"MINIMUM COST DIFFERENTIATION METHODS AND THEIR USES IN UNDERSTANDING, DESIGNING AND OPTIMIZING COMPLEX SYSTEMS"

by Paul J. Werbos, Quality Assurance Division, 3/20/82

CONTENTS

# "MINIMUM COST DIFFERENTIATION METHODS AND THEIR USES IN UNDERSTANDING, DESIGNING AND OPTIMIZING COMPLEX SYSTEMS"

by Paul J. Werbos, Quality Assurance Division, 3/20/82

## ABSTRACT

This paper derives low-cost methods for computing "ordered derivatives," which are required in numerous applications. Applications to statistical estimation, optimization and equation solving are pointed out. Applications to the sensitivity analysis of nonlinear dynamic models are discussed in detail. To illustrate the possibilities for adapting these methods to deal with large "network" systems, it is shown how to compute the derivatives required by a previously proposed method for decision-making over time under uncertainty. This example leads to an approach to the design of generalized, adaptive decision-making systems; applications to artificial intelligence and model evaluation concepts are discussed, along with structural analogies to the human brain.

## (I) INTRODUCTION

It is well known that the derivative and partial derivative of elementary calculus have applications throughout the natural and social sciences. This paper is concerned with a related mathematical concept, the "ordered derivative," which also has many applications, but which goes by different names in different fields of study. The purpose of this paper is two-fold: to discuss the applications of ordered derivatives, and to show how they may be calculated at minimum cost either directly or as part of a "network design." Roughly speaking, it will be shown that all of the first or second derivatives needed for most applications can be obtained exactly for a cost comparable to that of exercising the original model or system itself.

Section (II), below, will explain what an ordered derivative is, and summarize the advantages of the alternative methods to be described. It will also mention applications to statistical estimation, deterministic optimization and certain equation-solving techniques, of particular importance to large systems. Related methods and literature will be mentioned. Section (III) will discuss "sensitivity analysis," which attempts to help one understand complex systems such as energy models by pinpointing the key inputs and providing related information. Section (IV) will list the equations for the methods discussed in section (II), for the case of a simple nonlinear dynamic system. Section (V) will derive and generalize the "backwards" methods of section (IV), by using a concept called "dynamic feedback." Section (VI) will use perturbation approaches taken from physics to derive the remaining methods.

Section (VII) will illustrate how these methods may be adapted to handle very difficult problems, by spelling out the calculations required to implement a method for dynamic optimization under uncertainty, while taking full advantage of the power of "parallel" or "vector" computers. It will also show that a combination of this and related work may ultimately lead to a generalized, adaptive artificial intelligence, without some of the limitations of those now being developed; structural analogies between this design and the human brain will be discussed. In the mathematical framework provided by this section, model development may be analyzed as one of several analysis activities required to support rational decision making.

## (II) OVERVIEW OF DIFFERENTIATION METHODS TO BE GIVEN

Figure 1 shows a simple example of the kind of "derivative" we are trying to compute. Suppose that we have a nonlinear system, with a vector $\underline{x}$ of N endogenous variables and a vector $\underline{u}$ of exogenous variables. Suppose that the system is governed by the equation shown in Figure 1. The cost of simulating the model over the whole time range is mNT, because in each of the T time periods we compute a forecast for each of the N variables in $\underline{x}$, and each such forecast involves m terms. Please note that N is often much larger than m. Given a small change in the variable $x_i$ in time period 0, we want to know how large the resulting change in $x_i$ is in the final time period T.

$\underline{x}(t+1) = \underline{f}(\underline{x}(t), \underline{u}(t))$

- N components of $\underline{x}$
- m terms per equation $f_i$
- T time periods (t = 0 to T-1)
- cost of simulation = mNT
- not a "simultaneous" (implicit) model

$$\frac{\partial^+ x_i(T)}{\partial x_j(0)}$$

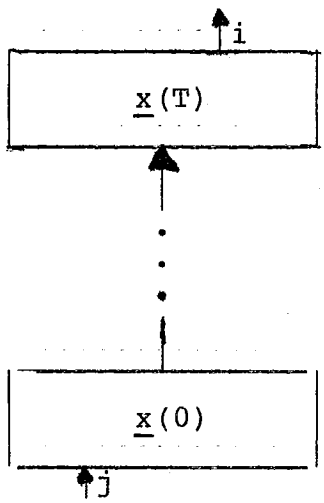



Figure 1: A Simple Example

The change in $x_i(T)$ per change in $x_i(0)$, holding the rest of $\underline{x}(0)$ constant, is a fundamental quantity of the sytem. It goes by many different names. In modelling, it is often called a "sensitivity coefficent." In economics, it is traditionally called an "impact multiplier." Electrical engineers often call it a "transient response," or "constrained derivative." Nuclear engineers sometimes use the term "adjoint." Here we will call it an "ordered derivative," using the notation shown in Figure 1, for two reasons: (1) the notation is somewhat more explicit than what is usually used; and (2) the concept of ordered derivative is somewhat more general rigorous, as will be seen. Some of the methods to be discussed below were published independently by control engineers[1] and by nuclear engineers[2] at about the same time (mid 1970's) as the ordered derivative concept was developed, and related methods for linear models were available even earlier. For differential equation systems (which this paper does not address) the nuclear engineering literature is perhaps the most extensive at present[3].

Well-known applications which require the use of such first-order derivatives are sensitivity analysis, maximization of a system result (i.e., "deterministic optimization"), and statistical estimation. In the last two cases, one actually is concerned with the derivative of a function of $\underline{x}(T)$ or of $\underline{x}(t \leq T)$ rather than the derivatives of $x_i(T)$ for some j, but it is easy to make this extension of the methods; for example, the function to be differentiated or a running total for it may be added to the list of system variables.

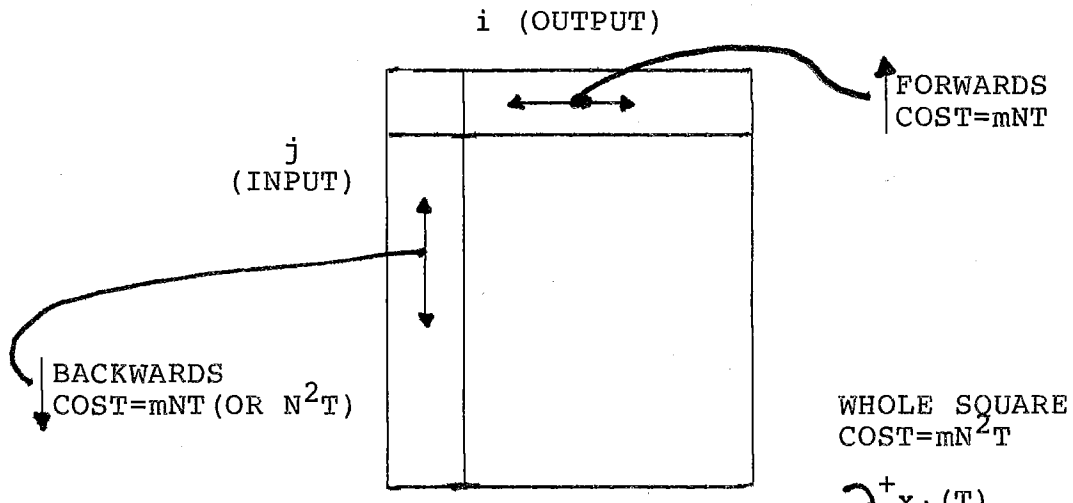



Figure 2: Two Ways To Obtain $\frac{\partial^+ x_i(T)}{\partial x_j(0)}$

Figure 2 describes two methods for computing ordered derivatives exactly in the example above. The corresponding equations are:

$$\uparrow: \quad \underline{z}(t) = \frac{\partial^+ \underline{x}(t)}{\partial x_j(0)}; \quad \underline{z}(t+1) = F(t)\underline{z}(t)$$

$$\downarrow: \quad \underline{z}'(t) = \frac{\partial^+ x_i(T)}{\partial \underline{x}(t+1)}; \quad \underline{z}'(t) = F^T(t)\underline{z}'(t+1), \text{ (or transpose)} \tag{2.1}$$

where F(t) is the matrix of derivatives of $\underline{f}(\underline{x}(t))$.
The large square in Figure 2 represents the entire matrix of ordered derivatives of all $x_i(T)$ with respect to all $x_j(0)$. The conventional or "forwards" method (indicated by an arrow pointing upwards) is based on perturbing one of the initials values $x_j(1)$, and observing the impact on all the final results, i.e., on the vector $\underline{x}(T)$. Each time we apply this method, we perturb only one of the initial values; thus we obtain only one row of the matrix of ordered derivatives, as shown in Figure 2. This costs us mNT calculations, as shown. Often the initial value $x_j(0)$ is actually changed, and the

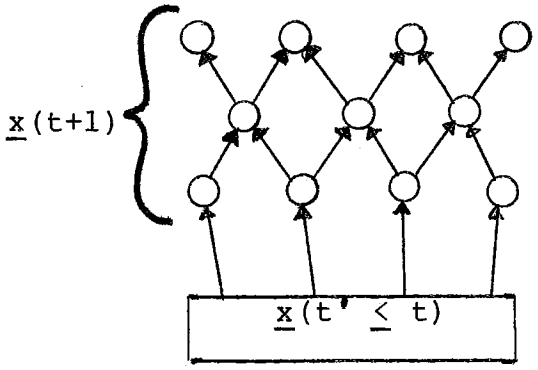
model resimulated. (This costs mNT operations, as did the original run of the model.) However, this leads to problems with the numerical accuracy of the results, because it requires that one compute each derivative by subtracting two numbers very close to each other in size. The forwards closed-form Jacobian formula, shown at the bottom of Figure 2, has the same cost but is more accurate.

The backwards method, shown with a downwards pointing arrow in Figure 2, computes an entire column of the matrix, using only mNT calculations. In control engineering, this sort of method has been used and related to the concept of "constrained derivatives," but has not been applied more generally[1].

The key point about these methods is that the forwards method is often used when the backwards method would be more appropriate. This can multiply costs (by a factor of N) to the point where it becomes infeasible to do what one wants to do. For example, it has long been known that economic data, like engineering measurements, are fraught with many errors, and that these errors invalidate conventional estimation methods. Statisticians[4,5] observed years ago that white noise converts a simple econometric model (like our example, but linear) into a "vector mixed autoregrassive moving average process." In other words, one can account for such errors in data by estimating the corresponding vector ARMA process. However, because of the sheer cost of such estimation, it has rarely been done in economics. Instead, an approximation suggested by Hibbs has become popular of late: a conventional model is estimated by regression, and then simple unvariate ARMA ("Box-Jenkins"[5]) modeling is used on the residuals, and the process may be iterated. Yet in statistical estimation, one only needs a single column of the derivative matrix (i.e., the derivatives of L), not the whole matrix; using the backwards method, one can compute all the derivatives needed in an iteration at the cost of only mNT, which is what it takes to exercise the model. This method was applied to vector ARMA estimation in the early 1970's, and inserted into a user-oriented software package at MIT[6] (TSP), but has yet to receive wide application in economics. It now appears that vector ARMA estimation (and thus Kalman filtering estimation, which is formally equivalent to it) may have less value in social science than other more robust methods based on a generalization of Hartley's simulation path approach[6,7,8]; however, those methods, too, require a set of derivatives, as part of minimizing a complicated loss function.

Likewise, in sensitivity analysis, a user often wants to know the sensitivity of a few key results to all the initial values, or to be sure he knows the largest of these sensitivity coefficients. Again, only a few columns of the matrix are required; it is wasteful to pay for the whole matrix.

With large models or network systems, N may range from the hundreds to the millions or more. Thus cutting the cost of computing derivatives by a factor of N is often crucial to feasibility. One may be sure that the cost of exercising the system (mNT) is affordable, or the system would be of no interest; more than this, by a multiple of N, may be unacceptably expensive.




"CHAIN RULE" (DYNAMIC FEEDBACK):

$$\frac{\partial^+ x_i}{\partial x_j} = \sum_{k=j+1}^{i} \frac{\partial^+ x_i}{\partial x_k} \cdot \frac{\partial f_k}{\partial x_j} \qquad i>j$$

CONVENTIONAL PERTURBATION:

$$\frac{\partial^+ x_i}{\partial x_j} = \sum_{k=j}^{i-1} \frac{\partial f_i}{\partial x_k} \cdot \frac{\partial^+ x_k}{\partial x_j} \qquad i>j$$

Figure 3: A More General Example: $\underline{x}(t+1) = f(\underline{x}(\text{all } t' \leq t+1), \underline{u}(\text{all } t'))$

The principle of "dynamic feedback" permits one to deal with more general, "network" models such as the one shown in Figure 3. Multisector models, for example, are usually best represented as a network. Because the proof of the "chain rule" for ordered derivatives[6] is not generally available, it is reproduced in Appendix A.

In Figure 3, the endogenous variables may appear with any nonnegative lag, including zero. However, we still assume here that the model has been reduced to "explicit" form. (In economics, one would call this a recursive model; in mathematics, one calls it a nonrecursive system.) We assume that the functions $f_i$, which make up $\underline{f}$, can be ordered in such a way that we can use them one by one to calculate the vector $\underline{x}(t+1)$. Actually, one can apply the methods given in this paper to simultaneous equation models as well, by using substitutions to be described in a forthcoming report from EIA[9].

Figure 3 illustrates an example where $\underline{x}(t+1)$ has eleven components, each represented by a circle; the arrows flowing into a circle represent inputs required to compute that component of $\underline{x}$.

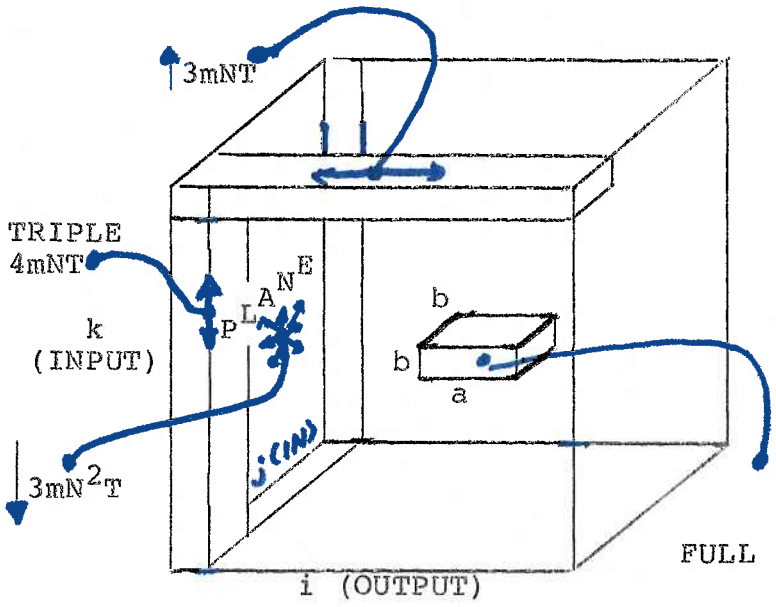
The forwards and backwards methods are generalized as shown in Figure 3. The subscripts here refer to an ordered index of all time/variable-number combinations; the formulas are given in more conventional form in the main paper. The key thing to note is that there are only m calculations per time/variable combination. Thus we still only need to make nMT calculations to get a complete row or column of ordered derivatives, as in our earlier example. This has

not previously been published. With conventional matrix methods for constrained derivatives, based on our earlier example, one would have to use N by N matrices f´, which would not usually be sparse; thus the generalization here makes it feasible to differentiate large network systems which would have been too expensive to differentiate with conventional methods.

The methods shown in Figure 3 remain efficient even if one uses "parallel" computers. Parallel computers - based on many processors operating in parallel rather than one CPU - are becoming increasingly common[10]. With a conventional computer, it would take roughly 11 calculation times to compute $\underline{x}(t+1)$ in our example (1 for each component of $\underline{x}$). With a parallel computer, it need only take 3: in the first period, 4 processors would calculate the lower tier in parallel, since none of the 4 lower components depends on the others; in the second period, the middle tier would be calculated; etc. The backwards method shown here allows similar economies: one can calculate ordered derivatives of a model result with respect to the top tier in the first period of calculation, then to the middle tier, and then to the bottom tier. The forwards method is similiar.

Large scale models or systems typically can be represented as relatively sparse networks, as in this example. Actual physical networks, made up of units operating in parallel, have a similar structure. To optimize such a system (except in unusual special cases) it is essential to know the derivatives of the desired performance measure with respect to all parameters in the system; for this to be feasible, it is essential to use a method such as the generalized backwards method which does not multiply the cost of getting the derivatives far beyond the cost of exercising the system. This reasoning also applies to the problem of minimizing or maximizing a complicated function, where most of the computing time is usually taken up with calculating derivatives[11]; it also applies to those methods for solving large systems of equations which require less than a full matrix of derivatives[12].

This overview has discussed derivatives with respect to initial values of the variables only; however, section (IV) will consider parameters, and the case of exogenous variables is a trivial extension of the endogenous variable case[9]. To avoid making a complicated discussion even more complicated, section (IV) will only mention our earlier example when discussing second derivatives; however, it is trivial to substitute the general formulas in Figure 3 for those in Figure 2, whenever they apply in the second derivative calculation, to arrive at more general methods. Section (VII) on stochastic optimization will provide a partial example of these possibilities.




CUBE AT LEFT REPRESENTS:

$$\frac{\partial^2 x_i(T)}{\partial x_j(0)\partial x_k(0)}$$

SECTION IV ALSO SPELLS OUT:

$$\frac{\partial^2 x_i(T)}{\partial \theta \partial x_j(0)}$$

and linear combos, etc.

FULL CUBE COSTS $3mN^3T$

Figure 4: Costs of Obtaining Various Sets of Second Derivatives

Figure 4 provides a summary of the properties of the four variable-variable second derivative calculation methods provided in section (IV). The set of ordered derivatives of $x_i(T)$ to $x_j(0)$ and $x_k(0)$ forms an N by N by N cube, as shown; each method computes a subset of the cube, at approximate costs shown. Again, in practice, the key point is to compute only the subset required, and not pay for the entire cube. The five methods for computing variable-parameter second derivatives offer the same subsets (except that an upwards column and a row pointing backwards count as two separate cases) for the same rough costs.

Section (III) will show that variable-parameter second derivatives provide meaningful information about a model, essentially equivalent to what MIT provides for linear systems[13,14] by looking at changes in eigenvalues. In effect, they tell us, for a change in a parameter of the system, how its dominant dynamics (revealed in the matrix of ordered derivatives to variables) change.

Among the possible applications is the use of Newton's method in estimation and optimization. It is straightforward to use the full backwards approach here for parameter-parameter derivatives; this allows computation of all the second derivatives one needs in order to use Newton's method, for a rough cost of only $3mN^2T$, about the same as what people have paid to get only first derivatives when using forwards methods.

## (III) SENSITIVITY ANALYSIS METHODS FOR DYNAMIC MODELS

The purpose of sensitivity analysis is to help one understand what inputs or model features really drive the results which come out of using a model. This has several possible applications:

- o It can help the consumer of model results to better interpret these results.
- o It can help the model user to better understand and use the model.
- o It can help the model developer or evaluator to focus effort on the most important assumptions of the model.

Comprehensive sensitivity analysis has at times pinpointed crucial inputs which had received serious attention beforehand either by modellers or by evaluators[9].

Because there is no limit to the number of ways in which a person can "understand" a model, the field of sensitivity analysis is open-ended. This paper has a more narrow focus in this section: to show how certain kinds of information now being provided for linear models in the user-oriented conversational software pacage Troll[14] can also be provided for nonlinear models, at acceptable cost and without approximation.

As part of their work under EIA contract, the developers of Troll have developed a linear sensitivity analysis package[13] which addresses the following questions:

- o Which inputs or assumptions have the greatest impact on the outputs? Is the impact positive or negative?
- o What patterns tend to dominate the system (i.e. the model variables) when the model is run out for several time periods?
- o Do the dominant patterns tend to oscillate or grow in intensity, and if so, with what rate and frequency?
- o Which inputs or assumptions have the greatest impact on the dominant patterns and on their rates of growth or oscillation?

The first of these questions can be answered by looking at the matrix of ordered derivatives, described in section (II). But with medium to large nonlinear models, it is expensive to calculate the entire matrix. Since our concern is usually to pinpoint those inputs, out of all the inputs, which most affect a few key results of interest, the "backwards" methods can provide the needed information at acceptable cost. As in Troll, we can convert these derivatives to "elasticities" by performing simple multiplications, in order to make the results more intelligible to economists.

The second and third of these questions are easily answered, for linear systems. It is well known in mathematics that linear systems come to be dominated by certain "characteristics vactors" or "eigenvectors," which can be calculated; their rates of growth and oscillation are given in their "eigenvalues." Roughly speaking, this says that all systems have certain "resonant frequencies" and patterns of fluctuation and growth which occur at those frequencies. When the initial state of the system is changed, the output at later times tends to change in proportion to the fastest growing eigenvector, regardless of which variables are changed.

With nonlinear systems, the situation is more complex. There are no eigenvectors. However, the matrix of ordered derivatives itself should show whether there exist dominant patterns, which lead to certain kinds of final impact (at time T) regardless of what variables are changed initially; if such patterns do exist, the rows of the matrix should all be close to proportional to each other. This implies that the columns should be proportional to each other as well. If several different results of interest are dominated by the same inputs, in the same proportions, as revealed by backwards sensitivity analysis, then there is some evidence that there exist dominant patterns in the nonlinear system. One run of the forwards sensitivity analysis, based on perturbing some inportant input, will indicate what the dominant pattern is; the behavior of the impact vector ("$\underline{z}$" in Figure 2) over time indicates what the growth and oscillation of the dominant pattern looks like, at least in the later periods.

Finally, to see how the dominant patterns of the system change, one can simply look at how the matrix of ordered derivatives changes when certain inputs are changed. This requires the calculation of selected ordered second derivatives. For example, the responsiveness of a result R to an initial value, x, is an ordered derivative; the impact of a parameter a in changing the responsiveness of the system to x is an ordered second derivative.

Ideally, a sytem like Troll should have all the methods of section (IV) available, and should pick the best method to use automatically on the basis of costs calculated for specific tasks. Eventually, a more complete sensitivity analysis package ahould also contain tools which address:

- o the degree of coupling between different subsystems
- o global sinks and sources and other topological properties
- o ergodic properties
- o changes in eigenvalues with respect to inputs

The methods of the following section are paricularly important for models which, like those of Jay Forrester[15], use small time intervals and recursive equations rather than simultaneous equations; however, as noted in section (II), they can be applied to both types of model.

## (IV) SUMMARY OF DIFFERENTIATION METHODS

For purposes of this summary, assume a nonlinear model in reduced form:

$$\underline{z}(t+1) = \underline{f}((\underline{z}(t), \underline{u}(t)),$$

which may be linearized about a solution trajectory to yield:

$$\underline{x}(t+1) = F(t)\underline{x}(t) + B(t)\underline{u}(t),$$

where F(t) is the Jacobian of $\underline{f}$ about the trajectory. The methods given below will be derived, and generalized to cases of multiple lags, network models and arbitrary result fuctions, in sections (V) and (VI). With those derivation techniques, it is simple to calculate derivatives with respect to exogenous variables, $\underline{u}(t)$; examples are given in another EIA report[9], but the details will not be given below, for the sake of brevity. The discussion below will be limited to derivatives of relevance to sensitivity analysis, as discussed in section (III).

Assume that time will range from 0 (initial values) to T (final outcome).

### Methods To Compute First Order Derivatives

Forward Sweep:

To compute derivatives of all outcomes with respect to a change in one initial value $z_j(0)$:

$$\underline{x}(0) = \underline{e}_j \qquad (4.1a)$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) \qquad (t=0 \text{ to } T-1) \qquad (4.1b)$$

$\underline{x}(T)$ contains the derivatives ("impact multipliers").

To compute derivatives of all outcomes with respect to one parameter "a":

$$\underline{x}(0) = \underline{0} \qquad (4.2a)$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) + \underline{f}'_a(t), \quad (t=0 \text{ to } T-1) \qquad (4.2b)$$

where:

$$(\underline{f}'_a) = \frac{\partial \underline{f}}{\partial a}(\underline{z}(t)) \qquad (4.3)$$

Backwards sweep:

To compute all derivatives of a target variable $z_i(T)$ with respect to initial values $z_j(0)$ for all j:

$$\underline{x}'(T) = \underline{e}_i^T \qquad (4.4a)$$

$$\underline{x}'(t) = \underline{x}'(t+1)F(t) \qquad (t=T-1 \text{ to } 0) \qquad (4.4b)$$

$\underline{x}'(0)$ contains the derivatives ("impact multipliers").

To compute all derivatives of $z_i(T)$ to a parameter "a," as well, add to equations (4.4) the equations to determine a scalar "W(t)":

$$W'(T) = 0 \qquad (4.5)$$

$$W'(t) = W'(t+1) + \underline{x}'(t)\underline{f}'_a(t)$$

Methods To Compute Second-Order Derivatives (Two Initial Values)

These are methods to compute second derivatives of model outcomes with respect to a pair of initial values. (i.e. Second derivatives of $\underline{z}(T)$ with respect to $\underline{z}(0)$ and $\underline{z}(0)$).

Forward sweep:

To compute changes in impacts of $z_j(0)$ when $z_k(0)$ is changed (i.e second derivatives of all targets with respect to $z_j(0)$ and $z_k(0)$):

$$\underline{x}(0) = \underline{e}_j \qquad (4.6a)$$

$$\underline{y}(0) = \underline{e}_k \qquad (4.6b)$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) \qquad (4.6c)$$

$$\underline{y}(t+1) = F(t)\underline{y}(t) \qquad (4.6d)$$

$$\underline{w}(t+1) = F(t)\underline{w}(t) + F'(t)\underline{x}(t)\underline{y}(t) \qquad (4.6e)$$

($\underline{w}(T)$ contains the second derivatives)

Where:

$$F'_{ijk}(t) = \frac{\partial F_{ij}(\underline{z}(t))}{\partial z_k(t)} \quad , \qquad (4.7)$$

and where (4.6) includes notation from physics representing:

$$\sum_{j,k} F'_{ijk}(t)x_j(t)y_k(t)$$

Exploiting the sparsity of $F'$ is crucial to computational efficiency here.

Backwards sweep:

To compute all second derivatives of a selected outcome $z_i(T)$:

$$\underline{x}'(T) = \underline{e}_i^T \qquad (4.8a)$$

$$H(T) = 0 \qquad \text{(a matrix)} \qquad (4.8b)$$

$$\underline{x}'(t) = \underline{x}'(t+1)F(t) \qquad (4.8c)$$

$$H(t) = F^T(t)H(t+1)F(t) + \underline{x}'(t+1)F'(t) \qquad (4.8d)$$

$H(0)$ is the matrix of second derivatives.

Triple sweep:

To compute changes in impacts of all other initial values on $z_i(T)$ when $z_k(0)$ is changed (i.e. second derivatives of $z_i(T)$ with respect to $z_j(0)$ and all $z_k(0)$):

$$\underline{x}(0) = \underline{e}_j \qquad (4.9a)$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) \qquad (4.9b)$$

$$\underline{x}'(T) = \underline{e}_i^T \qquad (4.9c)$$

$$\underline{w}(T) = \underline{0}^T \qquad (4.9d)$$

$$\underline{x}'(t) = \underline{x}'(t+1)F(t) \quad (t=T-1 \text{ to } 0) \qquad (4.9e)$$

$$\underline{w}(t) = \underline{w}(t+1)F(t) + \underline{x}'(t+1)F'(t)\underline{x}(t) \qquad (4.9f)$$

$\underline{w}(t)$ contains the second derivatives.

Synchronistic sweep:

To compute how much responses by $z_i(T)$ with respect to $z_j(0)$ change when $z_k(0)$ changes (i.e. to compute just one second derivative):

$$\underline{x}'(t) = \underline{e}_i^T \qquad (4.10a)$$

$$\underline{x}'(t) = \underline{x}'(t+1)F(t) \quad (t=T-1 \text{ to } 0) \qquad (4.10b)$$

$$\underline{x}(0) = \underline{e}_j \qquad (4.10c)$$

$$\underline{y}(0) = \underline{e}_k \qquad (4.10d)$$

$$W(0) = 0 \qquad (4.10e)$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) \qquad (4.10f)$$

$$\underline{y}(t+1) = F(t)\underline{y}(t) \qquad (4.10g)$$

$$W(t+1) = W(t) + \underline{x}'(t)F'(t)\underline{x}(t)\underline{y}(t) \qquad (4.10h)$$

Notice that in practice one would sweep back a set of $\underline{x}'$, to represent a se of targets, and sweep forward a set of $\underline{x}(t)$, for the sake of efficiency; the approximate costs in Figure 4 assume that this is done. One would do likewise with the forwards sweep. Both in (4.8h) and, to a lesser degree, (4.6e), one can perform the multiplications cheaply by rationally considering the total set of what is needed. Thus, whichever is smallest - the set of $\underline{x}'$ or that of $\underline{x}$ - should be multiplied first by $F'$, in (4.8h), so as to reduce the dimensionality of the problem; index combinations irrelevant to the set of multipliers being requested may be left out. The synchronistic sweep through (4.8g) may also allow one to be sure that some multipliers are small, even without calculating them.

Methods to Compute Second Order Derivatives (Variable-Parameter)

These are methods to measure changes in ordered derivatives in response to changes in one or more parameter "a" (i.e. second derivatives of $\underline{z}(T)$ to $\underline{z}(0)$ and a set of "a").
They are basically just straightforward alterations of the variable-parameter methods above. In effect, equation (4.2b) represents the forwards propagation equation for a parameter, while equation (4.1b) represents the same for a variable; whenever (4.1b) has been used, we now use (4.2b) instead. See above for considerations in using the various methods.

Forward Sweep:

For responses of all targets to changes in $z_j(0)$, the change in response by increasing "a" (i.e. second derivatives of all targets with respect to $z_j(0)$ and "a"):

$$\underline{x}(0) = \underline{e}_j \qquad (4.11a)$$

$$\underline{y}(0) = \underline{0} \qquad (4.11b)$$

$$\underline{w}(0) = \underline{0} \qquad (4.11c)$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) \qquad (4.11d)$$

$$\underline{y}(t+1) = F(t)\underline{y}(t) + \underline{f}'_a(t) \qquad (4.11e)$$

$$\underline{w}(t+1) = F(t)\underline{w}(t) = F(t)\underline{x}(t) + F'_a(t)\underline{x}(t)\underline{y}(t) \qquad (4.11f)$$

($\underline{w}(T)$ holds the derivatives.)

Backwards Sweep:

For responses of $z_i(T)$ to changes in all components of $\underline{z}(0)$, the changes in response by increasing "a" (i.e. the second derivatives of $z_i(T)$ with respect to "a" and $z_j(0)$ for all j):

Add to equations (4.8) the equations:

$$\underline{w}(T) = 0 \qquad (4.12a)$$

$$\underline{w}(t) = \underline{w}(t+1)F(t) + \underline{x}'(t+1)F'_a(t) + \underline{f}'_a(t)H(t+1)F(t) \qquad (4.12b)$$

Triple sweep to all variables:

To compute the second derivative of $z_i(T)$ with respect to "a" and to $z_j(0)$, for all j simultaneously, when equations (4.8) are not being used:

$$\underline{y}(0) = \underline{0} \qquad (4.13a)$$

$$\underline{y}(t+1) = F(t)\underline{y}(t) + \underline{f}'_a(t) \qquad (4.13b)$$

$$\underline{x}'(T) = \underline{e}_i{}^T \qquad (4.13c)$$

$$\underline{w}(T) = \underline{0}^T \qquad (4.13d)$$

$$\underline{x}'(t) = \underline{x}'(t+1)F(t) \quad (t = T-1 \text{ to } 0) \qquad (4.13e)$$

$$\underline{w}(t) = \underline{w}(t+1)F(t) + \underline{x}'(t+1)F'_a(t) + \underline{x}'(t+1)F'(t)\underline{y}(t) \qquad (4.13f)$$

Triple sweep to all parameters:

To compute the second derivative of $z_i(T)$ with respect to $z_j(0)$ and

parameters "a" (typically for several "a", with equation (4.14h) replicated for each "a"):

$$\underline{x}(0) = \underline{e}_j \tag{4.14a}$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) \tag{4.14b}$$

$$W(T) = 0 \tag{4.14c}$$

$$\underline{x}'(T) = \underline{e}_i^T \tag{4.14d}$$

$$\underline{y}'(T) = \underline{0}^T \tag{4.14e}$$

$$\underline{x}'(t) = \underline{x}'(t+1)F(t) \tag{4.14f}$$

$$\underline{y}'(t) = \underline{y}'(t+1)F(t) + \underline{x}'(t+1)F'(t)\underline{x}(t) \tag{4.14g}$$

$$W(t) = W(t+1) + \underline{x}'(t+1)F'_a(t)\underline{x}(t) + \underline{y}'(t+1)\underline{f}'_a(t) \tag{4.14h}$$

The equations (4.14) answer the question,"How much does the responsiveness of $z_i(T)$ to parameter 'a' change when $z_j(0)$ is changed?"

Synchronistic sweep:

For the second derivative of $z_i(T)$ with respect to a parameter "a" and a variable $z_j(0)$:

$$\underline{x}'(T) = \underline{e}_i^T \tag{4.15a}$$

$$\underline{x}'(t) = \underline{x}'(t+1)F(t) \tag{4.15b}$$

$$\underline{x}(0) = \underline{e}_j \tag{4.15c}$$

$$W(0 = 0 \tag{4.15d}$$

$$\underline{y}(0) = \underline{0} \tag{4.15e}$$

$$\underline{x}(t+1) = F(t)\underline{x}(t) \tag{4.15f}$$

$$\underline{y}(t+1) = F(t)\underline{y}(t) + \underline{f}'_a(t) \tag{4.15g}$$

$$W(t+1) = W(t) + \underline{x}'(t+1)F'(t)\underline{x}(t)\underline{y}(t) + \underline{x}'(t+1)F'_a(t)\underline{x}(t) \tag{4.15h}$$

With sets of $z_i(T)$, $z_j(0)$ and "a", we typically pick one method out of the five, and sweep all sets together to avoid duplication of effort. The triple-sweep to all parameters, for example, is far more efficient when we sweep a set of parameters together; in that situation, only (4.14h) needs to be recalculated for each parameter.

## (V) DERIVATION OF BACKWARDS SWEEPS: DYNAMIC FEEDBACK

### Basics of Dynamic Feedback

Equations (4.4) may be derived in several ways. Section (VI) will give a relatively easy and informal derivation, based on methods borrowed from physics. Here, we will use the "dynamic feedback" principle illustrated in Figure 3, in section (II). This principle is slightly more complicated to explain than the other derivation is, but the principle can be applied easily to a wide range of problems. It is based on a "chain rule" for ordered derivatives, the prrof of which is reproduced from the original source[6] in Appendix A.

Given an ordered or partially ordered system of variables, with fundamental dynamic relations expressable as:

$$x_i = f_i(x_{i-1}, \ldots, x_1, \text{parameters}), \tag{5.1}$$

it is proven for the corresponding ordered derivatives that:

$$\frac{\partial^+ x_n}{\partial x_j} = \sum_{i=j+1}^{n} \frac{\partial f_i}{\partial x_j} \cdot \frac{\partial^+ x_n}{\partial x_i}, \tag{5.2}$$

where we are combining an ordinary partial derivative (simply the derivative of the function $f_i$ as it appears as an algebraic expression in equation (5.1)) with ordered derivatives. Also, from the definition of ordered derivative (see Appendix A):

$$\frac{\partial^+ x_n}{\partial a} = \frac{\partial x_n}{\partial a}$$

The simple model assumed in section (IV) may be written as:

$$z_{k,t+1} = f_k(z_{1,t}, \ldots, z_{p,t}, \ldots z_{n,t}) \qquad \text{(for all } k,t)$$

If we order the numbers $z_{k,t}$ chronologically (i.e. later times are treated as later causally), and if we choose "$x_i$" and "$x_j$" in equation (5.2) to represent $z_{k,t}$ and $z_{p,t}$ respectively, then substitution into equation (5.2) tells us:

$$\frac{\partial^+ x_n}{\partial z_p(t)} = \sum_k \frac{\partial f_k(\underline{z}(t))}{\partial z_p(t)} \cdot \frac{\partial^+ x_n}{\partial z_k(t+1)}$$

Note that the simple partial derivatives of $f_k(\underline{z}(t'))$, for $t'$ not equal to t, are simply zero in this case, so that they do not contribute to the sum. In other words, terms enter the sum only

when they represent a direct causal link. "$x_n$" in equation (5.2) can be associated with any measure of the final outcome of the system, such as $z_{i,T}$; with this substitution, the equation above becomes identical to equation (4.4b), with the definition:

$$x'_k(t) = \frac{\partial^+ x_n}{\partial z_k(t)}$$

Likewise, the remainder of equations (4.4) is also correct, under these substitutions. These substitutions are the same kind of substitution that one makes when applying the ordinary chain rule of basic calculus to specific functions and problems.

Using the same procedure, but associating "$x_j$" with a parameter "a", and treating parameters as causally prior to all other variables (as in Appendix A), equations (4.5) follow directly from substitution. With a more complex model, however (e.g. a large network of relations which is essentially sparse), it may be less expensive to apply (5.2) directly to each node (variable) in the network. For example, direct use of (5.2) may be essential with synthetic "neuron networks." Likewise with multiple lags, for which the equations are given below.

Second Order Backwards Sweeps

For convenience, let us write the outcome whose derivatives are desired as $L = x_n$. We may then rewrite equation (5.2) as:

$$\frac{\partial^+ L}{\partial x_j} = \sum_{k=j+1}^{n} \frac{\partial f_k}{\partial x_j} \cdot \frac{\partial^+ L}{\partial x_k} \qquad (5.3)$$

Differentiating with respect to i, for i>j:

$$\frac{\partial^2 L}{\partial x_i \partial x_j} = \sum_{k=j+1}^{n} \frac{\partial f_k}{\partial x_j} \cdot \frac{\partial^+}{\partial x_i}\left(\frac{\partial^+ L}{\partial x_k}\right) + \frac{\partial^+ L}{\partial x_k} \cdot \frac{\partial^+}{\partial x_i}\left(\frac{\partial f_k}{\partial x_j}\right) \qquad (5.4)$$

(Note: in applying the chain rule, 5.2, to 5.4, it is important that the new $x_n$ will now itself be a derivative of L at time T, and will not equal L itself. The only complication to worry about from this is that the double ordered derivatives equal one when i=j=T. Also, note that the last double derivative on the right is basically a total differential of the algebraic expression which results from taking a partial derivative.) In our situation it is practical to apply

the chain rule and substitution again to get:

$$\frac{\partial^2 L}{\partial x_i \partial x_j} = \sum_{k,l} \frac{\partial^2 L}{\partial x_k \partial x_l} \cdot \frac{\partial f_k}{\partial x_j} \cdot \frac{\partial f_l}{\partial x_i} + \sum_{k} \frac{\partial L}{\partial x_k} \cdot \frac{\partial^2 f_k}{\partial x_i \partial x_j} \qquad (5.5)$$

if we assume that i and j represent components of $\underline{z}$ at the same time, and k and l components at later times. This reduces immediately to (4.8), for the simple model assumed in section (IV). The case of a variable and a parameter is more complex; see Section (VI).

Extensions and Applications

In sensitivity analysis, the user is often not interested in the model variables per se at the time T, but in a function of those variables:

$$R(\underline{z}(T))$$

The more general equations above tell us that we can modify our procedures, to pick an arbitrary R as target, by inserting modified boundary conditions:

$$\underline{x}'_i(T) = \frac{\partial R(\underline{z}(T))}{\partial z_i(T)} \qquad (4.4')$$

(4.8a′)

(4.9c′)

(4.13c′)

(4.14d′)

(4.15a′)

Likewise, one could set R to be the sum over time of a utility function or loss function, and thereby use these methods to speed up the key calculations of an estimation or deterministic optimization problem. In such problems, there is only one target result (e.g., loss); therefore, the backwards sweep methods tend to be far more efficient than conventional methods. For example, equation (5.2) has been applied[6] to calculating derivatives for estimating multivariate ARMA processes, and implemented within the Time-Series Processor (TSP) on the MIT Honeywell Multics computer.

Equation (5.2) leads to an obvious extension of the methods of Section 4, to the case of multiple lags. Wherever we have written:

$$\underline{x}'(t) = \underline{x}'(t+1)F(t)$$

(for $\underline{x}'$ or any other vector), we have actually intended, for any recursive system of the form:

$$z_i(t) = f_{i,t}(z_{i-1}(t)\ldots z_1(t), \underline{z}(t-1), \ldots \underline{z}(t-k)),$$

the equation:

$$x'_j(t) = \sum_{i=1}^{j-1} x'_i(t) \frac{\partial f_{i,t}}{\partial z_j(t)} + \sum_{s=1}^{k} \sum_{i=1}^{n} x'_i(t+s) \frac{\partial f_{i,t+s}}{\partial z_j(t)} \tag{5.6}$$

## (VI) DERIVATION OF METHODS FROM PERTURBATION APPROACHES OF PHYSICS

Most physicists would have preferred a simpler, but more specialized, derivation of (4.4). They would begin by saying that the validity of (4.1) is obvious from our intuitive understanding of what an impact multiplier is. Then they would define a "propagator," "D" (where D stands for "Dyson"), as the matrix product:

$$D(s:t) = F(s)F(s-1)\ldots F(t) \tag{6.1}$$

For convenience, we also define $D(s:s+1) = I$.

Equation (4.1) clearly implies:

$$\underline{x}(T) = D(T-1:0)\,\underline{x}(0) \tag{6.2}$$

The impact multiplier from $z_j(0)$ to $z_i(T)$ is then:

$$D_{ij}(T-1:0) = \underline{e}_i{}^T D(T-1:0)\underline{e}_j\ , \tag{6.3}$$

where "$\underline{e}_j$" is defined as the vector whose j-th component is one and whose other components are zero. If we define:

$$\underline{x}'(t) = \underline{e}_i{}^T D(T-1:t) \tag{6.4}$$

then (4.4) follows very simply from the definition of D. More precisely, (6.1) implies:

$$D(s:t) = D(s:t+1)F(t) \tag{6.5}$$

$$D(s+1:t) = F(s+1)D(s:t) \tag{6.6}$$

and (4.4) follows directly from (6.4) and (6.5). Likewise, (4.6) would be considered intuitively obvious. If so, it implies:

$$\underline{x}(t) = D(t-1:0)\underline{e}_j$$

$$\underline{y}(t) = D(t-1:0)\underline{e}_k$$

$$F'(t)\underline{x}(t)\underline{y}(t) = F'(t)D(t-1:0)\underline{e}_j D(t-1:0)\underline{e}_k$$

$$\frac{\partial^{2+} z_i(T)}{\partial z_j(0)\partial z_k(0)} = \sum_{t=0}^{T-1} \underline{e}_i{}^T D(T-1:t+1)F'(t)D(t-1:0)\underline{e}_j D(t-1:0)\underline{e}_k \tag{6.7}$$

This expression has the appearance of a "Feynmann integral," which is well known to physicists. If we further define:

$$H_{jk}(t') = \sum_{t=t'}^{T-1} \underline{e}_i{}^T D(T-1:t+1) F'(t) D(t-1:t') \underline{e}_j D(t-1:t') \underline{e}_k \quad (6.8)$$

then equation (6.7) is equivalent to (4.8d), when we use (6.5) to establish the recursion relations. From a physicist's perspective, this verifies (6.7) and (4.6). In like manner, (4.9) is a simple consequence of (6.7), using relations (6.5) and (6.6).

Finally, in the variable-parameter case, it is most convenient to combine the methods of this section with those of the previous section. We have seen that:

$$\frac{\partial^+ z_i(T)}{\partial z_j(0)} = \underline{e}_i{}^T F(T-1) \ldots F(0) \underline{e}_j$$

Differentiating:

$$\frac{\partial^+}{\partial a} \left( \frac{\partial^+ z_i(T)}{\partial z_j(0)} \right) = \frac{\partial^+}{\partial a} \left( \underline{e}_i{}^T F(T-1) \ldots F(0) \underline{e}_j \right)$$

$$= \sum_{t=0}^{T-1} \underline{e}_i{}^T F(T-1) \ldots F(t+1) \left( \frac{\partial^+}{\partial a} F(t) \right) F(t-1) \ldots F(0) \underline{e}_j$$

But if we recall the definition of F(t), we get:

$$\frac{\partial^+ F_{ij}(t)}{\partial a} = \frac{\partial F_{ij}(t)}{\partial a} + \sum_k \frac{\partial^2 f_i(\underline{z}(t))}{\partial z_j(t) \partial z_k(t)} \cdot \frac{\partial^+ z_k(t)}{\partial a}$$

The first of these two terms we have denoted $F_a'(t)$; the second is

$F'(t)\underline{y}(t)$, where $\underline{y}$, as in equation (4.2b), is the forwards-propagated influence of the parameter "a". Thus by analogy with (6.7), we now have:

$$\frac{\partial^{+} z_i(T)}{\partial a \partial z_j(0)} = \sum_{t=0}^{T-1} \underline{e}_i^T D(T-1:t+1) F_a'(t) D(t-1:0) \underline{e}_j$$

$$+ \sum_{t=1}^{T-1} \sum_{s=0}^{t-1} \underline{e}_i^T D(T-1:t+1) F'(t) D(t-1:0) \underline{e}_j D(t-1:s) f_a'(s) \tag{6.9}$$

With the recursion relations (6.5) and (6.6), the definitions of H in (6.8), and the obvious definitions of propagated influence, the variable-parameter methods in Section (IV) all follow from (6.9) in a straightforward manner.

## (VII) APPLICATION TO STOCHASTIC OPTIMIZATION AND INTELLIGENT SYSTEMS

The remainder of this paper will discuss an example of how these methods can be adapted to solve very difficult numerical problems. It will show how to implement "Global Dual Heuristic Programming" (GDHP), a previously proposed method for (approximately) optimizing decisions over time, subject to uncertainty[16]. The methods above make it feasible for the first time to apply GDHP to very large scale models or systems. This includes systems or models which involve many variables or sectors, whose interrelationships are better described as a "network" than as a matrix.

Effective decision-making over time, despite uncertainty, is the purpose of many systems, from computer systems through to government agencies. Therefore, this example may itself have many applications.

GDHP is not complete. To use the calculations below, one must supply additional information, such as a model of the environment one is trying to influence, and other information. Nevertheless, there do exist prototype methods for providing the required information, for essentially any environment; these methods reuire further development and extension, but there is a large body of generalized work in statistics and numerical analysis which such developments can build upon. By combining GDHP with generalized methods to provide the inputs required by GDHP, one arrives at a recipe for a generalized decision-making system, with foresight, which learns almost everything it knows from its environment rather than a preprogrammed "knowledge base" or "feature set."

This example provides a mathematical framework in which model development can be treated as only one of several fundamental activities required to support rational decision making; this framework may be useful in allowing one to address issues which can go beyond the internal structure of a model, issues such as the relation between robustness and utility.

The section below will first describe GDHP, and then show how the differentiation methods above make it possible to implement GDHP. It will then discuss applications and structural analogies with the human brain. Comparison with the human brain suggests possibilities for enhancing the decision-making system proposed here.

## GDHP: An Approach to Optimizing Complex, Nonlinear Stochastic Systems

GDHP provides an approximate solution to the following problem:

GIVEN THAT $\underline{x}(t+1) = \underline{F}(\underline{x}(t), \underline{u}(t), \underline{e}(t))$,
where $\underline{e}(t)$ is random, $\underline{u}(t)$ a control (decision variables)
MAXIMIZE $\langle U(\underline{x}) \rangle$, the expectation across all future times

(Note that we use the notation "<...>" to represent expected value.) For example, "$\underline{F}$" may be a multiequation model of the economy, and "U" may be a measure of the success of economic policy.

The exact solution to this problem, as described in Howard's book[17] on dynamic programming, requires the calculation of both a scalar function $J(\underline{x})$ and a vector function $\underline{u}(\underline{x})$. $\underline{u}(\underline{x})$ represents the optimal action (or motor output or policy variables) as a function of the state of the environment, $\underline{x}$. $J(\underline{x})$ is a measure of how good the results of the actions are, in terms of their total long-term impact. Howard proves that one can normally converge on a choice for $\underline{u}(\underline{x})$ which maximizes the future expected value of utility ($U(\underline{x})$ over future $\underline{x}$) by alternately: (1) picking values for $\underline{u}(\underline{x}(t))$, for all possible $\underline{x}(t)$, which maximize the expected value of J at time t+1; (2) finding the unique function J which solves the following equation for the current guess for $\underline{u}(\underline{x})$ across all possible $\underline{x}$:

$$J(\underline{x}) = \langle J(\underline{F}(\underline{x},\underline{u}(\underline{x}),\underline{e})) \rangle + U(\underline{x}) - U_o, \qquad (7.1)$$

where $U_o$ is a constant to be solved for. (In "crossroads" situations this may not work; however, GDHP may still be usable[16].)

Howard's procedure, like other exact forms of dynamic programming, becomes impossible expensive when the number of variables in the system ($\underline{x}$) becomes more than a half-dozen or so. The problem is that there are too many possible values of the vector $\underline{x}$ to consider. In other words, $J(\underline{x})$ and $\underline{u}(\underline{x})$ contain too many possible degrees of freedom in the general case.

In GDHP, one tries to find the "best possible" $J(\underline{x})$ and $\underline{u}(\underline{x})$ from a more limited set of alternatives, in order to find an approximately optimal strategy of action[16]. One assumes that the user (or a higher level computer system) has already proposed functional forms for J and $\underline{u}$, $J^*(\underline{x}, \underline{a})$ and $\underline{u}^*(\underline{x}, \underline{b})$ respectively, which depend on vectors of parameters $\underline{a}$ and $\underline{b}$. GDHP attempts to adjust the parameters of $J^*$ and $\underline{u}^*$ to make these factors fit the conditions for optimality as closely as possible, over a finite (but not necessarily fixed) set of possible scenarios, $\underline{x}$, and randomly simulated vectors $\underline{e}$. In other words, GDHP does for models of J what statisticians and econometricians have done for years with models to predict other dependent variables.

GDHP entails two steps to be carried out alternately or in parallel until convergence:

- Maximize $< J^*(\underline{F}(\underline{x},\underline{u}^*(\underline{x},\underline{b}),\underline{e})) >$ (7.2)
  over parameters $\underline{b}$, scenarios $\underline{x}$, random $\underline{e}$.

- Pick $\underline{a}$ in $J^*(\underline{x},\underline{a})$ to minimize the average over scenarios $\underline{x}$ and simulated $\underline{e}$ of:

$$E = \sum_i W_i \left( \frac{\partial}{\partial x_i} < J^*(\underline{F}(\underline{x},\underline{u}(\underline{x},\underline{b}),\underline{e}),\underline{a}) - U(\underline{x}) - J^*(\underline{x},\underline{a}) > \right)^2, \quad (7.3)$$

where the $W_i$ are a set of weights. If $J^*$ can solve Howard's equation exactly with $\underline{u}^*$, for some $\underline{a}$, then E will always equal zero for that $\underline{a}$.

The origin of Howard's equation suggests that one can treat the $J^*$ on the left as fixed in each iteration and still converge to a valid result. The calculations below will implement GDHP in that way, although it would be straightforward (if tedious) to treat both $J^*$ as variable instead in the calculations below. The decision on whether to treat both $J^*$ as variable involves a tradeoff between the rate of convergence and the stability of the process. The tradeoff depends on the minimization method used, and has not been fully resolved.

The scenarios $\underline{x}$ would normally be provided by simulating ahead from the present, using the model $\underline{F}$. Or they could be suggested by combining sets of scenarios of interest to users. The theory of Monte Carlo integration may make it possible to be more scientific about this process in the future.

The approximations used by GDHP require some explanation.

First, consider the use of specific functional forms instead of general functions J and $\underline{u}$. No realistic system can actually use an estimate of J which involves a functional form which exceeds the available storage space; thus it is necessary to limit explicit attention to specific realizable functional forms. We can first analyze the problem of parameter estimation for a fixed functional form, before studying how to compare and improve functional form, exactly as in statistics. In artificial intelligence, successful game playing machines have required "static position evaluators," which correspond with the approximate J function used in GDHP to evaluate $\underline{u}(\underline{x})$.

There are several obvious problems in assuming a functional form for $J^*$:

- The true J which obeys Howard's equation will usually not be expressable exactly as $J^*(\underline{a})$ for any $\underline{a}$
- The need for a user or metaprogram to choose the functional form
- The problem of choosing weights $W_i$
- Dangers that autocorrelation might invalidate the adjustment process
- The need to worry about robustness of the results, influential observations and unobserved (estimated) variables.

All of these problems are precisely parallel to known problems in estimating statistical forecasting models $\underline{F}^*$ with fixed functional forms; as in statistics, these problems require analysis but do not invalidate the approach. Indeed, the methods of analysis needed to extend GDHP are very similar to those used in statistics.

Reasons for choosing E as a loss function for $J^*$ include:

- The derivatives of J are the familiar "shadow prices" or "Lagrange multipliers."
- Decisions ($\underline{u}$) are based on comparing alternatives (usually nearby alternatives), such that the accuracy of the derivatives of J determines the accuracy of the decisions.
- This eliminates Howard's $U_o$ constant, and related potential problems in "crossroads" situations[16].
- Because $\underline{e}$ and $\underline{x}$ are both held constant in any one scenario comparison, this method has the increased statistical efficiency which one expects with "paired comparisons[18]."
- In effect, this procedure "explains" successes or failures through the causal model $\underline{F}$, and reinforces or weakens precisely those action parameters which caused the result (or reinforces nothing, if the result was due to chance).

Other generalized methods have been proposed to address the problem as formulated above[16], and others may be proposed in the future; however, because of the factors discussed above, GDHP appears to be more promising than the current alternatives. More numerical and statistical study is needed in order to fully determine the properties of GDHP.

## Application of Differentiation Methods

The major difficulty in implementing GDHP is to minimize or maximize the expressions in equations (7.2) and (7.3). In the general case, where no strong special assumptions are made about J and $\underline{F}$ and $\underline{u}$, and where there are many parameters in these functions, this requires that one obtain the derivatives of the quantities to be minimized or maximized. Even when derivatives are available, minimizing a function of many variables is not trivial. However, the EIA long term model now converges in 10 to 100 iterations in solving 100,000 nonlinear simultaneous equations; the approach used is general, and there is reason to hope that numerical analysts will be able to increase the reliability and efficiency of such methods for dealing with large systems[12]. As a practical metter, it is probably best to carry out the minimization and maximization in parallel, scenario by scenario, so as to avoid a costly process of iteration within iteration.

This paper will indicate how to calculate the required derivatives efficiently. The real challenge in this case is that E already contains first derivatives; thus the required derivatives of E involve second derivatives. Furthermore, in order to emphasize the possibilities of "parallel processing"[10] and network implementation mentioned in section (II), we will assume that parallel processors can be used to implement J and $\underline{F}$ and $\underline{u}$ at an acceptable cost; the challenge here is to calculate the required derivatives on a similar basis, without restricting what J and $\underline{F}$ and $\underline{u}$ can be. (In "hierarchical control theory," for example, one restricts the form of these functions to make the problem tractible.) The case of ordinary sequential computers is a simple special case of the calculations to be described.

First let us introduce some notation. Let us define:

$$\begin{aligned} \underline{f}(\underline{x},\underline{e},\underline{b}) &= \underline{F}(\underline{x},\underline{u}^*(\underline{x},\underline{b}),\underline{e}) \\ U_i(\underline{x}) &= \frac{\partial}{\partial x_i}(U(\underline{x}(t)) \\ d_i &= \frac{\partial}{\partial x_i}(J^*(\underline{f}(\underline{x},\underline{e},\underline{b})) - U(\underline{x}) - J^*(\underline{x})) \end{aligned} \qquad (7.4)$$

In this notation, the required derivatives of the expressions in equations (7.2) and (7.3) are simply the averages across all pairs of $\underline{x}$ and $\underline{e}$ considered of:

$$\frac{\partial^+}{\partial b_i} J^*(\underline{f}(\underline{x},\underline{e},\underline{b})) \qquad \text{(all j)} \qquad (7.5)$$

$$\frac{\partial^+ E'}{\partial a_i} = \sum_j 2W_j d_j \left( \frac{\partial^2 f}{\partial a_i \partial x_j} J^*(\underline{x},\underline{a}) \right) \quad , \text{(all i)} \qquad (7.6)$$

when we hold the rightmost "$\underline{a}$" in (7.3) constant.

Next we invoke the assumption that $J^*$, $\underline{F}$ and $\underline{u}^*$ (and thus $\underline{f}$) can be realized as a "network" of calculations. This is illustrated in Figure 5; each circle in Figure 5 represents a variable to be calculated, and each arrow represents a flow of information required in the calculations. When $J^*$, $\underline{F}$ and $\underline{u}^*$ are expressed in terms of elementary calculations, our differentiation procedures will appear more complicated, but this is deceptive; the key point is that the elementary calculations are each very simple to perform, and to differentiate.

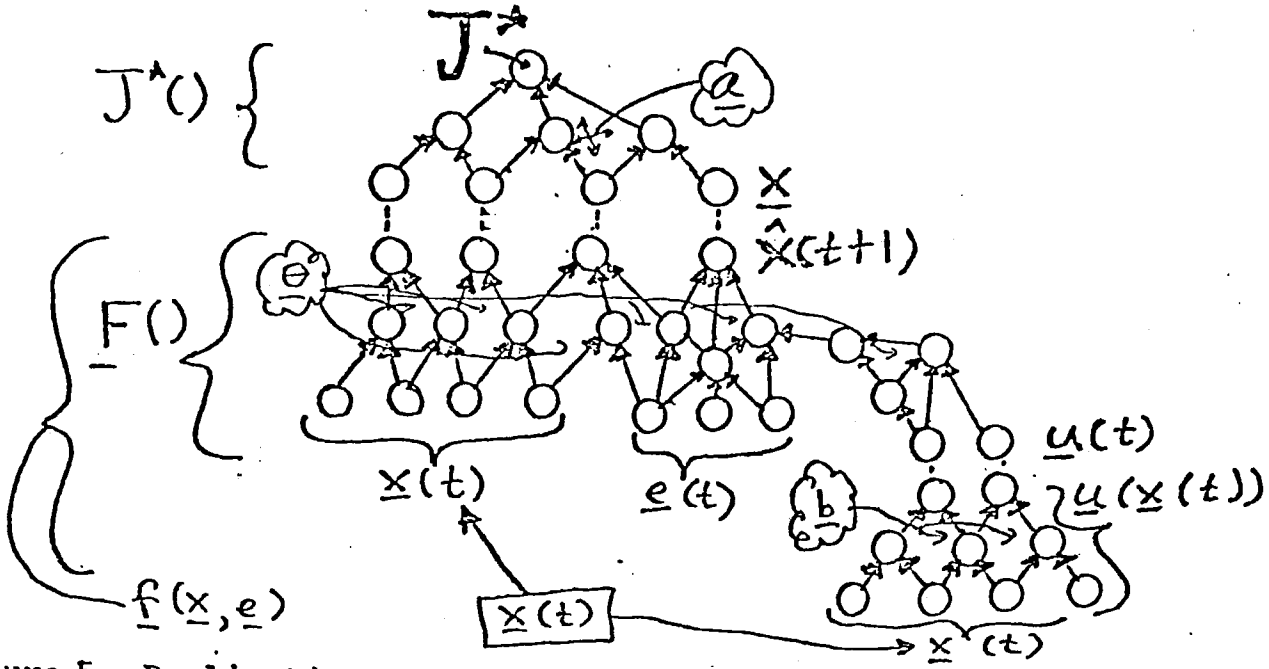



Figure 5: Realization of GDHP As a Triple Network to Make Decisions