J)
              FNORM = FNORM + F(I,J)**2
 2        CONTINUE
 1    CONTINUE
      FNORM=DSQRT(FNORM)
C
      RETURN
      END
C
```

```
      SUBROUTINE CHARPOLY(A,CPA,N,NA,EPS)
C
C     Description
C     -----------
C
C                  This subroutine computes the coefficients of
C     the characteristic polynomial of a general square matrix A.
C     This involves the reduction of the matrix to a block frobenius
C     matrix via stable elementary operations,and then using the
C     polynomials associated with the blocks to obtain the
C     coefficients of the characteristic polynomial of the matrix.
C
C     Input Parameters
C     ----------------
C
C
C     A   - A general square matrix
C     N   - The size of the matrix A
C     NA  - The row dimension of the matrix A in the calling routine
C     EPS - If  |k| <= eps  then k is considered as zero
C
C     Output Parameters
C     -----------------
C
C
C     A   - Square matrix containing the block frobenius matrix
C     CPA - An n-vector containing the coefficients of the
C           characteristic polynomial of the matrix A
C
C     Subroutine Called
C     -----------------
C
C
C     BLOFROB   - Determines the block frobenious form of a matrix
C     TRANSFORM - Transforms block frobenious matrix into a
C                 polynomial matrix
C
      DOUBLE PRECISION A(NA,N),CPA(N),EPS
      INTEGER N,NA,KBLOCKS,I
C
      CALL BLOFROB(A,N,NA,EPS)
      CALL TRANSFORM(A,N,CPA,KBLOCKS,NA,EPS)
C
C     KBLOCKS is equal to the number of frobenius blocks on diagonal
C
      IF(KBLOCKS.EQ.1)THEN
         DO 1 I=1,N
            CPA(I)=-1.0D0*A(1,I)
 1       CONTINUE
                        ELSE
      ENDIF
C
      RETURN
      END
C
```

```
      SUBROUTINE BLOFROB(A,N,NA,EPS)
C
C     Description
C     -----------
C
C                  This subroutine computes the block frobenius
C     matrix associated with a matrix A. The block form consists
C     of companion matrices on the diagonal. The blocks in the
C     upper triangle have zero entries everywhere except the
C     elements in the first row of the block. The sub-diagonal
C     blocks have elements k(i) only in the top right hand corners
C     of the blocks.
C
C     Input Parameters
C     ----------------
C
C
C     A   - The matrix to be transformed to frobenius form.
C     N   - The size of the matrix
C     NA  - The row dimension of the matrix in the calling routine
C     EPS - Used to determine whether an element may be considered
C           to be zero
C
C     Output Parameters
C     -----------------
C
C
C     A   - The block frobenius matrix
C     CPA - The coefficients of the characteristic polynomial of A
C
      DOUBLE PRECISION A(NA,N),W(50),S(50),EPS,SUM1
      INTEGER N,NA,IR,I1,J1,I,J,K
C
      J = 1
 200  SUM1 = 0.0D0
      IR = J + 1
      DO 1 I=J+1,N
          IF(DABS(A(I,J)) .GT. SUM1) THEN
              SUM1 = DABS(A(I,J))
              IR = I
          ENDIF
 1    CONTINUE
      IF (IR .EQ. (J+1)) GO TO 400
      DO 2 K=1,N
          SUM1 = A(J+1,K)
          A(J+1,K) = A(IR,K)
 2        A(IR,K) = SUM1
      DO 3 K=1,N
          SUM1 = A(K,J+1)
          A(K,J+1) = A(K,IR)
 3        A(K,IR) = SUM1
 400  CONTINUE
      IF (DABS(A(J+1,J)) .LE. EPS) GO TO 600
```

```
      DO 5 I=J+2,N
          W(I) = A(I,J)/A(J+1,J)
          IF (DABS(A(I,J)) .LE. EPS) THEN
              A(I,J)=0.0D0
              GO TO 500
          ENDIF
          DO 4 K=1,N
 4            A(I,K)=A(I,K)-A(J+1,K)*W(I)
 500      CONTINUE
 5    CONTINUE
C
      DO 6 I=J+2,N
          DO 7 K=1,N
 7            A(K,J+1) = A(K,J+1) + A(K,I)*W(I)
 6    CONTINUE
      DO 8 K=1,N
          IF (DABS(A(J+1,K)) .LT. EPS) GO TO 8
 8        IF (DABS(A(J+1,K)) .LT. (EPS*A(J+1,K))) GO TO 600
      SUM1 = A(J+1,J)
      DO 9 K=1,N
 9        A(J+1,K) = A(J+1,K)/SUM1
      DO 10 K=1,N
 10       A(K,J+1) = A(K,J+1)*SUM1
 600  CONTINUE
      DO 11 I1=1,N
          DO 12 J1=1,N
 12           IF (DABS(A(I1,J1)) .LE. EPS) A(I1,J1) = 0.0D0
 11   CONTINUE
      IF (J .LT. (N-1)) THEN
          J=J+1
          GO TO 200
      ENDIF
      I = N
 700  IF (A(I,I-1) .NE. 1) GO TO 800
      DO 13 J=I,N
          S(J) = A(I,J)
         DO 14 K=1,N
 14          A(K,J) = A(K,J) - A(K,I-1)*S(J)
 13   CONTINUE
      DO 15 J=I,N
          DO 16 K=1,N
 16           A(I-1,K) = A(I-1,K) + A(J,K)*S(J)
 15    CONTINUE
 800  IF (I .EQ. 2) GO TO 900
      I = I - 1
      GO TO 700
 900  CONTINUE
      DO 17 I=1,N
          DO 18 J=1,N
 18           IF (DABS(A(I,J)) .LT. EPS) A(I,J) = 0.0D0
 17   CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE TRANSFORM(A,N,CPA,KBLOCKS,NA,EPS)
C
C     Description
C     -----------
C
C                This subroutine classifies the blocks, their
C     sizes and their positions in the matrix obtained from
C     subroutine BLOFROB thus we effectively obtain an upper
C     hessenburg block matrix with each block representing a
C     polynomial.
C
C     Input Parameters
C     ----------------
C
C
C     A    - N*N matrix obtained from subroutine BLOFROB
C     NA   - Row dimension of A in the calling routine
C     EPS  - Tolerance to determine when an element may be
C            taken as zero
C
C     Output Parameters
C     -----------------
C
C
C     CPA      - The coefficients of the characteristic
C                polynomial of A
C     KBLOCKS  - The number of blocks on the diagonal of
C                the block matrix
C
C     Subroutine Called
C     -----------------
C
C
C     POLYDET - Determines the determinant of a polynomial matrix
C
      DOUBLE PRECISION EPS,CPA(N),POLY(50,50,50),A(NA,N)
      INTEGER*4 POS(50),SIZE(50,50)
      INTEGER N,NA,KBLOCKS,IMP1,IMP2,NN,IJ,KL,INK,I,J
C
      KBLOCKS = 1
      POS(1) = 1
      DO 1 I=2,N
         IMP1 = 0
         IMP2 = 0
         DO 2 J=I-1,N
             IF (DABS(A(I,J)) .LT. EPS) IMP1 = IMP1+1
             IF (DABS(A(I,J)-1.0D0) .LT. EPS) IMP2 = IMP2+1
 2       CONTINUE
         IMP = IMP1 + IMP2
         NN = N - I + 2
         IF (IMP .NE. NN) THEN
             KBLOCKS = KBLOCKS + 1
             POS(KBLOCKS) = I
         ENDIF
 1    CONTINUE
```

```
C     There are KBLOCK frobenius blocks :
C         the r'th beginning at          (pos(r),pos(r))
C              and ending    at          (pos(r+1)-1,pos(r+1)-1)
C
      IF (KBLOCKS .EQ. 1) RETURN
      DO 3 I=1,N
         DO 4 L=1,KBLOCKS
            IF (I .EQ. POS(L)) THEN
                DO 5 IJ=1,KBLOCKS
                    IF (IJ. EQ. KBLOCKS) POS(IJ+1) = N + 1
                    DO 6 KL = POS(IJ),POS(IJ+1)-1
                        IF (IJ .EQ. L) THEN
                            POLY(L,IJ,1) = 1.D0
                            INK = KL-(POS(IJ) - 1)
                            POLY(L,IJ,INK+1)=-A(I,KL)
                            SIZE(L,IJ)=INK+1
                        ELSE
                            INK=KL-(POS(IJ)-1)
                            POLY(L,IJ,INK)=-A(I,KL)
                            SIZE(L,IJ)=INK
                        ENDIF
 6                  CONTINUE
 5              CONTINUE
            ENDIF
 4       CONTINUE
 3    CONTINUE
C
      CALL POLYDET(POLY,SIZE,EPS,CPA,KBLOCKS,N)
C
      RETURN
      END
C
```

```
      SUBROUTINE POLYDET(P,SIZE,EPS,CPA,N,NA,NSIZE)
C
C     Description
C     -----------
C
C               This subroutine computes the determinant of the n*n
C     polynomial matrix P.
C
C     Input Parameters
C     ----------------
C
C
C     P      - The matrix containing the coefficients of the
C              polynomials in the blocks.
C     SIZE   - SIZE(i,j) equals the order of the polynomial in the
C              (i,j)'th position of the block matrix P
C     N      - The number of blocks in the matrix
C     NA     - Row dimension of the array P
C     EPS    - Tolerance to determine when an element is zero
C     NSIZE - The length of the vector CPA
C
C     Output Parameters
C     -----------------
C
C
C     CPA    - an N-vector containing the coefficients of the
C              characteristic polynomial of the matrix
C
C     Subroutines called
C     ------------------
C
C
C     POLYMULT - Determines the product of two polynomials
C     POLYADD  - Determines the sum of two polynomials
C
      DOUBLE PRECISION P(NA,50,50),AV(50),BV(50),D(50,50),CV(50)
      DOUBLE PRECISION CPA(NSIZE),EPS
      INTEGER*4 DS(50),SIZE(NA,N)
      INTEGER N,NA,NSIZE,KIP,JDIM,ICT,ISIZE1,ISIZE2,I,K,L
C
      KIP = 0
      DO 1 I=1,N-1
          ICT = SIZE(I+1,I)
          IF (DABS(P(I+1,I,ICT)) .GT. EPS) THEN
              DO 2 K=I,N
                  DO 3 L=1,SIZE(I+1,K)
 3                    AV(L) = P(I+1,K,L)
                  DO 4 L=1,SIZE(I,I)
 4                    BV(L) = P(I,I,L)
                  CALL POLYMULT(AV,SIZE(I+1,K),BV,SIZE(I,I),CV,IDIM)
                  DO 5 L=1,SIZE(I,K)
 5                    BV(L) = P(I,K,L)
                  CALL POLYADD(BV,SIZE(I,K),CV,IDIM,AV,JDIM)
                  DO 6 L=1,JDIM
 6                    P(I+1,K,L) = AV(L)
                  SIZE(I+1,K) = JDIM
 2            CONTINUE
          ELSE
```

```
                KIP = KIP + 1
                DO 200 L=1,SIZE(I,I)
200                  D(KIP,L) = P(I,I,L)
                DS(KIP) = SIZE(I,I)
            ENDIF
1       CONTINUE
        IF (KIP .EQ. 0) GO TO 99
        IF (KIP .GT. 1) THEN
            DO 41 I=1,KIP
                IF (I .EQ. 1) THEN
                    AV(1) = 1.D0
                    ISIZE1 = 1
                ELSE
                    DO 42 L=1,ISIZE3
42                      AV(L) = CV(L)
                    ISIZE1 = ISIZE3
                ENDIF
                DO 43 L=1,DS(I)
43                  BV(L) = D(I,L)
                ISIZE2 = DS(I)
                CALL POLYMULT(AV,ISIZE1,BV,ISIZE2,CV,ISIZE3)
41          CONTINUE
        ELSE
            DO 44 L=1,DS(1)
44              CV(L) = D(1,L)
            ISIZE3 = DS(1)
        ENDIF
        DO 45 L=1,SIZE(N,N)
45          BV(L) = P(N,N,L)
        ISIZE2=SIZE(N,N)
        CALL POLYMULT(CV,ISIZE3,BV,ISIZE2,AV,ISIZE1)
        DO 46 L=1,ISIZE1
46          P(N,N,L) = AV(L)
        SIZE(N,N) = ISIZE1
99      CONTINUE
C
        DO 98 I=1,NSIZE
98          CPA(I) = P(N,N,I+1)
C
        RETURN
        END
C
```

```
      SUBROUTINE POLYMULT(A,M,B,N,C,MN)
C
C     Description
C     -----------
C
C                  This subroutine computes the product of two
C     polynomials, not necessarily of the same degree.
C
C     Input Parameters
C     ----------------
C
C
C     A  - An M-vector containing the coefficients of the polynomial
C          of degree M
C     B  - An N-vector containing the coefficients of the polynomial
C          of degree N
C
C     Output Parameters
C     -----------------
C
C
C     C  - The mn-vector containing the product of the polynomials
C          A and B
C     MN - Degree of the polynomial C. MN = M + N
C
      DOUBLE PRECISION A(M),B(N),C(50)
      INTEGER M,N,MN,I,J
C
      MN = M + N - 1
      DO 1 I=1,MN
 1        C(I) = 0.0D0
      DO 2 I=1,M
         DO 3 J=1,N
 3          C(I+J-1) = C(I+J-1) + A(I)*B(J)
 2    CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE POLYADD(A,M,B,N,C,IMAX)
C
C     Description
C     -----------
C
C                This subroutine computes the sum of two
C     polynomials not necessarily of the same degree.
C
C     Input Parameters
C     ----------------
C
C
C     A - M-vector containing the coefficients of the
C         polynomial of degree M
C     B - N-vector containing the coefficients of the
C         polynomial of degree M
C
C     Output Parameters
C     -----------------
C
C
C     C    - vector of length IMAX containing the coefficients of the
C            sum of the polynomials A and B.
C     IMAX - Degree of the polynomial C. IMAX = max(M,N)
C
      DOUBLE PRECISION A(M),B(N),C(50)
      INTEGER M,N,IMAX,IMIN
      IMAX=MAX(M,N)
      IMIN=MIN(M,N)
      DO 1 I=1,IMIN
         IF(IMIN .EQ. M)THEN
               C(IMAX-I+1)=A(IMIN-I+1)+B(IMAX-I+1)
         ELSE
               C(IMAX-I+1)=A(IMAX-I+1)+B(IMIN-I+1)
         ENDIF
 1    CONTINUE
      DO 2 I=1,(IMAX-IMIN)
         IF(IMIN .EQ. M)THEN
               C(I)=B(I)
         ELSE
               C(I)=A(I)
         ENDIF
 2    CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE EIGENVALUES(A,ER,EI,N,NA,IFAIL)
C
C     Description
C     -----------
C
C
C         This subroutine uses the NAG routine F02AFF to calculate the
C     eigenvalues of a general matrix by reduction to the hessenberg
C     and schur forms using the QR algorithm.
C
C     Input
C     -----
C
C
C     A  - The square matrix whose eigenvalues are desired
C     N  - The size of the matrix A
C     NA - The row dimension of the matrix A in the calling routine
C
C     Output
C     ------
C
C
C     ER - A n-vector containing the real parts of the eigenvalues of
C     EI - An n-vector containing the imaginary parts of the eigenvalues
C     IFAIL - an error indicator :
C             If IFAIL = 1,  more than 30*n iterations are required to
C                       isolate all the eigenvalues
C
      DOUBLE PRECISION A(NA,N),ER(N),EI(N),A1(50,50)
      INTEGER*4 INTGER(50)
      INTEGER IA,IFAIL,N,NA,I,J
C
C     The NAG routine overwrites the matrix A, so make a copy of
C     so that it remains unchanged on exit
C
      DO 1 I=1,N
          DO 2 J=1,N
 2            A1(I,J) = A(I,J)
 1    CONTINUE
      IA = NA
      IFAIL = 1
      CALL F02AFF(A1,IA,N,ER,EI,INTGER,IFAIL)
C
      RETURN
      END
C
```

```
      SUBROUTINE SQROOT(ER,EI,N,TOL)
C
C     Description
C     -----------
C
C              This subroutine computes the square roots of N
C     complex numbers
C
C     Input Parameters
C     ----------------
C
C     ER  - A vector containing the real parts of the
C           complex numbers
C     EI  - A vector containing the imaginary parts of the
C           complex numbers
C     N   - The size of the vectors ER, EI
C     TOL - If  || ER(i) || <=  then  ER(i) = 0.0
C
C     Output Parameters
C     -----------------
C
C     ER - An N-vector containing the real part of the
C          square root
C     EI - An N-vector containing the imaginary part of the
C          square root
C
      DOUBLE PRECISION ER(N),EI(N),XR(50),XI(50),TOL,SUM1
      INTEGER LIR,LII,N,I
C
      DO 1 I=1,N
          LIR = 1
          IF (ER(I) .LT. 0.0D0) LIR = -1
          LII = 1
          IF (EI(I) .LT. 0.0D0) LII = -1
          IF (DABS(EI(I)) .LE. TOL) THEN
              IF (LIR .EQ. 1) THEN
                  XR(I) = DSQRT(ER(I))
                  XI(I) = 0.0D0
              ELSE
                  XR(I) = 0.0D0
                  XI(I) = DSQRT(LIR*ER(I))
              ENDIF
          ELSE
              IF (DABS(ER(I)) .LE. TOL) THEN
                  XR(I) = DSQRT(LII*EI(I)/2.0D0)
                  XI(I) = LII*SQRT(LII*EI(I)/2.0D0)
              ELSE
                  SUM1  = DSQRT(ER(I)**2+EI(I)**2)
                  SUM1  = (SUM1+ER(I))/2.0D0
                  XR(I) = DSQRT(SUM1)
                  SUM1  = XR(I)**2-ER(I)
                  XI(I) = LII*DSQRT(SUM1)
              ENDIF
          ENDIF
    1 CONTINUE
      DO 2 I=1,N
         ER(I) = -XR(I)
         EI(I) = -XI(I)
    2 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE COROOT(COEFF,ROOTR,ROOTI,N)
C
C     Description
C     -----------
C
C              This subroutine computes the coefficients of a
C     polynomial given its roots.
C
C     Input Parameters
C     ----------------
C
C     ROOTR - A vector containing the real parts of the roots
C             of the polynomial
C     ROOTI - A vector containing the immaginary parts of the
C             roots of the polynomial
C     N     - The size of the vectors ROOTR, ROOTI
C
C     Output Parameters
C     -----------------
C
C     COEFF - An N-vector containing the coefficients of the
C             polynomial
C
      DOUBLE PRECISION ROOTR(N),ROOTI(N),COEFF(N),CI(50)
      DOUBLE PRECISION SUM1,SUM2,SUM3
      INTEGER*4 IPT(50)
      INTEGER INT,N,I,J,K
C
      DO 5 I=1,N
         COEFF(I)=0.0D0
         CI(I)=0.0D0
         K=0
 1       K=K+1
             IF(K.EQ.1)THEN
                  INT=0
             ELSE
                  INT=IPT(K-1)
             ENDIF
             IPT(K)=INT
 2           IPT(K)=IPT(K)+1
                 IF (K .LT. I)GO TO 1
                 SUM1=1.0D0
                 SUM2=0.0D0
                 DO 3 J=1,I
                     SUM3=SUM1
                     SUM1=SUM1*ROOTR(IPT(J))-SUM2*ROOTI(IPT(J))
 3                   SUM2=SUM2*ROOTR(IPT(J))+SUM3*ROOTI(IPT(J))
                 COEFF(I)=COEFF(I)+SUM1
                 CI(I)=CI(I)+SUM2
                 DO 4 L=1,I
                     IF(IPT(I+1-L).LT.(N-(L-1)))THEN
                         K=I+1-L
                         GO TO 2
                     ENDIF
 4               CONTINUE
 5    CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE CPXSOSMIN(CPA,CPX,N,TOL,ITERMAX,ITERNO,FNORM,NF,IFAIL)
C
C     Description
C     -----------
C
C              This subroutine solves a system of N non-linear equation
C     in the N unknowns.
C     The function relates the known variables, CPA(i), the
C     coefficients of the characteristic polynomial of a matrix
C     A to the unknown variables, cpx(i), the coefficients of the
C     characteristic polynomials of a matrix X.
C     the matrices A and X are related by the equation :
C                          X**2 - A = 0
C
C     Input Parameters
C     ----------------
C
C     CPA      - The coefficients of the characteristic polynomial
C                of the matrix A
C     CPX      - Contains the initial estimates to the C.C.P of X
C     N        - The length of the vector CPA
C     TOL      - convergence criteria on the norm of the function vector
C                If ¦¦ F(i) ¦¦ <= tol  then newtons method is deemed to
C                have converged.
C     ITERMAX  - The maximum number of iterations to be executed by the
C                routine
C
C     Output Parameters
C     -----------------
C
C     CPX      - The computed estimates to the coefficients of the
C                characteristic polynomial of X.
C
C     For futher details of output parameters and a decsription of the
C     NAG rotuine E04HEF used here, see Subroutine QMESOSMIN.
C
      DOUBLE PRECISION CPA(N),CPX(N)
      DOUBLE PRECISION S(50),V(50,50),W(200),Y(50),FJAC(50,50),FVEC(50)
      DOUBLE PRECISION FNORM,ETA,TOL,STEPMX
      INTEGER IFAIL,IPRINT,LIW,LJ,LV,LW,ITERMAX,NF,ITERNO,N,NA,NN,I,J
      INTEGER IW(99)
      EXTERNAL CPXFUN,CPXHES,CPXMON
      COMMON /BLK1/CPA
      COMMON /BLK2/N

      IPRINT = 1
      ETA = 0.9
      STEPMX = 100000.0
      LJ=50
      LV=50
      LIW=99
      LW=200
      IFAIL=1
      CALL E04HEF(N,N,CPXFUN,CPXHES,CPXMON,IPRINT,ITERMAX,ETA,TOL,
     1STEPMX,Y,FNORM,FVEC,FJAC,LJ,S,V,LV,ITERNO,NF,IW,LIW,W,LW,IFAIL)
C
      RETURN
      END
C
```

```
      SUBROUTINE CPXFUN(IFLAG,N,N,XC,FVECC,FJACC,LJC,IW,LIW,W,LW)
C
C
C     Description
C     -----------
C       This subroutine computes the function and Jacobian.
C     The calling routine is the NAG library routine E04HEF which
C     itself is called from CPXSOSMIN.
C     For details of input and output parameters consult CPXSOSMIN.
C
      DOUBLE PRECISION FJACC(LJC,N),FVECC(N),W(LW),XC(N),CPA(N),SUM
      INTEGER IFLAG,LIW,LJC,LW,M,N,I,J,K
      INTEGER IW(LIW)
      COMMON /BLK1/CPA
      COMMON /BLK2/N
C
C     Compute the Function vector
C
      DO 1 I=1,N
          SUM=0.0D0
          DO 2 J=1,I
              K=2*I-J
 2            IF (K .LE. N) SUM = SUM + ((-1)**J)*XC(J)*XC(K)
 1        FVECC(I) = CPA(I) - SUM + ((-1)**J)*(XC(I)**2)
      DO 5 I=1,INT(N/2)
 5        FVECC(I) = FVECC(I) + XC(2*I)
C
C     Compute the Jacobian matrix
C
      DO 3 I=1,N
          DO 4 J=1,N
              K = 2*I-J
              IF ((K .GE. 1) .AND. (K .LE. N))
     1            FJACC(I,J)=((-1)**(J+1))*2.0D0*XC(K)
 4        CONTINUE
 3    CONTINUE
      K = INT(N/2)
      DO 6 I=1,K
          FJACC(I,2*I) = 1.0D0
 6    CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE CPXHES(IFLAG,N,N,FVECC,XC,B,LB,IW,LIW,W,LW)
C
C
C     Description
C     -----------
C
C       This subroutine computes the Hessian type term
C     relating to the function in cpxfun. The calling routine is
C     the NAG library routine E04HEF which itself is called
C     from CPXSOSMIN. For details of input and output parameters
C     consult CPXSOSMIN.
C
      DOUBLE PRECISION B(LB),FVECC(N),W(LW),XC(N),SUM
      INTEGER IFLAG,LB,LW,LIW,N,M,I,J,K,L
      INTEGER IW(LIW)

      DO 1 I=1,N
       L=I*(I-1)/2
       DO 2 J=1,I
              K=(I+J)/2
           IF (INT(((I+J)/2)-INT((I+J)/2)).EQ.0)THEN
               SUM=FVECC(K)
           ELSE
               SUM=0.0D0
           ENDIF
              B(L+J)=((-1)**(J-1))*SUM
 2        CONTINUE
 1    CONTINUE
      RETURN
      END
C
```

```
      SUBROUTINE CPXMON(N,N,XC,FVECC,FJACC,LJC,S,IGRADE,ITERNO,NF,IW,
     1LIW,W,LW)
C
C     Description
C     -----------
C
C
C       This subroutine moniters the minimisation process.
C     At each iteration it prints the iteration number, the number of
C     function evaluations, the norm of the residual and the current
C     estimate to the solution.
C     For details of input/output parameters refer to CPXSOSMIN.
C
C     Functions called
C     ----------------
C
C     F01DEF - used to determine the Euclidean norm of vector
C
      DOUBLE PRECISION FJACC(LJC,N),FVECC(N),S(N),W(LW),XC(N),FNORM
      INTEGER IGRADE,LIW,LJC,LW,N,NF,ITERNO,M
      INTEGER IW(LIW)
C
      FNORM=F01DEF(FVECC,FVECC,N)
      WRITE(*,99)ITERNO,NF,FNORM
      WRITE(*,98)
 99   FORMAT(//,14H  ITERATION: ,I2,4X,11H FUN EVALS ,I2,4X,10H FUN NORM
     1 ,D14.6)
 98   FORMAT(/,24H CURRENT ESTIMATES ARE : ,/)
       DO 1 I=1,N
       WRITE(*,97)XC(I)
 97   FORMAT(D10.4)
 1    CONTINUE
C
      RETURN
      END
C
```