Applications of Advances in Nonlinear Sensitivity Analysis

Paul J. Werbos, U.S. Department of Energy
Forecast Analysis and Evaluation Team

The following paper summarizes the major properties and applications of a collection of algorithms involving differentiation and optimization at minimum cost. The areas of application include the sensitivity analysis of models, new work in statistical or econometric estimation, optimization, artificial intelligence and neuron modelling. The details, references and derivations can be obtained by requesting "Sensitivity Analysis Methods for Nonlinear Systems" from Forecast Analysis and Evaluation Team, Quality Assurance, OSS/EIA, Room 7413, Department of Energy, Washington, DC 20461.

Context of the Work

The Energy Information Administration (EIA) provides data and analysis on all aspects of energy supply and demand. It uses dozens of models, including econometric (statistical, empirical) models, linear programming models based on technological data, a nonlinear micro equilibrium model solving for thousands of variables simultaneously across a 50-year span, hybrids and combinations of these, etc.

Many users of EIA´s analyses do not accept EIA´s conclusions at face value, especially when reports from other sources disagree. Thus the Forecast Evaluation and Analysis Team of EIA and its predecessors have carried out a broad program to evaluate and explain the qualitative assumptions of EIA models and forecasts. This program includes the development of tools to characterize the properties of large models, studies of estimation methods which are robust against outliers or model misspecification (i.e., correlated errors), proofs of convergence and existence properties, and many other projects. The first part of this paper describes how a small part of this work - the minimum cost calculation of first and second order derivatives of nonlinear systems - makes an essential contribution to the rest. The second part elaborates on another application, a method for stochastic optimization which becomes feasible only with the help of low-cost derivatives. This method opens up a wholly new approach to the field of artifical intelligence and neuron modelling; it is especially efficient with the new generation of "parallel" computers.

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

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

$$\frac{\partial^+ x_i(T)}{\partial x_j(1)}$$

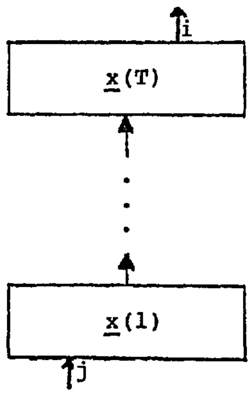


Figure 1: A Simple Example

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_j in time period 1, we want to know how large the resulting change in x_i is in the final time period T.

The change in $x_i(T)$ per change in $x_j(1)$, holding the rest of $\underline{x}(1)$ constant, is a fundamental quantity of the system. 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." 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 and rigorous, as will be seen.

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 derivatives of a function of $\underline{x}(T)$ or of $\underline{x}(t \leq T)$ rather than the derivatives of $x_j(T)$ 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.

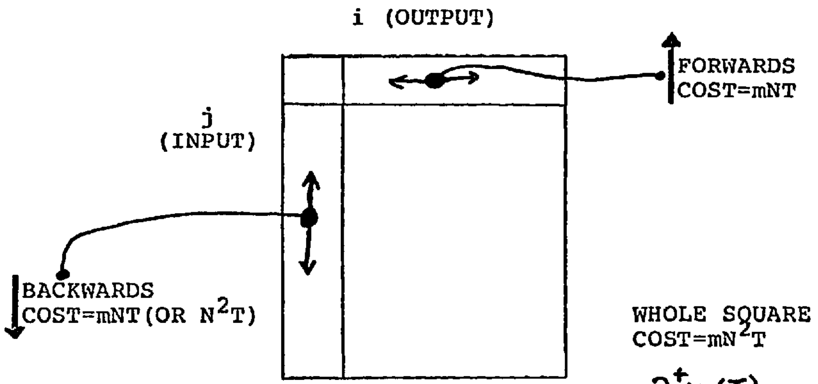


Figure 2: Two Ways to Obtain $\dfrac{\partial^+ x_i(T)}{\partial x_j(1)}$

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

$$\uparrow: \quad \underline{z}(t) = \frac{\partial^+ \underline{x}(t)}{\partial x_j(1)} \; ; \quad \underline{z}(t+1) = f'(t)\underline{z}(t)$$

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

The large square in Figure 2 represents the entire matrix of ordered derivatives of all $x_i(t)$ with respect to all $x_j(1)$. 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

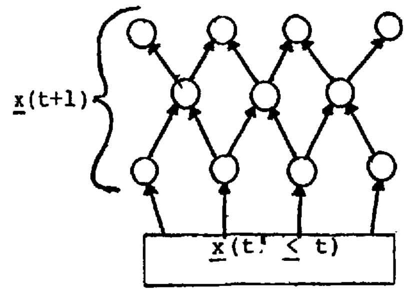
$\underline{x}(T)$. Each time we apply this method, we perturb only one of the initial values; thus we obtain only one row of the matrix of ordered derivatives, as shown in Figure 2. This costs us mNT calculations, as shown. Often the initial value $x_j(1)$ is actually changed, and the model resimulated. (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 one computes each derivative by subtracting two numbers very close to each other in size. MIT's Troll System uses the forwards closed-form Jacobian formula, shown at the bottom of Figure 2, which 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 engineering, this sort of method has been used for many years with "constrained derivatives," but has not been applied more generally.

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 like Hannan observed years ago that white noise converts a simple econometric model (like our example, but linear) into a "vector mixed autoregressive 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") 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 error), 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, 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; 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 hunreds 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: $\underline{x}(t+1) = f(\underline{x}(\text{all } t' \leq t+1), \underline{u}(\text{all } t'))$

Now let us consider the more general model shown in Figure 3. All nonnegative lags - including zero - are permitted in the endogenous variables. However, we still assume here (as in the paper) 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 $\underline{x}(t+1)$. The dynamic feedback "chain rule" has not been published before.

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 mNT 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. 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.

In this paper, we have discussed derivatives with respect to initial values of the variables only; however, the EIA report does consider parameters, and the case of exogenous variables is a trivial extension of the endogenous variable case. To avoid making a complicated discussion even more complicated, the EIA report only mentions 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. The section on stochastic optimization provides a partial example of the possibilities.

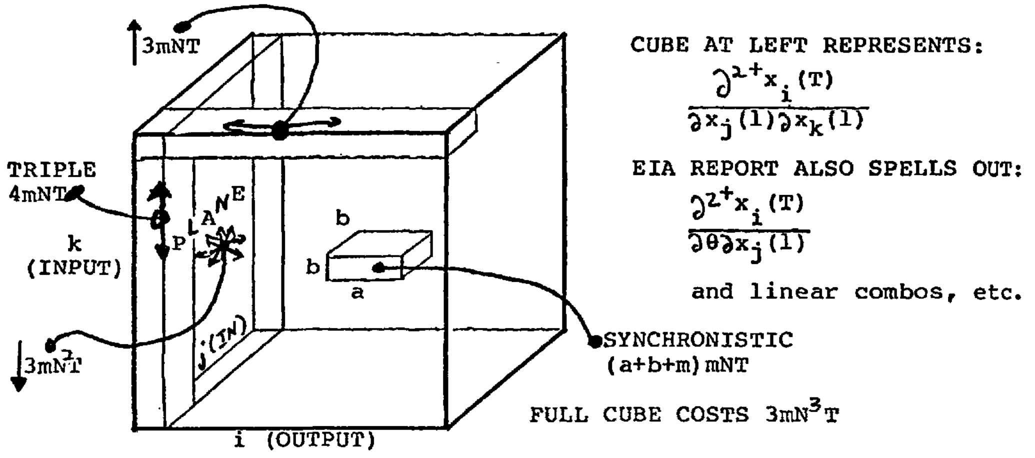


Figure 4: Costs of Obtaining Various Sets of Second Derivatives

Figure 4 provides a summary of the properties of the four variable-variable second derivative calculation methods provided in the EIA report. The set of ordered derivatives of $x_i(T)$ to $x_j(1)$ and $x_k(1)$ form 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.

The EIA report notes that variable-parameter second derivatives provide meaningful information about a model, essentially equivalent to what MIT provides for linear systems 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.

Applications of Stochastic Optimization Over Time

The remainder of this paper will discuss a way to implement "GDHP," a previously published approach to stochastic optimization over time. The methods discussed above make GDHP cheap enough to be feasible with very large scale models or systems. This opens up many possible applications.

GDHP provides an approximate solution to Howard's 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,
MAXIMIZE $\langle \overline{U}(\underline{x}) \rangle$, the expectation across all future times

Applications include: (1) use of large stochastic policy models in decision making; (2) devising distributed/decentralized systems to output optimal policy; (3) general adaptive artifical intelligence; and (4) modelling of learning in the brain ("neural plasticity").

The first of these applications is straightforward. Let "$\underline{F}$" be the policy model, and let "U" represent the values to be served by the decisions. How to formulate "U" is an old and unavoidable issue, beyond the scope of this paper.

The second application is an extension of the fields of micro-economics and hierarchical control theory. Economists have proven many theorems about how a large deterministic optimization problem (devising an efficient pattern of production) may be decentralized; however, these theorems typically assume no uncertainty or fundamental structural "change" (nonlinearity). If the actual problem is general and stochastic (beset by uncertainty), one needs a method of solution to the stochastic problem which can be implemented efficiently in a decentralized (i.e., parallel processor) system. The EIA report describes how GDHP may be implemented in this way.

In artificial intelligence, if one does not impose severe constraints on the range of models $\underline{F}$ or of action strategies to be considered (which would be inappropriate in an adaptive system), one must develop a generalized approach to optimization in order to find the optimal action strategy. However, numerical analysts have found, even with the much simpler task of maximizing a function of a few variables, that first derivatives at a minimum are essential to finding a maximum in a reliable, efficient manner. Until the failure of the Minsky-Selfridge "jitters" machine, there was widespread recognition of the fact that generalized optimization and adaptive intelligence are almost inseparable concepts. The "jitters" failure does not invalidate the original concept, however; it merely shows that one needs a full set of valid derivatives in each iteration (as is well known to numerical analysts) and that one cannot make do with a factor of N less information. (i.e., One derivative per time period, as in "jitters.") Likewise, a system needs to have an explicit model of its environment in order to properly optimize actions over time as in dynamic programming.

There is an additional reason why artificial intelligence research abandoned the idea of explicit optimization in the 1960's. Much of the work at that time was inspired by an early description of the neuron (brain cell) developed by McCulloch and Pitts. In that description, the variables x_i computed by the brain could take on only two values, 0 or 1 ("all" or "none"); thus the functions $\underline{F}$, J and $\underline{u}$ above could not be differentiable. However, more recent work has shown that neurons in the human brain use a "code" based on "volleys," such that the variables vary over a continuous range; thus optimization of a network of model neurons is not intrinsically different from conventional optimization based on a specified functional form.

Similar concepts apply to the field of actual neuron modelling. Like objects in solid state physics, neurons are very complicated, and will never be totally encompassed by theory; as with solid state physics, however, a general theory of neuron networks as intelligent systems may help increase the range of important phenomena which can be understood. This is a far cry from the ad hoc approach to brain modelling, in which a given set of equations is derived by appeals to simplification and common sense only. When the actual functional ability of a model to reproduce intelligence is considered, many otherwise credible models can be quickly ruled out; for example, many theorems have been proven about the Grossberg neuron model, but in statistical terms, this model uses simple correlation coefficients as the parameters of multivariate forecasting equations. Some neurologists have compared themselves to a man who studies a radio as follows: he pulls out a transistor, notices that the radio whines, and calls the transistor the "whine center"; a deeper understanding of brain functioning requires more consideration of the mathematical elements which are necessary in order to produce generalized intelligence, defined as the ability to learn and adapt to totally new problems.

The method discussed here - GDHP - does not address all aspects of stochastic optimization. The theory needs further development. However, there is a very close parallel between GDHP and conventional statistical estimation methods. As with statistics, specific examples of problems will be important to improving our understanding of stochastic optimization. However, as with statistics, examples alone will not be enough; the underlying theory needs explicit development at a generalized level, if we are ever to cope with very complex problems and improve the general methods.

General Approach Used in GDHP

The exact solution to Howard's problem, as described in Howard's book on dynamic programming, requires the calculation of a scalar function $J(\underline{x})$ and a vector function $\underline{u}(\underline{x})$. $\underline{u}(\underline{x})$ yields 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 ($U(\underline{x})$) by alternately: (1) finding the unique function J which solves a certain equation for the current guess for $\underline{u}(\underline{x})$; and (2) picking a new guess for $\underline{u}(\underline{x}(t))$ so as to maximize the expected value of J at time t+1.

In GDHP, one does not find the exact theoretical J and $\underline{u}$, because this tends to be infeasible with large systems. Instead, one assumes that the user or a higher-level system has proposed functional forms, J* and $\underline{u}$*, for J and $\underline{u}$. GDHP attempts to adjust the parameters of J* and $\underline{u}$* to make them fit the conditions for optimality as closely as possible, over a finite (not necessarily fixed) set of possible scenarios, $\underline{x}$. This involves two steps, to be carried out alternately or in parallel until convergence:

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

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

$$E \triangleq \sum_i W_i \left(\frac{\partial}{\partial x_i} \langle J^*(\underline{F}(\underline{x},\underline{e})) - \underline{u}^*(\underline{x}) - J^*(\underline{x}) \rangle \right)^2,$$

where $\underline{f}(\underline{x},\underline{e})$ is defined as $\underline{F}(\underline{x},\underline{e},\underline{u}(\underline{x}))$ and where the W_i are a set of weights. If J* can solve Howard's equation exactly with u*, for some a, then E will always equal zero for that a.

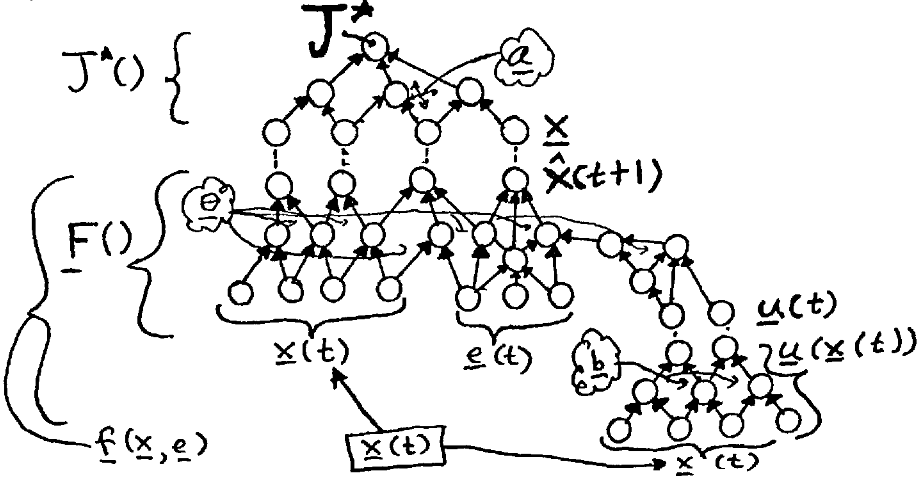


Figure 5 : Realization of GDHP As a Triple Network to Make Decisions

GDHP can be implemented as part of a procedure to optimize the choices of actions over time, in a situation where the dynamics of the environment and the optimal actions are both to be learned from empirical observation rather than specified a priori. This requires updating the parameters of three functions: J*, u* and F*. (Updating F* is a well studied problem in statistics.) Such a system can be implemented as a three-level network as shown in Figure 5; each level contains a network realization of one of the three functions. GDHP and statistics would be used to adopt the parameters; the network needs no external guidance except for data input and functional forms.

Figure 5 has an interesting analogy to the mammalian brain, which is essentially made up of three interlocking networks of brain cells: (1) the "limbic" network, which, like J calculates system values ("values" as in "values we cherish") which are the basis for reinforcing actions; (2) the cortico-thalamic system which, like F, embodies a model of the external environment; and (3) the brain stem, which, like u, directly controls actions, in response to what is known about the external environment. There are further correspondences and features to improve performances of this system far beyond the scope of this paper.

Rationale of the Approximations Used in GDHP

The approximations used by GDHP require some explanation.

First, consider the use of specific functional forms instead of general functions J and 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 forms, 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 u(x).

There are several obvious problems in assuming a functional form for J*: (1) the true J which obeys Howard's equation will usually not be expressable exactly as J*(a) for any a; (2) the need for a user or metaprogram to choose the functional form; (3) the problem of choosing weights W_i; (4) dangers of autocorrelation invalidating the adjustment process; and (5) the need to worry about robustness of the results, influential observations and unobserved variables. All these problems are precisely parallel to known problems in using statistical forecasting models 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 GHDP are very similar to those used in statistics.

Reasons for choosing E as a loss function for J* are discussed in the EIA report: (1) the derivatives of J are the "shadow prices" or "Lagrange multipliers"; (2) decisions (u) are based on local comparisons, such that the accuracy of the derivatives of J determine the accuracy of the decisions; (3) this eliminates Howard's "U" constant; (4) other reasons are mentioned in Policy Analysis and Information Systems (Elsevier), 6/79.

Implementation of GDHP

The major difficulty in implementing GDHP is the difficulty of performing the minimization and maximization described above. In the general case, where no strong special assumptions are made about J and F and u, and where there are many parameters in these functions, this requires that we get the derivatives of the quantities to be minimized or maximized. Here, however, E already involves first derivatives; the derivatives of E involve second derivatives. The EIA report derives in detail the calculationswhich yield the required derivatives. These calculations only cost about three or four times what it costs to compute J(F(x,u(x))) itself. If the functional forms for J, F and u are "realizable," this means that the cost of calculating J(F(x,u(x))) is bearable; the cost of obtaining derivatives, then, is also bearable for all realizable J, F and u. The EIA report explicitly spells out the details in the case where multiple processors can be used to reduce the cost of calculating J, F and u; it demonstrates that the cost of getting derivatives can be kept in line with that of getting J, F and u even in that case.

Minimization and maximization are nontrivial problems, even with the derivatives available. "Sparse quasiNewtonian" methods are being developed, however, which show promise in handling large systems. EIA has a crude but adequate method which it now uses to solve 100,000 nonlinear simultaneous equations.

In a real-time system, one would want to carry out all the iteration processes above in parallel rather than wait, for example, to complete a minimization before getting new data. For analytical purposes, however, it is convenient for now to consider decision problems focused at one moment in time.

Implications of GDHP Feasibility

This approach makes it possible to develop generalized, adaptive artifical intelligence, capable of achieving results comparable to what is discussed in science fiction, by a rational development of statistics, optimization theory and numerical analysis in the directions indicated above. The implications for psychology and economics raise issues too complex to permit adequate discussion here.