October, 1967 Report ESL-R-326 Copy 139

OPTIMAL PIECEWISE CONSTANT SOLUTIONS OF THE LINEAR REGULATOR PROBLEM

by

Thomas Fortmann

This research was carried out at the M.I.T. Electronic Systems Laboratory with partial support extended by the National Aeronautics and Space Administration under Research Grant NGR-22-009(124), M.I.T. DSR Project No. 76265.

Electronic Systems Laboratory
Department of Electrical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ACKNOWLEDGEMENT

The author wishes to express his gratitude to Professor Michael Athans for his encouragement and supervision during the course of this research. Special thanks are also due to Dr. David Kleinman for his cooperation, and to Bill Levine and Alex Levis for helpful discussions

This research was supported in part by a National Science Foundation Graduate Fellowship and in part by the National Aeronautics and Space Administration under Research Grant NGR-22-009(124). The computations were performed at the M.I.T. Computation Center.

ABSTRACT

The solution of the optimal linear regulator problem with a quadratic cost functional involves a time-varying matrix (the solution of a matrix Riccati differential equation) in the feedback loop. One suboptimal approach to the problem which simplifies implementation is to let the time-varying matrix be piecewise constant. A computer program is presented here which calculates the (sub) optimal feedback gains under this constraint, where the plant, cost functional, and switching times are specified a priori, and results are obtained for a third-order example.

CONTENTS

LIST OF FIGURES

I. INTRODUCTION

The solution of the optimal linear regulator problem with quadratic cost in feedback form expresses the control vector as a gain matrix multiplying the state vector. For problems with a fixed terminal time, this gain comes from the time-varying solution of a matrix Riccati equation. Since the Riccati equation must be solved off-line, implementation of such a control may lead to severe storage and synchronization problems.

Kleinman recognized this drawback and in his Ph.D. thesis[1*] he derived a necessary condition for optimality when the time dependence of the feedback gain matrix is constrained to have a certain form. In the case of a piecewise constant gain, he also suggested an iterative procedure for determining a solution of the necessary condition. The purpose of this work is to incorporate Kleinman's iterative scheme into a complete computer algorithm.

In Section II the optimal linear regulator problem is formulated and modified slightly to accommodate piecewise constant feedback gains; then the new optimization problem is stated and the necessary condition for optimality is given. In Section III the iterative scheme for satisfying the necessary condition is described, and the basic steps of the computer algorithms are shown in the form of a flow chart. The program steps and problems of error control are then discussed in more detail. Numerical results are discussed in Section IV.

Given a system to be controlled, a cost functional, and switching times, the program generates the appropriate gain matrices for a suboptimal feedback control law. A basic assumption here, in the absence of a sufficient condition for optimality, is that the optimal gains exist and are unique. It is also assumed that the necessary condition has no extraneous solutions to which the iterative procedure may converge.

---

* Superscripts refer to numbered items in the References.

II. PROBLEM FORMULATION AND NECESSARY CONDITION

The linear regulator problem involves a time-invariant system governed by a differential equation

$$\begin{aligned} \dot{\underline{x}}(t) &= \underline{A}\,\underline{x}(t) + \underline{B}\,\underline{u}(t) \\ \underline{x}(t_0) &= \underline{x}_0 \end{aligned} \tag{1}$$

where $\underline{x}(t)$ is the state and $\underline{u}(t)$ is the control.* The optimal control $\underline{u}^*(t)$, $t_0 \leq t \leq T$, is that which minimizes the quadratic cost functional,

$$J[\underline{x}_0, t_0; \underline{u}(\cdot)] \triangleq \frac{1}{2}\underline{x}'(T)\underline{F}\,\underline{x}(T) + \frac{1}{2}\int_{t_0}^{T} [\underline{x}'(t)\underline{Q}\,\underline{x}(t) + \underline{u}'(t)\underline{u}(t)]\,dt \tag{2}$$

where $\underline{F}$ and $\underline{Q}$ are positive semidefinite, constant matrices which are not both zero. The optimal solution is well-known (see Athans and Falb, Reference 2, Chapter 9), and is specified in the feedback form,

$$\underline{u}^*(t) = -\underline{B}\,\underline{K}(t)\underline{x}(t) \triangleq -\underline{L}^*(t)\underline{x}(t) \tag{3}$$

where $\underline{K}(t)$ is the symmetric, positive semidefinite solution of the matrix Riccati differential equation,

$$\begin{aligned} \dot{\underline{K}}(t) &= -\underline{K}(t)\underline{A} - \underline{A}\,\underline{K}(t) - \underline{Q} + \underline{K}(t)\underline{B}\,\underline{B}'\underline{K}(t) \\ \underline{K}(T) &= \underline{F} \end{aligned} \tag{4}$$

To implement the optimal system in real time, the optimal feedback matrix $\underline{L}^*(t)$ must be stored on the interval $[t_0, T]$, as solutions of the Riccati equation, Eq. 4, are unstable in the forward time direction. Recognizing that this could render such a system

---

* Capital underscored letters denote matrices; lower-case underscored letters are vectors; the dot indicates differentiation; and a prime means transpose.

impractical, Kleinman[1] considered a class of suboptimal feedback matrices, including the case where $\underline{L}^*(t)$ is constrained to be piecewise constant.

Suppose now that one does not wish to use the optimal control law of Eq. 3. In particular, let $\underline{L}(t)$ denote an arbitrary feedback gain matrix, so that the closed-loop system becomes

$$\begin{aligned} \dot{\underline{x}}(t) &= [\underline{A} - \underline{B}\,\underline{L}(t)]\,\underline{x}(t) \triangleq \underline{H}(t)\underline{x}(t) \\ \underline{x}(t_0) &= \underline{x}_0 \end{aligned} \tag{5}$$

Now the cost functional, Eq. 2, may be evaluated to be

$$J[\underline{x}_0, t_0; \underline{L}(\cdot)] = \frac{1}{2}\underline{x}_0'\underline{V}(t_0)\underline{x}_0 \tag{6}$$

where the symmetric, positive semidefinite matrix $\underline{V}(t_0)$ is given by

$$\underline{V}(t_0) \triangleq \underline{\Phi}'(T, t_0)\underline{F}\,\underline{\Phi}(T, t_0) + \int_{t_0}^{T} \underline{\Phi}'(\tau, t_0)[\underline{Q} + \underline{L}'(\tau)\underline{L}(\tau)]\,\underline{\Phi}(\tau, t_0)\,d\tau \tag{7}$$

and where $\underline{\Phi}(\tau, t)$ is the state transition matrix for the closed-loop system, Eq. 5, i.e.,

$$\begin{aligned} \frac{d}{d\tau}\underline{\Phi}(\tau, t) &= \underline{H}(\tau)\underline{\Phi}(\tau, t) \\ \frac{d}{d\tau}\underline{\Phi}(\tau, t) &= -\underline{\Phi}(\tau, t)\,\underline{H}(t) \end{aligned} \tag{8}$$

$$\underline{x}(\tau) = \underline{\Phi}(\tau, t)\,\underline{x}(t) \tag{9}$$

By letting the argument of $\underline{V}(\cdot)$ vary and differentiating, one obtains

$$\underline{V}(t) = \underline{\Phi}'(T, t)\underline{F}\,\underline{\Phi}(T, t) + \int_{t}^{T} \underline{\Phi}'(\tau, t)[\underline{Q} + \underline{L}'(\tau)\underline{L}(\tau)]\,\underline{\Phi}(\tau, t)\,d\tau \tag{10}$$

or

$$\begin{aligned} \dot{\underline{V}}(t) &= -\underline{H}'(t)\underline{V}(t) - \underline{V}(t)\underline{H}(t) - \underline{Q} - \underline{L}'(t)\underline{L}(t) \\ \underline{V}(T) &= \underline{F} \end{aligned} \tag{11}$$

In the system to be considered here, the feedback gain matrix is constrained to be piecewise constant, i.e.,

$$\left.\begin{aligned} \underline{L}(t) &= \underline{L}(n) \\ \underline{H}(t) &= \underline{H}(n) = \underline{A} - \underline{B}\,\underline{L}(n) \end{aligned}\right\} \text{for } t_n < t \leq t_{n+1} \tag{12}$$

where the switching times

$$t_0 < t_1 < \ldots < t_{N-1} < t_N \triangleq T$$

are given. In this case the transition matrix takes on the particularly simple form

$$\underline{\Phi}(t, t_0) = e^{\underline{H}(n)(t-t_n)}\,\underline{\Phi}(t_n, t_0), \quad t_n \leq t \leq t_{n+1}$$

$$\underline{\Phi}(t_n, t_0) = \prod_{i=0}^{n-1} e^{\underline{H}(i)(t_{i+1}-t_i)} \tag{13}$$

$$\underline{\Phi}(t_0, t_0) = \underline{I}$$

In order to avoid a solution which depends on the initial state $\underline{x}_0$, Kleinman chose to minimize the functional*

$$\mu[\underline{L}(\cdot)] = \mathrm{tr}\underline{V}(t_0)$$

with respect to $\underline{L}(n)$, $n = 0, 1, 2, \ldots N-1$, rather than the conventional cost functional, Eq. 6.

To summarize, the optimization problem which is to be solved is the following:

A suboptimal linear regulator problem

Given a dynamical system, Eq. 1, and a set of switching times $\{t_0, t_1, t_2, \ldots T_n = T\}$, determine the set of feedback gain matrices

$$L^*(n), \quad n = 0, 1, 2, \ldots N-1$$

---

* The implications of this are discussed in Ref. 1, pp. 72-76.

such that the trajectory of the state of the closed-loop system

$$\dot{\underline{x}}(t) = [\underline{A} - \underline{B}\,\underline{L}^*(n)]\underline{x}(t) = \underline{H}^*(n)\underline{x}(t), \quad t_n \leq t < t_{n+1}$$

$$\underline{x}(t_0) = \underline{x}_0$$

on the interval $[0, T]$ minimizes the functional

$$\mu[\underline{L}(\cdot)] = \mathrm{tr}\underline{V}(t_0)$$

where $\underline{V}(t_0)$ is given by Eq. 7.

Appealing to elementary calculus of variations, a necessary condition for $\underline{L}^*(\cdot)$ to extremize $\mathrm{tr}\underline{V}(t_0)$ is that

$$\left.\frac{\partial \mathrm{tr}V(t_0)}{\partial \ell_{ij}(n)}\right|_{\ell_{ij}(n)=\ell^*_{ij}(n)} = 0 \qquad \begin{array}{l}\text{for all } i,\ j \text{ and} \\ \text{for all } n=0, 1, \ldots N-1\end{array} \tag{14}$$

where $\ell_{ij}(n)$ are the elements of the matrix $\underline{L}(n)$. The notation may be simplified considerably at this point by using the concept of a "gradient matrix"[3]

$$\frac{\partial \mathrm{tr}\underline{V}(t_0)}{\partial \underline{L}(n)}$$

whose (i, j)-th element is

$$\frac{\partial \mathrm{tr}\underline{V}(t_0)}{\partial \ell_{ij}(n)}$$

The calculation of the gradient matrix for this problem is straightforward but tedious, and will not be repeated here.* The result is that the necessary condition, Eq. 14, becomes

$$\left.\frac{\partial \mathrm{tr}\underline{V}(t_0)}{\partial \underline{L}(n)}\right|_{\underline{L}(n)=\underline{L}^*(n)} = -2\int_{t_n}^{t_{n+1}} [\underline{L}^*(n) - \underline{B}'\underline{V}^*(t)]\underline{\Phi}^*(t, t_0)\underline{\Phi}^{*\prime}(t, t_0)\,dt = 0$$

$$\text{for } n = 0, 1, 2, \ldots N-1 \tag{15}$$

---

* This is done in Kleinman, Ref. 1, pp. 84-86, for a more general problem.

where $\underline{V}^*(t)$ satisfies Eq. 10 with $\underline{\Phi} = \underline{\Phi}^*$ and $\underline{\Phi}^*$ satisfies Eq. 13 with $\underline{H}(n) = \underline{H}^*(n) = \underline{A} - \underline{B}\,\underline{L}^*(n)$. Equation 15 may also be written in the form

$$\underline{L}^*(n) = \underline{B}'\left[\int_{t_n}^{t_{n+1}} \underline{V}^*(t)\underline{\Phi}^*(t,t_0)\Phi^{*\prime}(t,t_0)\,dt\right]\left[\int_{t_n}^{t_{n+1}} \underline{\Phi}^*(t,t_0)\underline{\Phi}^{*\prime}(t,t_0)\,dt\right]^{-1}$$

$$\text{for } n = 0,1,2,\ldots N-1 \tag{16}$$

In the next section, an iterative procedure for finding $\underline{L}^*(n)$, $n=0,1,2,\ldots N-1$, which satisfy Eq. 16 will be discussed. But it should be noted that Eq. 16 is not a sufficient condition for optimality, and hence one must assume a priori that an optimum exists and is unique. Furthermore, the iterative scheme is not foolproof, and it may converge to relative minima or inflection points, if these exist.

III. ITERATIVE CALCULATION OF GAIN MATRICES

Since the right-hand side of the necessary condition, Eq. 16, depends on the unknown gains $\underline{L}^*(n)$, the solution must be found iteratively. The method proposed by Kleinman (Ref. 1, pp. 98-111) is the following:*

$$\underline{L}^{i+1}(n) = \underline{L}^{i}(n) + \epsilon[\hat{\underline{L}}^{i}(n) - \underline{L}^{i}(n)], \qquad 0 < \epsilon \leq 1 \qquad (17)$$

$$n=0, 1, \ldots N-1$$

where

$$\hat{\underline{L}}^{i}(n) \triangleq \underline{B}'\left[\int_{t_n}^{t_{n+1}} \underline{V}^{i}(t)\underline{\Phi}^{i}(t, t_0)\underline{\Phi}^{i'}(t, t_0)dt\right]\left[\int_{t_n}^{t_{n+1}} \underline{\Phi}^{i}(t, t_0)\Phi^{i'}(t, t_0)dt\right]^{-1}$$

$$n=0, 1, 2, \ldots N-1 \qquad (18)$$

and where $\underline{V}^{i}$ and $\underline{\Phi}^{i}$ are obtained by using $\underline{L}^{i}$ in Eqs. 10 and 13. Kleinman verified that this converges to $\underline{L}^*(n)$ by showing that the quantity

$$\mu[\underline{L}^{i}(\cdot)] - \mu[\underline{L}^{i+1}(\cdot)] = \mathrm{tr}\underline{V}^{i}(t_0) - \mathrm{tr}\underline{V}^{i+1}(t_0)$$

is always positive to first order in ϵ. Thus if $\mu^{i+1} > \mu^{i}$ at any iteration, ϵ can be decreased and the iteration repeated until $\mu^{i+1} < \mu^{i}$. For a close enough initial guess, one would expect convergence with $\epsilon = 1$, in which case this scheme is analogous to the method of successive approximations.

The basic steps of the computer algorithm for determining $\underline{L}^*(n)$, $n=0, 1, \ldots, N-1$, are indicated in the flow chart of Fig. 1, and then are discussed in more detail.

A. GENERATION OF STARTING VALUES

The initial values $\underline{L}^{1}(n)$ may be chosen quite arbitrarily, as long as the resulting system is stable. The closer they are to the optimum values, of course, the faster the algorithm will converge. Kleinman

---

* Superscripts are the iteration index.

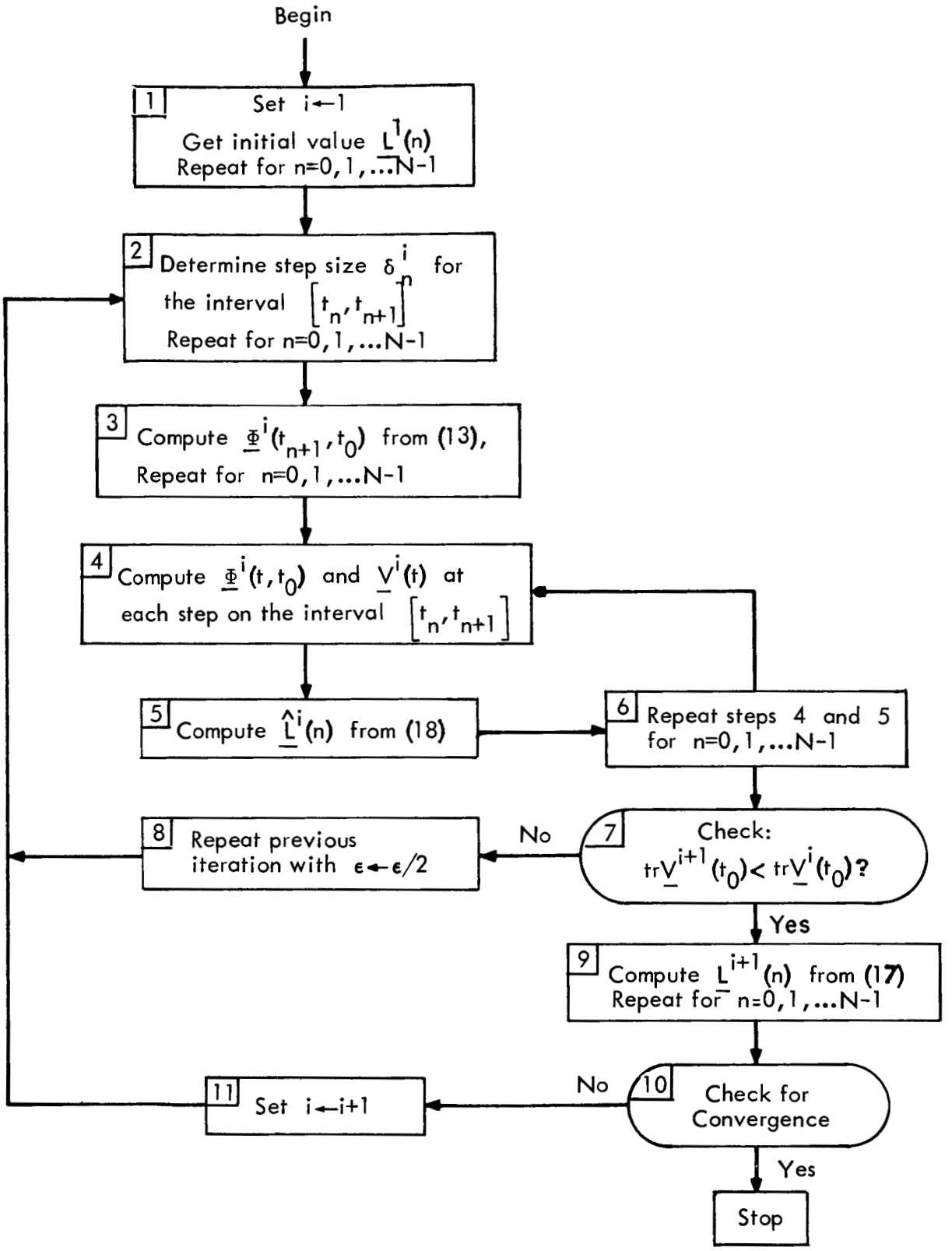

Fig. 1 Basic Flow Chart

proposes the average of the Riccati solution on each time interval as a starting value, i.e.,

$$\underline{L}^1(n) = \frac{1}{t_{n+1}-t_n} \int_{t_n}^{t_{n+1}} \underline{B}\,\underline{K}(t)dt, \quad n=0,1,\ldots N-1 \tag{19}$$

Another possibility is to set all the $L^1(n)$ equal to the steady-state solution of the Riccati equation.

This program will assume that the starting values are given, although a subroutine could easily be added which would generate them.

B. DETERMINATION OF STEP SIZES

In any continuous-time problem which is to be solved by digital computer, the time axis must be made discrete. In the present case this consists of dividing each time interval $[t_n, t_{n+1}]$ into M_n subintervals, or steps, each of length

$$\delta_n = \frac{t_{n+1}-t_n}{M_n}, \quad n=0,1,\ldots N-1 \tag{20}$$

The values of $\underline{V}(t)$ and $\underline{\Phi}(t,t_0)$ are considered only at these discrete times. The size of the δ_n's is chosen to give some desired accuracy in the final solutions. Consideration of this choice will be postponed until part E, when the other computations have been discussed.

C. COMPUTATION OF STATE TRANSITION MATRICES

Calculating $\underline{\Phi}(t,t_0)$ is straightforward, since from Eq. 13,

$$\underline{\Phi}(t_n+k\delta_n, t_0) = \left[e^{\underline{H}(n)\delta_n}\right]^k \underline{\Phi}(t_n, t_0) \tag{21}$$

where k is an integer between 1 and M_n. Thus the $\underline{\Phi}$ matrix at each step is just $\underline{\Phi}$ at the previous step multiplied by the matrix exponential

$$\underline{\Phi}(t_n + k\delta_n + \delta_n,\ t_n + k\delta_n) = e^{\underline{H}(n)\delta_n} \tag{22}$$

which needs to be calculated only once for each interval $[t_n, t_{n+1}]$.

This may be done conveniently with the defining series,

$$e^{\underline{H}\delta} = \underline{I} + \underline{H} + \frac{\underline{H}^2\delta^2}{2!} + \frac{\underline{H}^3\delta^3}{3!} + \ldots \tag{23}$$

Since δ will generally be quite small, this series will converge rapidly, the number of terms to be kept depending on the desired accuracy of $\underline{\Phi}(t, t_0)$. However, the error bound on $[e^{\underline{H}\delta}]^k$ is related in a complicated way* to the error bound on $e^{\underline{H}\delta}$, so the series will be terminated when the terms become smaller than a certain tolerance (TOL2), which is left as a program variable.

D. COMPUTATION OF $\underline{V}(t)$

Two forms are available for $\underline{V}(t)$: Eqs. 10 and 11. The latter is a differential equation which could be integrated (backwards in time) by any of the usual methods. However, the piecewise linearity of the system leads to a more convenient method using Eq. 10:

$$\underline{V}(t-\delta_n) = \underline{\Phi}'(T, t-\delta_n)\underline{F}\,\underline{\Phi}(T, t-\delta_n)$$

$$+ \int_{t-\delta_n}^{T} \underline{\Phi}'(\tau, t-\delta_n)[\underline{Q} + \underline{L}'(n)\underline{L}(n)]\underline{\Phi}(\tau, t-\delta_n)\,d\tau \tag{24}$$

where $t = t_n + k\delta_n$, $1 \leq k \leq M_n$. Using the familiar semigroup property of transition matrices,

$$\underline{\Phi}(t_2, t_0) = \underline{\Phi}(t_2, t_1)\underline{\Phi}(t_1, t_0) \tag{25}$$

this becomes

---

* See Appendix A. One convenient (but potentially inaccurate) rule of thumb is that the error in A^k is k times the error in A.

$$\underline{V}(t-\delta_n) = \underline{\Phi}'(t, t-\delta_n)\underline{\Phi}'(T, t)\underline{F}\,\underline{\Phi}(T, t)\underline{\Phi}(t, t-\delta_n)$$

$$+ \underline{\Phi}'(t, t-\delta_n)\int_t^T \underline{\Phi}'(\tau, t)[\underline{Q}+\underline{L}'(n)\underline{L}(n)]\,\underline{\Phi}(\tau, t)\,d\tau\,\underline{\Phi}(t, t-\delta_n)$$

$$+\int_{t-\delta_n}^{t} \underline{\Phi}'(\tau, t-\delta_n)[\underline{Q}+\underline{L}'(n)\underline{L}(n)]\,\underline{\Phi}(\tau, t-\delta_n)\,d\tau \qquad (26)$$

Since $t_n + \delta_n \le t \le t_{n+1}$, this becomes

$$\underline{V}(t-\delta_n) = e^{\underline{H}'(n)\delta_n}\underline{V}(t)e^{\underline{H}(n)\delta_n} + \int_0^{\delta_n} e^{\underline{H}'(n)\tau}[\underline{Q}+\underline{L}'(n)\underline{L}(n)]e^{\underline{H}(n)\tau}\,d\tau \qquad (27)$$

The first term may be computed from $\underline{V}(t)$ with two multiplications, and the quantity $e^{\underline{H}(n)\delta_n}$ will already have been calculated in part C above. The second term is invariant for all t in the interval $[t_n + \delta_n, t_{n+1}]$, and may be computed with a series, using Eq. 23:

$$\int_0^{\delta} e^{\underline{H}'\tau}\,\underline{C}e^{\underline{H}\tau}\,d\tau$$

$$=\int_0^{\delta}\left[\underline{I} + \underline{H}'\tau + \frac{\underline{H}'^2\tau^2}{2!} + \ldots\right]\underline{C}\left[\underline{I} + \underline{H}\tau + \frac{\underline{H}^2\tau^2}{2!} + \ldots\right]d\tau$$

$$=\int_0^{\delta} \underline{C} + [\underline{H}'\underline{C} + \underline{C}\underline{H}]\tau + \tfrac{1}{2!}[\underline{H}'^2\underline{C} + 2\underline{H}'\underline{C}\underline{H} + \underline{C}\underline{H}^2]\tau^2 + \ldots\,d\tau$$

$$= \underline{C}\,\delta + [\underline{H}'\underline{C} + \underline{C}\underline{H}]\frac{\delta^2}{2!} + [\underline{H}'^2\underline{C} + 2\underline{H}'\underline{C}\underline{H} + \underline{C}\underline{H}^2]\frac{\delta^3}{3!} + \ldots \qquad (28)$$

This is an easy series to compute, since each term (denoted $\underline{T}_j$) follows from the previous one:

$$\underline{T}_j = \frac{\delta}{j}[\underline{H}'\underline{T}_{j-1} + \underline{T}_{j-1}\underline{H}] \qquad (29)$$

The series will be terminated when the terms become smaller than TOL2, just as in part C above.

E. COMPUTATION OF $\hat{\underline{L}}(n)$

Once $\underline{V}(t)$ and $\underline{\Phi}(t, t_0)$ have been calculated and stored for each step in the interval $[t_n, t_{n+1}]$, any of the common quadrature methods may be used to evaluate the integrals in Eq. 18. The scheme to be used here was devised by Romberg, and is essentially a Richardson-type extrapolation of the trapezoidal rule.* It is felt that this procedure is worth the extra program complexity because it provides substantial error reduction at very little cost in terms of running time. Since this decreases M_n, the number of steps required to give $\hat{\underline{L}}(n)$ a certain error tolerance, the computation time and storage requirements for $\underline{V}(t)$ and $\underline{\Phi}(t, t_0)$ can be significantly reduced.

At this point a rule for choosing M_n (which must be a power of 2 for this method) may be formulated. Unfortunately, the error term for Romberg integration contains a high-order derivative of the integrand, whose computation is both impractical and inaccurate. However, the error formula for the trapezoidal rule on M_n subintervals, from which this method is derived, contains only a second derivative:**

$$|\underline{E}(n)| \leq M_n \max_{\eta} \left| \frac{1}{12} \delta_n^3 \ddot{\underline{I}}(\eta) \right|$$

$$\doteq \frac{(t_{n+1}-t_n)^3}{12M_n^2} \cdot \max_{\eta} |\ddot{\underline{I}}(\eta)|, \quad t_n \leq \eta \leq t_{n+1} \tag{30}$$

where $\underline{E}(n)$ is the error, $\underline{I}(t)$ is the integrand, and the inequality and absolute values are to be interpreted element by element. Since the actual Romberg error is directly related to $|\underline{E}(n)|$, M_n will be chosen to make this error less than TOL1, a program variable. The Romberg scheme provides an a posteriori error estimate, and TOL1 may be adjusted on successive runs until an appropriate value evolves.

---

* For details see Ralston, Ref. 4, pp. 121-124.

** Ibid, pp. 116-117.

Since the integrand is not actually known when the step size is chosen, $\underline{E}(n)$ can only be an estimate of the error, rather than a bound on it. Hence, it is convenient to replace Eq. 30 by

$$\|\underline{E}(n)\| \simeq \frac{(t_{n+1}-t_n)^3}{12M_n^2} \|\ddot{\underline{I}}(n)\| \tag{31}$$

where $\ddot{\underline{I}}(n)$ approximates the second derivative of the integrand on $[t_n, t_{n+1}]$ and $\|\cdot\|$ is the norm defined by

$$\|\underline{A}\| \triangleq \frac{1}{n}\left[\sum_{i,j=1}^{n} a_{ij}^2\right]^{1/2} \tag{32}$$

Differentiating the first integrand in Eq. 18 twice yields

$$\begin{aligned}\ddot{\underline{I}}(t) &= \underline{H}'^2(n)\underline{V}(t)\underline{F}(t) - 2\underline{H}'(n)\underline{V}(t)\underline{F}(t)\underline{H}'(n) \\ &+ \underline{V}(t)\underline{F}(t)\underline{H}'^2(n) + \underline{H}'(n)[\underline{Q} + \underline{L}'(n)\underline{L}(n)]\underline{F}(t) \\ &- [\underline{Q} + \underline{L}'(n)\underline{L}(n)][\underline{H}(n)\underline{F}(t) + \underline{F}(t)\underline{H}'(n)], \quad t_n \le t \le t_{n+1}\end{aligned} \tag{33}$$

where $\underline{F}(t) \triangleq \underline{\Phi}(t, t_0)\underline{\Phi}'(t, t_0)$. Since $\|\underline{A}\,\underline{B}\| \le \|\underline{A}\| \cdot \|\underline{B}\|$, $\|\ddot{\underline{I}}(n)\|$ will be approximated by

$$\begin{aligned}\|\ddot{\underline{I}}(n)\| &\simeq 4\|\underline{H}(n)\|^2 \|\underline{V}(t_0)\| \, \|\underline{\Phi}(t_n, t_0)\|^2 \\ &+ 4\|\underline{H}(n)\| \, \|\underline{\Phi}(t_n, t_0)\|^2 \|\underline{Q} + \underline{L}'(n)\underline{L}(n)\|\end{aligned} \tag{34}$$

where $\underline{\Phi}$ and $\underline{V}$ are taken from the previous iteration.

Thus the number of steps for the n-th time interval, M_n, will be taken as the smallest power of 2 such that

$$M_n \ge \left[\frac{(t_{n+1}-t_n)^3}{12(TOL1)} \|\ddot{\underline{I}}(n)\|\right]^{1/2} \tag{35}$$

where $\|\ddot{\underline{I}}(n)\|$ is given by Eq. 34 and TOL1 is an adjustable program variable.

It is clear from the above discussion that Eq. 35 is at best a "ball-park" estimate, but a more accurate determination of the step

sizes would require a substantial increase in program complexity. No problems were encountered with this scheme when the program was tested.

F. FINAL STEPS

With $\underline{\hat{L}}^{i}(n)$, $n=0,1,\ldots N-1$ calculated, the new feedback matrices $\underline{L}^{i+1}(n)$ may be computed from Eq. 17, but first it is necessary to check whether the cost has increased. If

$$tr\underline{V}^{i}(t_0) > (TOL4) \cdot tr\underline{V}^{i-1}(t_0) \tag{36}$$

then $\underline{L}^{i}(n)$ is replaced by $\underline{L}^{i-1}(n)$, $n=0,1,\ldots N-1$, ϵ is halved, and the previous iteration is repeated (TOL4 is a program variable slightly larger than 1).*

Otherwise the $\underline{\hat{L}}^{i}(n)$ are checked for convergence, one entry at a time. If

$$1-(TOL3) \leq \frac{\ell_{jk}(n)}{\ell_{jk}(n)} \leq 1+(TOL3) \quad \text{for all } j,k,n \tag{37}$$

where $\ell_{ij}(n)$ are the elements of $\underline{L}_{ij}(n)$, then the iterations are terminated (TOL3 is another program variable to be supplied at run time). If Eq. 37 is not satisfied then Eq. 17 is used to get $L^{i+1}(n)$, $n=0,1,\ldots N-1$, and the iterations continue.

---

* In several test runs it was found that $tr\underline{V}^{i}(t_0)$ converged before $\underline{L}^{i}(n)$, and inequality 36 appeared to be true, due to roundoff noise. Thus TOL4 was added to avoid unnecessary repetitions.

IV. NUMERICAL RESULTS

The algorithm as outlined above has been programmed in PL/I language and run on the IBM System/360 computer at the M.I.T. Computation Center. A more detailed description of the program is given in Appendix B.

The program was tested with the two numerical examples of Kleinman, a double integrator plant with $N=1$ and $N=2$. His results were duplicated to within roundoff errors and the reader is referred to his thesis (Ref. 1, pp. 111-118) for a discussion of them. The program was then used to generate suboptimal feedback matrices for the following output regulator problem involving a third order plant:

System dynamics:

$$\dot{\underline{x}}(t) = \underline{A}\,\underline{x}(t) + \underline{b}\,u(t) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & -2 & 0 \end{bmatrix} \underline{x}(t) + \begin{bmatrix} 2 \\ 2 \\ -1 \end{bmatrix} u(t) \tag{52}$$

$$y(t) = \underline{c}'\,\underline{x}(t) = [\,1 \quad -1 \quad 0\,]\,\underline{x}(t) \tag{53}$$

Transfer function:

$$H(s) = \underline{c}'(s\underline{I} - \underline{A})^{-1}\underline{b} = \frac{10}{(s+1)(s^2+4)} \tag{54}$$

Impulse response:

$$h(t) = \underline{c}'\,e^{\underline{A}t}\underline{b} = 2e^{-t} - 2\cos 2t + \sin 2t \tag{55}$$

Cost functional for the optimal linear regulator problem:

$$J[\underline{x}_0,\, u(\cdot)] = \int_0^2 [\,y^2(t) + u^2(t)]\,dt$$

$$= \int_0^2 \{\underline{x}'(t) \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \underline{x}(t) + u^2(t)\}\,dt \tag{56}$$

i.e., $$\underline{Q} = \underline{c}\,\underline{c}' = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \underline{R} = [1], \quad \underline{F} = \underline{0}$$

Cost functional for the suboptimal linear regulator problem:

$$\mu[\underline{L}(\cdot)] = \operatorname{tr} \underline{V}(0)$$

$$= \operatorname{tr} \int_0^2 \underline{\Phi}_L'(\tau, 0)[\underline{Q} + \underline{L}'(\tau)\underline{L}(\tau)]\,\underline{\Phi}_L(\tau, 0)\, d\tau \tag{57}$$

The optimal control for this problem is

$$\underline{u}^*(t) = \underline{L}^*(t)\underline{x}(t) \triangleq [\ell_1^*(t) \quad \ell_2^*(t) \quad \ell_3^*(t)]\ \underline{x}(t) \tag{58}$$

where the optimal feedback matrix is obtained by solving the matrix Riccati equation (see Eqs. 3-4). $\underline{L}^*(t)$ was computed using a program developed by A. Levis,[11] and its components are shown as broken lines in Figs. 2-5. The steady-state ($T \to \infty$) values are indicated by arrows.

The suboptimal (piecewise constant) feedback matrix $\underline{L}^o(t) \triangleq [\ell_1^o(t) \quad \ell_2^o(t) \quad \ell_3^o(t)]$ was computed for 1, 2, 4, and 16 subintervals. These solutions are shown in Figs. 2, 3, 4, and 5. The costs for these various cases are given below, together with the optimal cost and the cost incurred by using the steady state gain $\underline{L}^*(-\infty)$.

a. Optimal solution: $\underline{L}(t) = \underline{L}^*(t)$, $0 < t \leq 2$

$$\underline{K}(0) = \begin{bmatrix} .4044 & -.3620 & -.3520 \\ -.3620 & .5600 & .2425 \\ -.3520 & .2425 & .6454 \end{bmatrix}, \quad \mu(\underline{L}^*) = \operatorname{tr} \underline{K}(0) = 1.6097$$

b. Steady-state solution: $\underline{L}(t) = \underline{L}^s = \underline{L}^*(-\infty)$, $0 < t \leq 2$

$$\underline{V}_s(0) = \begin{bmatrix} .4101 & -.3771 & -.3512 \\ -.3771 & .6029 & .2314 \\ -.3512 & .2314 & .6839 \end{bmatrix}, \quad \mu(\underline{L}^s) = \operatorname{tr} \underline{V}_s(0) = 1.6969$$

c. One subinterval: $\underline{L}(t) = \underline{L}^o(0)$, $0 < t \leq 2$ (Fig. 2)

$$\underline{V}_o(0) = \begin{bmatrix} .4090 & -.3729 & -.3504 \\ -.3729 & .6001 & .2297 \\ -.3504 & .2297 & .6643 \end{bmatrix}, \quad \mu(\underline{L}^o) = \text{tr } \underline{V}_o(0) = 1.6735$$

d. Two subintervals: $\underline{L}(t) = \underline{L}^o(i)$, $i < t \leq i+1, i = 0, 1$ (Fig. 3)

$$\underline{V}_o(0) = \begin{bmatrix} .4067 & -.3608 & -.3505 \\ -.3608 & .5620 & .2415 \\ -.3505 & .2415 & .6505 \end{bmatrix}, \quad \mu(\underline{L}^o) = \text{tr } \underline{V}_o(0) = 1.6193$$

e. Four subintervals: $\underline{L}(t) = \underline{L}^o(i)$, $\frac{i}{2} < t \leq \frac{i+1}{2}$, $i=0, 1, 2, 3$ (Fig. 4)

$$\underline{V}_o(0) = \begin{bmatrix} .4054 & -.3613 & -.3513 \\ -.3613 & .5613 & .2417 \\ -.3513 & .2417 & .6490 \end{bmatrix}, \quad \mu(\underline{L}^o) = \text{tr } \underline{V}_o(0) = 1.6157$$

f. Sixteen subintervals: $\underline{L}(t) = \underline{L}^o(i)$, $\frac{i}{8} < t \leq \frac{i+1}{8}$, $i=0, 1, \ldots 15$ (Fig. 5)

$$\underline{V}_o(0) = \begin{bmatrix} .4045 & -.3620 & -.3520 \\ -.3620 & .5699 & .2424 \\ -.3520 & .2424 & .6459 \end{bmatrix}, \quad \mu(\underline{L}^o) = \text{tr } \underline{V}_o(0) = 1.6104$$

Comparing (b) and (c) above, one finds that the best constant matrix on $(0, T]$ provides a slight improvement over using the steady-state solution on this interval, although both cases are within five percent of the optimum cost. Using two subintervals in (d) brings the cost to within one percent of the optimum, and further divisions in (e) and (f) improve upon this only slightly.

One might expect the suboptimal piecewise constant gains to be fairly close to the "average" values of the optimal gains, but it is interesting to note in Figs. 3 and 4 that this is not the case. In Fig. 5 the constant gains begin to converge to the optimal solution, as predicted by Theorem 11 of Ref. 1.

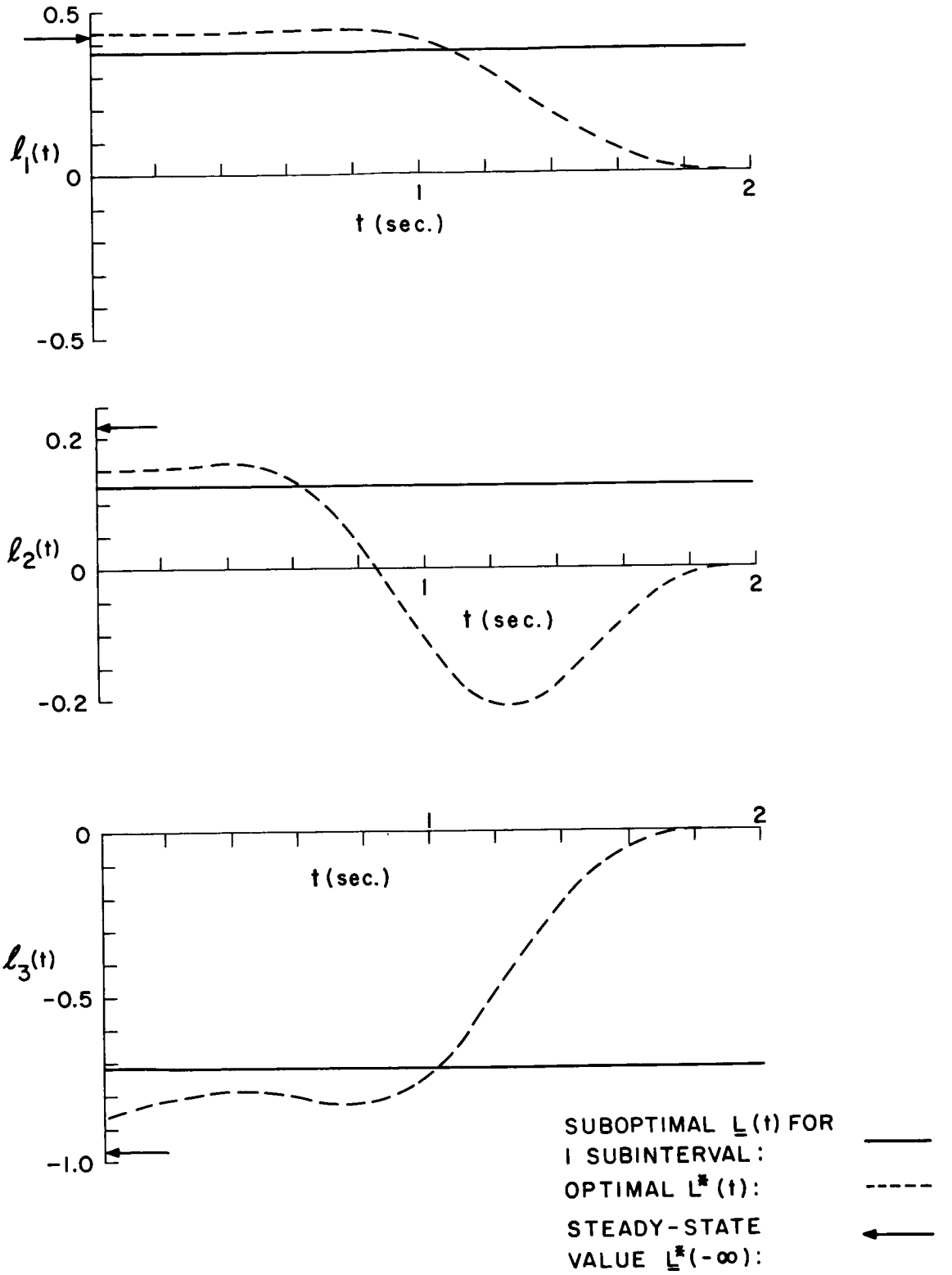

Fig. 2 Feedback Gain Matrix $\underline{L}(t) \triangleq [\ell_1(t) \quad \ell_2(t) \quad \ell_3(t)]$ for One Subinterval

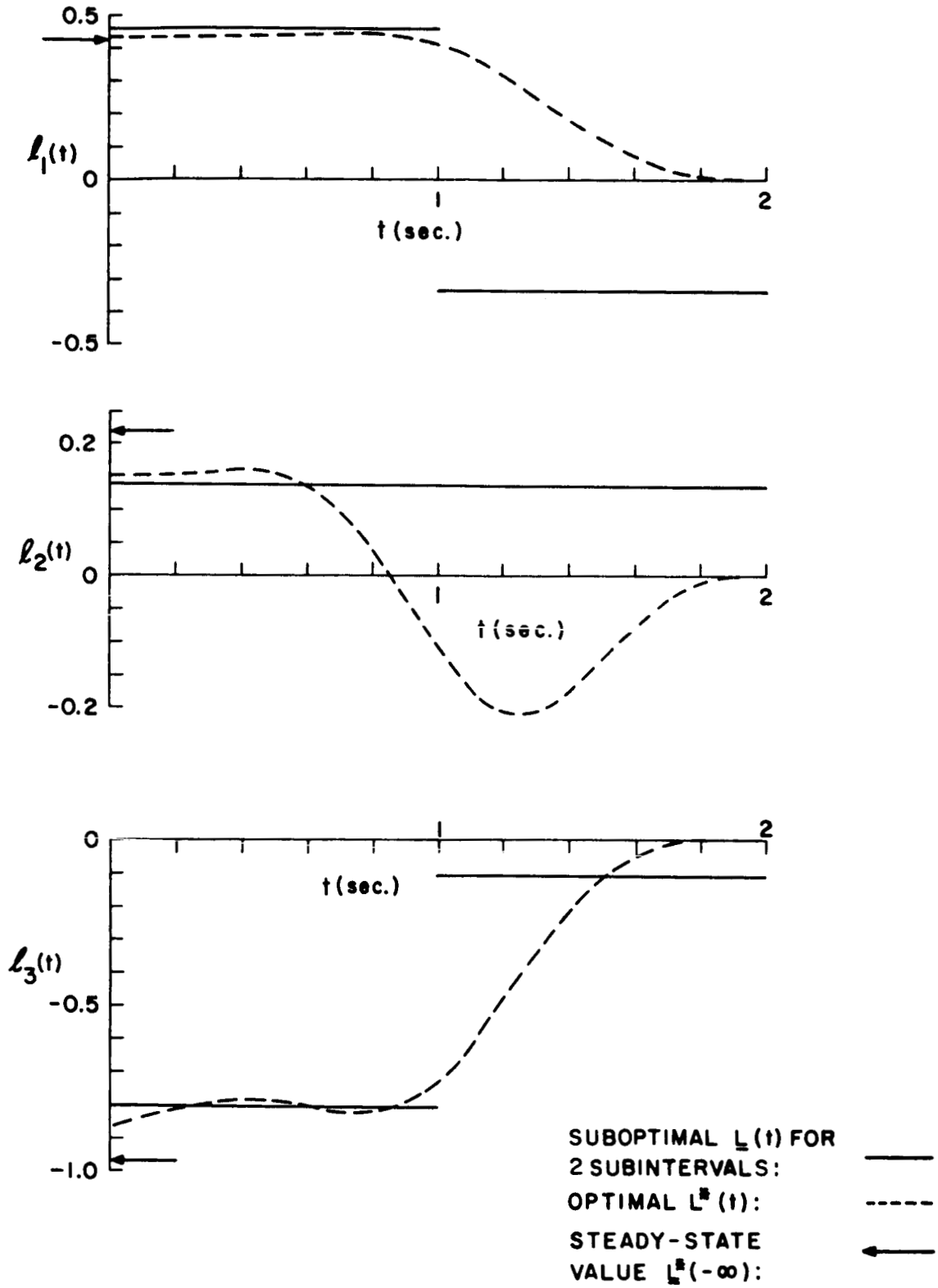

Fig. 3 Feedback Gain Matrix for Two Subintervals

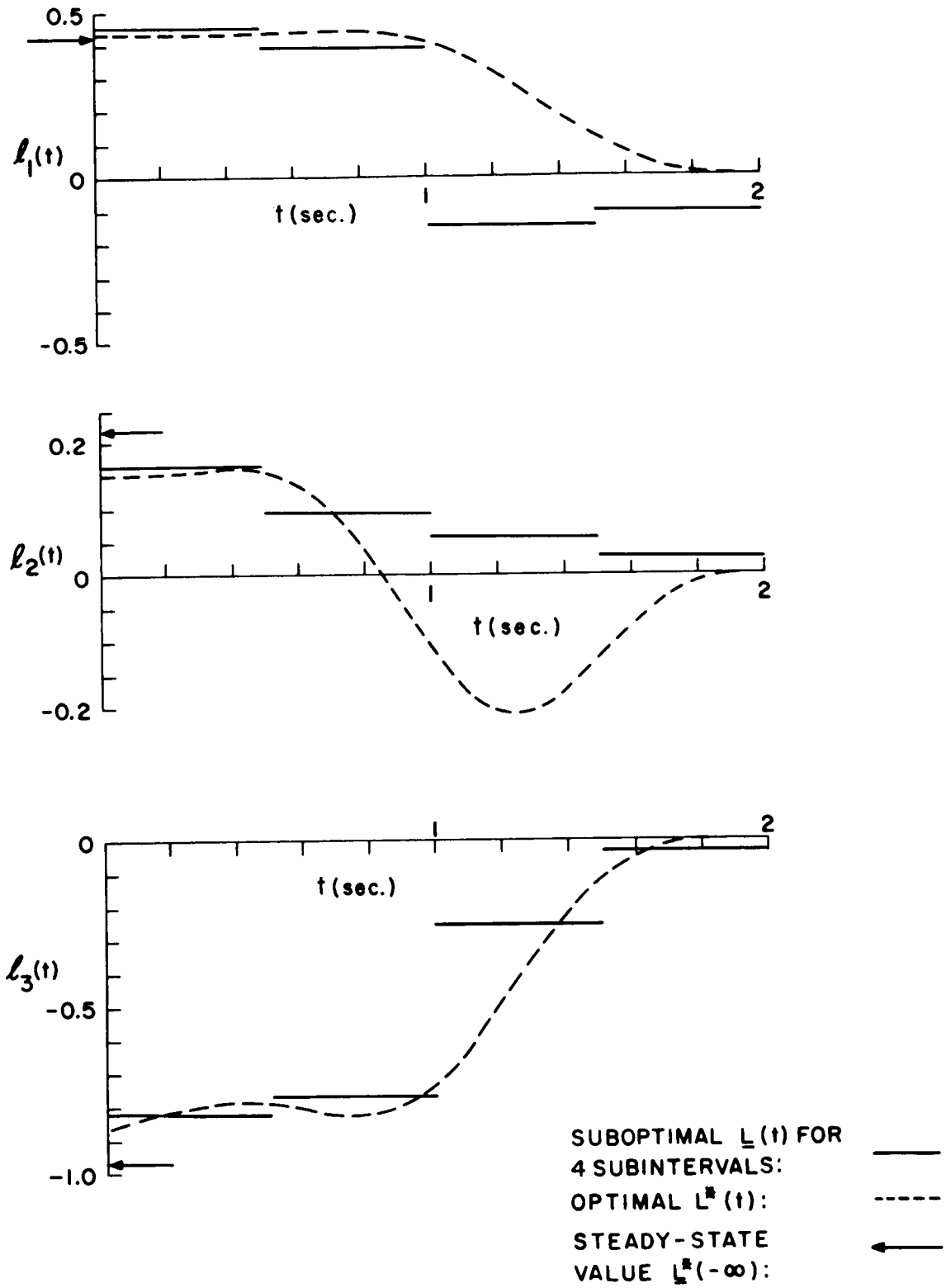

Fig. 4 Feedback Gain Matrix for Four Subintervals

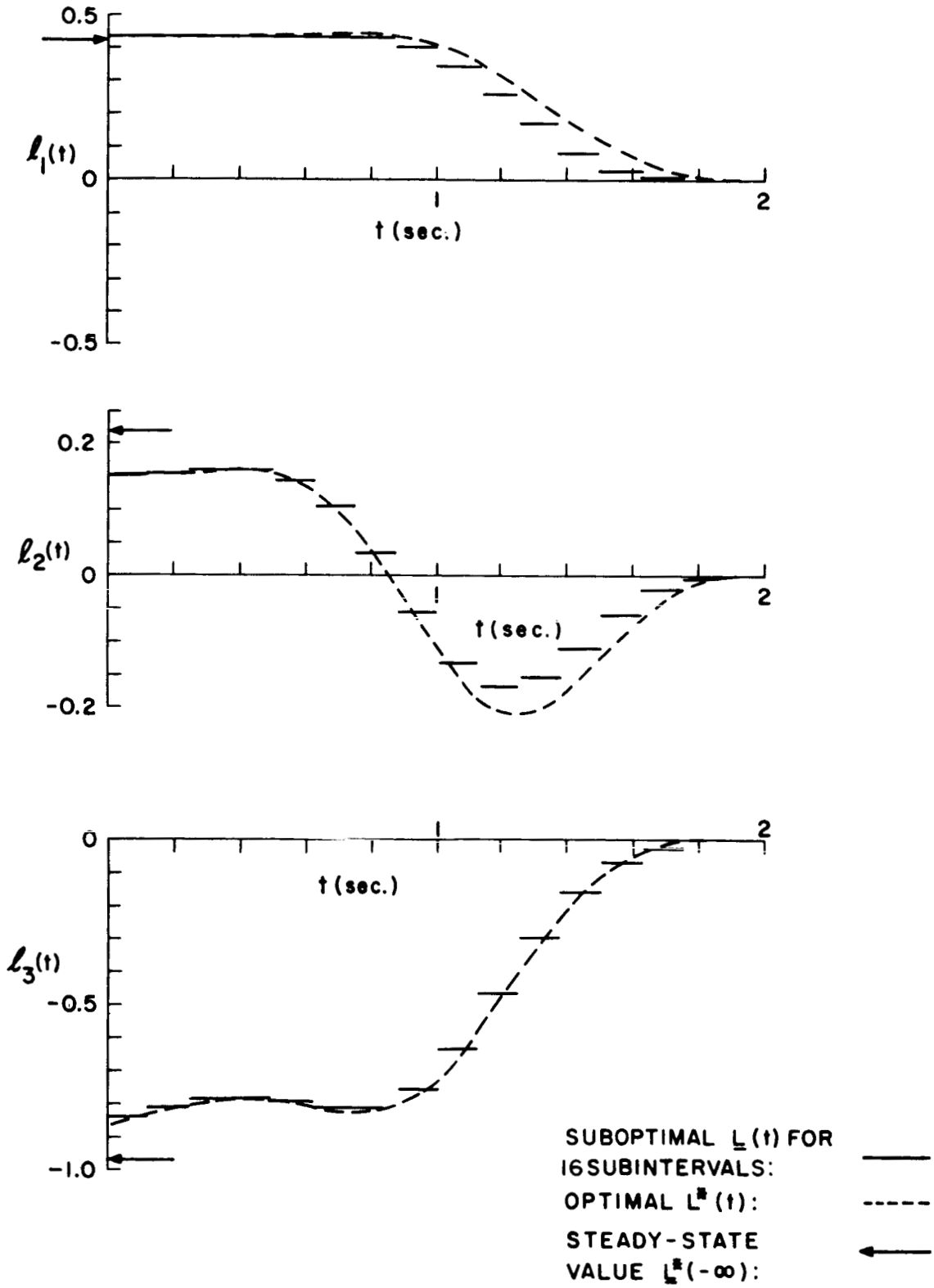

Fig. 5 Feedback Gain Matrix for Sixteen Subintervals

V. CONCLUSIONS

A computer program has been presented which iteratively finds the (sub)optimal feedback gains for a linear regulator system under the constraint that they be piecewise constant. Implementation is greatly simplified by such a constraint, and results obtained for a third-order example indicate that near-optimal performance may be achieved with piecewise-constant gains in the feedback loop.

APPENDIX A

ERROR BOUNDS

In Section III C the matrix exponential $e^{\underline{H}k\delta}$ is computed by raising $e^{\underline{H}\delta}$ to the k-th power. This raises the question of how much error is allowable in $e^{\underline{H}\delta}$ (i.e., how many terms in the series should be kept) in order to maintain a given tolerance on $[e^{\underline{H}\delta}]^k$.

The basic problem is as follows: Suppose $\overline{\underline{A}}$ is the true value of a matrix and the apparent value is

$$\underline{A} = \overline{\underline{A}} + \underline{E} \tag{A.1}$$

where $\underline{E}$ is an error matrix. Then if

$$\underline{A}^n = (\overline{\underline{A}} + \underline{E})^n = \overline{\underline{A}}^n + \underline{E}_n \tag{A.2}$$

what is a bound on the error $\underline{E}_n$? This problem, and the closely related one of bounding the error in

$$e^{\underline{A}t} = e^{(\overline{\underline{A}} + \underline{E})t}$$

arise in a variety of situations.[5] Expanding Eq. A.2 term by term yields

$$\underline{A}^n = \overline{\underline{A}}^n + (\overline{\underline{A}}^{n-1}\underline{E} + \overline{\underline{A}}^{n-2}\underline{E}\,\overline{\underline{A}} + \ldots + \underline{E}\,\overline{\underline{A}}^{n-1}) + \ldots + \underline{E}^n \tag{A.3}$$

which is difficult to evaluate, since $\underline{A}$ and $\underline{E}$ do not commute in general. One way to avoid this difficulty is to use a norm on $\underline{E}_n$ which obeys the Schwarz inequality, i.e.,

$$\|\underline{A}\,\underline{B}\| \leq \|\underline{A}\| \;\; \|\underline{B}\| \tag{A.4}$$

Then from Eq. A.3

$$\|\underline{E}_n\| = \|\underline{A}^n - \overline{\underline{A}}^n\| \leq (\|\overline{\underline{A}}\| + \|\underline{E}\|)^n - \|\overline{\underline{A}}\|^n = n\|\overline{\underline{A}}\|^{n-1}\|E\| + 0(\|\underline{E}\|^2) \tag{A.5}$$

However, this bound is not very useful in the present situation, since $\|\overline{\underline{A}}\|^{n-1}$ diverges for large n whenever $\|\underline{A}\| > 1$, even though $\|\underline{A}^n\| \to 0$ (assuming a stable system).

The problem of bounding $\underline{A}^n$ was investigated at length, but no easy solution was found. One scheme is outlined below for reference, but is not used in the program due to its complexity.

Let $\underline{A}$ be a matrix with distinct eigenvalues* $\lambda_1, \lambda_2, \ldots, \lambda_N$ and eigenvectors $\underline{u}_1, \underline{u}_2, \ldots \underline{u}_N$; let $\underline{v}_1, \underline{v}_2, \ldots \underline{v}_N$ be a reciprocal basis, i.e.,

$$\underline{v}_i' \underline{u}_j = \delta_{ij} = \begin{cases} 1 & i=j \\ 0 & i \neq j \end{cases} \tag{A.6}$$

Then the spectral representation of $\underline{A}$ is (Ref. 6, p. 314)

$$\underline{A} = \sum_{i=1}^{N} \lambda_i \underline{E}_i \triangleq \sum_{i=1}^{N} \lambda_i \underline{u}_i \underline{v}_i' \tag{A.7}$$

and it is clear from multiplying Eq. A.7 by itself that

$$\underline{A}^n = \sum_{i=1}^{N} \lambda_i^n \underline{E}_i \tag{A.8}$$

If the eigenvalues are available,** then the matrices $\underline{E}_i$ may be found quite easily using a result of Morgan.[7] Suppose the transfer function (of the system $\dot{\underline{x}} = \underline{A}\underline{x}$) is

$$\underline{H}(s) = (s\underline{I} - \underline{A})^{-1} = \frac{\underline{B}(s)}{d(s)} \tag{A.9}$$

where $d(s) = \det(sI - \underline{A}) = s^N + d_1 s^{N-1} + \ldots + d_N$ is the characteristic polynomial and $\underline{B}(s) = \underline{B}_0 s^{N-1} + \underline{B}_1 s^{N-2} + \ldots + \underline{B}_{N-1}$ is a matrix polynomial often called the adjoint matrix. Then the $\underline{E}_i$ are given by

---

* This is a reasonable assumption because the matrix entries must be approximated by a finite number of digits in computer storage and an arbitrarily small perturbation of their values will separate any multiple eigenvalues.

** They may be found, for instance, by solving for the roots of the characteristic polynomial d(s), which is available from the recursion formula Eq. A.11.

$$\underline{E}_i = \underline{u}_i \underline{v}_i' = \frac{\underline{B}(\lambda_i)}{tr\underline{B}(\lambda_i)}, \quad i=1,2,\ldots N \qquad (A.10)$$

The coefficients of $\underline{B}(s)$ may be generated with a well-known algorithm given in Zadeh and Desoer:[6]

$$\underline{B}_0 = \underline{I}$$

$$d_k = -(1/k)tr(\underline{B}_{k-1}\underline{A}) \qquad k=1,2,\ldots N \qquad (A.11)$$

$$\underline{B}_k = \underline{B}_{k-1}\underline{A} + d_k\underline{I} \qquad k=1,2,\ldots N-1$$

With the $\underline{E}_i$ and λ_i available, Eq. A.8 suggests an easy way to bound $\underline{A}^n$:

$$|\underline{A}^n| \leq \sum_{i=1}^{N} |\lambda_i|^n |\underline{E}_i| \leq N[\,|\lambda_i|_{max}\,]^n\, |\underline{E}_i|_{max} \qquad (A.12)$$

where the inequalities and absolute values are to be interpreted entry by entry. Assuming the matrix $\underline{A}^n$ to be stable (i.e., $|\lambda_i| < 1$ for all i), this bound does not diverge for large n. An analogous procedure gives a bound for $e^{\underline{A}t}$:

$$|e^{\underline{A}t}| \leq \sum_{i=1}^{N} e^{(Re\lambda_i)t} |\underline{E}_i| \leq N\, e^{(Re\lambda_i)_{max}\, t} |\underline{E}_i|_{max} \qquad (A.13)$$

Here the bound goes to zero for large t if $e^{\underline{A}t}$ is stable (i.e., $Re\lambda_i < 0$ for all i).

The above method for bounding $\underline{A}^n$ is straightforward but tedious to program, and no simpler scheme was found. For this reason, it was not used in the program.

APPENDIX B

USE OF THE PROGRAM

1. INPUT VARIABLES

Quantities which must be assigned values as input are as follows:

MSYS = dimension of system state vector $\underline{x}$

NSYS = dimension of system input vector $\underline{u}$

NTIMES = N, the number of intervals into which the time axis $[0, T]$ is divided

TIMES : array containing $t_0, t_1, \ldots t_N$

EPS = ϵ. See Eq. 17

TOL1, TOL2, TOL3, TOL4 : error tolerances. See Sections III C-F

OUTPARM indicates matrices to be printed out

= 0 : none

= 1 : H(n), HPHI(n)

= 2 : H(n), HPHI(n), HEXP(n)

= 3 : H(n), HPHI(n), HEXP(n), HINT(n)

= 4 : H(n), HPHI(n), HEXP(n), HINT(n), on the first iteration only

ITERMAX : maximum number of iterations

A, B, F, Q, : arrays containing matrices of the same names

L : array containing $\underline{L}(n)$, n=0, 1, ...N-1. Initial values must be supplied

VKEEP : array providing temporary storage for V(t). Initial value (for reference) is Riccati solution at time t_0.

Values for these variables are supplied on input cards. The format is illustrated by an example.

$$\underline{A} = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \quad \underline{B} = \begin{bmatrix} 4 \\ 5 \end{bmatrix} \quad \underline{F} = \begin{bmatrix} 6 & 7 \\ 8 & 9 \end{bmatrix} \quad \underline{Q} = \begin{bmatrix} 10 & 11 \\ 12 & 13 \end{bmatrix}$$

$$t_0 = 0.1,\ t_1 = 0.2,\ t_2 = T = 0.3$$

$$\underline{L}^1(0) = [-1 \quad -2],\ \underline{L}^1(1) = [-3 \quad -4]$$

Solution of Riccati equation at $t_0 = \begin{bmatrix} -5 & -6 \\ -7 & -8 \end{bmatrix}$

The input cards are then as follows:

```
NSYS = 2, MSYS = 1, NTIMES = 2;
EPS = 1, TOL1 = 0.01, TOL 2 = 0.0001, TOL3 = 0.01,
        TOL4 = 1.01, OUTPARM = 4, ITERMAX = 25;
0.1, 0.2, 0.3,
0, 1, 2, 3,
4, 5,
6, 7, 8, 9,
10, 11, 12, 13,
-1, -2,
-3, -4,
-5, -6, -7, -8,
```

2. OTHER VARIABLES

LHAT : array containing $\hat{\underline{L}}(n)$

H: array containing $\underline{H}(n) = \underline{A} - \underline{B}\,\underline{L}(n)$

HPHI : array containing $\underline{\Phi}(t_n, t_0)$

HEXP : array containing $e^{\underline{H}(n)\,\delta_n}$

HINT: array containing $\int_0^{\delta_n} e^{\underline{H}(n)'\tau}[\underline{Q}+\underline{L}(n)'\underline{L}(n)]\,e^{\underline{H}(n)\tau}\,d\tau$

STEPO : array containing $\log_2 M_n$

STEP1 : array containing M_n = number of steps in n-th interval

STEP2 : array containing $\delta_n = (t_{n+1}-t_n)/M_n$

(all for $n = 0, 1, \ldots N-1$)

3. MAIN PROGRAM : SUBOPT

This reads in all the inputs, including the initial values for $\underline{L}(n)$ (Box 1 in Fig. 1), and then calls COMPSTP. Next it calls INTEG once for each interval, n = N-1, N-2, ... 1, 0. Finally, it updates

$\underline{L}(n)$, checks for convergence, and repeats the iteration with smaller ϵ or goes on to the next iteration (Boxes 7-11 in Fig. 1).

4. SUBROUTINES COMPSTP AND INTEG

COMPSTP (Boxes 2-3 in Fig. 1) calculates the step sizes for each interval and the matrices $\underline{\Phi}(t_n, t_0)$, $e^{\underline{H}(n)\delta_n}$, and $[\int_0^{\delta_n} e^{\underline{H}(n)'\tau}$ $[\underline{Q}+\underline{L}(n)'\underline{L}(n)]\, e^{\underline{H}(n)\tau} d\tau]$ for $n=0, 1, 2, \ldots N-1$.

INTEG (Boxes 4-5) computes $\underline{\Phi}(t, t_0)$ and $\underline{V}(t)$ at each of the M_n steps on the n-th interval $[t_n, t_{n+1}]$. Then it performs the two integrations in Eq. 18 and returns a new $\hat{\underline{L}}(n)$.

5. SUBROUTINES MATMULT AND MINV

MATMULT has several entry points, which are just a collection of matrix manipulation and output routines. The listing of this subroutine is self-explanatory.

MINV is a matrix inversion routine which has been translated into PL/1 from the IBM system/360 Scientific Subroutine Package.

6. LISTINGS

On the following pages are listings of the various parts of the program.

```
SUBOPT:     PROCEDURE OPTIONS(MAIN);
   DECLARE (NSYS,MSYS,NTIMES)FIXED;
   BIGLOOP: GET DATA(NSYS,MSYS,NTIMES);
          PUT PAGE EDIT(
    'BEGIN PROCEDURE TO CALCULATE SUBOPTIMAL GAIN MATRICES')(X(20),A);
   PUT SKIP(2) DATA(NSYS,MSYS,NTIMES);
  BEGIN;  DECLARE EPS,TOL1,TOL2,TOL3,TOL4,TEMP,TEMPA,TEMPB,
                  (ITER,ITERX,ITERY,ITERCNT,ITERMAX,OUTPARM,
                   I,J,K,M,N)FIXED,  TIMES(0:NTIMES),
                  (IDENT,A,F,Q,VKEEP,C,D)(NSYS,NSYS)FLOAT,
                  B(NSYS,MSYS), BP(MSYS,NSYS),
                  (L,LHAT)(MSYS,NSYS,0:NTIMES-1,0:1)FLOAT,
                  (H,HPHI,HEXP,HINT)(NSYS,NSYS,0:NTIMES-1),
                  (STEP0,STEP1)(0:NTIMES-1)FIXED, STEP2(0:NTIMES-1),
                  VTRACE(0:1),  NORM RETURNS(FLOAT);
   GET DATA(EPS,TOL1,TOL2,TOL3,TOL4,OUTPARM,ITERMAX);
   PUT SKIP(2)  DATA(EPS,TOL1,TOL2,TOL3,TOL4,OUTPARM,ITERMAX);
   GET LIST(TIMES);  PUT SKIP(2) DATA(TIMES);
   GET LIST(A);  CALL MXOUT(A,'A MATRIX');
   GET LIST(B);  CALL MXOUT(B,'B MATRIX');
   GET LIST(F);  CALL MXOUT(F,'F MATRIX');
   GET LIST(Q);  CALL MXOUT(Q,'Q MATRIX');
   IDENT=0; DO I=1 TO NSYS; IDENT(I,I)=1;  END; HPHI(*,*,0)=IDENT;
   ITER,ITERX,ITERCNT=1; ITERY=0;  CALL TRANSPS(B,BP);
            /* INITIALIZATION OF L MATRICES*/
   DO I=0 TO (NTIMES-1); GET LIST(L(*,*,I,1));  END;
   GET LIST(VKEEP);
   PUT SKIP(3); CALL MXOUT(VKEEP,'RICCATI SOLUTION AT TIME 0');
   TEMP=0; DO I=1 TO NSYS; TEMP=TEMP+VKEEP(I,I); END;
   PUT SKIP(2) EDIT('TRACE OF RICCATI SOLUTION =',TEMP)(A,E(16,5,6));
   CALL MXS(L(*,*,*,1),NTIMES,'INITIAL FEEDBACK MATRICES (L)');
 LOOP:  PUT PAGE EDIT('*   *   *   ITERATION NUMBER ',ITERCNT,
                      '   *   *   *')(X(30),A,F(3),A);
   IF ITER ¬= ITERCNT THEN PUT SKIP LIST('ITER =',ITER);
   IF ITER>1 & OUTPARM>3 THEN OUTPARM=0;
       /*   COMPUTE H = A - B*L    */
   DO I=0 TO (NTIMES-1);
   CALL MATMULT(B,L(*,*,I,ITERX),C);
   H(*,*,I)=A-C;    END;
   IF OUTPARM>0 THEN
   CALL MXS(H,NTIMES,'CLOSED-LOOP SYSTEM MATRICES (H)');
      CALL COMPSTP(VKEEP,ITER,H,HPHI,HEXP,HINT,L,Q,TIMES,OUTPARM,
                   TOL1,TOL2,STEP0,STEP1,STEP2);
   VKEEP=F;
   PUT SKIP(3) LIST('      INTERVAL      ROMBERG INTEGRATION ERRORS');
   DO J= (NTIMES-1) TO 0 BY -1;
      CALL INTEG(J,STEP0,STEP1,VKEEP,BP,HPHI,HINT,HEXP,
                 TIMES,LHAT,ITER,FLUSH);     GO TO NOFLUSH;
 FLUSH:  LHAT(*,*,J,ITERX)=L(*,*,J,ITERX);  NOFLUSH: END;
   CALL MXOUT(VKEEP,'MATRIX V(T=0)');
   VTRACE(ITERX)=0;DO I=1 TO NSYS;
                    VTRACE(ITERX)=VTRACE(ITERX)+VKEEP(I,I); END;
   PUT SKIP(2) LIST('TRACE(V) = ',VTRACE(ITERX));
   IF EPS¬=1 THEN PUT SKIP LIST('EPSILON =',EPS);

   IF ITER =1 THEN GO TO HERE;
   IF VTRACE(ITERX)>TOL4*VTRACE(ITERY) THEN DO;

     ITER=ITER-1;ITERY=ITERX;ITERX=MOD(ITER,2);  EPS=EPS/2;
     PUT SKIP(2) EDIT('THIS ITERATION WILL REPEAT WITH EPS=EPS/2')
                                                    (X(20),A);
     GO TO THERE;   END;
```

```
HERE: DO J=0 TO (NTIMES-1);  DO M=1 TO MSYS;  DO N=1 TO NSYS;
      TEMPA=L(M,N,J,ITERX);  TEMPB=LHAT(M,N,J,ITERX);
      IF TEMPA=0 | TEMPB=0 THEN IF ABS(TEMPA+TEMPB)>TOL3
                                THEN GO TO THERE;ELSE GO TO QUIT;
      IF ABS((TEMPB/TEMPA)-1)>TOL3 THEN GO TO THERE;
      QUIT:  END;   END;   END;    PUT SKIP(8);
  CALL MXS(LHAT(*,*,*,ITERX),NTIMES,
     'THE PROCEDURE HAS CONVERGED TO THE FOLLOWING MATRICES(L)');
  PUT SKIP(8);    GO TO BIGLOOP;

THERE:  IF EPS=1 THEN DO J=0 TO (NTIMES-1);
                       L(*,*,J,ITERY)=LHAT(*,*,J,ITERX);  END;
  ELSE DO J=0 TO (NTIMES-1);
       L(*,*,J,ITERY)=(1-EPS)*L(*,*,J,ITERX)+EPS*LHAT(*,*,J,ITERX);
       END;
  CALL MXS(L(*,*,*,ITERY),NTIMES,'NEW FEEDBACK MATRICES(L)');
  IF ITERCNT>=ITERMAX THEN DO;
    PUT SKIP(6) EDIT('PROCEDURE TERMINATED AT ',ITERCNT,
                                    'ITERATIONS.')(A,F(3),A);
  IF EPS<1 THEN CALL MXS(LHAT(*,*,*,ITERX),NTIMES,
                         'LHAT MATRICES IN FINAL ITERATION WERE');
  PUT SKIP(8);    GO TO BIGLOOP;    END;
  ITER=ITER+1;ITERCNT=ITERCNT+1;ITERY=ITERX;ITERX=MOD(ITER,2);
  GO TO LOOP;
END;    /*FIRST BEGIN STATEMENT*/
END SUBOPT;




COMPSTP:  PROCEDURE(VKEEP,ITER,H,HPHI,HEXP,HINT,L,Q,TIMES,OUTPARM,
                    TOL1,TOL2,STEP0,STEP1,STEP2);
    DECLARE (NTIMES,ITER,NSYS,MSYS,OUTPARM)FIXED, TOL1,TOL2,TIMES(*),
            (STEP0,STEP1)(*)FIXED, STEP2(*), (VKEEP,Q)(*,*),
            (H,HPHI,HEXP,HINT)(*,*,*),  L(*,*,*,*)FLOAT,
            NORM RETURNS(FLOAT);
                    /*COMPUTES STEP SIZES, CALLS COMPEXP, COMPINT,
                       AND COMPUTES PHI AT ENDS OF INTERVALS*/
    NSYS=DIM(L,2); MSYS=DIM(L,1); NTIMES=HBOUND(TIMES,1);
    BEGIN;
    DECLARE (FDOT,MAX,NORMV,NORMH,NORMPHI)FLOAT,LP(NSYS,MSYS)FLOAT,
            LHOLD(MSYS,NSYS)FLOAT, (C,D)(NSYS,NSYS),
            (I,J,K,ITERX)FIXED;
    NORMV=NORM(VKEEP);  IF NORMV<1 THEN NORMV=1;
    ITERX=MOD(ITER,2);
   DO J=0 TO (NTIMES-1);
    NORMH=NORM(H(*,*,J));    NORMPHI=NORM(HPHI(*,*,J));
    LHOLD=L(*,*,J,ITERX);
    CALL TRANSPS(LHOLD,LP);
    CALL MATMULT(LP,LHOLD,C);
    C=C+Q;
    FDOT=4*NORMH*NORMPHI**2*(NORMH*NORMV+NORM(C));
    MAX=SQRT(FDOT*(TIMES(J+1)-TIMES(J))**3/(12*TOL1));
    IF MAX >=512  THEN DO;      I=512;     K=9;
    PUT SKIP(2) EDIT('INDICATED NUMBER OF STEPS FOR INTERVAL ',J,' IS '
          ,MAX,'.   512 STEPS WERE USED.')(A,F(3),A,F(6),A);
```

```
      GO TO HERE;      END;
    I=16;   K=4;
    DO WHILE (I<MAX);  I=I*2;  K=K+1;    END;
  HERE:  STEP0(J)=K; STEP1(J)=I; STEP2(J)=(TIMES(J+1)-TIMES(J))/I;
    CALL COMPINT;
    CALL COMPEXP;
    IF J<(NTIMES-1) THEN DO;
    K=1;   C=HEXP(*,*,J);
    DO WHILE (K<I);      K=2*K;
      CALL MATMULT(C,C,D);  C=D;      END;
    CALL MATMULT(C,HPHI(*,*,J),HPHI(*,*,J+1));
    END;     /*  IF STATEMENT  */
   END;   /* END OF DO LOOP */
    PUT SKIP(3) LIST('      INTERVAL      SUBINTERVALS      INCREMENT');
    DO J=0 TO (NTIMES-1);
      PUT SKIP EDIT(J,STEP1(J),STEP2(J))(X(6),F(3),X(12),F(4),
                                          X(10),E(12,5,6));   END;
    IF OUTPARM>0 THEN CALL MXS(HPHI,NTIMES,'HPHI MATRICES');
    IF OUTPARM>1 THEN CALL MXS(HEXP,NTIMES,'HEXP MATRICES');
    IF OUTPARM>2 THEN CALL MXS(HINT,NTIMES,'HINT MATRICES');
    RETURN;

COMPINT:  PROCEDURE;              /*COMPUTES HINT ON INTERVAL J*/
   DECLARE       (SUM,TERM,HN,HP,TEMP1,TEMP2)(NSYS,NSYS),  (K,M,N)FIXED;
    TERM=     C*STEP2(J);      SUM=TERM;
    HN=H(*,*,J)*STEP2(J);      CALL TRANSPS(HN,HP);
    DO K=2 TO NSYS;
    CALL MATMULT(HP,TERM,TEMP1);  CALL MATMULT(TERM,HN,TEMP2);
    TERM=(TEMP1+TEMP2)/K;    SUM=SUM+TERM;        END;
  LOOP:  K=K+1;
    CALL MATMULT(HP,TERM,TEMP1);  CALL MATMULT(TERM,HN,TEMP2);
    TERM=(TEMP1+TEMP2)/K;    SUM=SUM+TERM;
    DO M=1 TO NSYS;    DO N=1 TO NSYS;
      IF SUM(M,N)=0 THEN IF ABS(TERM(M,N))>TOL2 THEN GO TO LOOP;
                         ELSE GO TO QUIT;
      IF ABS(TERM(M,N)/SUM(M,N))>TOL2 THEN GO TO LOOP;
    QUIT:  END;  END;
    HINT(*,*,J)=SUM;
 /*COMPINT*/          END;

COMPEXP: PROCEDURE;      /*COMPUTES E TO THE H(J)T FOR INTERVAL J  */
    DECLARE  (K,M,N)FIXED,(HT,SUM,TERM,TEMP)(NSYS,NSYS);
    SUM,TERM=HPHI(*,*,0);      /* = IDENTITY MATRIX*/
    HT=H(*,*,J)*STEP2(J);
    DO K=1 TO NSYS;
      CALL MATMULT(HT,TERM,TEMP);
      TERM=TEMP/K;  SUM=SUM+TERM;           END;
  LOOP:  K=K+1;    CALL MATMULT(HT,TERM,TEMP);
    TERM=TEMP/K;    SUM=SUM+TERM;
    DO M=1 TO NSYS;    DO N=1 TO NSYS;
      IF SUM(M,N)=0 THEN IF ABS(TERM(M,N))>TOL2 THEN GO TO LOOP;
                         ELSE GO TO QUIT;
      IF ABS(TERM(M,N)/SUM(M,N))>TOL2 THEN GO TO LOOP;
    QUIT:  END;  END;
    HEXP(*,*,J)=SUM;
 /*COMPEXP*/      END;
   END;      /*FIRST BEGIN*/
 /*COMPSTP*/      END;
```

```
INTEG:  PROCEDURE(J,STEP0,STEP1,VKEEP,BP,HPHI,HINT,HEXP,
                  TIMES,LHAT,ITER,FLUSH);
    DECLARE (J,NSYS,MSYS,ITER,MM)FIXED,(STEP0,STEP1)(*)FIXED, TIMES(*),
            VKEEP(*,*), BP(*,*), (HINT,HPHI,HEXP)(*,*,*),
            LHAT(*,*,*,*)FLOAT, FLUSH LABEL, NORM RETURNS(FLOAT);
          /* DOES ALL COMPUTATIONS AND INTEGRATIONS TO GET LHAT
                                ON INTERVAL J    */
    MM=STEP1(J); NSYS=DIM(LHAT,2);  MSYS=DIM(LHAT,1);
    BEGIN;  DECLARE (PHI,V)(NSYS,NSYS,0:MM), DET, I FIXED,
                    (C,D,E)(NSYS,NSYS), LHOLD(MSYS,NSYS)FLOAT;

      /* COMPUTE PHI ON INTERVAL J    */
    PHI(*,*,0)= HPHI(*,*,J);       C= HEXP(*,*,J);
    DO I=1 TO MM; CALL MATMULT(C,PHI(*,*,I-1),PHI(*,*,I)); END;

      /*  COMPUTE V ON INTERVAL J     */
    V(*,*,MM)=VKEEP;              CALL TRANSPS(HEXP(*,*,J),E);
    DO I=(MM-1) TO 0 BY -1;
      CALL MATMULT(E,V(*,*,I+1),D);
      V(*,*,I)=HINT(*,*,J);
      CALL MULTADD(D,C,V(*,*,I));          END;
    VKEEP = V(*,*,0);

       /*    ROMBERG INTEGRATION OF THE INTEGRALS C AND D   */
    BEGIN;
    DECLARE(K,M,N)FIXED, Z, (TEMP,SAVET,SAVES)(NSYS,NSYS),
           (S,T)(0:STEP0(J),NSYS,NSYS);
    M=STEP0(J);     Z= (TIMES(J+1)-TIMES(J))/2;
    CALL TRANSQR(PHI(*,*,0),TEMP);    T(0,*,*)=TEMP;
    CALL MATMULT(V(*,*,0),TEMP,S(0,*,*));
    CALL TRANSQR(PHI(*,*,MM),TEMP);
    T(0,*,*)=Z*(T(0,*,*)+TEMP);
    CALL MULTADD(V(*,*,MM),TEMP,S(0,*,*));  S(0,*,*)=Z*S(0,*,*);
    DO K=1 TO M;
    IF K=M THEN DO; SAVET=T(0,*,*); SAVES=S(0,*,*);  END;
    I=2**(M-K);    T(K,*,*),S(K,*,*)=0;
    DO N=I TO (MM-I) BY (2*I);
      CALL TRANSQR(PHI(*,*,N),TEMP);      T(K,*,*)= T(K,*,*)+TEMP;
      CALL MULTADD(V(*,*,N),TEMP,S(K,*,*));          END;
    T(K,*,*) = T(K-1,*,*)/2 + Z*T(K,*,*);
    S(K,*,*) = S(K-1,*,*)/2 + Z*S(K,*,*);
    Z=Z/2;
    DO N=1 TO K;
      T(K-N,*,*) = ((4**N)*T(K-N+1,*,*) - T(K-N,*,*)) / (4**N-1);
      S(K-N,*,*) = ((4**N)*S(K-N+1,*,*) - S(K-N,*,*)) / (4**N-1);  END;
    END;
       C=T(0,*,*);       D=S(0,*,*);
    SAVET=ABS(SAVET-C);       SAVES=ABS(SAVES-D);
    PUT SKIP EDIT(J,NORM(SAVES),NORM(SAVET))(X(6),F(3),
                                    X(10),E(9,2,3),X(1),E(9,2,3));
    END;       /*      END OF ROMBERG   */

    E=C;    CALL MINV(C,DET);
    IF DET=0 THEN DO;
      CALL MXOUT(E,'RESULT OF ROMBERG INTEGRATION WAS SINGULAR');
      PUT SKIP;  GO TO FLUSH;  END;
    CALL MATMULT(D,C,E);     I=MOD(ITER,2);
    CALL MATMULT(BP,E,LHOLD);  LHAT(*,*,J,I)=LHOLD;
    END;    /*FIRST BEGIN*/
  END INTEG;
```

```
MATMULT: PROCEDURE(A,B,C);  /*C=A*B, MATRIX INDICES START WITH 1 */
    DECLARE (I,J,K,NTIMES,N)FIXED,  (A,B,C)(*,*),D(*,*,*),
            NAME CHARACTER(80)VARYING;
    DO I=1 TO DIM(A,1);
    DO J=1 TO DIM(B,2);
      C(I,J)= SUM(A(I,*)*B(*,J)); END;  END; RETURN;
MULTADD:  ENTRY(A,B,C);   /* C = C + A*B  */
    DO I= 1 TO DIM(A,1);
    DO J= 1 TO DIM(B,2);
      C(I,J)=C(I,J)+SUM(A(I,*)*B(*,J)); END;  END; RETURN;
TRANSPS:  ENTRY(A,B);   /* B = A TRANSPOSED  */
    DO I=1 TO DIM(A,1);
    DO J=1 TO DIM(A,2);
    B(J,I)=A(I,J);   END;   END;     RETURN;
TRANSQR:   ENTRY(A,B);    /* B = A*A'  */
    DO I=1 TO DIM(A,1);
      B(I,I)= SUM(A(I,*)*A(I,*));
      DO J=1 TO (I-1);
        B(I,J),B(J,I)= SUM(A(I,*)*A(J,*));  END;  END; RETURN;
NORM:   ENTRY(A) FLOAT;   /* RETURNS NORM OF A  */
    RETURN(SQRT(SUM(A*A))/DIM(A,1));
MXOUT:   ENTRY(A,NAME); /*PRINTS NAME AND MATRIX*/
    IF NAME ¬= 'NULL' THEN PUT SKIP(2) LIST(NAME);
    N=DIM(A,2);     DO I=1 TO DIM(A,1);
                    PUT SKIP EDIT((A(I,J) DO J=1 TO N))(E(16,5,6));
                    END;       RETURN;
MXS:  ENTRY(D,NTIMES,NAME);  /*PRINTS SEVERAL MATRICES*/
    IF NAME ¬= 'NULL' THEN PUT SKIP(2) LIST(NAME);
    N=DIM(D,2);
    DO K=0 TO (NTIMES-1); PUT SKIP(2) EDIT('INTERVAL',K)(A,F(3));
      DO I=1 TO DIM(D,1);
        PUT SKIP EDIT((D(I,J,K) DO J=1 TO N))(E(16,5,6));  END;  END;
  END MATMULT;
```

```
MINV: PROCEDURE(B,D);  /*RETURNS WITH D= DET B, B = B INVERSE  */
    DECLARE B(*,*),D,BIGA,HOLD,
              (N,I,J,K,IJ,IK,IZ,JI,JK,JP,JQ,JR,KI,KJ,KK,NK)FIXED;
    N=DIM(B,1);
  BEGIN;  DECLARE (L,M)(N),A(N**2);
    DO J=1 TO N; DO I=1 TO N; A(J*N-N+I)=B(I,J);   END;   END;
          /*BEGIN SSP SUBROUTINE*/
    D=1;   NK=-N;
  DO K = 1 TO N;
    NK=NK+N;   L(K),M(K)=K;   KK=NK+K;   BIGA=A(KK);
    DO J=K TO N;   IZ=N*(J-1);
      DO I=K TO N;     IJ=IZ + I;
        IF ABS(BIGA)<ABS(A(IJ)) THEN DO;
          BIGA=A(IJ);   L(K)=I;   M(K)=J;   END;      END;    END;
                      /*   LINE 20   */
    J=L(K);
    IF J>K THEN DO;   KI=K-N;
      DO I=1 TO N;  KI=KI+N;   HOLD=-A(KI);
        JI=KI-K+J;   A(KI)=A(JI);   A(JI)=HOLD;  END;   END;
```

```
          /*     LINE  35    */
    I=M(K);
    IF I>K THEN DO;    JP=N*(I-1);
      DO J=1 TO N;  JK=NK+J;    JI=JP+J;
        HOLD=-A(JK);    A(JK)=A(JI);    A(JI)=HOLD;     END;     END;
                /*     LINE  45     */
    IF BIGA=0 THEN DO;  D=0;  RETURN;    END;
    DO I=1 TO N;     IF I ¬= K THEN DO;
      IK=NK+I;    A(IK)=A(IK)/(-BIGA);     END;    END;
                   /*    LINE  55       */
    DO I=1 TO N;   IK=NK+I;   IJ=I-N;
      DO J=1 TO N;  IJ=IJ+N;    IF I¬=K & J¬=K THEN DO;
        KJ=IJ-I+K;    A(IJ)=A(IK)*A(KJ)+A(IJ);     END;     END;     END;
                  /*       LINE65       */
    KJ=K-N;
    DO J=1 TO N;  KJ=KJ+N;
      IF J ¬= K THEN A(KJ)=A(KJ)/BIGA;          END;
    D=D*BIGA;       A(KK)=1/BIGA;
  END;                  /*     LINE80     */
    K=N;
  L100:   K=K-1;
    IF K>0 THEN DO;    I=L(K);
      IF I>K THEN DO;    JQ=N*(K-1);    JR=N*(I-1);
        DO J=1 TO N;    JK=JQ+J;    HOLD=A(JK);
          JI=JR+J;    A(JK)=-A(JI);    A(JI)=HOLD;      END;    END;
      J=M(K);  IF J<=K THEN GO TO L100;  KI=K-N;
      DO I=1 TO N;    KI=KI+N;    HOLD=A(KI);
        JI=KI-K+J;    A(KI)=-A(JI);    A(JI)= HOLD;    END;
      GO TO L100;
    END;
          /*END OF SSP SUBROUTINE    */
    DO J=1 TO N;  DO I=1 TO N;  B(I,J)=A(J*N-N+I);     END;    END;
  END;
 /*MINV*/       END;
```

REFERENCES

1. Kleinman, D. L., "Suboptimal Design of Linear Regulator Systems Subject to Computer Storage Limitations," Report ESL-R-297, M.I.T., February, 1967.

2. Athans, M. and Falb, P. L., Optimal Control: An Introduction to the Theory and Its Applications, McGraw-Hill, 1966.

3. Athans, M. and Schweppe, F. C., "Gradient Matrices and Matrix Calculus," Technical Note 1965-53, M.I.T. Lincoln Laboratory, November, 1965.

4. Ralston, A., A First Course in Numerical Analysis, McGraw-Hill, 1965.

5. Levis, A., "Error Bounds in Some Matrix Calculations," Research Note 1967-2, Control Theory Group, Electronic Systems Lab., M.I.T., May, 1967.

6. Zadeh, L. A. and Desoer, C. A., Linear System Theory, McGraw-Hill, 1963.

7. Morgan, B. S., Jr., "Sensitivity Analysis and Synthesis of Multivariable Systems," IEEE Transactions on Automatic Control, Vol. AC-11, No. 3, July 1966, p. 506.

8. Hamming, R. W., Numerical Methods for Scientists and Engineers, McGraw-Hill, 1962.

9. Wilkinson, J. H., Rounding Errors in Algebraic Processes, Prentice-Hall, 1964.

10. Berezin, I. S., and Zhidkov, N. P., Computing Methods, Pergamon Press, 1965.

11. Levis, A., "Optimal Control of Linear Sampled-Data Systems," Sc. D. Thesis, Dept. of Mechanical Engineering, M.I.T., (to appear).