SIAM REVIEW
Vol. 20, No. 4, October 1978

0036-1445/78/2004-0031$01.00/0

NINETEEN DUBIOUS WAYS TO COMPUTE THE EXPONENTIAL OF A MATRIX*

CLEVE MOLER† AND CHARLES VAN LOAN‡

Abstract. In principle, the exponential of a matrix could be computed in many ways. Methods involving approximation theory, differential equations, the matrix eigenvalues, and the matrix characteristic polynomial have been proposed. In practice, consideration of computational stability and efficiency indicates that some of the methods are preferable to others, but that none are completely satisfactory.

1. Introduction. Mathematical models of many physical, biological, and economic processes involve systems of linear, constant coefficient ordinary differential equations

$$\dot{x}(t) = Ax(t).$$

Here A is a given, fixed, real or complex n-by-n matrix. A solution vector $x(t)$ is sought which satisfies an initial condition

$$x(0) = x_0.$$

In control theory, A is known as the state companion matrix and $x(t)$ is the system response.

In principle, the solution is given by $x(t) = e^{tA}x_0$ where e^{tA} can be formally defined by the convergent power series

$$e^{tA} = I + tA + \frac{t^2A^2}{2!} + \cdots.$$

The effective computation of this matrix function is the main topic of this survey.

We will primarily be concerned with matrices whose order n is less than a few hundred, so that all the elements can be stored in the main memory of a contemporary computer. Our discussion will be less germane to the type of large, sparse matrices which occur in the method of lines for partial differential equations.

Dozens of methods for computing e^{tA} can be obtained from more or less classical results in analysis, approximation theory, and matrix theory. Some of the methods have been proposed as specific algorithms, while others are based on less constructive characterizations. Our bibliography concentrates on recent papers with strong algorithmic content, although we have included a fair number of references which possess historical or theoretical interest.

In this survey we try to describe all the methods that appear to be practical, classify them into five broad categories, and assess their relative effectiveness. Actually, each of the "methods" when completely implemented might lead to many different computer programs which differ in various details. Moreover, these details might have more influence on the actual performance than our gross assessment indicates. Thus, our comments may not directly apply to particular subroutines.

In assessing the effectiveness of various algorithms we will be concerned with the following attributes, listed in decreasing order of importance: generality, reliability,

* Received by the editors July 23, 1976, and in revised form March 19, 1977.

† Department of Mathematics, University of New Mexico, Albuquerque, New Mexico 87106. This work was partially supported by NSF Grant MCS76-03052.

‡ Department of Computer Science, Cornell University, Ithaca, New York 14850. This work was partially supported by NSF Grant MCS76-08686.

stability, accuracy, efficiency, storage requirements, ease of use, and simplicity. We would consider an algorithm completely satisfactory if it could be used as the basis for a general purpose subroutine which meets the standards of quality software now available for linear algebraic equations, matrix eigenvalues, and initial value problems for nonlinear ordinary differential equations. By these standards, none of the algorithms we know of are completely satisfactory, although some are much better than others.

Generality means that the method is applicable to wide classes of matrices. For example, a method which works only on matrices with distinct eigenvalues will not be highly regarded.

When defining terms like reliability, stability and accuracy, it is important to distinguish between the inherent sensitivity of the underlying problem and the error properties of a particular algorithm for solving that problem. Trying to find the inverse of a nearly singular matrix, for example, is an inherently sensitive problem. Such problems are said to be poorly posed or badly conditioned. No algorithm working with finite precision arithmetic can be expected to obtain a computed inverse that is not contaminated by large errors.

An algorithm is said to be reliable if it gives some warning whenever it introduces excessive errors. For example, Gaussian elimination without some form of pivoting is an unreliable algorithm for inverting a matrix. Roundoff errors can be magnified by large multipliers to the point where they can make the computed result completely erroneous, but there is no indication of the difficulty.

An algorithm is stable if it does not introduce any more sensitivity to perturbation than is inherent in the underlying problem. A stable algorithm produces an answer which is exact for a problem close to the given one. A method can be stable and still not produce accurate results if small changes in the data cause large changes in the answer. A method can be unstable and still be reliable if the instability can be detected. For example, Gaussian elimination with either partial or complete pivoting must be regarded as a mildly unstable algorithm because there is a possibility that the matrix elements will grow during the elimination and the resulting roundoff errors will not be small when compared with the original data. In practice, however, such growth is rare and can be detected.

The accuracy of an algorithm refers primarily to the error introduced by truncating infinite series or terminating iterations. It is one component, but not the only component, of the accuracy of the computed answer. Often, using more computer time will increase accuracy provided the method is stable. For example, the accuracy of an iterative method for solving a system of equations can be controlled by changing the number of iterations.

Efficiency is measured by the amount of computer time required to solve a particular problem. There are several problems to distinguish. For example, computing only e^A is different from computing e^{tA} for several values of t. Methods which use some decomposition of A (independent of t) might be more efficient for the second problem. Other methods may be more efficient for computing $e^{tA}x_0$ for one or several values of t. We are primarily concerned with the order of magnitude of the work involved. In matrix eigenvalue computation, for example, a method which required $O(n^4)$ time would be considered grossly inefficient because the usual methods require only $O(n^3)$.

In estimating the time required by matrix computations it is traditional to estimate the number of multiplications and then employ some factor to account for the other operations. We suggest making this slightly more precise by defining a basic

floating point operation, or "flop", to be the time required for a particular computer system to execute the FORTRAN statement

$$A(I, J) = A(I, J) + T^*A(I, K).$$

This involves one floating point multiplication, one floating point addition, a few subscript and index calculations, and a few storage references. We can then say, for example, that Gaussian elimination requires $n^3/3$ flops to solve an n-by-n linear system $Ax = b$.

The eigenvalues of A play a fundamental role in the study of e^{tA} even though they may not be involved in a specific algorithm. For example, if all the eigenvalues lie in the open left half plane, then $e^{tA} \to 0$ as $t \to \infty$. This property is often called "stability" but we will reserve the use of this term for describing numerical properties of algorithms.

Several particular classes of matrices lead to special algorithms. If A is symmetric, then methods based on eigenvalue decompositions are particularly effective. If the original problem involves a single, nth order differential equation which has been rewritten as a system of first order equations in the standard way, then A is a companion matrix and other special algorithms are appropriate.

The inherent difficulty of finding effective algorithms for the matrix exponential is based in part on the following dilemma. Attempts to exploit the special properties of the differential equation lead naturally to the eigenvalues λ_i and eigenvectors v_i of A and to the representation

$$x(t) = \sum_{i=1}^{n} \alpha_i e^{\lambda_i t} v_i.$$

However, it is not always possible to express $x(t)$ in this way. If there are confluent eigenvalues, then the coefficients α_i in the linear combination may have to be polynomials in t. In practical computation with inexact data and inexact arithmetic, the gray area where the eigenvalues are nearly confluent leads to loss of accuracy. On the other hand, algorithms which avoid use of the eigenvalues tend to require considerably more computer time for any particular problem. They may also be adversely effected by roundoff error in problems where the matrix tA has large elements.

These difficulties can be illustrated by a simple 2-by-2 example,

$$A = \begin{bmatrix} \lambda & \alpha \\ 0 & \mu \end{bmatrix}.$$

The exponential of this matrix is

$$e^{tA} = \begin{bmatrix} e^{\lambda t} & \alpha \dfrac{e^{\lambda t} - e^{\mu t}}{\lambda - \mu} \\ 0 & e^{\mu t} \end{bmatrix}.$$

Of course, when $\lambda = \mu$, this representation must be replaced by

$$e^{tA} = \begin{bmatrix} e^{\lambda t} & \alpha t e^{\lambda t} \\ 0 & e^{\lambda t} \end{bmatrix}.$$

There is no serious difficulty when λ and μ are exactly equal, or even when their difference can be considered negligible. The degeneracy can be detected and the resulting special form of the solution invoked. The difficulty comes when $\lambda - \mu$ is small

but not negligible. Then, if the divided difference

$$\frac{e^{\lambda t} - e^{\mu t}}{\lambda - \mu}$$

is computed in the most obvious way, a result with a large relative error is produced. When multiplied by α, the final computed answer may be very inaccurate. Of course, for this example, the formula for the off-diagonal element can be written in other ways which are more stable. However, when the same type of difficulty occurs in nontriangular problems, or in problems that are larger than 2-by-2, its detection and cure is by no means easy.

The example also illustrates another property of e^{tA} which must be faced by any successful algorithm. As t increases, the elements of e^{tA} may grow before they decay. If λ and μ are both negative and α is fairly large, the graph in Fig. 1 is typical.

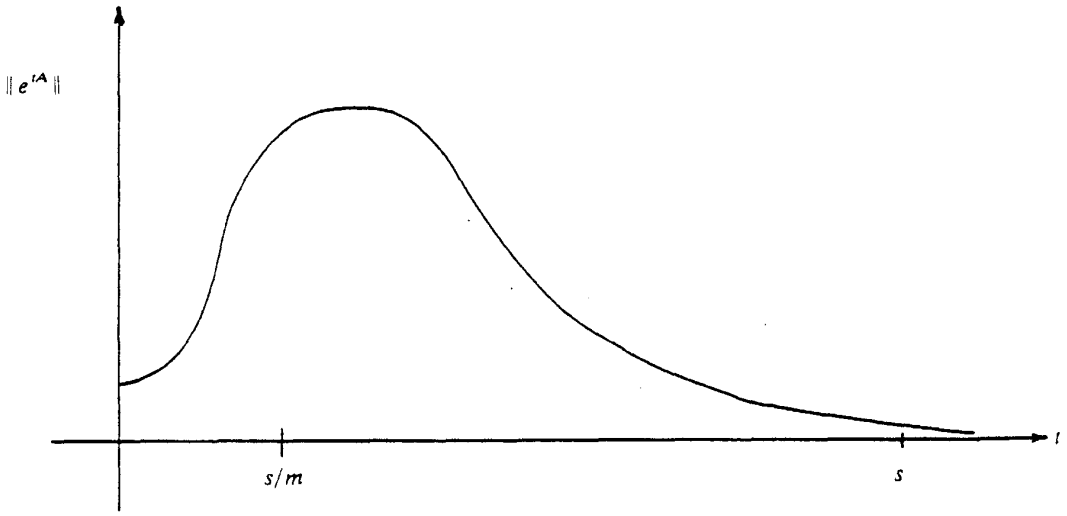

FIG. 1. *The "hump".*

Several algorithms make direct or indirect use of the identity

$$e^{sA} = (e^{sA/m})^m.$$

The difficulty occurs when s/m is under the hump but s is beyond it, for then

$$\|e^{sA}\| \ll \|e^{sA/m}\|^m.$$

Unfortunately, the roundoff errors in the mth power of a matrix, say B^m, are usually small relative to $\|B\|^m$ rather than $\|B^m\|$. Consequently, any algorithm which tries to pass over the hump by repeated multiplications is in difficulty.

Finally, the example illustrates the special nature of symmetric matrices. A is symmetric if and only if $\alpha = 0$, and then the difficulties with multiple eigenvalues and the hump both disappear. We will find later that multiple eigenvalue and hump problems do not exist when A is a normal matrix.

It is convenient to review some conventions and definitions at this time. Unless otherwise stated, all matrices are n-by-n. If $A = (a_{ij})$ we have the notions of transpose, $A^T = (a_{ji})$, and conjugate transpose, $A^* = (\overline{a_{ji}})$. The following types of matrices have

an important role to play:

$$\begin{aligned}
&A \text{ symmetric} \leftrightarrow A^T = A,\\
&A \text{ Hermitian} \leftrightarrow A^* = A,\\
&A \text{ normal} \leftrightarrow A^*A = AA^*,\\
&Q \text{ orthogonal} \leftrightarrow Q^TQ = I,\\
&Q \text{ unitary} \leftrightarrow Q^*Q = I,\\
&T \text{ triangular} \leftrightarrow t_{ij} = 0, \qquad i > j,\\
&D \text{ diagonal} \leftrightarrow d_{ij} = 0, \qquad i \neq j.
\end{aligned}$$

Because of the convenience of unitary invariance, we shall work exclusively with the 2-norm:

$$\|x\| = \left[\sum_{i=1}^{n} |x_i|^2\right]^{1/2}, \qquad \|A\| = \max_{\|x\|=1} \|Ax\|.$$

However, all our results apply with minor modification when other norms are used.

The condition of an invertible matrix A is denoted by cond (A) where

$$\text{cond}\,(A) = \|A\|\,\|A^{-1}\|.$$

Should A be singular, we adopt the convention that it has infinite condition. The commutator of two matrices B and C is $[B, C] = BC - CB$.

Two matrix decompositions are of importance. The Schur decomposition states that for any matrix A, there exists a unitary Q and a triangular T, such that

$$Q^*AQ = T.$$

If $T = (t_{ij})$, then the eigenvalues of A are $t_{11}, \cdots, t_{nn}$.

The Jordan canonical form decomposition states that there exists an invertible P such that

$$P^{-1}AP = J.$$

where J is a direct sum, $J = J_1 \oplus \cdots \oplus J_k$, of Jordan blocks

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & & & \vdots \\ & & & & 1 \\ 0 & 0 & 0 & \cdots & \lambda_i \end{bmatrix} \qquad (m_i\text{-by-}m_i).$$

The λ_i are eigenvalues of A. If any of the m_i are greater than 1, A is said to be defective. This means that A does not have a full set of n linearly independent eigenvectors. A is derogatory if there is more than one Jordan block associated with a given eigenvalue.

2. The sensitivity of the problem. It is important to know how sensitive a quantity is before its computation is attempted. For the problem under consideration we are interested in the relative perturbation

$$\phi(t) = \frac{\|e^{t(A+E)} - e^{tA}\|}{\|e^{tA}\|}.$$

In the following three theorems we summarize some upper bounds for $\phi(t)$ which are derived in Van Loan [32].

THEOREM 1. *If* $\alpha(A) = \max\{\mathrm{Re}(\lambda) | \lambda$ *an eigenvalue of* $A\}$ *and* $\mu(A) = \max\{\mu | \mu$ *an eigenvalue of* $(A^* + A)/2\}$, *then*

$$\phi(t) \leqq t\|E\| \exp[\mu(A) - \alpha(A) + \|E\|]t \qquad (t \leqq 0).$$

The scalar $\mu(A)$ is the "log norm" of A (associated with the 2-norm) and has many interesting properties [35]–[42]. In particular, $\mu(A) \geqq \alpha(A)$.

THEOREM 2. *If* $A = PJP^{-1}$ *is the Jordan decomposition of* A *and* m *is the dimension of the largest Jordan block in* J, *then*

$$\phi(t) \leqq t\|E\| M_J(t)^2 e^{M_J(t)\|E\|t} \qquad (t \geqq 0),$$

where

$$M_J(t) = m \,\mathrm{cond}\,(P) \max_{0 \leqq j \leqq m-1} t^j/j!.$$

THEOREM 3. *If* $A = Q(D+N)Q^*$ *is the Schur decomposition of* A *with* D *diagonal and* N *strictly upper triangular* $(n_{ij} = 0,\ i \geqq j)$, *then*

$$\phi(t) \leqq t\|E\| M_S(t)^2 e^{M_S(t)\|E\|t} \qquad (t \geqq 0),$$

where

$$M_S(t) = \sum_{k=0}^{n-1} (\|N\|t)^k/k!.$$

As a corollary to any of these theorems one can show that if A is normal, then

$$\phi(t) \leqq t\|E\| e^{\|E\|t}.$$

This shows that the perturbation bounds on $\phi(t)$ for normal matrices are as small as can be expected. Furthermore, when A is normal, $\|e^{sA}\| = \|e^{sA/m}\|^m$ for all positive integers m implying that the "hump" phenomenon does not exist. These observations lead us to conclude that the e^A problem is "well conditioned" when A is normal.

It is rather more difficult to characterize those A for which e^{tA} is very sensitive to changes in A. The bound in Theorem 2 suggests that this might be the case when A has a poorly conditioned eigensystem as measured by cond (P). This is related to a large $M_S(t)$ in Theorem 3 or a positive $\mu(A) - \alpha(A)$ in Theorem 1. It is unclear what the precise connection is between these situations and the hump phenomena we described in the Introduction.

Some progress can be made in understanding the sensitivity of e^{tA} by defining the "matrix exponential condition number" $\nu(A, t)$:

$$\nu(A, t) = \max_{\|E\|=1} \left\| \int_0^t e^{(t-s)A} E\, e^{sA}\, ds \right\| \frac{\|A\|}{\|e^{tA}\|}.$$

A discussion of $\nu(A, t)$ can be found in [32]. One can show that there exists a perturbation E such that

$$\phi(t) \cong \frac{\|E\|}{\|A\|} \nu(A, t).$$

This indicates that if $\nu(A, t)$ is large, small changes in A can induce relatively large changes in e^{tA}. It is easy to verify that

$$\nu(A, t) \geqq t\|A\|,$$

with equality if and only if A is normal. When A is not normal, $\nu(A, t)$ can grow like a high degree polynomial in t.

3. Series methods. The common theme of what we call series methods is the direct application to matrices of standard approximation techniques for the scalar function e^t. In these methods, neither the order of the matrix nor its eigenvalues play a direct role in the actual computations.

METHOD 1. TAYLOR SERIES. The definition

$$e^A = I + A + A^2/2! + \cdots$$

is, of course, the basis for an algorithm. If we momentarily ignore efficiency, we can simply sum the series until adding another term does not alter the numbers stored in the computer. That is, if

$$T_k(A) = \sum_{j=0}^{k} A^j/j!$$

and $\mathrm{fl}\,[T_k(A)]$ is the matrix of floating point numbers obtained by computing $T_k(A)$ in floating point arithmetic, then we find K so that $\mathrm{fl}\,[T_K(A)] = \mathrm{fl}\,[T_{K+1}(A)]$. We then take $T_K(A)$ as our approximation to e^A.

Such an algorithm is known to be unsatisfactory even in the scalar case [4] and our main reason for mentioning it is to set a clear lower bound on possible performance. To illustrate the most serious shortcoming, we implemented this algorithm on the IBM 370 using "short" arithmetic, which corresponds to a relative accuracy of $16^{-5} \cong 0.95 \cdot 10^{-6}$. We input

$$A = \begin{bmatrix} -49 & 24 \\ -64 & 31 \end{bmatrix}$$

and obtained the output

$$e^A \approx \begin{bmatrix} -22.25880 & -1.432766 \\ -61.49931 & -3.474280 \end{bmatrix}.$$

A total of $K = 59$ terms were required to obtain convergence. There are several ways of obtaining the correct e^A for this example. The simplest is to be told how the example was constructed in the first place. We have

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -17 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^{-1},$$

and so

$$e^A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} e^{-1} & 0 \\ 0 & e^{-17} \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^{-1},$$

which, to 6 decimal places is,

$$e^A \approx \begin{bmatrix} -0.735759 & 0.551819 \\ -1.471518 & 1.103638 \end{bmatrix}.$$

The computed approximation even has the wrong sign in two components.

Of course, this example was constructed to make the method look bad. But it is important to understand the source of the error. By looking at intermediate results in the calculation we find that the two matrices $A^{16}/16!$ and $A^{17}/17!$ have elements

between 10^6 and 10^7 in magnitude but of opposite signs. Because we are using a relative accuracy of only 10^{-5}, the elements of these intermediate results have absolute errors larger than the final result. So, we have an extreme example of "catastrophic cancellation" in floating point arithmetic. It should be emphasized that the difficulty is not the truncation of the series, but the truncation of the arithmetic. If we had used "long" arithmetic which does not require significantly more time but which involves 16 digits of accuracy, then we would have obtained a result accurate to about nine decimal places.

Concern over where to truncate the series is important if efficiency is being considered. The example above required 59 terms giving Method 1 low marks in this connection. Among several papers concerning the truncation error of Taylor series, the paper by Liou [52] is frequently cited. If δ is some prescribed error tolerance, Liou suggests choosing K large enough so that

$$\|T_K(A)-e^A\| \leqq \left(\frac{\|A\|^{K+1}}{(K+1)!}\right)\left(\frac{1}{1-\|A\|/(K+2)}\right) \leqq \delta.$$

Moreover, when e^{tA} is desired for several different values of t, say $t = 1, \cdots, m$, he suggests an error checking procedure which involves choosing L from the same inequality with A replaced by mA and then comparing $[T_K(A)]^m x_0$ with $T_L(mA)x_0$. In related papers Everling [50] has sharpened the truncation error bound implemented by Liou, and Bickhart [46] has considered relative instead of absolute error. Unfortunately, all these approaches ignore the effects of roundoff error and so must fail in actual computation with certain matrices.

METHOD 2. PADÉ APPROXIMATION. The (p, q) Padé approximation to e^A is defined by

$$R_{pq}(A) = [D_{pq}(A)]^{-1} N_{pq}(A),$$

where

$$N_{pq}(A) = \sum_{j=0}^{p} \frac{(p+q-j)!\,p!}{(p+q)!\,j!\,(p-j)!} A^j$$

and

$$D_{pq}(A) = \sum_{j=0}^{q} \frac{(p+q-j)!\,q!}{(p+q)!\,j!\,(q-j)!} (-A)^j.$$

Nonsingularity of $D_{pq}(A)$ is assured if p and q are large enough or if the eigenvalues of A are negative. Zakian [76] and Wragg and Davies [75] consider the advantages of various representations of these rational approximations (e.g. partial fraction, continued fraction) as well as the choice of p and q to obtain prescribed accuracy.

Again, roundoff error makes Padé approximations unreliable. For large q, $D_{qq}(A)$ approaches the series for $e^{-A/2}$ whereas $N_{qq}(A)$ tends to the series for $e^{A/2}$. Hence, cancellation error can prevent the accurate determination of these matrices. Similar comments apply to general (p, q) approximants. In addition to the cancellation problem, the denominator matrix $D_{pq}(A)$ may be very poorly conditioned with respect to inversion. This is particularly true when A has widely spread eigenvalues. To see this again consider the (q, q) Padé approximants. It is not hard to show that for large enough q, we have

$$\text{cond}\,[D_{qq}(A)] \simeq \text{cond}\,(e^{-A/2}) \geqq e^{(\alpha_1-\alpha_n)/2}$$

where $\alpha_1 \geqq \cdots \geqq \alpha_n$ are the real parts of the eigenvalues of A.

When the diagonal Padé approximants $R_{qq}(A)$ were computed for the same example used with the Taylor series and with the same single precision arithmetic, it was found that the most accurate was good to only three decimal places. This occurred with $q = 10$ and cond $[D_{qq}(A)]$ was greater than 10^4. All other values of q gave less accurate results.

Padé approximants can be used if $\|A\|$ is not too large. In this case, there are several reasons why the diagonal approximants ($p = q$) are preferred over the off diagonal approximants ($p \neq q$). Suppose $p < q$. About qn^3 flops are required to evaluate $R_{pq}(A)$, an approximation which has order $p+q$. However, the same amount of work is needed to compute $R_{qq}(A)$ and this approximation has order $2q > p+q$. A similar argument can be applied to the superdiagonal approximants ($p > q$).

There are other reasons for favoring the diagonal Padé approximants. If all the eigenvalues of A are in the left half plane, then the computed approximants with $p > q$ tend to have larger rounding errors due to cancellation while the computed approximants with $p < q$ tend to have larger rounding errors due to badly conditioned denominator matrices $D_{pq}(A)$.

There are certain applications where the determination of p and q is based on the behavior of

$$\lim_{t\to\infty} R_{pq}(tA).$$

If all the eigenvalues of A are in the open left half plane, then $e^{tA} \to 0$ as $t \to \infty$ and the same is true for $R_{pq}(tA)$ when $q > p$. On the other hand, the Padé approximants with $q < p$, including $q = 0$, which is the Taylor series, are unbounded for large t. The diagonal approximants are bounded as $t \to \infty$.

METHOD 3. SCALING AND SQUARING. The roundoff error difficulties and the computing costs of the Taylor and Padé approximants increases as $t\|A\|$ increases, or as the spread of the eigenvalues of A increases. Both of these difficulties can be controlled by exploiting a fundamental property unique to the exponential function:

$$e^A = (e^{A/m})^m.$$

The idea is to choose m to be a power of two for which $e^{A/m}$ can be reliably and efficiently computed, and then to form the matrix $(e^{A/m})^m$ by repeated squaring. One commonly used criterion for choosing m is to make it the smallest power of two for which $\|A\|/m \leqq 1$. With this restriction, $e^{A/m}$ can be satisfactorily computed by either Taylor or Padé approximants. When properly implemented, the resulting algorithm is one of the most effective we know.

This approach has been suggested by many authors and we will not try to attribute it to any one of them. Among those who have provided some error analysis or suggested some refinements are Ward [72], Kammler [97], Kallstrom [116], Scraton [67], and Shah [56], [57].

If the exponential of the scaled matrix $e^{A/2^j}$ is to be approximated by $R_{qq}(A/2^j)$, then we have two parameters, q and j, to choose. In Appendix 1 we show that if $\|A\| \leqq 2^{j-1}$ then

$$[R_{qq}(A/2^j)]^{2^j} = e^{A+E},$$

where

$$\frac{\|E\|}{\|A\|} \leqq 8\left[\frac{\|A\|}{2^j}\right]^{2q}\left(\frac{(q!)^2}{(2q)!(2q+1)!}\right).$$

This "inverse error analysis" can be used to determine q and j in a number of ways. For example, if ε is any error tolerance, we can choose among the many (q, j) pairs for which the above inequality implies

$$\frac{\|E\|}{\|A\|} \leqq \varepsilon.$$

Since $[R_{qq}(A/2^j)]^{2^j}$ requires about $(q+j+\frac{1}{3})n^3$ flops to evaluate, it is sensible to choose the pair for which $q+j$ is minimum. The table below specifies these "optimum" pairs for various values of ε and $\|A\|$. By way of comparison, we have included the corresponding optimum (k, j) pairs associated with the approximant $[T_k(A/2^j)]^{2^j}$. These pairs were determined from Corollary 1 in Appendix 1, and from the fact that about $(k+j-1)n^3$ flops are required to evaluate $[T_k(A/2^j)]^{2^j}$.

TABLE 1

Optimum scaling and squaring parameters with diagonal Padé and Taylor series approximation.

$\|A\|$ \ ε	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}
10^{-2}	(1, 0) (1, 0)	(1, 0) (2, 1)	(2, 0) (3, 1)	(3, 0) (4, 1)	(3, 0) (5, 1)
10^{-1}	(1, 0) (3, 0)	(2, 0) (4, 0)	(3, 0) (4, 2)	(4, 0) (4, 4)	(4, 0) (5, 4)
10^{0}	(2, 1) (5, 1)	(3, 1) (7, 1)	(4, 1) (6, 3)	(5, 1) (8, 3)	(6, 1) (7, 5)
10^{1}	(2, 5) (4, 5)	(3, 5) (6, 5)	(4, 5) (8, 5)	(5, 5) (7, 7)	(6, 5) (9, 7)
10^{2}	(2, 8) (4, 8)	(3, 8) (5, 9)	(4, 8) (7, 9)	(5, 8) (9, 9)	(6, 8) (10, 10)
10^{3}	(2, 11) (5, 11)	(3, 11) (7, 11)	(4, 11) (6, 13)	(5, 11) (8, 13)	(6, 11) (8, 14)

To read the table, for a given ε and $\|A\|$ the top ordered pair gives the optimum (q, j) associated with $[R_{qq}(A/2^j)]^{2^j}$ while the bottom ordered pair specifies the most efficient choice of (k, j) associated with $[T_k(A/2^j)]^{2^j}$.

On the basis of the table we find that Padé approximants are generally more efficient than Taylor approximants. When $\|A\|$ is small, the Padé approximant requires about one half as much work for the same accuracy. As $\|A\|$ grows, this advantage decreases because of the larger amount of scaling needed.

Relative error bounds can be derived from the above results. Noting from Appendix 1 that $AE = EA$, we have

$$\frac{\|[R_{qq}(A/2^j)]^{2^j} - e^A\|}{\|e^A\|} = \frac{\|e^A(e^E - I)\|}{\|e^A\|}$$

$$\leqq \|E\|\, e^{\|E\|} \leqq \varepsilon \|A\|\, e^{\varepsilon\|A\|}.$$

A similar bound can be derived for the Taylor approximants.

The analysis and our table does *not* take roundoff error into account, although this is the method's weakest point. In general, the computed square of a matrix R can be severely affected by arithmetic cancellation since the rounding errors are small when compared to $\|R\|^2$ but not necessarily small when compared to $\|R^2\|$. Such cancellation can only happen when cond (R) is large because $R^{-1}R^2 = R$ implies

$$\text{cond}\,(R) \geqq \frac{\|R\|^2}{\|R^2\|}.$$

The matrices which are repeatedly squared in this method can be badly conditioned. However, this does not necessarily imply that severe cancellation actually takes place. Moreover, it is possible that cancellation occurs only in problems which involve a large hump. We regard it as an open question to analyze the roundoff error of the repeated squaring of $e^{A/m}$ and to relate the analysis to a realistic assessment of the sensitivity of e^A.

In his implementation of scaling and squaring Ward [72] is aware of the possibility of cancellation. He computes an a posteriori bound for the error, including the effects of both truncation and roundoff. This is certainly preferable to no error estimate at all, but it is not completely satisfactory. A large error estimate could be the result of any of three distinct difficulties:

(i) The error estimate is a severe overestimate of the true error, which is actually small. The algorithm is stable but the estimate is too pessimistic.

(ii) The true error is large because of cancellation in going over the hump, but the problem is not sensitive. The algorithm is unstable and another algorithm might produce a more accurate answer.

(iii) The underlying problem is inherently sensitive. No algorithm can be expected to produce a more accurate result.

Unfortunately, it is currently very difficult to distinguish among these three situations.

METHOD 4. CHEBYSHEV RATIONAL APPROXIMATION. Let $c_{qq}(x)$ be the ratio of two polynomials each of degree q and consider $\max_{0 \leqq x < \infty} |c_{qq}(x) - e^{-x}|$. For various values of q, Cody, Meinardus, and Varga [62] have determined the coefficients of the particular c_{qq} which minimizes this maximum. Their results can be directly translated into bounds for $\|c_{qq}(A) - e^A\|$ when A is Hermitian with eigenvalues on the negative real axis. The authors are interested in such matrices because of an application to partial differential equations. Their approach is particularly effective for the sparse matrices which occur in such applications.

For non-Hermitian (non-normal) A, it is hard to determine how well $c_{qq}(A)$ approximates e^A. If A has an eigenvalue λ off the negative real axis, it is possible for $c_{qq}(\lambda)$ to be a poor approximation to e^λ. This would imply that $c_{qq}(A)$ is a poor approximation to e^A since

$$\|e^A - c_{qq}(A)\| \geqq |e^\lambda - c_{qq}(\lambda)|.$$

These remarks prompt us to emphasize an important facet about approximation of the matrix exponential, namely, there is more to approximating e^A than just approximating e^z at the eigenvalues of A. It is easy to illustrate this with Padé approximation. Suppose

$$A = \begin{bmatrix} 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Since all of the eigenvalues of A are zero, $R_{11}(z)$ is a perfect approximation to e^z at the eigenvalues. However,

$$R_{11}(A) = \begin{bmatrix} 1 & 6 & 18 & 54 \\ 0 & 1 & 6 & 18 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

whereas

$$e^A = \begin{bmatrix} 1 & 6 & 18 & 36 \\ 0 & 1 & 6 & 18 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and thus,

$$\|e^A - R_{11}(A)\| = 18.$$

These discrepancies arise from the fact that A is not normal. The example illustrates that non-normality exerts a subtle influence upon the methods of this section even though the eigensystem, per se, is not explicitly involved in any of the algorithms.

4. Ordinary differential equation methods. Since e^{tA} and $e^{tA}x_0$ are solutions to ordinary differential equations, it is natural to consider methods based on numerical integration. Very sophisticated and powerful methods for the numerical solution of general nonlinear differential equations have been developed in recent years. All worthwhile codes have automatic step size control and some of them automatically vary the order of approximation as well. Methods based on single step formulas, multistep formulas, and implicit multistep formulas each have certain advantages. When used to compute e^{tA} all these methods are easy to use and they require very little additional programming or other thought. The primary disadvantage is a relatively high cost in computer time.

The o.d.e. programs are designed to solve a single system

$$\dot{x} = f(x, t), \qquad x(0) = x_0,$$

and to obtain the solution at many values of t. With $f(x, t) = Ax$ the kth column of e^{tA} can be obtained by setting x_0 to the kth column of the identity matrix. All the methods involve a sequence of values $0 = t_0, t_1, \cdots, t_j = t$ with either fixed or variable step size $h_i = t_{i+1} - t_i$. They all produce vectors x_i which approximate $x(t_i)$.

METHOD 5. GENERAL PURPOSE O.D.E. SOLVER. Most computer center libraries contain programs for solving initial value problems in ordinary differential equations. Very few libraries contain programs that compute e^{tA}. Until the latter programs are more readily available, undoubtedly the easiest and, from the programmer's point of view, the quickest way to compute a matrix exponential is to call upon a general purpose o.d.e. solver. This is obviously an expensive luxury since the o.d.e. routine does not take advantage of the linear, constant coefficient nature of our special problem.

We have run a very small experiment in which we have used three recently developed o.d.e. solvers to compute the exponentials of about a dozen matrices and have measured the amount of work required. The programs are:

(1) RKF45. Written by Shampine and Watts [108], this program uses the Fehlberg formulas of the Runge–Kutta type. Six function evaluations are required per

step. The resulting formula is fifth order with automatic step size control. (See also [4].)

(2) DE/STEP. Written by Shampine and Gordon [107], this program uses variable order, variable step Adams predictor-corrector formulas. Two function evaluations are required per step.

(3) IMPSUB. Written by Starner [109], this program is a modification of Gear's DIFSUB [106] and is based on implicit backward differentiation formulas intended for stiff differential equations. Starner's modifications add the ability to solve "infinitely stiff" problems in which the derivatives of some of the variables may be missing. Two function evaluations are usually required per step but three or four may occasionally be used.

For RKF45 the output points are primarily determined by the step size selection in the program. For the other two routines, the output is produced at user specified points by interpolation. For an n-by-n matrix A, the cost of one function evaluation is a matrix-vector multiplication or n^2 flops. The number of evaluations is determined by the length of the integration interval and the accuracy requested.

The relative performance of the three programs depends fairly strongly on the particular matrix. RKF45 often requires the most function evaluations, especially when high accuracy is sought, because its order is fixed. But it may well require the least actual computer time at modest accuracies because of its low overhead. DE/STEP indicates when it thinks a problem is stiff. If it doesn't give this indication, it usually requires the fewest function evaluations. If it does, IMPSUB may require fewer.

The following table gives the results for one particular matrix which we arbitrarily declare to be a "typical" nonstiff problem. The matrix is of order 3, with eigenvalues $\lambda = 3, 3, 6$; the matrix is defective. We used three different local error tolerances and integrated over $[0, 1]$. The average number of function evaluations for the three starting vectors is given in the table. These can be regarded as typical coefficients of n^2 for the single vector problem or of n^3 for the full matrix exponential; IBM 370 long arithmetic was used.

TABLE 2
Work as a function of subroutine and local error tolerance.

	10^{-6}	10^{-9}	10^{-12}
RKF45	217	832	3268
DE/STEP	118	160	211
IMPSUB	173	202	1510

Although people concerned with the competition between various o.d.e. solvers might be interested in the details of this table, we caution that it is the result of only one experiment. Our main reason for presenting it is to support our contention that the use of any such routine must be regarded as very inefficient. The scaling and squaring method of § 3 and some of the matrix decomposition methods of § 6 require on the order of 10 to 20 n^3 flops and they obtain higher accuracies than those obtained with 200 n^3 or more flops for the o.d.e. solvers.

This excessive cost is due to the fact that the programs are not taking advantage of the linear, constant coefficient nature of the differential equation. They must repeatedly call for the multiplication of various vectors by the matrix A because, as far as they know, the matrix may have changed since the last multiplication.

We now consider the various methods which result from specializing general o.d.e. methods to handle our specific problem.

METHOD 6. SINGLE STEP O.D.E. METHODS. Two of the classical techniques for the solution of differential equations are the fourth order Taylor and Runge–Kutta methods with fixed step size. For our particular equation they become

$$x_{j+1} = \left(I + hA + \cdots + \frac{h^4}{4!}A^4\right)x_j = T_4(hA)x_j$$

and

$$x_{j+1} = x_j + \tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4,$$

where $k_1 = hAx_j$, $k_2 = hA(x_j + \frac{1}{2}k_1)$, $k_3 = hA(x_j + \frac{1}{2}k_2)$, and $k_4 = hA(x_j + k_3)$. A little manipulation reveals that in this case, the two methods would produce identical results were it not for roundoff error. As long as the step size is fixed, the matrix $T_4(hA)$ need be computed just once and then x_{j+1} can be obtained from x_j with just one matrix-vector multiplication. The standard Runge–Kutta method would require 4 such multiplications per step.

Let us consider $x(t)$ for one particular value of t, say $t = 1$. If $h = 1/m$, then

$$x(1) = x(mh) \simeq x_m = [T_4(hA)]^m x_0.$$

Consequently, there is a close connection between this method and Method 3 which involved scaling and squaring [54], [60]. The scaled matrix is hA and its exponential is approximated by $T_4(hA)$. However, even if m is a power of 2, $[T_4(hA)]^m$ is usually not obtained by repeated squaring. The methods have roughly the same roundoff error properties and so there seem to be no important advantages for Runge–Kutta with fixed step size.

Let us now consider the possibility of varying the step size. A simple algorithm might be based on a variable step Taylor method. In such a method, two approximations to x_{j+1} would be computed and their difference used to choose the step size. Specifically, let ε be some prescribed local relative error tolerance and define x_{j+1} and x^*_{j+1} by

$$x_{j+1} = T_5(h_jA)x_j,$$
$$x^*_{j+1} = T_4(h_jA)x_j.$$

One way of determining h_j is to require

$$\|x_{j+1} - x^*_{j+1}\| \simeq \varepsilon \|x_j\|.$$

Notice that we are using a 5th order formula to compute the approximation, and a 4th order formula to control step size.

At first glance, this method appears to be considerably less efficient than one with fixed step size because the matrices $T_4(h_jA)$ and $T_5(h_jA)$ cannot be precomputed. Each step requires $5n^2$ flops. However, in those problems which involve large "humps" as described in § 1, a smaller step is needed at the beginning of the computation than at the end. If the step size changes by a factor of more than 5, the variable step method will require less work.

The method does provide some insight into the costs of more sophisticated integrators. Since

$$x_{j+1} - x_{j+1}^* = \frac{h_j A^5}{5!} x_j,$$

we see that the required step size is given approximately by

$$h_j \approx \left[\frac{5!\,\varepsilon}{\|A^5\|}\right]^{1/5}.$$

The work required to integrate over some fixed interval is proportional to the inverse of the average step size. So, if we decrease the tolerance ε from, say 10^{-6} to 10^{-9}, then the work is increased by a factor of $(10^3)^{1/5}$ which is about 4. This is typical of any 5th order error estimate—asking for 3 more figures roughly quadruples the work.

METHOD 7. MULTISTEP O.D.E. SOLVER. As far as we know, the possibility of specializing multistep methods, such as those based on the Adams formulas, to linear, constant coefficient problems has not been explored in detail. Such a method would not be equivalent to scaling and squaring because the approximate solution at a given time is defined in terms of approximate solutions at several previous times. The actual algorithm would depend upon how the starting vectors are obtained, and how the step size and order are determined. It is conceivable that such an algorithm might be effective, particularly for problems which involve a single vector, output at many values of t, large n, and a hump.

The problems associated with roundoff error have not been of as much concern to designers of differential equation solvers as they have been to designers of matrix algebra algorithms since the accuracy requested of o.d.e. solvers is typically less than full machine precision. We do not know what effect rounding errors would have in a problem with a large hump.

5. Polynomial methods. Let the characteristic polynomial of A be

$$c(z) = \det(zI - A) = z^n - \sum_{k=0}^{n-1} c_k z^k.$$

From the Cayley–Hamilton theorem $c(A) = 0$ and hence

$$A^n = c_0 I + c_1 A + \cdots + c_{n-1} A^{n-1}.$$

It follows that any power of A can be expressed in terms of $I, A, \cdots, A^{n-1}$:

$$A^k = \sum_{j=0}^{n-1} \beta_{kj} A^j.$$

This implies that e^{tA} is a polynomial in A with analytic coefficients in t:

$$e^{tA} = \sum_{k=0}^{\infty} \frac{t^k A^k}{k!} = \sum_{k=0}^{\infty} \frac{t^k}{k!} \left[\sum_{j=0}^{n-1} \beta_{kj} A^j\right]$$

$$= \sum_{j=0}^{n-1} \left[\sum_{k=0}^{\infty} \beta_{kj} \frac{t^k}{k!}\right] A^j$$

$$\equiv \sum_{j=0}^{n-1} \alpha_j(t) A^j.$$

The methods of this section involve this kind of exploitation of the characteristic polynomial.

METHOD 8. CAYLEY–HAMILTON. Once the characteristic polynomial is known, the coefficients β_{kj} which define the analytic functions $\alpha_j(t)=\sum \beta_{kj}t^k/k!$ can be generated as follows:

$$\beta_{kj}=\begin{cases}\delta_{kj} & (k<n)\\ c_j & (k=n)\\ c_0\beta_{k-1,n-1} & (k>n,\ j=0)\\ c_j\beta_{k-1,n-1}+\beta_{k-1,j-1} & (k>n,\ j>0).\end{cases}$$

One difficulty is that these recursive formulas for the β_{kj} are very prone to roundoff error. This can be seen in the 1-by-1 case. If $A=(\alpha)$ then $\beta_{k0}=\alpha^k$ and $\alpha_0(t)=\sum(\alpha t)^k/k!$ is simply the Taylor series for $e^{\alpha t}$. Thus, our criticisms of Method 1 apply. In fact, if $\alpha t=-6$, no partial sum of the series for $e^{\alpha t}$ will have any significant digits when IBM 370 short arithmetic is used.

Another difficulty is the requirement that the characteristic polynomial must be known. If $\lambda_1,\cdots,\lambda_n$ are the eigenvalues of A, then $c(z)$ could be computed from $c(z)=\prod_1^n(z-\lambda_i)$. Although the eigenvalues could be stably computed, it is unclear whether the resulting c_i would be acceptable. Other methods for computing $c(z)$ are discussed in Wilkinson [14]. It turns out that methods based upon repeated powers of A and methods based upon formulas for the c_i in terms of various symmetric functions are unstable in the presence of roundoff error and expensive to implement. Techniques based upon similarity transformations break down when A is nearly derogatory. We shall have more to say about these difficulties in connection with Methods 12 and 13.

In Method 8 we attempted to expand e^{tA} in terms of the matrices $I, A,\cdots,A^{n-1}$. If $\{A_0,\cdots,A_{n-1}\}$ is some other set of matrices which span the same subspace, then there exist analytic functions $\beta_j(t)$ such that

$$e^{tA}=\sum_{j=0}^{n-1}\beta_j(t)A_j.$$

The convenience of this formula depends upon how easily the A_j and $\beta_j(t)$ can be generated. If the eigenvalues $\lambda_1,\cdots,\lambda_n$ of A are known, we have the following three methods.

METHOD 9. LAGRANGE INTERPOLATION.

$$e^{tA}=\sum_{j=0}^{n-1}e^{\lambda_j t}\prod_{\substack{k=1\\k\neq j}}^{n}\frac{(A-\lambda_k I)}{(\lambda_j-\lambda_k)}.$$

METHOD 10. NEWTON INTERPOLATION.

$$e^{tA}=e^{\lambda_1 t}I+\sum_{j=2}^{n}[\lambda_1,\cdots,\lambda_j]\prod_{k=1}^{j-1}(A-\lambda_k I).$$

The divided differences $[\lambda_1,\cdots,\lambda_j]$ depend on t and are defined recursively by

$$[\lambda_1,\lambda_2]=(e^{\lambda_1 t}-e^{\lambda_2 t})/(\lambda_1-\lambda_2),$$

$$[\lambda_1,\cdots,\lambda_{k+1}]=\frac{[\lambda_1,\cdots,\lambda_k]-[\lambda_2,\cdots,\lambda_{k+1}]}{\lambda_1-\lambda_{k+1}}\qquad(k\geq 2).$$

We refer to MacDuffee [9] for a discussion of these formulae in the confluent eigenvalue case.

METHOD 11. VANDERMONDE. There are other methods for computing the matrices

$$A_j = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{(A - \lambda_k I)}{(\lambda_j - \lambda_k)}$$

which were required in Method 9. One of these involves the Vandermonde matrix

$$V = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \lambda_1 & \lambda_2 & \cdots & \lambda_n \\ \vdots & \vdots & & \vdots \\ \lambda_1^{n-1} & \lambda_2^{n-1} & \cdots & \lambda_n^{n-1} \end{bmatrix}$$

If v_{jk} is the (j, k) entry of V^{-1}, then

$$A_j = \sum_{k=1}^{n} v_{jk} A^{k-1},$$

and

$$e^{tA} = \sum_{j=1}^{n} e^{\lambda_j t} A_j.$$

When A has repeated eigenvalues, the appropriate confluent Vandermonde matrix is involved. Closed expressions for the v_{jk} are available and Vidysager [92] has proposed their use.

Methods 9, 10, and 11 suffer on several accounts. They are $O(n^4)$ algorithms making them prohibitively expensive except for small n. If the spanning matrices $A_0, \cdots, A_{n-1}$ are saved, then storage is n^3 which is an order of magnitude greater than the amount of storage required by any "nonpolynomial" method. Furthermore, even though the formulas which define Methods 9, 10, and 11 have special form in the confluent case, we do not have a satisfactory situation. The "gray" area of near confluence poses difficult problems which are best discussed in the next section on decomposition techniques.

The next two methods of this section do not require the eigenvalues of A and thus appear to be free of the problems associated with confluence. However, equally formidable difficulties attend these algorithms.

METHOD 12. INVERSE LAPLACE TRANSFORMS. If $\mathcal{L}[e^{tA}]$ is the Laplace transform of the matrix exponential, then

$$\mathcal{L}[e^{tA}] = (sI - A)^{-1}.$$

The entries of this matrix are rational functions of s. In fact,

$$(sI - A)^{-1} = \sum_{k=0}^{n-1} \frac{s^{n-k-1}}{c(s)} A_k,$$

where $c(s) = \det(sI - A) = s^n - \sum_{k=0}^{n-1} c_k s^k$ and for $k = 1, \cdots, n$:

$$c_{n-k} = -\text{trace}\,(A_{k-1}A)/k, \qquad A_k = A_{k-1}A - c_{n-k}I \qquad (A_0 = I).$$

These recursions were derived by Leverrier and Faddeeva [3] and can be used to evaluate e^{tA}:

$$e^{tA} = \sum_{k=0}^{n-1} \mathcal{L}^{-1}[s^{n-k-1}/c(s)]A_k.$$

The inverse transforms $\mathcal{L}^{-1}[s^{n-k-1}/c(s)]$ can be expressed as a power series in t. Liou [102] suggests evaluating these series using various recursions involving the c_k. We suppress the details of this procedure because of its similarity to Method 8. There are other ways Laplace transforms can be used to evaluate e^{tA} [78], [80], [88], [89], [93]. By and large, these techniques have the same drawbacks as Methods 8–11. They are $O(n^4)$ for general matrices and may be seriously effected by roundoff error.

METHOD 13. COMPANION MATRIX. We now discuss techniques which involve the computation of e^C where C is a companion matrix:

$$C = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & & \vdots \\ & & & & 1 \\ c_0 & c_1 & c_2 & \cdots & c_{n-1} \end{bmatrix}.$$

Companion matrices have some interesting properties which various authors have tried to exploit:

(i) C is sparse.

(ii) The characteristic polynomial of C is $c(z) = z^n - \sum_{k=0}^{n-1} c_k z^k$.

(iii) If V is the Vandermonde matrix of eigenvalues of C (see Method 11), then $V^{-1}CV$ is in Jordan form. (Confluent Vandermonde matrices are involved in the multiple eigenvalue case.)

(iv) If A is not derogatory, then it is similar to a companion matrix; otherwise it is similar to a direct sum of companion matrices.

Because C is sparse, small powers of C cost considerably less than the usual n^3 flops. Consequently, one could implement Method 3 (scaling and squaring) with a reduced amount of work.

Since the characteristic polynomial of C is known, one can apply Method 8 or various other techniques which involve recursions with the c_k. However, this is not generally advisable in view of the catastrophic cancellation that can occur.

As we mentioned during our discussion of Method 11, the closed expression for V^{-1} is extremely sensitive. Because V^{-1} is so poorly conditioned, exploitation of property (iii) will generally yield a poor estimate of e^A.

If $A = YCY^{-1}$, then from the series definition of the matrix exponential it is easy to verify that

$$e^A = Ye^CY^{-1}.$$

Hence, property (iv) leads us to an algorithm for computing the exponential of a general matrix. Although the reduction of A to companion form is a rational process, the algorithm for accomplishing this are extremely unstable and should be avoided [14].

We mention that if the original differential equation is actually a single nth order equation written as a system of first order equations, then the matrix is already in companion form. Consequently, the unstable reduction is not necessary. This is the only situation in which companion matrix methods should be considered.

We conclude this section with an interesting aside on computing e^H where $H = (h_{ij})$ is lower Hessenberg ($h_{ij} = 0$, $j > i+1$). Notice that companion matrices are lower Hessenberg. Our interest in computing e^H stems from the fact that any real matrix A is orthogonally similar to a lower Hessenberg matrix. Hence, if

$$A = QHQ^T, \qquad Q^TQ = I,$$

then

$$e^A = Qe^HQ^T.$$

Unlike the reduction to companion form, this factorization can be stably computed using the EISPACK routine ORTHES [113].

Now, let f_k denote the kth column of e^H. It is easy to verify that

$$Hf_k = \sum_{i=k-1}^{n} h_{ik}f_i \qquad (k \geqq 2),$$

by equating the kth columns in the matrix identity $He^H = e^HH$. If none of the superdiagonal entries $h_{k-1,k}$ are zero, then once f_n is known, the other f_k follow immediately from

$$f_{k-1} = \frac{1}{h_{k-1,k}}\left[Hf_k - \sum_{i=k}^{n} h_{ik}f_i\right].$$

Similar recursive procedures have been suggested in connection with computing e^C [104]. Since f_n equals $x(1)$ where $x(t)$ solves $Hx = \dot{x}$, $x(0) = (0, \cdots, 0, 1)^T$, it could be found using one of the o.d.e. methods in the previous section.

There are ways to recover in the above algorithm should any of the $h_{k-1,k}$ be zero. However, numerically the problem is when we have a small, but non-negligible $h_{k-1,k}$. In this case rounding errors involving a factor of $1/h_{k-1,k}$ will occur precluding the possibility of an accurate computation of e^H.

In summary, methods for computing e^A which involve the reduction of A to companion or Hessenberg form are not attractive. However, there are other matrix factorizations which can be more satisfactorily exploited in the course of evaluating e^A and these will be discussed in the next section.

6. Matrix decomposition methods. The methods which are likely to be most efficient for problems involving large matrices and repeated evaluation of e^{tA} are those which are based on factorizations or decompositions of the matrix A. If A happens to be symmetric, then all these methods reduce to a simple very effective algorithm.

All the matrix decompositions are based on similarity transformations of the form

$$A = SBS^{-1}.$$

As we have mentioned, the power series definition of e^{tA} implies

$$e^{tA} = Se^{tB}S^{-1}.$$

The idea is to find an S for which e^{tB} is easy to compute. The difficulty is that S may be close to singular which means that cond (S) is large.

METHOD 14. EIGENVECTORS. The naive approach is to take S to be the matrix whose columns are eigenvectors of A, that is, $S = V$ where

$$V = [v_1| \cdots |v_n]$$