Techniques in Computational Stochastic Dynamic Programming[1]

Floyd B. Hanson[2]

University of Illinois at Chicago
Chicago, Illinois 60607-7045

I. INTRODUCTION

When Bellman introduced dynamic programming in his original monograph [8], computers were not as powerful as current personal computers. Hence, his description of the extreme computational demands as the *Curse of Dimensionality* [9] would not have had the super and massively parallel processors of today in mind. However, massive and super computers can not overcome the *Curse of Dimensionality* alone, but parallel and vector computation can permit the solution of higher dimension than was previously possible and thus permit more realistic dynamic programming applications. Today such large problems are called *Grand and National Challenge* problems [45, 46] in high performance computing. Today's availability of high performance vector supercomputers and massively parallel processors have made it possible to compute optimal policies and values of control systems for much larger dimensions than was possible earlier. Advances in algorithms have also paid a large role.

In this chapter, the focus will be on the stochastic dynamic programming in continuous time, yet related problems and methods will be discussed where appropriate. The primary stochastic noise considered here is Markov noise in continuous time, since this type of noise is separable in time just as the optimization steps in the *principle of optimality*. Thus, the stochastic perturbations treated here will be of the continuous, but non-smooth Gaussian type or the discontinuous, randomly distributed Poisson type. Due to its continuity property, Gaussian noise is suitable for modeling background randomness. In contrast, Poisson noise is suitable for modeling the catastrophic, rare random events. For some stochastic models, random shocks to the system are more important than the continuous perturbations, although the continuous changes are more easy to treat.

Unlike deterministic applications of dynamical programming, the use of general stochastic noise in continuous time makes it difficult to use different formulations other than the partial differential equation of dynamic programming or Bellman equation. For deterministic problems in continuous time, there is the option of applying the maximum principle to formulate a dual set of forward and adjoint backward ordinary differential equations coupled with information on the critical points of the Hamiltonian, so the method of solution is quite different from the dynamic programming approach. Other methods will be discussed later.

Numerical partial differential equation (PDE) methods have been modified for the nonstandard characteristics of the PDE of stochastic dynamic programming. In order to manage the large computational requirements, high performance supercomputers have been employed [17, 116, 117, 118]. For instance, the problems with up to 5 states and 32 mesh points per state have been successfully solved using finite difference methods [116, 117, 58, 118] on both Cray and Connection Machines. Larger problems are possible with recent hardware and software advances.

---

[1] This chapter appeared in F. B. Hanson, "Techniques in Computational Stochastic Dynamic Programming" in *Stochastic Digital Control System Techniques,* within series *Control and Dynamic Systems: Advances in Theory and Applications,* vol. 76, (C. T. Leondes, Editor), Academic Press, New York, NY, pp. 103-162, April 1996.

[2] Supported in part by National Science Foundation Grant DMS 93-0117, National Center for Supercomputing Applications, Pittsburgh Supercomputing Center, and Los Alamos National Laboratory's Advanced Computing Laboratory; written while on sabbatical in Division of Applied Mathematics at Brown University, Providence, RI 02912

The finite element method has computational and memory advantages. This method requires a smaller number of nodes than the corresponding finite difference method of similar accuracy. We have shown [20] that the finite element method not only helps to alleviate Bellman's *Curse of Dimensionality* in dynamical programming computations by permitting the solution of higher dimension problems, but also saving supercomputer storage requirements.

The general aim is to develop fast and efficient parallel computational algorithms and data structures for optimal feedback control of large scale, continuous time, nonlinear, stochastic dynamical systems. Since the finite element procedure requires formulation of the mesh data structures, it is desirable to study the mapping from the problem conceptual structure to the machine configuration for either Cray or Connection Machine computational models [116].

However, the computational treatment of Poisson noise is a particularly unique feature of this chapter. The numerical approach directly treats the partial differential equation of stochastic dynamic programming. Results give the optimal feedback control variables and the expected optimal performance index in terms of state variables and time.

For the stochastic optimal control problem, Monte Carlo and other simulations using random number generation are a primary alternative for direct dynamic programming computations, but disadvantages result from determining the sufficient sample size (complicated for general problems) and there is a question of maintaining feedback control. Furthermore, for simulation calculations, very complicated Markov processes have to be randomly generated and a tremendous number of sample trajectories would have to be averaged, whereas in the stochastic dynamic programming approach the averaging over the stochastic processes is built into the equation of dynamic programming. Hence, there is a great need to develop the use of high performance computing techniques in stochastic dynamic programming for direct solutions of stochastic optimal control problems.

The report of the panel on the *Future Directions in Control Theory* [42] confirms the need for advanced scientific computing, both parallelization and vectorization, in control problems. The *National Computing Initiative* [97] calls stochastic dynamic programming computationally demanding, but avoids the opportunity to classify it as a *Grand Challenge* along with other problems of similar computational demands as it should be classified.

Applications of stochastic dynamic programming arise in many areas, such as aerospace dynamics, financial economics, resource management, robotics and power generation. Another main effort in this area, in addition to our own, has been in France, with Quadrat and his coworkers [1] at INRIA developing an expert system that produces a multitude of results for stochastic differential equations with Gaussian noise, provided that discounting is constant and the problem can be transformed to a stationary one. Dantas de Melo, Calvet and Garcia [13, 26] in France have used the Cray-2 multitasking for discrete time dynamic programming problems.

Kushner and coworkers [75, 76, 77] have recently described many numerical approaches to stochastic control, with special emphasis on the well-developed *Markov chain approximation* method. Also, much theoretical progress has been made for using viscosity solutions [21, 108, 22].

Shoemaker [79, 16, 24, 25] and coworkers have applied several variants of the deterministic differential dynamic programming algorithm groundwater applications. Differential dynamic programming is a modification of dynamic programming based upon quadratic expansions in state and control differentials and was originally developed by Mayne [91].

Luus [87, 88] has developed a method for deterministic, high dimensional dynamic programming problems using grid size reduction in both state and control, or in just control alone, such that the method converges to optimal control and state trajectories as the region reduction iterations proceed.

The author and his co-workers have been developing computational mathematics solutions for fairly general stochastic dynamic programming problems in continuous time using high performance computing techniques [51, 52, 55, 54, 17, 18, 19, 57, 116, 117, 58, 118, 20, 102, 60].

The presentation in this chapter is in the formal manner of classical applied mathematics in order to focus on the methods and their implementation. In Section II computational stochastic dynamic programming is discussed for continuous time problems and advanced techniques are discussed in Section III. In Section IV, the direct stochastic dynamic programming approach is compared in some detail with the algorithm models of differential dynamic programming and the Markov chain approximation. These methods are selected for comparison in some depth since that they are actively used to solve similar type optimal control problems, rather than present a broad survey without much depth. They are reformulated in such a way to facilitate comparison. In Section V, research directions are briefly mentioned.

II. COMPUTATIONAL STOCHASTIC DYNAMIC PROGRAMMING IN CONTINUOUS TIME

The development of fast and efficient computational algorithms is the goal for larger dimension relatively general optimal feedback control of nonlinear dynamical systems perturbed by stochastic diffusion and Poisson jump processes. The diffusion processes represent the continuous, background component of the perturbations, such as that due to fluctuating population death rates, randomly varying winds and other background environmental noise. The Poisson processes represent the discontinuous, rare event processes, such as occasional mass mortalities, large random weather changes or other large environmental effects. The Poisson perturbations model the more disastrous disturbances and these disastrous disturbances are more important for many realistic models than the phenomena modeled by continuous but nonsmooth disturbances resulting from Markov diffusions.

The treatment of Poisson noise is a major feature here. However, there has been much more research on Markov diffusions, and this in undoubtedly due to the fact that they are much easier to analyze than the discontinuous Poisson noise. Random deviations from deterministic results tend to occur in regions of high costs and possible failure, indicating the need for fast algorithms for large fluctuations. Our goal is that our results should be in a practical form suitable for applications.

Our motivation for this research comes from bioeconomic modeling, but the procedures developed are applicable to a wide range of biological, physical, chemical, and engineering applications with a stochastic dynamical system governing the motion or growth of the system, and with a performance or cost function that needs to be optimized. Our applications so far have been primarily the optimal harvesting of fisheries resources. Athans et al. [6] analyze a flight dynamics application perturbed by Gaussian noise, but this application could be treated with the more general random noise described here to model more realistic test conditions. Quadrat and coworkers [1] have made applications to the control of electric power systems. One emphasis here is the use high performance computing techniques on a wider range of applications.

A. FORMULATION OF PDE FOR STOCHASTIC DYNAMIC PROGRAMMING

Due to the fact that the mathematics of stochastic dynamic programming is not very accessible at the level of application, we present here a relatively general formulation. Much of this formulation, but not all, can be gleaned from Gihman and Skorohod [43, 44] with some difficulty, or from Kushner and Dupuis [76], or from the many other accounts restricted to just continuous, Gaussian noise, such as Fleming and Rishel [38], and Stengel [109]. Additional information on stochastic differential equations can be obtained from Arnold [3], Jazwinski [67], and Schuss [106].

The lumped continuous state variable, $\mathbf{X}(t)$, denotes an $n \times 1$ vector of positions, velocities, orientation angles or other important variables. The feedback control variable, $\mathbf{U}(\mathbf{X}(t), t)$, is an $m \times 1$ vector of other regulating dynamic quantities or orientation variables. The basic formal stochastic differential is given by,

$$d\mathbf{X}(t) = \mathbf{F}(\mathbf{X}, \mathbf{U}, t)dt + G(\mathbf{X}, t)d\mathbf{W}(t) + \int_{\mathcal{D}_q} H(\mathbf{X}, Q, t)\mathbf{P}(dt, dQ), \tag{1}$$

for $\mathbf{X}(t_0) = \mathbf{x}_0$; $0 < t_0 < t < t_f$; $\mathbf{X}$ in $\mathcal{D}_x$ and $\mathbf{U}$ in $\mathcal{D}_u$. In (1), $d\mathbf{W}(t)$ is the differential of a standard r-dimensional vector-valued Wiener process, so it has independent Gaussian components, zero mean and Covar$[d\mathbf{W}, d\mathbf{W}^T] = I_r dt$. The term $\mathbf{P}(dt, dQ)$ is a q-dimensional Poisson random measure with independent components, Mean$[\mathbf{P}(dt, dQ)] = \boldsymbol{\psi}(dQ)dt$ and Covar$[\mathbf{P}, \mathbf{P}^T] = \Psi(dQ)dt$, where $\boldsymbol{\psi}(dQ)$ is a q-dimensional distribution of the jump amplitudes indexed by the mark Q in marker domain $\mathcal{D}_q$, $\lambda_i \equiv \int_{\mathcal{D}_q} \psi_i(dQ)$ is the ith rate and $\Psi \equiv [\psi_i \delta_{i,j}]_{q \times q}$ is its diagonal representation. In addition, Covar$[\mathbf{P}, \mathbf{W}^T] = 0$. The coefficients $\mathbf{F}$, G, and H are matrices whose sizes are compatible with the multiplications indicated above in (1).

For the performance criterion or objective functional, we assume the Bolza type,

$$V[\mathbf{X}, \mathbf{U}, \mathbf{P}, \mathbf{W}, t] = \int_t^{t_f} d\tau\, C(\mathbf{X}(\tau), \mathbf{U}(\mathbf{X}(\tau), \tau), \tau) + Z(\mathbf{X}(t_f)), \tag{2}$$

where $C(\mathbf{x}, \mathbf{u}, t)$ is the instantaneous cost function and Z is the terminal or salvage cost function. In (2), the variable time t is taken at the lower limit of the cost integral, rather that t_0, to treat the integral as a variable integral for necessary further analysis, understanding that $t_0 \le t \le t_f$. Obviously, other forms could be used in place of (2) without much difference in effort. Our objective is to optimize the expected performance on the variable time horizon

(t, t_f),

$$v^*(\mathbf{x},t) \quad = \quad \min_u[\underset{\{P,W\}}{\text{Mean}}[V[\mathbf{X},\mathbf{U},\mathbf{P},\mathbf{W},t]|\mathbf{X}(t)=\mathbf{x},\mathbf{U}(t)=\mathbf{u}]], \tag{3}$$

in order to minimize costs of production, costs of extraction, fuel consumption, or lateral perturbations of motion.

Due to the Markov properties of $\mathbf{P}$ and $\mathbf{W}$, the *principle of optimality* holds as it does in the deterministic case, so both minization and conditional expectation operations can be separated into the operations over the current time increment $[t, t+dt)$ and the future time interval $[t+dt, t_f)$:

$$\begin{aligned} v^*(\mathbf{x},t) \quad &= \quad \min_{u[t,t+dt)}\left[\underset{\{P,W\}[t,t+dt)}{\text{Mean}}\left[\int_t^{t+dt} d\tau\ C(\mathbf{X},\mathbf{U},\tau)\right.\right. \\ &+ \quad \left.\left. v^*(\mathbf{X}(t+dt),t+dt)|\mathbf{X}(t)=\mathbf{x},\mathbf{U}(t)=\mathbf{u}\right]\right]. \end{aligned} \tag{4}$$

Next, it is assumed that the formal SDE (1) is interpreted under Itô integrations rules, so an application of the Itô chain rule for Markov processes,

$$\begin{aligned} d\Phi(\mathbf{X}(t),t) \quad &= \quad \left[\frac{\partial\Phi}{\partial t} + \mathbf{F}^T(\mathbf{X},\mathbf{U},t)\cdot\nabla_x\Phi(\mathbf{X},t) + \frac{1}{2}GG^T:\nabla_x\nabla_x^T\Phi\right]\cdot dt \\ &+ \quad \nabla_x^T\Phi(\mathbf{X},t)\cdot G(\mathbf{X},t)d\mathbf{W}(t) \\ &+ \quad \sum_\ell \int_{\mathcal{D}_q}[\Phi(\mathbf{x}+\mathbf{H}_\ell(\mathbf{x},Q,t),t)-\Phi(\mathbf{X}(t),t)]\cdot\mathbf{P}_\ell(dt,dQ), \end{aligned} \tag{5}$$

is required, where $\mathbf{H}_\ell(\mathbf{x},Q,t)$ is the ℓth column vector of the jump amplitude matrix H, the scalar matrix product $A:B=\text{Trace}[AB^T]$ denotes the trace of the matrix product AB^T, and A^T denotes the transpose of matrix A. The generalized Itô chain rule is given in Gihman and Skorohod [43, 44]. See Florentin [39], Dreyfus [33], Wonham [114], and Kushner and Dupuis [76] for combined noise problems, with Poisson in addition to Gaussian noise. These combined processes are also referred to as *jump diffusions* (cf., Kushner and Dupuis [76] and Snyder and and Miller [107] for additional references). Fleming and Rishel [38] give treatments for the control of stochastic systems perturbed by Gaussian white noise. The Itô chain rule is basically generalization of the chain rule of differentiable functions, modified for the non-smoothness of the diffusion processes and the jump discontinuities of the Poisson processes. In fact, this generalized chain rule is probably more about discontinuities in value and derivatives than it is about stochasticity. In contrast to the ordinary chain rule in calculus, the non-smoothness of the Gaussian processes results in a second order Hessian matrix term for Φ, while the jump discontinuities of the Poisson processes result in the jump of Φ at all Poisson process jumps, represented in (5).

Finally, substitution of the chain rule (5) into the principle of optimality (4) result in the optimal expected performance v^* satisfying the Hamilton-Jacobi-Bellman partial differential equation of dynamic programming,

$$\begin{aligned} 0 \quad &= \quad \frac{\partial v^*}{\partial t} + \mathcal{L}_x[v^*](\mathbf{x},t) \\ &\equiv \quad \frac{\partial v^*}{\partial t} + \frac{1}{2}GG^T:\nabla_x\nabla_x^T v^* + S^*(\mathbf{x},t) \\ &+ \quad \sum_\ell \int_{\mathcal{D}_q}\psi_\ell(dQ)[v^*(\mathbf{x}+\mathbf{H}_\ell(\mathbf{x},Q,t),t)-v^*(\mathbf{x},t)], \end{aligned} \tag{6}$$

where $0<t<t_f$ and $\mathbf{x}\in\mathcal{D}_x$. The control switching term in (6) is given by

$$\begin{aligned} S^*(\mathbf{x},t) \quad &= \quad \min_u[S(\mathbf{x},\mathbf{u},t)] \text{ with} \\ S(\mathbf{x},\mathbf{u},t) \quad &\equiv \quad C(\mathbf{x},\mathbf{u},t) + \mathbf{F}^T(\mathbf{x},\mathbf{u},t)\nabla_x v^*(\mathbf{x},t). \end{aligned} \tag{7}$$

One advantage of (6) is that it is a deterministic partial differential equation, in contrast to its origin in the stochastic performance criterion (2) subject to stochastic averaging, minimization and constraints by the stochastic ordinary differential equation (1). The output of a program for (6) in the general nonlinear case is the optimal expected performance, $v^*(\mathbf{x},t)$, and the optimal feedback control, $\mathbf{u}^*(\mathbf{x},t)$, for arbitrary values of $(\mathbf{x},t)$. Knowledge of the control is usually the most important resultant output for applications, because the optimal control is input required by the control user or manager.

The final condition for the optimal, expected performance index is that

$$v^*(\mathbf{x},t_f)=\overline{Z}^*(\mathbf{x},t_f)=\min[\text{Mean}[Z|\mathbf{X}(t_f)=\mathbf{x}]] \tag{8}$$

or salvage costs. The final value problem, rather than initial value problem, property here is due to the fact that (6) is a backward equation with respect to time.

1. Boundary Conditions

The boundary conditions depend more heavily on the precise nature of the stochastic noise and the nature of the boundaries. In many cases, there are no simple boundary specifications, but natural, Dirichlet boundary values sometimes can be obtained by integrating the Bellman equations along the boundaries. It should be noted that, unlike the corresponding forward equation for the optimal trajectory, the Dirichlet boundary conditions are implicitly contained in the backward, Bellman equation, due the conditioning of the optimal expected performance (3) and the inhomogeneous property of the equation due to the instantaneous cost, provided (1) accurately portrays the dynamics and is valid at the boundary. This is due to the fact that the Dirichlet boundary conditions by first principles are calculated from the application of the boundary values to (3) along with (2). Clearly, the boundary version of the Bellman equation will be the same as the interior version of the Bellman equation (6) including (7) with the boundary values applied, except in the most degenerate cases. In other words, the application of the Dirichlet boundary values and the derivation of the Bellman equation by the principle of optimality together with the Itô's rule can be interchanged, again ignoring very exceptional cases.

However, some types of boundary conditions such as Neumann type boundary conditions will require the modification of the SDE (1) to account for the boundary processes as part of the modeling procedure. In this case, proper treatments of boundary conditions are given by Kushner and Dupuis [76]. They construct compensating processes that are added to the unconstrained process and force boundary constraints, such as using a reflecting process in the case of reflecting boundaries. In the case of singular control, free boundaries are another problem that needs consideration [85]. Proper treatment of boundary solutions is of major importance.

2. Nearly Quadratic Costs

The principal advantage of the dynamic programming formulation, (6), is that the optimization step is reduced to minimizing the function argument of the switching term (7) over the control, rather than directly optimizing the objective functional in (2) over all realizations or paths in the state and control spaces, as in gradient and Monte Carlo-like simulation methods. The latter optimization, on the original integral objective (2) is much more difficult than just optimizing the pure deterministic function appearing in the argument of the minimum in (7).

In order to facilitate the calculation of the minimum in (7), it is assumed that the cost function is quadratic in the control,

$$C(\mathbf{x},\mathbf{u},t)=C_0(\mathbf{x},t)+\mathbf{C}_1^T(\mathbf{x},t)\mathbf{u}+\frac{1}{2}\mathbf{u}^TC_2(\mathbf{x},t)\mathbf{u}, \tag{9}$$

and similarly that the dynamics are linear in the control,

$$\mathbf{F}(\mathbf{x},\mathbf{u},t)=\mathbf{F}_0(\mathbf{x},t)+F_1(\mathbf{x},t)\mathbf{u}. \tag{10}$$

In the case of nearly quadratic costs and nearly linear dynamics, (9–10) can be considered as local approximations for the instantaneous cost function and dynamical drift vector, respectively.

The quadratic costs assumption is not uncommon in applications, since it may be more realistic to have costs grow faster than a linear rate in the control due to increased inefficiencies, e.g., as with the inclusion of less efficient machinery or less skilled workers with the rise in production. Also, a quadratic, or near-quadratic, costs assumption makes the control determination more straight forward for the algorithm encoding.

Note that this is not the classical linear quadratic (LQ) problem, in general, because the problem can be nonlinear in the state $\mathbf{x}$. Restricting the linear dynamics and quadratic costs assumption only to the control permits more realism at the modeling stage, since the complexities of the physical application usually determine the state nonlinearities. However, control is an input determined by the user so the control only LQ assumption permits better and simpler user management of control input.

Also, the proper linear control problem may be approached through the *cheap control limit* as $C_2 \to 0^+$ using the same model, especially when the determination of linear control and related convexity conditions are not standard. This is somewhat similar to the use of artificial viscosity in fluid models.

With control–only quadratic costs and linear dynamics, the regular or unconstrained control $\mathbf{u}_R$ can be calculated explicitly, using

$$\nabla_u S(\mathbf{x}, \mathbf{u}, t) = \mathbf{0},$$

to yield,

$$\mathbf{u}_R(\mathbf{x}, t) = \arg\min_u [S(\mathbf{x}, \mathbf{u}, t)] = -C_2^{-1} \cdot (\mathbf{C}_1 + F_1^T \nabla_x v^*), \tag{11}$$

where $C_2(\mathbf{x}, t)$ is assumed to be symmetric and nonsingular. For coefficient functions, $\mathbf{F}$, G, H and C, with more general control dependency, the regular control may be calculated by appropriate methods of nonlinear optimization, such as Newton's method with quadratic costs as the first approximation.

The optimal feedback control $\mathbf{u}(\mathbf{x}, t)$ is calculated as the restriction of the regular control $\mathbf{u}_R(\mathbf{x}, t)$ to the set of control constraints $\mathcal{D}_u$,

$$u_i^*(\mathbf{x}, t) = \min[U_{\max,i}, \max[U_{\min,i}, u_{R,i}(\mathbf{x}, t)]], \tag{12}$$

as in the use of component-wise or *hypercube* constraints,

$$U_{\min,i} \le U_i(\mathbf{x}, t) \le U_{\max,i}, \quad \text{for } i = 1 \text{ to } m,$$

for example. For symmetric C_2, the switch term has the simplified form,

$$S^*(\mathbf{x}, t) = S(\mathbf{x}, \mathbf{u}^*, t) = C_0 + \mathbf{F}_0 \nabla_x v^* + \frac{1}{2}(\mathbf{u}^*)^T C_2 \left(\mathbf{u}^* - 2\mathbf{u}_R\right), \tag{13}$$

which shows that the switch term (7) is quadratic (or nearly quadratic in the approximate case) in the optimal control $\mathbf{u}$ for quadratic costs. Also, since the regular control $\mathbf{u}_R$ from (11) is linear in the solution gradient $\nabla_x v^*$ and since the optimal control $\mathbf{u}$ is a piecewise linear function (including constant pieces) of $\mathbf{u}_R$ from (12), the Bellman equation (6) is a genuine nonlinear, functional partial differential equation. This will be elaborated on later.

3. Forward Computations for Optimal, Expected Trajectory

In order to obtain the expected trajectory of the dynamical system, the solution to the forward Kolmogorov equation,

$$\begin{aligned} \frac{\partial p^*}{\partial T} &= -\nabla_x^T(\mathbf{F}^* p^*) + \frac{1}{2}\nabla_x \nabla_x^T : (GG^T p^*) - \Lambda p^* \\ &+ \sum_\ell \int_{\mathcal{D}_q} \psi_\ell(dQ) p^*(\mathbf{X} - \widetilde{\mathbf{H}}_\ell^*(\mathbf{X}, Q, T), T) \\ &\cdot \ \text{Det}\left[\left(I + \left(\nabla_x \mathbf{H}_\ell^{*T}(\mathbf{X} - \widetilde{\mathbf{H}}_\ell^*(\mathbf{X}, Q, T), Q, T)\right)^T\right)^{-1}\right], \end{aligned} \tag{14}$$

is needed using the optimal feedback control $\mathbf{u}^*(\mathbf{X}, T)$ found in the backward dynamic programming sweep. The forward equation (14) is the formal adjoint of the corresponding backward Kolmogorov equation, to which the Bellman equation is related. Here,

$$p^* = p^*(\mathbf{X}, T) = p^*(\mathbf{X}, T; \mathbf{x}, t)$$

is the density of the dynamical process for the state $\mathbf{X}$ at forward time T and starting at $\mathbf{X}(t) = \mathbf{x}$ using the optimal control $\mathbf{u}^*(\mathbf{X}, t)$. Also,

$$\mathbf{F}^* = \mathbf{F}^*(\mathbf{X}, T) \equiv \mathbf{F}(\mathbf{X}, \mathbf{u}^*(\mathbf{X}, T), T)$$

is the vector drift coefficient evaluated at the optimal feedback control.

The Poisson term is more complicated since it is a jump process. Since the Poisson term is not described well elsewhere, the transformations are described in more detail than usual. The total jump intensity is

$$\Lambda = \sum_\ell \lambda_\ell = \sum_\ell \int_{\mathcal{D}_q} \psi_\ell(dQ).$$

Since forward and backward Kolmogorov equations are adjoints, and since the new state $\mathbf{X} + \mathbf{H}_\ell$ where the process jumped to appears in the backward dynamic programming equation (6), the old state $Y - \widetilde{\mathbf{H}}^*_\ell$ where the process jumped from appears in the forward equation (14). Since after an ℓ-jump the new state is

$$\mathbf{Y} = \mathbf{X} + \mathbf{H}^*_\ell(\mathbf{X}, Q, T) \equiv (I + \widehat{H}^*_\ell)[\mathbf{X}](Q, T).$$

Hence, the inverse of the transition is

$$\mathbf{X} = \left(I + \widehat{H}^*_\ell\right)^{-1}[\mathbf{Y}] \equiv \mathbf{Y} + \widetilde{\mathbf{H}}^*_\ell(\mathbf{Y}, Q, T),$$

so the equation,

$$\widetilde{\mathbf{H}}^*_\ell(\mathbf{Y}, Q, T) = \left(I - (I - \widehat{H}^*_\ell)^{-1}\right)[\mathbf{Y}](Q, T),$$

relates the inverse jump amplitude vector $\widetilde{\mathbf{H}}^*_\ell$ to the direct jump amplitude operator $\widehat{H}^*_\ell$ appearing in (14). The Jacobian in (14) comes from from the vector differential

$$d\mathbf{Y} = d\mathbf{X} + (d\mathbf{X})^T \nabla_x \mathbf{H}^*_\ell(\mathbf{X}, Q, T) = \left(I + (\nabla_x \mathbf{H}^{*T}_\ell(\mathbf{X}, Q, T))^T\right) d\mathbf{X},$$

with inverse

$$d\mathbf{X} = \left(I + (\nabla_x \mathbf{H}^{*T}_\ell(\mathbf{X}, Q, T))^T\right)^{-1} d\mathbf{Y},$$

so that the Jacobian is

$$\frac{\partial(\mathbf{X})}{\partial(\mathbf{Y})} = \text{Det}\left[\left(I + (\nabla_x \mathbf{H}^{*T}_\ell(\mathbf{X}, Q, T))^T\right)^{-1}\right].$$

Note that we must solve the backward equation, (6) with (12) for $\mathbf{u}^*$ before we can solve the forward equation (14) for the optimal density, i.e., p^*. Equation (14) is essentially given in Gihman and Skorohod [43], but here the unusual vector product in the Poisson integral is given clearly and explicitly. See also Kushner and Dupuis [76]. The diffusion contribution of this equation has been more extensively investigated and so is much better understood than the Poisson contribution.

Finally, the optimal state vector is calculated as the first moment of p,

$$\overline{\mathbf{X}^*}(T) = \overline{\mathbf{X}^*}(T; \mathbf{x}, t) \equiv \text{Mean}[\mathbf{X}^*(T)] = \int_{\mathcal{D}_x} \mathbf{dX}\; p^*(\mathbf{X}, T)\mathbf{X}, \tag{15}$$

starting at $\overline{\mathbf{X}^*}(t) = \mathbf{x}$, with similar expressions for the variance and the other moments of the state trajectory. Taking the first moment, with respect to the ith state component X_i, of both sides of the forward equation (14) results in an ordinary differential equation (ODE), in vector representation,

$$\frac{d\overline{\mathbf{X}^*}(T)}{dT} = \overline{\mathbf{F}^*}(T) + \sum_\ell \lambda_\ell \overline{\overline{\mathbf{H}^*_\ell}}(T), \tag{16}$$

with mean optimal vector drift

$$\overline{\mathbf{F}^*}(T) = \int_{\mathcal{D}_x} \mathbf{dX}\; p^*(\mathbf{X}, T)\mathbf{F}^*(\mathbf{X}, T),$$

and mean optimal vector jump amplitude

$$\overline{\overline{\mathbf{H}_\ell^*}}(T) = \frac{1}{\lambda_\ell}\int_{\mathcal{D}_q}\psi_\ell(dQ)\int_{\mathcal{D}_x}\mathbf{dX}\,p^*(\mathbf{X},T)\mathbf{H}_\ell^*(\mathbf{X},Q,T),$$

averaging over both state and mark spaces, assuming that p^* and its derivatives vanish on the state space boundary $\partial\mathcal{D}_x$. The first moment will satisfy a linear equation if the optimal feedback control $\mathbf{u}^*(\mathbf{X},t)$, plant dynamics $\mathbf{F}(\mathbf{X},\mathbf{u}^*(\mathbf{X},t),t)$, and the component jump amplitude $\mathbf{H}_\ell(\mathbf{X},Q,T)$ are linear in the state $\mathbf{X}$, with linear optimal feedback control, with sufficient restrictions on the stochastic coefficients, then the first moment will simplify to a linear equation. In general, (16) will not be a closed system of equations in the first moment $\overline{\mathbf{X}^*}$, so the whole density function p^* may be needed.

The primary results for a given application are the expected control law and the optimal expected performance response in time parametrized by current state variables.

B. COMPUTATIONAL APPROACH

For general nonlinear dynamics and performance, the backward partial differential equation of dynamic programming, Eq. (6), together with switching term, Eq. (7), can not be solved exactly. Although special formal solutions for the linear dynamics, quadratic criterion and Gaussian noise are well known (the LQG problem, e.g. [12, 2]), they require the numerical solution of matrix Ricatti equations.

1. Computational Difficulties

Two particular features make numerical approximation of (6) with (7) nonstandard. The Poisson integral term, in general, makes the problem that of solving a functional differential equation, while the particular case of a discrete jump size leads to a problem for a delayed differential equation. In either the general or the particular case (Hanson, [49]), the functional inverse image of any finite element will not in general be existing finite elements. The technique of tracking the delayed nodes was used by Hanson and co-workers for a functional differential equation [63], for a Galerkin approximation of the Bellman equation [61], and for a finite difference approximation [104]. We have had a great deal of experience in the modeling, analysis, and computation of Poisson noise models. Mesh refinement or interpolation is required to prevent *numerical pollution* of the numerical accuracy expected of standard PDE methods. The reduction of this pollution problem is closely related to Feldstein and Neves' [36] argument concerning the need for accurate determination of the locations of jump discontinuities when applying higher order methods on delay differential equations.

The second nonstandard feature is the nonlinear control switch term that is more pertinent to the constrained control problem, whether stochastic or deterministic. The fact that $\nabla_x[v^*]$ appears in the argument of the minimum of S means that S is really a nonlinear functional of $\nabla_x[v^*]$, and that (6) is a nonlinear partial differential equation. In the general constrained control case, the nonlinear PDE also has discontinuous coefficients when $\mathcal{D}_u$ is finite (see Naimipour and Hanson [96]), as it would be in most practical applications. The constrained case thus leads to switching surfaces where the control passes through the boundary of $\mathcal{D}_u$ [61].

It can be shown that the optimal switch term S^* of (7) is a piecewise quadratic function of the shadow cost $\nabla_x[v^*]$ with discontinuous coefficients. Here, the term piecewise quadratic includes pieces that are either constant, linear or quadratic. Although, the regular or unconstrained control $\mathbf{u}_R$ in (11) is a continuous linear function of $\nabla_x[v^*]$, provided the quadratic cost and linear dynamic coefficients are continuous, the optimal constrained control $\mathbf{u}^*$ in (12) is a continuous piecewise linear function of $\nabla_x[v^*]$, but with discontinuous coefficients when decomposed as coefficients of $\nabla_x[v^*]$. That is,

$$\begin{aligned}
\mathbf{u}^* &= \mathbf{u}_0^* + u_1^*\cdot\nabla_x[v^*], \qquad (17)\\
\mathbf{u}_0^* &\equiv \left[\left\{\begin{array}{ll} U_{\max,i}, & U_{\max,i} < u_{R,i} \\ -\left(C_2^{-1}\mathbf{C}_1\right)_i, & U_{\min,i} < u_{R,i} < U_{\max,i} \\ U_{\min,i}, & u_{R,i} < U_{\min,i}\end{array}\right\}\right]_{m\times 1},\\
u_1^* &\equiv \left[\left\{\begin{array}{ll} 0, & U_{\max,i} < u_{R,i} \\ -\left(C_2^{-1}F_1^T\right)_{i,j}, & U_{\min,i} < u_{R,i} < U_{\max,i} \\ 0, & u_{R,i} < U_{\min,i}\end{array}\right\}\right]_{m\times n}.
\end{aligned}$$

Clearly, the array coefficients $\mathbf{u}_0^*$ and u_1^* are discontinuous by component for nontrivial cost and dynamics, and hence discontinuous. However,$\mathbf{u}^*$ is continuous in state and time, while the decomposition in (17) leads to discontinuous coefficients $\mathbf{u}_0^*$ and u_1^*. Introducing the discontinuous coefficient decomposition of the optimal control into the optimal switching term yields

$$\begin{aligned} S^* &= S_0^* + \mathbf{S}_1^{*T} \cdot \nabla_x[v^*] + \frac{1}{2}\nabla_x^T[v^*] \cdot S_2^* \cdot \nabla_x[v^*], \\ S_0^* &\equiv C_0 + \frac{1}{2}\mathbf{u}_0^{*T} \cdot C_2 \cdot (\mathbf{u}_0^* - 2C_2^{-1} \cdot \mathbf{C}_1), \\ \mathbf{S}_1^* &\equiv \mathbf{F}_0 + \frac{1}{2}u_1^{*T} \cdot C_2 \cdot (\mathbf{u}_0^* - 2C_2^{-1} \cdot \mathbf{C}_1) \\ &\qquad + \frac{1}{2}(u_1^* - 2C_2^{-1} \cdot F_1^T)^T \cdot C_2 \cdot \mathbf{u}_0^*, \\ S_2^* &\equiv \frac{1}{2}u_1^{*T} \cdot C_2 \cdot (u_1^* - 2C_2^{-1} \cdot F_1^T). \end{aligned} \tag{18}$$

Thus, S^* is piecewise quadratic in the cost gradient $\nabla_x[v^*]$, but it inherits the discontinuous coefficients of $\nabla_x[v^*]$ from the decomposition (17), even though S^* is continuous in state and time.

The appearance of $\nabla_x[v^*]$ in the argument of the minimum, S^*, also means that the calculated $\nabla_x[v^*]$ should be smooth enough in $\mathbf{x}$ to make the minimum computations well-conditioned. Further significance of the genuine nonlinear behavior is that it requires predictor-corrector or other nonlinear techniques for (6). Predictor-corrector methods and related methods in space will be used to handle the nonlinear aspects, and with Crank-Nicolson approximations used in time for their enhanced accuracy and stability properties. Our approach is basically an optimal control modification of the work on nonlinear parabolic equations of Douglas [31, 32] and his co-workers: Dupont, Hayes and Percell.

2. Crank–Nicolson, Predictor–Corrector Finite Difference Algorithm

The integration of the Bellman equation (6-7) is backward in time, because $v^*(\mathbf{x}, t)$ is specified finally at the final time $t = t_f$, rather than at the initial time. The finite difference discretization in states and backward time is summarized below

$$\begin{aligned} \mathbf{x} &\rightarrow \mathbf{X}_\mathbf{j} = [X_{i,j_i}]_{n\times 1} = [X_{i1} + (j_i - 1) \cdot h_i]_{n\times 1}, \\ \mathbf{j} &= [j_i]_{n\times 1}, \text{ where } j_i = 1 \text{ to } M, \text{ for } i = 1 \text{ to } n; \\ t &\rightarrow T_{=} t_f - k \cdot \Delta T, \text{ for } k = 0 \text{ to } K; \\ v^*(\mathbf{X}_\mathbf{j}, T_k) &\rightarrow \mathbf{V}_{\mathbf{j},k}; \\ v_t^*(\mathbf{X}_\mathbf{j}, T_k) &\rightarrow -\frac{1}{2 \cdot \Delta T}\left(\mathbf{V}_{\mathbf{j},k+1} - \mathbf{V}_{\mathbf{j},k}\right); \\ \nabla_x[v^*](\mathbf{X}_\mathbf{j}, T_k) &\rightarrow \mathbf{DV}_{\mathbf{j},k}; \\ \nabla_x\nabla_x^T[v^*](\mathbf{X}_\mathbf{j}, T_k) &\rightarrow \mathbf{DDV}_{\mathbf{j},k}; \\ v^*(\mathbf{X}_\mathbf{j} + \mathbf{H}_{\ell,\mathbf{j}}, T_k) &\rightarrow \mathbf{VH}_{\ell,\mathbf{j},k}; \\ u_R(\mathbf{X}_\mathbf{j}, T_k) &\rightarrow \mathbf{UR}_{\mathbf{j},k}; \\ u^*(\mathbf{X}_\mathbf{j}, T_k) &\rightarrow \mathbf{U}_{\mathbf{j},k}; \\ \mathcal{L}_x[v^*](\mathbf{X}_\mathbf{j}, T_{k+0.5}) &\rightarrow \mathcal{L}_{\mathbf{j},k+0.5} \end{aligned} \tag{19}$$

where h_i is the mesh size for state i and ΔT is the step size in backward time.

The numerical algorithm is a modification of the Crank-Nicolson, predictor-corrector methods for nonlinear parabolic PDEs in [31]. Modifications are made for the switch term and delay term calculations. Derivatives are approximated with an accuracy that is second order in the local truncation error, $\mathcal{O}(h^2)$, at all interior and boundary points, where $h_i = \mathcal{O}(h)$.

The Poisson induced functional or delay term, $v^*(\mathbf{x} + \mathbf{H}_\ell, t)$, changes the local attribute of the usual PDE to a global attribute, such that the value at a node $[\mathbf{X} + \mathbf{H}_\ell]_\mathbf{j}$ will, in general, not be a node. Linear interpolation with second order error maintains the numerical integrity that is compatible with the numerical accuracy of the derivative

approximations. Even though the Bellman equation (6-7) is a single PDE, the process of solving it not only produces the optimal expected value v^*, but also the optimal expected control law u^*.

Prior to calculating the values, $\mathbf{V}_{\mathbf{j},k+1}$, at the new $(k+1)$st time step for $k = 0$ to $K-1$, the old values, $\mathbf{V}_{\mathbf{j},k}$ and $\mathbf{V}_{\mathbf{j},k-1}$, are assumed to be known, with $\mathbf{V}_{\mathbf{j},-1} \equiv \mathbf{V}_{\mathbf{j},0}$ when two final starting conditions are needed for extrapolation.

The algorithm begins with an convergence accelerating *extrapolator* (x) *start*:

$$\mathbf{V}^{(x)}_{\mathbf{j},k+0.5} = \frac{1}{2}\left(3 \cdot \mathbf{V}_{\mathbf{j},k} - \mathbf{V}_{\mathbf{j},k-1}\right), \tag{20}$$

which are then used to compute updated values of finite difference arrays such as the gradient **DV**, the second order derivatives **DDV**, the Poisson functional terms **VH**, the regular controls **UR**, the optimal controls **U**, and finally the new value of the Bellman equation spatial functional $\mathcal{L}_{\mathbf{j},k+0.5}$. These extrapolator evaluations are used in the *extrapolated predictor (xp) step*:

$$\mathbf{V}^{(xp)}_{\mathbf{j},k+1} = \mathbf{V}_{\mathbf{j},k} + \frac{1}{2} \cdot \Delta T \cdot \mathcal{L}^{(x)}_{\mathbf{j},k+0.5}. \tag{21}$$

which are then used in the *predictor evaluation (xpe) step*:

$$\mathbf{V}^{(xpe)}_{\mathbf{j},k+0.5} = \frac{1}{2}\left(\mathbf{V}^{(xp)}_{\mathbf{j},k+1} + \mathbf{V}_{\mathbf{j},k}\right), \tag{22}$$

an approximation which preserves numerical accuracy and which is used to evaluate all terms comprising $\mathcal{L}_{\mathbf{j},k+0.5}$. The evaluated predictions are used in the *corrector (xpec) step*:

$$\mathbf{V}^{(xpec,\gamma+1)}_{\mathbf{j},k+1} = \mathbf{V}_{\mathbf{j},k} + \Delta T \cdot \mathcal{L}^{(xpece,\gamma)}_{\mathbf{j},k+0.5} \tag{23}$$

for $\gamma = 0$ to $\gamma_{\max}$ until the stopping criterion is met, with *corrector evaluation (xpece) step*:

$$\mathbf{V}^{(xpece,\gamma+1)}_{\mathbf{j},k+0.5} = \frac{1}{2}\left(\mathbf{V}^{(xpec,\gamma+1)}_{\mathbf{j},k+1} + \mathbf{V}_{\mathbf{j},k}\right). \tag{24}$$

The predicted value is taken as the zero-th ($\gamma = 0$) correction,

$$\mathbf{V}^{(xpece,0)}_{\mathbf{j},k+0.5} \equiv \mathbf{V}^{(xpe)}_{\mathbf{j},k+0.5}.$$

Upon satisfying the corrector stopping criterion, then the value for the next time-step is set:

$$\mathbf{V}_{\mathbf{j},k+1} = \mathbf{V}^{(xpec,\gamma_{\max})}_{\mathbf{j},k+1}.$$

The stopping criterion for the corrections is formally derived from a comparison to a predictor corrector convergence criterion for a linearized, constant coefficient PDE [96, 59].

3. Finite Element Version of Solution Algorithm for SDP

Due to potential higher order interpolation, it is possible to reduce the number of state nodes by using the Galerkin Finite Element Method (FEM) in place of the Finite Difference Method in dynamic programming problems [20], while retaining the same level of accuracy. Thus, the Galerkin approximation is used for the optimal expected value

$$v^*(\mathbf{x},t) \simeq \widehat{V}(\mathbf{x},t) = \sum_{j=1}^{\widehat{M}} \widehat{V}_j(t) \cdot \phi_j(\mathbf{x}), \tag{25}$$

where $[\phi_i(\mathbf{x})]_{\widehat{M}\times 1}$ is a set of $\widehat{M}$ linearly independent, piecewise continuous basis functions. The basis functions have the normalization property that

$$\phi_j(\mathbf{X}_i) = \delta_{i,j},$$

at element node $\mathbf{X}_i$, implying the interpolation property $\widehat{V}(\mathbf{X}_i,t) = \widehat{V}_j(t)$. As in [64] the basis or shape functions

could be taken as a set of multi–linear functions (products of linear Lagrange interpolation functions in each state dimension) on hyper–rectangular elements (rectangular elements in two dimensions).

The conditions to determine the optimal costs $\widehat{V}_i(t)$ at each node $\mathbf{X}_i$ are given in the weak sense by the Galerkin variational equation for the residuals of the Bellman dynamic programming equation (6) residuals with respect to the basis functions ϕ_i as weights:

$$0 = \int_{\mathcal{D}_x} \mathbf{dx}\ \phi_i(\mathbf{x}) \left[\frac{\partial v^*}{\partial t}(\mathbf{x},t) + \mathcal{L}_x[v^*](\mathbf{x},t) \right], \tag{26}$$

for $i = 1$ to $\widehat{M}$. However, Dirichlet boundary condition nodes must be excluded from the set of weights forming the component Galerkin equations (26), although they remain in the applied Galerkin approximation (25), since known costs are specified for Dirichlet nodes. Before the Galerkin approximation can be used, the second order terms of the spatial operator $\mathcal{L}_x[v^*]$ must be reduced to first order by Green's theorem, i.e.,

$$\begin{aligned} \frac{1}{2}\int_{\mathcal{D}_x} \mathbf{dx}\ \phi_i(\mathbf{x}) GG^T : \nabla_x \nabla_x^T[v^*] &= -\frac{1}{2}\int_{\mathcal{D}_x} \mathbf{dx}\ \nabla_x^T[\phi_i GG^T]\nabla_x[v^*] \\ &+ \frac{1}{2}\int_{\partial\mathcal{D}_x} ds\ \phi_i \cdot \hat{\mathbf{n}}^T GG^T \nabla_x[v^*], \end{aligned} \tag{27}$$

where $\hat{\mathbf{n}}$ is the unit outward normal to the state space boundary, $\partial\mathcal{D}_x$. Now, substituting the control linear dynamics, quadratics costs model (6,13,27) into the Galerkin equation (26) yields the matrix ODE for the cost node vector

$$\widehat{\mathbf{V}} = [\widehat{V}_i(t)]_{\widehat{M}\times 1} :$$

$$\begin{aligned} 0 = \Phi \cdot \frac{d\widehat{\mathbf{V}}}{dt}(t) &+ (\widehat{B}_F(t) + \widehat{B}_G(t) + \widehat{B}_H(t) + \widehat{\partial B}_G(t)) \cdot \widehat{\mathbf{V}}(t) \\ &+ \widehat{\mathbf{C}}(t) + \widehat{\mathbf{S}}(t), \end{aligned} \tag{28}$$

where

$$\begin{aligned} \Phi &= \left[\int_{\mathcal{D}_x} \mathbf{dx}\ \phi_i(\mathbf{x})\phi_j(\mathbf{x}) \right]_{\widehat{M}\times\widehat{M}}, \\ \widehat{B}_F(t) &\equiv \left[\int_{\mathcal{D}_x} \mathbf{dx}\ \phi_i(\mathbf{x}) \mathbf{F}_0^T(\mathbf{x},t) \nabla_x[\phi_j(\mathbf{x})] \right]_{\widehat{M}\times\widehat{M}}, \\ \widehat{B}_G(t) &\equiv -\frac{1}{2}\left[\int_{\mathcal{D}_x} \mathbf{dx}\ \nabla_x^T[\phi_i(\mathbf{x}) GG^T(\mathbf{x},t)] \nabla_x[\phi_j(\mathbf{x})] \right]_{\widehat{M}\times\widehat{M}}, \\ \widehat{B}_H(t) &\equiv \left[\sum_\ell \int_{\mathcal{D}_x} \mathbf{dx} \int_{\mathcal{D}_q} \psi(dQ)\phi_i(\mathbf{x}) \left[\phi_j(\mathbf{x} + \mathbf{H}_\ell(\mathbf{x},Q,t)) \right.\right. \\ &- \left.\left. \phi_j(\mathbf{x}) \right] \right]_{\widehat{M}\times\widehat{M}}, \\ \widehat{\partial B}_G(t) &\equiv \frac{1}{2}\left[\int_{\partial\mathcal{D}_x} ds\ \phi_i(\mathbf{x}) \cdot \hat{\mathbf{n}}^T \cdot GG^T(\mathbf{x},t) \nabla_x[\phi_j(\mathbf{x})] \right]_{\widehat{M}\times\widehat{M}}, \\ \widehat{\mathbf{C}}(t) &\equiv \left[\int_{\mathcal{D}_x} \mathbf{dx}\ \phi_i(\mathbf{x}) C_0(\mathbf{x},t) \right]_{\widehat{M}\times\widehat{M}}, \\ \widehat{\mathbf{S}}(t) &\equiv \frac{1}{2}\left[\int_{\mathcal{D}_x} \mathbf{dx}\ \phi_i(\mathbf{x}) \mathbf{u}^{*T}(\mathbf{x},t) C_2(\mathbf{x},t)(\mathbf{u}^* - 2\mathbf{u}_R)(\mathbf{x},t) \right]_{\widehat{M}\times\widehat{M}}. \end{aligned}$$

For general coefficients, like $\mathbf{F}_0$, G, H and C_0, some approximate quadrature, such as Simpson's rule or Gauss-

Legendre rules, is needed to evaluate these FEM integrals of basis and coefficient functions. However, the approximate quadrature must be at least as accurate as using the selected basis functions on the given elements, e.g., $\mathcal{O}(h^{n+2})$ where the order of the size of the elements is $\mathcal{O}(h)$ for sufficiently small h using multi–linear basis on hyper–rectangular elements [64]. Note that the optimal switch term $\widehat{\mathbf{S}}$ implicitly depends on $\widehat{\mathbf{V}}$ in a nonlinear way through the optimal and regular control vectors, $\mathbf{u}^*$ and $\mathbf{u}_R$, and thus subject to calculations like (11,12), except that the Galerkin approximation (25) is used for v^*.

The Crank–Nicolson, predictor–corrector scheme used for the finite difference formulation above can be modified for the finite element method here. The basic Crank–Nicolson algorithm, sometimes difficult to see from the canonical diffusion equation example, is the mid-point quadrature for the temporal integral, followed by averages to approximate the midpoints of the unknown variable, while leaving the midpoint value for the explicit time-dependence. For the dynamic programming equation, special modifications are needed to handle the unknown control vector that augments the unknown optimal cost variable and to handle the non–local functional dependence due to the Poisson noise contributions. Thus, Crank-Nicolson modification of the Galerkin equation (28) is

$$\begin{aligned}\Phi \cdot \left[\widehat{\mathbf{V}}_{k+1} - \widehat{\mathbf{V}}_k\right] &= \Delta T \cdot \Big\{ \frac{1}{2} \Big(\widehat{B}_{F,k+0.5} + \widehat{B}_{G,k+0.5} + \widehat{B}_{H,k+0.5} \\ &+ \widehat{\partial B}_{G,k+0.5} \Big) \left[\widehat{\mathbf{V}}_{k+1} + \widehat{\mathbf{V}}_k\right] + \widehat{\mathbf{C}}_{k+0.5} + \widehat{\mathbf{S}}_{k+0.5} \Big\}, \end{aligned} \tag{29}$$

where

$$\begin{gathered} t \simeq T_k = t_f - k \cdot \Delta T \ \text{ for } \ k = 0 \ \text{ to } \ K, \\ \widehat{\mathbf{V}}(T_k) \simeq \widehat{\mathbf{V}}_k, \qquad \widehat{\mathbf{V}}_{k+0.5} \simeq 0.5 \cdot (\widehat{\mathbf{V}}_{k+1} + \widehat{\mathbf{V}}_k), \\ \widehat{B}_F(T_{k+0.5}) = \widehat{B}_{F,k+0.5}, \qquad \widehat{B}_G(T_{k+0.5}) = \widehat{B}_{G,k+0.5}, \\ \widehat{B}_H(T_{k+0.5}) = \widehat{B}_{H,k+0.5}, \qquad \widehat{\partial B}_G(T_{k+0.5}) = \widehat{\partial B}_{G,k+0.5}, \\ \widehat{\mathbf{C}}(T_{k+0.5}) = \widehat{\mathbf{C}}_{k+0.5}, \ \text{ and } \ \widehat{\mathbf{S}}(T_{k+0.5}) \simeq \widehat{\mathbf{S}}_{k+0.5}. \end{gathered}$$

An alternative collection of terms in (29) leads to a form less susceptible to catastrophic cancellation in the case of small backward time steps ΔT:

$$\widehat{A}_{k+0.5} \cdot \Delta \widehat{\mathbf{V}}_k = \widehat{\Delta B}_{k+0.5} \cdot \widehat{\mathbf{V}}_k + \Delta T \cdot \left(\widehat{\mathbf{C}} + \widehat{\mathbf{B}}\right)_{k+0.5}, \tag{30}$$

where

$$\begin{gathered} \Delta \widehat{\mathbf{V}}_k \equiv \widehat{\mathbf{V}}_{k+1} - \widehat{\mathbf{V}}_k, \\ \widehat{\Delta B}_{k+0.5} \equiv -\Delta T \cdot \left(\widehat{B}_F + \widehat{B}_G + \widehat{B}_H + \widehat{\partial B}_G\right)_{k+0.5} \end{gathered}$$

and

$$\widehat{A}_{k+0.5} \equiv \Phi + \frac{1}{2} \cdot \widehat{\Delta B}_{k+0.5},$$

with bulk subscript notation for the temporal midpoint.

The form (30) is still implicit, but the use of extrapolation, prediction and correction will convert it to a more explicit form [20]. The procedure at this point is similar to that of the finite difference method, except for the evaluation of the regular and optimal control vectors, and the more complicated matrix structure of the Galerkin equation approximations. The starting values of the backward time iteration begins with the interpolation of the final condition (8) between nodes given the node vector

$$\widehat{\mathbf{V}}_0 = [\widehat{V}_i(t_f)]_{\widehat{M} \times 1} = [\overline{Z}^*(\mathbf{X}_i)]_{\widehat{M} \times 1}. \tag{31}$$

The extrapolation step needs two starting values, so a simple expedient is to use a *post–final* value $\widehat{\mathbf{V}}_{-1} = \widehat{\mathbf{V}}_0$ to start it off, although a more intelligent guess is desirable. The *extrapolated (x) acceleration step* supplies the evaluation for the next temporal midpoint:

$$\widehat{\mathbf{V}}^{(x)}_{k+0.5} = 0.5 \cdot (3 \cdot \widehat{\mathbf{V}}_k - \widehat{\mathbf{V}}_{k-1}), \tag{32}$$

for $k = 0$ to $K - 1$. Then, the cost node vector is used to compute the regular control $\mathbf{u}_R$ in (11) and the optimal control $\mathbf{u}^*$ in (12), but based upon the Galerkin approximation (25) at the temporal midpoint $t = T_{k+0.5}$. This permits evaluation of the nonlinear optimization term $\widehat{S} \simeq \widehat{S}^{(x)}$ leading to a reduced *extrapolated predictor (xp) Galerkin equation*,

$$\widehat{A}_{k+0.5} \cdot \Delta\widehat{\mathbf{V}}_k^{(xp)} = \widehat{\Delta B}_{k+0.5} \cdot \widehat{\mathbf{V}}_k + \Delta T \cdot \left(\widehat{\mathbf{C}} + \widehat{\mathbf{S}}^{(x)}\right)_{k+0.5}. \tag{33}$$

When solved, the solution to (33) is used in the *predictor evaluation (xpe) step*,

$$\widehat{\mathbf{V}}_{k+0.5}^{(xpe)} = \widehat{\mathbf{V}}_k + 0.5 \cdot \Delta\widehat{\mathbf{V}}_k^{(xp)}, \tag{34}$$

which, again, in turn is used to update the regular and optimal control vectors. Then there is a set of corrector iterations that continue until the change is sufficiently small to meet the stopping criterion. The *corrector (xpec) Galerkin equation* is

$$\widehat{A}_{k+0.5} \cdot \Delta\widehat{\mathbf{V}}_k^{(xpec,\gamma+1)} = \widehat{\Delta B}_{k+0.5} \cdot \widehat{\mathbf{V}}_k + \Delta T \cdot \left(\widehat{\mathbf{C}} + \widehat{\mathbf{S}}^{(xpece,\gamma)}\right)_{k+0.5}, \tag{35}$$

coupled with the *corrector evaluation (xpece) step*,

$$\widehat{\mathbf{V}}_{k+0.5}^{(xpece,\gamma+1)} = \widehat{\mathbf{V}}_k + 0.5 \cdot \Delta\widehat{\mathbf{V}}_k^{(xpec,\gamma+1)}, \tag{36}$$

where the predictor evaluation is the starting correction

$$\widehat{\mathbf{V}}_k^{(xpece,0)} = \widehat{\mathbf{V}}_k^{(xpe)}.$$

See Chung, Hanson and Xu [20] for the analysis of the stability and convergence of this procedure using the heuristic comparison equation (39), presented in the next subsection, except that an eigenvalue analysis is used in [20].

4. Bellman's Curse of Dimensionality

The main difficulty in treating large systems is the dimensional computational complexity or *Curse of Dimensionality*. The order of magnitude of this complexity can simply be approximated by assuming that computation is dominated by computation of vector functions such as the nonlinearity function $\mathbf{F}(\mathbf{x}, \mathbf{u}(\mathbf{x}, t), t)$, the cost gradient $\nabla_x[v^*](\mathbf{x}, t)$ and, in the case of uncorrelated noise, the diagonalized cost Hessian array $[(\partial^2 v^*/\partial x_i^2)(\mathbf{x}, t)]_{n\times 1}$. Their computation gives a fair representation of the order of the computational and memory requirements. For either the finite difference or finite element methods [20], the order of the number of component vector function evaluations as well the memory requirements at any time step can be calculated. Here the Finite Difference representation will be used to motivate this section. Since the ith state component will have its own node index j_i in the finite approximation representation,

$$\mathbf{x} = [x_i]_{n\times 1} \longrightarrow [X_{i,j_i}]_{n\times 1},$$

the cost gradient transformed from continuous representation $\nabla_x[v^*]$ to finite difference representation **DV** will depend on the all state components and all state finite approximation indices,

$$\nabla_x[v^*](\mathbf{x}, T_k) \longrightarrow [DV_{i,j_1,j_2,\ldots,j_n}]_{n\times M_1\times M_2\times\ldots\times M_n}, \tag{37}$$

for fixed time-to-go step k. Here, in the case that each ith state component has a common number of nodes $M_i = M$, so the total number of finite representation array components are

$$n_{dv} = n \cdot \prod_{i=1}^{n} M_i = n \cdot M^n,$$

and similarly for other vector functions. The order of the computation or storage requirements will then be some multiple of this,

$$O\left(n \cdot M^n\right) = O\left(n \cdot e^{n\cdot\ln(M)}\right), \tag{38}$$

for $i = 1$ to n states and fixed time-to-go step k. Hence, the number of nodes grows exponentially with the dimension of the state space n or with the logarithm of the number of the common number of nodes per state M. Equation (38) is an analytical representation of Bellman's *Curse of Dimensionality*. This exponential growth of the *Curse of Dimensionality* is illustrated in Figure 1. Since the amount of storage is a hardware limitation, the selection the number

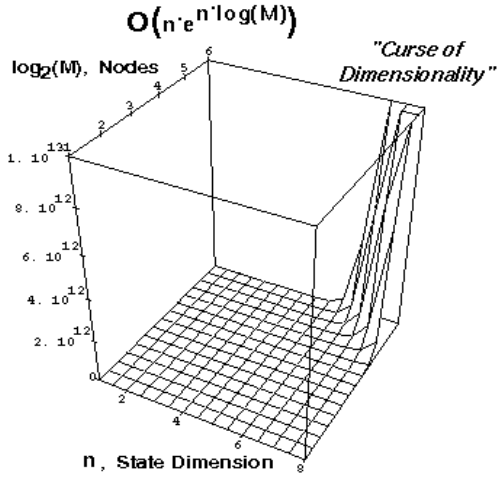

Figure 1: Order of magnitude representation of the computational or storage requirements illustrating the curse of dimensionality for stochastic dynamic programming for n states, M common nodes per state and uncorrelated noise.

of nodes, given the number of states, will typically be determined to avoid memory bound computations as nodes are chosen to satisfy accuracy requirements. The case $n = 4$ states and $M = 32$ double precision nodes requires about 48MW (where 1MW is 1 million words).

In the case of correlated noise, then a full, rather than the diagonal of the Hessian array needs to be calculated, so the Hessian array is transformed from continuous $\nabla_x \nabla_x^T[v^*]$ to finite **DDV** representation as

$$\nabla_x \nabla_x^T[v^*](\mathbf{x}, T_k) \longrightarrow [DDV_{i_1,i_2,j_1,j_2,\ldots,j_n}]_{n\times n\times M_1\times M_2\times\ldots\times M_n} ,$$

increasing the order of the computational curse of dimensionality n times to

$$\mathcal{O}\left(n^2 \cdot M^n\right) = \mathcal{O}\left(n^2 \cdot e^{n\cdot \ln(M)}\right),$$

for fixed time-to-go step k.

Thus, exponential growth in computing and storage requirements are the main bottlenecks associated with the *Curse of Dimensionality*. High performance computing (discussed in the next section) permits the solution of larger dimension problems than for mainframe computers.

C. ALGORITHMIC CONVERGENCE

An important component of our stochastic programming code, has been the mesh selection criteria by which we can be assured that the stochastic dynamic programming corrections will converge [96, 59]. This criteria follows from a heuristically constructed a linearized, constant-coefficient, comparison PDE,

$$0 = \frac{\partial \widehat{V}}{\partial t} + A : \nabla_x \nabla_x^T \widehat{V} + \mathbf{B} \cdot \nabla_x \widehat{V}, \tag{39}$$

that models the behavior of the original nonlinear stochastic dynamical programming PDE. Here, A is a constant $n \times n$ diagonal matrix and $\mathbf{B}$ is a constant n-vector. The comparison equation (39) formally corresponds to the SDE,

$$d\mathbf{X} = \mathbf{B} \cdot dt + \sqrt{2 \cdot A} \cdot d\mathbf{W},$$

provided it is interpreted in terms of the Itô calculus.

Estimates of the constant coefficients in (39) can be appropriate bounds on the *control optimized* infinitesimal moments of the diffusion approximation:

$$\begin{aligned} B_i &= \max_{(\mathbf{x},\mathbf{u},T)} \left[\text{Mean}\left[dX_i(t)|\mathbf{X}(T)=\mathbf{x},\mathbf{U}=\mathbf{u}\right]\right]/dT \\ &= \max_{(\mathbf{x},\mathbf{u},t)} [F_i(\mathbf{x},\mathbf{u},t) + \sum_{\ell=1}^{q} H_{i\ell}(\mathbf{x},t)\cdot\lambda_\ell], \end{aligned} \tag{40}$$

and

$$\begin{aligned} A_{ii} &= \frac{1}{2}\cdot\max_{(\mathbf{x},\mathbf{u},T)} \left[\text{Var}\left[dX_i(t)|\mathbf{X}(T)=\mathbf{x},\mathbf{U}=\mathbf{u}\right]\right]/dT \\ &= \frac{1}{2}\cdot\max_{(\mathbf{x},t)} \left[\sum_{k=1}^{r} G_{i,k}^2(\mathbf{x},t) + \sum_{\ell=1}^{q} H_{i,\ell}^2(\mathbf{x},t)\cdot\lambda_\ell\right], \end{aligned} \tag{41}$$

for $i = 1$ to n. However, other coefficient estimates could be used in place of (40,41).

A von Neumann Fourier analysis of the Crank-Nicolson, predictor-corrector, finite difference method applied to the linear comparison equation yields a generalized time-space mesh ratio condition that is uniform in the parameters, valid for both parabolic-like ($A \neq 0$) or hyperbolic-like ($A \equiv 0$) PDE forms

$$\sigma = \Delta T\cdot\sqrt{\left(\sum_{i=1}^{n}\frac{2A_{ii}}{h_i^2}\right)^2 + \left(\sum_{i=1}^{n}\frac{B_i}{h_i}\right)^2} < 1. \tag{42}$$

Also, since the drift appears in the corrector convergence criterion (42), upwinding schemes to enhance stability by compensating for drift in convection dominated flow should not be necessary. Since the predictor-corrector part of the method is not needed for the linear comparison equation (39) itself, the application of the predictor-corrector part of method has no utility for linear equations by themselves. However, the same methods must be used on both linear comparison and nonlinear equations so the derivation of the corrector convergence criterion for the linear comparison equation is valid for the nonlinear PDE of interest (6).

Note that if (42) is written as averages over the state space (i.e., $\overline{Q} = \frac{1}{n}\sum_{i=1}^{n} Q_i$ for some state quantity Q_i), then (42) becomes

$$\overline{\sigma}_n \equiv \frac{1}{n}\sigma = \Delta T\cdot\sqrt{(\overline{2A/h^2})^2 + (\overline{B/h})^2} < \frac{1}{n}, \tag{43}$$

where $\overline{B/h} \equiv \frac{1}{n}\sum_{i=1}^{n} B_i/h_i$, for instance. Hence, the *root-sum-squared-mean* condition (43) based on per state averages is more stringent in that $\overline{\sigma}_n < 1/n \to 0^+$, as n becomes large, i.e., the higher the state dimension the smaller the time mesh ΔT has to be relative to the mean measure of the state mesh, i.e.,

$$\Delta T < \frac{1}{n}\cdot\frac{1}{\sqrt{(\overline{2A/h^2})^2 + (\overline{B/h})^2}}.$$

When the measure of the state mesh size h (i.e., such that $h_i = \mathcal{O}(h)$) is sufficiently small, or more precisely $\overline{2A/h^2} \gg \overline{|B|/h}$, the multidimensional parabolic PDE mesh ratio condition for diffusion dominated flows is approached:

$$\sigma \longrightarrow n\cdot\Delta T\cdot\overline{2A/h^2} < 1.$$

However, in the opposite case, when the measure of the state mesh size h is sufficiently large, or more precisely $\overline{|B|/h} \gg \overline{2A/h^2}$, the multidimensional Courant-Friedrichs-Lewy (CFL) hyperbolic mesh ratio condition for convection dominated flows is approached:

$$\sigma \longrightarrow n\cdot\Delta T\cdot\overline{|B|/h} < 1.$$

However, since the full PDE of stochastic dynamic programming (6) is nonlinear for quadratic costs, the corrector

convergence criterion for the full problem is to choose the time step ΔT relative to the state mesh size h in (42) so the corrector convergence parameter σ is actually selected to be a good deal less than one to account for nonlinear and constrained control effects.

In [20], similar results were obtained for the finite element method, but using eigenvalue methods on the comparison equation.

The convergence aspect of stochastic dynamic programming calculations is extremely critical, because of the ability to predetermine convergence, maximum corrections and accuracy from the bounds on the SDE coefficients. Many other attempts to encode the dynamic programming solution algorithm have met with failure due to the lack of adequate convergence criteria.

D. OTHER COMPUTATIONAL METHODS

However, further improvements in the numerical aspects of dynamic programming can be made in the case where storage is more critical than computation. Decreasing the number of nodes while maintaining global accuracy using more accurate nodes, such as higher order finite element bases, can decrease both the storage requirements and the exponential dependence on the logarithm of the number of nodes. Finite element methods or Galerkin methods [110, 93], depending on the type of basis functions, are usually more accurate than finite difference methods, but require more costly function evaluations.

Multigrid or multilevel methods of Brandt [10] (see also [11, 89]) can also used, in conjunction with the finite element method or with other methods, in order to reduce the necessary number of nodes by successive use of fine and coarse methods to enhance accuracy beyond the accuracy of such grids when used only as single grids. Akian, Chancelier and Quadrat [1] have successfully used a variant of the multigrid for the stationary dynamic programming problem and have incorporated it into the expert system *Pandore*. Kushner and Dupuis [76] discuss the use of the multigrid methods for stochastic optimal control problems. In [77], Kushner and Jarvis apply multigrid methods to solve telecommunication control problems under the heavy traffic approximation. Hackbusch [48], and Horton and Vandewalle [65] describe a parallel multigrid method for parabolic equations that simultaneously treats both space and time grids.

The collocation method will be used as a comparative benchmark for the numerical performance of other methods. Ascher, Christiansen and Russell [4, 5] describe an efficient code for ODE boundary value problems. Dyksen, Houstis, Lynch and Rice [34], and Flaherty and O'Malley [37] find that collocation tends to out perform the Galerkin method in numerical experiments. Rabbani and Warner [103] points out difficulties with the finite element formulation when its approximation properties are not consistent with flow properties in groundwater models. The Galerkin procedure has the advantage that more theoretical results are available for it.

Other techniques have been applied to optimal control problems. Polak [99, 100] surveys gradient-type, Newton-type and other methods, mostly suitable for a deterministic problem. Mufti [94] also has surveyed computational methods in control. Larson [80, 81, 82] surveys dynamical programming methods, discusses their computational requirements and presents the state increment dynamic programming method. Jacobson and Mayne [66] discuss differential dynamic programming, based upon successive approximations and dynamic programming. They point out that their method is only suitable for optimal open loop control when applied to stochastic control problems. Shoemaker and co-workers [79, 15] have continued to make progress on the convergence and parallelization of discrete time differential dynamic programming. Guan and Luh [47] have a parallel differential dynamic programming method in which Lagrange multipliers are used to relax (parallel variable metric method) the coupling between discrete time interconnected subsystems.

Kushner [73, 74] developed a convergent (in the sense the weak convergence) finite difference method based on Markov chain approximations. It has advantages such as weakened smoothness requirements and the preservation of probabilistic properties of the stochastic model. Kusher [75] and Kushner and Dupuis [76] cover the more recent developments in the Markov chain approach such as applications to jump and reflected diffusions. However, solving by the Markov Chain Approximation may be computationally lengthy in some problems, according to Kushner [74]. This method is currently being developed for parallel computation [76, 77].

Crandall and Lions [21], Souganidis [108] and Crandall, Ishii and Lions [22] present results for vanishing viscosity method finite difference approximations for somewhat abstract Hamilton-Jacobi equations. Their results are not useful for the applications that we have modeled, due to the unrealistic restriction that the Hamiltonian be continuous, but it is expected that viscosity solution will be shown valid for the jump case eventually if not already (N. Barron, private communication). We have already mentioned the discontinuous properties that would correspond to the Hamiltonian, since in most applications the controls are bounded rather than unbounded. Their approach using

vanishing viscosity is on the right track, and the viscosity in our model can be given either a stochastic or numerical interpretation. Our ultimate goal is to be able to treat Poisson noise with fast and efficient algorithms. Gaussian noise is relatively trivial to treat compared to Poisson noise.

In Ludwig [85] and Ludwig and Varah's [86] numerical solution for optimal control of stochastic diffusions, they applied combinations of collocation and Newton's method.

Our computational results have emphasized finite difference methods [105, 50, 62, 53, 55, 54, 17, 18, 19], in order to facilitate the development of optimal parallel and vector algorithms. These results began with the one state, one control case. Currently, results are available for up to five state, five control problems, but six state problems are potentially computable with the current generation of parallel computers. Some applications may require more state dimensions.

The application in [53] treated a two species model for Lake Michigan and that strained the mainframes at that time. However, it is important to treat more interspecific interactions, especially with the high degree of turnover in species dominance in this lake. The complexity of the interactions both biologically and economically require very general control models. This is just one application, but it has a great deal of complexity with markedly different lumped species (one predator and one prey) and different economics (one sport and one commercial fishery). Complex resource applications are a primary motivation for developing algorithms for very general control problems.

III. PARALLEL COMPUTATIONAL DYNAMIC PROGRAMMING

Fast, parallel and vector algorithms appropriate for massive memory supercomputers and massively parallel processors are being developed for the modified numerical methods mentioned in the previous section. These methods are applied to the partial differential equation of dynamic programming for stochastic optimal control problems. The vector form of the finite difference equations permits advantages to be gained from both parallelization and vectorization (or matrization). The methods discussed result in execution *speed-ups*, making it more practical to numerically solve dynamic programming problems of higher dimensions than would be possible on serial processors. This is a contribution toward relieving *Bellman's Curse of Dimensionality*.

A. HARDWARE ADVANCES

Our previous supercomputing efforts [51, 52, 53, 55, 54, 17, 18, 19] have been directed toward implementations on the Alliant vector multiprocessor FX/8, on the Cray multiprocessors X-MP/48, 2s/4-128 and Y-MP/4-64, and on the Connection Machine massively parallel processors CM-2 and CM-200. Work is currently proceeding on the Connection Machine CM-5 massively parallel processor and Cray vector multiprocessor C90. These implementations greatly enhanced performance by the removal of almost all data dependent relations [101, 57]. From the Cray-1 in the 1970s to the today's vector supercomputers and massively parallel processors,machine performance has gone from megaflops (millions of floating point operations per seconds) to gigaflops (billions of floating point operations per second) and heading towards the ultracomputing goal of teraflops (trillions of floating point operations per second) [7]. Supercomputers have major differences in architecture. However, each compiler uses some variant of Fortran 90 [71, 72], so that many code optimizations are portable from one machine to the next. Vectorization can be viewed as a basic form of parallelism implemented by pipelining and so shares many optimization techniques with multiple processor type of parallel optimization. This also makes the hardware or architecture more transparent to the user.

The CM-2, CM-200 and CM-5 with their distributed memory processing have additional Fortran 90 extensions that enhance the power of the computations, but which make them somewhat dissimilar to the shared memory architecture. However, there has been a noticeable convergence of Fortran 90 extensions. Our methods require some knowledge of the architecture and the compiler, since the best optimal code must fit the template the compiler is written to optimize [56]. The main thrust in the future will be implementation on a wide range of architecture to maintain portability and avoidance of over-reliance on machines currently under development that will not survive the high performance computing environment. Getting access to the current generation of *ultracomputers* [7], such as the Cray C90, CM-5 and Intel Paragon, is essential for solving large scale computing problems. The largest problem that we have computed is 6 states with 16 nodes per state, using about 60MW double precision memory with a total of 1M nodes (i.e., one million discrete states). A dedicated Cray 2S has 128MW (64 bit words), but this requires special requests and costs many extra units. Similarly, the CM-2 has 32KB per processor, or 64MW (64 bit word) for the 16K processor machine, 128MW for the 32K processor machine and 256MW (2GB) for the full 64K processor machine. The new generation Connection Machine CM-5 has up to 1056 Sparc based processors with 32MB RAM memory

and 4 vector units each, with the property that 32 of these CM-5 processors have the power of 2.8 Y-MPs. The new generation Cray C90 can have up to 16 proprietary processors and up to 256MW RAM (MW means one mega-word of 64 bits length) per processor, while each processor is as powerful as 2.22 Y-MPs. In addition, Cray C90 may have a Cray T3D massively parallel processor attached with up to 1024 processor nodes, and 32 of these T3D processor nodes have the power of 6.7 Y-MPs. When actual maximal LINPACK performance [30] is used as a benchmark, then the CM-5 performs as well as 6.8 Y-MPs per 32 CM-5 processors, the Cray T3D performs as well as 11.8 Y-MPs per 32 T3D processors, and the C90 performs as well as 3.2 Y-MPs per processor.

A goal is the treatment of 6 or more state variables in realistic models with the present level of accuracy. Five states with 32 nodes per state requires about 32M total nodes, but only about 32K nodes if only 8 nodes per state are needed.

B. SOFTWARE ADVANCES: FASTER AND MORE EFFICIENT NUMERICAL ALGORITHMS

Many of the advances in high performance computing are due to the use of better numerical algorithms or software [111]. In order to develop algorithms for higher dimensional state spaces, a major future direction will be devising new methods for computational stochastic dynamic programming in continuous time. In place of finite difference methods, it is found that using more powerful methods which will require a smaller number of nodes for the same level of accuracy. Some of these more powerful methods are the finite element (Galerkin), the multigrid (multilevel) and collocation methods, as previously mentioned. Originally, our serial implementation was with the Galerkin method [61] as well as our first parallel implementation on the first commercial parallel processor, the Denelcor HEP, at Argonne National Laboratory (Hanson, unpublished results), but we switched to the finite difference method for ease of parallel implementation. However, we have found that we needed to return to the finite element method to reduce memory requirements [20, 64].

Some of the early work on parallel algorithms for dynamic programming was by Larson and Tse [83], and Casti, Richardson and Larson [14], but was essentially theoretical and for discrete time.

Johnsson and Mathur [90] discuss advanced procedures for programming the finite element on the massively parallel Connection Machine and also recommend efficient data structures for data parallel computers [69]. Xirouchakis and Wang [115] survey the finite elements using parallel architectures, as well as other methods such as conjugate gradient, multigrid and domain decomposition, applicable to control problems. Crow, Tylavsky and Bose survey the solution of dynamic power systems by hybrid Newton methods on parallel processors [23]. Frederickson and McBryan [40] have found a superconvergent parallel multigrid method, but Decker [28] has found that their method is not really significantly more efficient than a good parallel version of the standard multigrid algorithm, although they achieve perfect processor utilization. We will focus on techniques for mapping high dimension grids to lower dimension grids for the parallel stochastic dynamic programming algorithm.

1. Other Advanced Techniques: Loop Optimizations, Decompositions, Broadcasting

Rearranging Fortran loops to eliminate data dependencies has been very important technique for getting the most out of the so-called automatic optimizing compilers. Some understanding of the optimizing compilers (i.e., *the machine model*) is essential to transform loops into a form recognizable by the compiler. Some loop reordering techniques are changing the loop nest order and changing variables. A crucial objective is putting most of the loop work in the most inner loops of a nest. Some supercomputers will optimize only the most inner loop such as the Crays in pure vector mode, while others may parallelize and/or vectorize more than one loop. Recall that vectorization is really a primitive kind of parallelization, where the parallelization is carried out by pipelined use of vector registers, so most optimization techniques will work for both parallelization and vectorization. One technique is the collapsing of loops into a smaller number by merging indices so a smaller number of indices are used to accomplish the same iteration tasks. The use of more efficient data structures to enhance code optimization will be discussed in the next subsection. Many of these techniques are discussed in [84, 56, 29], for instance.

As we have already mentioned, that most supercomputers use similar Fortran extensions, such as Fortran 90 [71, 72], so the use of advanced computer features can be greatly facilitated, that codes can be very portable and that the hardware can be essentially transparent to the user. In addition, most extensions of the Unix language C will have most of the optimizations of Fortran 90, including the loop optimization techniques just discussed.

With many of the distributed memory, massively parallel processors, the user has the opportunity to spread the workload over the massive memory distributed over many processors. This spreading property results in suppressing

difficulties due to the growth in problem size, making many algorithms such as dynamic programming very scalable in that the workload can be divided up into many processors [112].

We have shown that our parallel stochastic dynamic algorithm exhibits scaled performance as the size of the problem increases [19]. Our CM-200 performance has exceeded that of the Cray 2S for the 5 state and 16 nodes per state problem. The Connection Machine shows great promise for applications, provided the company making it remains viable. We are starting to develop purely parallel algorithms (i.e., algorithms beginning as parallel algorithms). The Connection Machine performs recursions very well using shift operations and so we have made good use of these operations. We have used operator decomposition techniques, broadcast techniques, front end memory management [116, 117], FORALL loop structures [58], and data vault methods to enhance performance. However, our data vault work [118] has been preliminary, and we plan to use the successors to this facility and the CM-5 to be able to do a six state application. With the introduction of the massively parallel Cray T3D running on a vector multiprocessor Cray C90 as a front end computer, the advantages of massively parallel processing and vector multiprocessing are combined.

2. Vector versus Hypercube Data Structures

One of our biggest accomplishments has been to change the naive hypercube type data structure to that of the a global vector data structure [55, 54]. In the usual hypercube (or hyper–rectangle) type representation

$$\mathbf{DV} = [DV_{i,j_1,j_2,\ldots,j_n}]_{n\times M_1\times M_2\times\ldots\times M_n}\ , \tag{44}$$

for finite difference or finite element representation of component derivatives the optimal value gradient $\nabla_x[v^*]$, parallel code development and generality is hindered. This is because there must be a highly nested DO-loop for the state component index and index for each state's nodes, for example,

```
      do 1 i=1,n
         do 1 j1=1,M1
            do 1 j2=1,M2
               .....
                  do 1 jn=1,Mn
                     DV(i,j1,j2,...,jn) = ........
1     continue
```

so when it is necessary to convert the code to a different dimension a good deal of the existing code must be changed, especially state DO-loops and state subscript numbers, and state array dimensioning. Further, although the overall scale of the stochastic dynamic programming problem can be very large, the call of the subproblem for each component may not be very large out of respect for the constraints caused by *Curse of Dimensionality* for the entire problem. Hence, there may not be sufficient workload on the component basis to achieve high load balance on the parallel or vector processors and consequently not achieve high efficiency on the advanced architecture. Many of these advanced computers will only parallelize or vectorize the most inner loop (e.g., a Cray will only vectorize the most inner loop by default), so other loops in the nest will not be highly optimized, if at all.

One way around this optimization hindering data structure is the use of a vector data structure to globally represent all of the state nodes. Thus, the hypercube data structure is replaced by

$$\overline{\mathbf{DV}} = \left[\overline{DV}_{i,J}\right]_{n\times M^n}\ , \tag{45}$$

in the case of a common number $M_i = M$ nodes per state, so M^n is the total number of nodes in the global, vector data structure, with index $J = 1$ to M^n. Hence for state loops involving the vector data structure gradient $\overline{\mathbf{DV}}$, the nest depth will be only be two, the state nodes will require only one global subscript and the array dimensioning need only be changed once in each routine, when changing dimension. For instance, a typical state loop would look have the form,

```
      do 2 i=1,n
         do 2 js=1,M**n
            DV(i,js) = ........
2     continue
```

Further, a large amount of the workload is then in the global state node loop, promoting more efficient use of parallel and vector supercomputers through load balancing and evenly spreading the work load.

In the case of a common number of nodes M, the vector data structure scalar index J can be computed from the hypercube vector index

$$\mathbf{j} = [j_i]_{n\times 1} = [j_1, j_2, \ldots, j_n]^T$$

by a Fortran linear storage technique that can be used to store the indices,

$$J = J(\mathbf{j}) = 1 + \sum_{i=1}^{n} (j_i - 1) \cdot M^{(i-1)},$$

given a given vector index $\mathbf{j}$ and where $J = 1$ to M^n includes all the state nodes linearly. Also, there must be a way to go from the vector data structure, back to the hypercube data structure for computing boundary conditions, state components of derivatives and similar quantities. This state index inverse transformation is

$$j_i = j_i(J) = 1 + \text{Int}\left[\frac{(J-1) - \sum_{k=i+1}^{M} (j_k(J) - 1) \cdot M^{(k-1)}}{M^{(i-1)}}\right],$$

for $i = M$ to 1 step (-1), assuming the notation $\sum_{k=M+1}^{M} a_k \equiv 0$. The direct and inverse transformations only have to be computed once, while the actual coding is much simpler that it seems.

The vector data structure has also been used by Kushner and Jarvis [77] for applications of controlled telecommunications systems under the heavy traffic approximation. In addition, they have improved the index representation by a technique called *compressed auxiliary array index* where spatial indices are compressed into a single array and bit operations are used to perform index operations. They have found enhanced vectorization and simplified multigrid calculations.

The vector data structure would also be useful for more standard PDEs as well, since the data structure problem is mainly a PDE problem. Although there are many general routines for multi–dimensional ODEs, hardly any exist for multi–dimensional PDEs.

C. Graphical Visualization of Multidimensional Results

Scientific visualization is essentially for examining super amounts of output from supercomputer calculations. A system for the visualization of multidimensional results called *I/O View* [102, 60], utilizing an *Inner* coordinate system inside an *Outer* coordinate system, has been developed for control applications. Although the development was intended for an application of resource management and control application in an uncertain environment to display both optimal costs and components of the optimal control vector against the state vector components and parametrized by other quantities, the system is applicable to almost any multidimensional output.

The management of renewable resources such as commercial and recreational fisheries can be difficult due to lack of data, environmental uncertainty and a multitude of species interactions. The data needed to manage the resource can be biological as well as economic and environmental. Biomathematical modeling can help fill in some of the gaps in the data. Stochastic modeling can approximate the effects of environmental uncertainty. Supercomputing enables the handling of a reasonable number of interacting biological species. However, electronic visualization is essential for interpreting the multidimensional supercomputer results. Further, visualization shows resource manager how changes in management policy effect the overall economic performance of the fishery, but also shows how sensitive the performance is with variations in the poorly known data parameters. An implementation of a *world within a world* vision concept Feiner and Beshers [35] permits visualization of a 3D solution surface in an inner world, which changes along with corresponding changes in the parameters of a 3D outer world.

Refinements were made in the original notion of inner and outer worlds to improve the implementation. For example, the inner and outer coordinate systems were detached so the outer world parameters would be easily readable. A detailed user interface was developed to allow rotations and translations of the image surface, as well as many other features. This implementation allows the resource manager to visualize multidimensional resources along with parameter sensitivity of optimal value and optimal controls.

Our implementation is called *I/O View* and is schematically represented for a particular case in Figure 2. In this figure, the optimal value surface $\mathcal{S}_v^*$ is represented in the inner world (I) coordinate system as a function of two other inner coordinates, the independents states X_1 and X_2. In the same representation is the outer world (O) in

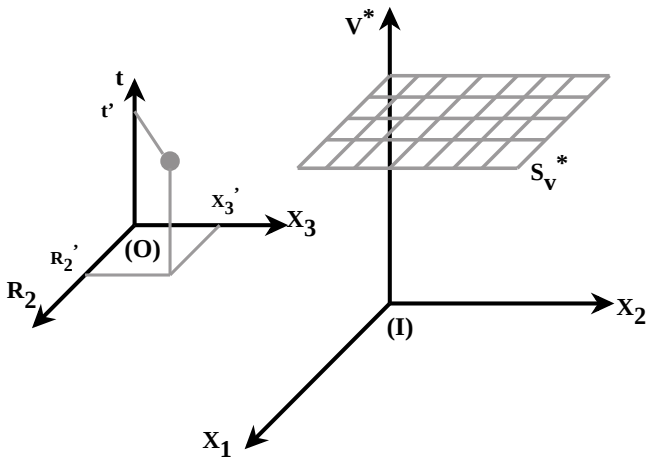

Figure 2: Schematic representation of multidimensional scientific visualization system. See the text for explanation of the Inner and Outer Worlds.

which three outer variables, a third state X_3, time t and a parameter R_2 determining the size of the X_2 state axis. The large dot in the outer world (O) represents the values of the three fixed outer variables-parameters (R_2', X_3', t'). These three parameters can be varied by moving the large dot by means of the cursor with the computed solution surface $\mathcal{S}_v^*$ changing in response to the change of outer variables to exhibit the evolution or parameter sensitivity of the optimal surface. Both inner and outer system axes have colored coded attributes as additional visual cues in the actual implementation [102]. Upon nesting more outer world coordinate systems, many more than the six dimensions displayed in Figure 2 can be represented.

The visualization was originally developed on Silicon Graphics hardware [102], but is being ported to other platforms like the NeXT [60] and improving its performance. Originally, the interface *Forms Library* of Overmars [98], designed for Silicon Graphics, was used. Remote supercomputer output is sent directly to the local visualizer by data streaming between sockets using the *Applications Communications Library* [68], thus simulating near real time access of output.

D. Numeric and Symbolic Interface

Akian, Chancelier and Quadrat [1] describe a expert system called *Pandore* that does an extraordinary number of tasks in addition to solving stationary stochastic dynamic programming problems with Gaussian noise perturbations. This system relies heavily on symbolic processing to produce proofs of existence of the solution, analysis of the solution, graphs of the solutions and several other features which are all summarized in a short LaTeX paper.

Wang and co-worker [113] have developed a symbolic computing system called *GENCRAY* that automatically generates vectorizable Cray Fortran code. *GENCRAY* can also generate parallel Cray Fortran.

Some future directions will be to integrate symbolic and numerical computation, by using symbolic computation to simplify the dynamic programming algorithm, and also by generating code that will remove general data dependencies while being portable to other machines. The **Future Directions for Research in Symbolic Computations** report [41] emphasizes the numeric-symbolic interface and the increased role parallel and super computers will play in this area.

IV. SOME RELATED METHODS

In this section, two related methods are presented as competing methods to provide contrast for stochastic dynamic programming. These methods have some similarities to SDP or are used to solve similar problems. These are differential dynamic programming and Markov chain approximation. There are many other methods that could be included, but only these two are used to conserve the scope of this chapter. In addition, these two are the ones that are most often mentioned in comparison to stochastic dynamic programming.

A. DIFFERENTIAL DYNAMIC PROGRAMMING

Differential dynamic programming (DDP) is a variant of dynamic programming in which a quadratic approximation of the cost about a nominal state and control plays an essential role. The method uses successive approximations and expansions in differentials or increments to obtain a solution of optimal control problems. The DDP method is due to Mayne [91, 66]. DDP is primarily used in deterministic problems in discrete time, although there are many variations. Mayne [91] in his original paper did give a straight-forward extension to continuous time problems, while Jacobson and Mayne [66] present several stochastic variations. The mathematical basis for DDP is given by Mayne in [92], along the relations between dynamic programming and the Hamiltonian formulation of the maximum principle. A concise, computationally oriented survey of DDP developments is given by Yakowitz [120] in an earlier volume of this series and the outline for deterministic control problems in discrete time here is roughly based on that chapter. Earlier, Yakowitz [119] surveys the use of dynamic programming in water resources applications, nicely placing DDP in the larger perspective of other dynamic programming variants. Also, Jones, Willis and Yeh [70], and Yakowitz and Rutherford [121] present brief helpful summaries with particular emphasis on the computational aspects of DDP.

1. Dynamic Programming in Discrete Time

Let κ be the discrete forward time corresponding to $t_\kappa = \kappa \cdot \Delta t$ with initial time $t_0 = 0$ or $\kappa = 0$ and final time $t_K = t_f = K \cdot \Delta t$ or $\kappa = K$ so the stages go from $\kappa = 0$ to K in steps of 1 (in opposite direction to the backward time $T_k = k \cdot \Delta T$ of SDP). Let $\mathbf{x}_\kappa = [\mathbf{x}_{i,\kappa}]_{n\times 1}$ be the n-dimensional state vector and $\mathbf{u}_\kappa = [\mathbf{u}_{i,\kappa}]_{m\times 1}$ be the m-dimensional control vector. The discrete time dynamics along the state trajectory is given recursively by

$$\mathbf{x}_{\kappa+1} = \mathbf{F}_\kappa(\mathbf{x}_\kappa, \mathbf{u}_\kappa), \text{ for } 0 \le \kappa \le K-1, \tag{46}$$

where discrete plant function $\mathbf{F}_\kappa$ is at least twice continuously differentiable in both state and control vectors.

The total cost of the trajectory from $\kappa = 0$ to K is

$$v[\mathbf{x}, \mathbf{u}; 0, K] = \sum_{\kappa=0}^{K-1} G_\kappa(\mathbf{x}_\kappa, \mathbf{u}_\kappa) + Z_K(\mathbf{x}_K), \tag{47}$$

where $G_\kappa(\mathbf{x}, \mathbf{u})$ is the discrete-time cost function, assumed to be a least twice continuously differentiable in the state and control vectors. Implicit in (47) is that the cost is separable with regard to the stages κ. The final or salvage cost is denoted by $Z_K(\mathbf{x}_K)$, assumed at least twice continuously differentiable.

The ultimate goal is to seek the minimum total cost

$$v_0^*(\mathbf{x}_\kappa) = \min_{\{\mathbf{u}_0, \cdots, \mathbf{u}_{K-1}\}} [v[\mathbf{x}, \mathbf{u}; 0, K]], \tag{48}$$

subject to the dynamic rule (46). However, for enabling dynamic programming analysis, the *variable* time-to-go cost

$$v[\mathbf{x}, \mathbf{u}; \kappa, K] = \sum_{i=\kappa}^{K-1} G_i(\mathbf{x}_i, \mathbf{u}_i) + Z_K(\mathbf{x}_K), \tag{49}$$

and its minimization,

$$v_\kappa^*(\mathbf{x}_\kappa) = \min_{\{\mathbf{u}_\kappa, \cdots, \mathbf{u}_{K-1}\}} [v[\mathbf{x}, \mathbf{u}; \kappa, K]], \tag{50}$$

is considered, subject to the final cost condition

$$v_K^*(\mathbf{x}) = Z_K(\mathbf{x}), \tag{51}$$

consistent with the definition that $\sum_{i=K}^{K-1} a_i \equiv 0$ and with salvage costs assumed in (47).

Applying to the *dynamic programming principle of optimality*,

$$v_\kappa^*(\mathbf{x}_\kappa) = \min_{\mathbf{u}_\kappa} \left[G_\kappa(\mathbf{x}_\kappa, \mathbf{u}_\kappa) + \min_{\{\mathbf{u}_{\kappa+1}, \cdots, \mathbf{u}_{K-1}\}} v[\mathbf{x}, \mathbf{u}; \kappa+1, K] \right],$$

decomposing the optimization into that of the current step plus that for the rest of the *cost-to-go*. Using the definition of time-to-go optimal cost (50) and substituting for $\mathbf{x}_{\kappa+1}$ from the recursive dynamic rule (46),

$$v^*_\kappa(\mathbf{x}_\kappa,\kappa)=\min_{\mathbf{u}_\kappa}\left[G_\kappa(\mathbf{x}_\kappa,\mathbf{u}_\kappa)+v^*_{\kappa+1}(\mathbf{F}_\kappa(\mathbf{x}_\kappa,\mathbf{u}_\kappa))\right]. \tag{52}$$

The calculation of this minimum simultaneously produces the optimal control,

$$u^*_\kappa(\mathbf{x}_\kappa)=\arg\min_{\mathbf{u}_\kappa}\left[G_\kappa(\mathbf{x}_\kappa,\mathbf{u}_\kappa)+v^*_{\kappa+1}(\mathbf{F}_\kappa(\mathbf{x}_\kappa,\mathbf{u}_\kappa))\right], \tag{53}$$

as the argument of the minimization, for $\kappa=K-1$ to 0 in backward steps (i.e., (-1) steps). The equations (52,53) comprise the *DP recursive backward sweep*. Here, the term *sweep* is used to indicate an iteration over all time steps, reserving the word *step* for either time or state steps. The use of the term sweep is not to be confused with the use in the related *Successive Sweep Method* as in [27].

The *DP recursive forward sweep* uses the optimal control u^*_κ found in the backward sweep in a forward recursion of the dynamical equation (46),

$$\mathbf{x}^*_{\kappa+1}=\mathbf{F}_\kappa(\mathbf{x}^*_\kappa,\mathbf{u}^*_\kappa(\mathbf{x}^*_\kappa)), \tag{54}$$

starting from the initial state $\mathbf{x}^*_0=\mathbf{x}_0$ and calculation future optimal states for $\kappa=0$ to $K-1$ in forward steps of 1 up to the final state $\mathbf{x}^*_K$.

While the backward and forward recursions of dynamic programming seem to give a method for computing a solution to the discrete time control problem, they do not give any actual computational method for calculation the minimum or the optimal trajectory motion that would be needed for computational implementation. The implementation is especially unclear if the problem is nonlinear (this is true also of the continuous time case). In order to make the actual computation well-posed, a quadratic approximation cost at each DDP stage k is applied.

2. Final Time DDP Backward Sweep

Each DDP iterate starts out with a current, approximate iterate c for the state-control set $\{\mathbf{x}^c_\kappa,\mathbf{u}^c_\kappa\}$ of near final time K pairs of state and control vectors, satisfying the discrete dynamics

$$\mathbf{x}^c_{\kappa+1}=\mathbf{F}_\kappa(\mathbf{x}^c_\kappa,\mathbf{u}^c_\kappa), \tag{55}$$

for $\kappa=0$ to $K-1$. Since dynamic programming takes backward steps from the final time, the starting iterate is really the *final* time iterate. It is assumed that these current, nominal iterates are somewhat close to the target optimal trajectory to justify Taylor approximations. The iterations proceed until the trajectories of successive iterates are sufficiently close.

For the final time-to-go cost (the starting step for each backward DDP sweep), a Taylor approximation about the current iterate state-control set $\{\mathbf{x}^c_K,\mathbf{u}^c_K\}$ up to quadratic terms leads to the approximate formula

$$\begin{aligned} v[\mathbf{x},\mathbf{u};K,K] &= G_K(\mathbf{x}_K,\mathbf{u}_K)\equiv Z_K(\mathbf{x}_K)\\ \simeq\widetilde{v}_K(\mathbf{x}_K,\mathbf{u}_K) &\equiv G^c_K+\nabla^T_x[G_K]\cdot\delta\mathbf{x}_K+\nabla^T_u[G_K]\cdot\delta\mathbf{u}_K\\ &+\frac{1}{2}\delta\mathbf{x}^T_K\cdot\nabla_x\nabla^T_x[G_K]\cdot\delta\mathbf{x}_K+\delta\mathbf{u}^T_K\cdot\nabla_u\nabla^T_x[G_K]\cdot\delta\mathbf{x}_K\\ &+\frac{1}{2}\delta\mathbf{u}^T_K\cdot\nabla_u\nabla^T_u[G_K]\cdot\delta\mathbf{u}_K, \end{aligned} \tag{56}$$

with increments $\delta\mathbf{x}_K=\mathbf{x}_K-\mathbf{x}^c_K$ and $\delta\mathbf{u}_K=\mathbf{u}_K-\mathbf{u}^c_K$. The following array Taylor coefficients are redefined for computational storage as

$$\begin{aligned} G^c_K &\equiv G_K=G_K(\mathbf{x}^c_K,\mathbf{u}^c_K)\\ (\mathbf{E}^c_K)^T &\equiv \nabla^T_x[G_K]=\nabla^T_x[G_K](\mathbf{x}^c_K,\mathbf{u}^c_K)\\ (\mathbf{D}^c_K)^T &\equiv \nabla^T_u[G_K]=\nabla^T_u[G_K](\mathbf{x}^c_K,\mathbf{u}^c_K)\\ A^c_K &\equiv \frac{1}{2}\nabla_x\nabla^T_x[G_K]=\frac{1}{2}\nabla_x\nabla^T_x[G_K](\mathbf{x}^c_K,\mathbf{u}^c_K) \end{aligned} \tag{57}$$

$$\begin{aligned} B_K^c &\equiv \nabla_u \nabla_x^T[G_K] = \nabla_u \nabla_x^T[G_K](\mathbf{x}_K^c, \mathbf{u}_K^c) \\ C_K^c &\equiv \frac{1}{2}\nabla_u \nabla_u^T[G_K] = \frac{1}{2}\nabla_u \nabla_u^T[G_K](\mathbf{x}_K^c, \mathbf{u}_K^c), \end{aligned}$$

where the notation A^T denotes the transpose of array A, ∇_x is the gradient in the state and ∇_u is the gradient in the control. In this section, gradients are evaluated at the current state-control set for the time considered. It is assumed that the set $\{\mathbf{x}_K, \mathbf{u}_K\}$ are variable points on a trajectory nearby to the current set $\{\mathbf{x}_K^c, \mathbf{u}_K^c\}$. However, we will not need the $\mathbf{x}_K$ state values themselves except here for the purposes of analysis, but only the coefficients defining the functional form of the quadratic approximation (note the analogous properties to that of the classical LQ control problem).

In absence of constraints, the improved approximation to the optimal control is then the current approximation to the optimal control is obtained from the critical points of $\widetilde{v}$ at $\kappa = K$,

$$\begin{aligned} \nabla_u[\widetilde{v}_K](\mathbf{x}_K, \widetilde{\mathbf{u}}_K^*) &= \mathbf{E}_K^c + B_K^c \cdot \delta\mathbf{x}_K + 2 \cdot C_K^c \cdot \delta\widetilde{\mathbf{u}}_K^* \\ &= \nabla_u^T[G_K] + \nabla_u \nabla_x^T[G_K] \cdot \delta\mathbf{x}_K + \nabla_u \nabla_u^T[G_K] \cdot \delta\widetilde{\mathbf{u}}_K^* \\ &= 0, \end{aligned} \tag{58}$$

so that

$$\begin{aligned} &\delta\widetilde{\mathbf{u}}_K^* \\ &\quad= \widetilde{\mathbf{u}}_K^*(\mathbf{x}_K) - \mathbf{u}_K^c \\ &\quad= -\frac{1}{2}C_K^{-c} \cdot (\mathbf{E}_K^c + B_K^c \cdot \delta\mathbf{x}_K) \\ &\quad= -(\nabla_u \nabla_u^T[G_K])^{-1} \cdot \left(\nabla_u[G_K] + \nabla_u \nabla_x^T[G_K] \cdot \delta\mathbf{x}_K\right) \\ &\quad\equiv \boldsymbol{\alpha}_K^c + \beta_K^c \cdot \delta\mathbf{x}_K. \end{aligned} \tag{59}$$

This a linear control law in the state perturbation $\delta\mathbf{x}_K$ with fixed coefficient $\boldsymbol{\alpha}_K^c$ and gain coefficient β_K^c, provided that the inverse $C_K^{-c} = (C_K^c)^{-1}$ is well defined (e.g., $C_K^c = 2\nabla_u \nabla_u^T[G_K]$ is positive definite and hence nonsingular; however, positive definiteness is only needed near the optimal trajectory). Since the expansion in Taylor approximations with only a few terms is good only if both $\delta\mathbf{x}_K$ and $\delta\mathbf{u}_K$ are small, a problem in consistency arises if the fixed part of $\delta\mathbf{u}_K$, $\boldsymbol{\alpha}_K^c$, is not small in (59). However, nearby the optimal trajectory, $\boldsymbol{\alpha}_K^c = -(\nabla_u \nabla_u^T[G_K])^{-1} \cdot \nabla_u[G_K]$ should be small since $\nabla_u[G_K]^* = 0$ on the optimal trajectory at the final time. In the case that $\boldsymbol{\alpha}_K^c$, as when the starting guess in not good, Jacobson and Mayne [66] suggest multiplying $\boldsymbol{\alpha}_K^c$ by a sufficiently small parameter ε until the iterations are closer to the optimal trajectory. These comments apply on the time horizon, $0 \le \kappa \le K$, not just for the time $\kappa = K$.

In the presence of constraints, the linear control law (59) yields only the regular control, which must be modified for the constraints, as in the previous section for SDP. However, for inequality constraints, the inconsistency problem of non-small $\boldsymbol{\alpha}_K^c$ can arise, so penalty functions must be used to move the constraints to the objective functional.

The corresponding approximation to the optimal cost value function follows from substituting the current approximation of the optimal control:

$$\begin{aligned} \widetilde{v}_K^*(\mathbf{x}_K) &= \widetilde{v}_K(\mathbf{x}_K, \widetilde{\mathbf{u}}_K^*(\mathbf{x}_K)) \\ &= R_K^c + (\mathbf{Q}_K^c)^T \cdot \delta\mathbf{x}_K + \delta\mathbf{x}_K^T \cdot P_K^c \cdot \delta\mathbf{x}_K, \end{aligned} \tag{60}$$

reduced to a quadratic function in $\delta\mathbf{x}_K$ only. Upon calculation, the *current coefficients* are given by

$$\begin{aligned} P_K^c &= A_K^c - \frac{1}{4}(B_K^c)^T C_K^{-c} B_K^c \\ (\mathbf{Q}_K^c)^T &= (\mathbf{E}_K^c)^T - \frac{1}{2}(\mathbf{D}_K^c)^T C_K^{-c} B_K^c \\ R_K^c &= G_K^c - \frac{1}{4}(\mathbf{D}_K^c)^T C_K^{-c} \mathbf{D}_K^c. \end{aligned} \tag{61}$$

The last, currently fixed coefficient R_K^c is not essential for the iteration, unless the value optimal cost needed as a result.

3. General DDP Backward Sweep in Time

The coefficients at final time K define the beginning of the DDP backward sweep for the current iteration c, while the coefficients for the remaining iteration times are found by marching backward from $\kappa = K-1$ to $\kappa = 0$ assuming the quadratic value induction hypothesis,

$$\widetilde{v}^*_{\kappa+1}(\mathbf{x}_{\kappa+1}) = R^c_{\kappa+1} + (\mathbf{Q}^c_{\kappa+1})^T \cdot \delta\mathbf{x}_{\kappa+1} + \delta\mathbf{x}^T_{\kappa+1} \cdot P^c_{\kappa+1} \cdot \delta\mathbf{x}_{\kappa+1}, \tag{62}$$

based on the formula (60) for $\kappa = K$, where $\delta\mathbf{x}_{\kappa+1} \equiv \mathbf{x}_{\kappa+1} - \mathbf{x}^c_{\kappa+1}$. Assuming both $\mathbf{x}_{\kappa+1}$ and $\mathbf{x}^c_{\kappa+1}$ obey the dynamics of (46) (if not, the calculation is more complex and not very suitable for implementation), then the increment in the vector dynamics is expanded as the quadratic,

$$\begin{aligned}
\delta\mathbf{x}_{\kappa+1} &= \mathbf{F}_\kappa(\mathbf{x}_\kappa, \mathbf{u}_\kappa) - \mathbf{F}_\kappa(\mathbf{x}^c_\kappa, \mathbf{u}^c_\kappa) \\
&= \mathbf{F}_\kappa(\mathbf{x}^c_\kappa + \delta\mathbf{x}_\kappa, \mathbf{u}^c_\kappa + \delta\mathbf{u}_\kappa) - \mathbf{F}_\kappa(\mathbf{x}^c_\kappa, \mathbf{u}^c_\kappa) \\
&\simeq (\nabla_x \mathbf{F}^T_\kappa)^T \cdot \delta\mathbf{x}_\kappa + (\nabla_u \mathbf{F}^T_\kappa)^T \cdot \delta\mathbf{u}_\kappa \\
&+ \left[\frac{1}{2}\delta\mathbf{x}^T_\kappa \cdot (\nabla_x \nabla^T_x F_{i,\kappa}) \cdot \delta\mathbf{x}_\kappa + \delta\mathbf{u}^T_\kappa \cdot (\nabla_u \nabla^T_x F_{i,\kappa}) \cdot \delta\mathbf{x}_\kappa \right. \\
&\qquad \left. + \frac{1}{2}\delta\mathbf{u}^T_\kappa \cdot (\nabla_u \nabla^T_u F_{i,\kappa}) \cdot \delta\mathbf{u}_\kappa \right]_{n\times 1},
\end{aligned} \tag{63}$$

which is similar to the expansion of the scalar cost G_K in (56).

Next the argument of the minimum of the Principle of Optimality in (52) is expanded by substituting the quadratic expansions for G_κ in (56) and $\mathbf{F}_\kappa$ in (63), while retaining only terms up to quadratic order (being careful to expand about the later current state $\mathbf{x}^c_{\kappa+1}$ when using $\widetilde{v}^*_{\kappa+1}(\mathbf{x}_{\kappa+1})$ as required by the induction hypothesis (62)), the argument becomes

$$\begin{aligned}
\widetilde{v}[\mathbf{x}, \mathbf{u}; \kappa, K] &\equiv G_\kappa(\mathbf{x}_\kappa, \mathbf{u}_\kappa) + \widetilde{v}^*_{\kappa+1}(\mathbf{F}_\kappa(\mathbf{x}_\kappa, \mathbf{u}_\kappa)) \\
\simeq \widetilde{v}_\kappa(\mathbf{x}_\kappa, \mathbf{u}_\kappa) &\equiv V^c_\kappa + (\mathbf{E}^c_\kappa)^T \cdot \delta\mathbf{x}_\kappa + (\mathbf{D}^c_\kappa)^T \cdot \delta\mathbf{u}_\kappa \\
&+ \delta\mathbf{x}^T_\kappa \cdot A^c_\kappa \cdot \delta\mathbf{x}_\kappa + \delta\mathbf{u}^T_\kappa \cdot B^c_\kappa \cdot \delta\mathbf{x}_\kappa + \delta\mathbf{u}^T_\kappa \cdot C^c_\kappa \cdot \delta\mathbf{u}_\kappa.
\end{aligned} \tag{64}$$

This is the quadratic approximation to the minimization argument in (52). The corresponding coefficients are

$$\begin{aligned}
V^c_\kappa &= G^c_\kappa + R^c_{\kappa+1}, \\
(\mathbf{E}^c_\kappa)^T &= \nabla^T_x[G_\kappa](\mathbf{x}^c_\kappa, \mathbf{u}^c_\kappa) + (\nabla_x \mathbf{F}^T_\kappa)^T(\mathbf{x}^c_\kappa, \mathbf{u}^c_\kappa)\mathbf{Q}^c_{\kappa+1}, \\
(\mathbf{D}^c_\kappa)^T &= \nabla^T_u[G_\kappa](\mathbf{x}^c_\kappa, \mathbf{u}^c_\kappa) + (\nabla_u \mathbf{F}^T_\kappa)^T(\mathbf{x}^c_\kappa, \mathbf{u}^c_\kappa)\mathbf{Q}^c_{\kappa+1}, \\
A^c_\kappa &= \frac{1}{2}\nabla_x \nabla^T_x[G_\kappa] + \frac{1}{2}(\nabla_x \nabla^T_x \mathbf{F}^T_\kappa)\mathbf{Q}^c_{\kappa+1} + (\nabla_x \mathbf{F}^T_\kappa)P^c_{\kappa+1} \cdot (\nabla_x \mathbf{F}^T_\kappa)^T \\
B^c_\kappa &= \nabla_u \nabla^T_x[G_\kappa] + \frac{1}{2}(\nabla_u \nabla^T_x \mathbf{F}^T_\kappa)\mathbf{Q}^c_{\kappa+1} + 2(\nabla_u \mathbf{F}^T_\kappa)P^c_{\kappa+1} \cdot (\nabla_x \mathbf{F}^T_\kappa)^T \\
C^c_\kappa &= \frac{1}{2}\nabla_u \nabla^T_u[G_\kappa] + \frac{1}{2}(\nabla_u \nabla^T_u \mathbf{F}^T_\kappa)\mathbf{Q}^c_{\kappa+1} + (\nabla_u \mathbf{F}^T_\kappa)P^c_{\kappa+1} \cdot (\nabla_u \mathbf{F}^T_\kappa)^T
\end{aligned} \tag{65}$$

where arguments of the gradients at the current time in the last three coefficients have been omitted.

Minimizing the quadratic approximation in (64),

$$0 = (\nabla_u[\widetilde{v}_\kappa])^* = \mathbf{D}^c_\kappa + B^c_\kappa \cdot \delta\mathbf{x}_\kappa + 2 \cdot C^c_\kappa \cdot \delta\widetilde{\mathbf{u}}^*_\kappa,$$

yields the current approximation to the optimal control at κ,

$$\delta\widetilde{\mathbf{u}}^*_\kappa = \boldsymbol{\alpha}^c_\kappa + \beta^c_\kappa \cdot \delta\mathbf{x}_\kappa, \tag{66}$$

where

$$\boldsymbol{\alpha}^c_\kappa \equiv -\frac{1}{2}C^{-c}_\kappa \cdot \mathbf{D}^c_\kappa \text{ and } \beta^c_\kappa \equiv -\frac{1}{2}C^{-c}_\kappa \cdot B^c_\kappa. \tag{67}$$

These latter two coefficients need only be saved for the current approximate optimal control since they define its linear function in the state increment. The quadratic approximation is critical for the straight forward determination of the control, in this case unconstrained. Recall that in the case where $\boldsymbol{\alpha}_\kappa^c$ is not small, then a small coefficient can used where $\boldsymbol{\alpha}_\kappa^c$ is applied [91] (i.e., $\boldsymbol{\alpha}_\kappa^c$ is replaced by $\varepsilon\boldsymbol{\alpha}_\kappa^c$ with sufficiently small ε until convergence of the successive approximations to v_κ^* is assured). The difficulty arises when the approximations are too far from the optimal trajectory, but close to the optimal trajectory $\mathbf{D}_\kappa^c = \nabla_u[H_\kappa^c]$ should be small if H_κ^C is some pseudo-Hamiltonian to which the minimum principle applies.

Upon substituting the linear control law (66) into the bivariate quadratic approximation (64), the formula for induction hypothesis (62) is obtained for time κ given the result for $\kappa+1$,

$$\widetilde{v}_\kappa^*(\mathbf{x}_\kappa) = R_\kappa^c + (\mathbf{Q}_\kappa^c)^T\cdot\delta\mathbf{x}_\kappa + \delta\mathbf{x}_\kappa^T\cdot P_\kappa^c\cdot\delta\mathbf{x}_\kappa, \tag{68}$$

proving the induction part of the current backward sweep. Further, the state dependent quadratic coefficients are given in the procedure as

$$\begin{aligned} P_\kappa^c &= A_\kappa^c + \frac{1}{2}(B_\kappa^c)^T\beta_\kappa^c \\ (\mathbf{Q}_\kappa^c)^T &= (\mathbf{E}_\kappa^c)^T - \frac{1}{2}(\mathbf{D}_\kappa^c)^T\beta_\kappa^c \\ R_\kappa^c &= V_\kappa^c - \frac{1}{4}(\mathbf{D}_\kappa^c)^T\boldsymbol{\alpha}_\kappa^c, \end{aligned} \tag{69}$$

for $\kappa = K$ to 0 in backward steps, including the final condition in (61) as well. This concludes the backward sweep for iterate c.

A primary property of DDP is that it is exact, in infinite precision, for the LQP (Linear Quadratic Problem), since the Taylor approximations would be exact in the LQP case [91].

4. DDP Forward Sweep

In the forward sweep for iterate c, the cth backward sweep approximation for the optimal control,

$$\widetilde{\mathbf{u}}_\kappa^* = \mathbf{u}_\kappa^c + \delta\widetilde{\mathbf{u}}_\kappa^* = \mathbf{u}_\kappa^c + \boldsymbol{\alpha}_\kappa^c + \beta_\kappa^c\cdot\delta\mathbf{x}_\kappa, \tag{70}$$

given in increment form by (66) for all the times $\kappa = 0$ to K, is used to calculate improvements in the optimal state trajectory. If $\boldsymbol{\alpha}_\kappa^c$ is not small, then (70) is replaced by

$$\widetilde{\mathbf{u}}_\kappa^* = \mathbf{u}_\kappa^c + \varepsilon\cdot\boldsymbol{\alpha}_\kappa^c + \beta_\kappa^c\cdot\delta\mathbf{x}_\kappa,$$

with $0 < \varepsilon \leq 1$ selected to foster convergence until $\boldsymbol{\alpha}_\kappa^c$ is small again [91, 121]. The discrete dynamic law (46),

$$\widetilde{\mathbf{x}}_{\kappa+1}^* = \mathbf{F}_\kappa(\widetilde{\mathbf{x}}_\kappa^*, \widetilde{\mathbf{u}}_\kappa^*), \tag{71}$$

can be solved recursively for $\kappa = 0$ to $K-1$ in forward steps since the linear control law approximation is given. Once done, the successor iterate is defined as

$$\mathbf{x}_\kappa^{c+1} = \widetilde{\mathbf{x}}_\kappa^*, \tag{72}$$

for $\kappa = 0$ to K. However, for computational storage considerations, the new iterate just replaces the old, $\mathbf{x}_\kappa^{c+1} \longrightarrow \mathbf{x}_\kappa^c$, saving the old iterate only for the next check of the iteration stopping criterion.

The combined backward and forward sweep iterations end when an appropriate stopping criterion is reached, such as

$$\max_\kappa ||\widetilde{v}_\kappa^*(\mathbf{x}_\kappa^{c+1}) - \widetilde{v}_\kappa^*(\mathbf{x}_\kappa^c)|| \leq \text{tol}_v,$$

in some suitable norm. This value stopping criterion may also be supplemented using successive states,

$$\max_\kappa ||\mathbf{x}_\kappa^{c+1} - \mathbf{x}_\kappa^c|| \leq \text{tol}_x,$$

or using successive controls,

$$\max_{\kappa} ||\mathbf{u}_{\kappa}^{c+1} - \mathbf{u}_{\kappa}^{c}|| \leq \text{tol}_u,$$

where the tolerances tol_γ are prescribed.

5. Similarities and Differences between DDP and SDP

The obvious difference between DDP and SDP, as presented here, is that DDP is primarily applied to deterministic problems, rather than stochastic as is SDP. However, Jacobson and Mayne [66] and others do give stochastic variants, including formulation for the LQGP (Linear, Quadratic and Gaussian Problem). These authors also present the continuous time case, as well as give estimates for the errors committed in DDP and step-size adjustments.

Another difference is that the DDP iterations are over the entire time horizon for each iterate, where as SDP iterations in the backward sweep range over only a single time step followed by a backward march in time for the next set of iterations. That is, an DDP backward sweep iteration ranges over both state and temporal spaces, whereas the SDP backward sweep ranges over only the state space and stop when successive values for that time step satisfy corrector stopping criterion. In contrast, an approximation for all times is found in DDP, before proceeding to the next iteration. In term of the time domain, roughly speaking, DDP is somewhat analogous to the point Jacobi method, while SDP is more like Gauss-Seidel.

Also, a mesh selection ratio recipe is available in SDP for selecting the ratio of the time step to an approximate measure of the space step. However, DDP does not discretize either the state $\mathbf{x}$ or the control $\mathbf{u}$ variables [121, 70] so the mesh selection ratio is not relevant, while the SDP here does discretize the state but the control is computed as output with the value. The computational and storage cost per time step of SDP is exponential $O(n \cdot M^n)$ in (38) consistent with the curse of dimensionality and with M finite difference nodes per state. However, for DDP the cost per stage is the cost of computing the coefficient arrays such as A_{κ}^{c}, B_{κ}^{c} and C_{κ}^{c} from the associated Hessian arrays, as well as the control law coefficients $\boldsymbol{\alpha}_{\kappa}^{c}$ and β_{κ}^{c}, so the storage cost is $O(n^2 + m \cdot n)$ or $O(n^3 + m \cdot n^2 + m^2 \cdot n)$ if the Hessian of the dynamics vector $\mathbf{F}_{\kappa}$ is stored for the stage and the computational cost is $O(m^3 + m^2 \cdot n)$ for the control law and $O(n^4 + m \cdot n^3 + m^2 \cdot n^2)$ for A_{κ}^{c}, B_{κ}^{c} and C_{κ}^{c}. depending how the multiplications in (65) are computed (see also [121, 70] for a somewhat different accounting). The DDP manages to keep the curse of dimensionality under control. A major reduction in storage and computation occurs in DDP since calculations are concentrated in the neighborhood of an optimal trajectory, whereas SDP covers the whole state space.

The assumption that cost are quadratic in the control for SDP is somewhat similar to the quadratic assumption in DDP, except that in DDP the quadratic is in both costs and dynamics with respect to both control and state. In SDP, the critical calculation of the optimal control requires only that the cost be quadratic and the dynamics be linear in the control, which is more realistic where the nonlinearity in the dynamics is an essential part of the model. In the version of SDP presented here, the quadratic costs were presented an part of the model rather than part of the method of solution, although more complex cost and dynamics could be treated by a quadratic Taylor approximation of the cost and a linear Taylor approximation of the dynamics to formulate an iterative procedure to solve the more general SDP problem. In DDP computation is mainly the computation of the temporal functional coefficients, where as in the SDP here the computation permits general state and time dependence.

Quadratic convergence for DDP has been shown by Murray and Yakowitz [95]. For SDP in continuous time, Naimipour and Hanson [96, 59] have a heuristic comparison argument for convergence, and the convergence is linear, in that,

$$V_{\mathbf{j},k}^{(\gamma+1)} - V_{\mathbf{j},k}^{(\infty)} = \theta \cdot \left(V_{\mathbf{j},k}^{(\gamma)} - V_{\mathbf{j},k}^{(\infty)} \right),$$

where γ is the correction counter and θ is the corrector convergence parameter for which the σ in (42) is an approximation.

The forward sweeps are quite different in the two cases, due to the difference in stochastic and deterministic formulations. The explicit recursion in DDP is relatively trivial compared to solving the forward Kolmogorov equations for the state density given the stochastic optimal control to get the optimal expected state trajectory.

6. DDP Variations and Applications

Yakowitz [119, 120] has given a thorough survey of the computation and techniques of differential dynamic programming in 1989. Liao and Shoemaker [79] studied convergence in unconstrained DDP methods and have found

that adaptive shifts in the Hessian are very robust and yield the fastest convergence in the case that the problem Hessian matrix is not positive definite. Chang, Shoemaker and Liu [16] solve for optimal pumping rates to remediate groundwater pollution contamination using finite elements and hyperbolic penalty functions to include constraints in the DDP method. Culver and Shoemaker [24, 25] include flexible management periods into the model and use a faster Quasi-Newton version of DDP. Earlier, Murray and Yakowitz [95] had compared DDP and Newton's methods to show that DDP inherited the quadratic convergence of Newton's method. Caffey, Liao and Shoemaker [15] develop a parallel implementation of DDP that is speeded up by reducing the number of synchronization points over time steps.

B. MARKOV CHAIN APPROXIMATION

Another approach to finite differences is the well developed *Markov Chain Approximation (MCA)* of Kushner [73, 74]. Recent developments are surveyed and further advanced by Kushner [75], and by Kushner and Dupuis [76], with special attention to methods for jump and reflected diffusions. This method applies a Markov chain approximation to continuous time, continuous state stochastic control problems by renormalizing finite differences forms as proper Markov chain transition probabilities. These transition probabilities arise when deriving finite difference versions of the dynamic programming equation. An important advantage of this method is that the Markov chain approximation facilitates convergence proofs for the numerical methods in terms of probabilistic arguments. Probabilistic interpretation of the approximation is a major motivation for the formulation of this method. Here, the MCA method is given a formal presentation, in the spirit of the SDP notation and formulation to facilitate comparison. The reader should refer to the above references for the greater detail, especially Kushner and Dupuis [76] for a multitude of variations and convergence proofs.

1. MCA Dynamic Programming Model Formulation

Consider the stochastic diffusion without Poisson jumps governed by the stochastic differential equation (SDE)

$$d\mathbf{X}(t) = \mathbf{F}(\mathbf{X}, \mathbf{U}, t)dt + G(\mathbf{X}, t)d\mathbf{W}(t), \tag{73}$$

where the notation is the same as in (1). It is assumed that drift $\mathbf{F}$ and Gaussian coefficient G are bounded, continuous and Lipshitz continuous in the state $\mathbf{X}$, while $\mathbf{F}$ is uniformly so in the control $\mathbf{U}$. Further, let the expected cost objective functional be

$$\begin{aligned} \overline{V}(\mathbf{x}, \mathbf{u}, t) &= \text{Mean}\left[\int_t^{t_f} d\tau\, C(\mathbf{X}(\tau), \mathbf{U}(\mathbf{X}(\tau), \tau), \tau)\right. \\ &\quad \left. \mid \mathbf{X}(t) = \mathbf{x}, \mathbf{U} = \mathbf{u}\right] + \overline{Z}(\mathbf{x}, t_f), \end{aligned} \tag{74}$$

with the same notation as in (2). It is assumed that the instantaneous cost C and the salvage cost $\overline{Z}$ are bounded and continuous. Much also may depend on the boundary conditions. The final side condition is the same as that in (8) for SDP.

The optimal costs are defined as

$$v^*(\mathbf{x}, t) = \inf_{\mathbf{u}} \left[\overline{V}(\mathbf{x}, \mathbf{u}, t)\right], \tag{75}$$

using the infimum ($\inf$), instead of the less general minimum ($\min$) in the spirit of [76], over all admissible controls.

Upon application of the principle of optimality, the dynamic programming equation for the optimal expected cost v^* is

$$\begin{aligned} 0 &= v_t^* + \mathcal{L}_x[v^*](\mathbf{x}, t) \\ &= v_t^* + \inf_{\mathbf{u}} \left[\mathbf{F}^T(\mathbf{x}, \mathbf{u}, t)\nabla_x[v^*] + \frac{1}{2}(GG^T)(\mathbf{x}, t) : \nabla_x \nabla_x^T[v^*]\right. \\ &+ \left. C(\mathbf{x}, \mathbf{u}, t)\right]. \end{aligned} \tag{76}$$

Since (76) is a backward equation and since we want to keep it simple, (76) is approximated with the Backward Euler

approximation

$$\begin{aligned} v^*_{\kappa-1}(\mathbf{x}) &= v^*_\kappa(\mathbf{x}) + \Delta t_{\kappa-1} \cdot \inf_{\mathbf{u}} \left[\mathbf{F}^T_\kappa(\mathbf{x},\mathbf{u}) \nabla_x [v^*_\kappa] \right. \\ &+ \left. \frac{1}{2}(GG^T)_\kappa(\mathbf{x}) : \nabla_x \nabla^T_x [v^*_\kappa] + C_\kappa(\mathbf{x},\mathbf{u}) \right], \end{aligned} \tag{77}$$

for $\kappa = 1$ to K, with $v^*_\kappa(\mathbf{x}) \simeq v^*(\mathbf{x}, t_\kappa)$, and similarly for $\mathbf{F}_\kappa$, G_κ and C_κ, while $t_\kappa = t_{\kappa-1} + \Delta t_{\kappa-1}$ is time in terms of the forward index κ. The final condition is $v^*_K(\mathbf{x}) = v^*(\mathbf{x}, t_f) = \overline{Z}^*(\mathbf{x}, t_f)$. (The correlation, in the case of constant time increments, between the forward index κ and the backward index k used in SDP is that $T_k = t_f - k \cdot \Delta T = (K-k) \cdot \Delta T = t_{K-k} = t_\kappa$, so the forward and backward time indices are related by $\kappa = K - k$. Hence, $T_0 = t_f = t_K$ and $T_K = 0 = t_0$, are the final and initial times, respectively, in either time direction.) The interpolation time increment is denoted by $\Delta t_{\kappa-1} = t_\kappa - t_{\kappa-1}$ here and has important roles to play for modeling and for numerical convergence.

However, since much of the literature on MCA, corresponding to much of the literature on stochastic control, is formulated as a stationary (time-independent) problem, the indices k and κ are treated as iteration indices for the time-independent problem. The time-independent problem may arise from ergodic problems or exit time problems or infinite horizon problems, for example.

2. MCA Local Consistency Conditions

The symbol h will indicate the the order of the spacing in the discretization of the state space for the Markov chain $\boldsymbol{\xi}_\kappa$ for discrete steps $\kappa \geq 0$ (note for the chain we use the forward time index κ instead of the backward time index k to give priority to the Markov chain properties usually defined in forward time, instead of completely forcing the backward time notion of SDP). The Markov chain transition probabilities are defined by

$$p(\boldsymbol{\xi}_\kappa, \boldsymbol{\eta} | \mathbf{u}_\kappa) = \text{Prob}[\boldsymbol{\xi}_{\kappa+1} = \boldsymbol{\eta} | \boldsymbol{\xi}_j, \mathbf{u}_j, j \leq \kappa]$$

for transitions of the Markov chain from stage $\boldsymbol{\xi}_\kappa$ to stage $\boldsymbol{\xi}_{\kappa+1} = \boldsymbol{\eta}$ under control policy $\mathbf{u}_\kappa$. These transition probabilities must satisfy the non-negativity ($p \geq 0$) and conservation ($\sum p = 1$) of probability properties of any proper probability law. Thus, the chain is defined on a finite state space $\mathcal{D}_\xi$ with discrete time parameter κ. Further, the Markov chain approximation increments $\Delta\boldsymbol{\xi}_\kappa \equiv \boldsymbol{\xi}_{\kappa+1} - \boldsymbol{\xi}_\kappa$, with $|\Delta\boldsymbol{\xi}_\kappa| = \mathcal{O}(h)$, must satisfy the *local consistency conditions*

$$\begin{aligned} E[\Delta\boldsymbol{\xi}_\kappa | \boldsymbol{\xi}_\kappa, \mathbf{u}_\kappa] &\equiv \sum_\eta (\boldsymbol{\eta} - \boldsymbol{\xi}_\kappa) \cdot p(\boldsymbol{\xi}_\kappa, \boldsymbol{\eta} | \mathbf{u}_\kappa) \\ &= \Delta t_\kappa \cdot [\mathbf{F}_\kappa(\boldsymbol{\xi}_\kappa, \mathbf{u}_\kappa) + o(1)] \\ &\text{and} \\ \text{Covar}[\Delta\boldsymbol{\xi}_\kappa | \boldsymbol{\xi}_\kappa, \mathbf{u}_\kappa] &\equiv \sum_\eta (\boldsymbol{\eta} - \boldsymbol{\xi}_\kappa - E[\Delta\boldsymbol{\xi}_\kappa])(\boldsymbol{\eta} - \boldsymbol{\xi}_\kappa - E[\Delta\boldsymbol{\xi}_\kappa])^T \\ &\cdot \quad p(\boldsymbol{\xi}_\kappa, \boldsymbol{\eta} | \mathbf{u}_\kappa) \\ &= \Delta t_\kappa \cdot \left[(G_\kappa G^T_\kappa)(\boldsymbol{\xi}_\kappa) + o(1) \right], \end{aligned} \tag{78}$$

as $h \longrightarrow 0^+$, for $\kappa = 0$ to $K-1$, consistent with the usual conditions for the first two infinitesimal moments of the stochastic diffusion approximation corresponding to the SDE (73). In (78), the conditioning on earlier states and controls $\{\boldsymbol{\xi}_j, \mathbf{u}_j, j < \kappa\}$ has been suppressed to simplify the notation, but is not meant to imply any assumption on earlier variables.

Here, Δt_κ is the local interpolation time increment, such that $\Delta t_\kappa \longrightarrow 0$ as $h \longrightarrow 0^+$, and such that the piecewise constant interpolated chain and control with continuous time parameter t are

$$\boldsymbol{\xi}(t) \equiv \boldsymbol{\xi}_\kappa \text{ and } \mathbf{u}(t) \equiv \mathbf{u}_\kappa \text{ for } t \text{ on } [t_\kappa, t_\kappa + \Delta t_\kappa),$$

which together with the local consistency conditions (78) are satisfied by the corresponding discrete time chain allow approximation of (73). For generality, $\Delta t_\kappa = \Delta t_\kappa(\boldsymbol{\xi}_\kappa, \mathbf{u}_\kappa)$ is permitted.

3. MCA Dynamic Programming Equation and Transition Probabilities Construction from Finite Differences

The MCA transition probabilities $p(\boldsymbol{\xi},\boldsymbol{\eta}|\mathbf{u})$ often are constructed from the finite difference in space of the parabolic, dynamic programming equation (77) (finite elements could be used as well). The usual mixed finite differences in space used in [76] are upwinded (forward and backward) for first order state derivatives,

$$\frac{\partial V}{\partial x_i}(\mathbf{x},t) \simeq \left\{ \begin{array}{ll} \frac{V(\mathbf{x}+h_i\mathbf{e}_i,t)-V(\mathbf{x},t)}{h_i}, & F_i(\mathbf{x},\mathbf{u}(\mathbf{x},t),t) \geq 0 \\ \frac{V(\mathbf{x},t)-V(\mathbf{x}-h_i\mathbf{e}_i,t)}{h_i}, & F_i(\mathbf{x},\mathbf{u}(\mathbf{x},t),t) < 0 \end{array} \right\}, \tag{79}$$

and central differences for the second order derivatives,

$$\frac{\partial^2 V}{\partial x_i^2}(\mathbf{x},t) \simeq \frac{V(\mathbf{x}+h_i\cdot\mathbf{e}_i,t)-2V(\mathbf{x},t)+V(\mathbf{x}-h_i\cdot\mathbf{e}_i,t)}{h_i^2}, \tag{80}$$

with $\mathbf{e}_i$ as the unit vector for the ith state component and $h_i = \mathcal{O}(h)$ as the size of the ith step. Here, we assume that the Gaussian noise is not correlated (i.e., we mean the coefficient is assumed to be diagonal), so the cross second derivatives are not needed. Also, the upwinding on first derivatives sacrifices accuracy (i.e., it has $\mathcal{O}(h_i)$ error rather than the smaller $\mathcal{O}(h_i^2)$ error of central differences as $h_i \longrightarrow 0^+$, so the larger error "numerically pollutes" the smaller) for potentially greater numerical stability in the case of convection dominated flows (i.e., the drift part $|F_i|/h_i$ is greater than the diffusion part $(GG^T)_{i,i}/h_i^2$). However, many refinements to fix up these problems are found in Kushner and Dupuis [76], such as higher order finite differences including those needed for cross second derivatives.

Upon substituting these finite differences, the dynamic programming equation (77) becomes

$$\begin{aligned} v^*(\mathbf{x},t_{\kappa-1}) &= \inf_{\mathbf{u}_{\kappa-1}} [p_\kappa(\mathbf{x},\mathbf{x},|\mathbf{u}_{\kappa-1})\cdot v^*(\mathbf{x},t_\kappa) \\ &+ \sum_{i=1}^n p_\kappa(\mathbf{x},\mathbf{x}+h_i\cdot\mathbf{e}_i|\mathbf{u}_{\kappa-1})\cdot v^*(\mathbf{x}+h_i\cdot\mathbf{e}_i,t_\kappa) \\ &+ \sum_{i=1}^n p_\kappa(\mathbf{x},\mathbf{x}-h_i\cdot\mathbf{e}_i|\mathbf{u}_{\kappa-1})\cdot v^*(\mathbf{x}-h_i\cdot\mathbf{e}_i,t_\kappa) \\ &+ \Delta t_{\kappa-1}\cdot C(\mathbf{x},\mathbf{u}_{\kappa-1},t_\kappa)], \end{aligned} \tag{81}$$

where the transition probabilities, now denoted by

$$p(\boldsymbol{\xi}_{\kappa-1},\boldsymbol{\xi}_\kappa|\mathbf{u}_{\kappa-1}) \simeq p_\kappa(\boldsymbol{\xi},\boldsymbol{\eta}|\mathbf{u}_{\kappa-1})$$

and activated by the control vector $\mathbf{u}_{\kappa-1}$ from $t_{\kappa-1}$ to t_κ, are given by

$$p_\kappa(\mathbf{x},\mathbf{x},|\mathbf{u}_{\kappa-1}) = 1-\Delta t_{\kappa-1}\cdot\sum_{j=1}^n\left[\frac{(GG^T)_{j,j,\kappa}}{h_j^2}+\frac{|F_{j,\kappa}|}{h_j}\right], \tag{82}$$

$$p_\kappa(\mathbf{x},\mathbf{x}+h_i\cdot\mathbf{e}_i|\mathbf{u}_{\kappa-1}) = \Delta t_{\kappa-1}\cdot\left[\frac{(GG^T)_{i,i,\kappa}}{2h_i^2}+\frac{[F_{i,\kappa}]_+}{h_i}\right], \tag{83}$$

$$p_\kappa(\mathbf{x},\mathbf{x}-h_i\cdot\mathbf{e}_i|\mathbf{u}_{\kappa-1}) = \Delta t_{\kappa-1}\cdot\left[\frac{(GG^T)_{i,i,\kappa}}{2h_i^2}+\frac{[F_{i,\kappa}]_-}{h_i}\right], \tag{84}$$

for sufficiently small $\Delta t_{\kappa-1}$ so $p_\kappa(\mathbf{x},\mathbf{x},|\mathbf{u}_{\kappa-1})$ in (82) is non-negative. Here $[F]_\pm \equiv \max[\pm F,0] \geq 0$ (i.e., $[F]_+ + [F]_- = |F|$ and $[F]_+ - [F]_- = F$). The drift component is $F_{i,\kappa} = F_i(\mathbf{x},\mathbf{u}_{\kappa-1},t_\kappa)$ and the diffusion coefficient is $G_{i,i,\kappa} = G_{i,i}(\mathbf{x},t_\kappa)$, assuming that only the diagonal part is needed. While the explicit time-dependence of the dynamical system coefficients are evaluated at the later time t_κ (this is not a critical requirement) as would be from the Backward Euler approximation, the control is evaluated at the earlier time $t_{\kappa-1}$. The procedure generates feedback control at the discrete level. This is because the control policy, when known, would be set at the beginning for the period in forward time, although this control policy is determined by the dynamic program in backward time sequence. Kushner and Dupuis [76] call this type of MCA an explicit method, but since, as a backward procedure, it is explicit in the costs v_κ^* while implicit in the control $\mathbf{u}_{\kappa-1}$, it is really quasi-explicit in the case of the simple approximations used here. However, a number of implicit methods are also presented by Kushner and Dupuis [76].

In (82,83,84), the really important benefit of upwinding is seen, in that upwinding ensures that the drift coefficient $F_{i,\kappa}$ only enters the transition probabilities as an absolute value, guaranteeing the non-negativity property of the non-self transition probabilities in (83,84), and (82) as well, provided the interpolation increment is sufficiently small. Note that the requirement that the self transition term (82) must be non-negative as a probability puts restrictions on the time step $\Delta t_{\kappa-1}$, i.e.,

$$\Delta t_{\kappa-1} \leq \frac{1}{\sum_{i=1}^{n}\left[\frac{(GG^T)_{i,i,\kappa}}{h_i^2}+\frac{|F_{i,\kappa}|}{h_i}\right]}, \tag{85}$$

which is something like the generalized Courant Friedrichs and Lewy (CFL) condition (42) for SDP. This is more readily seen in the form:

$$\sigma_{MCA} = \Delta t_{\kappa-1}\cdot\sum_{i=1}^{n}\left[\frac{(GG^T)_{i,i,\kappa}}{h_i^2}+\frac{|F_{i,\kappa}|}{h_i}\right] \leq 1, \tag{86}$$

which is more like the "$\mathcal{L}_1$" norm version of the "$\mathcal{L}_2$" norm version of the generalized CFL condition in (42). Thus the conditions for both methods require the same order of magnitude of the time step. The differences in quasi-norms is not significant given the approximations in both methods.

One can check that the transition probabilities do indeed satisfy the MCA local consistency conditions (78). For instance, the first infinitesimal moment of the MCA is calculated as follows,

$$\begin{aligned}
E[\Delta\boldsymbol{\xi}_\kappa|\boldsymbol{\xi}_\kappa=\mathbf{x}] &= p_\kappa(\mathbf{x},\mathbf{x}|\mathbf{u}_{\kappa-1})\cdot[0] \\
&+ \sum_{i=1}^{n} p_\kappa(\mathbf{x},\mathbf{x}+h_i\cdot\mathbf{e}_i|\mathbf{u}_{\kappa-1})\cdot[h_i\cdot\mathbf{e}_i] \\
&+ \sum_{i=1}^{n} p_\kappa(\mathbf{x},\mathbf{x}-h_i\cdot\mathbf{e}_i|\mathbf{u}_{\kappa-1})\cdot[-h_i\cdot\mathbf{e}_i] \\
&= \Delta t_{\kappa-1}\cdot\sum_{i=1}^{n}([F_{i,\kappa}]_+-[F_{i,\kappa}]_-)\cdot\mathbf{e}_i \\
&= \Delta t_{\kappa-1}\cdot\sum_{i=1}^{n}F_{i,\kappa}\cdot\mathbf{e}_i = \Delta t_{\kappa-1}\cdot\mathbf{F}(\mathbf{x},\mathbf{u}_{\kappa-1},t_\kappa),
\end{aligned}$$

demonstrating that the condition is satisfied for the first moment. The consistency condition can be similarly demonstrated for the second moment.

In the case of a stationary dynamic programming equation, i.e., the elliptic case, such as for an exit time problem, optimal stopping problem or infinite horizon, then the coefficients from the finite difference formula are renormalized to satisfy properties of transition probabilities. Renormalization is not essential for the non-stationary case described above. In the stationary case, renormalization is computationally useful in that it can help speed up the computation if wisely done. Typically, the stationary self-transition term $\widehat{p}_\kappa(\mathbf{x},\mathbf{x}|\mathbf{u})$ is set to zero and the central cost $v^*(\mathbf{x})$ is solved for as a function of the nontrivial transition costs, so the interpolation or iteration time $\widehat{\Delta t}$ is selected to be the resultant coefficient of the instantaneous cost $C(\mathbf{x},\mathbf{u})$. For example, suppose that the final horizon time $t_f=\tau_e$, the first exit time for the chain to reach to the boundary, then the non-stationary case equations (81,83,84) are reformulated (presumably before taking the infimum) for the stationary case as the iteration

$$\begin{aligned}
\widehat{v}^*_{\hat{\kappa}+1}(\mathbf{x}) &= \inf_{\widehat{\mathbf{u}}_{\hat{\kappa}+1}}\Bigg[\sum_{i=1}^{n}\widehat{p}(\mathbf{x},\mathbf{x}+h_i\cdot\mathbf{e}_i|\widehat{\mathbf{u}}_{\hat{\kappa}+1})\cdot\widehat{v}^*_{\hat{\kappa}}(\mathbf{x}+h_i\cdot\mathbf{e}_i) \\
&+ \sum_{i=1}^{n}\widehat{p}(\mathbf{x},\mathbf{x}-h_i\cdot\mathbf{e}_i|\widehat{\mathbf{u}}_{\hat{\kappa}+1})\cdot\widehat{v}^*_{\hat{\kappa}}(\mathbf{x}-h_i\cdot\mathbf{e}_i) \\
&+ \widehat{\Delta t}\cdot C(\mathbf{x},\widehat{\mathbf{u}}_{\hat{\kappa}+1})\Bigg].
\end{aligned} \tag{87}$$

A new iteration index $\hat{\kappa}$ replaces the former κ which loses its meaning a time index in the stationary problem. The transition probabilities are now denoted by

$$p(\boldsymbol{\xi}_{\hat{\kappa}+1}, \boldsymbol{\xi}_{\hat{\kappa}} | \widehat{\mathbf{u}}_{\hat{\kappa}+1}) \simeq \widehat{p}_{\hat{\kappa}}(\boldsymbol{\xi}_{\hat{\kappa}+1}, \boldsymbol{\xi}_{\hat{\kappa}} | \widehat{\mathbf{u}}_{\hat{\kappa}+1}),$$

conditioned by the control vector $\widehat{\mathbf{u}}_{\hat{\kappa}+1}$ from iterations $\hat{\kappa}$ to $\hat{\kappa}+1$. The condition probabilities are now given by

$$\begin{aligned} \widehat{p}(\mathbf{x}, \mathbf{x} | \widehat{\mathbf{u}}_{\hat{\kappa}+1}) &\equiv 0, \\ \widehat{p}(\mathbf{x}, \mathbf{x} + h_i \cdot \mathbf{e}_i | \widehat{\mathbf{u}}_{\hat{\kappa}+1}) &= \widehat{\Delta t} \cdot \left[\frac{(GG^T)_{i,i}}{2h_i^2} + \frac{[F_i]_+}{h_i} \right], \qquad (88) \\ \widehat{p}(\mathbf{x}, \mathbf{x} - h_i \cdot \mathbf{e}_i | \widehat{\mathbf{u}}_{\hat{\kappa}+1}) &= \widehat{\Delta t} \cdot \left[\frac{(GG^T)_{i,i}}{2h_i^2} + \frac{[F_i]_-}{h_i} \right]. \qquad (89) \end{aligned}$$

The starting interpolation time increment $\widehat{\Delta t}$ is given by the reciprocal of $[1 - p(\mathbf{x}, \mathbf{x}|\mathbf{u})]/\Delta t_{\kappa-1}$ from the non-stationary case as

$$\widehat{\Delta t} = \frac{1}{\sum_{j=1}^{n} \left[\frac{(GG^T)_{j,j}}{h_j^2} + \frac{|F_j|}{h_j} \right]}.$$

For this stationary case it is assumed that the SDE coefficients are autonomous, i.e., $F_i = F_i(\mathbf{x}, \mathbf{u})$ and $G_{i,i} = G_{i,i}(\mathbf{x})$, for consistency. The former non-stationary interpolation time increment $\Delta t_{\kappa-1}$ cancels out of the discrete time dynamic programming equation (77).

4. MCA Dynamic Programming Equation Solution

The dynamic programming equation (81) can be solved for $v^*_{\kappa-1}$ and simultaneously for the optimal control vector approximation for MCA as the infimum or minimum argument:

$$\begin{aligned} \mathbf{u}^*(\mathbf{x}, t_{\kappa-1}) &= \arg \inf_{\mathbf{u}_{\kappa-1}} [p_\kappa(\mathbf{x}, \mathbf{x}, |\mathbf{u}_{\kappa-1}) \cdot v^*(\mathbf{x}, t_\kappa) \qquad (90) \\ &+ \sum_{i=1}^{n} p_\kappa(\mathbf{x}, \mathbf{x} + h_i \cdot \mathbf{e}_i | \mathbf{u}_{\kappa-1}) \cdot v^*(\mathbf{x} + h_i \cdot \mathbf{e}_i, t_\kappa) \\ &+ \sum_{i=1}^{n} p_\kappa(\mathbf{x}, \mathbf{x} - h_i \cdot \mathbf{e}_i | \mathbf{u}_{\kappa-1}) \cdot v^*(\mathbf{x} - h_i \cdot \mathbf{e}_i, t_\kappa) \\ &+ \Delta t_{\kappa-1} \cdot C(\mathbf{x}, \mathbf{u}_{\kappa-1}, t_\kappa)]. \end{aligned}$$

Kushner and Dupuis [76] discuss many computational methods for solving problems like (81), such as Point-Jacobi, Gauss-Seidel, Successive-Over-Relaxation accelerations, Multigrid or Multilevel, and Domain Decomposition. They also discuss both the benefits and disadvantages of these methods. Please refer to their book for more information. The paper of Kushner and Jarvis [77] gives the recent state of MCA computations using parallel and vector supercomputers. Quadrat and coworkers [1] have used MCA in their expert system for solving stationary optimal control problems.

Jump diffusions or Poisson processes included with stochastic diffusions only requires a small amount of extra effort to calculate the modifications of the transition probabilities for MCA. The changes due diffusion and the jumps are separately accounted for since one is continuous and the other discontinuous.

Reflecting boundaries are handled for reflecting diffusions and reflecting jump diffusions by adding a border of extra discrete elements around the domain. The width of the border elements is $\mathcal{O}(h)$. Any chain that winds up in the border element if reflected back into the interior elements in a way consistent with the application and the stochastic noise, and locally consistent with the local direction of reflection. Non-normal reflections may be difficult to handle in multi-dimensions. The reflections are modeled by added auxiliary stochastic processes to the SDE (73) and these auxiliary stochastic processes are only activated when a chain hits the border element. This seems to be a proper way to handle the complexities of boundary conditions for stochastic processes, whether reflecting or otherwise. The recent paper of Kushner and Yang [78] gives a clear example of the MCA method with reflecting boundary conditions for telecommunication networks in a heavy traffic environment.

5. Similarities and Differences between MCA and SDP

The obvious difference between MCA and SDP, as presented here, is that MCA is is primarily applied to stationary problems, in spite for the dynamic model presentation here. The MCA method is extremely adaptable. Kushner and Dupuis [76] give variations for exit time problems, ergodic problems, jump diffusions, discounted costs, average costs, optimal filtering and many other applications. A major advantage or the method is that convergence proofs are facilitated by the probabilistic frame work, while they are generally difficult or not available for traditional numerical PDE methods presented for SDP here. However, the MCA method does introduce a indirect feature in the use of the Markov chain to solve a continuous time, continuous state optimal control problem. A principal similarity is that both MCA and SDP basically use Bellman's dynamic programming equations, although under quite different probabilistic and numerical interpretations. They are both variants of the stochastic dynamic programming algorithm. Although the control is usually the most important output for the control user, sometimes the expected optimal trajectories, i.e., using the optimal control, are desired to study the behavior of the dynamic system in the optimal state. There does not appear to be a formulation of MCA that covers the use of the forward equations for this purpose in Kushner and Dupuis [76], although simulations using random number generators are suggested.

V. RESEARCH DIRECTIONS

Future directions continue to be improvements in software and algorithms. There are always bigger or grander challenging problems to solve and any advantage is welcomed. Increasing the level of parallelism is desired for dealing with large scale computational and memory requirements. Algorithms that are able to parallelize dynamic programming problems over time such as time-clustering and that reduce the state space computation by targeting the neighborhood of the optimal trajectory. Also desired is the use of accurate upwinding schemes for more stability and the use of methods like multigrid which make convergence less sensitive to mesh size by using small and large mesh grids to filter out errors.

A future research direction is the direct and fair comparison of methods like SDP, DDP and MCA solving the same problem. It should be possible to determine what features are more effective and to determine which methods have robust convergence properties, but also which have faster rates of convergence under typical circumstances. Region restricting techniques, such as DDP effectively uses with its focus on the neighborhood of the optimal trajectory, will be important. Also, methods, like multigrid, which do not demand very fine grid structures will also be important for reducing the computational and memory requirements.

ACKNOWLEDGEMENT

The author thanks his collaborators Dr. S.-L. Chung, Dr. H. H. Xu, Prof. K. Naimipour, Dr. D. J. Jarvis, J. J. Westman, C. J. Pratico and M. S. Vetter for their contributions. He also thanks his hosts, Prof. H. J. Kushner of Brown University and C. A. Shoemaker of Cornell University, for their hospitality and insights about computational control when visiting for a sabbatical year. In addition, he thanks Prof. P. G. Dupuis for his helpful comments.

VI. REFERENCES

1. M. Akian, J. P. Chancelier and J. P. Quadrat, "Dynamic Programming Complexity and Applications," *Proceedings of 27th IEEE Conference on Decision and Control* **2**, pp. 1551-1558 (1988).

2. B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*, Prentice Hall, Englewood Cliffs, NJ, 1990.

3. L. Arnold, *Stochastic Equations: Theory and Applications*, Wiley, New York, NY, 1974.

4. U. Ascher, J. Christiansen and R. D. Russell, "A Collocation Solver for Mixed Order Systems of Boundary Value Problems," *Mathematics of Computation* **33**, pp. 659-679 (1979).

5. U. Ascher, J. Christiansen and R. D. Russell, "Collocation Software for Boundary-Value ODEs," *ACM Transactions on Mathematical Software* **7**, pp. 209-222 (1981).

6. M. Athans, D. Castanon, K. P. Dunn, C. S. Greene, W. H. Lee, N. R. Sandell, Jr., and A. S. Willsky, "The Stochastic Control of the F-8c Aircraft Using a Multiple Model Adaptive Control (MMAC) Method - Part I: Equilibrium Flight," *IEEE Transactions on Automatic Control* **AC-22**, pp. 768-780 (1977).

7. G. Bell, "Ultracomputers: A Teraflop Before Its Time," *Communications of ACM* **35** (8), pp. 26-47 (1992).

8. R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

9. R. E. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, NJ, 1961.

10. A. Brandt, "Multilevel Computations: Review and Recent Developments," *Multigrid Methods: Theory, Applications, and Supercomputing* (S. F. McCormick, Editor), Marcel Dekker, New York, pp.35-62, 1988.

11. W. L. Briggs, *Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

12. A. E. Bryson and Y. Ho, *Applied Optimal Control*, Ginn, Waltham, MA, 1969.

13. J. L. Calvet, J. Dantas de Melo and J. M. Garcia, "A Fast Parallel Dynamic Programming Algorithm for Optimal Control," *Proceedings of 1990 American Control Conference*, 3 pages (1990).

14. J. Casti, M. Richardson and R. Larson, "Dynamic Programming and Parallel Computers" *Journal of Optimization Theory and Applications* **12** (4), pp. 423-786 (1973).

15. H. Caffey, L.-Z. Liao and C. A. Shoemaker, "Parallel Processing of Large Scale Discrete–Time Unconstrained Differential Dynamic Programming," *Parallel Computing* **19**, pp. 1003-1017 (1993).

16. L.-C. Chang, C. A. Shoemaker, and P. L.-F. Liu, "Optimal Time–Varying Pumping Rates for Groundwater Remediation: Application of a Constrained Optimal Control Algorithm," *Water Resources Research* **28** (12), pp. 3157-3173 (1992).

17. S.-L. Chung and F. B. Hanson, "Parallel Optimizations for Computational Stochastic Dynamic Programming," *Proceedings of 1990 International Conference on Parallel Processing* **3**: *Algorithms and Applications* (P.-C. Yew, Editor), pp. 254-260 (1990).

18. S.-L. Chung and F. B. Hanson, "Optimization Techniques for Stochastic Dynamic Programming," *Proceedings of 29th IEEE Conference on Decision and Control* **4**, pp. 2450-2455 (1990).

19. S.-L. Chung, F. B. Hanson and H. Xu, "Supercomputer Optimizations for Stochastic Optimal Control Applications," *Proceedings of 4th NASA Workshop on Computational Control of Flexible Aerospace Systems, NASA Conf. Publ. 10065, Part 1* (L. W. Taylor, Editor), NASA Langley Research Center, pp. 57-70, March 1991.

20. S.-L. Chung, F. B. Hanson and H. H. Xu, "Parallel Stochastic Dynamic Programming: Finite Element Methods," *Linear Algebra and Applications* **172**, pp. 197-218 (1992).

21. M. G. Crandall and P. L. Lions, "Two Approximations of Solutions of Hamilton Jacobi Equations," *Mathematics of Computation* **43**, pp. 1-19 (1984).

22. M. G. Crandall, H. Ishii and P. L. Lions, "User's Guide to Viscosity Solutions of Second Order Partial Differential Equations," *Bulletin of the AMS* **27** (1), pp. 1-67 (1992).

23. M. L. Crow, D. J. Tylavsky and A. Bose,"Concurrent Processing in Power System Analysis," *Control and Dynamical Systems: Advances in Theory and Applications* **42** (C. T. Leondes, Editor), Academic Press, New York, NY, pp. 1-55, 1991.

24. T. Culver and C. A. Shoemaker "Dynamic Optimal Control for Groundwater Remediation with Flexible Management Periods," *Water Resources Research* **28** (3), pp. 629-641 (1992).

25. T. Culver and C. A. Shoemaker "Optimal Control for Groundwater Remediation by Differential Dynamic Programming with Quasi-Newton Approximations," *Water Resources Research* **29** (4), pp. 823-831 (1993).

26. J. Dantas de Melo, J. L. Calvet and J. M. Garcia, "Vectorization and Multitasking of Dynamic Programming in Control: Experiments on a Cray-2," *Parallel Computing* **13**, pp. 261-269 (1990).

27. P. Dyer and S. R. McReynolds, *The Computation and Theory of Optimal Control*, Academic Press, New York, NY, 1970.

28. N. H. Decker, "On the Parallel Efficiency of the Frederickson-McBryan Multigrid Algorithm," *ICASE Report No. 90-17*, NASA Langley Research Center, Hampton, VA, 8 pages, February 1990.

29. J. J. Dongarra, I. S. Duff, D. C. Sorensen and H. A. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991.

30. J. J. Dongarra, H. W. Meyer and E. Strohmaier, "TOP500 Supercomputer Sites," Computer Center, University of Mannheim, ftp://ftp.uni-mannheim.de/info/rumdoc/top500.ps.Z, 31 pages, November 1994.

31. J. Douglas Jr. and T. Dupont, "Galerkin Methods of Parabolic Equations," *SIAM J. Numerical Analysis* **7**, pp. 575-626 (1970).

32. J. Douglas Jr., "Time Step Procedures of Nonlinear Parabolic PDEs," *Mathematics of Finite Elements and Applications, MAFELAP* (J. Whiteman, Editor), Academic Press, London, pp. 289-304, 1979.

33. S. E. Dreyfus, *Dynamic Programming and the Calculus of Variations*, Academic Press, New York, NY, 1965.

34. W. R. Dyksen, E. N. Houstis, R. E. Lynch and J. R. Rice, "The Performance of the Collocation and Galerkin Methods with Hermite Bicubics," *SIAM J. Numerical Analysis* **21**, pp. 695-715 (1984).

35. S. Feiner and C. Beshers, "Visualizing n-Dimensional Virtual Worlds with n-Vision," *Computer Graphics: Special Issue on* $3D$ *Interactive Graphics* **24** (2), pp. 37-38 (1990).

36. A. Feldstein and K. W. Neves, "High Order Methods for State-Dependent Delay Differential Equations with Nonsmooth Solution," *SIAM J. Numerical Analysis* **21**, pp. 844-863, (1984).

37. J. E. Flaherty and R. E. O'Malley Jr., "Numerical Methods of Stiff Systems of Two-Point Boundary Value Problems," *SIAM J. Scientific and Statistical Computing* **5**, pp. 865-886, (1984).

38. W. H. Fleming and R. W. Rishel, *Deterministic and Stochastic Optimal Control*, Springer-Verlag, New York, NY, 1975.

39. J. J. Florentin, "Optimal Control of Systems with Generalized Poisson Inputs," *ASME Trans.* **85D** *(J. Basic Engr.* **2***)*, pp. 217-221 (1963).

40. P. O. Frederickson and O. A. McBryan, "Parallel Superconvergent Multigrid," *Multigrid Methods: Theory, Applications, and Supercomputing* (S. F. McCormick, Editor), Marcel Dekker, New York, pp. 195-210, 1988.

41. *Future Directions for Research in Symbolic Computations: Report of Workshop on Symbolic and Algebraic Computing* (A. Boyle and B. F. Caviness, Editors; A. C. Hearn, Workshop Chairperson), Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

42. *Future Directions in Control Theory: A Mathematical Perspective* (W. H. Fleming, Chairman), Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

43. I. I. Gihman and A. V. Skorohod, *Stochastic Differential Equations*, Springer-Verlag, New York, NY, 1972.

44. I. I. Gihman and A. V. Skorohod, *Controlled Stochastic Processes*, Springer-Verlag, New York, NY, 1979.

45. *Grand Challenges: High Performance Computing and Communications: The FY 1993 U.S. Research and Development Program*, Federal Coordinating Council for Science, Engineering, and Technology, Committee on Physical, Mathematical, and Engineering Sciences, c/o National Science Foundation (CISE), Washington, DC, 1993.

46. *High Performance Computing and Communications: Technology and Communications for the National Information Infrastructure: The 1995 Blue Book*, National Coordination Office for HPCC (http://www.hpcc.gov/); Federal Coordinating Council for Science, Engineering, and Technology; Committee on Physical, Mathematical, and Engineering Sciences; c/o National Science Foundation (CISE), Arlington, VA, 1995.

47. X.-H. Guan and P. B. Luh, "A New Parallel Algorithm for Optimal Control Problems of Interconnected Systems," *International Journal of Control* **56** (6), pp. 1275-1297 (1992).

48. W. Hackbusch, "Parabolic Multigrid Methods," *Computing Methods in Applied Sciences and Engineering* **6** (R. Glowinski and J.-L. Lions, Editors), North Holland, Amsterdam, pp. 189-197, 1984.

49. F. B. Hanson, "Analysis of a Singular Functional Differential Equation that Arises from Stochastic Modeling of Population Growth," *Journal of Mathematical Analysis and Applications* **96**, pp. 73-100 (1983).

50. F. B. Hanson, "Bioeconomic Model of the Lake Michigan Alewife Fishery," *Canadian Journal of Fisheries and Aquatic Sciences* **44** (Suppl. 2), pp. 298-305 (1987).

51. F. B. Hanson, "Vector Multiprocessor Implementation for Stochastic Dynamic Programming," *IEEE Distributed Processing Technical Committee Newsletter* **10**, pp. 44-48 (1988).

52. F. B. Hanson, "Computational Dynamic Programming for Stochastic Optimal Control on a Vector Multiprocessor," *Argonne National Laboratory Technical Memorandum* **ANL/MCS-TM-113**, 26 pages, June 1988.

53. F. B. Hanson, "Parallel Computation for Stochastic Dynamic Programming: Row versus column code orientation," *Proc. 1988 IEEE Conf. on Parallel Processing* **3**: *Algorithms and Applications*, pp. 117-119 (1988).

54. F. B. Hanson, "Parallel and vector computation for optimal control applications," *Proceedings of 3rd Annual Conference on Aerospace Computational Control, JPL Publication 85-45* **1** (D. E. Bernard and G. K. Man, Editors), Jet Propulsion Laboratory, Pasadena, pp. 184-197, December 15, 1989.

55. F. B. Hanson, "Stochastic Dynamic Programming: Advanced Computing Constructs," *Proceedings of 28th IEEE Conference on Decision and Control* **1**, pp. 901-903 (1989).

56. F. B. Hanson, "A Real Introduction to Supercomputing: A User Training Course," *Proceedings of Supercomputing '90*, pp. 376-385 (1990).

57. F. B. Hanson, "Computational Dynamic Programming on a Vector Multiprocessor," *IEEE Transactions on Automatic Control* **36** (4), pp. 507-511 (1991).

58. F. B. Hanson, D. J. Jarvis and H. H. Xu, "Applications of FORALL-Formed Computations in Large Scale Stochastic Dynamic Programming," *Proceedings of Scalable High Performance Computing Conference: SHPCC-92*, IEEE Computer Society, pp. 182-185 (1992).

59. F. B. Hanson and K. Naimipour, "Convergence of Numerical Method for Multistate Stochastic Dynamic Programming," *Proceedings of International Federation of Automatic Control 12th World Congress* **9**, pp. 501-504 (1993).

60. F. B. Hanson, C. J. Pratico, M. S. Vetter, and H. H. Xu, "Multidimensional Visualization Applied to Renewable Resource Management," *Proceedings of Sixth SIAM Conference on Parallel Processing for Scientific Computing* **2**, pp. 1033-1036 (1993).

61. F. B. Hanson and D. Ryan, "Optimal harvesting in a Stochastic Environment with Density Dependent Jumps," *Mathematical Models of Renewable Resources* **3** (R. Lamberson, Editor), Association of Resource Modelers, Arcata, Cal., pp. 117-123, 1984.

62. F. B. Hanson and D. Ryan, "Optimal Harvesting with Density Dependent Random Effects," *Natural Resource Modeling* **2**, pp. 439-455 (1987).

63. F. B. Hanson and H. C. Tuckwell, "Logistic Growth with Random Density Independent Disasters," *Theoretical Population Biology* **19**, pp. 1-18 (1981).

64. F. B. Hanson and J. J. Westman, " Computational Stochastic Dynamic Programming Problems: Groundwater Quality Remediation," *Proceedings of 33rd IEEE Conference on Decision and Control* **1**, 6 pages (1994).

65. G. Horton and S. Vandewalle, "A Space–Time Multigrid Method for Parabolic PDEs," *SIAM J. Scientific Computing*, 17 pages, 1995.

66. D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*, American Elsevier, New York, NY, 1970.

67. A. H. Jazwinski, *Stochastic Processes and Filtering Theory*, Academic Press, New York, NY, 1970.

68. Michael B. Johnson, *The Application Communication Library*, MIT Media Laboratory, Cambridge, MA, 1992.

69. S. L. Johnsson and K. K. Mathur, "The Finite Element Method on a Data Parallel Computing," *International Journal of High Speed Computing* **1**, pp. 29-44 (1989).

70. L.D. Jones, R. Willis and W. W.-G. Yeh, "Optimal Control of Nonlinear Groundwater Hydraulics Using Differential Dynamic Programming," *Water Resources Research* **23**, pp. 2097-2106 (1987).

71. J. F. Kerrigan, *Migrating to Fortran 90*, O'Reilly & Associates, Sebastopol, CA, 1993.

72. C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele Jr. and M. E. Zosel, *The High Performance Fortran Handbook*, MIT Press, Cambridge, MA, 1994.

73. H. J. Kushner, "A Survey of Some Applications of Probability and Stochastic Control Theory to Finite Difference Methods for Degenerate Elliptic and Parabolic Equations," *SIAM Review* **18**, pp. 545-577, (1976).

74. H. J. Kushner, *Probability Methods for Approximations in Stochastic Control and for Elliptic Equations*, Academic Press, New York, NY, 1977.

75. H. J. Kushner, "Numerical Methods for Stochastic Control in Continuous Time," *SIAM J. Control and Optimizations* **28**, pp. 999-1048 (1990).

76. H. J. Kushner and P. G. Dupuis, *Numerical Methods for Stochastic Control in Continuous Time*, Springer-Verlag, New York, NY, 1992.

77. H. J. Kushner and D. J. Jarvis, "Large-Scale Computations for High Dimension Control Systems," *Proceedings of 33rd IEEE Conference on Decision and Control* **1**, 6 pages (1994).

78. H. J. Kushner and J.-C. Yang, "A Numerical Method for Controlled Routing in Large Trunk Line Networks via Stochastic Control Theory," *ORSA Journal on Computing* **6** (3), pp. 300-316 (1994).

79. L.-Z. Liao and C. A. Shoemaker, "Convergence in Unconstrained Discrete-Time Differential Dynamic Programming," *IEEE Transactions on Automatic Control* **36** (6), pp. 692-706 (1991).

80. R. E. Larson, "Dynamic Programming with Reduced Computational Requirements," *IEEE Transactions on Automatic Control* **AC-14**, pp. 135-143 (1965).

81. R. E. Larson, "A Survey of Dynamic Programming Computational Procedures," *IEEE Transactions on Automatic Control* **AC-16**, pp. 767-774 (1967).

82. R. E. Larson, *State Increment Dynamic Programming*, American Elsevier, New York, NY, 1968.

83. R. E. Larson and E. Tse, "Parallel Processing Algorithms for the Optimal Control of Nonlinear Systems," *IEEE Transactions on Automatic Control* **AC-22**, pp. 777-786 (1973).

84. John M. Levesque and J. W. Williamson, *A Guidebook to FORTRAN on Supercomputers*, Academic Press, NY, 1988.

85. D. Ludwig, "Optimal Harvesting of a Randomly Fluctuating Resource. I: Application of Perturbation Methods," *SIAM J. Applied Mathematics* **37**, pp. 185-205 (1979).

86. D. Ludwig and J. M. Varah, "Optimal Harvesting of a Randomly Fluctuating Resource. II: Numerical Methods and Results," *SIAM J. Applied Mathematics* **37**, pp. 185-205 (1979).

87. R. Luus, "Optimal Control by Dynamic Programming Using Systematic Reduction in Grid Size," *International Journal of Control* **51** (5), pp. 995-1013 (1990).

88. R. Luus, "Application of Dynamic Programming to Higher-Dimensional Non-Linear Optimal Control Problems," *International Journal of Control* **52** (1), pp. 239-250 (1990).

89. S. F. McCormack, *Multigrid Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

90. K. K. Mathur and S. L. Johnsson, "Data Structures and Algorithms for the Finite Element Method on a Data Parallel Supercomputer," *International Journal on Numerical Methods in Engineering*, (1989).

91. D. Q. Mayne, "A Second–Order Gradient Method for Determining Optimal Control of Non–Linear Discrete Time Systems," *International Journal of Control* **3**, pp. 85-95 (1966).

92. D. Q. Mayne, "Differential Dynamic Programming — A Unified Approach to the Optimization of Dynamical Systems," *Control and Dynamical Systems: Advances in Theory and Applications* **10** (C. T. Leondes, Editor), Academic Press, New York, NY, pp. 179-254, 1973.

93. A. R. Mitchell and R. Wait, *The Finite Element Method in Partial Differential Equations*, Wiley, New York, NY, 1977.

94. I. H. Mufti, *Computational Methods in Optimal Control Problems, Lecture Notes in Operations Research and Mathematical Systems* **27**, Springer-Verlag, Berlin, Germany, 1970.

95. D. M. Murray and S. J. Yakowitz, "Differential Dynamic Programming and Newton's Method for Discrete Optimal Control Problems," *Journal of Optimization and Applications* **43**, pp. 395-414 (1984).

96. K. Naimipour and F. B. Hanson, "Convergence of a Numerical Method for the Bellman Equation of Stochastic Optimal Control with Quadratic Costs and Constrained Control," *Dynamics and Control* **3** (3), pp.237-259 (1993).

97. *A National Computing Initiative: The Agenda for Leadership* (H. J. Revecheé, Chair), Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

98. Mark H. Overmars, "A Graphical User Interface Toolkit for Silicon Graphics Workstations," Dept. Computer Science, Utrecht University, P.O. Box 80.089 TB, Utrecht, The Netherlands.

99. E. Polak, *Computational Methods in Optimization*, Academic Press, New York, NY, 1971.

100. E. Polak, "An Historical Survey of Computational Methods in Optimal Control," *SIAM Review* **15**, pp. 553-584 (1973).

101. C. D. Polychronopoulos, *Parallel Programming and Compilers*, Kluwer Academic, Boston, MA, 1988.

102. C. J. Pratico, F. B. Hanson, H. H. Xu, D. J. Jarvis and M. S. Vetter, "Visualization for the Management of Renewable Resources in an Uncertain Environment," *Proceedings of Supercomputing '92*, pp. 258-266, color plates p. 843 (1992).

103. R. G. Rabbani and J. W. Warner, "Shortcomings of Existing Finite Element Formulations for Subsurface Water Pollution Modeling and Its Rectification: One-Dimensional Case," *SIAM J. Applied Mathematics* **54** (3), pp. 660-673 (1994).

104. D. Ryan and F. B. Hanson, "Optimal Harvesting with Exponential Growth in an Environment with Random Disasters and Bonanzas," *Mathematical Biosciences* **74**, pp. 37-57 (1985).

105. D. Ryan and F. B. Hanson, "Optimal Harvesting of a Logistic Population in an Environment with Stochastic Jumps," *Journal of Mathematical Biology* **24**, pp. 259-277 (1986).

106. Z. Schuss, *Theory and Applications of Stochastic Differential Equations*, Wiley, New York, NY, 1980.

107. D. L. Snyder and M. I. Miller, *Random Point Processes in Time and Space*, Springer-Verlag, New York, NY, 1991

108. P. E. Souganidis, "Approximation Schemes for Viscosity Solutions of Hamilton-Jacobi Equations," *Journal of Differential Equations* **59**, pp. 1-43 (1985).

109. R. F. Stengel, *Stochastic Optimal Control*, Wiley, New York, NY, 1986.

110. G. Strang and G. J. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

111. *System Software and Tools for High Performance Computing Environments* (P. Messina and T. Sterling, Editors), Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

112. Thinking Machines Corporation, *Connection Machine Model CM-5 Technical Summary*, Cambridge, MA, November 1992.

113. S. Weerawarana and P. S. Wang, "A Portable Code Generator for Parallel Cray Fortran," *Kent State University Preprint*, 23 pages, June 1989.

114. W. M. Wonham, "Random Differential Equations in Control Theory," *Probabilistic Methods in Applied Mathematics* **2**, Academic Press, New York, pp. 131-212, 1970.

115. P. C. Xirouchakis and P. Y.Wang, "Finite Element Computing on Parallel Architectures," *Control and Dynamical Systems: Advances in Theory and Applications* **58** (C. T. Leondes, Editor), Academic Press, New York, NY, pp. 233-260, 1993.

116. H. H. Xu, F. Hanson and S. L. Chung, "Optimal Data Parallel Methods for the Stochastic Dynamical Programming," *Proceedings of 1991 International Conference on Parallel Processing* **3**: *Algorithms & Applications* (K. So, Editor), pp. 142-146, 1991.

117. H. H. Xu, F. Hanson and S. L. Chung, "Parallel Solutions of Dimensionality Problems for the Stochastic Dynamical Programming," *Proceedings of 30th IEEE Conference on Decision and Control* **2**, pp. 1717-1723 (1991).

118. H. H. Xu, D. J. Jarvis and F. B. Hanson, "Parallel Data Vault Methods for Larger Scale Stochastic Dynamic Programming," *Proceedings of 1992 American Control Conference* **1**, pp. 142-146 (1992).

119. S. Yakowitz, "Dynamic Programming Applications in Water Resources," *Water Resources Research* **18**, pp. 673-696 (1982).

120. S. Yakowitz, "Algorithms and Computational Techniques in Differential Dynamic Programming," *Control and Dynamical Systems: Advances in Theory and Applications* **31** (C. T. Leondes, Editor), Academic Press, New York, NY, pp. 75-91, 1989.

121. S. Yakowitz and B. Rutherford, "Computational Aspects of Discrete-Time Control," *Applied Mathematics and Computation* **15**, pp. 29-45 (1984).