Numerical Solution of AtS+SA+Q=0

Hagander, Per

1969

Document Version:
Publisher's PDF, also known as Version of record

NUMERICAL SOLUTION OF $A^TS+SA+Q=0$

PER HAGANDER

REPORT 6920 OCTOBER 1969
LUND INSTITUTE OF TECHNOLOGY
DIVISION OF AUTOMATIC CONTROL

NUMERICAL SOLUTION OF $A^TS + SA + Q = 0$ †

Per Hagander

ABSTRACT

A survey of possible techniques to solve $A^TS + SA + Q = 0$ is presented and the best algorithms are coded and tested on a batch of examples. Two general purpose methods (one of which is supposed to be new) are recommended depending on the order of the matrices and the computer storage available.

† This work has been supported by the Swedish Board for Technical Development under Contract 69-631/U489

TABLE OF CONTENTS

1. INTRODUCTION

In recent time [2,4,6,10,14] great attention is drawn to the numerical treatment of the matrix equation

$$A^TS + SA + Q = 0 \tag{1}$$

solved for S. Q is symmetric and thus also the solution S.

This equation plays a very central role in the linear theory of stability, both in the qualitative criterium (Routh - Hurwite etc)[17], as in the more quantitative ones: Liapunov functions [13] and eigenvalue calculations. It is also valuable for the pole assignment problem and fundamental for evaluating loss functions in optimal control and covariance matrices in filtering and estimation.

The more general equation:

$$A^TS + SA + Q - SBQ_2^{-1}B^TS = 0 \tag{2}$$

essential in the spectral factorization problem [1] and in optimal control [21],has been solved by iteration of (1), [12].

In many of the applications above it is necessary to solve (1) many times. We therefore state that the equation (1) is worth severe numerical interest and almost as important as a usual matrix inversion. It is my intention to give a survey of possible methods and to present algorithms. These algorithms are used on a set of examples, and their accuracy and computing time are compared. All programming is done in Fortran for a CD-3600.

It can be said as a result that it is valuable to have the origin of eq. (1) in mind; in what form the data is obtainable etc. Very important is also the computing facilities available and the order of the system, dictating the memory requirements. Sometimes the calculation of S is made parallel to other routines which gives side results that are useful to the $A^TS + SA + Q = 0$ - calculation and perhaps makes different algorithms most favourable. This is for instance the case when eigenvalues and eigenvectors are calculated.

2. REVIEW OF POSSIBLE TECHNIQUES.

2.1. Simple finite algebraic methods.

2.1.1. Solution of n(n+1)/2 linear equations.

Inspecting the equation:

$$A^T S + SA + Q = 0 \qquad (1)$$

we find that we have $\frac{n(n+1)}{2}$ unknown and therefore it is very natural to rewrite (1) as n(n+1)/2 linear equations and solve these straight forward.

2.1.2. Eigenvalue, Eigenvector methods.

The method above is not necessarily the best one. When we solve the big system of linear equations with the general method, we are applying too strong a tool. These equations have very much "structure" in them, and it might be better to use that, at least to some extent, by transformation techniques.

One way of doing this is transforming A and A^T to diagonal form [10]. Since this is no easy matter and by no means completely solved, it seems to be no promising attack to the problem. However, the information obtained from the eigenvalues might be so valuable for the further analysis that the method must be taken into consideration.

Eigenvectors can also be used to solve eq. (1) and even the more general eq. (2) in the following way [16], [18]:

Form:

$$\begin{pmatrix} A & -BQ_2^{-1}B^T \\ -Q & -A^T \end{pmatrix}$$

(in the special case (1) $BQ_2^{-1}B^T = 0$)

and calculate the eigenvectors:

$$\begin{pmatrix} b_1 \\ \hline c_1 \end{pmatrix}, \ldots, \begin{pmatrix} b_n \\ \hline c_n \end{pmatrix}, \ldots, \begin{pmatrix} b_{2n} \\ \hline c_{2n} \end{pmatrix}$$

Pick out of these the ones that correspond to the n eigenvalues with negative real part. Then S is obtained by:

$$S = (c_1, \ldots, c_n)(b_1, \ldots, b_n)^{-1}$$

2.1.3. Companion form methods

Other structures that might be useful are the companion form (phase variable form), the Schwartz's canonical form and transformations related to these, based upon the Cayley - Hamilton theorem for matrices [2,10,14]. The companion form or equivalently the transfer function is basic also for the recursive algorithm in 2.3.

These methods are quite useful, especially the one in 2.3., if the matrix A is already on e.g. companion form, but they are out of the question as general methods, because the transformation to companion form is time-consuming and not accurate for large systems.

2.2. Integration algorithm

The other type of algorithms proposed arise from the fact that the solution S to (1) can be expressed as [3, p 231]:

$$S = \int_0^\infty e^{A^T\sigma} Q\, e^{A\sigma} d\sigma = \lim_{t\to\infty} \int_0^t e^{A^T(t-s)} Q\, e^{A(t-s)} ds \qquad (3)$$

that is the steady state solution to the equation:

$$\frac{dS}{dt} = A^T S + SA + Q \qquad (4)$$

if the matrix A is stable.

It is therefore possible to obtain S by integrating the diff. eq. (4) to steady state.

Different methods of integration, containing various degrees of approximation, are possible.

2.2.1. Runge-Kutta and other direct methods.

A simple 4th order Runge-Kutta algorithm is briefly examined, and the method to use the Euler fundamental matrix Σ:

$$\Sigma(t) = \exp\left\{ t \cdot \begin{bmatrix} -A & 0 \\ Q & A^T \end{bmatrix} \right\} = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} = \begin{bmatrix} \left(\Sigma_{22}^T\right)^{-1} & 0 \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \qquad (5)$$

is also used straight forward:

$$S = \lim_{t\to\infty} \left[\Sigma_{21}(t) + \Sigma_{22}(t)\, S(0)\right] \Sigma_{22}^T(t) = \lim_{t\to\infty} \Sigma_{21}(t)\, \Sigma_{22}^T(t)$$

i.e. iterate:

$$S(k \cdot h) = \{\Sigma_{21}(h) + \Sigma_{22}(h)\, S(kh - h)\}\, \Sigma_{22}^T(h) \qquad (6)$$

to stationarity [11], [16], [19].

2.2.2. Davison's algorithm

Davison and Man in 1968 published a quite promising algorithm [6] based upon numerical integration of the expression (3) by the very simple Euler approximation. It can also be understood as equation (6) with instead of $\Sigma_{21}(h)\, \Sigma_{22}^T(h)$ the first term of its series expansion, $Q \cdot h$. The algorithm also contains a smart way of successively increasing the sampling interval.

2.2.3. New algorithm.

This has been adopted in a new algorithm, which uses the full eq. (6). Call $\phi = \Sigma_{22}^T(h)$ and $\tilde{Q} = \Sigma_{21}(h)\ \Sigma_{22}^T(h)$; then

$$S_0 = \tilde{Q}$$

$$S_1 = \phi^T S_0 \phi + S_0$$

$$S_2 = (\phi^T)^3 S_0 \phi^3 + (\phi^T)^2 S_0 \phi^2 + \phi^T S_0 \phi + S_0 =$$

$$= (\phi^T)^2 S_1 \phi^2 + S_1$$

.

.

.

$$S_{k+1} = (\phi^T)^{2^k} S_k \phi^{2^k} + S_k$$

and

$$S = \lim_{k \to \infty} S_k$$

Two methods can be used for calculating $\tilde{Q}$ and ϕ; The one described above in 2.2.1. using

$$\Sigma(h) = \exp\left\{h \cdot \begin{bmatrix} -A & 0 \\ Q & A^T \end{bmatrix}\right\}$$

and another one calculating $\tilde{Q}$ by direct series expansion of

$$\int_0^h e^{A^T s}\ Q\ e^{As}\ ds$$

and ϕ from

$$\phi = \exp\{A \cdot h\}.$$

2.3. Finite transfer function method

The interest in $A^TS + SA + Q = 0$ often arises when the incremental covariance matrix Vdt is to be calculated for the output y from the system:

$$\begin{cases} dx = A^T x dt + B d\omega \\ y = Cx \end{cases} \tag{8}$$

where ω is a Wiener process with incremental covariance matrix equal to Idt.

Then we have:

$$V = C\,S\,C^T \tag{9}$$

and Q has a natural rank factorization

$$Q = B\,B^T \tag{10}$$

B is supposed to have linear independent columns, or else the number of noise sources could be decreased.

By calculating the transfer function G(s) from $\frac{d\omega}{dt}$ to y in (8) we get V as:

$$V = \frac{1}{2\pi i} \int_{-i\infty}^{i\infty} G(s)\, G^T(-s)\, ds \tag{11}$$

and it is possible to evaluate this integral by use of the recursive algorithm developed by K.J. Åström [21].

3. COMPUTATIONAL AND PROGRAMMING ASPECTS.

3.1. Simple finite algebraic methods

Different methods are proposed in the literature to rewrite the equation:

$$A^TS + SA + Q = 0 \qquad (1)$$

to

$$n(n+1)/2$$

linear equations, [4], [2b], etc.

The main difference is found to be if the algorithm uses logical operations (ATPPAQ) or indexing vectors (KJATP) (see appendix). K.J. Åström has suggested the latter method, which is slightly more efficient. The algorithm for arbitrary systems of linear equations is chosen with reference to Forsythe - Moler [7].

The algorithms are quite easily coded and are very straight forward. It can be proved that "the big matrix" is always regular if A is regular, which is necessary to assure the existence of a unique solution; A and $-A^T$ must not have eigenvalues in common [9, p 225]. Errors are only introduced in the solvation of the linear equations, and this can be done to machine accuracy by the routine IMPRUV, if only the big matrix is not very illconditioned (and thus also A).

The main disadvantage with this attack is the rapid increase of $n(n+1)/2$ as n goes up. The memory requirements are soon overwhelming. For n = 20 45k words core memory would be necessary, with IMPRUV, 90 k, which is totally out of sight for a standard method. Matrix operations are time-consuming if the whole matrix is not in core at the same time, so that is no solution either.

The eigenvalue, eigenvector algorithms are complicated. The simpliest to code is the one proposed by Potter [18] and Mårtensson [16], EIGATP, that computes the 2n eigenvalues and eigenvectors to:

$$\begin{pmatrix} A & 0 \\ -Q & -A^T \end{pmatrix}$$

and then obtains the matrix S as the solution to a system of n linear equations with n right hand side vectors. However, since A generally is not symmetric, the eigenvalues and eigenvectors will be complex and the equations to be solved are consequently complex, which increases the number of operations to be done about six times. It also introduces a lot of new error sources. The methods used, subroutine CLES, refer to Fröberg [8, p 103].

The eigenvalue, eigenvector calculations are done in an up to date subroutine package, EIGENPAK, coded by K. Mårtensson. Although this seems to be about the best there is, quite a lot of problems are connected with this calculation. Eigenvalues very alike causes difficulties and considerable errors, especially when n increases.

The diagonalization method described by e.g. Jameson [10] has the advantage of less memory requirement. Instead of eigenvectors of a 2n x 2n matrix only eigenvectors of two n x n matrices (A and A^T) are needed, but it is on the other hand necessary to solve two systems of complex equations and to perform two complex matrix multiplications and one division of a complex matrix with a complex number, so no gain in computation rate and accuracy is achieved. It is also remarkable and disadvantageous for both the eigenvector algorithms that it seems impossible to use the symmetry of Q (and S) to any extent.

The companion form methods give short and neat coding, but they contain procedures that are not very numerically accurate and quite slow. No good transformation to companion form exists! The one based on the Newton identities, [20, p 50], the so called Leverriere's algorithm, gives a short program, TOBSKAN, but bad accuracy for large matrices. If, however, this step is already done or still has to be done, this attack to use the full structure is tempting (see 3.3).

Jameson's method has been coded but found slow and inaccurate for large n and no better than KJATP for small n. No attempt was therefore done to use the symmetry to reduce the number of operations and the already quite small memory requirement ($\sim n^2$).

None of the Barnett - Storey algorithms [2a - c] are coded or tested on the examples in chapter 4.

3.2. Integration algorithms

The Runge-Kutta integration in 2.2.1. was done with a fast routine, described in CACM [5] in Algol. Especially when A has widely spread eigenvalues or eigenvalues with low damping the step size has to be chosen short to assure sufficient accuracy, and the number of iterations to steady state can be large.

The step size can be made larger for the direct method with Euler matrix, but there is still the problem of widely spread eigenvalues and eigenvalues with low damping. The routine is considerably faster than the Runge-Kutta method. However, the memory requirement is larger because of the exponentiation of a 2n x 2n matrix. MEXP7T, the subroutine used for this, written by K. Mårtensson [15], is perhaps wasting a little with memory in order to gain speed, but it is fast, general and accurate, and uses all the advantages of scaling and zero-shift.

Since the methods below have made the convergence much faster by successively increasing the step length without introducing any further errors, the 2.2.1.-algorithms are not further examined on the examples in chapter 4. One possible use of them, especially the Runge-Kutta integration, is, however, to improve an approximative solution.

The Davison algorithm [6] was coded straight forward and examined in some examples. No advantage was taken of the symmetry. The fact that it converged in quite a few steps was verified, but the limit value was not the correct solution if the step length was not chosen very small. The truncation error in the Euler approximation of

the integral, or equivalently, in the series expansion of $\Sigma_{21}(h)\ \Sigma_{22}^{T}(h)$, has considerable effect. A very short step size on the other hand results in a $\phi = \Sigma_{22}^{T}(h)$ very near the unit matrix, and thus a loss of numerical significance.

Davison also recommends the Crank - Nicholson method for the calculation of $\phi = \exp Ah$. This has not been adopted since MEXP7T is more accurate and more general; the Crank - Nicholson method contains solution of a system of linear equations with a matrix, which is not necessarily regular.

All these complaints about the Davison algorithm inspired to the new algorithm in 2.2.3., which is using most of the advantages of the former.

$$\begin{cases} S_0 = \tilde{Q} \\ S_{k+1} = (\phi^{2^k})^T\ S_k\ \phi^{2^k} + S_k \end{cases} \tag{2}$$

The new algorithm is still very easily coded because of the use of iterative and recursive methods. A great deal of the work is done in MEXP7T, the routine mentioned above.

One variant, PHATPE, of the algorithm uses MEXP7T for determining both ϕ and $\tilde{Q}$ (see 2.2.3.), the other variant, PHATP, only for the ϕ calculation. $\tilde{Q}$ is then calculated in a special subroutine (QKRULL) using the series expansion:

$$\tilde{Q} = Q \cdot h + \frac{h^2}{2}\{A^TQ + QA\} + \frac{h^3}{3!}\{(A^T)^2Q + 2A^TQA + QA^2\} + \ldots$$

leading to the algorithm:

$$\begin{cases} T_1 = Q \cdot h \\ T_{k+1} = \frac{h}{k+1}\{A^TT_k + T_kA\} = \frac{h}{k+1}\{(T_kA)^T + T_kA\} \end{cases}$$

$$\begin{cases} S_1 = T_1 \\ S_{k+1} = S_k + T_{k+1} \end{cases} \tag{3}$$

with the stop condition:

$$|| T_k || / || S_k || < 10^{-10} \quad \text{or} \quad k > 35$$

The numbers 35 and 10^{-10} depend on the word length of the computer.

The choice between PHATP and PHATPE is quite hard. The main disadvantage of PHATPE is the larger memory requirement (35 resp. $12 * n^2$ words in matrices). It also contains quite a lot of meaningless multiplications with zero. The only favour is the smarter use of scaling in MEXP7T than in QKRULL. This makes PHATPE more fool proof. Less care has to be taken of the choice of the step size h. It can be difficult to choose h to get the best balance between the errors from algorithm (2) and algorithm (3) as well as to get the optimal computing time without deap knowledge of the matrix A and its eigenvalues.

There may be better methods for the computing of $\tilde{Q}$. Numerical integration of:

$$\int_0^h e^{A^T s} \, Q \, e^{As} \, ds$$

by Simpson's method for instance is not investigated, and it is possible to use the better scaling of PHATPE without needing the 2n x 2n exponentiation with a lot of zero multiplications, if that scaling is specially coded.

Another difficult point is the stopping condition for the iteration (2). This is linked together with the question:"When is a solution accurate enough?"

There are mainly two possibilities:

1. Stop when $S_{k+1} - S_k$ is small, i.e. when the iteration has converged.
2. Stop when the $A^T S_k + S_k A + Q$ is sufficiently small, i.e. when S_k has come close to the real solution.

The condition 1. is the only realistic alternative. It is no good to continue to iterate if no change is to be expected other than that, caused by numerical inaccuracy. But 1. is hard to relate to the wanted accuracy based upon the errors in A and Q. A large change in S might not effect the sum of $A^TS + SA + Q$ and it would then not be reasonable to require S to be very accurate. The problem is analogue to that in usual linear equations: Is the exactness of the solution to $Ax = b$ to be measured in b or in x? The condition number of A, which can vary very much, is the proportion factor between the two measures.

In the algorithms 1. is used, but alarm is given if 2. is not fulfilled. Consequently, it is possible that the iteration could be stopped earlier with sufficient resulting accuracy and a little waste of time is thinkable.

It is also possible that the limit value does not satisfy the equation very well, because of bad conditioning or because of poor numerical accuracy.

In that case an improvement algorithm would be valuable; some sort of iterative procedure with no or at least very small errors (no first stage computions like ϕ and $\tilde{Q}$), that takes advantage of a good starting value. The Runge-Kutta algorithm is possible to use for this purpose, but there are probably simpler methods that also can guarantee convergence.Further research is recommended.

For the PHATP and PHATPE routines it is necessary that A is stable for convergence,and full advantage of the symmetry is perhaps not taken.

3.3. Finite transfer function method

In ref. [21] the variance calculation is described for a system with single input, single output given as a transfer function. This is easily generalized to get the covariance between two outputs from a single input system and also to get the covariance of a multi input, multi output system. COLOMU (see appendix) performs the latter calculation. An important fact is that common factors in the transfer functions do not effect the result.

COLOMU contains very little internal storage; The only problem is the inefficient way of storing a system as a multi-variable transfer function. Fields with three indices are necessary. The number of operations seems to be minimized in COLOMU, but it might be possible that difficulties with accuracy arise. The algorithm contains a lot of adding-subtracting operations where numerical significance could be lost. Favourable is, however, that the terms in the final addition [21].

$$\begin{cases} I_0 = 0 \\ I_k = I_{k-1} + \beta_k^2/2\alpha_k \end{cases}$$

are positive.

The accuracy is often a problem in the recursive algorithms compared with the iterative ones.

COLOMU also tests if the denominator polynomial is stable. If not, no result is obtained.

If the system is given in the state variable form, S(A, B, C), the transfer function must be computed before COLOMU is applicable. This can be done by TOBSMVS, a Leverrière algorithm, the whole of which is administrated by NEKABC. When only a few of the variance or covariance elements are wanted, the computation can be greatly simplified by a proper choice of the C matrix. Noteworth is that TOBSMVS, which has considerable accuracy problems, computes the first coefficients of the transfer function with the smallest errors. The error increases fast in the last few coefficients. COLOMU, on the other hand, uses the last coefficient only once, but the first coefficients several times. The two routines are in balance

with respect to accuracy. No multiplicative error propagation seems to appear.

If COLOMU is to be used to solve the general

$$A^TP + PA + Q = 0 \qquad (1)$$

equation, one more nontrivial step has to be performed. The matrix Q must be rank factorized into:

$$BB^T = Q$$

This can be done with the routine DECOMSYM [7 , p 114]. The rank of Q gives the number of independent noise sources e_i in the model:

$$\begin{cases} \dot{x} = A^Tx + Be \\ y = x \qquad\qquad (C = I) \end{cases}$$

i.e. the number of columns in the matrix B.

This solution of (1) is administrated from the subroutine NEKATP. An additional ENTRY to the routine is provided for the case when the rank factorization is already done (but still C = I).

The method does not work for unstable A matrices and needs a positive semidefinite Q matrix, but takes full advantage of the symmetry of (1).

4. COMPARISON OF THE RESULTS ON A SAMPLE OF TEST EXAMPLES. RECOMMENDATIONS.

4.1. Description of the test examples.

1) $n = 2$

$$A = \begin{pmatrix} -3 & 0 \\ 0 & -2 \end{pmatrix} \qquad \Rightarrow S = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$Q = \begin{pmatrix} 6 & 5 \\ 5 & 4 \end{pmatrix}$$

Eigenvalues: - 3, - 2, stable
Infinite Q

2) $n = 2$

$$A = \begin{pmatrix} -2 & -3 \\ -5 & -10 \end{pmatrix} \qquad \Rightarrow S = \begin{pmatrix} -1,0833 \ldots & 0,333 \ldots \\ 0,333 \ldots & -0,1500 \end{pmatrix}$$

$$Q = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

Eigenvalues: $-6 \pm \sqrt{31} = -11,56 \ldots, -0,44 \ldots$ stable
Negative definite Q.

3) $n = 2$

$$A = \begin{pmatrix} -1 & 2 \\ 0 & -2 \end{pmatrix} \qquad \Rightarrow S = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$Q = \begin{pmatrix} 2 & -2 \\ -2 & 4 \end{pmatrix}$$

Eigenvalues: -1, -2 stable
Positive definite Q.

4) n = 3

$$A = \begin{pmatrix} -0.1 & 1.0 & 0 \\ 0 & 0 & 1.0 \\ 0 & -2.26 & -0.2 \end{pmatrix} \qquad Q = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Rightarrow S = \begin{pmatrix} 50.0... & 4.641\ 704\ 6108 & 22.103\ 355\ 289 \\ & 9.710\ 185\ 8710 & 2.275\ 090\ 5357 \\ & & 13.875\ 452\ 678 \end{pmatrix}$$

Eigenvalues: - 0.01, - 0.1 ± 1.5 i stable
Positive definite Q.

5) n = 3

$$A = \begin{pmatrix} -1 & 0 & -3 \\ -3 & -3 & 4 \\ 0 & 0 & -2 \end{pmatrix} \qquad Q = \begin{pmatrix} 16 & 7 & 20 \\ 7 & 6 & -1 \\ 20 & -1 & 26 \end{pmatrix}$$

$$\Rightarrow S = \begin{pmatrix} 5 & 1 & 3 \\ 1 & 1 & 0 \\ 3 & 0 & 2 \end{pmatrix}$$

Eigenvalues: - 1, - 2, - 3 stable
Indefinite Q.

6) n = 3

$$A = \begin{pmatrix} -1 & 0 & 3 \\ -3 & -3 & 4 \\ 0 & 0 & -2 \end{pmatrix} \qquad Q = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Rightarrow P = \begin{pmatrix} 0.875 & -0.125 & -1.25 \\ -0.125 & 0.16667 & 0.20833... \\ -1.25 & 0.20833... & 2.54166\underline{7} \end{pmatrix}$$

Eigenvalues: - 1, - 2, - 3
Positive definite Q

7) n = 4

$$A = \begin{pmatrix} -10 & -7 & -8 & -7 \\ -7 & -5 & -6 & -5 \\ -8 & -6 & -10 & -9 \\ -7 & -5 & -9 & -10 \end{pmatrix} = P = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix}$$

$$Q = \begin{pmatrix} 152 & 82 & 124 & 131 \\ 82 & 38 & 49 & 52 \\ 124 & 49 & 68 & 71 \\ 131 & 52 & 71 & 76 \end{pmatrix}$$

Eigenvalues:
- 0.010 150 05
- 0.843 107 15
- 3.858 057 45
-30.288 685 33

Q is indefinite.

8) n = 6

$$A = \begin{pmatrix} -0.1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2.0 & 10 & 10 & 5 \\ 0 & 3 & 0 & -3.0 & 1 & 0 \\ 7 & 2 & 0 & 0 & -10 & 0 \\ 32 & 15 & 0 & 0 & 100 & -50 \end{pmatrix}$$

$$Q = \begin{pmatrix} -127.6 & -31 & 0 & 0 & -207 & -59.8 \\ -31 & +2 & 0 & -3 & -2 & -75 \\ 0 & 0 & 4 & -10 & -10 & -5 \\ 0 & -3 & -10 & 6 & -1 & 0 \\ -207 & -2 & -10 & -1 & 20 & -500 \\ -59.8 & -75 & -5 & 0 & -500 & 500 \end{pmatrix}$$

$$\Rightarrow P = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

Eigenvalues: -0.1, -1.0, -2.0, -3.0, -10.0, -50.0

Q is indefinite.

9) n = 6

A is the same as in ex. 8.
Q is the unit matrix (positive definite).

=>

$$P = \begin{pmatrix} 1498.5525 & 90.1989 & 2.3057 & 12.2917 & 7.3124 & 0.2488 \\ 90.1989 & 21.1272 & 0.9204 & 4.5048 & 2.8109 & 0.0994 \\ 2.3057 & 0.9204 & 0.2500 & 0.5000 & 0.4503 & 0.0240 \\ 12.2917 & 4.5048 & 0.5000 & 1.8333 & 1.2698 & 0.0517 \\ 7.3134 & 2.8109 & 0.4503 & 1.2698 & 1.2580 & 0.0631 \\ 0.2488 & 0.0994 & 0.0240 & 0.0517 & 0.0631 & 0.0124 \end{pmatrix}$$

10) n = 8

$$A = \begin{pmatrix} -1 & -5 & 3 & 7 & -9 & -2 & -8 & 0 \\ 20 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 1 & -3 & -100 & -0.3 \\ 0 & 0 & 0 & -5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & -0.1 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & -0.01 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 \end{pmatrix}$$

$$Q = \begin{pmatrix} 2 & -15 & -3 & -7 & 9 & 2 & 8 & 0 \\ -15 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 4 & -2 & -1 & 3 & 100 & 0.3 \\ -7 & 0 & -2 & 10 & 0 & -6 & 4 & 0 \\ 9 & 0 & -1 & 0 & 40 & 0 & 0 & 0 \\ 2 & 0 & 3 & -6 & 0 & 0.4 & 0 & 0 \\ 8 & 0 & 100 & 4 & 0 & 0 & 0.04 & -2 \\ 0 & 0 & 0.3 & 0 & 0 & 0 & -2 & 2 \end{pmatrix}$$

=> P =

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

Eigenvalues: $-1 \pm 10\,i$, -2, -5, -10, -0.1, -0.01, -0.5
Q is indefinite.

11) n = 8

A is the same as in ex. 10.

Q is the unit matrix (pos. def.).

=> P =

1.243	0.037	0.123	-0.186
0.037	0.314	-0.168	-0.207
0.123	-0.168	0.435	4.903
-0.186	-0.207	4.903	27233.797
-0.596	0.225	-0.205	0.408
0.054	0.146	-0.661	-38.933
3.043	1.266	-17.567	-68663.982
0.302	-0.128	-6.717	-135574.474

-0.596	0.054	3.043	0.302
0.225	0.146	1.266	-0.128
-0.205	-0.661	-17.567	-6.717
0.408	-38.933	-68663.982	-135574.474
0.566	0.065	-1.967	-1.080
0.065	23.754	1020.775	1734.203
-1.967	1020.775	173283.105	341093.451
-1.080	1734.203	341093.451	682191.932

4.2. Results of the test batch

Subroutine	Example number											
	1	2	3	4	5	6	7	8	9	10	11	
ATPPAQ	8	9	8	20	23	21	52	201	No	660	No	Time, ms
	11	10-11	11	10	10	10	10	9-10	test	7	test	Signif. digits in Q
	11	11	11	10	10	10	8	7-8		5		Signif. digits in S
KJATP	9	7	9	16	17	16	39	147	151	521	547	Time, ms
	11	10	11	10	10	10	10	9-10	9	7	5-6	Signif. digits in Q
	11	11	11	10-11	10	10	8	7-8	7	4	3-4	Signif. digits in S
PHATP	110	80	100	190	200	200	430	635	No	1420	No	Time, ms
	≥ 7	≥ 7	≥ 7	≥ 7	≥ 7	≥ 7	≥ 7	≥ 7	test	≥ 7	test	Signif. digits in Q
	10	8-9	10	8	10	9	7	5		3-4		Signif. digits in S
NEKATP			16	43(32)		38(32)			361(337)		1002	Time, ms
(NEKAB)			11	10		10			5-6		1-2	Signif. digits in Q
			10	10		11			5		2-3	Signif. digits in S
EIGATP	120	180	123	358	471	445	867	2245	2690	5815	5975	Time, ms
	10	10	10	9	10	9-10	9	7	4-5	4	0	Signif. digits in Q
	11	11	11	9	10	10	8	8	7	5	0	Signif. digits in S

4.3. Recommendations

KJATP is the best general method if the computer is not very small.

Very large systems are difficult to handle. Best routine seems to be PHATP.

If eigenvalues, eigenvectors or transfer function have to be computed although use of this might be valuable.

Improvement routine would be useful if an exact answer is required.

REFERENCES

[1] Anderson: An Algebraic Solution to the Spectral Factorization Problem, IEEE Trans Automatic Control AC-12, No. 4, pp 410 - 414, August, 1967.

[2 a] Barnett & Storey: The Liapunov Matrix Equation and Schwartz's Form, IEEE Trans Automatic Control (Correspondence) AC-12, pp 117 - 118, February, 1967.

[2 b] Barnett & Storey: Remarks on Numerical Solution of the Lyapunov Matrix Equation, Electronic Letters, Vol. 3, pp 417 - 418, September, 1967.

[2 c] Barnett & Storey: On the General Functional Matrix for a Linear System, IEEE Trans Automatic Control (Short Paper) AC-12, pp 436 - 438, August,1967.

[3] Bellman: Introduction to Matrix Analysis, Mac Graw Hill, 1960.

[4] Chen & Shieh: A Note on Expanding $PA + A^TP = -Q$, IEEE Trans Automatic Control (Correspondence) AC-13, pp 122 - 123, February, 1968.

[5] Collected Algorithms from CACM, Algorithm No. 9, 1960.

[6] Davison & Man: The Numerical Solution of $A^1Q + QA = -C$, IEEE Trans Automatic Control (Correspondence) AC-13, pp 448 - 449, August, 1968.

[7] Forsythe & Moler: Computer Solution of Linear Algebraic Systems, Prentice Hall, 1967.

[8] Fröberg: Lärobok i Numerisk Analys, 2:a uppl., Svenska Bokförlaget/Bonniers, 1967.

[9] Gantmacher: The Theory of Matrices, Chealsea, New York, 1959.

[10] Jameson: Solution of the Equation AX + XB = C by Inversion of an M x M or N x N Matrix, SIAM J. Appl. Math. Vol. 13, No. 5, September, 1968.

[11] Kalman & Englar: A User's Manual for the Automatic Syntethis Program, NASA CR-475, Washington, June, 1966.

[12] Kleinman: Solution of Linear Regulator Problem with Infinite Terminal Time, IEEE Trans Automatic Control (Correspondence) AC-13, p 114, February, 1968.

[13] LaSalle & Lefochetz: Stability by Liapunov's Direct Method with Applications,New York Academic Press, 1961.

[14] Ma: A Finite Series Solution of the Matric Equation AX - XB = C, SIAM J. Appl. Math., Vol. 14, No. 3, May, 1966.

[15] Mårtensson: Linear Quadratic Control Package, Part I, Report 6802, Lund Institute of Technology, 1968.

[16] Mårtensson: On the Riccati Equation (To appear).

[17] Parks: A New Look at the Routh-Huewitz Problem Using Liapunov's Second Method, Bull. Polish Sci, Tech. Sci. Ser., Vol. 12, pp 19 - 21, June, 1964.

[18] Potter: Matrix Quadratic Solutions, SIAM J. Appl. Math., Vol. 14, No. 13, May, 1966.

[19] Vaughan: A Negative Exponential Solution for the Matrix Riccati Equation, IEEE Trans Automatic Control (Short Paper), AC-14, pp 72 - 75, February, 1969.

[20] Åström: Reglerteori, Almqvist & Wiksell, 1968.

[21] Åström: Stochastic Control (To appear on Academic Press).

APPENDIX

Description of the following routines, available from the library of Regleringsteknik, LTH.

Computer used: CD-3600, wordlength 48bit, cycle time 1.0μs

```
      SUBROUTINE ATPPAQ(A,QP,N,IFAIL,IA)
C
C     SOLVES THE EQUATION (A)T*X+X*A=-Q, WHERE X AND Q ARE SYMMETRIC
C     MATRICES,AND A IS REQUIRED TO HAVE EIGENVALUES WITH NEGATIVE
C     REAL PART.
C     REFERENCE,CHEN,IEEE FEB.68,P122.
C     AUTHOR,K.MORTENSSON 05/06-68.
C
C     A-MATRIX OF ORDER NXN(MAX 8X8).
C     QP-SYMMETRIC MATRIX OF ORDER NXN.WHEN CALLING ATPPAQ,QP SHOULD
C     CONTAIN Q,UPON RETURN QP CONTAINS THE SOLUTION X.
C     IFAIL-RETURNED 0 IF THE ROUTINE HAS EXECUTED CORRECTLY,1 IF NOT.
C     IA-DIMENSION PARAMETER.
C
C     SUBROUTINE REQUIRED
C             MIART
C
      DIMENSION P(36,37),A(IA,IA),QP(IA,IA),IBETA(36),JBETA(37)
C
```

```
      SUBROUTINE KJATP    (A,QP,N,IFAIL,IA)
C
C     SOLVES THE EQUATION A(TRANSPOSED)*P + P*A = -Q, WHERE P AND Q ARE
C     SYMMETRIC MATRICES, AND A IS REQUIRED TO HAVE EIGENVALUES WITH
C     NEGATIVE REAL PART.
C     REFERENCE, KJ ASTROEM.
C     AUTHOR, PER HAGANDER 16/3-69
C
C     A- MATRIX OF ORDER NXN (MAX 8X8)
C     QP- SYMMETRIC MATRIX OF ORDER NXN . WHEN CALLING, QP SHOULD
C     CONTAIN Q, AT RETURN QP CONTAINS THE SOLUTION P.
C     IFAIL- RETURNED 0, IF THE ROUTINE HAS EXECUTED CORRECTLY,ELSE 1 OR
C     2.  NOTE THE ACCURACY MIGHT BE LOW IF THE EQUATION IS ALMOST
C     SINGULAR, AND THAT A SINGULARITY MIGHT BE COVERED BY NUMERICAL
C     INACCURACY.
C     IA- DIMENSION PARAMETER.
C     MATRIX A IS NOT DESTROYED.
C
C     SUBROUTINES REQUIRED.
C             DECOM
C             SOLVE
C
      DIMENSION B(36,36),IND(8),X(36),HL(36)
      DIMENSION A(IA,IA),QP(IA,IA)
C
```

```
      SUBROUTINE DECOM (NN,A,UL,ISING,IA)
C
C     REFERENS FORSYTHE,MOLER  COMPUTER SOLUTION OF
C     LINEAR ALGEBRAIC SYSTEMS
C     AUTHOR,PER HAGANDER 5/9 68.
C     COMPUTES TRIANGULAR MATRICES L AND U  AND PERMUTATIONMATRIX P
C     SO THAT LU = PA,
C     USING GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING.
C     IPS-VECTOR  OF ORDER 50, RETURNED CONTAINING ROWINDICES.
C     A-MATRIX OF ORDER NNXNN (MAX(NN) = 50).
C     UL-MATRIX OF ORDER  NNXNN (MAX(NN)=50),RETURNED CONTAINING
C     THE TRIANGULAR MATRICES  L-I  AND U
C     ISING-ISING IS RETURNED 1 IF ANY OF THE ROWS OF A IS ZERO
C                             2   IF A IS OTHERWISE SINGULAR
C     THE MATRIX A IS NOT DESTROYED.
      DIMENSION A(IA,IA), UL(IA,IA)
      DIMENSION SCALES(50)
      COMMON/BLOCKU/ IPS(50)
```

```
      SUBROUTINE SOLVE (NN,UL,B,X,IA)
C
C     REFERENS FORSYTHE,MOLER  COMPUTER SOLUTION OF
C     LINEAR ALGEBRAIC SYSTEMS
C     AUTHOR,PER HAGANDER 5/9 68.
C     SOLVES AX=B  USING UL FROM DECOM.
C     UL-MATRIX OF ORDER NNXNN, FROM DECOM
C     IPS-VECTOR OF ORDER NN CONTAINING PERMUTED ROWINDICES,
C     FROM DECOM.
C     B-VECTOR OF ORDER NN, CONTAINING THE RIGHT-HAND VECTOR OF
C     THE SYSTEM OF EQUATIONS TO BE SOLVED.
C     X-VECTOR OF ORDER NN, RETURNED CONTAINING THE SOLUTION.
C         MAX(NN) = 50, MIN(NN) = 2.
C     THE MATRIX UL AND THE VECTORS B AND IPS ARE NOT DESTROYED.
C
      DIMENSION UL(IA,IA), B(IA), X(IA)
      COMMON/BLOCKU/ IPS(50)

        SUBROUTINE IMPRUV  (NN, A, UL, B, X,ISING,IA)
C
C       IMPROVES AN APPROXIMATE SOLUTION TO MACHINE ACCURACY USING SOLVE,
C       AND UL FROM DECOM.
C       A-MATRIX OF ORDER NN.
C       UL-MATRIX OF ORDER NNXNN., FROM DECOM.
C       B-VECTOR OF ORDER NN.
C       X-VECTOR OF ORDER NN, CONTAINING THE APPROXIMATE SOLUTION,
C       MOSTLY FROM SOLVE,
C       RETURNED CONTAINING THE MORE ACCURATE SOLUTION.
C       ISING-ISING IS RETURNED 3 IF IMPRUV DOES NOT CONVERGE,
C       THAT IS IF A IS ALMOST SINGULAR.
C       MAX(NN) = 50, MIN(NN) = 2.
C       THE MATRICES A AND UL  AND THE VECTOR B ARE NOT DESTROYED.
C
C       NEEDED SUBROUTINES
C              SOLVE
C
        DIMENSION A(IA,IA), UL(IA,IA), B(IA), X(IA), R(50), DX(50)
        DOUBLE PRECISION SUM

          SUBROUTINE CLES(N,NM,A,B,U,V,IFAIL,IA,IM)
C
C       SOLUTION OF COMPLEX LINEAR EQUATIONS,(A+IB)(INVERSE)*(U+IV),
C       TRIES FIRST IF A IS REGULAR.  IF NOT, THEN IF R, THEN IF  F=A+R*B
C       WITH R=0.5(0.5)5.0 .   USES SUBROUTINE CLESAR.
C       REFERENSE, FROBENIUS.
C       AUTHOR, PER HAGANDER 2278 69.
C
C       A- MATRIX OF ORDER NXN, REAL PART.          NOT DESTROYED.
C       B- MATRIX OF ORDER NXN IMAGINARY PART.      NOT DESTROYED.
C       U- MATRIX OF ORDER NXNM, REAL PART.           NOT DESTROYED.
C       U- MATRIX OF ORDER NXNM, REAL PART OF THE RIGHT HAND SIDE.
C            RETURNED CONTAINING THE REAL PART OF THE SOLUTION.
C       V- MATRIX OF ORDER NXNM, IMAGINARY PART OF THE RIGHT HAND SIDE.
C            RETURNED CONTAINING THE IMAGINARY PART OF THE SOLUTION.
C       IFAIL- IS RETURNED EQUAL TO   0      IF OK.
C                                     1      IF  NO SUCCESS.
C       IA, IM- DIMENSION PARAMETERS.
C       MAX(N)=MAX(NM)=20
C
C       SUBROUTINES REQUIRED
C              DECOM
C              SOLVE
C              IMPRUV
C              CLESAR
C
C
        DIMENSION A(IA,IA),B(IA,IA),U(IA,IM),V(IA,IM)
        DIMENSION F(20,20),G(20,20),US(20,20),VS(20,20)
C
```

FTN5.48 01/04-69

```
      SUBROUTINE EIGENPAK(A,N,IA,T,EVR,EVI,VECR,VECI,INDIC)
C
C     THIS SUBROUTINE FINDS ALL THE EIGENVALUES AND THE EIGENVECTORS OF
C     A REAL GENERAL MATRIX A OF ORDER N.FIRST IN THE SUBROUTINE SCALE
C     THE MATRIX IS SCALED SO THAT CORRESPONDING ROWS AND COLUMNS ARE
C     APPROXIMATELY BALANCED AND THEN THE MATRIX IS NORMALISED SO THAT
C     THE VALUE OF THE EUCLIDIAN NORM OF THE MATRIX IS EQUAL TO ONE.
C     THE EIGENVALUES ARE COMPUTED BY THE QR DOUBLE-STEP METHOD IN THE
C     SUBROUTINE HESQR.THE EIGENVECTORS ARE COMPUTED BY INVERSE ITERA-
C     TION IN THE SUBROUTINE REALVE,FOR THE REAL EIGENVALUES,OR IN THE
C     SUBROUTINE COMPVE,FOR THE COMPLEX EIGENVALUES.
C     REFERENCE,ALG.243,CACM,VOL11,NUMB.12,68.
C     AUTHOR,K.MORTENSSON 21/03-69.
C
C     A-MATRIX OF ORDER NXN (N MAX 20).
C     IA-DIMENSION PARAMETER.
C     T-NUMBER OF BINARY DIGITS IN THE MANTISSA OF SINGLE PRECISION
C     FLOATING-POINT NUMBER.(FOR CDC-3600 T=36).
C     EVR-VECTOR OF DIMENSION N,CONTAINING THE REAL PARTS OF THE EIGEN-
C     VALUES.
C     EVI-VECTOR OF DIMENSION N,CONTAINING THE CORRESPONDING IMAGINARY
C     PARTS OF THE EIGENVALUES.
C     VECR-MATRIX OF OFDER NXN,CONTAINING THE REAL PARTS OF THE NORMA-
C     LIZED EIGENVECTORS.EIGENVECTOR NUMBER I IS FOUND IN THE FIRST N
C     POSITIONS OF COLUMN I.
C     VECI-MATRIX OF ORDER NXN,CONTAINING THE CORRESPONDING IMAGINARY
C     PARTS OF THE EIGENVECTORS.
C     INDIC-VECTOR OF DIMENSION N INDICATING THE SUCCESS OF THE SUBROU-
C     TINE AS FOLLOWS.
C     VALUE OF INDIC(I)   EIGENVALUE I   EIGENVECTOR I
C           0               NOT FOUND      NOT FOUND
C           1               FOUND          NOT FOUND
C           2               FOUND          FOUND
C     THE REAL EIGENVECTOR IS NORMALISED SO THAT THE SUM OF THE SQUARES
C     OF THE COMPONENTS IS EQUAL TO ONE.
C     THE COMPLEX EIGENVECTOR IS NORMALISED SO THAT THE COMPONENT WITH
C     THE LARGEST VALUE IN MODULUS HAS ITS REAL PART EQUAL TO ONE AND
C     THE IMAGINARY PART EQUAL TO ZERO.
C     THE ORIGINAL MATRIX A IS DESTROYED IN THE SUBROUTINE.
C
C     SUBROUTINE REQUIRED
C             SCALE
C             HESQR
C             REALVE
C             COMPVE
C
      DIMENSION A(IA,IA),VECR(IA,IA),VECI(IA,IA),EVR(IA),EVI(IA),
     1INDIC(IA)
      DIMENSION IWORK(20),LOCAL(20),PRFACT(20),SUBDIA(20),WORK1(20),
     1WORK2(20),WORK(20)
      DOUBLE PRECISION D1,D2,D3,PRFACT
C
```

```
      SUBROUTINE EIGAIP(A,QP,N,IFAIL,IA)
C
C     SOLUTION OF A(TRANSPOSED)*P + P*A + Q = 0  BY USE OF EIGENVALUES
C     AND EIGENVECTORS.  Q SYMMETRIC.
C     REFERENSE. K. MORTENSSON.
C     AUTHOR. PER HAGANDER 22/8 69.
C     A- MATRIX OF ORDER NXN.     NOT DESTROYED.
C     QP- MATRIX OF ORDER NXN,  RETURNED CONTAINING THE SOLUTION P.
C     IA- DIMENSION PARAMETER.
C     MAX(N)=10.
C     IFAIL- RETURNED EQUAL TO
C            0    IF  OK
C            1    IF  INVERSION FAILS
C            2    IF  EIGENVALUES-EIGENVECTORS ARE NOT FOUND.
C            3    IF  MORE THAN N EIGENVALUES HAVE NEGATIVE REAL PART.
C            4    IF  THE NORM OF THE IMAGINARY PART OF THE SOLUTION IS
C                     GREATER THAN 1.E-7.
C
C     SUBROUTINES REQUIRED
C              EIGEMPAK
C              SCALE
C              HESQR
C              REALVE
C              COMPVE
C              CLES
C              CLESAR
C              DECOM
C              SOLVE
C              IMPROV
C              NORM
C
      DIMENSION A(IA,IA),QP(IA,IA)
      DIMENSION BM(20,20),EVR(20),EVI(20),VECR(20,20),VECI(20,20),INDIC(
     220),BR(10,10),BI(10,10),CR(10,10),CI(10,10)
```

```
      SUBROUTINE TOBSKAN(NN,A,C,T,P,S,IA)
C
C     AUTHOR PER HAGANDER 30/10 68
C
C     THIS SUBROUTINE COMPUTES FOR A GIVEN SYSTEM S(A,B,C,D) THE
C     TRANSFORMATIONMATRIX TO OBSERVABLE CANONICAL FORM.
C     THE COEFFICIENTS OF THE CHARACTERISTIC EQUATION ARE COMPUTED TOO.
C     THE ROUTINE CAN BE USED TO DETERMINE THE TRANSFORMATIONMATRIX TO
C     THE CONTROLLABLE CANONICAL FORM.
C
C     A, T, S ARE MATRICES OF ORDER NNXNN.
C     C AND P ARE VECTORS OF ORDER NN.
C     MAX(NN) = 50.
C
C     T IS RETURNED CONTAINING THE TRANSFORMATIONMATRIX.
C     P IS RETURNED CONTAINING THE COEFFICIENTS OF THE CHARACTERISTIC
C     EQUATION--  S**N + P(1)*S**(N-1) + ....+ P(N) = 0.
C     S IS USED IN THE COMPUTATIONS AND THE RETURNED ELEMENTS ARE
C     A CHECK ON THE ROUNDING OFF ERRORS.  S SHOULD THEORETICALLY BE 0.
C
C     IF THE ACTUAL INPUT MATRIX IS A-TRANSPOSED INSTEAD OF A;  AND
C     THE ACTUAL INPUT VECTOR  IS B  INSTEAD OF C,  THEN THE RESULTING
C     T IS THE INVERTED, TRANSPOSED TRANSFORMATIONMATRIX TO THE CONTROL-
C     LABLE CANONICAL FORM.
C
C     THE MATRIX A AND THE VECTOR C ARE NOT DESTROYED.
C
      DIMENSION A(IA,IA), C(IA), T(IA,IA), P(IA),S(IA,IA)
      DOUBLE PRECISION X(50)
C
```

```
      SUBROUTINE  PHATP  (A,QP,N,IFAIL,H,IA)
C
C     INTEGRATES THE EQUATION DP/DT = A(TRANSP)*P + P*A + Q
C     TO STATIONARITY USING FI=EXP(A*H)  AND QKRULL.
C     AUTHOR, PER HAGANDER 19/4 69.
C
C     A- MATRIX OF ORDER NXN  INPUT   NOT DESTROYED.
C     QP-MATRIX OF ORDER NXN   CONTAINING Q AS INPUT, RETURNED CON-
C     TAINING THE SOLUTION P.
C     MAX(N) = 20.
C     IA - DIMENSION PARAMETER.
C     H- IS THE STEPLENGTH, NOTE THE POSSIBILITY TO USE H FOR MATRIX
C     SCALING.
C     IFAIL-IF THE ROUTINE HAS EXECUTED CORRECTLY,
C           IFAIL IS RETURNED =0.
C           IF QKRULL WAS BAD, IFAIL IS SET =1,  AND RETURN TO CALLING
C           PROGRAM.
C           NOTE THAT QP IS NOT CHANGED, THUS DECREASE H AND CALL PHATP
C           AGAIN.
C           IF THE NORM OF THE CHANGE OF THE SOLUTION P AFTER 20
C           ITERATIONS STILL IS MORE THAN E-7, IFAIL IS SET =10.
C           IF THE DIFFERENCE BETWEEN THE INPUT Q AND THE RESULTING Q
C       HAS A NORM GREATER THAN E-7,  IFAIL IS INCREASED BY 100.
C
C     SUBROUTINES REQUIRED
C             NORM
C             MEXP7T
C             QKRULL
C
      DIMENSION A(IA,IA),QP(IA,IA)
       DIMENSION S(20,20),S0(20,20),FI(20,20),RR(20)
      EPS =1.E-7
C

      SUBROUTINE QKRULL(AH,QH,S,N,IA,IFAIL)
C
C      INTEGRATES EXP(A T(TRANSP)) * Q * EXP(AT)    OVER A SAMPLINGINTERVAL H
C     BY SERIES EXPANSION.
C     AUTHOR, PER HAGANDER 19/4 69
C
C     AH-MATRIX OF ORDER NXN     INPUT     NOT DESTROYED.
C     QH-MATRIX OF ORDER NXN     INPUT     NOT DESTROYED.
C      S -MATRIX OF ORDER NXN    OUTPUT.
C      QKRULL(AH,QH,QH,N,IA,IFAIL)  OVERWRITES QH BY S.
C     MAX(N) = 20.
C     IA -DIMENSIONPARAMETER.
C      IFAIL IS RETURNED =0 IF THE SERIES EXPANSION HAS CONVERGED IN
C      35 STEPS, OTHERWISE =1.
C
C     SUBROUTINES REQUIRED
C           NORM
C
      DIMENSION AH(IA,IA),QH(IA,IA),S(IA,IA)
      DIMENSION T(20,20),SLASK(20,20)
C
```

```
      SUBROUTINE MEXP7T(A,B,N,IA,NOTRACE)
C
C     COMPUTES B=EXP(A) BY ORIGIN SHIFT AND SERIES EXPANSION USING 7
C     TERMS.
C     AUTHOR,K.MORTENSSON 15/11-67.
C
C     A-NXN-MATRIX.
C     B-NXN-MATRIX.
C     IA-DIMENSION PARAMETER.
C     NOTRACE=0 MEANS THAT NO TRACE
C     COMPUTATION WILL BE PERFORMED.
C     MAXIMUM ORDER OF A AND B=20.
C     THE MATRIX A IS DESTROYED.
C
C     SUBROUTINE REQUIRED
C             NORM
C
      DIMENSION  A(IA,IA),B(IA,IA),C(7,20,20)
```

```
      SUBROUTINE  PHATPE (A,Q,N,IFAIL,H,IA)
C
C     INTEGRATES THE EQUATION DP/DT=A(TRANSP)*P + P*A + Q TO STATIONA-
C     RITY USING THE EULERMATRICES.
C     AUTHOR, PER HAGANDER 19/4 69.
C
C     A- MATRIX OF ORDER NXN   INPUT   NOT DESTROYED.
C     QP-MATRIX OF ORDER NXN CONTAINING Q AS INPUT, RETURNED
C     CONTAINING THE SOLUTION P.
C     MAX(N) = 10.
C     IA-DIMENSION PARAMETER.
C     H- IS THE STEPLENGTH.
C     IFAIL-  IS RETURNED  =0  IF THE ROUTINE HAS EXECUTED CORRECTLY.
C
C     SUBROUTINES REQUIRED
C             MEXP7T
C             NORM
C
      DIMENSION A(IA,IA),Q(IA,IA)
      DIMENSION S(10,10),S0(10,10),FI(10,10),RR(10),EA(20,20),EB(20,20)
      EPS =1.E-7
C
```

```
      SUBROUTINE NEKABC(A,B,C,N,NB,NC,T,P,V,IFAIL,IA,IB,IC)
C     COMPUTES THE COVARIANCE MATRIX OF THE OUTPUT OF THE SYSTEM
C                X(DOT)=A*X+B*E
C                Y=C*X
C     BY USE OF THE ROUTINE COLOMU.  IF C IS THE IDENTITY MATRIX, USE
C     NEKAB (ENTRY TO NEKATP) INSTEAD.  IF THE TRANSFER FUNCTION IS
C     GIVEN, USE COLOMU DIRECTLY.
C     AUTHOR, PER HAGANDER 3/9 69.
C     A   IS MATRIX OF ORDER  NXN      NOT DESTROYED.
C     B   IS MATRIX OF ORDER NXNB        NOT DESTROYED.
C     C   IS MATRIX OF ORDER NCXN        NOT DESTROYED.
C     P   IS VECTOR OF ORDER  N       RETURNED CONTAING THE DENOMINATOR
C         OF THE TRANSFER FUNCTION G.
C     T   IS MATRIX OF ORDER  NXNCXNB  (DIMENSION IA,IC,IA)    RETURNED
C         CONTAINING THE NUMERATOR OF THE TRANSFER FUNCTION G=
C         B(1,,,)*S**(N-1)+           +B(N,,,)
C         -----------------------------------
C         S**N+P(1)*S**(N-1)+.............+P(N)
C     IFAIL IS RETURNED  0  IF A IS STABLE
C                        1  IF A IS UNSTABLE
C     MAX(N)=20,  MAX(NB,NC)=N.
C     V IS MATRIX OF ORDER NCXNC   RETURNED CONTAINING THE COVARIANCE
C       MATRIX.
C     IA,IB,IC     ARE DIMENSION PARAMETERS.
C
C     SUBROUTINES REQUIRED
C             NORM
C             TOBSMVS
C             COLOMU
      DIMENSION A(IA,IA),B(IA,IB),C(IC,IA),T(IA,IC,IA),V(IC,IC),P(IA)
      DIMENSION Q(20)
C
```

```
      SUBROUTINE COLOMU(A,B,N,NB,NC,IFAIL,V,IA,IC)
C
C     CALCULATES THE COVARIANCE MATRIX FOR THE SYSTEM WITH THE TRANSFER
C     FUNCTION G=
C     B(1,,,)*S**(N-1)+..........+B(N,,,)
C     ----------------------------------
C     S**N+A(1)*S**(N-1)+.........+A(N)
C     REFERENCE,K.J.ASTROEM,  STOCHASTIC CONTROL THEORY.
C     AUTHOR, PER HAGANDER  24/6 69.
C
C     A IS THE DENOMINATOR POLYNOMIAL,  VECTOR OF ORDER N,  DESTROYED.
C     B IS THE NUMERATOR MATRIX POLYNOMIAL, MATRIX OF ORDER (N,NC,NB)
C     B IS ALSO DESTROYED.
C     V IS THE RESULTING COVARIANCE MATRIX , ORDER NCXNC.
C     MAX(NC)=20, NO LIMIT ON N, MAX(NB)=N.
C     IFAIL IS 0  IF A STABLE.
C     IFAIL IS 1  IF A IS INSTABLE.
C
C     SUBROUTINES REQUIRED
C             NONE
C
      DIMENSION A(IA),B(IA,IC,IA),V(IC,IC),BETA(20,20)
```

.48 10/01-69

```
      SUBROUTINE TOBSMVS(A,C,NA,NC,T,P,SN,IA,IC)
C
C     AUTHOR, PER HAGANDER 8/12 68.
C
C     THIS SUBROUTINE COMPUTES FOR A MULTIVARIABLE SYSTEM S(A,B,C,D)
C     THE COEFFICIENTS OF THE CHARACTERISTIC EQUATION, THAT IS THE
C     DENOMINATOR OF THE TRANSFER FUNCTION.
C     P IS RETURNED CONTAINING THESE COEFFICIENTS SO THAT THE POLYNO-
C     MIAL IS
C     S**NA + P(1)*S**(NA-1) + ..............+ P(NA-1)*S + P(NA)
C
C     MATRIX CONTAINING
C     T(1,I,J) = C(I,J)
C     T(2,.,.) = C*A + P(1)*C
C     T(3,.,.) = C*A*A + P(1)*C*A + P(2)*C
C     ETC.
C     IS ALSO  COMPUTED
C
C     THE NA*NC ROWS OF THE COLUMN T( 1,.,.)
C                                 T( 2,.,.)
C                                 .........
C                                 T(NA,.,.)
C     SPAN  THE ORTHOGONAL COMPLEMENT SPACE OF THE NONOBSERVABLE SUB-
C     SPACE.  THUS S(A,B,C,D) IS OBSERVABLE IF THE RANK OF THIS MATRIX
C     IS NA.  (CF. SUBROUTINE OBSTEST.)
C
C     THE COEFFICIENT MATRICES BI OF THE NUMERATOR POLYNOMIAL OF THE
C     TRANSFERFUNCTION ARE EASY TO OBTAIN FROM T.
C     (CF. KJA, REGLERTEORI.)
C     B1(I,J) = T( 1,I,K)*B(K,J)  , SUM OVER K.
C     ............
C     BN(I,J) = T(NA,I,K)*B(K,J)  , SUM OVER K.
C     OBS.
C     THE TRANSFER FUNCTION THUS OBTAINED
C     B1*S**(NA-1)+.........................+BN
C     ---------------------------------------
C     S**NA + P(1)*S**(NA-1)+.............+P(NA)
C     MAY CONTAIN NOT CONTROLLABLE OR NOT OBSERVABLE MODES.
C
C     IF A IS REPLACED BY A-TRANSPOSED AND
C        C IS REPLACED BY B-TRANSPOSED
C     THEN THE ROWS OF THE COLUMN MATRIX BUILT BY T SPAN  THE
C     CONTROLLABLE SUBSPACE.
C
C     A IS  NAXNA MATRIX                    NOT DESTROYED
C     C IS  NCXNA  MATRIX                   NOT DESTROYED
C     T  IS  NAXNCXNA  MATRIX
C     P IS NA VECTOR
C     MAX NA = 20
C     SN IS RETURNED AS TEST ON THE ACCURACY AND IS THE NORM OF THE
C     RIGHTHAND ZERO-MATRIX IN THE CAYLEY-HAMILTON EQUATION.
C
C     NEEDED SUBROUTINES
C          NORM
C
      DIMENSION A(IA,IA),C(IC,IA),T(IA,IC,IA),P(IA)
      DIMENSION S(20,20)
      DOUBLE PRECISION X(20),Z(20)
```

```
      SUBROUTINE MEKATP(A,QP,N,IFAIL,IA)
C
C     SOLVES THE EQUATION A(TRANSPOSED)*P+P*A=-Q. BY USE OF THE ROUTINE
C     COLOMO.   P AND Q ARE SYMMETRIC MATRICES.  A IS REQUIRED TO BE
C     STABLE AND Q POSITIVE SEMIDEFINITE.
C     REFERENCE. K.J.ASTROEM. STOCHASTIC CONTROL THEORY.
C     AUTHOR. PER HAGANDER 24/5 69.
C
C     A IS MATRIX OF ORDER NXN   NOT DESTROYED.
C     QP IS MATRIX OF ORDER NXN.  WHEN CALLING QP SHOULD CONTAIN Q.
C     AT RETURN QP CONTAINS THE SOLUTION P.
C     MAX(N)=20
C     IFAIL IS RETURNED 0  IF  OK
C                      -1  IF  A IS UNSTABLE.  NO SOLUTION.
C                       1  IF  RANK FACTORIZATION HAS FAILED.
C                       2  IF  RANK(Q)=0
C     IA IS DIMENSION PARAMETER
C     AN ADDITIONAL ENTRY IS PROVIDED TO CALCULATE THE COVARIANCE MATRIX
C     OF THE SYSTEM  X(DOT) = A*X + B*E.  OR SOLVING THE EQUATION
C     A*P + P*A(TRANSPOSED) = -B*B(TRANSPOSED).
C     B IS ENTERED IN QP AND NB IN IFAIL.  THE CALL SHOULD BE
C     CALL MEKATP(A,QP,N,NB,IA)
C     OBSERVE THAT QP IS ALTERED IN THE ROUTINE.
C
C     SUBROUTINES REQUIRED
C             DECOMSYM
C             FORSMVS
C             NORM
C             COLOMO
C
      DIMENSION A(IA,IA),QP(IA,IA)
      DIMENSION T(20,20,20),B(20,20),SL(20,20),P(20)
      II=20
C
```

```
      SUBROUTINE DECOMSYM(A,G,IA,N,EPS,IERR,TK,IRANK)
C
C     DECOMPOSES A POSITIVE DEFINITE (AND PROBABLY ALSO A POSITIVE
C     SEMIDEFINITE) SYMMETRIC MATRIX A INTO G*GTRANPOSE WHERE G IS A
C     LOWER TRIANGULAR MATRIX
C     REFERENCE FORSYTHE MOLER COMPUTER SOLUTION OF LINEAR ALGEBRAIC
C     SYSTEM  PG 114
C     AUTHOR IVAR GUSTAVSSON 27/1 1969
C
C     A         MATRIX TO BE DECOMPOSED
C     G         THE RESULTING LOWER TRIANGULAR MATRIX
C     IA        DIMENSION PARAMETER  MAX 20
C     N         DIMENSION OF MATRIX A AND G
C     EPS       QUANTITY USED FOR THE TEST OF DEFINITENESS
C     IERR=0    G EVALUATED WITHOUT PROBLEMS
C     IERR=1    DECOMPOSITION IMPOSSIBLE
C     TK        MAX DIFFERENCE BETWEEN ELEMENTS OF A AND OF THE PRODUCT
C               G*GTRANSPOSE
C     IRANK     THE RANK OF MATRIX G
C
C     SUBROUTINE REQUIRED
C          NONE
C
      DIMENSION A(IA,IA),G(IA,IA),B(20,20)
C
```

NUMERICAL SOLUTION OF $A^TS + SA + Q = 0$

Per Hagander

ABSTRACT

A survey of techniques to solve $A^TS + SA + Q = 0$ is presented, and nine algorithms are coded and tested on a batch of examples. Which algorithm to be recommended depends mainly on the order of the system.

1. INTRODUCTION

In recent time [2,4,5,6,9,13,14,19,20,21] great attention is drawn to the equation

$$A^TS + SA + Q = 0 \qquad (1)$$

solved for S with Q symmetric of order n x n and thus also the solution S.

This equation plays a central role in the theory of stability for linear continuous systems. It also arises in the pole assignment [11], in the sensitivity analysis [2d] and when evaluating loss functions in optimal control and covariance matrices in filtering and estimation for continuous systems.

The more general equation

$$A^TS + SA + Q_1 - SBQ_2^{-1}B^TS = 0 \qquad (2)$$

appearing in spectral factorization [1], estimation and optimal control, has been solved by iteration of (1) [10].

Another generalization of (1) used in e.g. network theory

$$A^TS + SB + Q = 0 \qquad (3)$$

is possible to solve by slight modifications of some of the methods indicated below.

The to (1) corresponding equation for discrete time systems

$$\phi^TS\phi + \tilde{Q} = S \qquad (4)$$

has somewhat different properties and use has been made of transformations between the two equations.

In many of the applications above it is necessary to solve (1) many times. The equation is therefore worth severe numerical interest.

It is my intention to give a survey of possible methods and to present algorithms. These algorithms are coded and tested on a set of examples and their accuracy and computing time are compared. All programming was done in FORTRAN for Univac 1108.

The algorithms are grouped in three sections:

1) Direct methods: Solution of a large system of linear equations by use of general methods.

2) Transformation methods: Use of the structure can be taken by similarity transformation of the A-matrix to some canonical form (Jordan or diagonal form, companion form, Schwarz' form). Some algorithms use the same technique without explicitly performing the transformations.

3) Iterative methods: The basic idea for these methods is that equation (1) is transformed to equation (4) either by sampling or by introducing a bilinear transformation.

The equation

$$S_{k+1} = \phi^T S_k \phi + \tilde{Q} \qquad (5)$$

is then iterated to stationarity by an accelerating formula.

2. COMPUTATIONAL AND PROGRAMMING ASPECTS

2.1. Direct methods

2.1.1. The equation (1) has n(n+1)/2 unknown variables. By organizing S and Q as vectors the system is rewritten as common linear equations

$$\mathcal{A}s = q \qquad (6)$$

and this can be solved by general methods, like Gauss elimination. $\mathcal{A}$ can be formed from A either by use of logical operations [5] or by use of an indexing matrix [4] or vector. The index vector form is found slightly more efficient.

If λ are the eigenvalues of A, then the eigenvalues of $\mathcal{A}$ are sums $\lambda_i+\lambda_j$. This implies that (6) certainly has a unique solution if A is stable. If A is unstable, $\mathcal{A}$ might be illconditioned or singular. The original equation (1) is however then also illconditioned or singular. The equation can be solved for different Q-matrices (q-vectors) with little extra effort without inversion of $\mathcal{A}$ [8]. This can for instance be valuable when improving a solution.

Algorithm 1 is utilizing these ideas. The programming is easily done. The main disadvantages are that the memory requirement is $(n(n+1)/2)^2$ cells, and that the number of multiplicative operations for large n is of the order $n^6/24$.

2.2 Transformation methods

2.2.1. By transforming A and A^T to Jordan form it is possible to express the solution S in the eigenvalues and eigenvectors of A and A^T [12]. This is greatly simplified when A is diagonalizable:

Theorem 1 [12]
Let U and V be the matrices that diagonalize A and A^T:

$A = U^{-1}DU \qquad A^T = V^{-1}DV \qquad D = \text{diag}\{\lambda_1, \ldots, \lambda_n\}$

and let $\hat{Q} = VQU^{-1}$ and $\hat{s}_{ij} = -q_{ij}/(\lambda_i+\lambda_j)$

Then $S = V^{-1}\hat{S}U$ (7)

This theorem is used in algorithm 2.1, thus requiring complex arithmetic. ~~can be used~~. The main drawback is however the eigenvalue eigenvector calculation for the nonsymmetric A. This is done in an up-to-date QR-algorithm with inverse iteration, but close eigenvalues lead to overwhelming problems. No advantage of symmetry can be taken. The approach is out of the question in other cases than when eigenvalues and eigenvectors are already obtained or wanted.

2.2.2. Eigenvectors can also be used in another way applicable even to the quadratic equation (2) [15,18].

Theorem 2 [18]
If $\begin{bmatrix} b_1 \\ \hline c_1 \end{bmatrix}, \ldots, \begin{bmatrix} b_n \\ \hline c_n \end{bmatrix}$

are the n eigenvectors corresponding to the eigenvalues with negative real part of the 2n x 2n matrix:

$$\begin{bmatrix} A & -BQ_2^{-1}B^T \\ -Q_1 & A^T \end{bmatrix} \qquad (8)$$

then the solution of (2) is

$$S = [c_1, \ldots, c_n][b_1, \ldots, b_n]^{-1} \qquad (9)$$

The computational effort is simplified by the observation:

Corollary.

Let $\begin{bmatrix} f_i \\ \hline g_i \end{bmatrix} = \begin{bmatrix} b_i \\ \hline c_i \end{bmatrix}$ for real eigenvalues λ_i and

$$\begin{bmatrix} f_i \\ \hline g_i \end{bmatrix} = \mathrm{Re} \begin{bmatrix} b_i \\ \hline c_i \end{bmatrix} + \mathrm{Im} \begin{bmatrix} b_i \\ \hline c_i \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} f_{i+1} \\ \hline g_{i+1} \end{bmatrix} = \mathrm{Re} \begin{bmatrix} b_i \\ \hline c_i \end{bmatrix} - \mathrm{Im} \begin{bmatrix} b_i \\ \hline c_i \end{bmatrix}$$

for pairs of complex eigenvalues λ_i, λ_{i+1}, $(\lambda_{i+1}=\bar{\lambda}_i)$,

then $S = [g_1, \ldots, g_n][f_1, \ldots, f_n]^{-1}$ (10)

Algorithm 3 is based on this corollary. Real arithmetic can be used but otherwise the remarks on algorithm 2 still hold. A new Q needs a recomputation of the whole eigenvalue problem. The method is only of theoretical interest for equation (1) but is reasonable for equation (2).

2.2.3. The companion form and its transformations lead to interesting algebraic manipulations and probably also to the fewest operations for large dimensions.

One method emanating from Nekolny and Benes [16] deals with the transfer function (G) of the system $S(A^T,B,C)$. The covariance of the output of the system for white noise input is

$$V = CSC^T \tag{11}$$

if S is the solution of (1) with

$$Q = BB^T \tag{12}$$

Åström [23] has described this for single input, single output systems and gives recursive formulas essentially using the Routh algorithm. These can be extended to the multivariable case and used for solution of (1).

This is done in algorithm 243 including decomposition of Q [8, p 114] and computation of G(s) for $S(A^T,B,I)$ by a Leverrier algorithm [7].

Full advantage of symmetry is taken, but different Q matrices are difficult to handle.

2.2.4. Other authors have used the companion form to obtain an explicit solution without performing the transformation.

Smith [21a] developed an expression in powers of A, and Müller [14] used the matrices A_i from the Leverrier algorithm and achieves a nice formula:

Theorem 3 [14]

Let

$$a_k = -\frac{1}{k} \operatorname{tr} AA_{k-1} \qquad a_o = 1,\ A_o = I$$

$$A_k = AA_{k-1} + a_k I \qquad k = 1, \ldots \qquad (13)$$

and let c be the solution of

$$Hc = \begin{bmatrix} 1/2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (14)$$

where H is the Hurwitz matrix, then

$$S = \sum_{j=0}^{n-1} c_{j+1} \sum_{i=0}^{2j} (-1)^i A^T{}_i Q A_{2j-i} \qquad (15)$$

Note that $A_i = 0$ for $i \geq n$, because of the Cayley-Hamilton theorem. From numerical point of view the formula is not a satisfactory solution.

Jameson ([9] and [20]) developed and tested a procedure closely related to these Leverrier computations:

Theorem 4 [9]

Let L and G be defined by

$$G = A^n - a_1 A^{n-1} + \ldots + (-1)^n a_n I \qquad (16)$$

$$L = C_n + a_1 C_{n-1} + \ldots + a_{n-1} C_1 \qquad (17)$$

where

$$C_1 = Q$$

$$C_k = A^T C_{k-1} + QA^{k-1}, \quad k = 2, \ldots, n \qquad (18)$$

Then

$$S = G^{-1} L^T \qquad (19)$$

Algorithm 254 solves (1) by this theorem. It is sensitive to round off errors, Jameson used triple precision accumulation in his tests, and the main part of the computation must be redone for new Q:s.

2.2.5. By using the Danilevskii algorithm [7], Molinari [13] reduced some of the difficulties with the companion form transformation.

Theorem 5 [13]
Let T transform A to the companion form $\hat{A}$.

$$TAT^{-1} = \hat{A}$$

and let

$$\hat{Q} = T^{-T} A T^{-1} \qquad \text{and} \qquad \hat{S} = T^{-T} S T^{-1}$$

Define the vector b by

$$b_i = \begin{cases} \frac{1}{2} \sum_{k=1}^{2i-1} (-1)^{k+1} q_{2i-k,k} & i=1,\ldots,\left[\frac{n+1}{2}\right] \\ \frac{1}{2} \sum_{k=1}^{2n+1-2i} (-1)^{k+1} q_{n+1-k,2i-n-1+k} & i=\left[\frac{n+1}{2}\right]+1,\ldots,n \end{cases} \tag{20}$$

and x as the solution of

$$Hx = b$$

with H as the Hurwitz matrix.

Then the first column of $\hat{S}$ is obtained by

$$\hat{s}_{i1} = (-1)^{i+1} x_i \tag{21}$$

and the other n-1 columns by the recursion for j=1,...,n-1:

$$\hat{s}_{ij+1} = -q_{ij} + p_j s_{i1} + p_i s_{j1} - s_{i+1j} \qquad i=1,\ldots,n-1$$

$$\hat{s}_{n,j+1} = -q_{nj} + p_j s_{n1} + p_n s_{j1} \tag{22}$$

Algorithm 265 is based on this work, which is an improvement of the foregoing. A general companion form might be used, not only the canonical forms corresponding to the transfer function, and this

decreases the computation necessary and increases the pivoting possibilities. New Q-matrices can be solved with reasonable effort and an improvement routine can be applied inside the companion form transformation.

2.2.6. The Schwarz and Routh [2a,2c,17,19] canonical forms have been used in order to formalize the above algebra. The transformations are usually done via the companion form, and show the same difficulties. The solution is simple only for diagonal Q and a few other special cases.

2.3 Iterative methods

2.3.1. The solution of (1) can for stable A be written as [3]

$$S = \int_0^\infty e^{A^T t} Q e^{At} dt \qquad (23)$$

Davidson and Man [6] integrated (23) by the simple Euler approximation and got

$$S_o = 0$$

$$S_{k+1} = \phi^T S_k \phi + \tilde{Q} \qquad (24)$$

where $\phi = \exp\{A \cdot h\}$ and $\tilde{Q} = h \cdot Q$, or accelerated

$$S_o = \tilde{Q}$$

$$S_{k+1} = (\phi^T)^{2^k} S_k \phi^{2^k} + S_k \qquad (25)$$

and with $|| S_{k+1} - S_k || < 10^{-6}$ as a possible stopping condition.

Algorithm 7 is used to test the above formula. The properties specified in [6] can be verified but one important fact is lacking. S_k does not converge to the correct solution of (1) if h is not chosen very small. The Euler approximation must be valid. On the other hand, the representation of ϕ will then contain less information and large round off errors will result. The iteration must be redone for new Q:s. Molinari [13] refers to 7 as the commonly preferred algorithm.

2.3.2. It is also possible to view (25) as the solution of

$$-\frac{dS}{dt} = A^T S + SA + Q \qquad (26)$$

which can be integrated either by Runge Kutta or other conventional methods or by using the linearity for a fundamental matrix approach.

$$\Sigma(t) = \exp\ t \cdot \begin{bmatrix} -A & 0 \\ Q & A^T \end{bmatrix} = \begin{bmatrix} (\Sigma_{22}{}^T)^{-1} & 0 \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \qquad (27)$$

$$S(t) = \left[\Sigma_{21}(t) + \Sigma_{22}(t)\ S(0)\ \Sigma_{22}{}^T(t)\right] \qquad (28)$$

Define $\phi = \Sigma_{22}{}^T(h)$ and $\tilde{Q} = \Sigma_{21}(h)\ \Sigma_{22}{}^T(h)$ then (28) can be rewritten as (24).

Algorithm 8 is based on the accelerated version (25). ϕ is computed by series expansion with 7 terms and automatic scaling. $\tilde{Q}$ is obtained by the iteration:

$$T_1 = Qh \qquad T_{k+1} = \frac{h}{k+1}\ \{(T_k A) + (T_k A)^T\} \qquad (29)$$

$$Q_1 = T_1 \qquad Q_{k+1} = Q_k + T_{k+1}$$

which is obtained by series expansion of $\tilde{Q} = \int_0^h e^{A^T t}\ Q\ e^{At}\ dt$.

No scaling is performed and the number of terms T_k is maximied to 35. The stopping condition for (29) is

$$||T_k||/||Q_k|| < 10^{-7} \qquad (30)$$

Both the $\tilde{Q}$ computation and the iteration must be redone for new Q. Better methods for the $\tilde{Q}$ computation might exist.

2.3.3. The two methods just described can be viewed as transforming equation (1) to equation (4) by

$$A \to \phi = \exp(Ah) \qquad (31)$$

Another possibility to go from (1) to (4) by introducing a one to one transformation mapping the left half plane on to the unit circle is the bilinear transform [2b,19,21b,22]

$$A \to \phi = -(A+I\cdot a)(A-Ia)^{-1} \quad . \qquad (32)$$

Q is then transformed to

$$\tilde{Q} = (A^T - aI)^{-1} Q (A-aI)^{-1}/2a$$

Algorithm 9 uses the acceleration formula to solve (4) obtained in this way [21b].

The convergence rate of 8 and 9 depends on the choice of a and h and on the spread of the eigenvalues of the matrix A. The eigenvalues of ϕ, z_i, are respectively:

$$z_i = e^{h\lambda_i} \qquad \text{and} \qquad z_i = \frac{a + \lambda_i}{a - \lambda_i}$$

The absolute largest eigenvalue, λ_{max}, times h is limiting for the convergence of the $\tilde{Q}$ computation in 8, and the absolute smallest one, λ_{min}, times h determines the convergence of (25). This implies that smaller h means better $\tilde{Q}$ but more iterations of (25).

In algorithm 9 the choice $a = \sqrt{\lambda_{min} \cdot \lambda_{max}}$ minimizes $\max_i |z_i|$, for real λ_i, thus leading to best convergence of (25).

The operations involved in 9 are simpler than in 8, and rough calculations with only real eigenvalues indicate that 9 manages a far larger spread in the A-eigenvalues, i.e. more ill-conditioned problems. A bad choice of a seems to be less critical than a bad choice of h.

3. THE NUMERICAL TEST

3.1. Test Examples

The algorithms are all tested for 17 different A matrices ranging in order from 2 to 10. The sample contains both stable and unstable A, as well as matrices with close eigenvalues and illconditioned matrices with a large eigenvalue spread. Some of the matrices were used in [2 b, 5, 13 or 20] but most of them are standard examples for matrix inversion and eigenvalue calculation. A few of the algorithms are also tested for matrices of order 20.

Six different easily generated Q matrices are used for each A matrix. Some additional testing is done with Q matrices designed to give simple integer valued S matrices.

The test batch is listed in Appendix 1.

3.2. Numerical Results

The results of the test are summarized in Table I. The accuracy is measured by comparing the solution with the solution of algorithm 1 in double precision.

For n = 20 the accuracy was evaluated as the error obtained when the computed S was substituted into (1). The accuracy of algorithm 1 in double precision is also estimated in this way.

When the two ways of measuring was compared for the low order systems no great difference was found. Call the first method "accuracy in S" and the second "accuracy in Q"! For illconditioned A matrices some divergence could happen in the test batch mainly so that the "accuracy in S" was one or two digits better than "the accuracy in Q". For the worst A matrices the difference could be even larger.

On the other hand for the specially designed Q matrices giving simple integer valued S matrices the "accuracy in S" was often worse than the "accuracy in Q".

When these differences occur they most often do it for all algorithms at the same time.

Table I - Result of the test

Order of the example	Algorithm									
	Direct methods		Eigenvalue methods		Companion form methods			Iterative methods		
	1 double prec.	1	2	3	4	5	6	7	8	9
2x2	2(17)	1(7)	13(7)	20(7)	2(7)	2(8)	1(6)	7(2)	10(6)	3(7)
3x3	5(16)	3(6)	35(6)	70(6)	6(7)	7(5)	3(5)	20(2)	25(5)	10(6)
4x4	12(16)	8(6)	80(5)	140(5)	13(5)	17(4)	5(4)	40(2)	50(4)	25(6)
6x6	62(16)	42(6)	200(4)	400(3)	50(1)	75(2)	14(1)	150(2)	170(4)	70(6)
8x8	245(14)	161(5)	350(2)	750(1)	155(2)	230(1)	27(0)	300(2)	300(4)	200(6)
9x9	440(17)	290(7)	325(6)	950(1)	240(6)	360(6)	-	260(2)	290(6)	140(7)
10x10	770(10)	500(2)	650(0)	1400(0)	340(1)	530(1)	50(-)	500(2)	500(3)	210(4)
20x20	-	-	4500(4)	-	6000(0)	8000(0)	340(-)	-	5000(5)	2000(5)

The first number means the execution time in milliseconds and the number in parenthesis gives the relative accuracy specified in significant digits.

For fixed matrix order the results show considerable variation depending on the actual A and Q. The figures in Table I represent an average for the test batch. Matrices giving failure exit are not included in the average.

It is not possible to draw general conclusions about new test examples. The only ninth order A matrix tested is for instance very simple, resulting in better accuracy than for the eighth order average.

Generally can be said, that equation (1) is difficult for very large systems, especially if A is illconditioned, that is mostly if A has a large spread in the eigenvalues. It also needs to be stated that different Q matrices can "hide" these difficulties to varying extent.

3.3. Discussion of the Results.

The described methods are tested as general purpose algorithms and as such there only remains two, algorithms 1 and 9.

The eigenvalue algorithms are neither accurate nor fast, and they often fail if two eigenvalues are close. Algorithm 3 is out of the question although the computing time could be almost halfed: Instead of computing all eigenvalues and eigenvectors of the $2n \times 2n$ matrix, it is sufficient to compute only the n eigenvectors corresponding to the stable eigenvalues. Algorithm 2 is probably the best of all methods if the eigenvalues and eigenvectors of A are known or useful in the future analysis.

Algorithm 5 has average properties, rather slow for small orders, proportional to n^4 for large orders, memory requirement proportional to n^2 and with bad accuracy for the difficult large order problems. Algorithm 1 is better for small orders and algorithm 9 for large orders.

Algorithm 4 has properties similar to those of algorithm 5. It has, however, some advantages. If Q is not full rank the calculations are considerably easier. Very favourable is also if G(s) is known or otherwise wanted. Moreover, if $V = CSC^T$ and not S is the quantity desired. No other method should be thought of, especially if C is just a vector.

Algorithm 4 needs positive semidefinite Q and stable A.

Algorithm 6 needs long code but small internal storage. Although there are pivoting possibilities the achieved accuracy is too bad. Double precision would make it possible to solve the equation (1) for higher orders, but even in double precision large systems are impossible to handle. The difficulties arise from the companion form representation. The execution time is by far the shortest, proportional to n^3. Noncyclic matrices, like the ninth order example, are not possible to transform to companion form, and failure exit of algorithm 6 results.

Algorithm 9 is always better than 7 and 8 both in accuracy and computing time. The figures presented are obtained for good values of the parameters a and h respectively. It is found that with a minimum of à priori knowledge of the system, both a and h can be estimated sufficiently well to give only a slight increase in execution time and round off errors. All the iterative methods give error indication for unstable A matrices.

Algorithm 1 is the best and easiest for small systems, and no free parameter a or h has to be chosen. For large systems, however, the execution time is proportional to n^6 and the internal storage proportional to n^4. It was not even possible to test for n = 20 on the big Univac 1108 machine with more than 40k words available memory.

4. RECOMMENDATIONS

Simpliest and best method for small orders is the direct solution 1. For large orders, say more than six or seven, other methods superseed it, for instance the iteration method 9. The fastest algorithm is Molinari's (6), which, however, is too sensitive to round off errors.

5. REFERENCES

[1] Anderson, B.D.O.: Solution of the Spectral Factorization Problem. IEEE Trans AC-12 pp 410-4 (Aug 67).

[2a] Barnett, S. and Storey, C.: The Liapunov Matrix Equation and Schwarz' Form. IEEE Trans AC-12 pp 117-8 (Feb 67).

[2b] Barnett, S. and Storey, C.: Remarks on Numerical Solution of the Liapunov Matrix Equation. Electronics Letters Vol 3 pp 417-8 (Sept 67).

[2c] Barnett, S. and Storey, C.: Some Applications of the Lyapunov Matrix Equation. J. Inst.Maths.Applic.s. (1968) pp 33-42.

[2d] Barnett, S. and Storey, C.: Insensitivity of Optimal Linear Control Systems to Persistent Changes in Parameters. Inst.J. Control Vol 4 pp 179-184 (Aug 66).

[3] Bellman, R.: Introduction to Matrix Analysis. McGraw-Hill 1960.

[4] Bingulac, S.P.: An Alternate Approach to Expanding PA+A'P=-Q. IEEE Trans AC-15 No 1 pp 135-7 (Feb 70).

[5] Chen, C.F., and Shieh, L.S.: A note on Expanding $PA+A^TP=-Q$. IEEE Trans AC-13 pp 122-3 (Feb 68).

[6] Davison, E.J., and Man[illegible] F.T.: The Numerical Solution of A'Q+QA=-C. IEEE Trans AC-13 pp 448-9 (Aug 68).

[7] Faddeeva, V.N.: Computational Methods of Linear Algebra. Dover 1959.

[8] Forsythe, G. and Moler, C.B.: Computer Solution of Linear Algebraic Systems. Prentice Hall, 1967.

[9] Jameson, A.: Solution of the Equation AX+XB=C by Inversion of an MxM or NxN Matrix. SIAM J.Appl.Math. Vol 13 pp 1020-3 (sept 68).

[10] Kleinman, D.L.: Solution of Linear Regulator Problem with Infinite Terminal Time. IEEE Trans AC-13, pp 114-5 (Feb 68)

[11] Luenberger, D.G.: Observers for Multivariable Systems. IEEE Trans AC-11 p 190-7 (April 60).

[12] Ma, E.C.: A Finite Series Solution of the Matrix Equation, AX-XB=C. SIAM J.Appl.Math. vol 14 pp 490-5 (May 66).

[13] Molinari, B.P.: Algebraic Solution of Matrix Linear Equations in Control Theory. Proc. IEE vol 116 pp 1748-1754 (Oct 69).

[14] Müller P.: Die Berechnung von Ljapunov-Funktionen und von quadratischen Regelflächen für Lineare, stetige, zeitinvariante Mehrgrössensysteme . Regelungstechnik 17, pp 341-5 (Aug 69).

[15] Mårtensson, K.: On the Riccati Equation. Thesis for the degree of Teknologie Licentiat. Report 7002 Lund Institute of Technology, Division of Automatic Control (April 70) (Accepted for publication in Information Sciences).

[16] Nekolny, J. and Benes, J.: Simultanous Control of Stability and Quality of Adjustment Application in Statistical Dynamics. Proc. of the 1:st IFAC Congress Moscow 1960, Vol 2 pp 734-744. (Butterworths 1961).

[17] Parks, P.C.: A new Proof of the Hurwitz Stability Criterion by the Second Method of Lyapunov with Applications to Optimum Transfer Function. Proc. JACC 1963 pp 471-8.

[18] Potter, J.E.: Matrix Quadratic Solutions. SIAM J.Appl.Math. Vol 14 pp 496-551 (May 66).

[19] Power, H.M.: Canonical Form for the Matrices of Linear Discrete Systems. Proc. IEE Vol 116, pp 1245-52 (July 69).

[20] Rothschild, D. and Jameson, A.: Comparison of four Numerical Algorithms for solving the Lyapunov Matrix Equation. Int.J.Control Vol 11 pp 181-198 (Feb 70).

[21a] Smith, R.A.: Matrix Calculations for Lyapunov Quadratic Forms. J.diff. Equations vol 2 pp 208-17 (April 66).

[21b] Smith, R.A.: Matrix Equation $XA+BX=C$. SIAM J.Appl.Math. vol 16 pp 198-201 (1968).

[22] Taussky,O.: Matrices C with $C^n \to 0$. J. Algebra vol 1 pp 5-10 (1964).

[23] Åström, K.J.: Introduction to Stochastic Control Theory. Academic Press 1970.

6. APPENDIX:

The A matrices of the test batch were:
(eigenvalues below)

2x2

$$\begin{bmatrix} -3 & 0 \\ 0 & -2 \end{bmatrix}, \quad \begin{bmatrix} -2 & -3 \\ -5 & -10 \end{bmatrix}, \quad \begin{bmatrix} -1 & +2 \\ 0 & -2 \end{bmatrix}$$

(-3,-2) (-11.56,-0.44) (-2,-1)

3x3

$$\begin{bmatrix} -0.1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -2.26 & -0.2 \end{bmatrix}, \quad \begin{bmatrix} -1 & 0 & -3 \\ -3 & -3 & 4 \\ 0 & 0 & -2 \end{bmatrix}, \quad \begin{bmatrix} -20 & 10 & 10 \\ -18 & 17 & 22 \\ 13 & -13 & -17 \end{bmatrix}$$

(-0.1±1.5i,-0.01) (-3,-2,-1) (-1,-0.5±0.87i)

4x4

$$\begin{bmatrix} -10 & -7 & -8 & -7 \\ -7 & -5 & -6 & -5 \\ -8 & -6 & -10 & -9 \\ -7 & -5 & -9 & -10 \end{bmatrix} \quad \begin{bmatrix} -8 & 1 & 0 & 0 \\ -19 & 0 & 1 & 0 \\ -22 & 0 & 0 & 1 \\ -10 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} -4 & 5 & 0 & -3 \\ 0 & -4 & 3 & 5 \\ -5 & 3 & -4 & 0 \\ -3 & 0 & -5 & -4 \end{bmatrix}$$

(-30.3,-3.86,-0.84,-0.010) (-5,-1±i,-1) (-12,-2,-1±5i)

6x6

$$\begin{bmatrix} -0.1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 10 & 10 & 5 \\ 0 & 3 & 0 & -3 & 1 & 0 \\ 7 & 2 & 0 & 0 & -10 & 0 \\ 32 & 15 & 0 & 0 & 100 & -50 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & -1 \\ -1 & 1 & 1 & 0 & 0 & 1 \\ 1 & -1 & 1 & 1 & 0 & -1 \\ -1 & 1 & -1 & 1 & +1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

(-50,-10,-3,-2,-1,-0.1) (-3.03,+1.31±1.20i,+1.47±0.35i, +1.48)

8x8

$$\begin{bmatrix} -1 & -5 & 3 & 7 & -9 & -2 & -8 & 0 \\ 20 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 1 & -3 & -100 & -0.3 \\ 0 & 0 & 0 & -5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & -0.1 & 0 & 0 \\ 0 & 0 & 0 & -2 & 00 & 0 & -0.01 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 \end{bmatrix},$$

(-10, -5, -2, -1±i, -0.5, -0.1, -0.01)

$$\begin{bmatrix} -0.021516 & -0.021516 & 0 & 0 & -0.001138 & 0.662 & 0 & 0 \\ 0.132 & -0.1469 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.4241 & 0 & 0 & 0 & 0 & 0.5561 \\ 0 & 0 & -0.516 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2.7073 & 0 & -0.4995 & 0 & 0 & 0 \\ 0 & 0 & 0.5166 & 0 & 0 & -1.834 & 0.1207 & 0 \\ 0 & 0 & 0.516 & 0 & 0 & -1.332 & 0 & 0 \\ 0 & 0 & -0.2346 & 0.0909 & 0 & 0 & 0 & -0.4546 \end{bmatrix}$$

(-1.7, -0.50, -0.39±0.30i, -0.12, -0.11, -0.09, -0.05)

8x8

$$\begin{bmatrix} -0.15365 & 0.0040173 & 0.17786 & -0.99009 & 0.075158 & 0 & 0 & 0 \\ 1.2482 & -2.8543 & 0 & 1.4324 & 0.72689 & 4.0383 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.56788 & -0.27685 & 0 & -0.28366 & -2.0496 & -0.13886 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ 0 & 00 & 0 & 0 & 0 & -20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -3 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(-20, -10, -2.79, -2, -1, -0.27±0.89i, +0.0336)

9x9

$$\begin{pmatrix} -1.6667 & 0 & 1.3333 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.16667 & 0 & -1.3333 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4.667 & 0 & 1.3333 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2.5 & -6 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1667 & 0 & -4.3333 \end{pmatrix}$$

(-6, -5, -4, -3, -3, -3, -2, -2, -1)

10x10

$$\begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & -3 & -3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & -3 & -4 & -4 & 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & -3 & -4 & -5 & -5 & 0 & 0 & 0 & 0 \\ -1 & -2 & -3 & -4 & -5 & -6 & -6 & 0 & 0 & 0 \\ -1 & -2 & -3 & -4 & -5 & -6 & -7 & -7 & 0 & 0 \\ -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & -8 & 0 \\ -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & -9 & -9 \\ -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & -9 & -10 \end{pmatrix},$$

(-25.58, -14.76, -8.05, -3.89, -1.62, -0.62, -0.26, -0.12, -0.065, -0.042)

$$\begin{pmatrix} -1 & -1 & -1 & 2 & -1 & 1 & -2 & 2 & -4 & 3 \\ 1 & -2 & -3 & 4 & -2 & 2 & -4 & 4 & -8 & 6 \\ 1 & 0 & -5 & 5 & -3 & 3 & -6 & 6 & -12 & 9 \\ 1 & 0 & -3 & 4 & -4 & 4 & -8 & 8 & -16 & 12 \\ 1 & 0 & -3 & 6 & -5 & 4 & -10 & 10 & -20 & 15 \\ 1 & 0 & -3 & 6 & -2 & 2 & -12 & 12 & -24 & 18 \\ 1 & 0 & -3 & 6 & -2 & 5 & -15 & 13 & -28 & 21 \\ 1 & 0 & -3 & 6 & -2 & 5 & -12 & 11 & -32 & 24 \\ 1 & 0 & -3 & 6 & -2 & 5 & -12 & 14 & -37 & 26 \\ 1 & 0 & -3 & 6 & -2 & 5 & -12 & 14 & -36 & 25 \end{pmatrix}$$

(-3, -3, -3, -3, -2, -2, -2, -2, -2, -1)

20x20

$a_{ij} = -2$ $\quad i = j$

$= 1$ $\quad |i-j| = 1$

$= 0$ $\quad$ otherwise

eigenvalues: $\quad \lambda_i = -2\left(1 - \cos \frac{\pi \cdot i}{n+1}\right)$

20x20

$a_{ij} = -1$ $\quad j = i+1$

$= -1.001$ $\quad j = i$

$= -(0.001)^{i-j+1}$ $\quad j < i$

$= 0$ $\quad$ otherwise

eigenvalues: $\quad \lambda_i = -1 \quad , i=1,\ldots,[n/2]$

$= -1 - 0.004 \cdot \cos^2 \frac{\pi(i - [n/2])}{n+2}, \quad i=[n/2]+1,\ldots,n$

The six Q matrices were generated by:

1) $q_{ij} = \begin{cases} 1 & i=j \\ 0 & i \neq j \end{cases}$ pos.def.

2) $q_{ij} = \begin{cases} 2 & i=j \\ -1 & |i-j| = 1 \\ 0 & \text{otherwise} \end{cases}$ pos.def.

3) $q_{ij} = \begin{cases} 1 & i=j \\ 0.2 & i \neq j \end{cases}$ pos.def.

4) $q_{ij} = 1$ all i,j pos.semi.def.

5) $q_{ij} = 0.999$ all i,j pos.semi.def.

6) $q_{ij} = 2*\max(i,j)-1$ indef.