IMPROVED DYNAMIC PROGRAMMING METHODS FOR OPTIMAL CONTROL OF LUMPED-PARAMETER STOCHASTIC SYSTEMS

C. RUSSELL PHILBRICK, Jr.
Astarte Esca Corporation, 11120 NE 33rd Place, Bellevue, Washington 98004, russ.philbrick@escu.com

PETER K. KITANIDIS
Department of Civil and Environmental Engineering, Stanford University, M24 Fermil Engineering Center, Stanford, California 94305-4220, peterk@stanford.edu

(Received November 1998; revision received August 1999; accepted September 1999)

New dynamic programming methods are developed to solve stochastic control problems with a larger number of state variables than previously possible. These methods apply accurate interpolation to numerical approximation of continuous cost-to-go functions, greatly reducing the number of discrete states that must be evaluated. By efficiently incorporating information on first and second derivatives, the approximation reduces computational effort by several orders of magnitude over traditional methods. Consequently, it is practical to apply dynamic programming to complex stochastic problems with a larger number of state variables than traditionally possible. Results are presented for hypothetical reservoir control problems with up to seven state variables and two random inputs.

Optimization is used to find control policies for complex problems and is particularly valuable for systems characterized by many state variables or subject to significant uncertainty. Examples can be found in the planning and operation of systems for water resources, transportation, inventory, financial planning, and environmental remediation.

However, according to Lamond and Boukhtouta (1995), no method exists that can optimize policies for systems that require the combined representation of

1. detailed spatial characteristics (i.e., numerous state variables),
2. detailed stochastic characteristics including serial and spatial correlation, and
3. nonlinear dynamics, complex objectives, and constraints.

Existing methods must compromise between solution accuracy and the ability to solve problems with numerous state variables. A practical example is the control of large water resource systems with many independent reservoirs and subject to stochastic hydlogic inputs. Yeh (1985) provides an extensive review of numerous optimization methods that have been applied to reservoir control.

Available optimization methods lie between two endpoints that can be classified as "stochastic" and "deterministic." Stochastic methods directly incorporate a model of uncertainty in the optimization process and can accurately identify optimal control policies. However, the computational cost of applying stochastic methods to large systems is prohibitive. In contrast, deterministic methods use forecasts of future inputs as if they were exact and are able to solve large problems. However, deterministic methods guarantee optimal policies only under the restrictive conditions that a system be "certainty equivalent" (Bar-Shalom 1981, Bar-Shalom and Tse 1976, Kitanidis 1983, Lamond and Boukhtouta 1995, Philbrick and Kitanidis 1999) or that there exists some prior knowledge of the solution (Bar-Shalom 1981; Philbrick 1997, Chapter 3).

Efforts to overcome these limitations have followed two distinct approaches. The first approach has been to incorporate some elements of uncertainty in deterministic optimization to produce "nearly optimal" policies using Monte Carlo analysis or the lower moments of probability distributions. These methods include chance constraints, penalty methods, regression, neural networks, and first-order analysis (Georgakakos 1989a, 1989b, Georgakakos and Marks 1987, Kitanidis 1987, Kitanidis and Andricic 1989, Marino and Mahamadi 1983, Marino and Simonovic 1981, Ramon and Candaramouli 1996, Revelle et al. 1969, Saad et al. 1996, Simonovic 1979). The second approach has been to improve the computational efficiency of stochastic methods to expand their applicability to problems of larger scale. These methods include improved stochastic dynamic programming (SDP) methods (Foufoula-Georgiou and Kitanidis 1988, Johnson et al. 1993, Philbrick 1997) and stochastic linear programming methods (Infanger 1991, Pereira and Pinto 1985, Pereira and Pinto 1991).

This work presents new methods that can solve problems of greater size than is possible with previous SDP methods. Such methods are applicable to continuous state-space models without limitation on the mathematical forms used to represent dynamics, objectives, constraints, or probabilities. However, SDP methods are limited by the "curse of dimensionality" (Bellman and Dreyfus 1962) to small-scale
problems that can be described by a limited number of variables. To solve larger stochastic problems, the new methods use more sophisticated multi-dimensional interpolation methods to approximate the "cost-to-go" function. Application of these methods demonstrates a reduction in computational effort by several orders of magnitude.

Kitanidis (1986) first presented the concept for these methods, and subsequent work demonstrated the numerical accuracy (Kitanidis and Foufoula Georgiou 1987) and an implementation by the "gradient dynamic programming" (GDP) method (Foufoula-Georgiou and Kitanidis 1988). Johnson et al. (1993) applied a similar concept using spline-based interpolation methods. GDP and spline methods can be used for the optimal control of stochastic problems with as many as four or five state variables (Johnson et al. 1993).

The methods presented here extend this earlier work by improving the computational efficiency of the GDP algorithm and by extending the approach to include additional information on some second derivatives of the cost-to-go function. The resulting methods are applied to stochastic problems with as many as seven state variables. Although still far short of the 50 or so state variables that can be included in stochastic linear programming problems, the methods presented here can be applied to problems with nonconvex cost-to-go functions and numerous stages. Furthermore, these methods can easily be applied to problems with nonlinear dynamics, objectives, and constraints.

Although these new methods can be applied to any stochastic system represented by a sufficiently simple lumped parameter model, development has been motivated by problems in optimal control of multireservoir systems. These problems result from the need to better use of available water resources, of the increasing complexity and interdependency of systems, of changing priorities for water-resources management, and of the impact of uncertain hydrology. Thus, it is important that operating policies consider a wide range of environmental and social variables. However, existing policies are often prescribed by rule curves based on limited criteria, such as current storage levels, season, and—where snow accumulations are significant—forecasts of future streamflows (Jain et al. 1998; U.S. Army Corps of Engineers 1987, 1997; U.S. Bureau of Reclamation 1992). Because system analysis tools have been limited, it has been difficult to identify integrated, systemwide policies for reservoir control, and existing policies have often been developed independently for each reservoir. However, such policies may leave significant room for improvement.

This paper presents new SDP methods to help address these limitations. Following a brief review of SDP, we present the interpolation methods used to develop the new SDP methods. We conclude with an analysis of the methods using a range of multireservoir stochastic control problems. Philbrick (1997) documents computer codes for these methods, which have been successfully applied to the problem of conjunctive use of surface and ground water (Philbrick and Kitanidis 1998) and to evaluate the impact of uncertain inputs on optimal control of reservoir releases (Philbrick and Kitanidis 1999).

1. CHARACTERISTICS OF AN EFFICIENT METHOD

Efficiency of the new SDP methods is a result of interpolation methods that combine the following features: (1) simplicity, allowing rapid evaluation; (2) smoothness, allowing application of efficient Newton-based search techniques that converge rapidly; and (3) accuracy, allowing use of coarse grids.

Low-order polynomials have significant advantages in the interpolation of a cost-to-go function. Polynomials are simple to program and easy to evaluate. For example, traditional multilinear interpolation uses an $n$-fold product of first-order polynomials. Unfortunately, multilinear interpolation is not accurate on coarse grids and produces nonsmooth approximations with undefined derivatives at boundaries between regions. Consequently, SDP that uses multilinear interpolation requires fine grids and slow—albeit robust—search methods.

Polynomials of somewhat higher order can produce approximations that are more accurate and smooth (Davis 1975), permitting application of efficient quasi-Newton search routines. This is the approach taken by both GDP and spline methods. In the case of spline methods, higher order polynomials are used to incorporate cost-to-go values at discrete locations some distance beyond the region over which interpolation is to be applied. In the case of GDP, higher order Hermite polynomials are used to incorporate gradients or other derivative information, though only at nodes that immediately surround this region. Figure 1 illustrates the higher order accuracy of Hermite interpolation using first derivatives for a simple one-dimensional function, as well as its rapid convergence with finer discretization. Foufoula-Georgiou and Kitanidis (1988) and Johnson et al. (1993) demonstrate that the benefit of higher order accuracy more than compensates for the increased computational effort per node.

1.1. Stochastic Dynamic Programming

The goal is to identify feasible decisions $(u)$ that achieve the best system performance for a given state $(x)$ and averaged over a range of uncertain inputs $(w)$. System dynamics depend on the state at the beginning of each state and on decisions and inputs applied during the stages

$$x_{t+1} = T_t(x, u, w).$$

For reservoir control, constraints on feasible decisions include requirements that releases cannot exceed available water or cause storage to exceed total storage capacity.

Performance is measured by total cost over the operating horizon. If the cost for each period is defined by a function $C_t(x, u, w)$, then the accumulated cost of operations is

$$V = \sum_{t=1}^{N} (1 - r)^t C_t,$$

for an $N$-period horizon using a discount rate $r$.  

Figure 1. Illustrated accuracy of linear interpolation and first-order Hermite interpolation. The true function \( F(x) = x^{-1} \) and its gradient \( G \) are indicated by the thin solid lines. The heavy dashed lines are values estimated by interpolation (\( \hat{F} \) and \( \hat{G} \)). For coarse discretization (nodes at \( x = 1 \) and \( 4 \)), neither interpolation method is accurate. With finer discretization (nodes at \( x = 1, 2, \) and \( 4 \)), Hermite interpolation converges rapidly on the true function.

With uncertain inputs, appropriate control decisions cannot be identified by a predetermined schedule of control decisions \( \{u_1, \ldots, u_N\} \) because the future states \( \{x_1, \ldots, x_N\} \) are not known in advance. Instead, managers make decisions in real time using only the information currently available (i.e., the current state). Thus, operations must be identified by policies \( U_1(x), \ldots, U_N(x) \) that identify decisions for all possible scenarios. Thus, the optimization problem is to minimize

\[
F_0(x) = \min_{u_1, \ldots, u_N} \left\{ \mathbb{E} \left[ \sum_{i=1}^{N} \left( C_i + F_i \right) \right] \right\},
\]

subject to system constraints. The expected cost-to-go functions, \( F_0 \) and \( F_N \), define the relative preference for different initial and final states of the system. The operator \( \mathbb{E}\{\cdot\} \) evaluates the expected value over the range of possible scenarios (e.g., of inflow).

However, the number of possible scenarios grows exponentially with the number of stages, and only the simplest problems with few stages can be solved by direct minimization as in Equation (1). Using the principle of optimality (Bellman 1957), dynamic programming divides the master problem into a series of smaller problems:

\[
F_{n-1}(x) = \min_{u_i} \left\{ \mathbb{E}_n \left[ C_i + F_i \right] \right\}.
\]

Each of these smaller problems identifies a policy that minimized the total of current costs \( C \) and the cost-to-go function \( F \). The computational effort to solve the master problem grows only linearly with the number of stages. A crucial step is to develop an accurate functional representation of future cost-to-go functions for each stage and state of the master problem.

Because the form of the cost-to-go function is generally not known in advance, a numerical approximation of the function is developed. This approximation relies on cost-to-go values evaluated at numerous discrete states and on interpolation to compute values at interposed states. The usual approach identifies discrete states using a grid to divide the state space, and interposed states are computed using multilinear interpolation. The application of such a numerical approximation to dynamic programming is called discrete dynamic programming (DDP). From this point on, when we refer to SDP we will be concerned with numerical implementations of the discretized problem using appropriate interpolation schemes.

The computational effort required to solve a DDP problem grows exponentially with the number of state variables (and stochastic variables when solving a SDP problem), an unfortunate feature known as the “curse of dimensionality” (Bellman and Dreyfus 1962). The total effort can be described by the equation

\[
J = N \Lambda^n K^m N_i Z_i,
\]

where

- \( N \) = number of stages,
- \( \Lambda^n \) = number of nodes (discrete values of state variables),
- \( K^m \) = number of scenarios (discrete values of stochastic variables),
- \( N_i \) = number of interpolations for each node and scenario,
- \( Z_i \) = computational effort for each interpolation,

where \( n \) is the number of state variables (i.e., the dimension of the problem) and \( \Lambda \) is the geometric-mean number of discrete values used to represent each state variable. Likewise, \( m \) and \( K \) represent the number and discretization of stochastic variables. \( N_i \) depends on the efficiency of the search routine employed to find optimal decisions for a given state and scenario.

SDP has been restricted to system models that include no more that two or three state variables (Johnson et al. 1993, Yakowitz 1982). To apply SDP to problems that require
additional state variables, various approximate methods have been developed (e.g., aggregation and decomposition) as summarized by Turgeon (1980, 1981), Saad and Turgeon (1988), and Johnson et al. (1991). In well-understood problems, numerical approximation of the cost to go may be avoided by assuming a predefined form for cost-to-go functions and using regression (Bellman and Dreyfus 1962, Bertsekas and Tsitsiklis 1996, Gal 1979). These and other methods allow the application of DDP to large-scale problems, but the simplifications and the use of anticipative decision rules mean that resulting solutions may be inaccurate (Lamond and Boukhtouta 1995).

1.2. Discretization of the Domain

To represent the cost-to-go function numerically, nodes \( \{x^i, i = 1, \ldots, N^i\} \) span the domain. Consistent with traditional DDP methods, these nodes define a rectangular grid. Associated with each node is a value of the cost to go, \( F^i = F(x^i) \), and a gradient, \( G^i = [G^i_1, \ldots, G^i_n]^T \), defined by derivatives \( G^i_k = dF/dx_k \). Values of the cost to go at intermediate points are evaluated by interpolation.

The nodes divide the domain into regions that can be described as hypercubes in \( n \) dimensions. Each hypercube is bound by \( 2^n \) nodes \( \{x^\gamma, \gamma = \gamma_1, \ldots, \gamma_n\} \) located at corners. Using the “lowest” corner node \( x^{n^i} \) as reference and \( \Delta x^\gamma = |\Delta x_1, \ldots, \Delta x_n| \) to represent the size of the hypercubes, the corner nodes are

\[
\begin{align*}
x^{n_1} &= [x^{n_1}_1, x^{n_1}_2, \ldots, x^{n_1}_n]^T, \\
x^{n_2} &= [x^{n_1}_1 + \Delta x_1, x^{n_1}_2, \ldots, x^{n_1}_n]^T, \\
x^{n_3} &= [x^{n_1}_1, x^{n_1}_2 + \Delta x_2, x^{n_1}_3, \ldots, x^{n_1}_n]^T, \\
x^{n_4} &= [x^{n_1}_1 + \Delta x_1, x^{n_1}_2, x^{n_1}_3 + \Delta x_3, \ldots, x^{n_1}_n]^T, \\
&\vdots \\
x^{n_{2^n}} &= [x^{n_1}_1 + \Delta x_1, x^{n_1}_2 + \Delta x_2, \ldots, x^{n_1}_n + \Delta x_n]^T.
\end{align*}
\]

When evaluating the cost to go for any state \( x \) in the domain, the appropriate hypercube used in interpolation is identified by “highest” node \( x^{\gamma_i} \) that has \( x^{\gamma_i}_j \leq x_j \) for all dimensions \( j \).

1.3. Notation for Interpolation

Nodes that immediately surround a state for which the cost to go is desired identify a hypercube. For the methods presented here, interpolation uses only information located at those nodes. When both the cost to go and its gradient are used in interpolation, intermediate values are evaluated by the weighted sum

\[
\hat{F}(x) = \sum_{\gamma = \gamma_1}^{\gamma_n} \left( \Phi^\gamma F^\gamma + \sum_{k=1}^{n} \Psi^\gamma_k G^\gamma_k \right) \tag{4a}
\]

where values \( F^\gamma \) and \( G^\gamma \) are the cost to go and gradient at each corner node \( \gamma \) of the current hypercube. The weights \( \Phi^\gamma \) and \( \Psi^\gamma_k \) are appropriate functions of \( x \). Gradients are evaluated by

\[
\hat{G}_i(x) = \frac{\partial \hat{F}}{\partial x_i} = \sum_{\gamma = \gamma_1}^{\gamma_n} \left( \frac{\partial \Phi^\gamma}{\partial x_i} F^\gamma + \sum_{k=1}^{n} \frac{\partial \Psi^\gamma_k}{\partial x_i} G^\gamma_k \right) \tag{4b}
\]

Careful selection of weighting functions can produce approximations of the cost-to-go function that are accurate, continuous, and smooth over the entire domain (as discussed in the next section).

To develop weighting functions, it is convenient to define a vector \( \eta^\gamma \) that represents the normalized distance of a state \( x \) from a corner node \( x^{\eta_i} \). The normalized distance is

\[
\eta^\gamma = \left[ \frac{|x_1 - x^{n_1}_1|}{\Delta x_1}, \frac{|x_2 - x^{n_2}_2|}{\Delta x_2}, \frac{|x_3 - x^{n_3}_3|}{\Delta x_3}, \ldots, \frac{|x_n - x^{n_n}_n|}{\Delta x_n} \right]^T.
\]

For example, the normalized distance of each corner node from the lowest node \( x^{n^i} \) is

\[
x^{n^i} : \eta^{n^i} = [0, 0, 0, \ldots, 0]^T, \\
x^{n^2} : \eta^{n^2} = [1, 0, 0, \ldots, 0]^T, \\
x^{n^3} : \eta^{n^3} = [0, 1, 0, \ldots, 0]^T, \\
x^{n^4} : \eta^{n^4} = [1, 1, 0, \ldots, 0]^T, \\
\vdots \\
x^{n^{2^n}} : \eta^{n^{2^n}} = [1, 1, 1, \ldots, 1]^T.
\]

This definition produces the useful result that \( \eta^\gamma = \mathbf{0} \) when \( x = x^{\eta_i} \). In addition, when \( x \) is at any other corner node (i.e., \( x = x^i, i \neq \gamma \)), then \( \eta^\gamma = 1 \) for at least one dimension \( j \). We can use \( \eta \) to define general interpolation functions that apply to all nodes.

2. DEVELOPMENT OF INTERPOLATION METHODS

The new methods use local approximations that produce continuous and smooth cost-to-go values over the entire state space, particularly along the boundaries between hypercubes. To accomplish this, the weighting functions must satisfy a number of requirements.

2.1. Requirements to Preserve Values at Nodes

In terms of normalized distance, \( \eta \), the weighting functions are \( \Phi(\eta) \) and \( \Psi_k(\eta) \). The first derivatives are

\[
\begin{align*}
\frac{d\Phi}{dx_k} &= \left( \frac{d\Phi}{d\eta_k} \right) \left( \frac{d\eta_k}{dx_k} \right), \\
\frac{d\Psi_k}{dx_k} &= \left( \frac{d\Psi_k}{d\eta_k} \right) \left( \frac{d\eta_k}{dx_k} \right)
\end{align*}
\]

When applied to Equations (4a) and (4b), the appropriate weight or its derivative must equal 1 when \( \eta^\gamma = \mathbf{0} \) for some \( \gamma \), and all other weights must equal zero. At all other nodes in the hypercube (identified by \( \eta^\gamma = 1 \) for at least one dimension \( j \)), all weights must equal zero.
2.2. Requirements to Ensure Continuity and Smoothness

If weighting functions are highly differentiable, they produce continuous and smooth approximations within the hypercube. However, this does not guarantee continuity and smoothness at the boundary between adjacent hypercubes. Each interposed state \( x \) that lies along a boundary is shared, and any one of the adjacent hypercubes can be used to independently evaluate \( \tilde{F}(x) \) and \( \tilde{G}(x) \). Continuity and smoothness exist only if interpolation over each hypercube yields the same values.

This can occur only if the functional forms of the weights converge at the boundary. One consequence is that weighting functions along a boundary can only use node values that lie on that boundary (i.e., interpolation must depend only on corner nodes that are shared), and weights applied to nodes not on that boundary must vanish. A second consequence is that weighting functions along a boundary can use only geometric information (such as hypercube dimensions) that is shared by the adjacent hypercubes. It is not possible to develop a general interpolation method that uses a mix of different weighting functions, such as those developed by Foufoula-Georgiou (1991) to produce strictly convex interpolations.

Table 1 contains requirements for continuity and smoothness when using a uniform rectangular grid (i.e., a grid with constant discretization intervals producing hypercubes of constant size). These requirements are not satisfied by the polynomial weighting functions used by GDP (Foufoula-Georgiou and Kitaniid 1988) because derivatives of the weights do not vanish for some \( \eta_j = 1 \) (Philbrick 1997). Thus, the resulting cost-to-go approximations are not smooth, and Newton-based search methods must be restarted whenever hypercube boundaries are crossed (Johnson et al. 1993).

2.3. Additional Requirements When Using a Nonuniform Rectangular Grid

If a nonuniform grid is used, then the requirements of Table 1 are insufficient to ensure smoothness. Because the geometry of one or more dimensions is not shared along a boundary (i.e., those dimensions orthogonal to the boundary), different \( \Delta x_j \) will produce an inconsistent evaluation of \( d\eta_j/dx_j \).

Table 2 contains requirements for a nonuniform grid. Additional constraints are applied to all weighting functions except \( \Phi(\eta) \). Consistent weights are achieved by requiring that weights equal zero when \( \eta_j = 0 \) for each appropriate orthogonal dimension \( j \). Note that the point \( \eta = 0 \) is included with any requirement for \( \eta_j = 0 \).

3. THE FIRST-ORDER SDP METHOD

We present new methods that use weighting functions composed of low-order polynomials that satisfy the requirements of Table 2. While numerous other functional forms were evaluated (including higher order polynomials, trigonometric and exponential functions), none was found to be as efficient or accurate.

As applied here, the “order” of a method refers to the cost-to-go derivatives used in interpolation. A “first-order” method uses up to first derivatives (i.e., the gradient). A “second-order” method uses up to second derivatives (i.e., values of the Hessian). The first-order method is a modification of the GDP method developed by Foufoula-Georgiou and Kitaniid (1988).

Because an \( n \)-dimensional hypercube contains \( 2^n(1+n) \) node values (including first derivatives), the interpolation must satisfy at least \( 2^n(1+n) \) constraints to preserve node values and satisfy requirements of Table 2. In the simplest case of \( n = 1 \), the interpolation must satisfy four constraints. A third-order polynomial is the lowest order

### Table 1

<table>
<thead>
<tr>
<th>Value of Weighting Function</th>
<th>Value of First Derivative in Dimension ( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Phi(\eta) )</td>
<td>1 where ( \eta = 0 )</td>
</tr>
<tr>
<td></td>
<td>0 where ( \eta_j = 1, j = 1, \ldots, n )</td>
</tr>
<tr>
<td>( \Psi_r(\eta) )</td>
<td>0 where ( \eta = 0 )</td>
</tr>
<tr>
<td></td>
<td>0 where ( \eta_j = 1, j = 1, \ldots, n )</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Value of Weighting Function</th>
<th>Value of First Derivative in Dimension ( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Phi(\eta) )</td>
<td>1 where ( \eta = 0 )</td>
</tr>
<tr>
<td></td>
<td>0 where ( \eta_j = 1, j = 1, \ldots, n )</td>
</tr>
<tr>
<td>( \Psi_r(\eta) )</td>
<td>0 where ( \eta = 0 )</td>
</tr>
<tr>
<td></td>
<td>0 where ( \eta_j = 1, j = 1, \ldots, n )</td>
</tr>
</tbody>
</table>

Note: The tables provide requirements for ensuring continuity and smoothness in both uniform and nonuniform rectangular grids. The addition of constraints for nonuniform grids highlights the necessity for a more nuanced approach in higher-dimensional spaces.
polynomial that can satisfy these constraints. When \( n = 1 \), the weighting functions can be identified as

\[
\Phi(\eta) = (1 + 2\eta)(1 - \eta)^2, \quad \Psi(\eta) = \eta(1 - \eta)^2 \left( \frac{d\eta}{dx} \right)^{-1}.
\]

(5a)

(5b)

When \( n > 1 \), then the requirements of Table 2 can be satisfied by an \( n \)-tuple product of polynomials. Let

\[
\alpha_j = (1 + 2\eta_j)(1 - \eta_j)^2,
\]

(6a)

\[
\beta_j = \eta_j(1 - \eta_j)^2 \left( \frac{d\eta_j}{dx_j} \right)^{-1},
\]

(6b)

\[
\omega_j = \frac{d\alpha_j}{dx_j} = -6\eta_j(1 - \eta_j) \frac{d\eta_j}{dx_j},
\]

(6c)

\[
\delta_j = \frac{d\beta_j}{dx_j} = (1 - 3\eta_j)(1 - \eta_j).
\]

(6d)

Note that the polynomials \( \alpha_j \) and \( \beta_j \) are the same as defined in Equations (5a) and (5b). Also,

\[
P = \prod_{j=1}^{n} \alpha_j,
\]

(7a)

\[
P_k = \prod_{j=1, j \neq k}^{n} \alpha_j.
\]

(7b)

With this notation, weighting functions for the first-order SDP method can be defined as

\[
\Phi(\eta) = P, \quad \Psi_k(\eta) = \beta_k P_k.
\]

(8a)

(8b)

Notice that all polynomials vanish when \( \eta = 1 \). Also, \( \beta \) and \( \omega \) vanish when \( \eta = 0 \) because they depend on the discretization interval \( \Delta x \).

The resulting polynomial functions have an order of \( 3n \), far less than expected when considering the \( 2^n(1+n) \) node values. However, the standardized distance \( \eta^7 \) is different for each node \( x^7 \), and the weighting functions applied to different corner nodes are independent. It is possible to propose even lower order polynomials, but they do not appear to be more accurate or offer a computational advantage. For example, Foufoula-Georgiou and Kitinidhis (1988) combine polynomials of order \( n+2 \) to produce weighting functions, but these functions require more computational effort to evaluate, and they fail to satisfy the requirements of Table 2.

The first derivatives of Equations (8a) and (8b) are

\[
\frac{d\Phi}{dx} = \omega_j P_j,
\]

(9a)

\[
\frac{d\Psi_k}{dx} = \delta_j P_j \quad \text{when } r = k,
\]

(9b)

\[
\frac{d\Psi_k}{dx} = \omega_j \beta_k P_{j,k} \quad \text{when } r \neq k
\]

(9c)

These weighting functions satisfy the requirements of Table 2 and produce the same cost-to-go values and gradients between adjacent hypercubes (i.e., the interpolation is continuous and smooth over the entire domain when used in Equations (4a) and (4b). Philbrick (1997) provides additional discussion and illustrations of the first-order SDP method.

4. THE SECOND-ORDER SDP METHOD

New SDP methods of an arbitrarily high order can be developed, but at the burden of increased computational effort. The benefit of improved accuracy must be balanced with the increased computational effort required to calculate higher order terms and to include them in an interpolation. A second-order method uses second derivatives

\[
H_{j,k} = \frac{d^2 F}{dx_j dx_k}.
\]

The cost to go and its gradient are evaluated by the weighted sums

\[
\bar{F}(x) = \sum_{\eta^2} \left\{ \Phi\eta F^\eta + \sum_{k=1}^{n} \left[ \Psi_k^\eta G_k^\eta \right] + \sum_{l=1, l \neq k}^{n} \left[ \Gamma_{k,l}^\eta H_{k,l}^\eta \right] \right\},
\]

(10a)

\[
\bar{G}_r(x) = \sum_{\eta^2} \left\{ \frac{d\Phi}{dx_r} F^\eta + \sum_{k=1}^{n} \left[ \frac{d\Psi_k}{dx_r} G_k^\eta \right] + \sum_{l=1, l \neq k}^{n} \left[ \frac{d\Gamma_{k,l}}{dx_r} H_{k,l}^\eta \right] \right\},
\]

(10b)

where weights \( \Phi^\eta(x), \Psi_k^\eta(x), \) and \( \Gamma_{k,l}^\eta(x) \) are appropriate functions of \( x \). Second-order SDP methods are more accurate than first-order methods because they preserve second-derivative information about the cost-to-go function.

If all elements of the Hessian are used, there are \( n^2 \) additional values at each node, or a total of \( 2^n(1+n+n^2) \) for each hypercube (or \( 2^n[1+2n+(n^2-n)/2] \) using symmetry of the Hessian). In the simplest case of \( n = 1 \), six values must be preserved, and the resulting weighting functions are \( n \)-tuple products of fifth-order polynomials. An example is developed by Philbrick (1997, p. 92-97).

We present a second-order method that uses only the off-diagonal elements of the Hessian (i.e., \( d^2 F/dx_j dx_k \), where \( j \neq k \)), and these derivatives are incorporated without increasing the order of polynomials used to define the weighting functions. To produce a continuous and smooth interpolation, the second-order method must satisfy the requirements for the first-order SDP method (Table 2) as well as additional requirements (Table 3) for the weighting function \( \Gamma_{k,l}(\eta) \). These requirements ensure that interpolation along boundaries depends only on properties (i.e., node values and hypercube dimensions) that are shared by the adjacent hypercubes. In addition, constraints of Table 4
are required to preserve the values of off-diagonal second derivatives at nodes.

Weighting functions are n-tuple products of polynomials, using the same definitions provided by Equations (6a)–(6d) and (7a)–(7b). Weighting functions $\Phi(\eta)$ and $\Psi(\eta)$ are identical to those for the first-order method defined by Equations (8a)–(8b) and (9a)–(9c). Weighting function $\Gamma_k, s(\eta)$ is

$$\Gamma_k, s(\eta) = \beta_k \beta_s \bar{P}(k, s),$$

and first derivatives are

$$\frac{d \Gamma_k, s}{dx_r} = \delta_k \delta_s \bar{P}(k, s) \quad \text{when } r = k,$$
$$\frac{d \Gamma_k, s}{dx_r} = \beta_k \delta_s \bar{P}(k, s) \quad \text{when } r = l,$$
$$\frac{d \Gamma_k, s}{dx_r} = \omega_s \beta_k \bar{P}(s, k) \quad \text{when } r \neq \{k, l\}.$$

Second derivatives are estimated by a finite difference approach. Although these estimates are imprecise, it is not necessary to know the values with great accuracy to significantly improve interpolation. Second derivatives at interposed states can also be estimated by the function

$$\hat{G}_{r, s}(x) = \sum_{\eta \in \bar{N}} \delta_{\eta r} \delta_{\eta s} \bar{P}(\eta) + \sum_{l=1}^{n} \frac{d^2 \Psi_l}{dx_r dx_s} G^*_k + \sum_{l=1}^{n} \left\{ \frac{d^2 \Gamma_{k, l}}{dx_r dx_s} H^*_k \right\},$$

However, this equation is not used in the second-order method because it is computationally expensive to evaluate. Nevertheless, inspection of Equation (12) demonstrates that the resulting interpolation is continuous and smooth (Philbrick 1997, pp. 99–100). Appendix A demonstrates that the resulting formulation also produces continuity of these “off-diagonal” second derivatives, though this was not an objective of the method. Continuity of the second derivatives appears to be of limited value in improving the accuracy of the cost-to-go estimates or the efficiency of Newton-based search methods.

5. Analysis of SDP Methods

This section compares results from applications of multilinear, first-order, and second-order SDP methods to a series of increasingly complex multireservoir problems. The SDP solutions identify release policies and the expected future cost of operations as a function of storage levels. Cost of operations is a quadratic penalty for deviations from desired releases, and the optimal solution minimizes the expected cost over three stages.

The following test problems and analysis parallel those of Johnson et al. (1993) used to compare multilinear SDP, GDP, and spline methods. This is accomplished by evaluating the trade-off between solution accuracy and execution time. Solution accuracy is improved by solving each model using progressively finer discretization (i.e., gridding) of the state.

5.1. Description of Test Problems

The reservoir system models are illustrated by Figure 2. The first five models are identical to those used by Johnson et al. (1993). Many other authors have also used the four-reservoir model to test stochastic control methods (Chow et al. 1975, Foufoula-Georgiou and Kitanidis 1988, Heidari et al. 1971, Kitanidis and Andrievevic 1989, Larson 1968, Murray and Yakowitz 1979, Saad et al. 1992, Sobel 1989, Yakowitz 1982). State variables and decision variables represent storage levels and releases from the reservoirs. Stochastic variables represent uncorrelated, normally distributed streamflows into each of the two upstream reservoirs.

Streamflows have a mean and standard deviation $\sim N[2.0, 1.5]$ and $\sim N[4.0, 2.5]$. Using Gaussian quadrature (Press et al. 1992), expected values can be accurately estimated using equally weighted discrete outcomes at
Table 3. Properties of second-order weighting functions required to ensure continuity and smoothness on a nonuniform rectangular grid.

<table>
<thead>
<tr>
<th>Value of Weighting Function</th>
<th>Value of First Derivative in Dimension $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma_{k,i}(\eta)$</td>
<td>0 where $\eta_i = 0$  0 where $\eta_i = 0$, $r \neq k$</td>
</tr>
<tr>
<td>0 where $\eta_j = 0$</td>
<td>0 where $\eta_i = 0$, $r \neq k$</td>
</tr>
<tr>
<td>0 where $\eta_j = 1, j = 1, \ldots, n$</td>
<td>0 where $\eta_i = 0$, $r \neq k$</td>
</tr>
<tr>
<td>0 where $\eta_j = 1, j = 1, \ldots, n$</td>
<td>0 where $\eta_i = 0$, $r \neq k$</td>
</tr>
</tbody>
</table>

Table 4. Properties of weighting functions required to preserve second derivatives at nodes of a nonuniform rectangular grid. Derivatives taken in dimensions $r$ and $s$, $r \neq s$.

\[
\begin{array}{ccc}
\frac{\partial^2 \Phi}{\partial x_r \partial x_s} & \frac{\partial^2 \Psi_r}{\partial x_r \partial x_s} & \frac{\partial^2 \Gamma_{k,i}}{\partial x_r \partial x_s} \\
0 \text{ where } \eta = 0 & 0 \text{ where } \eta = 0 & 1 \text{ where } \eta = 0, \{r, s\} = \{k, l\} \\
0 \text{ where } \eta_j = 1, j = 1, \ldots, n & 0 \text{ where } \eta_j = 1, j = 1, \ldots, n & 0 \text{ where } \eta_j = 0, \{r, s\} \neq \{k, l\} \\
0 \text{ where } \eta_j = 1, j = 1, \ldots, n & 0 \text{ where } \eta_j = 1, j = 1, \ldots, n & 0 \text{ where } \eta_j = 1, j = 1, \ldots, n \\
\end{array}
\]

are required to preserve the values of off-diagonal second derivatives at nodes.

Weighting functions are $n$-tuple products of polynomials, using the same definitions provided by Equations (6a)-(6d) and (7a)-(7b). Weighting functions $\Phi(\eta)$ and $\Psi_r(\eta)$ are identical to those for the first-order method defined by Equations (8a)-(8b) and (9a)-(9c). Weighting function $\Gamma_{k,i}(\eta)$ is

\[
\Gamma_{k,i}(\eta) = \beta_k \beta_i P_{k,i},
\]

and first derivatives are

\[
\frac{d \Gamma_{k,i}}{dx_r} = \delta_i \beta_r P_{k,i} \quad \text{when } r = k,
\]

\[
\frac{d \Gamma_{k,i}}{dx_s} = \beta_i \delta_r P_{k,i} \quad \text{when } r = l,
\]

\[
\frac{d \Gamma_{k,i}}{dx_r} = \omega_i \beta_k \beta_r P_{(k,i)} \quad \text{when } r \neq \{k, l\},
\]

Second derivatives are estimated by a finite difference approach. Although these estimates are imprecise, it is not necessary to know the values with great accuracy to significantly improve interpolation. Second derivatives at interposed states can also be estimated by the function

\[
\tilde{H}_{r,s}(x) = \frac{2^n}{n+1} \left\{ \frac{d^2 \Phi}{dx_r dx_s} f_r^s + \sum_{i=1}^{n} \left\{ \frac{d^2 \Psi_r}{dx_r dx_s} G_r^s \\
+ \sum_{l=1}^{n} \left\{ \frac{d^2 \Gamma_{k,i}}{dx_r dx_s} H_{k,i}^s \right\} \right\} \right\}.
\]

However, this equation is not used in the second-order method because it is computationally expensive to evaluate. Nevertheless, inspection of Equation (12) demonstrates that the resulting interpolation is continuous and smooth (Philbrick 1997, p. 99–100). Appendix A demonstrates that the resulting formulation also produces continuity of these “off-diagonal” second derivatives, though this was not an objective of the method. Continuity of the second derivatives appears to be of limited value in improving the accuracy of the cost-to-go estimates or the efficiency of Newton-based search methods.

5. ANALYSIS OF SDP METHODS

This section compares results from applications of multilinear, first-order, and second-order SDP methods to a series of increasingly complex multireservoir problems. The SDP solutions identify release policies and the expected future cost of operations as a function of storage levels. Cost of operations is a quadratic penalty for deviations from desired releases, and the optimal solution minimizes the expected cost over three stages.

The following test problems and analysis parallel those of Johnson et al. (1993) used to compare multilinear SDP, GDP, and spline methods. This is accomplished by evaluating the trade-off between solution accuracy and execution time. Solution accuracy is improved by solving each model using progressively finer discretization (i.e., gridding) of the state.

5.1. Description of Test Problems

The reservoir system models are illustrated by Figure 2. The first five models are identical to those used by Johnson et al. (1993). Many other authors have also used the four-reservoir model to test stochastic control methods (Chow et al. 1975, Foufoula-Georgiou and Kitamidis 1988, Heidari et al. 1971, Kitamidis and Andrieveic 1989, Larson 1968, Murray and Yakowitz 1979, Saad et al. 1992, Sobel 1989, Yakowitz 1982). State variables and decision variables represent storage levels and releases from the reservoirs. Stochastic variables represent uncorrelated, normally distributed streamflows into each of the two upstream reservoirs.

Streamflows have a mean and standard deviation $\sim N[2.0, 1.5]$ and $\sim N[4.0, 2.5]$. Using Gaussian quadrature (Press et al. 1992), expected values can be accurately estimated using equally weighted discrete outcomes at
Figure 2. Structure of multireservoir systems used to test the stochastic dynamic programming methods.

\[ n = 1 \quad n = 2 \quad n = 3 \quad n = 4 \quad n = 5 \quad n = 6 \quad n = 7 \]

\[ \begin{array}{c}
1 \\
1 \\
1 \\
2 \\
2 \\
3 \\
4 \\
4 \\
4 \\
5 \\
5 \\
6 \\
6 \\
7 \\
7 \\
7 \\
7 \\
\end{array} \]

Combinations of these produce four outcomes for \( w \), and release decisions are optimized for each assuming limited foresight (Philbrick 1997, Chapter 2). In theory, numerical integration using these outcomes and weights will be exact when the integration is a convolution of the probability distribution with a third-order polynomial. While the application of these weights to Equation (2) produces an accurate numerical integration, it is not exact because of the piece-wise nature of the cost-to-go approximation and the impact of constraints.

The transition function for each of three stages is a linear function consistent with Figure 2. For example, when \( n = 4 \), the transition function is

\[
T(x, u, w) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} x + \begin{bmatrix}
-1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 \\
0 & 1 & -1 & 0 \\
1 & 0 & 1 & -1
\end{bmatrix} u + \begin{bmatrix}
1 \\
0 \\
0 \\
0
\end{bmatrix} w.
\]

Releases cannot be negative (\( u \geq 0 \)), and each reservoir has total storage of 12 units (\( 12 \geq x \geq 0 \)).

The cost function in each of three stages is a quadratic function of deviations from desired releases. The desired releases in each stage are \( u = 1 \) for all reservoirs, and the cost of deviations in each of the three stages is

\[
C(u) = \sum_{i=1}^{n} a_i (u_i - 1)^2, \quad n \neq 2,
\]

\[
C(u) = (u_1 + u_2 - 2)^2, \quad n = 2.
\]

The cost to go at the end of the time horizon (i.e., the cost for the terminal state) is

\[
F_3(x) = \sum_{i=1}^{n} (x_i - b_i)^2.
\]

Coefficients \( a \) and \( b \) for each of the seven models are given in Table 5.

Table 5. Parameter definitions for multireservoir systems used to test the stochastic dynamic programming methods.

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n=1 )</td>
<td>( a=1.0 )</td>
</tr>
<tr>
<td>( n=2 )</td>
<td>( \mathbf{a}=[1.0, 1.0]^T )</td>
</tr>
<tr>
<td>( n=3 )</td>
<td>( \mathbf{a}=[1.1, 1.2, 1.3]^T )</td>
</tr>
<tr>
<td>( n=4 )</td>
<td>( \mathbf{a}=[1.1, 1.2, 1.0, 1.3]^T )</td>
</tr>
<tr>
<td>( n=5 )</td>
<td>( \mathbf{a}=[1.1, 1.2, 1.0, 1.3, 1.1]^T )</td>
</tr>
<tr>
<td>( n=6 )</td>
<td>( \mathbf{a}=[1.1, 1.2, 1.0, 1.3, 1.1, 1.0]^T )</td>
</tr>
</tbody>
</table>

with the other six models: Performance is measured by the cumulative output of the two reservoirs rather than by the individual output of each reservoir. As the results will demonstrate, this increases the importance of second derivatives included in the second-order SDP method.

6. COMPUTATIONAL EFFORT OF SDP METHODS

Each SDP method was applied to the reservoir control problems with discretization as fine as \( \Lambda = 17 \). However, not all combinations of problem dimension and discretization were solved because of the large computer time required. Multilinear SDP was applied to problems with only as many as five reservoirs. Larger problems required excessive computer time, even with exceptionally crude discretization. The first-order and second-order SDP methods were applied to test problems with up to seven reservoirs. Computer time and solution error are reported only for the first of three stages. Solutions were developed using an HP-9000 series 755 workstation and related HP workstations.

Because the multilinear SDP method produces discontinuous cost-to-go derivatives at hypercube boundaries, we used different solvers as search engines in the SDP methods. For the multilinear SDP method, we applied the downhill simplex method (Press et al. 1992, p. 402-406) with constraints enforced by penalty methods. For the first-order and second-order SDP methods, we applied the quasi-Newton method of NPSOL (Gill et al. 1986) to take advantage of the smooth cost-to-go approximation.
Table 6. Computational effort (milliseconds) per subproblem for different SDP methods and for the range of multireservoir problems. Each subproblem represents a search for one set of control decisions applied to a specific state and a specific outcome of the stochastic variables.

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Multilinear SDP Method</td>
<td>32</td>
<td>90</td>
<td>145</td>
<td>323</td>
<td>988</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First-Order SDP Method</td>
<td>11</td>
<td>25</td>
<td>24</td>
<td>34</td>
<td>65</td>
<td>94</td>
<td>200</td>
</tr>
<tr>
<td>Second-Order SDP Method</td>
<td>25</td>
<td>31</td>
<td>29</td>
<td>54</td>
<td>128</td>
<td>335</td>
<td>932</td>
</tr>
</tbody>
</table>

6.1. Breakdown of Computational Effort

Equation (3) demonstrates that the total computational effort required to solve a SDP problem depends on the number stages, the number of discrete states (i.e., nodes), and the number of outcomes of the stochastic variables. Together, these identify the number of subproblems in a master problem. In addition, the computational effort of each subproblem depends on the number of interpolations performed per search (i.e., per call of the search engine) and on the computational effort per interpolation. Table 6 summarizes how computational effort per subproblem changes with dimension (n) and with the different SDP methods.

Total computational effort is primarily controlled by the time spent on interpolating the cost-to-go function, and Table 7 identifies the computational effort of each interpolation method. Differences between these values and the values in Table 6 represent overhead (such as setting up the problems and other high-level activities), and the fraction of total effort is relatively larger for the multilinear SDP method because multilinear interpolation is easier to evaluate. Nevertheless, these differences decrease with n and become insignificant with sufficiently large n.

While the computational effort per interpolation depends on the complexity of the interpolation method, the total time spent on interpolation also depends on the solver and the number of interpolations required. Table 7 also identifies the computational effort per interpolation, the number of interpolations per search, and the number of searches per subproblem. The computational effort per interpolation depends only on the interpolation method. The interpolations per search and the number of searches per subproblem both depend on the solver. Note that one interpolation is required for each evaluation of the objective function (i.e., one iteration of the solver).

The minimum number of searches per subproblem is two because the computer code was written to verify the results of each search by at least one restart of the solver. This restart was also used to ensure accurate estimates of the cost-to-go gradient used by the higher order methods. The restart is also important for the downhill simplex method because the simplex can degenerate (Press et al. 1992), as demonstrated by the increasing number of restarts with problem dimension (at a rate of about n1.5). In contrast, the first-order and second-order SDP methods require few additional restarts.

The number of interpolations per search also increases with dimension. The rate of increase is about n1.5 for the slow (but robust) simplex solver and about n0.5 for NPSOL. Furthermore, the solver appears to converge slightly more rapidly when applied with the second-order SDP method than with the first-order SDP method.

The number of interpolations per search also varies with the desired tolerance of the solver. The tolerance for the simplex solver is set higher (a relative error of 10⁻⁶ versus 10⁻¹² for the quasi-Newton solver). A high solver tolerance has a negligible impact on solution accuracy of the multilinear SDP method since errors of less than about 1% are difficult to achieve without very fine state discretiza-

Table 7. Computational effort (milliseconds) required by interpolation per subproblem and per interpolation. (The number of interpolations per subproblem is the product of the number of searches per subproblem and the number of interpolations per search. Computational effort per subproblem is less than in Table 6 because overhead is excluded.)

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Multilinear SDP Method</td>
<td>3</td>
<td>11</td>
<td>17</td>
<td>56</td>
<td>304</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total interpolation time</td>
<td>2.2</td>
<td>2.2</td>
<td>2.3</td>
<td>3.5</td>
<td>5.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td># Searches per subproblem</td>
<td>22.1</td>
<td>73.0</td>
<td>106.1</td>
<td>167.4</td>
<td>272.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td># Interpolations per search</td>
<td>0.06</td>
<td>0.06</td>
<td>0.07</td>
<td>0.10</td>
<td>0.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time per interpolation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>First-Order SDP Method</td>
<td>0.4</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>25</td>
<td>54</td>
<td>142</td>
</tr>
<tr>
<td>Total interpolation time</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td># Searches per subproblem</td>
<td>3.8</td>
<td>7.9</td>
<td>7.0</td>
<td>7.8</td>
<td>8.2</td>
<td>9.2</td>
<td>9.9</td>
</tr>
<tr>
<td># Interpolations per search</td>
<td>0.05</td>
<td>0.14</td>
<td>0.27</td>
<td>0.58</td>
<td>1.55</td>
<td>2.95</td>
<td>7.12</td>
</tr>
<tr>
<td>Time per interpolation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Second-Order SDP Method</td>
<td>0.5</td>
<td>3</td>
<td>6</td>
<td>17</td>
<td>56</td>
<td>189</td>
<td>594</td>
</tr>
<tr>
<td>Total interpolation time</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td># Searches per subproblem</td>
<td>3.8</td>
<td>8.1</td>
<td>6.5</td>
<td>7.5</td>
<td>8.0</td>
<td>8.7</td>
<td>9.5</td>
</tr>
<tr>
<td># Interpolations per search</td>
<td>0.06</td>
<td>0.17</td>
<td>0.43</td>
<td>1.14</td>
<td>3.52</td>
<td>10.84</td>
<td>31.37</td>
</tr>
<tr>
<td>Time per interpolation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
tion (discussed in the next section). Also, a lower solver tolerance does not significantly increase search time of the quasi-Newton method and helps to ensure accurate estimates of cost-to-go gradients.

6.2. Measured Versus Estimated Computer Time

We can compare the measured computational effort of each interpolation with the effort estimated by direct counting of floating-point operations (Johnson et al. 1993). A floating-point operation, or "flop," is a floating point multiplication and an addition.

Table 8 displays the average number of flops required to interpolate the cost-to-go function by each method. Because implementation of quasi-Newton search methods is more efficient if cost-to-go gradients are calculated analytically, the number of flops also includes the effort to interpolate gradients. Because the gradients are calculated only during a restart of the solver, the number of flops per interpolation is an average. When \( n \) is large, the growth in the number of flops is of order \( 3(2^n) \) for multilinear interpolation, \( 3n^2(2^n-1) \) for the first-order interpolation, and \( 3n^3(2^{n-2}) \) for the second-order interpolation.

Table 8 also displays the time per flop calculated by dividing the average time per interpolation (Table 7) by the average number of flops. These results are consistent and demonstrate that direct counting of flops provides a reasonable estimate of computational effort, particularly for higher dimension problems. Direct counting can also be used to identify inefficiencies of computer code or compilers used to develop executable SDP programs.

The computer time per flop has a minimum of about 0.5 \( \mu \)sec (or about 2 million flops per second). This is low for the published benchmark speed of around 60 million flops per second for the HP-9000 series 755 workstations, and indicates that there may be significant improvements possible with better compilers or better structured code. Time per flop also decreases with \( n \), indicating that measured computer time still contains overhead that is significant for small problems.

6.3. Trade-Off Between Interpolation Effort and Search Effort

As demonstrated by the results in Table 8, the time required for each interpolation is smallest for the multilinear method and is significantly larger for the other methods. This is because the multilinear method is the simplest and because there is no interpolation of the gradient.

For all methods, interpolation effort increases with the number of values used in interpolation. This number depends on the number of corner nodes in a hypercube and on whether the interpolation uses derivatives. The number of nodes for all methods is \( 2^n \); and the number of values at each node increases from one for the multilinear method, to \( n+1 \) for the first-order method, and to \( (n^2-n)/2+n+1 \) for the second-order method. Although the increase in interpolation effort is significant for higher order methods, careful evaluation and reuse of values calculated for the simple polynomial Equations (6a) through (6d) can significantly reduce computational effort. This results in a significant computational advantage over the weighting functions of GDP (Foufoula-Georgiou and Kitanidis 1988) that does not permit such reuse. Note also that the number of values used by spline methods is of order \( 4^n \) (Johnson et al. 1993), which is larger than the number of values used by any of these other methods when \( n \) is large.

Although the effort to perform each interpolation is much larger for the new methods than for multilinear interpolation, the effort is more than compensated for by the rapid convergence of the quasi-Newton solver. Consequently, the total interpolation time of both the first-order and second-order methods is significantly smaller than the total interpolation time of the multilinear method.

7. ACCURACY OF SDP METHODS

Finer state discretization rapidly increases the effort required to solve SDP problems. However, in return, solutions become more accurate. This section evaluates the accuracy of solutions introduced in the previous section.

Accuracy of solutions is evaluated by comparing each cost-to-go function approximation with an "exact" solution. Because we do not know the true solution, it is estimated using the most accurate (i.e., finest discretization) second-

---

Table 8. Number of floating point operations (flops) per interpolation and resulting computer time per flop (milliseconds). (Results are provided for each SDP method and for the range of multireservoir problems.)

<table>
<thead>
<tr>
<th>Method</th>
<th>( n = )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multilinear SDP Method</td>
<td>Avg # flops per interpolation</td>
<td>7</td>
<td>14</td>
<td>27</td>
<td>52</td>
<td>101</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Time per flop</td>
<td>0.0082</td>
<td>0.0043</td>
<td>0.0026</td>
<td>0.0019</td>
<td>0.0020</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>First-Order SDP Method</td>
<td>Avg # flops per interpolation</td>
<td>26</td>
<td>84</td>
<td>242</td>
<td>680</td>
<td>1862</td>
<td>4948</td>
<td>12770</td>
</tr>
<tr>
<td></td>
<td>Time per flop</td>
<td>0.0018</td>
<td>0.0017</td>
<td>0.0011</td>
<td>0.0009</td>
<td>0.0008</td>
<td>0.0006</td>
<td>0.0006</td>
</tr>
<tr>
<td>Second-Order SDP Method</td>
<td>Avg # flops per interpolation</td>
<td>23</td>
<td>92</td>
<td>362</td>
<td>1352</td>
<td>4662</td>
<td>14932</td>
<td>45026</td>
</tr>
<tr>
<td></td>
<td>Time per flop</td>
<td>0.0027</td>
<td>0.0018</td>
<td>0.0012</td>
<td>0.0008</td>
<td>0.0008</td>
<td>0.0007</td>
<td>0.0007</td>
</tr>
</tbody>
</table>
order SDP solution. Accuracy is measured by the average absolute relative error (AARE) of the cost to go:

\[ \text{AARE} = \frac{1}{N} \sum_{i=1}^{N} \frac{\hat{F}(x_i) - F(x_i)}{F(x_i)}, \]

where \( \hat{F} \) is the estimate and \( F \) is the "exact" value. AARE was calculated for the set of states defined by the finest grid (\( \Lambda = 17 \)).

7.1. Error Reduction with Finer Discretization

As expected, solutions with finer discretization have lower error. For multilinear SDP, AARE declines from an average of 300% with \( \Lambda = 2 \) to about 1% with \( \Lambda = 17 \). For the first-order and second-order SDP methods, AARE declines dramatically from an average of 30% to less than 0.002%.

The second-order SDP method is several times more accurate than the first-order method for problems with fine discretization. This result is found to hold even when the most accurate first-order SDP solution is used to approximate the exact solution. Note that for the one-reservoir problem, the Hessian has no off-diagonal elements, and the first-order and second-order SDP methods produce the same solutions.

In addition, solutions of the two-reservoir problem are less accurate than solutions of other problems. Because of its unique cost function, the AARE of the two-reservoir problem is higher for all discretizations of the state.

7.2. Error Analysis

Kitanidis and Foufoula-Georgiou (1987) evaluated the convergence of GDP and multilinear SDP solutions as functions of discretization interval \( \Delta x \). Using multilinear SDP, errors of the cost function and control policy decrease in proportion to \( (\Delta x)^2 \) and \( \Delta x \), respectively. Using GDP, solutions converge far more rapidly, decreasing in proportion to \( (\Delta x)^4 \) and \( (\Delta x)^3 \).

The current results are consistent with these observations. Consider each pair of values of \( \Lambda \) in the set \{(2,3), (3,5), (4,7), (5,9), (7,13), (9,17)\}. Each pair represents a halving of the discretization interval. For solutions of each pair, we should expect a factor of 4 reduction in error using multilinear SDP and a factor of 16 reduction in error using the first-order SDP method. Actual reductions in error are in rough agreement with these expectations (Table 9). Results for the second-order SDP method appear to agree with even higher order convergence expected of a second-order interpolation method. Note that error reductions are smaller for the two-reservoir problem, particularly when using the first-order SDP method.

These results have also been verified qualitatively by solving release decisions for the four-reservoir problem given specific initial states (Philbrick 1997, Chapter 6). In all cases, the first-order and second-order SDP solutions are accurate with few discrete values (\( \Lambda = 3 \) or 4). Multilinear SDP solutions do not reach a comparable level of accuracy until \( \Lambda = 13 \) to 17. These results are consistent with results for GDP (Foufoula-Georgiou and Kitanidis 1988).

The relative error is always positive for multilinear SDP and is always negative for the first-order and second-order SDP methods. Figure 1 demonstrates that when a cost-to-go function is convex, multilinear interpolation overestimates and Hermite interpolation underestimates the true values. Hermite polynomials provide an interpolation with lower curvature than the true function. When the curvature changes severely, oscillation of the approximation can produce false local minima and incorrect solutions. Appendix B defines conditions under which nonconvex oscillations develop.

8. NET COMPUTATIONAL EFFICIENCY OF SDP METHODS

Figures 3 through 5 plot the trade-off between accuracy and computational effort (CPU time) for each SDP method and for the range of reservoir control problems. Note that computational effort increases rapidly with dimension, and plotted values are as large as 10³ seconds per stage (approximately one day, which is too large for most applications). The trade-off between accuracy and effort is clearly worse for multilinear SDP than that for either the first-order or second-order SDP method, and the trade-off for the first-order and second-order SDP methods appear comparable.

<table>
<thead>
<tr>
<th>Table 9</th>
<th>Ratio of average absolute relative error with a halving of the discretization interval for multilinear, first-order, and second-order SDP. (Ratios are provided for different levels of discretization and for the range of multireservoir problems. Note that for ( n = 1 ), the first-order and second-order methods produce identical cost-to-go estimates.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Lambda \rightarrow \Delta x )</td>
<td>Multilinear SDP</td>
</tr>
<tr>
<td>( n )</td>
<td>1</td>
</tr>
<tr>
<td>( \Lambda = 2 \rightarrow 3 )</td>
<td>3</td>
</tr>
<tr>
<td>( \Lambda = 3 \rightarrow 5 )</td>
<td>5</td>
</tr>
<tr>
<td>( \Lambda = 4 \rightarrow 7 )</td>
<td>4</td>
</tr>
<tr>
<td>( \Lambda = 5 \rightarrow 9 )</td>
<td>4</td>
</tr>
<tr>
<td>( \Lambda = 7 \rightarrow 13 )</td>
<td>4</td>
</tr>
<tr>
<td>( \Lambda = 9 \rightarrow 17 )</td>
<td>5</td>
</tr>
</tbody>
</table>

* Used as "exact" solution.
Figure 3. Average absolute relative error of the cost-to-go approximation using multilinear SDP. Error decreases with the increasing computational effort of finer discretization, and results are plotted for problems with up to five state variables.

Figure 4. Average absolute relative error of the cost-to-go approximation using the first-order SDP method. Error decreases with the increasing computational effort of finer discretization, and results are plotted for problems with up to seven state variables.

Figure 5. Average absolute relative error of the cost-to-go approximation using the second-order SDP method. Error decreases with the increasing computational effort of finer discretization, and results are plotted for problems with up to seven state variables.

Figure 6. Computational effort required to approximate the cost to go with a 10% and 1% average absolute relative error. For all SDP methods, effort grows exponentially with the number of state variables, but growth is much slower for new high-order methods.

Note that the two-reservoir problem breaks the trend established by the other problems, especially in the case of the first-order SDP method (Figure 4). This appears less true for the second-order method because it includes off-diagonal elements of the Hessian that are significant for the two-reservoir problem. Under some circumstances, the two-reservoir problem is harder to solve than the three-reservoir problem.

Because of their greater accuracy, the first-order and second-order SDP methods can solve larger problems using coarser grids. A solution with an AARE less than 2% can be obtained with a discretization as coarse as $\Lambda = 3$. To solve the same problem using multilinear SDP requires a discretization finer than $\Lambda = 13$. Consequently, it is practical to solve problems with as many as seven state variables—double the number possible with multilinear SDP. Note that for the special case of the two-reservoir problem, AARE is not reduced below 2% until $\Lambda = 4$ using the second-order SDP method and until $\Lambda = 5$ using the first-order SDP method.

Figure 6 illustrates the greater efficiency of the new SDP methods by plotting the amount of computer time required by each method to achieve 1% and 10% AARE. It is observed that the first-order and second-order SDP methods are several orders of magnitude more efficient than multilinear SDP and that the difference in effort increases with the dimension of the state vector, unlike SDP methods based on splines (Johnson et al. 1993).

Although the first-order and second-order SDP methods have comparable performance, their relative efficiency depends on the problem at hand. For the current series of test problems, off-diagonal elements of the Hessian do not appear significant. The exception is the two-reservoir problem where we see some evidence for the value of second derivatives. It seems reasonable to expect that the
second-order Hermite method may perform significantly better than the first-order method in more realistic problems, in spite of the additional effort required.

9. SUMMARY

This work has presented and tested new discrete stochastic dynamic programming methods that offer significant computational advantages over earlier methods. The improvement is achieved by using higher order interpolation methods to develop local cost-to-go approximations that are smooth and continuous. Hermite interpolation using gradients and limited second derivatives permits the development of accurate solutions with coarse state-space grids. This also produces smooth approximations that permit the application of efficient quasi-Newton search routines. The resulting methods are orders of magnitude more efficient than methods that rely on standard multilinear interpolation.

It is not clear from the current results which of the two new methods is most suitable for general application. The short time horizon (i.e., three stages) and the simple cost function (i.e., quadratic and separable, except for the two-reservoir test problem) appear to diminish the value of second derivatives used by the second-order method. In applications that are more realistic, it is likely that the second-order method will perform better, particularly for problems with dynamics or costs that depend on the interaction of state variables. In addition, the second-order method is better at preserving convexity of cost-to-go functions. These expectations are supported by other applications of the first-order and second-order SDP methods (Philbrick and Kitanidis 1998).

Current results permit the application of SDP to dynamic control problems with as many as seven state variables. This is a significant improvement over multilinear SDP that requires fine discretization of the state and is limited to fewer than four state variables. The new methods permit discretizations as coarse as $\Delta = 3$ for simple problems, although application to more complex problems may require finer discretization, especially when cost-to-go functions have large changes in curvature. This is also true when values of the Hessian are significant, as demonstrated by the two-reservoir problem. Hence, it should be practical to solve problems with five or six state variables with the moderately powerful workstations currently available.

An additional advantage is that the structure of algorithms for the new methods is appropriate for parallel computing, and continuing advances in computational capabilities will continue to increase the size of problems for which SDP solutions can be developed.

APPENDIX A: SECOND DERIVATIVES OF THE WEIGHTING FUNCTIONS

Table A.1 summarizes additional properties of the weighting functions required to produce continuity of derivatives used by the second-order method (i.e., off-diagonal elements of the Hessian). The second derivatives of the weighting-function Equations (8a), (8b), and (11a) are

\[
\frac{d^2 \Phi}{dx, dx_i} = \omega_i \omega_j P_{(r, i)}, \quad (A.1)
\]

\[
\frac{d^2 \Psi_k}{dx, dx_i} = \delta_k \omega_i P_{(r, i)} \quad r = k, \quad (A.2a)
\]

\[
\frac{d^2 \Psi_k}{dx, dx_i} = \omega_i \delta_k P_{(r, i)} \quad s = k, \quad (A.2b)
\]

\[
\frac{d^2 \Psi_{(r, s, k)}}{dx, dx_i} = \omega_i \omega_j \beta_{(r, s, k)} P_{(r, s, k)} \quad k \not\in \{r, s\}, \quad (A.2c)
\]

\[
\frac{d^2 \Gamma_{(r, l)}^{(l, r, l)}}{dx, dx_i} = \delta_{(r, l)} \delta_{(r, l)} P_{(l, r, l)} \quad \{r, s\} = \{k, l\}, \quad (A.3a)
\]

\[
\frac{d^2 \Gamma_{(r, l)}^{(l, r, l)}}{dx, dx_i} = \omega_i \beta_{(r, l)} P_{(r, l, l)} \quad r \neq k, \quad s = l, \quad (A.3b)
\]

likewise for $(r \neq l, s = k), (r = k, s \neq l),$ and $(r = l, s \neq k), (A.3c)$

\[
\frac{d^2 \Gamma_{(r, l)}^{(l, r, l)}}{dx, dx_i} = \omega_i \omega_j \beta_{(r, l) P_{(r, l, l)}} \quad \{r, s\} \cap \{k, l\} = 0.
\]

These weighting functions satisfy the requirements of Table 4 and Table A.1.

APPENDIX B: CONVEXITY OF INTERPOLATION

Frequently, cost-to-go functions are convex because high costs (and the highest marginal costs) are associated with extreme states of a system. It is important that interpolation preserve this convexity. A local minimum that is an artifact of the interpolation routine can trap a search method and produce incorrect solutions. Unfortunately, oscillation of the Hermite interpolation methods can produce false local minima, although local minima can also result from multilinear interpolation.
A function is convex only when the Hessian is positive semidefinite (Ecker and Kupferschid 1991), and this is true if and only if all principal minors are nonnegative. For the one-dimensional case, the Hessian is a one by one matrix and the only principal minor is the second derivative

$$\frac{d^2 F}{dx^2} = \frac{6}{\Delta x} \left( \frac{1 - 2 \eta'}{\Delta x} (F^{i+1} - F^i) + \left( \eta' - \frac{2}{3} \right) G^i \right) + \left( \eta' - \frac{1}{3} \right) G^{i+1}. \quad (B1)$$

The principal minor is a linear function of \( \eta' \). Thus, the principal minor is nonnegative over the entire subdomain \( 0 \leq \eta' \leq 1 \) if it is nonnegative at the boundaries \( \eta' = 0 \) and \( \eta' = 1 \). This is true if and only if

$$\frac{2}{3} G^i + \frac{1}{3} G^{i+1} \leq \frac{F^{i+1} - F^i}{\Delta x} \leq \frac{2}{3} G^i + \frac{2}{3} G^{i+1}. \quad (B2)$$

Consequently, an interpolation is convex if each pair of adjacent nodes satisfies these constraints.

Figure 1 illustrated this using two different discretizations of the state variable. Interpolation using the coarser discretization is not convex; the second derivative is negative at the node \( x = 4 \), and the upper bound of Equation (B2) is violated. However, with finer discretization, the constraints are satisfied for each pair of nodes \( \{1,2\} \) and \( \{2,4\} \).

In most cases, we can produce a convex approximation by selecting a sufficiently small discretization interval. Even when the constraints of Equation (B2) cannot be satisfied, an SDP method can tolerate some small concave features because stochasticity results in an averaging process that prevents reinforcement of small concave features.

An alternate approach is to develop interpolations that are strictly convex. As an example, Foutoula-Georgiou (1991) uses exponential functions to produce a strictly convex interpolation, but the method requires the use of two different functions that prevent smooth interpolation over a gridded domain.

For the multidimensional case, the Hessian is more complex and evaluation of the principal minors is more difficult. We might guess that if the one-dimensional convexity constraints are satisfied for each dimension, then the multidimensional interpolation is convex. This seems reasonable since the proposed multidimensional interpolating functional is the \( n \)-fold product of one-dimensional functions. However, while this is roughly true, it is exactly true only for a full second-order SDP method that includes all elements of the Hessian. Problems with nonconvexity (and with poor cost-to-go approximation) appear to be worse when the off-diagonal elements of the Hessian are significant. Under these conditions, the second-order SDP performs better.

Acknowledgments

The authors thank Ben Hobbs of the Johns Hopkins University and the referees for their detailed and helpful comments. Support for this work has been provided by the Graduate Fellowship Program of Associated Western Universities Incorporated and by the Department of Energy. Computer resources used in this work have been provided by National Science Foundation Grant BSC-8957186 and the Hewlett Packard Foundation.

References


Kitanidis, P. K. 1983. Real-time forecasting of river flows and stochastic optimal control of multiservoir systems. IRRI report no 258, ISWRRI completion report no 133, Iowa Institute of Hydraulic Research, The University of Iowa, Iowa City, IA.

—. 1986. Hermite interpolation on an n-dimensional rectangular grid. St. Anthony Falls Hydraulics Laboratory, University of Minnesota, Minneapolis.


—, —, 1999. Limitations of deterministic optimization applied to reservoir operations. J. Water Resources Planning and Management 125(3) 135–142.


