IN-63

131949

p_ 57

NASA
Technical
Memorandum

NASA TM-108384

(NASA-TM-108384) OPTIMAL CONTROL COMPUTER PROGRAMS (NASA) 57 p

N93-13289

Unclas

G3/63 0131949

OPTIMAL CONTROL COMPUTER PROGRAMS

By F. Kuo

Structures and Dynamics Laboratory
Science and Engineering Directorate

November 1992

NASA

National Aeronautics and
Space Administration

George C. Marshall Space Flight Center

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 1992	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE Optimal Control Computer Programs	5. FUNDING NUMBERS
6. AUTHOR(S) F. Kuo	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM-108384

11. SUPPLEMENTARY NOTES

Prepared by Structures and Dynamics Laboratory, Science and Engineering Directorate.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified—Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The solution of the optimal control problem, even with low order dynamical systems, can usually strain the analytical ability of most engineers. The understanding of this subject matter, therefore, would be greatly enhanced if a software package existed that could simulate simple generic problems. Surprisingly, despite a great abundance of commercially available control software, few, if any, address the part of optimal control in its most generic form. The purpose of this paper is, therefore, to present a simple computer program that will perform simulations of optimal control problems that arise from the first necessary condition and the Pontryagin's maximum principle.

14. SUBJECT TERMS Optimal Control, First Necessary Condition, Maximum Principle, Dynamic Programming	15. NUMBER OF PAGES 59
	16. PRICE CODE NTIS

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TECHNICAL MEMORANDUM

OPTIMAL CONTROL COMPUTER PROGRAMS

I. INTRODUCTION

The theory of optimal control can be better appreciated if simulation tools are available in the classroom to supplement class notes. Most optimal control problems, even with low-order systems, are not amenable to easy analytical solutions. Therefore, a generic simulation program, if it existed, could greatly improve one's knowledge of the theory. Although commercially available control software is in great abundance, few, if any, address the control problem in its most fundamental form, i.e., the first necessary condition (FNC) and the maximum principle. The intent of this report is to fill this need for a simple simulation tool which would be able to simulate most optimal control problems based on these two principles.

II. STATEMENT OF THE OPTIMAL CONTROL PROBLEM

The optimal control problem can be simply stated as: Find an admissible control u^* which causes the system

$$\dot{x}(t) = f(x(t),u(t),t) \ , \tag{1}$$

to follow an admissible trajectory x^* and minimizes the performance measure

$$J = G(x(T),T) + \int_{t_0}^{T} L(x,t,u)\, dt \ . \tag{2}$$

In general, additional constraints may be placed on the control variable u^* and the system variable x^*.

Variations in the performance index J lead to many different types of optimal control problems. A few typical problems are described briefly in the following sections.

A. Minimum Time Problem

Given the time t_0 and the initial state $x(t_0) = x^0$, the final state is to lie in a specified region S of the $n\times 1$ dimensional state-time space. The objective is to transfer a system from the initial state x^0 to the specified target set S in the minimum time. The performance to be minimized is therefore

$$J = t_1 - t_0 = \int_{t_0}^{t_1} dt \ , \tag{3}$$

where t_1 is the first instant of time when $x(t)$ and S intercept.

B. Minimum Energy Problem

The objective of this problem is to transfer a system from a given initial state $x(t_0) = x^0$ to a specified target set S with a minimum expenditure of energy. The performance index in this case will then be

$$J = \int_{t_0}^{t_1} \{u^T(t)Ru(t)\}dt \ , \tag{4}$$

where R is a positive definite constant matrix.

C. State Regulator Problem

The objective is to transfer a system from the initial state $x(t_0) = x^0$ to the desired state x^d with the minimum integral square error. Relative to the desired x^d, the quantity $(x(t)-x^d)$ can be viewed as the instantaneous system error. If the system coordinates are transformed such that x^d becomes the origin, then the new state $x(t)$ is itself the error.

For the state regulator problem, a useful performance measure is therefore:

$$J = \frac{1}{2} x^T(T)Hx(T) + \frac{1}{2}\int_{t_0}^{T} \{x^T(t)Qx(t)+u^T(t)Ru(t)\}dt \ , \tag{5}$$

where Q is a constant matrix not required to be positive semidefinite.[1]

If the terminal time is not constrained, $(T\rightarrow\infty)$, then $x^d = 0$ (assuming a stable system). In which case, the performance index will be

$$J = \frac{1}{2}\int_{t_0}^{\infty} \{x^T(t)Qx(t)+u^T(t)Ru(t)\}dt \ . \tag{6}$$

An extension to the state regulator problem is the output regulator problem, where the state error is replaced by output error $y(t)$, then

$$J = \frac{1}{2}\int_{t_0}^{\infty} \{y^T(t)Qy(t)+u^T(t)Ru(t)\}dt \ . \tag{7}$$

D. Tracking Problem

The objective of the tracking problem is to maintain the system state $x(t)$ as close as possible to the desired state $r(t)$ in the interval $[t_0,T]$. Therefore;

$$J = \frac{1}{2} e^T(T)He(T) + \frac{1}{2}\int_{t_0}^{T} \{e^T(t)Qe(t)+u^T(t)Ru(t)\}dt \ , \tag{8}$$

where

$$e(t) = (x(t)-r(t)) \tag{9}$$

III. SOLVING OPTIMAL CONTROL PROBLEM

Solutions to the optimal control problem can be categorized into three separated approaches. The first one is based on the calculus of variation from which the FNC is derived. The second one is based on the Pontryagin's maximum (minimum) principle when the control effort is constrained. The third effort is based on the concept of dynamic programming and the principle of optimality.

The theoretical background of these approaches is discussed in most optimal control textbooks and, therefore, is not discussed in this paper. The procedure of these methods, however, is the foundation of the numerical methods.

A. FNC Algorithm

The FNC algorithm can be summarized in the following; for the plant

$$\dot{x} = f(x,t,u);\ f = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}, \tag{10}$$

and the performance index

$$J = G(x(T),T) + \int_{t_0}^{T} L(x,t,u)dt \ , \tag{11}$$

and the boundary conditions $(x(t_0),t_0)$ given $(x(T),T) \in \mathfrak{I}$, where u is unbounded, piecewise continuous scalar control and G,L are real-valued, sufficiently smooth scalar functions, $\mathfrak{I}$ is the given m-dimensional "terminal manifold." Then, the FNC for the optimal control $u^0(t)$ and the associated optimal trajectory $x^0(t)$ can be determined by the following procedure:

(a) Define Hamiltonian H:

$$H = H(x,p,t,u) \overset{\text{def}}{=} \sum_i p_i f_i(x,t,u) - L(x,t,u) \ . \tag{12}$$

(b) The first integral condition is

$$\left.\frac{\partial H}{\partial u}\right|_{u=u^0} = 0 \rightarrow u^0 = u^0(x,p,t) \ . \tag{13}$$

(c) Define:

$$H^0(x,p,t) \overset{\text{def}}{=} H(x,p,t,u^0(x,p,t)) \ . \tag{14}$$

(d) Form the Euler-Lagrange equations:

$$x_i^0 = \frac{\partial H^0}{\partial p_i} \;, \tag{15a}$$

$$p_i^0 = \frac{-\partial H^0}{\partial x_i} \;. \tag{15b}$$

(e) At terminal time T^0, the transversality condition

$$\left[H^0(T^0) - \frac{\partial G}{\partial t}\right] dt|_{T^0,X^0} - \left[\sum_i p_i(T^0) + \frac{\partial G}{\partial x_i}\right] dx_i|_{T^0,X^0} = 0 \;, \tag{16}$$

must be satisfied for every perturbation $(dt, dx_1, ..., dx_n)$ in the tangent plane to $\Im$ at the point $(X^0(T^0),T^0)$. This procedure leads to $2n+1$ conditions with $2n+1$ unknowns.

Example 1

$$\dot{x} = ax+u \;,$$

$$J = x(T) + \int_{t^0}^{T} u^2(t)dt \;,$$

$$\Im = \begin{cases} T & = \text{fixed;} \\ x(T) & = \text{free;} \\ x(t_0) & = \text{given .} \end{cases}$$

Solution:

$$H = pax+pu-u^2 \;,$$

$$\frac{\partial H}{\partial u} = p-2u = 0 \Rightarrow u^0 = \frac{p}{2} \;.$$

Then the optimal Hamiltonian becomes;

$$H^0 = pax + \frac{p^2}{4} \;.$$

From the Euler-Lagrange equations, the following state and co-state equations were derived:

$$\dot{x} = \frac{\partial H^0}{\partial p} = ax + \frac{1}{2}p \;,$$

$$\dot{p} = -\frac{\partial H^0}{\partial x} = -pa \;.$$

Applying TV condition to the problem results in the following boundary conditions:

$$\{x(t_0) = \text{given},\ p_1(T) = -1\ .\ T = \text{given}\}\ .$$

This is a two-point boundary condition problem that can be solved by one of the programs developed in this paper.

B. Pontryagin's Minimum Principle

In the case of the closed and bounded control region, the optimal control $u^0(x,p,t)$ is found by minimizing $H(x,u,p,t)$ with respect to controls u in the given controls region U, while treating the other variables as constants. In other words, $u^0(x,p,t)$ is the admissible control vector for which $H(x,u,p,t)$ has its minimum value. The minimum principle procedure for optimal control can be summarized as:

For the plant

$$\dot{x} = f(x,t,u);\ f = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}, \tag{17}$$

and the performance index

$$J = G(x(T),T) + \int_{t_0}^{T} L(x,t,u)dt\ , \tag{18}$$

and the boundary conditions $(x(t_0),t_0,x(T))$ given $u \in U$ for all $t \in [t_0,T]$.

(a) Form the Hamiltonian

$$H = H(x,p,t,u) \overset{\text{def}}{=} \sum_i p_i f_i(x,t,u) - L(x,t,u)\ . \tag{19}$$

(b) Find the optimal control u when it is not saturated

$$\left.\frac{\partial H}{\partial u}\right|_{u=u^0} = 0 \rightarrow u^0 = u^0(x,p,t)\ . \tag{20}$$

(c) Then the optimal control $u^0(t)$ is

$$u^0(t) = \begin{cases} U & \text{for } u^0 > U; \\ u^0(t) & \text{for } |u^0| < U; \\ -U & \text{for } u^0 < -U\ . \end{cases} \tag{21}$$

(d) Solve the set of $2n$ equations

$$\dot{x}_i^0 = \frac{\partial H^0}{\partial p_i} , \tag{22a}$$

$$\dot{p}_i^0 = \frac{-\partial H^0}{\partial x_i} . \tag{22b}$$

<u>Example 2</u>

Plant

$$\dot{x}_1(t) = x_2(t) ,$$

$$\dot{x}_2(t) = -x_2(t)+u(t) .$$

The performance index to be minimized is

$$J = \frac{1}{2}\int_{t_0}^{t_1} (x_1^2+u^2)dt .$$

The control constraints are given by

$$|u(t)| \le 1 \quad \text{for } t \in [t_0,t_1] .$$

The Hamiltonian in this case is

$$H(x,u,p,t) = \frac{1}{2}x_1^2 + \frac{1}{2}u^2+p_1x_2-p_2x_2+p_2u .$$

To determine the control that minimizes H subject to the inequality constraints, we first separated all of the terms containing $u(t)$

$$\frac{1}{2}u^2+p_2u .$$

When the control is unsaturated, we have

$$\frac{\partial H}{\partial u} = 0 \Rightarrow u^*(t) = -p_2 .$$

Thus, the optimal control in this case is

$$u^*(y) = \begin{cases} -1 & \text{for } p_2(t) > 1; \\ -p_2 & \text{for } |p_2| < 1; \\ +1 & \text{for } p_2(t) < -1 . \end{cases}$$

Therefore, in order to determine $u(t)$ explicitly, state and co-state equations must be solved subject to the given boundary conditions.

C. Dynamic Programming Approach[3]

The dynamic programming approach to optimal control was derived by Bellman (1962) from his principle of optimality. The algorithm for the linear time-invariant (LTI) system is derived in this section. Let us consider a discrete LTI system described by the matrix-vector difference equation

$$x_{k+1} = Ax_k + Bu_k \ . \tag{23}$$

We would like to find a sequence of control vectors, $u_0, u_1, \ldots u_{N-1}$, to minimize a performance index or cost function

$$J = \frac{1}{2} x_N^T H x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k \ , \tag{24}$$

where H and Q are symmetric $n \times n$ matrices, R is an $m \times m$ positive definite matrix, and N is a fixed integer. Then the cost at the end point N is

$$J_{N,N} = \frac{1}{2} x_N^T H x_N = \frac{1}{2} x_N^T P_N x_N \ . \tag{25}$$

It is obvious from the definition that $P_N = H$. The cost over the last two points is

$$J_{N-1,N} = \frac{1}{2} x_{N-1}^T Q x N-1 + \frac{1}{2} u_{N-1}^T R u_{N-1} + \frac{1}{2} x_N^T P_N x_N \ , \tag{26}$$

and the minimum cost at this point is

$$J_{N-1,N}^0 = \min_{u_{N-1}} J_{N-1,N} \ . \tag{27}$$

After substituting the state equation $x_n = Ax_{N-1} + Bu_{N-1}$ into equation (26), the new equation becomes

$$J_{N-1,N}^0 = \min_{u_{N-1}} \left\{ \frac{1}{2} x_{N-1}^T Q x_N - 1 + \frac{1}{2} u_{N-1}^T R u_{N-1} + \frac{1}{2} (Ax_{N-1} + Bu_{N-1})^T P_N (Ax_{N-1} + Bu_{N-1}) \right\} \ , \tag{28}$$

minimizing this with respect to u_{N-1} gives

$$\frac{\partial J_{N-1,N}}{\partial u_{N-1}} = 0 = u_{N-1}^T R + (Ax_{N-1} + Bu_{N-1}^T) P_N B \ . \tag{29}$$

Therefore, the optimum control effort at $N-1$ stage is

$$u_{N-1} = -(R + B^T P_N B)^{-1} B^T P_N A x_{N-1} \ . \tag{30}$$

Clearly this is negative feedback control which is proportional to the system state or

$$u_{N-1} = -F_{N-1} x_{N-1} \ . \tag{31}$$

Repeating the same procedure backward in stages, a recursive algorithm can be developed as in the following:

$$F_{k-1} = \left[R + B^T P_k B\right]^{-1} B^T P_k A \ , \tag{32}$$

$$P_{k-1} = A^T P_k A - F_{k-1}^T (R + B^T P_k B) F_{k-1} + Q \ . \tag{33}$$

This algorithm is implemented in the computer program.

IV. COMPUTER SOLUTION OF OPTIMAL CONTROL PROBLEM

A. Continuous Time Systems

The two-point boundary value problem derived from the FNC can be solved by several algorithms. The steepest descent method, Fletcher-Powell method, and the shooting method are a few very common ones. In this report, the steepest descent method was implemented because of its ease and simplicity. The algorithm of the maximum descent method in this application is shown in figure 1. In order to understand this approach, it is best to illustrate it with an example.

Example 3

Plant (state equations):

$$\dot{x}_1(t) = -2[x_1(t)+0.25] + [x_2(t)+0.5] \exp\left[\frac{25x_1(t)}{x_1(t)+2}\right] - [x_1(t)+0.25]u(t) \ ,$$

$$\dot{x}_2(t) = 0.5 - x_2(t) - [x_2(t)+0.5] \exp\left[\frac{25x_1(t)}{x_1(t)+2}\right] \ ,$$

$$x(0) = [0.05 \ \ 0]^T \ , \ R = 0.1 \ .$$

Performance index:

$$J = \int_0^{0.78} \left[x_1^2(t) + x_2^2(t) + 0.1u^2(t)\right] dt \ .$$

The Hamiltonian is:

$$H(x(t),u(t),p(t)) = x_1^2 + x_2^2 + Ru^2 + p_1(t)\left[-2[x_1(t)+0.25] + [x_2(t)+0.5] \exp\left[\frac{25x_1(t)}{x_1(t)+2}\right] - [x_1(t)+0.25]u(t)\right]$$

$$+ p_2(t)\left[0.5 - x_2(t) - [x_2(t)+0.5] \exp\left[\frac{25x_1(t)}{x_1(t)+2}\right]\right] \ .$$

Co-state equations:

$$\dot{p}_1(t) = -2x_1(t)+2p_1(t)-p_1(t)[x_2(t)+0.5]\left[\frac{50}{[x_1(t)+2]^2}\right]\exp\left[\frac{25x_1(t)}{x_1(t)+2}\right]$$

$$+p_1(t)u(t)+p_2(t)[x_2(t)+0.5]\left[\frac{50}{[x_1(t)+2]^2}\right]\exp\left[\frac{25x_1(t)}{x_1(t)+2}\right]$$

$$\dot{p}_2(t) = -2x_2(t)-p_1(t)\exp\left[\frac{25x_1(t)}{x_1(t)+2}\right]+p_2(t)\left\{1+\exp\left[\frac{25x_1(t)}{x_1(t)+2}\right]\right\} .$$

The optimal control $u^*(t)$ was calculated from:

$$\frac{\partial H}{\partial u} = 0.2u(t)-p_1(t)[x_1(t)+0.25] = 0 .$$

The norm used was

$$\left\|\frac{\partial H}{\partial u}\right\|^2 = \int_0^{0.78}\left[\frac{\partial H}{\partial u}\right]^2 dt , \qquad (34)$$

and the iteration procedure is terminated when

$$\left\|\frac{\partial H}{\partial u}\right\|^2 < 10^{-2} . \qquad (35)$$

In the steepest descent method, the step size τ is determined by checking the performance J at each major iteration. If J calculated in the current iteration is greater than the previous iteration, the τ is halved, and the calculation is repeated before moving on to the next iteration. The results of the simulation employing the steepest descent method are shown in figures 2 through 4. The optimal control u^* and state trajectories $x_1^*(t), x_2^*(t)$ and their intermediate values are given to show convergence. Table 1 shows the convergence comparison for various initial guesses on $u(0)$ and on τ.

This was a problem with free terminal conditions. Another problem with fixed terminal conditions can be demonstrated by using the same program.

Example 4

Plant model:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = u$$

Performance index:

$$J = \frac{1}{2}x(T)^T Hx(T) + \frac{1}{2}\int_0^T u(t)^2 dt .$$

The terminal conditions being:

$$x(0) = [1\ 1]^T \ ;\ x(T) = [0\ 0]^T \ ;\ T = 10 \ .$$

The co-state equations can be found from the Hamiltonian and are

$$\dot{p}_1 = 0$$

$$\dot{p}_2 = -p_1 \ .$$

The optimal control $u(t)^*$ was calculated from:

$$\frac{\partial H}{\partial u} = u(t) + p_2(t) = 0 \ .$$

The norm used was

$$\left\| \frac{\partial H}{\partial u} \right\|^2 = \int_0^{10} \left[\frac{\partial H}{\partial u} \right]^2 dt \ .$$

The termination criterion remained the same when the norm reached 10^{-2}. The two cases of H were simulated to demonstrate the effect of weighing on the terminal conditions. Figures 5 through 11 show the results of this simulation.

A special case of the continuous time optimal control problem is the linear quadratic regulator (LQR) problem. The LQR problem was solved by Kalman[4] and can be summarized in the following:

For the plant

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) \ ;\ x(t_0) = x^0 \ , \tag{36}$$

and the performance index is

$$J = \frac{1}{2} x^T(t_1) H x(t_1) + \frac{1}{2} \int_{t_0}^{t_1} \{ x^T(t) Q(t) x(t) + u^T(t) R(t) u(t) \} dt \ , \tag{37}$$

the optimal control law is

$$u^*(t) = K(t)x(t) \ , \tag{38}$$

where the feedback gain is

$$K(t) = -R^{-1}(t) B^T(t) P(t) \ , \tag{39}$$

$$-\dot{P}(t) = Q(t) - P(t)B(t)R^{-1}(t)B^T(t)P(t) + P(t)A(t) + A^T(t)P(t) \ , \tag{40}$$

$$P(t_1) = H \ , \tag{41}$$

the optimal performance index is

$$J^* = \frac{1}{2} x^T(t)P(t)x(t) \ . \tag{42}$$

This procedure is developed for generic, time-varying LQR problems. A subclass of this is the steady-state LTI LQR. In computational terms, this can be accomplished by integrating the P matrix until steady state is reached. The following example (Kirk) will serve to illustrate this technique. For the plant model:

$$\dot{x}_1(t) = x_2(t) \ ,$$

$$\dot{x}_2(t) = 2x_1(t) - x_2(t) + u(t) \ ,$$

$$J = \int_0^T \left[x_1^2(t) + \frac{1}{2} x_2^2(t) + \frac{1}{4} u^2(t)\right] dt \ .$$

The A, B, Q, and R matrices are then

$$A = \begin{pmatrix} 0 & 1 \\ 2 & -1 \end{pmatrix}; \ B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \ Q = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}; \ R = \frac{1}{2} \ .$$

The results of the simulations are shown in figures 12 and 13.

B. The Discrete Time LQR Problem (5)

Consider the discrete time dynamic system described by the vector-matrix difference equation

$$x(k+1) = Ax(k) + Bu(k) \ , \ k = 0,1,\ldots,N-1 \ . \tag{43}$$

The objective is to find a sequence of control vectors, $u(0),u(1),\ldots,u(N-1)$, to minimize the following performance index:

$$J = \frac{1}{2} x^T(N)Hx(N) + \sum_{k=0}^{N-1} \frac{1}{2} x^T(k)Qx(k) + \frac{1}{2} u^T(k)Ru(k) \ . \tag{44}$$

The derivation of the feedback control law can be found in reference 3. The recurring feedback gain calculation is summarized in the following:

$$u(k) = -F(k)x(k) \ , \tag{45}$$

$$F(k) = R^{-1}B^T(A^T)^{-1}(P(k)-Q) \ , \tag{46}$$

$$G(k+1) = P(k+1) - P(k+1)B\left[B^TP(k+1)B+R\right]^{-1} B^TP(k+1) \ , \tag{47}$$

$$P(k) = A^TG(k+1)A + Q \ . \tag{48}$$

The terminal condition is

$$P(N) = H \ . \tag{49}$$

Given the terminal condition, we can evaluate $G(N)$, $P(N-1)$, and $F(N-1)$ and continue cyclically until the gain matrix is evaluated at all points in time to yield $F(0),F(1),F(2),\ldots,F(N-1)$.

As an illustrative example, let us consider a second-order discrete system with the following plant dynamics and performance index:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.6277 & 0.3597 \\ 0.0899 & 0.8526 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0.025 \\ 0.115 \end{bmatrix} u(k) \ ,$$

$$J = \frac{1}{2} \sum_{k=0}^{9} x^T(k)Qx(k)+Ru^2(k) \ ,$$

$$Q = \begin{bmatrix} 50 & 0 \\ 0 & 10 \end{bmatrix} , \quad R = 1 \ .$$

Since there is no terminal penalty, then

$$P_{10} = H = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} .$$

The discrete feedback gain and the optimal state trajectories are shown in figures 14 through 16.

C. Dynamic Programming Approach to Optimal Control

The theory of dynamic programming was introduced by Bellman (1957). Although the theory was primarily developed for the solution of certain problems by a digital computer (which implies discrete-time data), it has been extended to continuous time analysis.

Consider a process described by the state equations

$$x(k+1) = f(x(k),u(k),k) \ ; \ k \in [0, N-1] \ . \tag{50}$$

We shall be interested in selecting the control $u(k)$; $k = 0,1,\ldots,N$-1 which minimizes a performance function of the form

$$J = h(x(N),N) + \sum_{k=0}^{N-1} g(x(k),u(k),k) \ . \tag{51}$$

This process is called the multistage decision process of N stages where the choice of $u(k)$ at each sample instant is considered the decision of interest. A recurring relationship for the optimal decision $u(k)$ is

$$u(k) = -F(k)x(k) \ , \tag{52}$$

$$F(k-1) = [R+B^TP(k)B]^{-1}B^TP(k)A \ , \tag{53}$$

$$P(k-1) = A^TP(k)A-F^T(k-1)(R+B^TP(k)B)F(k-1)+Q \ , \tag{54}$$

$$P(N) = H \ . \tag{55}$$

A very important result of the dynamic programming approach is the idea of obtaining the total cost over the entire stages, and it is given as:

$$J^0_{0,N} = \frac{1}{2}x^T(0)P(0)x(0) \ . \tag{56}$$

Using the same example as in section B, the feedback gain and the state trajectories calculated by using this algorithm are shown in figures 17 through 19.

The method of dynamic programming can be extended to a continuous-time system based on the multistage decision process as described in reference 2. A natural approach is to replace the continuous-time problem by its finite difference approximation. Then the results of equations (50) to (56) can be readily applied. It is of interest to compare the results obtained from the discrete LQR with that from the dynamic programming approach. Figures 20 through 24 show the comparison of the state trajectories, optimal feedback gains, and the optimal control effort.

V. CONCLUSION

In this report, several computer programs were developed that provide simulation tools for a large class of optimal control problems. The programs written are very simple in nature. They are all based on the manipulations of a few subroutines for matrix operations. Although the numerical examples shown were of second-order systems, there is no reason why a higher-order system cannot be simulated by appropriate changes in array dimensions. It is hoped that these programs will be of value to engineers to develop a "feel" for some optimal control problems at hand.

REFERENCES

1. Johnson, C.D.: “Limits of Propriety for Linear-Quadratic Regulator Problems.” Int. J. of Control, vol. 45, No. 5, 1987.

2. Gopal, M.: “Modern Control System Theory.” John Wiley and Sons, 1984.

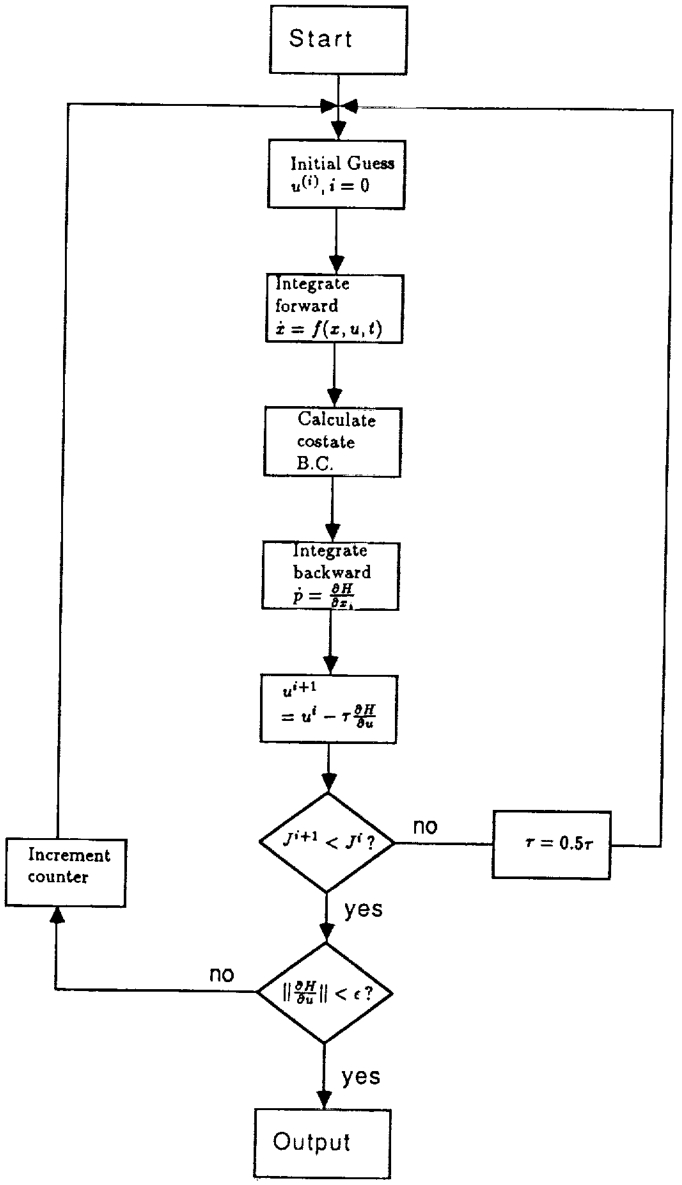
3. Kirk, D.E.: “Optimal Control Theory.” Prentice-Hall, 1970.

4. Kalman, R.E.: “Contribution to the Theory of Optimal Control.” Bol. Soc. Mat., Maxicana, vol. 5, 1960, pp. 102–119.

5. Jacquot, R.G.: “Modern Digital Control System.” Dekker, 1981.

Table 1. Convergence comparison for steepest descend.

Initial Control	Initial τ	Number of Iterations	Minimum J	Final τ	Stopping Criterion
1.0	5.0	8	0.03166	0.3125	Norm
1.0	1.0	40	0.03184	0.25	Norm
1.0	0.25	92	0.03185	0.125	Norm
–1.0	0.25	161	0.03186	0.0625	Norm
–1.0	1.0	28	0.03180	0.5	Norm
0	5.0	6	0.03163	0.3125	Norm



Start
Initial Guess $u^{(i)}, i = 0$
Integrate forward $\dot{x} = f(x,u,t)$
Calculate costate B.C.
Integrate backward $\dot{p} = \frac{\partial H}{\partial x_i}$
$u^{i+1} = u^i - \tau\frac{\partial H}{\partial u}$
$J^{i+1} < J^i$?
no
$\tau = 0.5\tau$
yes
$\|\frac{\partial H}{\partial u}\| < \epsilon$?
no
Increment counter
yes
Output

Figure 1. Steepest descend algorithm for two-point boundary value problem.

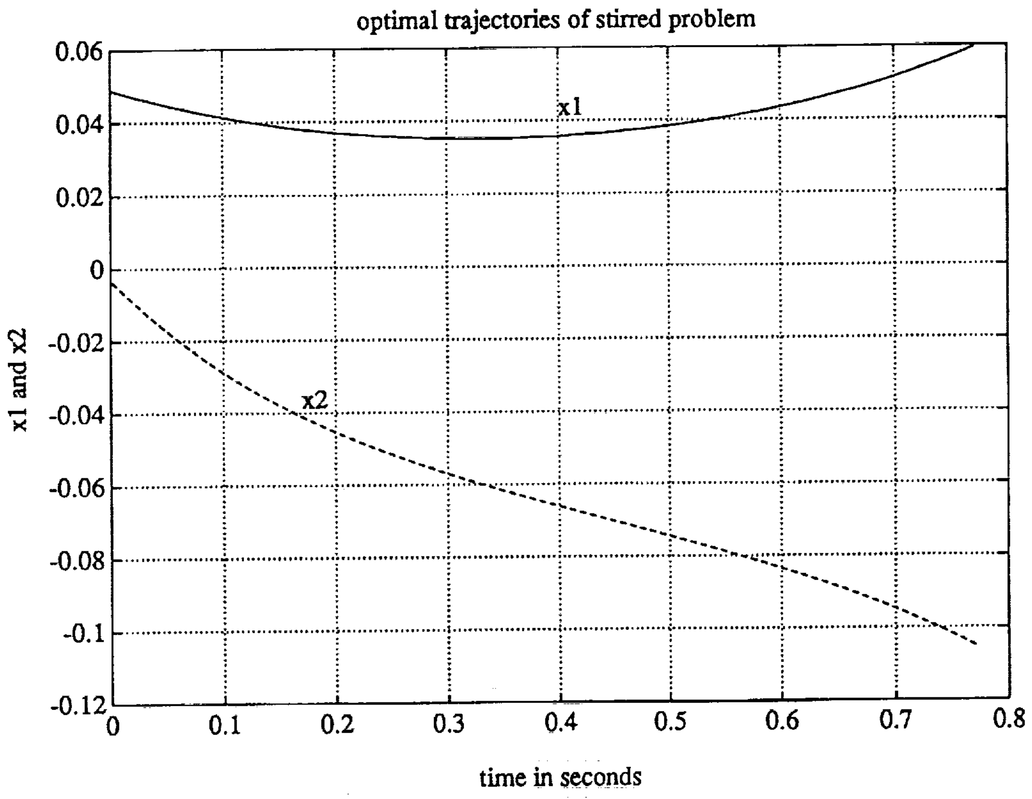


Figure 2. Optimal trajectories of stirred tank problem.

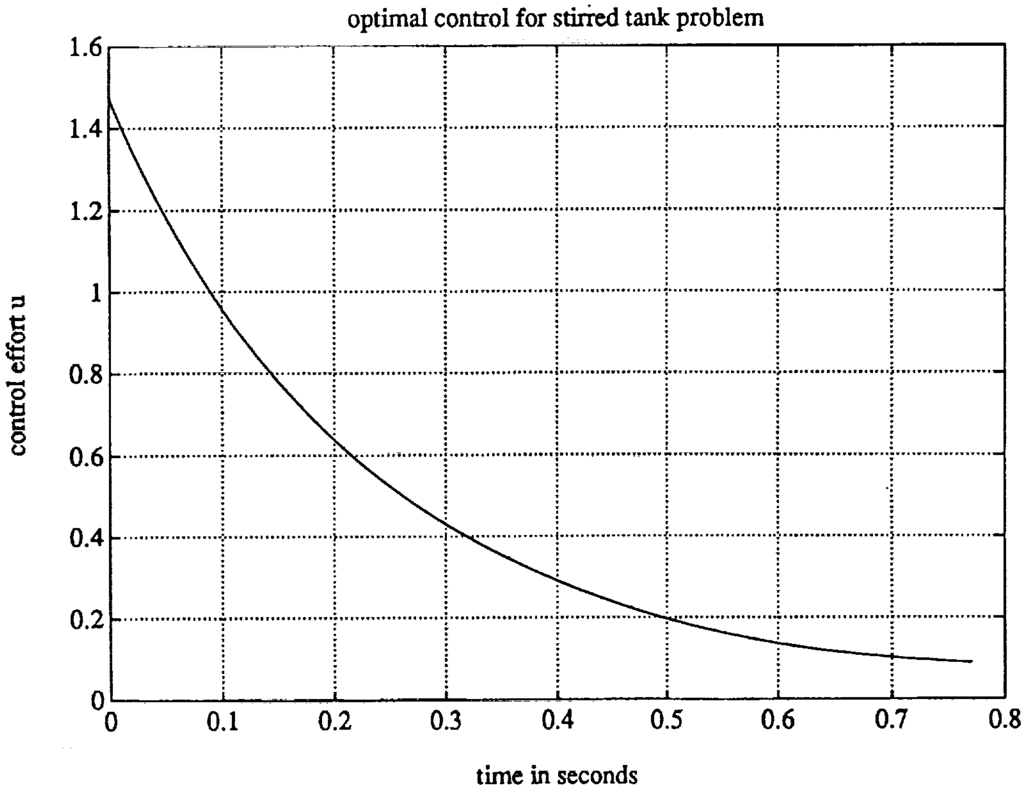


Figure 3. Optimal control effort for stirred tank problem.

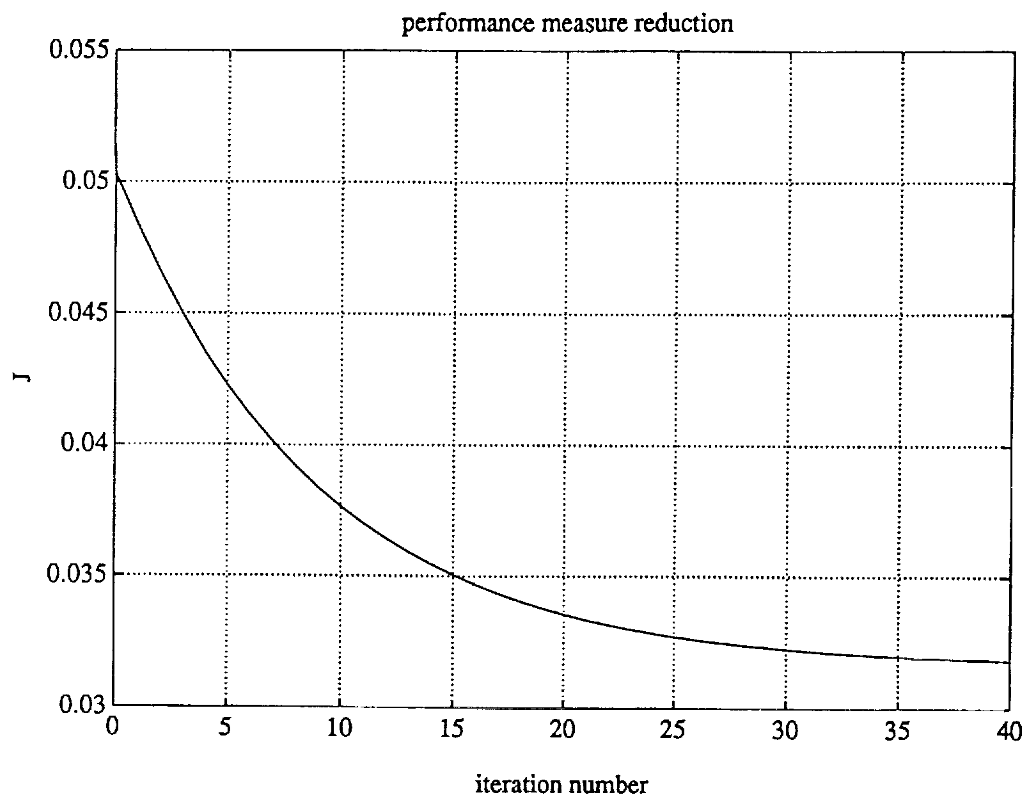


Figure 4. Performance measure reduction versus number of iteration.

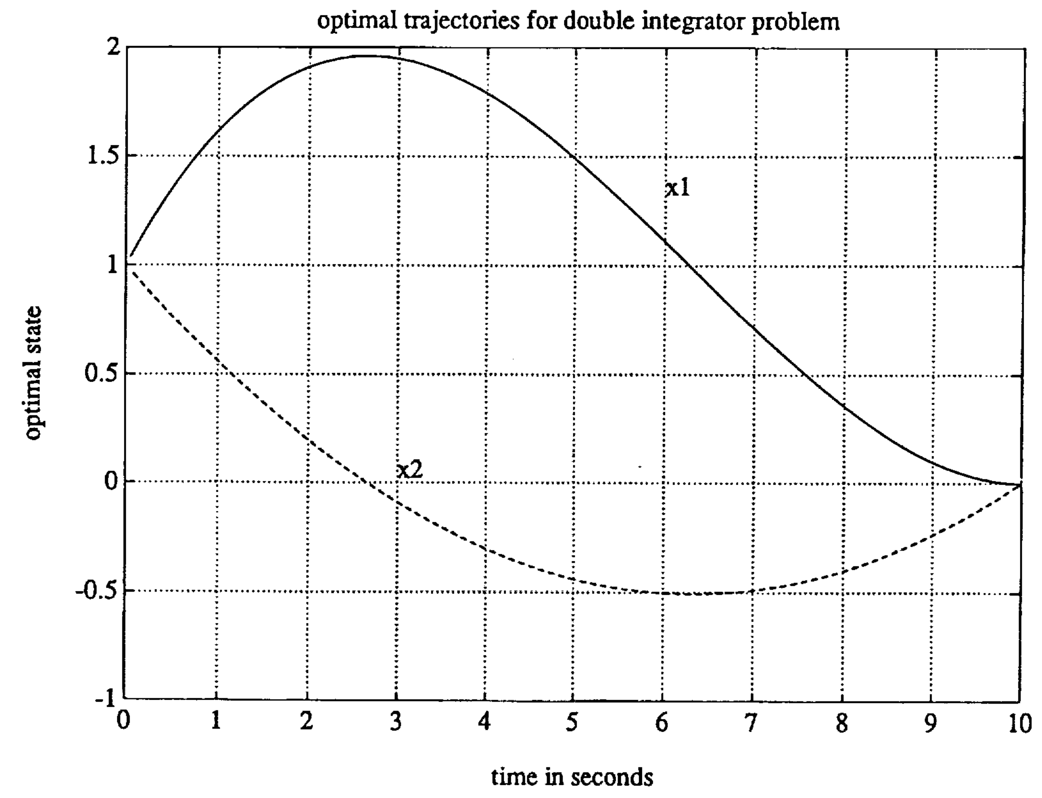


Figure 5. Optimal trajectories for double integrator problem.

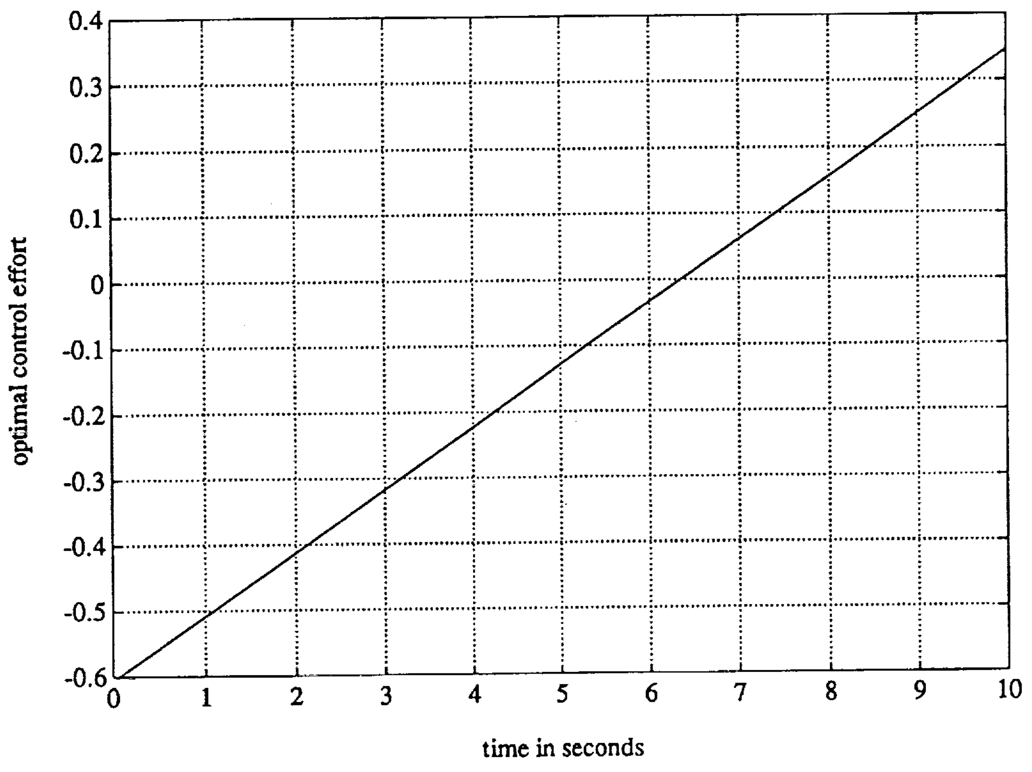


Figure 6. Optimal control effort for double integrator problem.

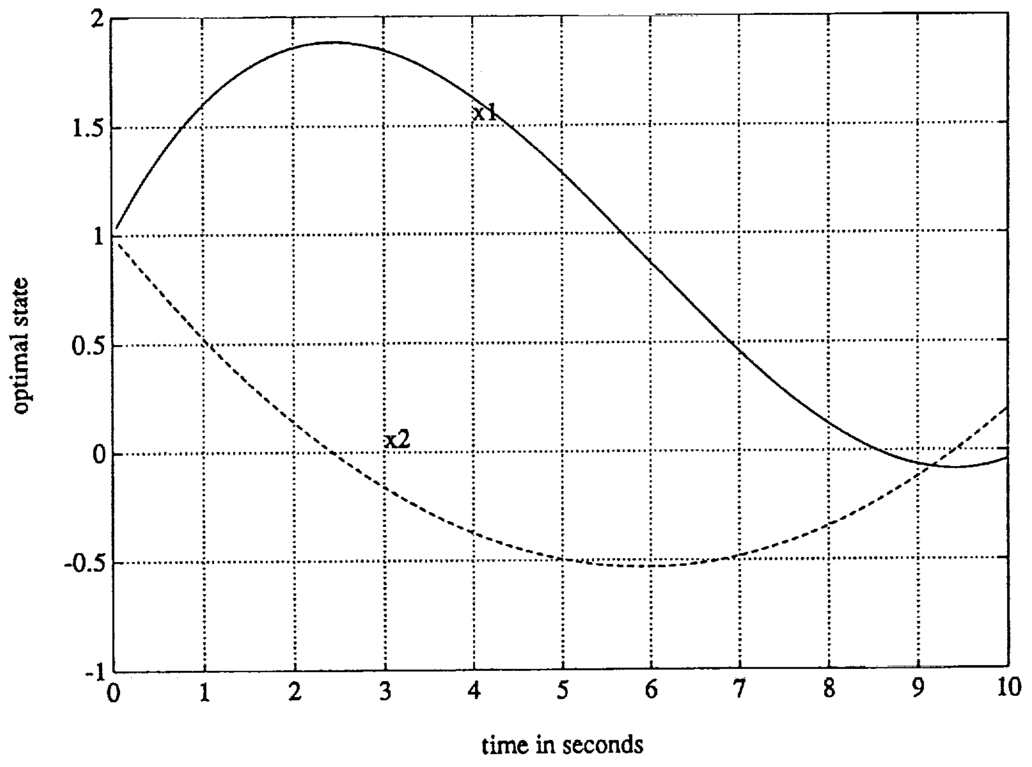


Figure 7. Optimal trajectories with relaxed terminal penalty.

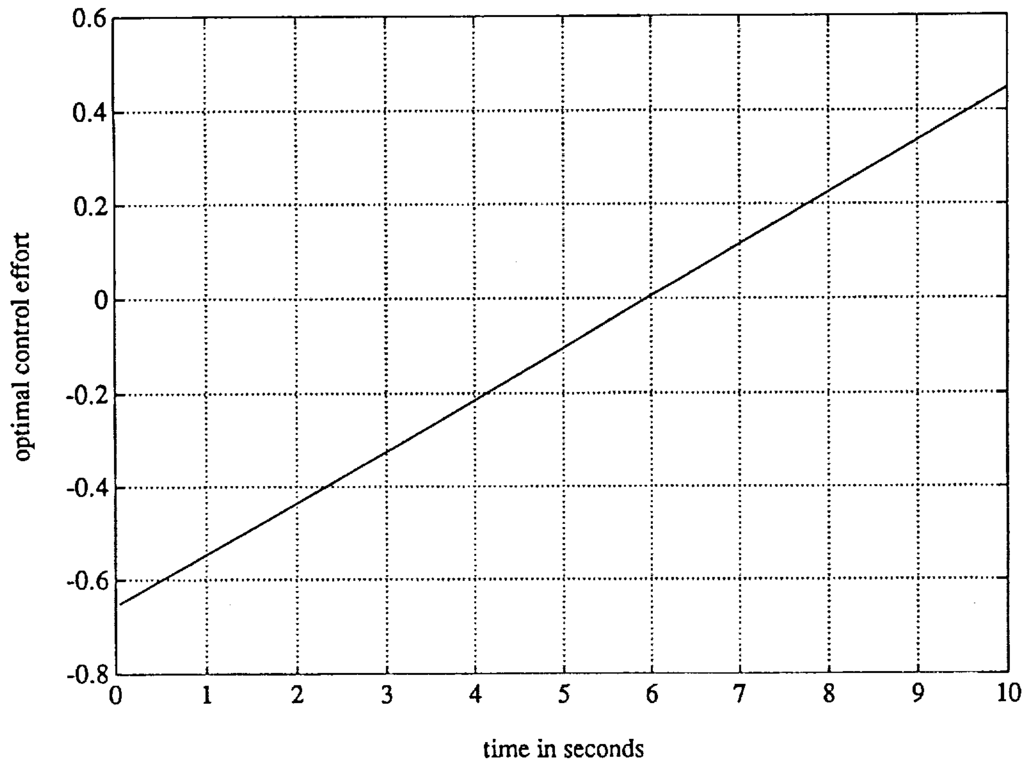


Figure 8. Optimal control effort with relaxed terminal penalty.

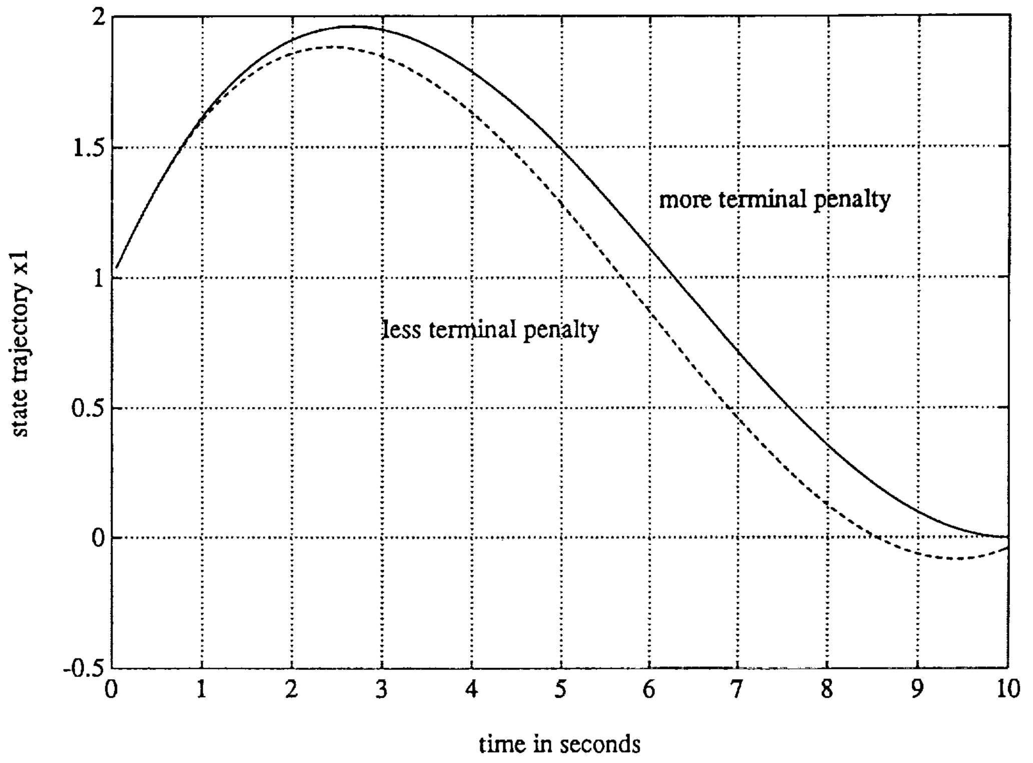


Figure 9. Optimal trajectories comparison.

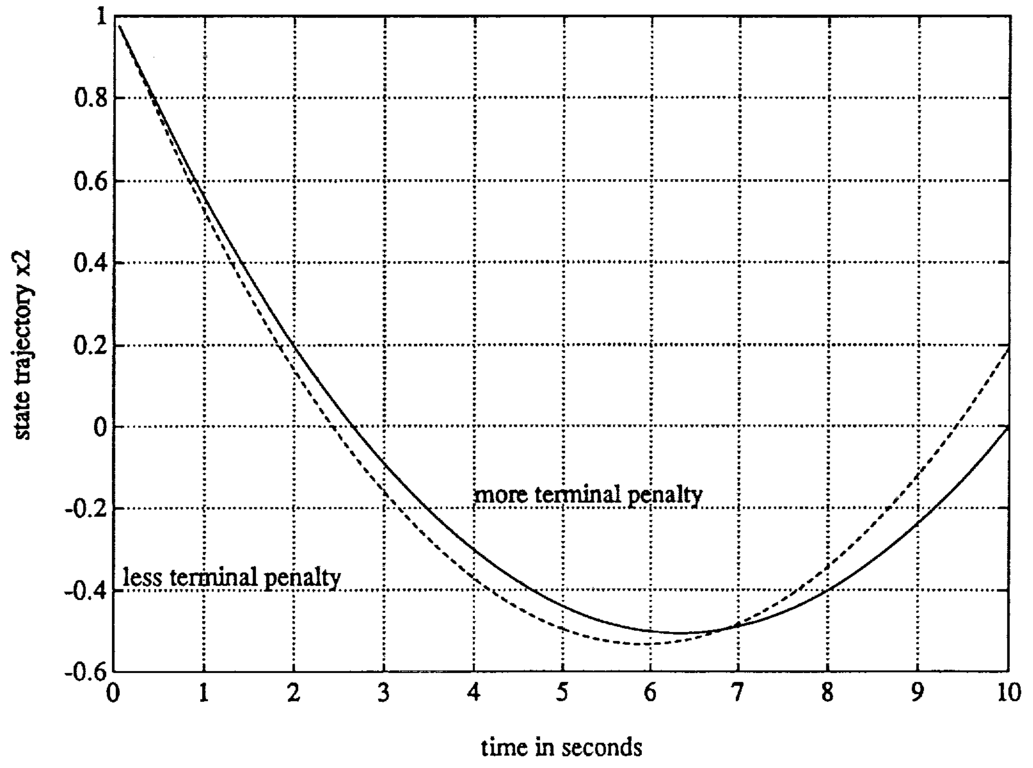


Figure 10. Optimal trajectories comparison.

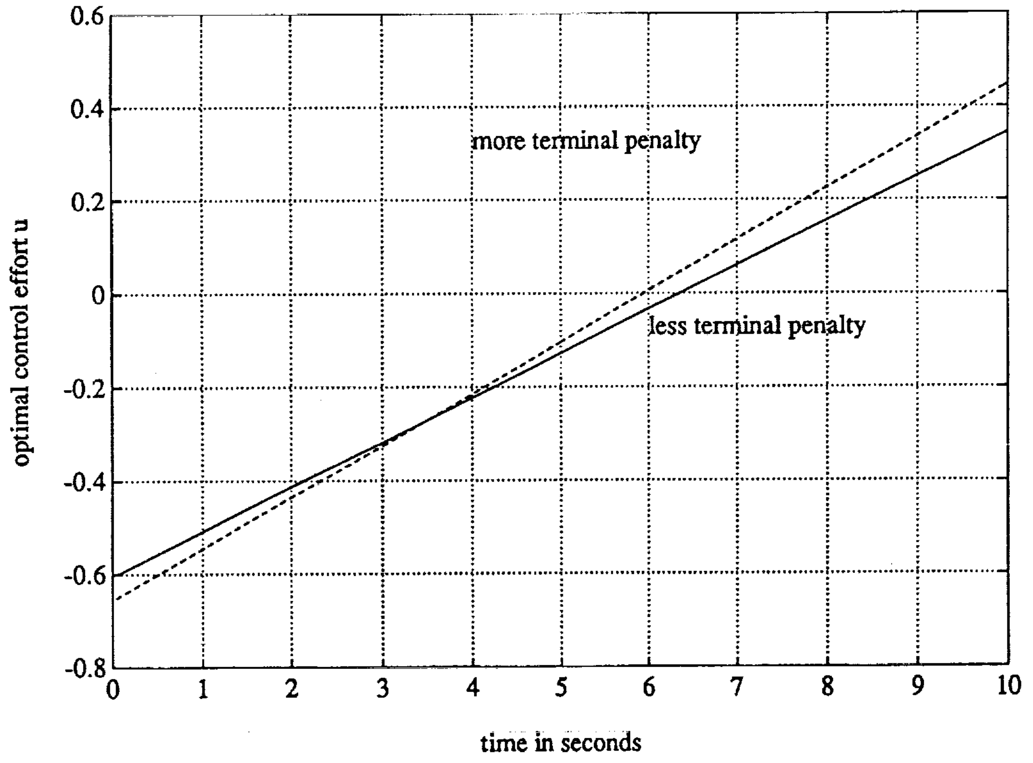


Figure 11. Optimal control efforts comparison.

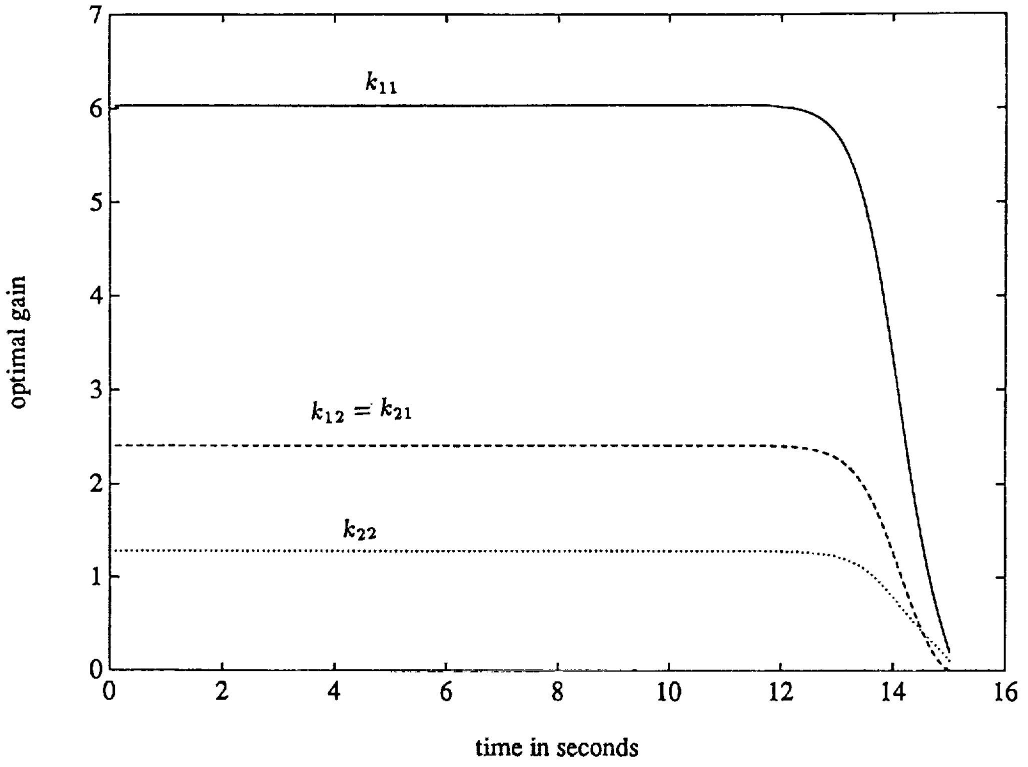


Figure 12. Optimal time-varying gain.

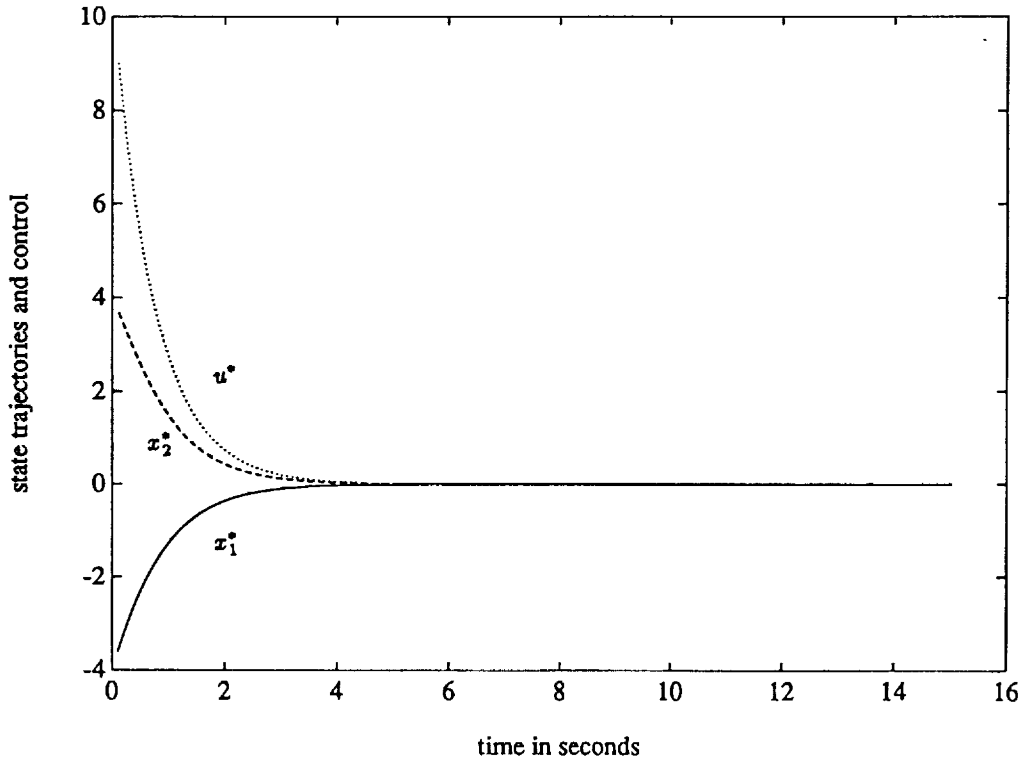


Figure 13. Optimal control and state trajectories.

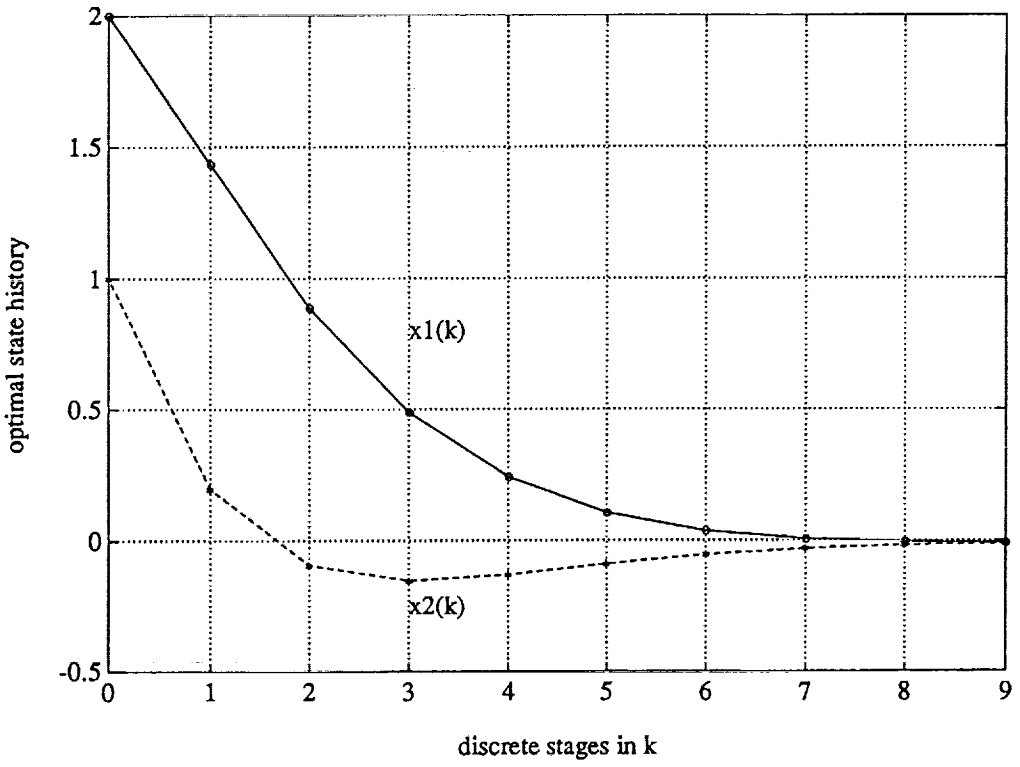


Figure 14. State histories from dynamic programming.

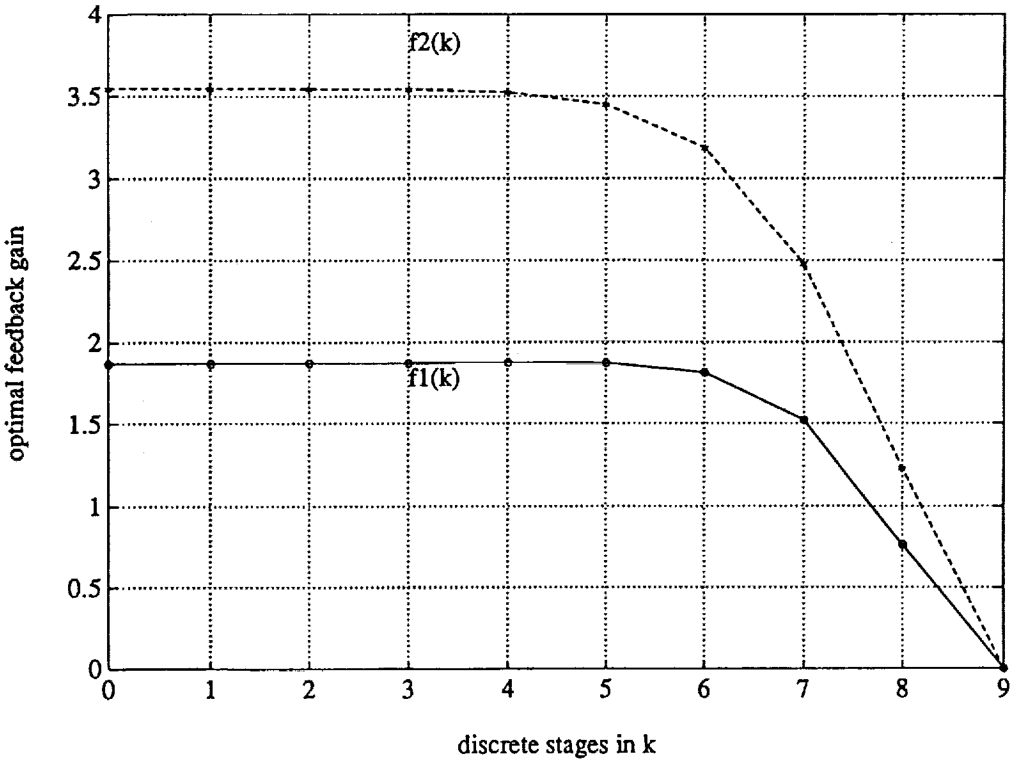


Figure 15. Feedback gain by dynamic programming.

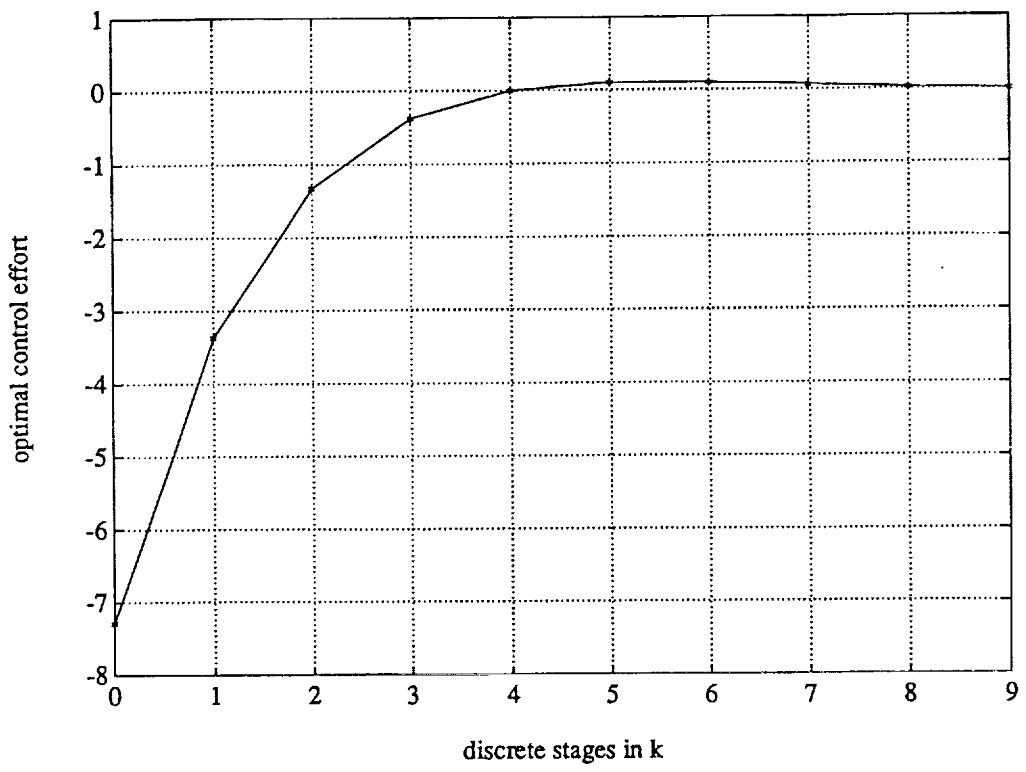


Figure 16. Optimal control by dynamic programming.

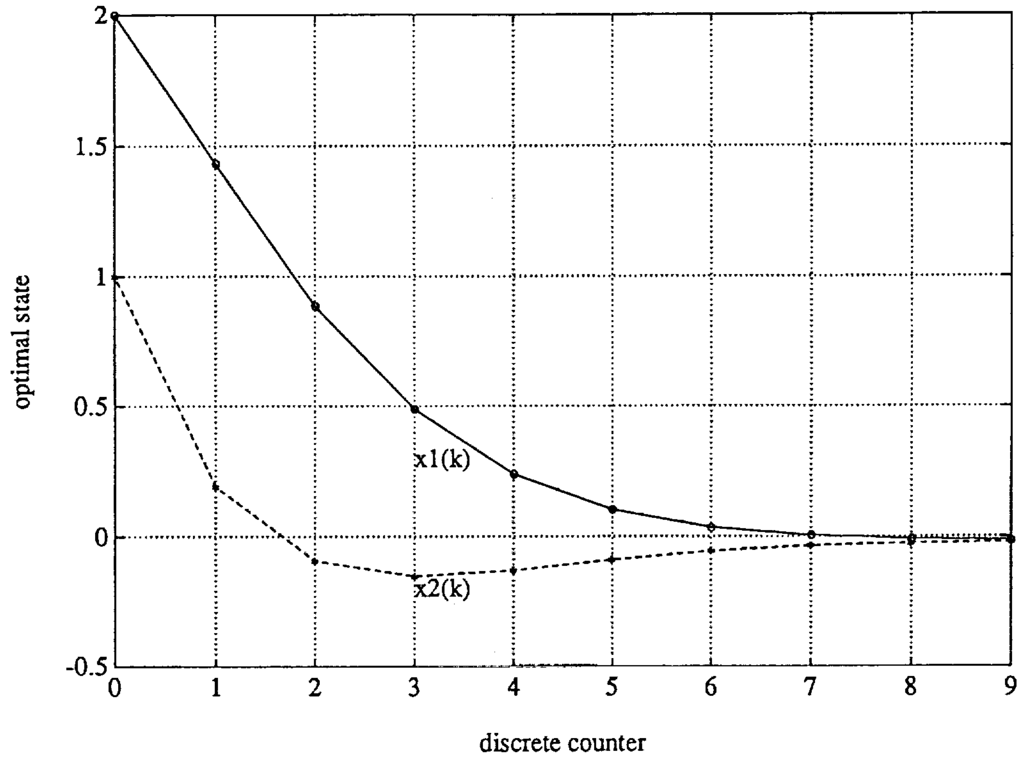


Figure 17. State histories by discrete LQR.

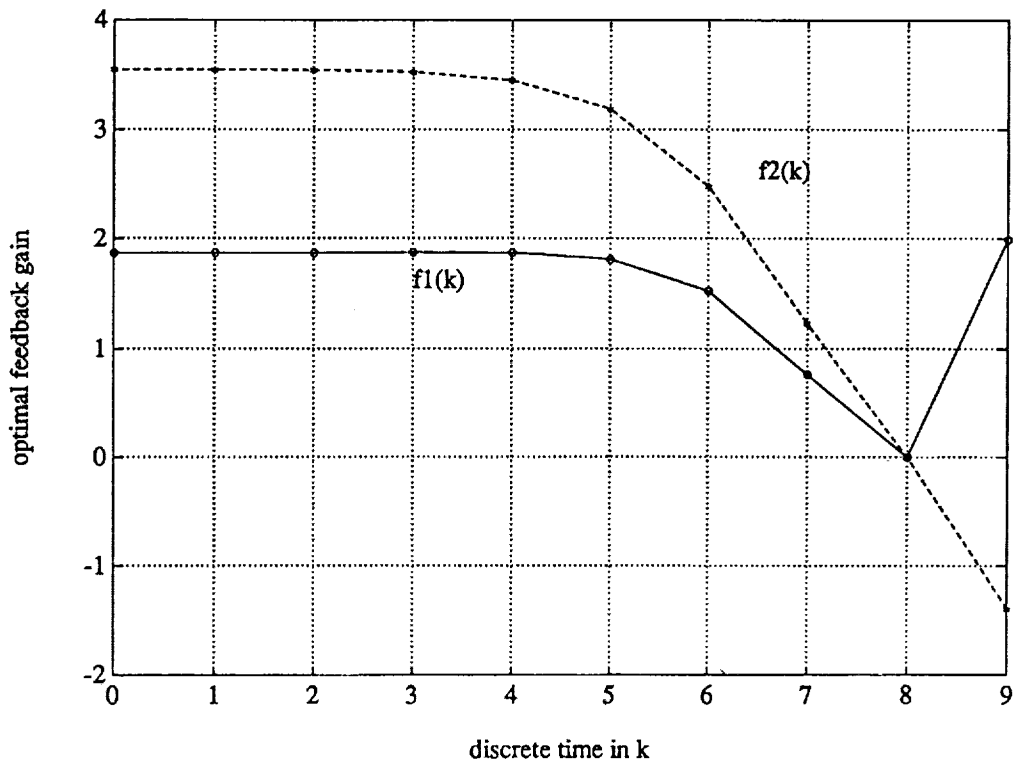


Figure 18. Feedback gain by LQR.

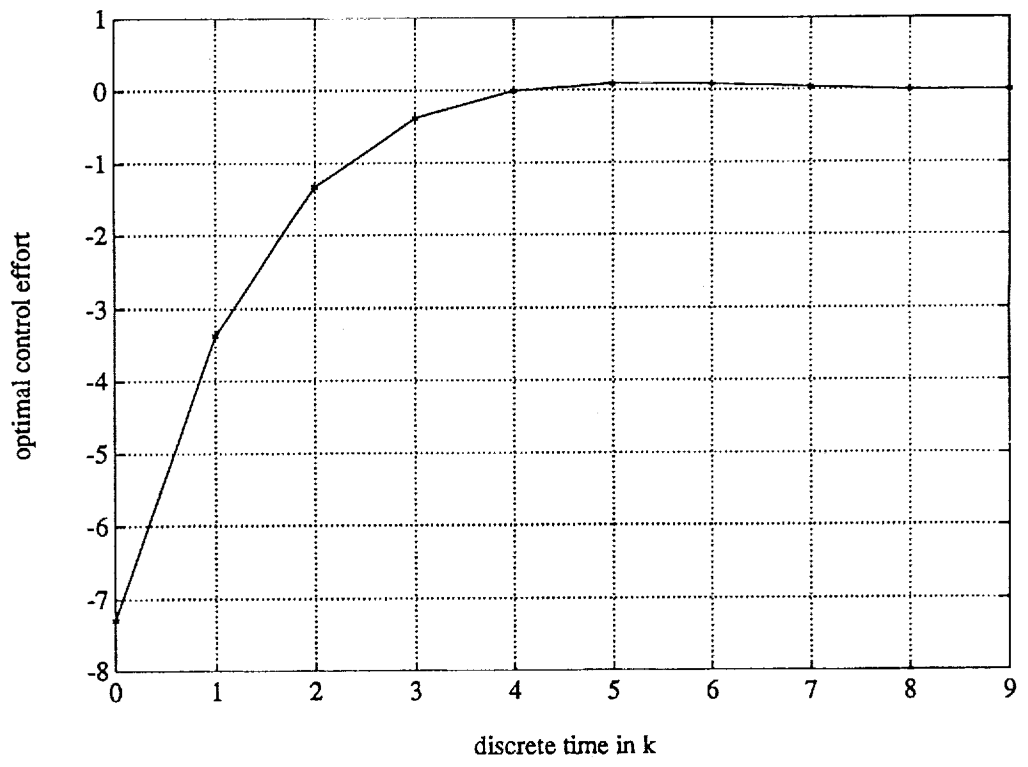


Figure 19. Optimal control by LQR.

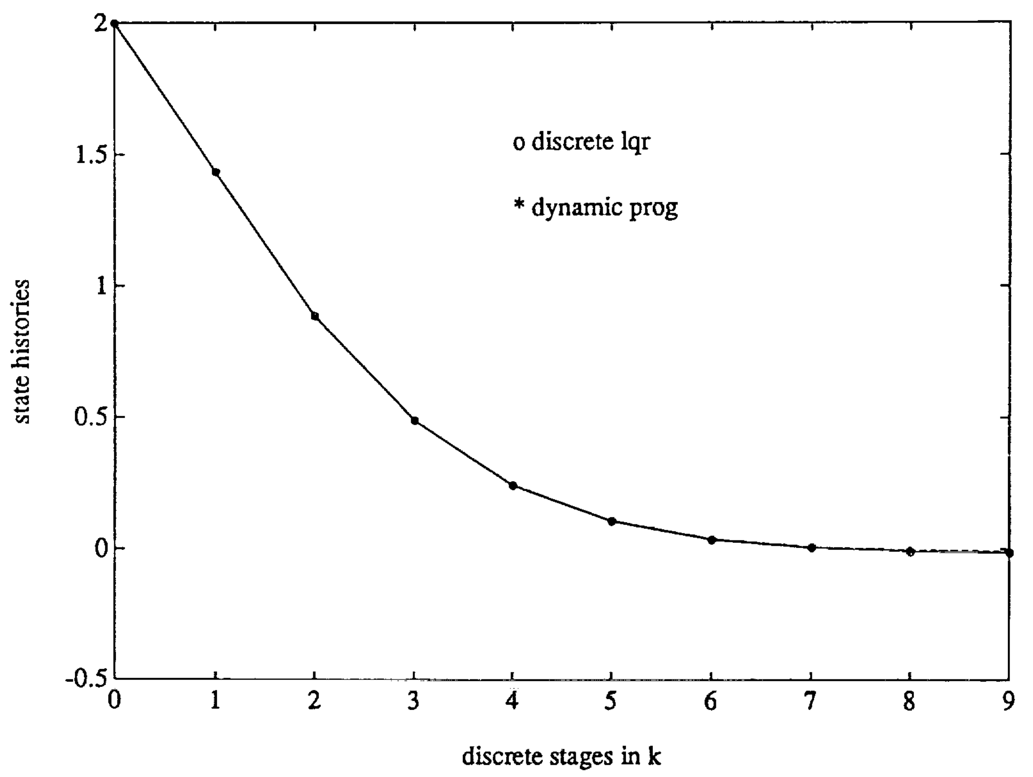


Figure 20. State histories comparison.

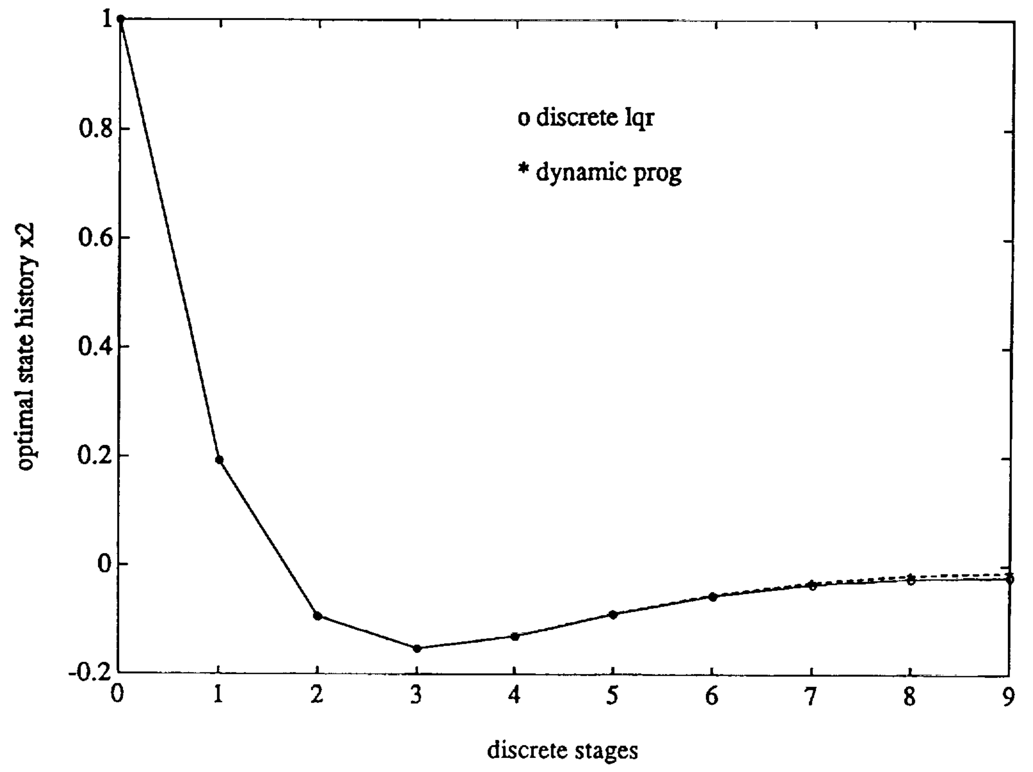


Figure 21. State histories comparison.

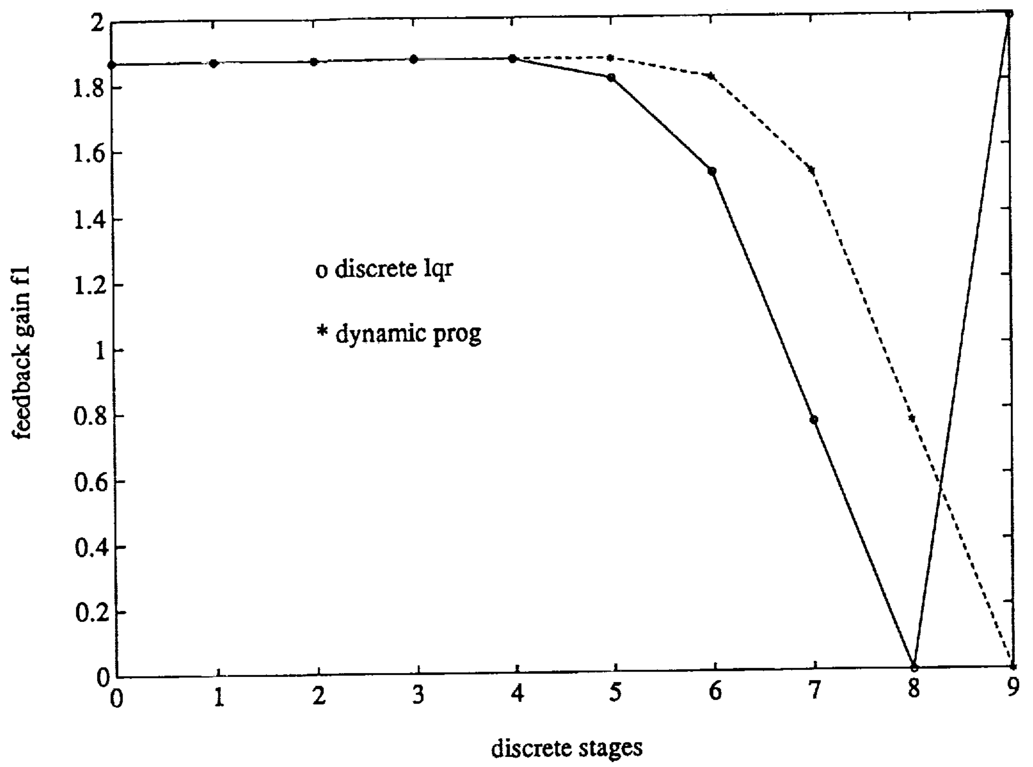


Figure 22. Feedback gains comparison.

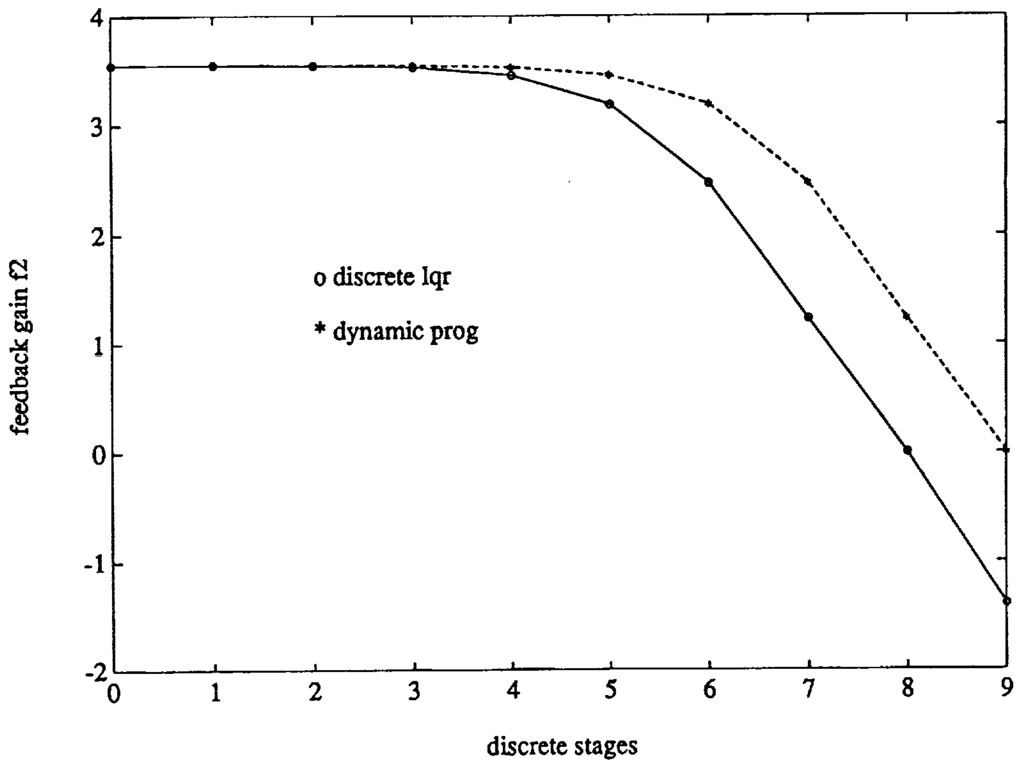


Figure 23. Feedback gains comparison.

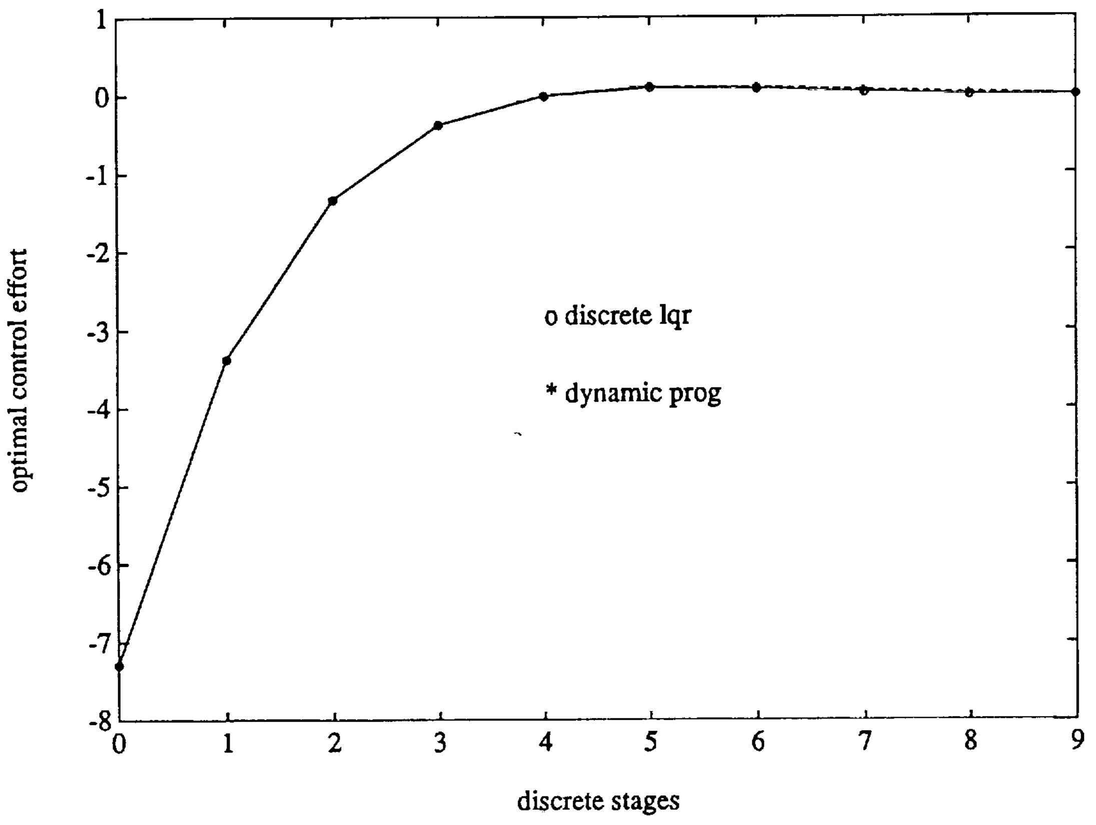


Figure 24. Optimal control efforts comparison.

APPENDIX

PROGRAM LISTINGS

Two-Point Boundary Value Solver

```
      program twopt
c
c     This program solves a generalized two point value problem
c     derived from the First Necessary Condition of the
c     optimal control theory based on calculus of variation.
c     The method of maximum descent in conjunction with 4th order
c     runge-kutta routine is used.
c     basically, it integrates the state equations forward in time
c     and set the boundary condition of the co-state equations
c     according to the Transversality Condition.
c     The co-state equation is then integrate backward in time.
c     The performance index is checked every iteration.
c     xdot and x are state derivative and state respectively
c     pdot and p are co-state derivative and co-state respectively
c     np = total number of points (np*dh=T=total simulation time)
c     dh = integration step time
c     n = number of states
c     m,k = runge-kutta routine indeices, set to 0 when initially called
c     subroutine called
c     runge(n,x,xdot,t,dh,m,k)
c     For higher order problems, change the dimension appropriately
c     This problem simulates the example on page 12 of the paper
c
      real x(2),xdot(2),p(2),pdot(2),x1(100),
     * dhdu(100),u(100), hold(100),h(100),p1(100),x2(100),p2(100)
      open(unit=3,file='hw1.out',status='old')
      open(unit=4,file='hw2.out',status='old')
      tau = 1.
c
c     store initial guess of u
c
      do 1 j=1,np
      u(j) = 1.
    1 continue
  111 n = 2
      t = 0.
      dh =.02
      np=78
      m=0
      x(1)=0.05
      x(2)=0.
c
c     Integrate state equations forward
c
      do 8 i=1,np
    6 call runge(n,x,xdot,t,dh,m,k)
      goto(10,20),k
c
c     insert state equations here
c
   10 xdot(1) = -2.*(x(1)+0.25)+(x(2)+0.5)*exp((25.*x(1))/(x(1)+2.))
     * -(x(1)+0.25)*u(i)
      xdot(2)=0.5-x(2)-(x(2)+0.5)*exp((25.*x(1))/(x(1)+2.))
      goto 6
c
c     Store state trajectories
c
   20 x1(i) = x(1)
      x2(i) = x(2)
c      print *,i,x(1),x(2),k
    8 continue
c
c     set initial condition for co-state equation
c     and find hamiltonian h
c
      p(1)=0.
```

PRECEDING PAGE BLANK NOT FILMED

PAGE 30 INTENTIONALLY BLANK

```
        p(2)=0.
        m = 0
        do 11 i1= 1,np
     11 hold(i1) = h(i1)
c
c       integrate co- state equations backward
c
        do 80 i2=np,1,-1
c        print *,pdot(1),pdot(2),p(1),p(2)
    60  call runge(2,p,pdot,t,dh,m,kk)
        goto(100,200),kk
c
c       insert co-state equations here
c
   100  dum1=x2(i2) + 0.5
        ep1=50./(x1(i2)+2)**2
        ep2=exp(25.*x1(i2)/(x1(i2)+2.))
        pdot(1)  = 2.*x1(i2)-2.*p(1)+p(1)*dum1*ep1*ep2-p(1)*u(i2)
       *  -p(2)*dum1*ep1*ep2
        pdot(2)  = 2.*x2(i2)+p(1)*ep2-p(2)*(1.+ep2)
        goto 60
c
c       calculate hamiltonian and dhdu
c
    200 h(i2) = 0.5*u(i2) - p(1)*x1(i2) + p(1)*u(i2)
        dhdu(i2) = 0.2*u(i2)-p(1)*(x1(i2)+0.25)
        p1(i2) = p(1)
        p2(i2) =p(2)
     80  continue
c
c       check convergeance and define new u(ik)
c       sum = the integral of the norm used for convergeance checking
c       sumj= performance sum for checking that J is indeed decreasing
c             otherwise reduce tau
c       icount = number of iteration desired, can arbitrarily set.
c
        sumjold = sumj
        sum = 0.
        sumj=0.
        do 4 i3=1,np
        u(i3) = u(i3) - tau* dhdu(i3)
        sum=sum + dhdu(i3)**2
        sumj = sumj + (x1(i3)**2 + x2(i3)**2+0.1*u(i3)**2)*dh
      4 continue
        if (sumj .lt. sumjold) goto 14
        tau = 0.5*tau
        goto 111
     14  write(4,44) icount,sum,sumj,tau
     44 format(i10,3f15.8)
        icount = icount + 1
        if(icount .ge. 200) goto 5
        if (sum .gt. .01) goto 111
c
c       after optimal control is found, the state trajectories
c       are recalculate based on stored u
c
        x(1)=0.05
        x(2)=0.
        do 88 i4=1,np
     66 call runge(n,x,xdot,t,dh,m,k)
        goto(110,88),k
    110 xdot(1)  = -2.*(x(1)+0.25)+(x(2)+0.5)*exp((25.*x(1))/(x(1)+2.))
       * -(x(1)+0.25)*u(i4)
        xdot(2)=0.5-x(2)-(x(2)+0.5)*exp((25.*x(1))/(x(1)+2.))
        goto 66
    88  continue
```

```
c
c     outputs to data file for plotting
c
  5   do 1000 iz=1,np
 1000 write(3,1001) icount,x1(iz),x2(iz),dhdu(iz),u(iz)
 1001 format(i4,4f10.4)
      stop
      end
      subroutine runge(n,y,f,t,h,m,k)
      dimension y(2),f(2),q(2)
      m= m+1
      goto(1,4,5,3,7),m
  1   do 2 i = 1,n
  2   q(i) = 0.
      a = .5
      goto 9
  3   a=1.707107
   4  t = t + .5*h
  5   do 6 i=1,n
      y(i) = y(i) + a*f(i)*h-q(i)
  6   q(i) = 2.*a*f(i)*h + (1.-3.*a)*q(i)
      a = .2928932
      goto 9
  7   do 8 i=1,n
  8   y(i) = y(i) + h*f(i)/6.-q(i)/3.
      m = 0
      k = 2
      goto 10
  9   k =1
  10  return
      end
```

Continuous Time Matrix Riccati Equation Solver Application to Time-Varying Optimal Control

```
      program clqr
c
c     This program presents a algorithm for solving continous
c     time LQR problem by solving time varying Matrix Riccati
c     equation.
c     This program is baseb on many subroutines for matrix
c     manipulations.
c     Euler integration routine is used for simplicity
c     np = number of integration steps
c     np*dt = terminal time
c     This routine can also integrate for large np for steaty
c     state (or infinite time LQR) problems.
c     This program simulates the example on page 15 of this paper
c
      real a(2,2),at(2,2),q(2,2),p(2,2),r(1,1),ri(1,1),pdot(2,2)
      real b(2,1),bt(1,2),pa(2,2),atp(2,2),rbp(1,2),pbrbp(2,2)
      real pbrbp2(2,2),u(1,1),x(2,1),h(2,2),brbp(2,2),pbrbp1(2,2)
      real up(1,1),ax(2,1),bu(2,1),xdot(2,1),p11(1000),p12(1000)
      real p21(1000),p22(1000)
      open(unit=3,file='clqr.out',status='old')
      l=2
      m=1
      np=150
c
c     define a,b,q,h matrices
c
      q(1,1)=2.
      q(1,2)=0.
      q(2,1)=0.
      q(2,2)=1.
c
      a(1,1)=0.
      a(1,2)=1.
      a(2,1)=2.
      a(2,2)=-1.
c
      b(1,1)=0.
      b(2,1)=1.
      r(1,1)=0.5
      dt =.1
c
      h(1,1)=0.
c
c     set initial state value x
c
      x(1,1)=-4.
      x(2,1)=4.
c
c     find constant transpose and inverse
c
      call matran(b,bt,l,m)
      call matran(a,at,l,l)
      call matinv(r,ri,m,m)
c
c     set p final to h
c
        do 1 ix=1,l
           do 2 iy=1,l
         p(ix,iy) = h(ix,iy)
    2      continue
    1   continue
      do 200 ii=np,1,-1
c
c      find f(k) = [R + Bt P B]^-1 Bt P A
c
      call matqual(ri,bt,p,rbp,m,l,l)
      call matmul(b,rbp,brbp,l,m,l)
      call matmul(p,brbp,pbrbp,l,l,l)
      call matmul(p,a,pa,l,l,l)
      call matmul(at,p,atp,l,l,l)
      call matadd(pa,pbrbp,pbrbp1,l,l,-1.)
      call matadd(pbrbp1,atp,pbrbp2,l,l,1.)
      call matadd(pbrbp2,q,pdot,l,l,1.)
      call euler(pdot,p,dt,l,l)
      p11(ii)=p(1,1)
      p12(ii)=p(1,2)
      p21(ii)=p(2,1)
      p22(ii)=p(2,2)
200   continue
      time = 0.
      do 300 ij=1,np
      time = time + dt
      p(1,1)=p11(ij)
      p(1,2)=p12(ij)
      p(2,1)=p21(ij)
      p(2,2)=p22(ij)
      call matqual(ri,bt,p,rbp,m,l,l)
      call matmul(rbp,x,up,m,l,m)
      call matneg(up,u,m,m)
      call matmul(a,x,ax,l,l,m)
      call matmul(b,u,bu,l,m,m)
      call matadd(ax,bu,xdot,l,m,1.)
      call euler(xdot,x,dt,l,m)
300   write(3,301) time,p(1,1),p(1,2),p(2,2),x(1,1),x(2,1),u(1,1)
301   format(7e15.5)
      stop
      end
      subroutine euler(dot,pp;dt,lr,lc)
      real dot(lr,lc),pp(lr,lc)
      do 10 i=1,lr
         do 20 j=1,lc
            pp(i,j)=pp(i,j) + dot(i,j)*dt
20    continue
10    continue
      return
      end
      subroutine matneg(mata,matb,lr,lc)
      real mata(lr,lc),matb(lr,lc)
          do 10 ix=1,lr
             do 20 iy=1,lc
                matb(ix,iy) = - mata(ix,iy)
20           continue
10        continue
      return
      end
      subroutine matqual(mata,matb,matc,matd,mm,ll,ll)
      real mata(mm,mm),matb(mm,ll),matc(ll,ll),matd(mm,ll),dum(1,2)
      call matmul(matb,matc,dum,mm,ll,ll)
      call matmul(mata,dum,matd,mm,mm,ll)
      return
      end
      subroutine matinv(a,ai,n,m)
      real a(n,m),ai(n,m),aa(2,2)
      do 10 i=1,n
         do 20 j=1,m
         aa(i,j)=a(i,j)
20       continue
10    continue
      call sgefa(aa,n,m,ipvt,info)
      call sgedi(aa,n,m,ipvt,det,work,01)
      do 1 ii=1,n
         do 2 ij=1,m
```

```
           ai(ii,ij)=aa(ii,ij)
   2     continue
   1   continue
       return
       end
       subroutine sgedi(a,lda,n,ipvt,det,work,job)
       integer lda,n,ipvt(1),job
       real a(lda,1),det(2),work(1)
c
c      sgedi computes the determinant and inverse of a matrix
c      using the factors computed by sgeco or sgefa.
c
c      on entry
c
c         a       real(lda, n)
c                 the output from sgeco or sgefa.
c
c         lda     integer
c                 the leading dimension of the array  a .
c
c         n       integer
c                 the order of the matrix  a .
c
c         ipvt    integer(n)
c                 the pivot vector from sgeco or sgefa.
c
c         work    real(n)
c                 work vector.  contents destroyed.
c
c         job     integer
c                 = 11   both determinant and inverse.
c                 = 01   inverse only.
c                 = 10   determinant only.
c
c      on return
c
c         a       inverse of original matrix if requested.
c                 otherwise unchanged.
c
c         det     real(2)
c                 determinant of original matrix if requested.
c                 otherwise not referenced.
c                 determinant = det(1) * 10.0**det(2)
c                 with  1.0 .le. abs(det(1)) .lt. 10.0
c                 or  det(1) .eq. 0.0 .
c
c      error condition
c
c         a division by zero will occur if the input factor contains
c         a zero on the diagonal and the inverse is requested.
c         it will not occur if the subroutines are called correctly
c         and if sgeco has set rcond .gt. 0.0 or sgefa has set
c         info .eq. 0 .
c
c      linpack. this version dated 08/14/78 .
c      cleve moler, university of new mexico, argonne national lab.
c
c      subroutines and functions
c
c      blas saxpy,sscal,sswap
c      fortran abs,mod
c
c      internal variables
c
       real t
       real ten
```

```
      integer i,j,k,kb,kp1,l,nm1
c
c
c     compute determinant
c
      if (job/10 .eq. 0) go to 70
         det(1) = 1.0e0
         det(2) = 0.0e0
         ten = 10.0e0
         do 50 i = 1, n
            if (ipvt(i) .ne. i) det(1) = -det(1)
            det(1) = a(i,i)*det(1)
c        ...exit
            if (det(1) .eq. 0.0e0) go to 60
   10       if (abs(det(1)) .ge. 1.0e0) go to 20
               det(1) = ten*det(1)
               det(2) = det(2) - 1.0e0
            go to 10
   20       continue
   30       if (abs(det(1)) .lt. ten) go to 40
               det(1) = det(1)/ten
               det(2) = det(2) + 1.0e0
            go to 30
   40       continue
   50    continue
   60    continue
   70 continue
c
c     compute inverse(u)
c
      if (mod(job,10) .eq. 0) go to 150
         do 100 k = 1, n
            a(k,k) = 1.0e0/a(k,k)
            t = -a(k,k)
            call sscal(k-1,t,a(1,k),1)
            kp1 = k + 1
            if (n .lt. kp1) go to 90
            do 80 j = kp1, n
               t = a(k,j)
               a(k,j) = 0.0e0
               call saxpy(k,t,a(1,k),1,a(1,j),1)
   80       continue
   90       continue
  100    continue
c
c        form inverse(u)*inverse(l)
c
         nm1 = n - 1
         if (nm1 .lt. 1) go to 140
         do 130 kb = 1, nm1
            k = n - kb
            kp1 = k + 1
            do 110 i = kp1, n
               work(i) = a(i,k)
               a(i,k) = 0.0e0
  110       continue
            do 120 j = kp1, n
               t = work(j)
               call saxpy(n,t,a(1,j),1,a(1,k),1)
  120       continue
            l = ipvt(k)
            if (l .ne. k) call sswap(n,a(1,k),1,a(1,l),1)
  130    continue
  140    continue
  150 continue
      return
      end
      subroutine sgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      real a(lda,1)
c
c     sgefa factors a real matrix by gaussian elimination.
c
c     sgefa is usually called by sgeco, but it can be called
c     directly with a saving in time if  rcond  is not needed.
c     (time for sgeco) = (1 + 9/n)*(time for sgefa)
c
c     on entry
c
c        a       real(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        info    integer
c                = 0  normal value.
c                = k  if  u(k,k) .eq. 0.0 .  this is not an error
c                     condition for this subroutine, but it does
c                     indicate that sgesl or sgedi will divide by zero
c                     if called.  use  rcond  in sgeco for a reliable
c                     indication of singularity.
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas saxpy,sscal,isamax
c
c     internal variables
c
      real t
      integer isamax,j,k,kp1,l,nm1
c
c
c     gaussian elimination with partial pivoting
c
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
c
c        find l = pivot index
c
         l = isamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
```

```
c
c        zero pivot implies this column already triangularized
c
         if (a(l,k) .eq. 0.0e0) go to 40
c
c           interchange if necessary
c
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
c
c           compute multipliers
c
            t = -1.0e0/a(k,k)
            call sscal(n-k,t,a(k+1,k),1)
c
c           row elimination with column indexing
c
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call saxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0e0) info = n
      return
      end
      subroutine saxpy(n,sa,sx,incx,sy,incy)
c
c     constant times a vector plus a vector.
c     uses unrolled loop for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      real sx(1),sy(1),sa
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (sa .eq. 0.0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        sy(iy) = sy(iy) + sa*sx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
```

```
c
c
c        clean-up loop
c
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sy(i) = sy(i) + sa*sx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        sy(i) = sy(i) + sa*sx(i)
        sy(i + 1) = sy(i + 1) + sa*sx(i + 1)
        sy(i + 2) = sy(i + 2) + sa*sx(i + 2)
        sy(i + 3) = sy(i + 3) + sa*sx(i + 3)
   50 continue
      return
      end
      subroutine sscal(n,sa,sx,incx)
c
c     scales a vector by a constant.
c     uses unrolled loops for increment equal to 1.
c     jack dongarra, linpack, 3/11/78.
c     modified to correct problem with negative increments, 9/29/88.
c
      real sa,sx(1)
      integer i,ix,incx,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      ix = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      do 10 i = 1,n
        sx(ix) = sa*sx(ix)
        ix = ix + incx
   10 continue
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sx(i) = sa*sx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        sx(i) = sa*sx(i)
        sx(i + 1) = sa*sx(i + 1)
        sx(i + 2) = sa*sx(i + 2)
        sx(i + 3) = sa*sx(i + 3)
        sx(i + 4) = sa*sx(i + 4)
   50 continue
      return
      end
      subroutine sswap (n,sx,incx,sy,incy)
c
c     interchanges two vectors.
```

```
c     uses unrolled loops for increments equal to 1.
c     jack dongarra, linpack, 3/11/78.
c
      real sx(1),sy(1),stemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c       code for unequal increments or equal increments not equal
c         to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        stemp = sx(ix)
        sx(ix) = sy(iy)
        sy(iy) = stemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c       code for both increments equal to 1
c
c
c       clean-up loop
c
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        stemp = sx(i)
        sx(i) = sy(i)
        sy(i) = stemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,3
        stemp = sx(i)
        sx(i) = sy(i)
        sy(i) = stemp
        stemp = sx(i + 1)
        sx(i + 1) = sy(i + 1)
        sy(i + 1) = stemp
        stemp = sx(i + 2)
        sx(i + 2) = sy(i + 2)
        sy(i + 2) = stemp
   50 continue
      return
      end
      integer function isamax(n,sx,incx)
c
c     finds the index of element having max. absolute value.
c     jack dongarra, linpack, 3/11/78.
c     modified to correct problem with negative increments, 9/29/88.
c
      real sx(1),smax
      integer i,incx,ix,n
c
      isamax = 0
      if( n .lt. 1 ) return
      isamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
```

```
c
c        code for increment not equal to 1
c
      ix = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      smax = abs(sx(ix))
      ix = ix + incx
      do 10 i = 2,n
         if(abs(sx(ix)).le.smax) go to 5
         isamax = i
         smax = abs(sx(ix))
    5    ix = ix + incx
   10 continue
      return
c
c        code for increment equal to 1
c
   20 smax = abs(sx(1))
      do 30 i = 2,n
         if(abs(sx(i)).le.smax) go to 30
         isamax = i
         smax = abs(sx(i))
   30 continue
      return
      end
      subroutine matmul(mata,matb,prod,ll,nn,mm)
      real mata(ll,nn),matb(nn,mm),prod(ll,mm),sum
      do 40 i =1,ll
         do 30 j=1,mm
           sum=0.0
            do 20 k=1, nn
               sum= sum + mata(i,k) * matb(k,j)
   20       continue
           prod(i,j) = sum
   30    continue
   40 continue
      return
      end
      subroutine matadd(mata,matb,sum,nn,mm,code)
      real mata(nn,mm),matb(nn,mm),sum(nn,mm)
      do 40 i=1,nn
         do 30 j=1,mm
         if(code .ge. 0.)   sum(i,j) = mata(i,j) + matb(i,j)
         if(code .lt. 0.)   sum(i,j) = mata(i,j) - matb(i,j)
   30    continue
   40 continue
      return
      end
      subroutine matran(mata,matb,nn,mm)
      real mata(nn,mm),matb(mm,nn)
      do 40 i=1,mm
         do 30 j=1,nn
          matb(i,j) = mata(j,i)
   30    continue
   40 continue
      return
      end
```

Discrete LQR Solver

```
      program dlqr
c     For discrete LQR problem
c     This program uses the reccursive gain formula developed on page
c     16 of the paper. This program is similar to program Dynam for
c     the dynamic programming approach and uses the same subroutines
c     and therefore this subroutines are not listed again.
c     One only needs to change the dimension of the array in order
c     to run a higher order system
c     Matrices a,h,q are of dimension lxl
c     matrix b is lxm
c     Matrix f is mxl
c     matrix r is mxm
c     bt, at, are transpose of a and b
c     ati,ri are inverse of at and r
c     dum() are intermediate matrices
c     nos = number of stages
c
      real a(2,2),ati(2,2),at(2,2),q(2,2),p(2,2),r(1,1),ri(1,1)
      real b(2,1),bt(1,2),dum(2,1),dum1(1,1),dum2(1,2),dum3(2,2),dum33(2,2)
      real g(2,2),fk(1,2),u(1,1),x(2,1),h(2,2),dum4(2,2),dum22(1,2)
      real f11(100),f12(100),up(1,1),ax(2,1),bu(2,1),xk1(2,1)
      open(unit=3,file='dlqr.out',status='old')
      l=2
      m=1
      nos = 10
c
c     define a,b,q,h matrices
c
      q(1,1)=50.
      q(1,2)=0.
      q(2,1)=0.
      q(2,2)=10.
c
      a(1,1)=0.6277
      a(1,2)=0.3597
      a(2,1)=0.0899
      a(2,2)=0.8526
c
      b(1,1)=.025
      b(2,1)=.115
      r(1,1)=1.
c
      h(1,1)=0.
      h(1,2)=0.
      h(2,1)=0.
      h(2,2)=0.
c
c     find constant transpose and inverse
c
      call matran(b,bt,l,m)
      call matran(a,at,l,l)
      call matinv(r,ri,m,m)
      call matinv(at,ati,l,l)
c
c     set p final to h
c
      do 10 ii = 1,l
         do 11 jj=1,l
         p(ii,jj) = h(ii,jj)
   11    continue
   10 continue
      do 200 iz=nos,1,-1
c
c      find f(k) = -r^(-1) b^t (a^t)^-1 (p-q)
c
      call matadd(p,q,dum3,l,l,-1.)
      call matmul(ati,dum3,dum4,l,l,l)
      call matmul(bt,dum4,dum2,m,l,l)
      call matmul(ri,dum2,fk,m,m,l)
c
c     find G(k+1)
c     matrix operation from left
c
      call matmul(p,b,dum,l,l,m)
      call matmul(bt,dum,dum1,m,l,m)
      call matadd(r,dum1,dum1,m,m,1.)
      call matinv(dum1,dum1,m,m)
      call matmul(bt,p,dum2,m,l,l)
      call matmul(dum1,dum2,dum22,m,m,l)
      call matmul(b,dum22,dum3,l,m,l)
      call matmul(p,dum3,dum33,l,l,l)
      call matadd(p,dum33,g,l,l,-1.)
c
c     find pk=a^t g(k+1) a + q
c
      call matmul(g,a,dum3,l,l,l)
      call matmul(at,dum3,dum4,l,l,l)
      call matadd(dum4,q,p,l,l,l)
c
c     save and decomposed fk
c
      f11(iz)=fk(1,1)
      f12(iz)=fk(1,2)
  200 continue
c
c     calculate optimal control and state trajectories
c
c     set initial state
c
      x(1,1)=2.
      x(2,1)=1.
      do 1000 ij=1,nos
      fk(1,1)=f11(ij)
      fk(1,2)=f12(ij)
      call matmul(fk,x,up,m,l,m)
      call matneg(up,u,m,m)
      call matmul(a,x,ax,l,l,m)
      call matmul(b,u,bu,l,m,m)
      call matadd(ax,bu,xk1,l,m,1.)
      icount= ij-1
      write(3,1001) icount,x(1,1),x(2,1),fk(1,1),fk(1,2),u(1,1)
 1001 format(i5,5f10.4)
 1000 call matequ(xk1,x,l,m)
      stop
      end
```

Discrete LQR Solver Based on Dynamic Programming

```
      program dynamic
c     This program uses the reccursive gain formula developed under
c     the multistage decision process(dynamic programming).
c     One only needs to change the dimension of the array in order
c     to run a higher order system
c     Matrices a,h,q are of dimension lxl
c     matrix x is lxm
c     matrix b is lxm
c     Matrix f is mxl
c     matrix r is mxm
c     bt, at, are transpose of a and b
c     ati,ri are inverse of at and r
c     dum() are intermediate matrices
c     nos = number of stages
c     This particular program solves the example on page 16 of the
c     paper.
c     One only needs to input the a,q,b,h matices for new problems
c     and change the appropriate array dimension.
c     The basis of this program is the matrix routine developed
c     to perform various matrix manipulations
c     Matrix inverse routine is from LINPAK
c
      real a(2,2),ati(2,2),at(2,2),q(2,2),p(2,2),r(1,1),ri(1,1)
      real b(2,1),bt(1,2),dum(2,2),bpb(1,1),rpb(1,1),rpbi(1,1),apa(2,2)
      real g(2,2),f(1,2),u(1,1),x(2,1),h(2,2),bpa(1,2),ft(2,1),dum1(2,2)
      real f11(100),f12(100),up(1,1),ax(2,1),bu(2,1),xk1(2,1)
      open(unit=3,file='dynam.out',status='old')
      l=2
      m=1
      nos =10
c
c     define a,b,q,h matrices
c
      q(1,1)=50.
      q(1,2)=0.
      q(2,1)=0.
      q(2,2)=10.
c
      a(1,1)=0.6277
      a(1,2)=0.3597
      a(2,1)=0.0899
      a(2,2)=0.8526
c
      b(1,1)=.025
      b(2,1)=.115
      r(1,1)=1.
c
      h(1,1)=0.
      h(1,2)=0.
      h(2,1)=0.
      h(2,2)=0.
c
c     find constant transpose and inverse
c     at is transpose of a
c     ri is inverse of r
c     ati is inverse of at
c     l,m are dimension of matrices
c
      call matran(b,bt,l,m)
      call matran(a,at,l,l)
      call matinv(r,ri,m,m)
      call matinv(at,ati,l,l)
c
c     set p final to h
c
      do 10 ii = 1,l
```

```
          do 11 jj=1,l
          p(ii,jj) = h(ii,jj)
   11     continue
   10 continue
c
      do 200 iz=nos,1,-1
c
c      find f(k) = [R + Bt P B]^-1 Bt P A
c
      call matqual(bt,p,a,bpa,m,l,l)
      call matqua(bt,p,b,bpb,m,l,m)
      call matadd(r,bpb,rpb,m,m,1.)
      call matinv(rpb,rpbi,m,m)
      call matmul(rpbi,bpa,f,m,m,l)
c
c     find P(k+1)
c     matrix operation from left
c
      call matran(f,ft,m,l)
      call matqual(at,p,a,apa,l,l,l)
      call matqual(ft,rpb,f,dum,l,m,l)
      call matqual(at,p,a,apa,l,l,l)
      call matadd(apa,dum,dum,l,l,-1.)
      call matadd(dum,q,p,l,l,1.)
c
c     save and decompose f
c
      f11(iz)=f(1,1)
      f12(iz)=f(1,2)
  200 continue
c
c     calculate optimal control and state trajectories
c     and set initial state values
c
      x(1,1)=2.
      x(2,1)=1.
      do 1000 ij=1,nos
      f(1,2)=f12(ij)
      f(1,1)=f11(ij)
      call matmul(f,x,up,m,l,m)
      call matneg(up,u,m,m)
      call matmul(a,x,ax,l,l,m)
      call matmul(b,u,bu,l,m,m)
      call matadd(ax,bu,xk1,l,m,1.)
      icount=ij-1
      write(3,1001) icount,x(1,1),x(2,1),f(1,1),f(1,2),u(1,1)
 1001 format(i5,5f10.4)
 1000 call matequ(xk1,x,l,m)
      stop
      end
      subroutine matequ(mata,matb,lr,lc)
      real mata(lr,lc),matb(lr,lc)
           do 10 i1=1,lr
              do 20 i2 = 1,lc
               matb(i1,i2)=mata(i1,i2)
   20         continue
   10      continue
      return
      end
      subroutine matneg(mata,matb,lr,lc)
      real  mata(lr,lc),matb(lr,lc)
            do 10 i1=1,lr
               do 20 i2=1,lc
                matb(i1,i2)=-mata(i1,i2)
   20          continue
   10       continue
```

```
      return
      end
      subroutine matqua(mata,matb,matc,matd,mm,ll,mm)
      real mata(mm,ll),matb(ll,ll),matc(ll,mm),matd(mm,mm),dum(2,1)
      l=ll
      m=mm
      call matmul(matb,matc,dum,l,l,m)
      call matmul(mata,dum,matd,m,l,m)
      return
      end
      subroutine matqual(mata,matb,matc,matd,mm,ll,kk)
      real mata(mm,ll),matb(ll,ll),matc(ll,kk),matd(mm,kk),dum(2,2)
      call matmul(matb,matc,dum,ll,ll,kk)
      call matmul(mata,dum,matd,mm,ll,kk)
      return
      end
      subroutine matinv(a,ai,n,m)
      real a(n,m),ai(n,m),aa(2,2)
      do 10 i=1,n
         do 20 j=1,m
         aa(i,j)=a(i,j)
 20      continue
 10   continue
      call sgefa(aa,n,m,ipvt,info)
      call sgedi(aa,n,m,ipvt,det,work,01)
      do 1 ii=1,n
         do 2 ij=1,m
         ai(ii,ij)=aa(ii,ij)
  2    continue
  1   continue
      return
      end
      subroutine sgedi(a,lda,n,ipvt,det,work,job)
      integer lda,n,ipvt(1),job
      real a(lda,1),det(2),work(1)
c
c     sgedi computes the determinant and inverse of a matrix
c     using the factors computed by sgeco or sgefa.
c
c     on entry
c
c        a       real(lda, n)
c                the output from sgeco or sgefa.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c        ipvt    integer(n)
c                the pivot vector from sgeco or sgefa.
c
c        work    real(n)
c                work vector.  contents destroyed.
c
c        job     integer
c                = 11   both determinant and inverse.
c                = 01   inverse only.
c                = 10   determinant only.
c
c     on return
c
c        a       inverse of original matrix if requested.
c                otherwise unchanged.
c
```

```
c        det     real(2)
c                determinant of original matrix if requested.
c                otherwise not referenced.
c                determinant = det(1) * 10.0**det(2)
c                with  1.0 .le. abs(det(1)) .lt. 10.0
c                or  det(1) .eq. 0.0 .
c
c     error condition
c
c        a division by zero will occur if the input factor contains
c        a zero on the diagonal and the inverse is requested.
c        it will not occur if the subroutines are called correctly
c        and if sgeco has set rcond .gt. 0.0 or sgefa has set
c        info .eq. 0 .
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas saxpy,sscal,sswap
c     fortran abs,mod
c
c     internal variables
c
      real t
      real ten
      integer i,j,k,kb,kp1,l,nm1
c
c
c     compute determinant
c
      if (job/10 .eq. 0) go to 70
         det(1) = 1.0e0
         det(2) = 0.0e0
         ten = 10.0e0
         do 50 i = 1, n
            if (ipvt(i) .ne. i) det(1) = -det(1)
            det(1) = a(i,i)*det(1)
c        ...exit
            if (det(1) .eq. 0.0e0) go to 60
   10       if (abs(det(1)) .ge. 1.0e0) go to 20
               det(1) = ten*det(1)
               det(2) = det(2) - 1.0e0
            go to 10
   20       continue
   30       if (abs(det(1)) .lt. ten) go to 40
               det(1) = det(1)/ten
               det(2) = det(2) + 1.0e0
            go to 30
   40       continue
   50    continue
   60    continue
   70 continue
c
c     compute inverse(u)
c
      if (mod(job,10) .eq. 0) go to 150
         do 100 k = 1, n
            a(k,k) = 1.0e0/a(k,k)
            t = -a(k,k)
            call sscal(k-1,t,a(1,k),1)
            kp1 = k + 1
            if (n .lt. kp1) go to 90
            do 80 j = kp1, n
               t = a(k,j)
               a(k,j) = 0.0e0
               call saxpy(k,t,a(1,k),1,a(1,j),1)
   80       continue
   90       continue
  100    continue
c
c        form inverse(u)*inverse(l)
c
         nm1 = n - 1
         if (nm1 .lt. 1) go to 140
         do 130 kb = 1, nm1
            k = n - kb
            kp1 = k + 1
            do 110 i = kp1, n
               work(i) = a(i,k)
               a(i,k) = 0.0e0
  110       continue
            do 120 j = kp1, n
               t = work(j)
               call saxpy(n,t,a(1,j),1,a(1,k),1)
  120       continue
            l = ipvt(k)
            if (l .ne. k) call sswap(n,a(1,k),1,a(1,l),1)
  130    continue
  140    continue
  150 continue
      return
      end
      subroutine sgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      real a(lda,1)
c
c     sgefa factors a real matrix by gaussian elimination.
c
c     sgefa is usually called by sgeco, but it can be called
c     directly with a saving in time if  rcond  is not needed.
c     (time for sgeco) = (1 + 9/n)*(time for sgefa) .
c
c     on entry
c
c        a       real(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        info    integer
c                = 0  normal value.
c                = k  if  u(k,k) .eq. 0.0 .  this is not an error
c                     condition for this subroutine, but it does
c                     indicate that sgesl or sgedi will divide by zero
c                     if called.  use  rcond  in sgeco for a reliable
```

```
c                        indication of singularity.
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas saxpy,sscal,isamax
c
c     internal variables
c
      real t
      integer isamax,j,k,kp1,l,nm1
c
c
c     gaussian elimination with partial pivoting
c
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
c
c        find l = pivot index
c
         l = isamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
c
c        zero pivot implies this column already triangularized
c
         if (a(l,k) .eq. 0.0e0) go to 40
c
c           interchange if necessary
c
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
c
c           compute multipliers
c
            t = -1.0e0/a(k,k)
            call sscal(n-k,t,a(k+1,k),1)
c
c           row elimination with column indexing
c
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call saxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0e0) info = n
      return
      end
      subroutine saxpy(n,sa,sx,incx,sy,incy)
```

```
c
c     constant times a vector plus a vector.
c     uses unrolled loop for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      real sx(1),sy(1),sa
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (sa .eq. 0.0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        sy(iy) = sy(iy) + sa*sx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sy(i) = sy(i) + sa*sx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        sy(i) = sy(i) + sa*sx(i)
        sy(i + 1) = sy(i + 1) + sa*sx(i + 1)
        sy(i + 2) = sy(i + 2) + sa*sx(i + 2)
        sy(i + 3) = sy(i + 3) + sa*sx(i + 3)
   50 continue
      return
      end
      subroutine sscal(n,sa,sx,incx)
c
c     scales a vector by a constant.
c     uses unrolled loops for increment equal to 1.
c     jack dongarra, linpack, 3/11/78.
c     modified to correct problem with negative increments, 9/29/88.
c
      real sa,sx(1)
      integer i,ix,incx,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      ix = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      do 10 i = 1,n
        sx(ix) = sa*sx(ix)
        ix = ix + incx
   10 continue
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sx(i) = sa*sx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        sx(i) = sa*sx(i)
        sx(i + 1) = sa*sx(i + 1)
        sx(i + 2) = sa*sx(i + 2)
        sx(i + 3) = sa*sx(i + 3)
        sx(i + 4) = sa*sx(i + 4)
   50 continue
      return
      end
      subroutine sswap (n,sx,incx,sy,incy)
c
c     interchanges two vectors.
c     uses unrolled loops for increments equal to 1.
c     jack dongarra, linpack, 3/11/78.
c
      real sx(1),sy(1),stemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c       code for unequal increments or equal increments not equal
c         to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        stemp = sx(ix)
        sx(ix) = sy(iy)
        sy(iy) = stemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c       code for both increments equal to 1
c
c
c       clean-up loop
c
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        stemp = sx(i)
        sx(i) = sy(i)
        sy(i) = stemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
```

```
         do 50 i = mp1,n,3
           stemp = sx(i)
           sx(i) = sy(i)
           sy(i) = stemp
           stemp = sx(i + 1)
           sx(i + 1) = sy(i + 1)
           sy(i + 1) = stemp
           stemp = sx(i + 2)
           sx(i + 2) = sy(i + 2)
           sy(i + 2) = stemp
      50 continue
         return
         end
         integer function isamax(n,sx,incx)
c
c        finds the index of element having max. absolute value.
c        jack dongarra, linpack, 3/11/78.
c        modified to correct problem with negative increments, 9/29/88.
c
         real sx(1),smax
         integer i,incx,ix,n
c
         isamax = 0
         if( n .lt. 1 ) return
         isamax = 1
         if(n.eq.1)return
         if(incx.eq.1)go to 20
c
c           code for increment not equal to 1
c
         ix = 1
         if(incx.lt.0)ix = (-n+1)*incx + 1
         smax = abs(sx(ix))
         ix = ix + incx
         do 10 i = 2,n
            if(abs(sx(ix)).le.smax) go to 5
            isamax = i
            smax = abs(sx(ix))
       5    ix = ix + incx
      10 continue
         return
c
c           code for increment equal to 1
c
      20 smax = abs(sx(1))
         do 30 i = 2,n
            if(abs(sx(i)).le.smax) go to 30
            isamax = i
            smax = abs(sx(i))
      30 continue
         return
         end
         subroutine matmul(mata,matb,prod,ll,nn,mm)
         real mata(ll,nn),matb(nn,mm),prod(ll,mm),sum
         do 40 i =1,ll
            do 30 j=1,mm
              sum=0.0
               do 20 k=1, nn
                  sum= sum + mata(i,k) * matb(k,j)
      20       continue
              prod(i,j) = sum
      30    continue
      40 continue
         return
         end
         subroutine matadd(mata,matb,sum,nn,mm,code)
      real mata(nn,mm),matb(nn,mm),sum(nn,mm)
      do 40 i=1,nn
         do 30 j=1,mm
         if(code .ge. 0.)   sum(i,j) = mata(i,j) + matb(i,j)
         if(code .lt. 0.)   sum(i,j) = mata(i,j) - matb(i,j)
30       continue
40    continue
      return
      end
      subroutine matran(mata,matb,nn,mm)
      real mata(nn,mm),matb(mm,nn)
      do 40 i=1,mm
         do 30 j=1,nn
          matb(i,j) = mata(j,i)
30       continue
40    continue
      return
      end
```

APPROVAL

OPTIMAL CONTROL COMPUTER PROGRAMS

By F. Kuo

The information in this report has been reviewed for technical content. Review of any information concerning Department of Defense or nuclear energy activities or programs has been made by the MSFC Security Classification Officer. This report, in its entirety, has been determined to be unclassified.

JCBlair

J.C. BLAIR
Director, Structures and Dynamics Laboratory

☆ U. S. GOVERNMENT PRINTING OFFICE 1992 733-050 / 60219