Constrained Differential Dynamic Programming and Its Application to Multireservoir Control

DANIEL M. MURRAY AND SIDNEY J. YAKOWITZ

Mathematics Department and Systems and Industrial Engineering Department, University of Arizona Tucson, Arizona 85721

This paper describes a modification of differential dynamic programming (DDP) which makes that technique applicable to certain constrained sequential decision problems such as multireservoir control problems discussed in the hydrology literature. The authors contend that the method proffered here is superior to available alternatives. This belief is supported by analysis (wherein it transpires that constrained DDP does not suffer the 'curse of dimensionality' and requires no discretization) and computational experimentation (wherein DDP is found to quickly locate solutions of 4-reservoir problems introduced by other investigations as well as the solution of a 10-reservoir problem thought to be beyond the capability of alternative methods).

1. INTRODUCTION

The purpose of this paper is to describe and analyze a successive approximation dynamic programming technique which we believe to be the most efficient procedure available for solving multireservoir control problems. A particularly important feature of the method to be proposed is that in contrast to other methods in the literature, the memory and computational requirements do not grow exponentially with the dimension of the state or decision variables. Another is that discretization of state and control space is not required.

This investigation was motivated by a class of multireservoir control problems which has been intensively attacked in the literature in recent years [*Becker and Yeh,* 1974; *Becker et al.,* 1976; *Chow et al.,* 1975; *Chow and Cortes-Rivera,* 1974; *Heidari et al.,* 1971; *Nopmongcol and Askew,* 1976; *Roefs and Bodin,* 1970; *Schweig and Cole,* 1968; *Sigvaldason,* 1976; *Takeuchi and Moreau,* 1974; *Trott and Yeh,* 1973]. The motivation for such studies is poignantly stated in the following excerpt from *Tennessee Valley Authority* [1974]:

> Reservoir operations are still essentially based on so-called "rule curves" which were developed by multiple objective considerations yet without the benefit of modern system analysis techniques and high speed digital computers. There are indications that significant improvements in water resource management can be achieved by more comprehensive planning and operation procedures using optimization techniques. The annual benefits derived from such improved methodology may easily amount to several million dollars.

Mathematically speaking, in most instances the multireservoir control problem is a discrete-time control problem in which the control sets are polyhedrons. Using the terminology of *Yakowitz* [1969, chapter 2], we will define the 'multireservoir control process' to be a discrete-time control problem such that the set A of decision times is the finite set $\{1, 2, \cdots, N\}$ of integers and the state x and control p are n- and m-tuples, respectively. For any state x and time t, $1 \leq t \leq N$, the set $P(x, t)$ of available controls is a polyhedron. That is, some m-tuple functions $a(x, t)$ and $b(x, t)$ depending continuously on x having been specified, $p \in P(x, t)$ if and only if

$$a(x, t) \leq p \leq b(x, t) \qquad (1)$$

The basic components of a multireservoir control problem are an initial state vector x_1, an additive loss function $J(\{p_t\}_{t=1}^N) = \sum_{t=1}^N L(x_t, p_t, t)$, and a law of motion $T(x, p, t)$, representing the state x_{t+1} in terms of the preceding state and control as

$$x_{t+1} = T(x_t, p_t, t) \qquad (2)$$

The control problem is to find a policy (i.e., a sequence of controls) which satisfies the constraints $p_t \in P(x_t, t)$ associated with the trajectory $\{x_t\}_{t=1}^N$ determined by x_1 and $\{p_t\}_{t=1}^N$ through (2) and which minimizes the loss function $J(\{p_t\})$ over all other policies. In this study the assumption that L and T have continuous second derivatives with respect to state and control will hold throughout. In the context of the multireservoir control problem the states are usually taken to be the amounts of water stored in the various reservoirs in the system, and the coordinates of the controls are the release levels for the reservoirs in the system; we also use these definitions here. We also adopt the assumption (shared by most investigators who have attempted to find numerical results) that the inflows to the reservoir system are somehow known or that we are permitted to work with predicted values. The effect of this assumption is to avoid a stochastic control problem. Except for the section reporting the computational results we will continue to speak of the multireservoir control problem in this system-theoretic language in the hope that our techniques will be accessible to investigators facing other discrete-time control problems with linear constraints, e.g., flows in networks [*Mays and Yen,* 1975].

As in many sequential decision problems, the applicability of dynamic programming to multireservoir control is limited by the severe demands it induces on computer memory and time. In particular, the state of the art in the previously published literature seems to be four to six reservoirs. *Heidari et al.* [1971, p. 273] attest to this fact in stating, 'One disadvantage of dynamic programming is that it requires a high-speed memory that is beyond the capacity of known computers when the dimensionality is higher than four or five.' In the same vein, *Becker et al.* [1976, p. 108] have asserted the following:

> Although the application of the incremental dynamic programming technique to this four-reservoir system is successful, a problem will exist when Auburn and New Melones Dams are added to the system. Each time a new reservoir is added to the system, the incremental dynamic programming algorithm exponentially increases computational requirements. With the addition of both reservoirs, the dimensionality is beyond the central memory capabilities of the modern-day computer.

The central developments of this paper are now briefly surveyed. In section 2, we give a brief review of the various

Paper number 9W0519.
0043-1397/79/009W-0519$01.00

optimization techniques which are applicable to multireservoir control. They include nonlinear programming, dynamic programming, state increment dynamic programming, discrete differential dynamic programming, and differential dynamic programming, the latter technique being new to the hydrology literature, as far as we know. In section 3, the differential dynamic programming algorithm is carefully explained, and some of its promising properties are discussed. The most important property is that it alleviates the 'curse of dimensionality' in the sense that the memory and computational requirements grow proportionally to k^α, where $k = \max\{m, n\}$, m and n being the dimensions of the control and state variables, instead of exponentially in k, as in discrete dynamic programming and the discrete differential dynamic programming method advanced by *Heidari et al.* [1971]. In our computations, $m = n$ and $\alpha = 2$ for memory requirements and $\alpha = 3$ for arithmetic operation counts. Another important advantage is that in contrast to discrete dynamic programming and discrete differential dynamic programming, discretization of state and control spaces is not required. Finally, differential dynamic programming achieves quadratic convergence, whereas the other applicable successive approximation methods are suspected to yield at best only linear convergence. It is noted, parenthetically, that the theory of differential dynamic programming seems to be more extensive and complete than that of alternative successive approximation schemes. In section 4, we give a specific prescription for using a quadratic programming technique to enable differential dynamic programming to handle linear constraints. To the best of our knowledge, this is the first discussion of constrained differential dynamic programming in the discrete-time control context. In sections 2–4, we believe that we build a strong case for concluding that differential dynamic programming has attractive analytic properties for obtaining the numerical solution of the multireservoir problem. In section 5, we compare the computational performance of our constrained differential dynamic programming (DDP) algorithm with that of a discrete differential dynamic programming (DDDP) study published by *Chow and Cortes-Rivera* [1974]. Differential dynamic programming obtained a better value after 2 iterations than DDDP did after 20 iterations, both procedures beginning at the same nominal policy. Then we describe a successful application of differential dynamic programming to a system of 10 reservoirs.

2. A SYNOPSIS OF METHODOLOGY FOR THE MULTIRESERVOIR CONTROL PROBLEM

Let us summarize the multireservoir control problem defined in section 1. We take the time set A to be the first N positive integers. The object is to choose the policy $\mathbf{p} = (p_1, \cdots, p_N)$ to minimize the separable loss function

$$J(\mathbf{p}) = \sum_{t=1}^{N} L(x_t, p_t, t) \tag{3}$$

where the x_t's satisfy $x_{t+1} = T(x_t, p_t, t)$, x_1 is assumed given, and the controls p_t are constrained to lie in the polyhedron $P(x, t) = \{p: a(x, t) \leq p \leq b(x, t)\}$. The states x_t are presumed to be n-tuples, and the controls p_t are m-tuples.

In principle, one may solve this problem by the methods of constrained nonlinear programming (such as described by *Luenberger* [1973], *Adby and Dempster* [1974], and *Gill and Murray* [1974]). In this approach, one would ignore the dynamic structure of the problem and iteratively minimize the loss function J, viewed as a function of the mN variables $\mathbf{p} = (p_1, \cdots, p_N)$. But there are some very real difficulties in following this approach. One is that the description of the constraint set in terms of $\mathbf{p}$ alone is very cumbersome. Another disadvantage of nonlinear programming methods, in comparison to stagewise methods, which use the dynamics and focus on a single stage at a time, is that the former methods, (such as quasi-Newton methods) are typically formulated in terms of an Nth order matrix projection of the gradient vector. Therefore such methods require effort proportional to N^2, while stagewise methods require effort linear in N. These considerations point to the need for a stagewise method for control problems.

The dynamic programming method is a stagewise procedure which, in principle, enables one to determine the optimal solution. In essence, this method requires the construction of $p_t(x)$ for $t = N, N-1, \cdots, 1$, where for each x,

$$p_t(x) = \arg\min_{p} [L(x, p, t) + V_{t+1}(T(x, p, t))] \tag{4}$$

the optimal value functions V_t being determined recursively by $V_{N+1}(x) = 0$ and

$$V_t(x) = L(x, p_t(x), t) + V_{t+1}(T(x, p_t(x), t)) \tag{5}$$

The functions $p_t(x)$ having been computed for $t = N, N-1, \cdots, 1$, the optimal policy for the problem can be proven to be the policy $\mathbf{p}^* = (p_1^*, p_2^*, \cdots, p_N^*)$, where $p_1^* = p_1(x_1)$, $p_2^* = p_2^*(T(x_1, p_1^*, 1))$, etc. Now the basic trouble with this prototypical dynamic programming algorithm is that with the exception of certain rare instances the functions $p_t(x)$ and $V_t(x)$ cannot be conveniently represented in the computer. That is, these functions cannot be stored accurately and with a minimum of memory requirement in the computer, as they must be in order to undertake numerical implementation of the dynamic programming procedure. *Bellman* [1957, 1971], *Bellman and Dreyfus* [1962], *Larson* [1968], and other investigators have tried to circumvent the computer representation problem by various ingenious schemes. The most obvious approach to the representation problem is to define the V_t and p_t functions only on a finite set of grid points. As Bellman observed in his earliest studies of dynamic programming, this approach, known as 'discrete dynamic programming,' leads to the 'curse of dimensionality' in that if each coordinate of the vector x is constrained to a set of K values, the number of possible states is K^n. Thus for a given magnitude of discretization the number of states grows exponentially with the dimension n of the state, and since V_t and p_t must be evaluated and stored for each state, we may conclude that both the memory and computational requirements grow exponentially with n. In the reservoir control problem this implies that all other factors remaining the same, the memory and computational burden grows exponentially with the number of reservoirs in a system.

The avenues toward overcoming the limitations of dynamic programming which are imposed by the computer representation problem and the curse of dimensionality are techniques known collectively as 'successive approximation methods' (or, in the terminology of numerical analysis, 'iteration methods'). Such methods have been studied by *Bellman* [1957] and *Bellman and Dreyfus* [1962, Appendix 3].

The successive approximation method which the *Tennessee Valley Authority* [1974, 1976] study found to be most effective for multireservoir control was developed under the nomenclature 'state increment dynamic programming' by *Larson* [1968, chapter 12] and 'theoretical properties,' by *Larson and Korsak* [1970]). It was refined and exploited by *Trott and Yeh* [1973] and has been termed by hydrologists the 'dynamic programming successive approximations' (DPSA) technique. It is basically a procedure for successive approximation on state trajectory space. Let $\mathbf{x}^{(0)}$ denote some nominal trajectory. In the DPSA method, one first chooses an index j, $1 \leq j \leq n$, and then constructs a new trajectory $\mathbf{x}^{(1)}$ which minimizes the loss function, subject to the constraint that for each decision time t, $x_{i,t}^{(1)} = x_{i,t}^{(0)}$, all $i \neq j$. The method requires that the state space be discretized and that the law of motion $y = T(x, p, t)$ be a one-to-one mapping of controls p into states y, for every fixed x and t. (This requirement will essentially imply that $m = n$.) In essence, the DPSA method succeeds in overcoming the curse of dimensionality by optimizing with respect to one variable at a time. A convergence proof for DPSA covering the multireservoir problem studied in our section 5 is given by *Larson and Korsak* [1970].

It is the opinion of the present authors that while it is the state of the art, DPSA has some distinct weaknesses. One is that the inverse mapping of the law of motion must be evaluated. For reservoirs in which there is branching (more decision variables (releases) than state variables (reservoir storages)) the inverse will not exist. When it does exist, if the law of motion is not linear, its computation can be anticipated to be expensive inasmuch as solution of multivariable nonlinear equations will be required. Even in the linear dynamics case (the only situation that we are aware of in which DPSA has been applied), computation of this inverse will require solution of an $(n - 1)$th order linear equation, a task whose computational requirements grow as n^3. Thus in this case, computational effort grows proportionally to n^3, not n, as claimed by *Larson* [1968] and *Trott and Yeh* [1973]. Actually, in order for DPSA to be comparable to other successive approximation techniques it would seem fair to count an iteration as being a repetition of DPSA for each state variable, in which case the computational burden grows as n^4, in the linear dynamics case.

A second drawback of DPSA is that it is closely related to the 'coordinate descent method' (CDM) [e.g., *Luenberger*, 1973, section 7.8]. (If the time horizon is 1, DPSA coincides with the coordinate descent method.) In the quadratic loss function case the best of CDM's (Gauss-Southwell) has a worse coefficient of convergence after n iterations than the gradient method has after one iteration. Furthermore, the gradient method itself has a slower order of convergence (linear) than DDP (quadratic). A computer study reported by *Tennessee Valley Authority* [1974] found that about 200 DPSA iterations were required for acceptable convergence. We conjecture that this large number of requisite iterations might have been anticipated for the theoretical reasons noted above.

Nevertheless, the DPSA method has merit in that it does overcome the curse of dimensionality and it has exhibited convergence for systems of six reservoirs [*Trott and Yeh*, 1973].

We will delay until the next section our detailed development of the successive approximation method, which we think is most promising, from both an analytic and a computational standpoint. This method, which is called 'differential dynamic programming,' was developed by *Mayne* [1966] and *Jacobson and Mayne* [1970]. Recently, *Murray* [1978] has developed some inexpensive computer implementations of the method and its convergence properties. We assert, at this point, the main advantages of differential dynamic programming.

The most important advantage, in our estimation, is that it provides a substantial step toward overcoming the curse of dimensionality: the memory requirement grows proportionally to n^2, and the computational burden grows as n^3. Next, no discretization of either state or control space is required. Then, since no decomposition is performed in order to consider single stages separately, the coupling among states is preserved. Finally, we note that differential dynamic programming achieves quadratic convergence of successive policy iterations to the optimal policy. That is, if $\{\mathbf{p}^{(v)}\}$ denotes successive policy vectors determined by differential dynamic programming and $\mathbf{p}^*$ denotes the optimal policy, then in Euclidean norm, provided $\|\mathbf{p}^{(0)} - \mathbf{p}^*\|$ is 'small enough,' for some constant C,

$$\|\mathbf{p}^{(v+1)} - \mathbf{p}^*\| \leq C\|\mathbf{p}^{(v)} - \mathbf{p}^*\|^2 \qquad v = 0, 1, \cdots \qquad (6)$$

We are unaware of any derivations of convergence rates in a general setting for the other stagewise successive approximation techniques in the literature, but we suspect that since (as *Bellman* [1971, 1957] has noted) they are closely related to Picard's method of successive approximations, like Picard's method they have linear convergence. (The linear convergence of Picard's method is proven, for instance, by *Szidarovszky and Yakowitz* [1978, section 8.1.2].)

Heidari et al. [1971], *Chow and Cortes-Rivera* [1974], and *Chow et al.* [1975] have developed what they call a discretized version of differential dynamic programming in connection with their studies of the multireservoir control problem. However, because of this discretization, they lose many advantages of differential dynamic programming. In particular, they are again beset with the curse of dimensionality in that the memory and the number of operations grow exponentially with the dimension of control space.

We have indicated that the use of discretizations invariably leads either to the curse of dimensionality or a decomposition scheme such as DPSA. To avoid such discretizations, one needs to consider methods which employ either first- or second-order expansions of the performance functional. The gradient method, a first-order method, is a notoriously poor method. A constrained version of the gradient method, the reduced gradient method, was found to perform worse than DPSA by the *Tennessee Valley Authority* [1974]. Other nonlinear programming methods, e.g., conjugate gradient or quasi-Newton methods, suffer from the problem of describing the constraints entirely in terms of $\mathbf{p}$. The opportunity of using a stagewise method which avoids discretizations and uses second-order expansions makes DDP an attractive method for constrained optimization problems.

3. UNCONSTRAINED DIFFERENTIAL DYNAMIC PROGRAMMING

In this section the basic differential dynamic programming algorithm is presented. The algorithm is the same as that given by *Jacobson and Mayne* [1970, chapter 4], and the asserted quadratic convergence results are due to *Murray* [1978].

At each iteration the DDP technique requires stagewise approximation of the term $L(x, p, t) + V_t(T(x, p, t))$ defined in connection with (4) by a quadratic in x and p. The optimal feedback control function $p_t(x)$ is then approximated by a linear function. As a result of these approximations and the associated computations, the DDP method is best introduced in terms of the application of prototypical dynamic programming to a linear dynamics quadratic performance (LQP) problem. It is important to recognize that the dynamic programming solution of such a problem can be found exactly (ignoring the effect of roundoff errors) by arithmetic operations. Furthermore, the memory requirement grows only quadratically in n and m (the dimension of state and control space, respectively), and the number of arithmetic operations required by the dynamic programming solution to the LQP problem grows cubically in m. These facts are easily confirmed by examination of the algorithm to be described next. After we have given the dynamic programming solution of the LQP problem, methodologies are presented for the iterative solution of non-LQP problems by algorithms resembling the dynamic programming solution of LQP problems. This process constitutes DDP. This section will conclude with some results concerning convergence rates.

An LQP problem is characterized by the fact that the loss function $J(\mathbf{p})$ is a positive definite quadratic and by the additional property that the law of motion T determines a linear system. Thus letting the superscript T denote 'transpose,' we have

$$L(x, p, t) = x^T A_t x + p^T B_t x + p^T C_t p + D_t^T p + E_t^T x \qquad (7)$$

and

$$T(x, p, t) = F_t x + G_t p \qquad (8)$$

where A_t and F_t are $n \times n$ matrices, B_t^T and G_t are $n \times m$, C_t is $m \times m$, and D_t and E_t are m- and n-tuples. A_t and C_t are assumed symmetric. *Bellman* [1967] provides much ancillary information concerning this important class of control problems.

Let x be a fixed state. The dynamic programming solution of the LQP problem proceeds as follows: A necessary and sufficient condition [*Luenberger*, 1973, chapter 6] for the control p^* to minimize $L(x, p, N)$ is that the gradient of L, with respect to p, be the zero vector at p^*. Thus

$$\nabla_p L(x, p, N) = 2C_N p^* + B_N x + D_N = 0 \qquad (9)$$

from which we obtain the optimum feedback control $p^* = p_N(x)$ by the relation

$$p_N(x) = \tfrac{1}{2} C_N^{-1}(D_N + B_N x) \qquad (10)$$

We write $p_N(x) = \alpha_N + \beta_N x$ where the vector α_N and matrix β_N are determined by (10) and must be stored in memory. This requires $m(n + 1)$ memory locations. If Gaussian elimination is used to calculate C_N^{-1}, the number of arithmetic operations required to compute α_N and β_N grows proportionally to m^3 [*Szidarovszky and Yakowtiz*, 1978, chapter 6]. The reader will easily confirm from (7) and (10) that the optimal value function

$$V_N(x) = L(x, p_N(x), N) \qquad (11)$$

is quadratic. Its coefficients must be stored until the next stage of computation is completed. Let us proceed inductively, by assuming that the optimal return function V_{t+1} is a known, positive definite quadratic form. Then the function

$$Q_t(x, p) = L(x, p, t) + V_{t+1}(F_t x + G_t p) \qquad (12)$$

is readily seen to be a positive definite quadratic form in x and p, and the optimal control may be calculated, as in the $t = N$ case, to have the representation

$$p_t(x) = \alpha_t + \beta_t x \qquad (13)$$

The coefficients of α_t and β_t are stored, and then the positive definite quadratic optimal value function $V_t(x) = Q_t(x, \alpha_t + \beta_t x)$ is computed and its coefficients stored (in place of those for V_{t+1}). This process is continued until all the terms (α_t, β_t) $1 \le t \le N$ are found. Then the optimal policy $\mathbf{p}^* = (p_1^*, p_2^*, \cdots, p_N^*)$ for the LQP problem is obtained by successively calculating

$$p_t^* = \alpha_t + \beta_t x_t \qquad (14)$$

and

$$x_{t+1} = F_t x_t + G_t p_t^* \qquad (15)$$

x_1 being, of course, one of the problem parameters. This process for obtaining the policy $\mathbf{p}^*$ from $p_t(x)$, $1 \le t \le N$, will be referred to as the 'forward run.'

The DDP procedure for using LQP-type formulae for non-LQP problems is now discussed. The first step requires selection of an arbitary policy $\bar{\mathbf{p}} = (\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_N)$, which is referred to as the 'nominal policy,' and the trajectory $\bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_N)$ generated by the policy, through the law of motion and the given initial state, is termed the 'nominal trajectory.' The technique which is presented by *Jacobson and Mayne* [1970] begins by obtaining a Taylor's series approximation, up to second-order terms, of $L(x, p, N)$ about $(\bar{x}_N, \bar{p}_N)$. This approximation is denoted by $\tilde{L}(x, p, N)$. One then calculates α_N and β_N, as in the LQP case, using the quadratic $\tilde{L}(x, p, N)$. The approximation of the optimal value function is then taken to be

$$V_N(x) = \tilde{L}(x, p_N(x), N) \qquad (16)$$

where $p_N(x) = \alpha_N + \beta_N x$. One then proceeds inductively. Assume that the quadratic approximation $V_{t+1}(x)$ of the optimal value function has been constructed. Define the quadratic $Q(x, p)$ to be the terms, up to quadratic, of the Taylor's series expansion of $L(x, p, t) + V_{t+1}(T(x, p, t))$, the expansion being taken about $(\bar{x}_t, \bar{p}_t)$. Let $p_t(x) = \alpha_t + \beta_t x$ be the linear feedback law minimizing $Q(x, p)$, where α_t and β_t are obtained as in the LQP problem, and define

$$V_t(x) = Q(x, p_t(x)) \qquad (17)$$

This completes the construction of the tth step. The pairs $\{(\alpha_t, \beta_t)\}_{t=1}^N$ of feedback parameters having been constructed according to the procedure just outlined, a new nominal policy $\bar{\mathbf{p}}'$ is determined by the recursive relationship (with $\bar{x}_1' = x_1$),

$$\bar{p}_t' = \alpha_t + \beta_t \bar{x}_t' \qquad \bar{x}_{t+1}' = T(\bar{x}_t', \bar{p}_t', t) \qquad (18)$$

and this new nominal policy $\bar{\mathbf{p}}'$ plays the role of $\bar{\mathbf{p}}$ in the next iteration of the policy construction algorithm.

Jacobson and Mayne [1970] note that because second-order approximation may not be accurate enough, the successor policy $\bar{\mathbf{p}}'$ may actually yield a worse policy than $\bar{\mathbf{p}}$, so they modify (18) according to the rule

$$\bar{p}_t' = \bar{p}_t + \epsilon\alpha_t + \beta_t x_t \qquad x_{t+1} = T(x_t, \bar{p}_t', t) \qquad (19)$$

where ϵ is some positive number less than or equal to 1. If the policy $\bar{\mathbf{p}}'$ yields a larger loss function value $J(\bar{\mathbf{p}}')$ than does $\bar{\mathbf{p}}$, the value of ϵ is reduced by half, and the process (19) repeated until $J(\bar{\mathbf{p}}') < J(\bar{\mathbf{p}})$. If at any stage, the quadratic form $Q_t(x, p)$ in (12) is not positive definite in p, then the 'direction' determined by DDP iteration may not yield an improvement, and the sequence of steps (19) with smaller and smaller ϵ may never terminate. The method of 'quadraticizing' in order to obtain the requisite quadratic forms for V_t and Q_t is subject to variations from the procedure of *Jacobson and Mayne* [1970]. *Murray* [1978] has provided an alternative approximation which assures that at each iteration all quadratics $Q_t(x, p)$ are positive definite in p and are tangent to $L(x, p, t) + V_t(T(x, p, t))$ at the nominal state and control. This property assures that the iteration process in connection with (19) must terminate.

Murray [1978] has obtained a stagewise recurrence representation of Newton's method for solving dynamic programming problems. Also, he has shown that up to second-order terms, Newton's method and DDP are equivalent. The implication of this observation is that the well-known quadratic convergence property of Newton's method [e.g., *Szidarovszky and Yakowitz,* 1978, chapter 5] implies that DDP is also quadratically convergent. That is, (6) holds. Previously, *Jacobson and Mayne* [1970] had only sketched a demonstration of convergence of DDP by alluding to a general nonlinear programming theorem [e.g., *Luenberger,* 1973, section 6.5] that says, in effect, that any algorithm which is continuous and yields an improvement at each iteration converges to a stationary policy. Such a line of reasoning does not lead to rates of convergence.

4. Differential Dynamic Programming With Linear Constraints

Jacobson and Mayne [1970] do not specifically spell out a methodology to account for constraints in discrete time DDP (although constraints are discussed in their exposition of the continuous time version of DDP). Consequently, the procedure that follows is believed to be a new extension of DDP theory.

The constrained DDP procedure begins, as before, with a quadratic approximation $\hat{L}(x, p, N)$ of the Nth stage loss function $L(x, p, N)$. But now, one must find the value $p_N^* = p_N(x)$ which minimizes $\hat{L}(x, p, N)$ subject to $a(x, N) \leq p \leq b(x, N)$. This is a quadratic programming problem. We will assume that $a(x, t)$ and $b(x, t)$ are continuously differentiable in x for every t. One may then confirm that the optimizing control $p_N(x)$, as a function of the state x, is a piecewise linear function, the 'pieces' corresponding to changes in active constraints as x varies. If the entire set of coefficients of the piecewise linear function $p_N(x)$ were to be stored, the storage requirement would grow exponentially with the dimension of state and control space, and consequently, one of the chief benefits of DDP would be lost. However, it is sufficient to store those coefficients corresponding to the portion of $p(\underline{x})$ which is valid for the constraints which are active when $x = \bar{x}_N$, where, as in the previous section, the overbar denotes the nominal state. By using this portion for the representation $p_N(x) = \alpha_N + \beta_N x$, the quadratic approximation of the optimal value function is then given by $V_N(x) = \hat{L}(x, p_N(x), N)$, as in the unconstrained case. For intermediate stages this process is repeated with the objective function $Q_t(x, p)$ being the quadratic approximation to $L(x, p, t) + V_{t+1}(T(x, p, t))$ and the decision vectors p constrained by the condition that

$$a(x, t) \leq p \leq b(x, t) \tag{20}$$

One must confront two primary procedural aspects in implementing the preceding constrained DDP algorithm:

1. A routine must be chosen for solving the quadratic programming problem which arises at every decision stage t of a DDP iteration. In order not to erode the computational advantage of DDP this routine should have computational complexity no greater than $O(m^3)$ operations, m being the dimension of the decision vector. (By $O(m^3)$ we mean some function which, for each m, is smaller than am^3, a being some positive number which does not depend on m.)
2. In implementing the forward run to obtain the successor policy, precautions must be taken to make sure that no constraints are violated.

The procedure chosen for solving quadratic programming problems was *Fletcher*'s [1971] quadratic programming algorithm. *Fletcher* [1971, p. 90] asserts that the number of arithmetic operations required by his algorithm is $O(m^3)$, and our experience is in keeping with this assertion. At each stage, *Wolfe*'s [1965] feasible point generator was employed to obtain a feasible control for 'starting' the Fletcher method.

In order to be more specific about the technique used to obtain the computed results described in section 5, the basic technique of *Fletcher*'s [1971] method is now sketched out.

The basic quadratic programming problem is to minimize a quadratic function of m variables subject to v linear constraints. That is,

$$\min f(p) = \tfrac{1}{2} p^T F p - b^T p \tag{21}$$

subject to

$$\hat{C}^T p \geq \hat{d} \tag{22}$$

Each column of the $m \times v$ matrix $\hat{C}$ is the normal vector of one of the constraint surfaces. Fletcher's approach is based on a sequence of so-called equality problems. An 'equality problem' is the task of minimizing the quadratic function $f(p)$ subject to k (with $k \leq v$) independent linear equality constraints

$$C^T p = d \tag{23}$$

A Lagrangian $\mathcal{L}$ as a function of p and the k additional variables λ is defined as $\mathcal{L}(p, \lambda) = \frac{1}{2} p^T F p - b^T p + \lambda^T (C^T p - d)$.

The method of Lagrange multipliers [*Luenberger,* 1973] implies that if control p minimizes $f(p)$, subject to (22), then there is a vector λ such that $(p, \lambda)^T$ satisfies the linear system

$$\begin{bmatrix} F & C \\ C^T & 0 \end{bmatrix} \begin{bmatrix} p \\ \lambda \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix} \tag{24}$$

If we introduce the notation

$$C^* = (C^T F^{-1} C)^{-1} C^T F^{-1} \qquad H^* = F^{-1}(I - CC^*) \tag{25}$$

and use the formula for the inverse of a partitioned matrix [e.g., *Szidarovszky and Yakowitz,* 1978, section 6.1.3]

$$\begin{bmatrix} F & C \\ C^T & 0 \end{bmatrix}^{-1} = \begin{bmatrix} H^* & C^{*T} \\ C^* & -C^T(F^{-1}C)^{-1} \end{bmatrix} \tag{26}$$

then the solution $\hat{p}$, $\hat{\lambda}$ of (24) may be written as

$$\hat{p} = H^* b + C^{*T} d \qquad \hat{\lambda} = C^* b - C^T(F^{-1}C)^{-1} d \tag{27}$$

By introducing, exchanging, and dropping constraints from the set of active constraints, *Fletcher*'s [1971] method requires solution of a sequence of equality problems. Every time a solution of an equality problem is obtained, the signs of the λ's associated with the currently active constraints are tested. According to Kuhn-Tucker theory [*Luenberger,* 1973, p. 233] the optimal point of the general inequality problem (21), (22) is characterized by the fact that all the multipliers λ are nonpositive.

Having outlined the Fletcher algorithm for the quadratic programming problem, we now relate its use in conjunction with DDP. Assume that somehow we have obtained a quadratic approximation $V_{t+1}(x)$ of the optimal value function for the constrained problem initiated at state time $(x, t + 1)$. We then desire to obtain the control $p^* = p_t(x)$, which minimizes $Q_t(x, p)$, where $Q_t(x, p)$ is a quadratic approximation of $L(x, p, t) + V_{t+1}(T(x, p, t))$, the approximation being taken about $(\bar{x}_t, \bar{p}_t)$. The control p is subject to the constraint that

$$a(\bar{x}_t, t) \leq p \leq b(\bar{x}_t, t) \tag{28}$$

where $\bar{x}_t$ is the tth state of the nominal trajectory. If, analogously to (7), we write

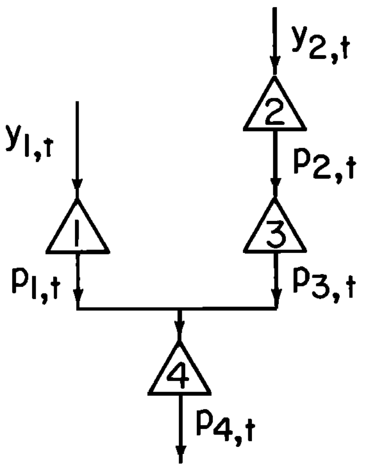

Fig. 1. Four-reservoir configuration.

$$Q_t(x_t, p_t) = \tfrac{1}{2}x_t^TD_tx_t + x_t^TE_tp_t + \tfrac{1}{2}p_t^TF_tp_t + G_t^Tx_t + H_t^Tp_t \quad (29)$$

then upon substituting the nominal value $\bar{x}_t$ of the state vector for x_t, the problem of finding $p_t(\bar{x}_t)$ assumes the form

$$\min \tfrac{1}{2}\, p_t^TF_tp - \tilde{q}^Tp \quad (30)$$

subject to

$$\tilde{C}^Tp \geq \tilde{d} \quad (31)$$

$$\tilde{q} = -H_t - E_t^T\bar{x}_t \qquad \tilde{C}^T = \begin{bmatrix} I_m \\ -I_m \end{bmatrix} \quad (32)$$

and

$$\tilde{d} = \begin{bmatrix} a(\bar{x}_t, t) \\ -b(\bar{x}_t, t) \end{bmatrix} \quad (33)$$

In (32) we have let I_m denote the mth order identity matrix. The optimal point having been obtained by Fletcher's method, the solution p_t^*, λ^* satisfies

$$\begin{bmatrix} F_t & C \\ C^T & 0 \end{bmatrix}\begin{bmatrix} p_t^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} \bar{q} \\ \bar{d} \end{bmatrix} \quad (34)$$

where the active constraints determine C and $\bar{d}$. Also, we have

$$C^Tp_t = \bar{d} \quad (35)$$

Additionally, in the notation of (26),

$$p_t^* = [H^*, C^{*T}]\begin{bmatrix} \bar{q} \\ \bar{d} \end{bmatrix} \quad (36)$$

If we reintroduce the variability of the state vector by letting

$$\tilde{q} = -H_t - E_t^Tx_t \quad (37)$$

$$\tilde{d} = \begin{bmatrix} a(\bar{x}_t, t) + J_a(x)(x_t - \bar{x}_t) \\ b(\bar{x}_t, t) + J_b(x)(x_t - \bar{x}_t) \end{bmatrix} = P + Rx_t \quad (38)$$

where $J_a(\bar{x}_t)$ and $J_b(\bar{x}_t)$ are the Jacobian matrices evaluated at $\bar{x}_t$ of $a(x, t)$ and $b(x, t)$, respectively, then we obtain the control law

$$p_t(x_t) = [H^*, C^{*T}]\left\{\begin{bmatrix} -H_t \\ P \end{bmatrix} + \begin{bmatrix} -C_t^T \\ R \end{bmatrix}x_t\right\} \triangleq \alpha_t + \beta_tx_t \quad (39)$$

TABLE 1. Loss Function Coefficients

t	$c_{1,t}$	$c_{2,t}$	$c_{3,t}$	$c_{4,t}$
1	−1.1	−1.4	−1.0	−2.6
2	−1.0	−1.1	−1.0	−2.9
3	−1.0	−1.0	−1.2	−3.6
4	−1.2	−1.0	−1.8	−4.4
5	−1.8	−1.2	−2.5	−4.2
6	−2.5	−1.8	−2.2	−4.0
7	−2.2	−2.5	−2.0	−3.8
8	−2.0	−2.2	−1.8	−4.1
9	−1.8	−2.0	−2.2	−3.6
10	−2.2	−1.8	−1.8	−3.1
11	−1.8	−2.2	−1.4	−2.7
12	−1.4	−1.8	−1.1	−2.5

The computation for the backward run for decision time t is completed by observing that the quadratic approximation to the optimal return function is given by

$$V_t(x) = Q_t(x, \alpha_t + \beta_tx) \quad (40)$$

For the forward run we have found that the procedure (18) for computing the successor nominal policy must be modified, the modification being motivated by the fact that successor policies may 'step outside' the constraint region, and so application of (19) may not work. The approach which overcomes this problem requires storing the coefficients of the quadratic $V_t(x)$ for $t = 1, 2, \cdots, N$. Assuming that the feasible successor controls $\bar{p}_1', \bar{p}_2', \cdots, \bar{p}_{t-1}'$, and the associated nominal trajectory $\bar{x}_1', \bar{x}_2', \cdots, \bar{x}_t'$ for the successor, up to decision time t, have already been calculated, the successor decision $\bar{p}_t'$ is chosen to be the solution of the quadratic programming problem

$$\min Q_t(\bar{x}_t', p) \quad (41)$$

subject to

$$a(\bar{x}_t', t) \leq p \leq b(\bar{x}_t', t) \quad (42)$$

In (41), $Q_t(\bar{x}_t', p)$ is the quadratic approximation in p of $L(\bar{x}_t', p) + V_{t+1}(T(\bar{x}_t', p))$. This procedure assures us that $\bar{p}_t'$ is feasible, and consequently, the successor policy $\bar{p}'$ will be feasible. It may happen, however, that the successor policy

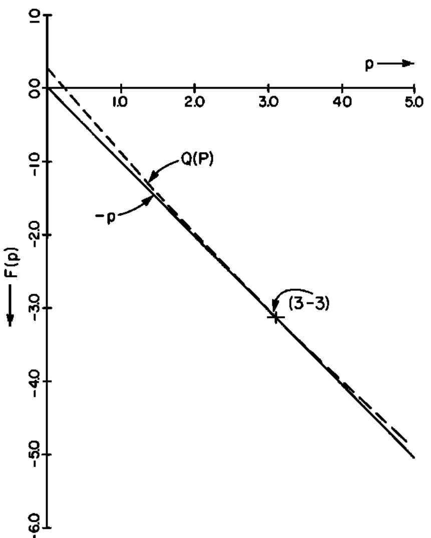

Fig. 2. A quadratic approximation to the objective function.

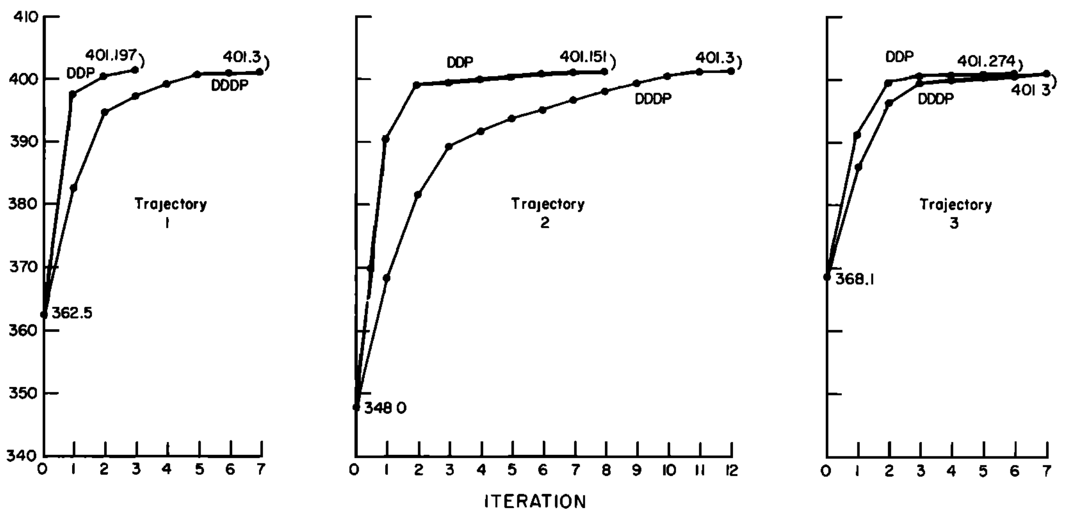

Fig. 3. Computational results for problem 1.

results in a worse objective value of J than the original policy. As we have explained, this possibility arises from the inaccuracies of the quadratic approximation (especially in early DDP iterations when the nominal policy is still far from optimal). A solution for this happenstance is to replace the objective function (41) at each step $t = 1, 2, \cdots, N$ with the modified objective function

$$\min Q_t(\bar{x}_t', p) + \tfrac{1}{2}(p - \bar{p}_t)^T \Gamma (p - \bar{p}_t) \qquad (43)$$

where Γ is a positive definite symmetric matrix. This modification, which is familiar in the control theory literature [e.g., *Jarmark,* 1976], will ensure that the successor control $\bar{p}_t'$ remains close in Euclidean norm to the nominal control $\bar{p}_t$, and this, in turn, assures that if the the eigenvalues of Γ are sufficiently positive, the successor policy $\bar{\mathbf{p}}'$ will achieve an improvement over the nominal policy $\bar{\mathbf{p}}$, provided, of course, that $\bar{\mathbf{p}}$ does not already satisfy the Kuhn-Tucker conditions.

5. Computational Studies in Multireservoir Control

The constrained DDP algorithm was applied to three different multireservoir control problems. The first of these problems was introduced into the literature by *Larson* [1968] and subsequently served as an illustrative example for discrete differential dynamic programming in an investigation by *Heidari et al.* [1971] as well as an example of multilevel incremental dynamic programming in the work of *Nopmongcol and Askew* [1976]. The second example was presented by *Chow and Cortes-Rivera* [1974] to illustrate DDDP with adaptive corridor width selection. Both of these examples are four-reservoir control problems, the first being particularly amenable to DDDP in that 'corridor width' selection can be dispensed with. Our final computational study provides the solution of a control problem having 10 reservoirs. It is intended to be a generalization of the second problem with regard to structure and complexity.

Multireservoir Control Problem 1. The discussion of this first problem is particularly lengthy because here some fine points (applicable to all our examples) of DDP implementation are described.

It is presumed that the four reservoirs comprising the system have the configuration shown in Figure 1. With respect to this figure, $p_{i,t}$, $x_{i,t}$, and $y_{i,t}$ denote the release, the beginning storage, and the inflow to the ith reservoir, respectively, at decision time t. The law of motion for this process is given by

$$x_{t+1} = T(x_t, p_t, t) = x_t + y_t + Mp_t \qquad (44)$$

where $x_t = (x_{1,t}, \cdots, x_{4,t})^T$, y_t and p_t being analogously defined, and M is a fourth-order matrix with -1's down the diagonal and $+1$ in the position (k, v) if reservoir k releases into reservoir v and zeros elsewhere. The vector y_t of inflows has 0's in locations corresponding to reservoirs with no inflow. Specifically, for the problem at hand the law of motion (44) can be written

$$\begin{bmatrix} x_{1,t+1} \\ x_{2,t+1} \\ x_{3,t+1} \\ x_{4,t+1} \end{bmatrix} = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \\ x_{4,t} \end{bmatrix} + \begin{bmatrix} y_{1,t} \\ y_{2,t} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} p_{1,t} \\ p_{2,t} \\ p_{3,t} \\ p_{4,t} \end{bmatrix} \qquad (45)$$

Additionally, state constraints are imposed to keep the storage nonnegative and within the reservoir capacity. The state space constraints were taken to be

$$\begin{aligned} 0 \le x_{1,t} \le 10 \qquad 0 \le x_{2,t} \le 10 \\ 0 \le x_{3,t} \le 10 \qquad 0 \le x_{4,t} \le 15 \end{aligned} \qquad (46)$$

TABLE 2. Computational Effort for Problem 1

Initial Trajectory	DDP			DDDP		
	Time, s	Iterations	Final Value	Time, s	Iterations	Final Value
1	9.63	3	401.197	35.32	8	401.3
2	19.3	6	401.274	48.39	13	401.3
3	25.7	8	401.151	31.04	8	401.3

TABLE 3*a*. Inflow Values $y_{j,t}$

	Value of j	
t	1	2
1	0.5	0.4
2	1.0	0.7
3	2.0	2.0
4	3.0	2.0
5	3.5	4.0
6	2.5	3.5
7	2.0	3.0
8	1.25	2.5
9	1.25	1.3
10	0.75	1.2
11	1.75	1.0
12	1.0	0.7

TABLE 4. Initial Trajectory for Problem 2

	Value of j			
t	1	2	3	4
1	6.0	6.0	6.0	8.0
2	6.0	6.0	6.0	8.3
3	6.5	6.5	6.0	8.5
4	7.0	8.0	6.0	10.0
5	8.0	9.5	6.0	11.0
6	8.0	11.0	7.5	8.0
7	8.0	12.0	8.0	8.0
8	8.0	12.0	7.0	8.0
9	8.0	10.0	8.0	8.0
10	8.0	10.0	7.0	6.0
11	8.0	9.0	8.0	4.0
12	9.0	8.5	7.0	7.0
13	6.0	6.0	6.0	8.0

The initial state is taken to be $(5, 5, 5, 5)^T$, and the terminal state is constrained to be $(5, 5, 5, 7)^T$. The final decision time N is 12. Furthermore, for each t, $y_{1,t} = 2$, and $y_{2,t} = 3$. The release constraints are determined by the condition that

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \le \begin{bmatrix} p_{1,t} \\ p_{2,t} \\ p_{3,t} \\ p_{4,t} \end{bmatrix} \le \begin{bmatrix} 3 \\ 4 \\ 4 \\ 7 \end{bmatrix} \qquad 1 \le t \le 12 \tag{47}$$

The loss function to be minimized is $J(\mathbf{p}) = \sum_{t=1}^{12} L(x_t, p_t, t)$, where

$$L(x_t, p_t, t) = \sum_{j=1}^{4} c_{j,t} p_{j,t} \tag{48}$$

The loss coefficients $c_{j,t}$ are given in Table 1. This completes the specification of the problem.

We now describe how the terminal constraint that x_{13} be equal to $(5, 5, 5, 7)^T$ can be enforced by translating this condition into constraints on preceding state and control spaces. For example, in view of the first row of the law of motion (45) the condition that $x_{1,13} = 5$ imposes the control constraint

$$p_{1,12} = 5 - x_{1,12} - y_{1,12} = 3 - x_{1,12} \tag{49}$$

Inasmuch as the release level $p_{1,12}$ is bounded above and below by 3 and 0, respectively, in order for the terminal state to be reachable we must insist that

$$3 \le x_{1,12} \le 6 \tag{50}$$

TABLE 3*b*. Maximum Permissible Storage Values for $x_{j,t}$

	Value of j			
t	1	2	3	4
2	12.0	15.0	8.0	15.0
3	12.0	15.0	8.0	15.0
4	10.0	15.0	8.0	15.0
5	9.0	12.0	8.0	15.0
6	8.0	12.0	8.0	15.0
7	8.0	12.0	8.0	15.0
8	9.0	15.0	8.0	15.0
9	10.0	17.0	8.0	15.0
10	10.0	18.0	8.0	15.0
11	12.0	18.0	8.0	15.0
12	12.0	18.0	8.0	15.0

This in turn necessitates bounds on the control $p_{1,11}$, and this prescribes allowable ranges (due to the upper and lower bounds for release) for $x_{1,11}$, etc. Keeping these considerations in mind, one may construct sequences $\{\eta_{1,t}\}$ and $\{\xi_{1,t}\}$ such that if for any t, $\eta_{1,t} \le x_{1,t} \le \xi_{1,t}$, then $x_{1,13} = 5$ is reachable from $x_{1,t}$ (without violation of release level bounds).

For reservoirs downstream from reservoir 1, the task of describing the states from which the terminal state is reachable is complicated by the fact that such states depend on reservoir policy and storage upstream. For example, while release constraints on discharging from reservoir 3 may not be violated, there may not be enough water in the system to meet the demand. However, it is not difficult to construct four-tuple sequences $\{\eta_t\}$ and $\{\xi_t\}$ such that the final state is reachable from state time (x_t, t) if

$$\eta_t \le x_t \le \xi_t \tag{51}$$

We may as well assume that (51) also satisfies the state constraint (46). It is evident that (46) may be rewritten as

$$\eta_t \le x_{t-1} + y_{t-1} + Mp_{t-1} \le \xi_t \tag{52}$$

from which the linear constraints may be rephrased in terms of controls as

$$\eta_{t-1}^c \le p_{t-1} \le \xi_{t-1}^c \tag{53}$$

where η_{t-1}^c and ξ_{t-1}^c are constructed from η_t, ξ_t, and $M^{-1}(x_{t-1} + y_{t-1})$ in an obvious fashion. The linear constraints $a(x, t-1)$ and $b(x, t-1)$ as described in connection with (1) are then formed by insisting that (47) and (53) be simultaneously satisfied.

The one procedural fine point remaining to be discussed is

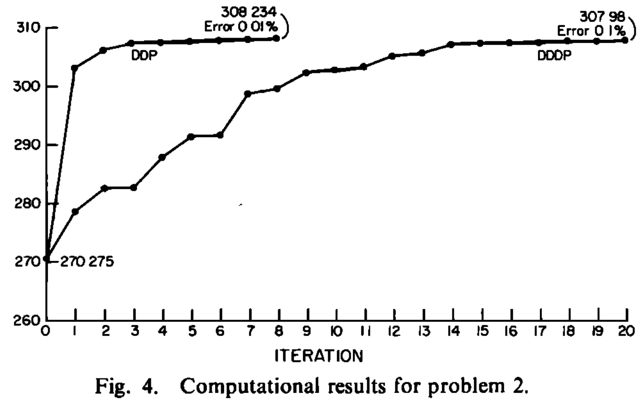

Fig. 4. Computational results for problem 2.

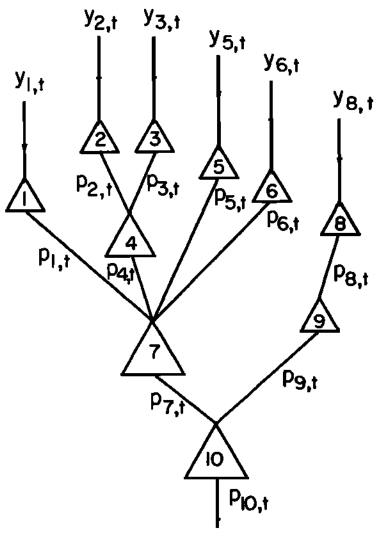

Fig. 5. Ten-reservoir configuration.

how to obtain positive definite quadratic approximations to the linear loss function $L(x, p, t)$ defined by (48). It is sufficient to describe the method we used to obtain an approximation to the function $-p$. We approximated $-p$ by $(\theta/2)(p - \gamma)^2 + C$, where γ is an arbitrary constant larger than the nominal control coordinate $\bar{p}$. We chose $\gamma = 20.0$. The parameters θ and C are determined by the requirement that $(\theta/2)(p - \gamma)^2 + C$ be tangent to the function $-p$ at $\bar{p}$. From this condition it is evident that $\theta = 1/(\gamma - \bar{p})$. Since the addition of constants to the loss functions does not affect the choice of optimal policy, C can summarily be set to 0. Figure 2 shows the quadratic approximation $Q(p)$ to the function $-p$ when $\gamma = 20.0$ and the quadratic is constrained to be tangent at the graph point (3, -3).

Figure 3 displays the return (negative of loss) after a given number of iterations of the DDP method. The three initial trajectories are available in the work of *Heidari et al.* [1971], and the curves marked DDDP are taken from the above reference.

This example, as well as the ones to follow, was constructed so as to be solvable by linear programming. From the simplex method it is known that the optimal return for this problem is 401.3. In Table 2, we have compared the performance of DDP and DDDP, the computations of the former being carried out on a CDC 6400 computer and, according to *Heidari et al.* [1971], those of the latter being undertaken on an IBM 360/75.

Problem 1 was also solved by a DPSA algorithm by *Larson* [1968]. His computations took 30 s on a B-5500 computer.

It should be pointed out that problem 1 is particularly well

TABLE 5. Parameters for Problem 3

	Values of *i*									
t	1	2	3	4	5	6	7	8	9	10
					Upper Bounds for States: max $x_{i,t}$					
2	12.0	17.0	6.0	19.0	19.1	14.0	30.0	13.16	7.9	30.0
3	12.0	15.0	6.0	18.0	18.1	13.0	30.0	12.23	7.3	30.0
4	10.0	15.0	6.0	17.0	17.1	12.0	30.0	11.37	6.8	30.0
5	9.0	15.0	6.0	16.0	16.1	11.0	30.0	10.20	6.4	30.0
6	8.0	12.0	6.0	15.0	15.2	10.0	30.0	9.6	6.2	30.0
7	8.0	12.0	6.0	14.0	14.1	8.5	30.0	9.0	6.1	30.0
8	9.0	15.0	6.0	14.0	14.2	9.6	30.0	9.6	6.4	30.0
9	10.0	17.0	6.0	15.0	15.3	10.7	30.0	10.2	6.7	30.0
10	10.0	18.0	6.0	16.0	16.1	11.8	30.0	11.58	7.0	30.0
11	12.0	18.0	6.0	17.0	17.2	12.9	30.0	12.96	7.4	30.0
12	12.0	18.0	6.0	18.0	18.3	14.0	30.0	13.18	8.0	30.0
					Loss Coefficients: $C_{i,t}$					
1	−1.1	−1.4	−1.0	−1.1	−1.0	−1.4	−2.6	−1.0	−1.0	−2.7
2	−1.0	−1.1	−1.0	−1.0	−1.1	−1.1	−2.9	−1.1	−1.0	−3.0
3	−1.0	−1.0	−1.2	−1.0	−1.2	−1.0	−3.6	−1.2	−1.2	−2.8
4	−1.2	−1.0	−1.8	−1.2	−1.3	−1.0	−4.4	−1.3	−1.8	−3.2
5	−1.8	−1.2	−2.5	−1.8	−1.4	−1.2	−4.2	−1.4	−2.5	−2.9
6	−2.5	−1.8	−2.2	−2.5	−1.5	−1.8	−4.0	−1.5	−2.2	−3.9
7	−2.2	−2.5	−2.0	−2.2	−1.67	−2.5	−3.8	−1.67	−2.0	−4.0
8	−2.0	−2.2	−1.8	−2.0	−1.56	−2.2	−4.1	−1.56	−1.8	−3.6
9	−1.8	−2.0	−2.2	−1.8	−1.45	−2.0	−3.6	−1.45	−2.2	−3.7
10	−2.2	−1.8	−1.8	−2.2	−1.34	−1.8	−3.1	−1.34	−1.8	−2.8
11	−1.8	−2.2	−1.4	−1.8	−1.25	−2.2	−2.7	−1.25	−1.4	−3.5
12	−1.4	−1.8	−1.1	−1.4	−1.14	−1.8	−2.5	−1.14	−1.1	−2.1
					Inflows: $y_{i,t}$					
1	0.5	0.4	0.8	0.0	1.5	0.32	0.0	0.71	0.0	0.0
2	1.0	0.7	0.8	0.0	2.0	0.81	0.0	0.83	0.0	0.0
3	2.0	2.0	0.8	0.0	2.5	1.53	0.0	1.0	0.0	0.0
4	3.0	2.0	0.8	0.0	2.5	2.16	0.0	1.25	0.0	0.0
5	3.5	4.0	0.8	0.0	3.0	2.31	0.0	1.67	0.0	0.0
6	2.5	3.5	0.8	0.0	3.5	4.32	0.0	2.5	0.0	0.0
7	2.0	3.0	0.8	0.0	3.5	4.81	0.0	2.8	0.0	0.0
8	1.25	2.5	0.8	0.0	3.0	2.24	0.0	1.87	0.0	0.0
9	1.25	1.3	0.8	0.0	2.5	1.63	0.0	1.45	0.0	0.0
10	0.75	1.2	0.8	0.0	2.5	1.91	0.0	1.2	0.0	0.0
11	1.75	1.0	0.8	0.0	2.0	0.8	0.0	0.93	0.0	0.0
12	1.0	0.7	0.8	0.0	1.5	0.46	0.0	0.81	0.0	0.0

TABLE 6. Nominal Trajectories for Problem 3

	Value of *i*									
t	1	2	3	4	5	6	7	8	9	10
					Nominal Trajectory 1					
1	0.5	0.4	0.8	1.2	1.5	0.32	3.52	0.71	0.71	4.23
2	1.0	0.7	0.8	1.5	2.0	0.81	5.31	0.83	0.83	6.14
3	2.0	2.0	0.8	2.8	2.5	1.53	8.83	1.0	1.0	9.83
4	3.0	2.0	0.8	2.8	2.5	2.16	10.46	1.25	1.25	11.71
5	3.5	4.0	0.8	4.8	3.0	2.31	13.61	1.67	1.67	15.28
6	2.5	3.5	0.8	4.3	3.5	3.79	14.09	2.5	2.5	16.59
7	2.0	3.0	0.8	3.8	3.5	3.79	13.09	2.8	2.8	15.89
8	1.25	2.5	0.8	3.3	3.0	3.79	11.34	1.87	1.87	13.21
9	1.25	1.3	0.8	2.1	2.5	1.63	7.48	1.45	1.45	8.93
10	0.75	1.2	0.8	2.0	2.5	1.91	7.16	1.2	1.2	8.36
11	1.75	1.0	0.8	1.8	2.0	0.8	6.35	0.93	0.93	7.28
12	1.0	0.7	0.8	1.5	1.5	0.46	4.46	0.81	0.81	5.27
					Nominal Trajectory 2					
1	3.5	3.0	2.0	6.0	6.0	2.0	17.0	3.0	4.0	18.0
2	3.5	3.0	2.0	6.0	3.0	2.0	17.0	3.0	4.0	18.0
3	2.0	2.0	1.0	6.0	3.0	2.0	15.0	1.5	3.0	18.0
4	2.0	2.0	0.5	2.0	3.0	2.0	15.0	1.0	2.0	18.0
5	3.0	3.0	0.9	2.0	3.0	2.0	9.7	1.0	1.0	18.0
6	2.0	3.0	0.5	2.0	3.0	2.0	8.0	2.5	0.1	15.0
7	1.0	3.0	1.0	1.5	3.0	2.0	8.0	3.0	0.1	8.0
8	1.0	2.0	1.0	1.5	2.0	2.0	8.0	1.6	2.5	6.0
9	1.0	0.1	0.5	1.5	2.0	2.0	6.0	0.1	0.1	3.0
10	1.0	0.8	0.05	1.5	2.0	2.0	1.8	0.1	0.02	0.1
11	0.4	0.1	0.05	1.5	2.0	2.0	0.1	0.12	0.1	0.1
12	0.1	0.3	0.1	0.4	1.0	1.3	0.1	0.1	0.1	0.52

suited to methods such as DDDP and DPSA which require discretization of state space because the solution vector turns out to be small multiples of 1. It is reported by *Heidari et al.* [1971] that when noninteger corridor width was used, in conjunction with initial trajectory 1, 18 DDDP iterations were needed to obtain a return of 399.06. Problem 2 below provides results for a less convenient problem in which DDDP and DPSA will need to adaptively change the corridor width or discretization level, respectively.

Multireservoir Control Problem 2. This problem, introduced by *Chow and Cortes-Rivera* [1974], is essentially the same as problem 1, the difference being that the inflow and constraint parameters are chosen so that the components of the solution vector are no longer small multiples of 1, and consequently, for methods (such as DDDP and DPSA) which use state discretization, adaptive step-size selection will have to be employed. For DDP, however, no state discretization is used, and thus it makes no difference whether the solution components are integer valued. For this problem the cost parameters are those given in Table 1. The inflow parameters are given in Table 3*a*; the maximum storages are given in Table 3*b*; the minimum storage is 1.0 for all $t > 1$. The nominal trajectory for both the DDP and the DDDP calculations is given in Table 4. The minimum permissible storage in any reservoir during times 2 through 12 is 1.0 for each t, and releases p_t are constrained by

$$\begin{bmatrix} 0.005 \\ 0.005 \\ 0.005 \\ 0.005 \end{bmatrix} \le p_t \le \begin{bmatrix} 4.0 \\ 4.5 \\ 4.5 \\ 8.0 \end{bmatrix} \qquad (54)$$

For this problem the initial state and final states are required to be $(6, 6, 6, 8)^T$ and $(6, 6, 6, 8)^T$, respectively.

In Figure 4, the return, as a function of the number of iterations, has been plotted for both the DDP algorithm and for a DDDP computation carried out by *Chow and Cortes-Rivera* [1974]. These authors reported that the linear programming solution is 308.2665. After 8 iterations the DDP return was 308.234, and after 20 iterations the DDDP return was 307.98. The computation time per iteration of the DDP algorithm was 3.21 s on a CDC 6400.

Multireservoir Control Problem 3. The reservoir configuration for this problem is shown in Figure 5. By this example we hope to show that our constrained DDP algorithm can solve large problems without demanding excessive storage or processing time.

For this problem the initial and final states were both required to be $(6, 6, 3, 8, 8, 7, 15, 6, 5, 15)^T$. The values $x_{i,t}$ are bounded from above as shown in Table 5, and the minimum permissible storages are $(1, 1, 0.3, 1, 1, 1, 1, 1, 0.5, 1)^T$ for each decision time 2 through 12. For each time the controls are constrained to be between $(0.005, 0.005, 0.005, 0.005, 0.006, 0.006, 0.01, 0.008, 0.008, 0.01)^T$ and $(4, 4.5, 2.12, 7, 6.43, 4.21, 17.1, 3.1, 4.2, 18.9)^T$. Table 5 also gives the inflows and loss coefficients, thus completing the problem specification. In Table 6 we give the initial nominal controls for the two DDP runs. The first initial policy trajectory was the static release policy: the release was adjusted so that the storage at the first iteration did not change. That is, with reference to (44), p_t is adjusted so that

$$x_{t+1} = T(x_t, p_t, t) = x_t + y_t + Mp_t = x_t$$

This condition determines the nominal policy to be $\bar{p}_t = -M^{-1}y_t$, $1 \le t \le 12$. For the second run the initial policy was a low-reservoir policy. The release at each reservoir was as large as the constraints permitted. The returns are shown in Figure 6. Execution time for each iteration of the solution was about 33 s on the CDC 6400. However, the total computation time (18 iterations) from the static release initial trajectory was only 53 CPU seconds on our recently installed CDC Cyber 175, and the 14 DDP iterations required to achieve the solution from

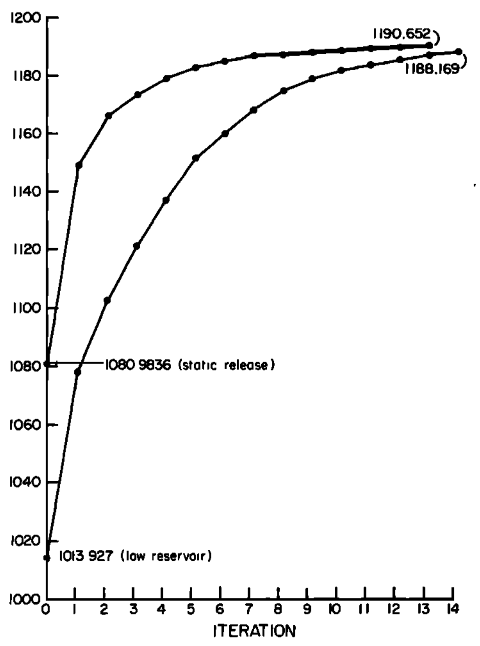

Fig. 6. Computational results for problem 3.

the low-reservoir initial trajectory required only 41.5 CPU seconds on the Cyber.

6. DISCUSSION

For theoretical reasons which we have touched on here, the differential dynamic programming method seems to have the most attractive properties of any of the available numerical methods for multivariate dynamic programming problems.

From a practical point of view, clearly the most interesting aspects of the DDP method are the following: (1) The memory and computational requirements grow only as n^2 and n^3, respectively (instead of exponentially in n); (2) discretization of state and control spaces need not be undertaken; and (3) no state variable decoupling (which characterizes the DPSA technique) is done, and thus coupling information is available to increase convergence rate, and the state transition function need not be invertible. It is to be admitted (in following examples established in the hydrology literature) that the structure of the problem studied here, with its linear loss function (48) and its linear law of motion (44) is particularly convenient. On the other hand, our experience is that if the requisite derivatives of the loss function and law of motion are easily obtained, there is no essential difficulty in solving more complicated discrete-time control problems. In particular, in work to be reported elsewhere, under our guidance, Brian Rutherford has solved quite complicated unconstrained dynamic programming problems having up to 40 state control variables.

It is suspected that problems even larger than our 10-reservoir example could be solved within reasonable budget limitations. For example, the storage capacity needed to run the 4-reservoir problems was 22,500 words, and that for the 10-reservoir problem was 27,100 words. The capacity on the CDC 6400 (a computer that has since been replaced by the larger, faster Cyber 175) was 45,000 words.

Acknowledgment. This work was supported, in part, by NSF grants ENG 76-20280, 'Uses of Modern Systems Theory in Water Resources,' and INT 78-12184.

REFERENCES

Adby, P., and M. Dempster, *Introduction to Optimization Methods,* John Wiley, New York, 1974.

Becker, L., and W. W.-G. Yeh, Optimization of real time operations of multiple reservoir system, *Water Resour. Res., 10*(6), 1107-1112, 1974.

Becker, L., W. W.-G. Yeh, D. Fults, and D. Sparks, Operations models for Central Valley project, *J. Water Resour. Plann. Manage. Div. Amer. Soc. Civil Eng., 102*(WR1), 101-115, 1976.

Bellman, R., *Dynamic Programming,* Princeton University Press, Princeton, N.J., 1957.

Bellman, R., *Introduction to the Mathematical Theory of Control Processes: Linear Equations and Quadratic Criteria,* Academic, New York, 1967.

Bellman, R., *Introduction to the Mathematical Theory of Control Processes: Nonlinear Processes,* Academic, New York, 1971.

Bellman, R., and S. Dreyfus, *Applied Dynamic Programming,* Princeton University Press, Princeton, N.J., 1962.

Chow, V. T., and G. Cortes-Rivera, Applications of DDDP in water resources planning, *Res. Rep. 78,* Univ. of Ill. Water Resour. Center, Urbana, 1974.

Chow, V. T., D. R. Maidment, and G. W. Tauxe, Computer time and memory requirements for DP and DDDP in water resource systems analysis, *Water Resour. Res., 11*(5), 621-628, 1975.

Fletcher, R., A general quadratic programming algorithm, *J. Inst. Math. Its Appl., 7,* 76-91, 1971.

Gill, P., and W. Murray, Newton-type methods for unconstrained and linearly constrained optimization, *Math. Programming, 7,* 311-350, 1974.

Heidari, M., V. T. Chow, P. V. Kokotovic, and D. Meredith, Discrete differential dynamic programing approach to water resources systems optimization, *Water Resour. Res., 7,* 273-282, 1971.

Jacobson, D., and D. Mayne, *Differential Dynamic Programming,* Elsevier, New York, 1970.

Jarmark, B., Convergence control in differential dynamic programming applied to air-to-air combat, *AIAA J., 14,* 118-121, 1976.

Larson, R., *State Increment Dynamic Programming,* Elsevier, New York, 1968.

Larson, R., and A. Korsak, A dynamic programming successive approximations technique with convergence proofs, *Automatica, 6,* 245-252, 1970.

Luenberger, D., *Introduction to Linear and Nonlinear Programming,* Addison-Wesley, New York, 1973.

Mayne, D., A second-order gradient method for determining optimal trajectories of nonlinear discrete-time systems, *Int. J. Contr., 3,* 85-95, 1966.

Mays, L., and B. Yen, Optimal cost design of branched sewer systems, *Water Resour. Res., 11,* 37-47, 1975.

Murray, D., Differential dynamic programming for the efficient solution of optimal control problems, Ph.D. thesis, Dep. of Math., Univ. of Ariz., Tucson, 1978.

Nopmongcol, P., and A. Askew, Multilevel incremental dynamic programing, *Water Resour. Res., 12*(6), 1291-1297, 1976.

Roefs, T., and L. Bodin, Multireservoir operation studies, *Water Resour. Res., 6*(2), 410-420, 1970.

Schweig, Z., and A. Cole, Optimal control of linked reservoirs, *Water Resour. Res., 4*(3), 479-498, 1968.

Sigvaldason, O., A simulation model for operating a multipurpose multireservoir system, *Water Resour. Res., 12*(2), 263-278, 1976.

Szidarovszky, F., and S. Yakowitz, *Principles and Procedures of Numerical Analysis,* Plenum, New York, 1978.

Takeuchi, K., and D. Moreau, Optimal control of multiunit interbasin water resource systems, *Water Resour. Res., 10*(3), 407-414, 1974.

Tennessee Valley Authority, Development of a comprehensive water resource management program, report for a conference of the TVA experience, Int. Inst. for Appl. Syst. Anal., Schloss Laxenburg, Austria, 1974.

Tennessee Valley Authority, Development of water resource management methods for the TVA reservoir system, project status, June 1976, A report for a short course, Div. of Water Manage., Water Syst. Develop. Br., Knoxville, Tenn., 1976.

Trott, W., and W. Yeh, Optimization of multiple reservoir system, *J. Hydraul. Div. Amer. Soc. Civil Eng., 99*(HY10), 1865-1884, 1973.

Wolfe, P., The composite simplex algorithm, *SIAM Rev., 7,* 42-54, 1965.

Yakowitz, S., *Mathematics of Adaptive Control Processes,* Elsevier, New York, 1969.

(Received June 1, 1978;
revised February 27, 1979;
accepted February 28, 1979.)