NBSIR 85-3222

Polyadic Third-Order Lagrangian Tensor Structure and Second-Order Sensitivity Analysis With Factorable Functions

Richard H. F. Jackson

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Applied Mathematics
Gaithersburg, MD 20899

and

Garth P. McCormick

Department of Operations Research
School of Engineering and Applied Science
George Washington University
Washington, DC 20052

August 1985

Sponsored by
U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
...or Applied Mathematics
...ns Research Division
...burg, MD 20899

NBSIR 85-3222

POLYADIC THIRD-ORDER LAGRANGIAN TENSOR STRUCTURE AND SECOND-ORDER SENSITIVITY ANALYSIS WITH FACTORABLE FUNCTIONS

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Applied Mathematics
Gaithersburg, MD 20899

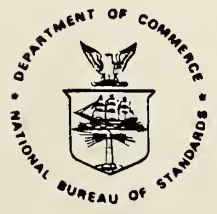
and

Garth P. McCormick

Department of Operations Research
School of Engineering and Applied Science
George Washington University
Washington, DC 20052

August 1985

Sponsored by
U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Applied Mathematics
Operations Research Division
Gaithersburg, MD 20899



U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

ABSTRACT

Second-order sensitivity analysis methods are developed for analyzing the behavior of a local solution to a constrained nonlinear optimization problem when the problem functions are perturbed slightly. Specifically, formulas involving third-order tensors are given to compute second derivatives of components of the local solution with respect to the problem parameters. When in addition, the problem functions are factorable, it is shown that the resulting tensors are polyadic in nature.

Keywords: Second-order sensitivity analysis, high-order methods, N^{th} derivatives, polyads, tensors.

Table of Contents

List of Tables

List of Figures

1. INTRODUCTION

In an earlier paper (Jackson and McCormick (1984)) the structure taken by N-dimensional arrays of N^{th} partial derivatives of the special class of factorable functions was examined. The N-dimensional arrays (or tensors as they are sometimes called) turn out to be computable naturally as the sum of generalized outer product matrices (polyads). For the benefit of the reader unfamiliar with polyads and factorable functions, some of the material in that paper is repeated here.

This natural polyadic structure has important computational implications for solving problems associated with nonlinear programming. It means for example that with some minor modification to existing software routines, high-order derivatives can be calculated efficiently, making previously intractable techniques that require them, again worthy of consideration. In the dissertation by Jackson (1983) from which most of the material in this paper is taken, these implications were pursued for second-order sensitivity analysis and high-order methods for solving the problem:

$$\begin{aligned} &\underset{x \in R^n}{\text{minimize}} \quad f(x), \\ &\text{subject to } g(x) \geq 0, \end{aligned} \tag{1.1}$$

for i=1, ..., m, when f(x) and g(x) are factorable functions.

The ability to compute third derivatives efficiently provides ready access to second-order nonlinear programming sensitivity information. In Section 3 of this paper, second-order sensitivity analysis methods are developed for analyzing the behavior of a local solution to (1.1) when the problem functions are perturbed slightly. Section 3 begins by summarizing results from first-order sensitivity analysis which provide formulas for the first derivatives of the components of the local solution with respect to the problem parameters. Also developed are formulas, involving third-order

tensors, for computing the second derivatives of the local solution with respect to the problem parameters. In addition, the polyadic structure of the tensors is investigated and displayed, and techniques for manipulating these three-dimensional arrays, capitalizing on this special structure, are developed. In general, this type of array manipulation is straightforward but time-consuming and requires significant computer storage. It is shown that these difficulties are ameliorated when the special structure of factorable functions is exploited. Examples of the use of these formulas for estimating the solution to perturbed problems using Taylor series approximations are also given.

Loosely, a factorable function is a multivariable function that can be written as the last of a finite sequence of functions, in which the first n functions in the sequence are just the coordinate variables, and each function beyond the n^{th} is a sum, a product, or a single-variable transformation of previous functions in the sequence. More rigorously, let $[f_1(x), f_2(x), \ldots, f_L(x)]$ be a finite sequence of functions such that $f_i: R^n \to R$, where each $f_i(x)$ is defined according to one of the following rules.

Rule 1. For $i=1, \ldots, n$, $f_i(x)$ is the value of the i^{th} Euclidean coordinate:

$$f_i(x) = x_i.$$

Rule 2. For $i=n+1, \ldots, L$, $f_i(x)$ is formed using one of the following compositions:

$$\begin{aligned} &\text{a.)} \quad f_i(x) = f_{j(i)}(x) + f_{k(i)}(x); \text{ or} \\ &\text{b.)} \quad f_i(x) = f_{j(i)}(x) \cdot f_{k(i)}(x); \text{ or} \\ &\text{c.)} \quad f_i(x) = T_i[f_{j(i)}(x)]; \end{aligned} \tag{1.2}$$

where $j(i) < i$, $k(i) < i$, and T_i is a function of a single variable. Then $f(x) = f_L(x)$ is a factorable function and $[f_1(x), f_2(x), \ldots, f_L(x)]$ is a

factored sequence. Thus a function, f(x), will be called factorable if it can be formed according to Rules 1 and 2, and the resulting sequence of functions will be called a factored sequence or, at times, the function written in factored form.

Although it is not always immediately grasped, the concept of a factorable function is actually a very natural one. In fact, it is just a formalization of the natural procedure one follows in evaluating a complicated function. Consider for example the function

$$f(x) = a^T x[\sin b^T x][\exp c^T x], \tag{1.3}$$

where a,b,c and x are (2 x 1) vectors. The natural approach to evaluating this function for specific values of x_1 and x_2 is first to compute the quantities within the parentheses, then to apply the sine and exponential functions, and finally to multiply the three resulting quantities. This might be done in stages as follows.

$$\begin{aligned}
f_1 &= x_1 & f_8 &= f_6 + f_7 \\
f_2 &= x_2 & f_9 &= c_1 f_1 \\
f_3 &= a_1 f_1 & f_{10} &= c_2 f_2 \\
f_4 &= a_2 f_2 & f_{11} &= f_9 + f_{10} \\
f_5 &= f_3 + f_4 & f_{12} &= \sin(f_8) \\
f_6 &= b_1 f_1 & f_{13} &= \exp(f_{11}) \\
f_7 &= b_2 f_2 & f_{14} &= f_5 \cdot f_{12} \\
& & f_{15} &= f_{13} \cdot f_{14}
\end{aligned} \tag{1.4}$$

This is one possible factored sequence for the function in (1.3).

For completeness, we also note here that a Factorable Programming problem may be compactly (though less transparently) written as

$$\underset{x \in R^n}{\text{minimize}} \quad f_L(x)$$

$$\text{subject to } \ell_i \leq f_i(x) \leq u_i,$$

for $i = 1, \ldots, L-1$, where it is possible that $\ell_i = -\infty$ and/or $u_i = +\infty$, and where $f_i(x) = x_i$, for $i \leq n$, and $f_i(x)$ is defined recursively for $i > n$ as

$$f_i(x) = \sum_{p=1}^{i-1} U_p^i[f_p(x)] + \sum_{p=1}^{i-1} \sum_{q=1}^{p} V_{q,p}^i[f_p(x)]\, W_{p,q}^i[f_q(x)],$$

where U, V, and W are functions of a single variable.

In order to appreciate fully the value of factorable functions the concept of an outer product matrix must be introduced. An (m x n) matrix A is called an <u>outer product matrix</u> if there exists a scalar α, an (m x 1) vector a, and an (n x 1) vector b such that

$$A = a\alpha b^T.$$

The expression $a\alpha b^T$ is called an <u>outer product</u> or a <u>dyad</u>. Note that a dyad is conformable since the dimensions of the product are (m x 1)(1 x 1)(1 x n), which yields the (m x n) outer product matrix A as desired. A useful property of outer product matrices is that, if kept as dyads, matrix multiplication with them is simplified to inner products alone, saving the computations required to form the matrices involved. For example,

$$Ac = a\alpha[b^T c],$$

$$d^T A = [d^T a]\alpha b^T, \text{ and}$$

$$AF = a\alpha[b^T F],$$

where c is (n x 1), d is (m x 1) and F is (n x m).

It is well-known (see for instance McCormick (1983)) that factorable functions possess two very special properties that can be exploited to produce efficient (fast and accurate) algorithms: i) once written in factorable form, their gradients and Hessians may be computed exactly, automatically, and efficiently; and ii) their Hessians occur naturally as sums of dyads whose vector factors are gradients of terms in the factored sequence. The first of these properties has eased the task of providing the derivatives of a nonlinear programming problem to a computer code that solves it, and has the potential eventually to trivialize it. The second has obviated the task of multiplying a matrix by a vector, reducing it to a series of inner products, as noted above, which in many cases results in less effort.

Since the discovery of factorable functions, the theory of Factorable Programming has been further developed and refined in a variety of ways. Ghaemi and McCormick (1970) developed a computer code (FACSUMT), which processes the functions in a factorable program and provides the interface to the SUMT nonlinear programming code developed originally by Mylander et al. (1971). An early version of this code is described in Pugh (1972). Recently the routines from FACSUMT that process the problem functions in factorable form have been separated out into a "stand-alone" package (FACTIN) that can be used with any nonlinear programming system to provide automatically the values of the functions, gradients and Hessians at a point. The basic requirement of FACTIN is that the user write the functions of the problem in factorable form.

Once the problem is written in factorable form, the routines in FACTIN automatically calculate the exact first and second derivatives of the functions for use in the optimization algorithm.

It is important to understand that the derivative calculations performed by the FACTIN code are not estimations, but mathematically exact calculations. Furthermore, they are also compact, since factored sequences mimic hand calculations, and thus this technique is different from symbolic manipulation techniques for differentiation, which tend to produce large amounts of code. The techniques used in Factorable Programming are efficient exploitations of the special structure inherent in factorable functions and their partial derivative arrays. Moreover, while it is true that some symbolic differentiaters also can recognize functions which can be described similarly as a sequence of rules, each of which can be differentiated, the similarity ends there. Such symbolic differentiaters continue to differentiate the rules, without exploiting the polyadic structure of the result. See,e.g., Kedem (1980), Rall (1980), Wengert (1964), Reiter and Gray (1967), Hillstrom (1982), and Warner (1975). It is this latter effort which provides the real value of factorable functions, and which therefore separates the two techniques. More on Factorable Programming codes and the Factorable Programming system (FACTPROG) under development at the National Bureau of Standards is given in Jackson and McCormick (1984).

Further extensions of Factorable Programming theory were provided by Shayan (1978), who developed an automatic method for computing the m^{th}-order directional derivative of a factorable function and noted that the efficiency of a solution technique can be evaluated when the functions are factorable by counting basic operations and basic functions. This is a more accurate measure of efficiency than the technique of counting the number of "equivalent function evaluations" discussed by Miele and Gonzalez (1978).

As was mentioned earlier, Hessians of factorable functions possess a natural dyadic structure which can be exploited. This structure was used in Emami (1978) to develop a matrix factorization scheme for obtaining a generalized inverse of the Hessian of a factorable function. Ghotb (1980) also capitalized on this structure and provided formulae for computing a generalized inverse of a reduced Hessian when it is given in dyadic form. Sofer (1983) has extended this last concept further, by utilizing the dyadic structure to obtain computationally efficient techniques for constructing a generalized inverse of a reduced Hessian and updating it from iteration to iteration.

Another direction was pursued by DeSilva and McCormick (1978), who developed the formulae and methodology to utilize the input to general nonlinear programs in factorable form to perform first-order sensitivity analysis on the solution vector.

Some last comments on notation are required. There are unavoidable complications in the theory that follows that require subscripted subscripts. In some cases these are used. In other cases, subscripted subscripts are replaced with subscript functions. For example, $i_j \rightarrow i(j)$. The choice in each case was made on the basis of clarity of resulting formulae. Also in what follows, all vectors are assumed to be column vectors, and, where not otherwise stated, differentiation is with respect to the vector $x = (x_1, x_2, \ldots, x_n)^T$. Lastly, we use ∂ and ∇ to indicate partial differentiation, and d and D indicate total differentiation.

2. THE SPECIAL STRUCTURE OF TENSORS OF FACTORABLE FUNCTIONS

2.1 Background and Notation

One fundamental value of factorable functions lies in the simple and computationally efficient forms that result for their Hessians. In fact factorable programming is based on the existence of, and the simplified operations that result from, these simple forms. The seminal result, (Fiacco and McCormick (1968), pp 184-188), is that the Hessian of a factorable function can be written as the sum of dyads, or outer products, of gradients of functions in the factored sequence. As was shown in Jackson and McCormick (1984) and will be summarized here, this basic result may be generalized, but first it is necesssary to generalize the concepts of Hessian and dyad.

Let $A \in R^{(n_1 \times \cdots \times n_N)}$, and let $A_{i_1,\ldots,i_N}$ denote the $(i_1, \ldots, i_N)^{th}$ element of this array. For the purposes of this paper, A is called the N^{th}-order tensor of a multivariable function f(x) if

$$A_{i_1,\ldots,i_N} = \partial^N f(x)/\partial x_{i_N} \ldots \partial x_{i_1}.$$

Note that gradients and Hessians are tensors of order 1 and 2 respectively.

An N-dimensional array A is called a generalized outer product matrix if there exists a scalar α, and an ordered set of vectors $a_1, \ldots, a_N$ (where each a_k is $(n_k \times 1)$) such that each element of A is generated by the product of the scalar α and certain specific elements of the vectors $a_1, \ldots, a_N$ as follows

$$A_{i_1,\ldots,i_N} = \alpha * a_{1,i_1} * \cdots * a_{N,i_N},$$

for $i_1 = 1, \ldots, n_1; \ldots; i_N = 1, \ldots, n_N$, where a_{k,i_k} represents the $(i_k)^{th}$ element of the $(n_k \times 1)$ vector a_k.

The scalar and set of vectors which generate a generalized outer product matrix taken together are called a polyad and are written

$$(\alpha:a_1 \cdots a_N). \tag{2.1}$$

where order is important, i.e. the vector in position j is associated with the j^{th} dimension. A polyad contafning N vector factors is called an N-ad. Also, an expression containing a sum of polyads is called a polyadic, and an expression containing a sum of N-ads is called an N-adic. (The actual addition here is performed as a direct sum of the associated generalized outer product matrices.) When vector factors in a polyad are repeated, exponential notation is used, as, e.g., in the case of the symmetric N-ad, $(\alpha:[a]^N)$. Note that the representation of a generalized outer product matrix by a polyad is not unique. For example, $(\alpha/\gamma:[a_1\gamma] \cdots a_N)$ generates the same N-dimensional array of numbers as does (2.1) for any nonzero scalar γ. Finally, note that a 2-ad of the form $(\alpha:ab)$ is equivalent to the more familiar dyad of the form $a\alpha b^T$, and the two will be used interchangeably.

Because the concepts of generalized outer product matrix and polyad are so fundamental to what follows, an example is included to help in understanding them. Consider first the dyad $(10:a_1a_2)$ where $a_1 = (4,3,2,1)^T$, and $a_2 = (3,2,1)^T$. The generalized outer product matrix of order 2 generated by this dyad is the (4 x 3) array

$$\begin{bmatrix} 120 & 80 & 40 \\ 90 & 60 & 30 \\ 60 & 40 & 20 \\ 30 & 20 & 10 \end{bmatrix}$$

If the vector $a_3 = (2,1)^T$ is added to form the triad $(10:a_1a_2a_3)$, the result

is a generalized outer product matrix of order 3 with dimensions (4 x 3 x 2). To form this three-dimensional array, the outer product between the matrix above and the vector a_3 is formed. Thus the matrix above is multiplied by 2 to obtain the front matrix in the three-dimensional array and by 1 to obtain the back matrix. These are:

$$\begin{bmatrix} 240 & 160 & 80 \\ 180 & 120 & 60 \\ 120 & 80 & 40 \\ 60 & 40 & 20 \end{bmatrix} \qquad \begin{bmatrix} 120 & 80 & 40 \\ 90 & 60 & 30 \\ 60 & 40 & 20 \\ 30 & 20 & 10 \end{bmatrix}$$

Front Back

2.2 The First and Second Order Cases

In this section, the special polyadic structure of the gradient and the Hessian of a factorable function is exhibited.

Theorem 1. (Monadic Gradients)

Let $f(x)$ be a factorable function in R^n, let $[f_1(x), f_2(x), \ldots, f_L(x)]$ be a factored sequence for $f(x)$, and assume that all functions are once continuously differentiable. Then the gradient $\nabla f(x) = \nabla f_L(x)$ can be written as a sum of outer product matrices of the form $(\alpha:a_1)$, where a_1 is a gradient of a factored-sequence function and the scalar α is composed of a product of factored-sequence functions and first derivatives of the single-variable transformations, T_i, used in the factored sequence.

Proof. See Jackson (1983).

The result given in Theorem 2 below is a formalization of a result which appeared without proof in McCormick (1983).

Theorem 2. (Dyadic Hessians)

Let $f(x)$ be a factorable function in R^n, let $[f_1(x), f_2(x), \ldots, f_L(x)]$ be a factored sequence for $f(x)$, and assume that all functions are twice continuously differentiable. Then the Hessian $\nabla^2 f(x) = \nabla^2 f_L(x)$ can be written as a sum of outer product matrices of the form $(\alpha:a_1a_2)$, where a_1 and a_2 are gradients of factored-sequence functions and the scalar α is composed of a product of factored-sequence functions and first and second derivatives of the single-variable transformations, T_i, used in the factored sequence.

Proof. See Jackson (1983).

Although the proofs of Theorems 1 and 2 are not included, the monadic and dyadic structure of the gradient and Hessian of a factorable function are exhibited by displaying the gradients and Hessians of the forms in (1.2) as in Tables 1 and 2.

In order to clarify these concepts, consider again the illustrative function in (1.3). Table 3 is a display of the gradient and Hessian of this function. The entries in each column are the summands in the expressions for the gradient and Hessian. For example,

$$\nabla f = [\sin[b^Tx]][\exp[c^Tx]]a + [a^Tx][\cos[b^Tx]][\exp[cTx]]b + [a^Tx][\sin[b^Tx]][\exp[c^Tx]]c,$$

or, in polyadic notation

$$\nabla f = ([\sin[b^Tx]][\exp[c^Tx]]:a) + ([a^Tx][\cos[b^Tx]][\exp[cTx]]:b) + ([a^Tx][\sin[b^Tx]][\exp[c^Tx]]:c).$$

The table also illustrates the left-to-right, tree-like structure of the derivatives involved. From the table it can be seen that both the gradient and Hessian naturally have the polyadic structure discussed above. Notice too that the vectors in the monads and dyads are drawn from the set {a,b,c}, each of which is the gradient of a factored sequence function in (1.4).

TABLE 1

GRADIENTS OF FACTORABLE FUNCTION FORMS*

Rule	f_i	∇f_i
1	x_i	e_i
2a	$f_{j(i)} + f_{k(i)}$	$\nabla f_{j(i)} + \nabla f_{k(i)}$
2b	$f_{j(i)} \cdot f_{k(i)}$	$\nabla f_{k(i)} f_{j(i)} + \nabla f_{j(i)} f_{k(i)}$
2c	$T_i[f_{j(i)}]$	$\nabla f_{j(i)} \dot{T}_i[f_{j(i)}]$

TABLE 2

HESSIANS OF FACTORABLE FUNCTION FORMS*

Rule	f_i	$\nabla^2 f_i$
1	x_i	0_{nxn}
2a	$f_{j(i)} + f_{k(i)}$	$\nabla^2 f_{j(i)} + \nabla^2 f_{k(i)}$
2b	$f_{j(i)} \cdot f_{k(i)}$	$f_{j(i)} \nabla^2 f_{k(i)} + \nabla f_{k(i)} \nabla^T f_{j(i)}$ $+ f_{k(i)} \nabla^2 f_{j(i)} + \nabla f_{j(i)} \nabla^T f_{k(i)}$
2c	$T_i[f_{j(i)}]$	$\dot{T}_i[f_{j(i)}] \nabla^2 f_{j(i)} + \nabla f_{j(i)} \ddot{T}_i[f_{j(i)}] \nabla^T f_{j(i)}$

*Notation: $\dot{T}[f] = \partial T/\partial f$, $\ddot{T}[f] = \partial^2 T/\partial f^2$, and e_i = the i^{th} unit vector in R^n.

TABLE 3

MONADIC AND DYADIC TERMS IN GRADIENT
AND HESSIAN OF ILLUSTRATIVE FUNCTION*

f	∇f (summands)	$\nabla^2 f$ (summands)
$a^Tx\sin[b^Tx]\exp[c^Tx]$	$(\sin[b^Tx]\exp[c^Tx]:a)$	$(\cos[b^Tx]\exp[c^Tx]:ab)$ $(\sin[b^Tx]\exp[c^Tx]:ac)$
	$(a^Tx\cos[b^Tx]\exp[c^Tx]:b)$	$(\cos[b^Tx]\exp[c^Tx]:ba)$ $(-a^Tx\sin[b^Tx]\exp[c^Tx]:bb)$ $(a^Tx\cos[b^Tx]\exp[c^Tx]:bc)$
	$(a^Tx\sin[b^Tx]\exp[c^Tx]:c)$	$(\sin[b^Tx]\exp[c^Tx]:ca)$ $(a^Tx\cos[b^Tx]\exp[c^Tx]:cb)$ $(a^Tx\sin[b^Tx]\exp[c^Tx]:cc)$

*N.B. Since the meaning is clear, we have dropped a level of parentheses in these expressions to be able to fit the table on one page. The same comment holds for Table 4.

2.3 The N^{th}-Order Case

The next theorem is fundamental to operations with higher derivatives of factorable functions, and provides the necessary tool to use in proving that all tensors of factorable functions are polyadics in gradients of factored-sequence functions.

Theorem 3. (Differentiation of Polyads)

Let $f(x)$ be a factorable function in R^n, let $[f_1(x), f_2(x), \ldots, f_L(x)]$ be its factored sequence, and assume that all functions are members of C^{N+1}. Consider the N-ad

$$(\alpha : a_1 a_2 \ldots a_N),$$

where the a_j are gradients of factored-sequence functions and α is a scalar composed in general of a product of factored-sequence functions and derivatives (no higher than order N) of the single-variable transformations used in forming the factored sequence. The gradient of this N-ad is the sum of (N+1)-ads, each term of which has the structure defined above.

Proof. Because the proof of this theorem illustrates the technique of polyadic differentiation, its salient features are included here. For convenience, write f_i for $f_i(x)$. Then the scalar α is of the form

$$\alpha = \prod_{\ell} \eta_\ell,$$

where

$$\eta_\ell = f_{r(\ell)} \quad \text{or} \quad \partial^{k(\ell)}\{T_{s(\ell)}[f_{r(\ell)}]\}/\partial f_{r(\ell)}^{k(\ell)},$$

for values of $r(\ell)$, $s(\ell)$, and $k(\ell)$ which respectively define, for the ℓ^{th} factor of α, the factored-sequence function, the transformation, and the

level of the derivative of the transformation used. Note that $k(\ell) \leq N$. By the chain rule of differentiation,

$$\nabla\alpha = \sum_{\ell} \gamma_\ell c_\ell,$$

where

$$\gamma_\ell = \frac{\alpha}{\eta_\ell}, \quad \underline{\text{or}} \quad \frac{\alpha}{\eta_\ell} \cdot \partial^{k(\ell)+1}[T_{s(\ell)}[f_{r(\ell)}]]/\partial f_{r(\ell)}^{k(\ell)+1},$$

and

$$c_\ell = \nabla f_{r(\ell)}.$$

Furthermore, since the a_j in the N-ad are gradients of factored-sequence functions, ∇a_j can be written by the previous theorem as the sum of outer product matrices of order 2 in gradients of factored-sequence functions as follows:

$$\nabla a_j = \sum_{(p,q)\in I_j} b_p \beta_{pq} b_q^T = \sum_{(p,q)\in I_j} (\beta_{pq} : b_p b_q),$$

where $I_j = \{(p,q) \mid b_p \beta_{pq} b_q^T$ is a term in the dyadic representation of $\nabla a_j\}$ and $\beta_{pq} = \beta_{qp}$, is of the same form as α. Then, by straightforward term-by-term differention and collection of terms (see Jackson and McCormick (1984)), it can be shown that the gradient of the N-ad is

$$\begin{aligned}\nabla(\alpha : a_1 a_2 \cdots a_N) = & \sum_{\ell} (\gamma_\ell : a_1 \cdots a_N c_\ell) \\ & + \sum_{(p,q)\in I_1} (\alpha\beta_{pq} : b_p \cdots a_N b_q) \\ & \vdots \\ & + \sum_{(p,q)\in I_N} (\alpha\beta_{pq} : a_1 \cdots a_{N-1} b_p b_q),\end{aligned}$$

which is a sum of (N+1)-ads in gradients of factored sequence functions with leading scalars of the proper form, and the result is proven.

Now the main theorem on the structure of high-order tensors of factorable functions can be given.

Theorem 4. (Polyadic Tensors)

Let $f(x)$ be a factorable function in R^n, let $[f_1(x), f_2(x), \ldots, f_L(x)]$ be a factored sequence for $f(x)$, and assume that $f(x) \in C^N$ and $f_i(x) \in C^N$, for $i=1, \ldots, L$. Then the N^{th}-order tensor of $f(x)$ can be written as the sum of generalized outer product matrices of the following special form. Let $(\alpha:a_1 \cdots a_N)$ be a polyad associated with one of the outer product matrices. Then each a_k is the gradient of some function in the factored sequence, and the scalar α is a product of functions in the factored sequence and derivatives of the single-variable transformations used in defining the functions in the sequence. Only derivatives $\partial^k T[f]/\partial f^k$, for $1 \leq k \leq N$, are used.

Proof. The proof is a straightforward application of Theorem 3 and induction; the reader is referred to Jackson and McCormick (1984) for the details.

At this point another example is offered to help illustrate these ideas. To keep matters as simple as possible, the same illustrative function in (1.3), whose gradient and Hessian were displayed in Table 3, will be used. Calculation of the third-order tensor of this function is a therapeutic exercise, but only the final result is given here in Table 4. The terms in the Hessian from Table 3 are also included in Table 4 to make it easier to see the left-to-right, tree-like connections among successive gradient terms. Notice that the vectors that make up each triad are once again drawn from the set {a,b,c} and that the scalar multiples in the triads are products of factored-sequence functions (e.g., a^Tx, $\sin[b^Tx]$, $\exp[c^Tx]$) and first and second derivatives of the transformations (e.g., $\cos[b^Tx]$, $-\sin[b^Tx]$). Notice

TABLE 4

DYADIC AND TRIADIC TERMS IN HESSIAN AND THIRD ORDER TENSOR OF ILLUSTRATIVE FUNCTION

$\nabla^2 f$ (summands)	$\nabla^3 f$ (summands)
$(\cos[b^T x]\exp[c^T x]:ab)$	$(-\sin[b^T x]\exp[c^T x]:abb)$ $(\cos[b^T x]\exp[c^T x]:abc)$
$(\sin[b^T x]\exp[c^T x]:ac)$	$(\cos[b^T x]\exp[c^T x]:acb)$ $(\sin[b^T x]\exp[c^T x]:acc)$
$(\cos[b^T x]\exp[c^T x]:ba)$	$(-\sin[b^T x]\exp[c^T x]:bab)$ $(\cos[b^T x]\exp[c^T x]:bac)$
$(-a^T x\sin[b^T x]\exp[c^T x]:bb)$	$(-\sin[b^T x]\exp[c^T x]:bba)$ $(-a^T x\cos[b^T x]\exp[c^T x]:bbb)$ $(-a^T x\sin[b^T x]\exp[c^T x]:bbc)$
$(a^T x\cos[b^T x]\exp[c^T x]:bc)$	$(\cos[b^T x]\exp[c^T x]:bca)$ $(-a^T x\sin[b^T x]\exp[c^T x]:bcb)$ $(-a^T x\cos[b^T x]\exp[c^T x]:bcc)$
$(\sin[b^T x]\exp[c^T x]:ca)$	$(\cos[b^T x]\exp[c^T x]:cab)$ $(\sin[b^T x]\exp[c^T x]:cac)$
$(a^T x\cos[b^T x]\exp[c^T x]:cb)$	$(\cos[b^T x]\exp[c^T x]:cba)$ $(-a^T x\sin[b^T x]\exp[c^T x]:cbb)$ $(-a^T x\cos[b^T x]\exp[c^T x]:cbc)$
$(a^T x\sin[b^T x]\exp[c^T x]:cc)$	$(\sin[b^T x]\exp[c^T x]:cca)$ $(-a^T x\cos[b^T x]\exp[c^T x]:ccb)$ $(a^T x\cos[b^T x]\exp[c^T x]:ccc)$

too that the only new calculations needed to form $\nabla^3 f$ are the third derivatives of the single variable transformations: sin, cos, and exp.

There are several computational advantages to be stressed here. One advantage is a result of the symmetry of partial derivatives, i.e., $\partial^3 f/\partial x_i \partial x_j \partial x_k = \partial^3 f/\partial x_j \partial x_k \partial x_i$, etc. Because of this symmetry, every triad in the $\nabla^3 f$ column in Table 4 whose vector factors are the same, disregarding order, has the same associated scalar. For example, abb, bba, and bab, all have the same associated scalar: $[-\sin[\, b^T x]][\exp[c^T x]]$. Thus only six distinct scalars need to be computed to form $\nabla^3 f$. Moreover, because of what may be called the "persistence property" of the derivatives of sin and exp in this example , the calculation of these six scalars is also simplified. That is, all derivatives of exp are equal and alternating derivatives of sin are equivalent except for their signs. This too can be exploited to reduce computational effort. Finally, since the vector factors in the triads are members of the same set from which the dyads in the Hessian and the monads in the gradient are formed, a computer code that calculates these need only store a, b, and c, and a set of pointers for each monad, dyad, and triad indicating which vector is required for each position. Of course, these pointers can be packed to save storage.

The current versions of the factorable programming input routines (FACSUMT and FACTIN) take advantage of each of the above in computing the gradients and Hessians. Work is underway to extend these codes to capitalize on the polyadic structure of factorable function tensors and compute high-order derivatives in the efficient manner discussed above. See Jackson and McCormick (1984) for more on this current work.

3. SECOND-ORDER SENSITIVITY ANALYSIS IN NONLINEAR PROGRAMMING

3.1 Basic First-Order Sensitivity Results

One application of the results in Section 2 is in obtaining high-order sensitivity information for nonlinear programming problems, although only the second-order case is considered in this paper. Sensitivity analysis in nonlinear programming is concerned with analyzing the behavior of a local solution when the problem functions are perturbed slightly. This perturbation might be due to an inexactness with which certain parameter values in the problem are calculated or because the optimization model was parameterized and one is interested in the solution for a variety of values of the parameters. For additional information on this topic, see Armacost and Fiacco (1974), Armacost and Fiacco (1978), Fiacco (1980) and especially Fiacco (1983) and its excellent bibliography. The parametric problem is written

$$\begin{aligned} P(\varepsilon): \quad & \underset{x \in R^n}{\text{minimize}} \quad f(x,\varepsilon), \\ & \text{subject to } g_i(x,\varepsilon) \geq 0, \end{aligned} \tag{3.1}$$

for $i=1, \ldots, m$, where ε is an $(r \times 1)$ vector of parameters. The more general version of the sensitivity problem includes equality constraints, but these are not included here for simplicity. The ideas and results presented in this section generalize readily to the equality-constrained problem.

The essence of sensitivity analysis in nonlinear programming is the application of the Implicit Function Theorem (see, e.g. Bliss (1946) to the Karush-Kuhn-Tucker (KKT) necessary conditions for the problem, $P(\varepsilon)$, in (3.1).

First, define the Lagrangian for $P(\varepsilon)$ as

$$L(x,u,\varepsilon) = f(x,\varepsilon) - \sum_{i=1}^{m} u_i g_i(x,\varepsilon).$$

Then, assuming continuous differentiability in x of the problem functions, the KKT conditions for $P(\varepsilon)$ are that there exists a feasible point, x, for (3.1) and associated vector of Lagrange multipliers, u, such that

$$\begin{aligned} \nabla L(x,u,\varepsilon) &= 0, \\ u_i g_i(x,\varepsilon) &= 0, \\ u_i &\geq 0, \end{aligned} \tag{3.2}$$

for $i = 1, \ldots, m$.

The statement of the first-order sensitivity results given in Theorem 6 below also requires that the second-order sufficient conditions (SOSC) hold at a particular solution $\hat{x}$, of $P(\hat{\varepsilon})$ (which is just (3.1) for a specified vector of parameter values, $\hat{\varepsilon}$). These conditions may be written as follows.

Theorem 5. (SOSC)

Let $\hat{x}$ be a feasible point for $P(\hat{\varepsilon})$ and assume that the functions of $P(\hat{\varepsilon})$ are twice-continuously differentiable in x in a neighborhood of $\hat{x}$. Let $(\hat{x},\hat{u},\hat{\varepsilon})$ be a triple that satisfies the KKT conditions in (3.2) and define

$$B = \{i \mid g_i(\hat{x},\hat{\varepsilon}) = 0\},$$

and

$$D = \{i \mid \hat{u}_i > 0\}.$$

Further, suppose that

$$d^T \nabla^2 L(\hat{x},\hat{u},\hat{\varepsilon}) d > 0,$$

for all $d \neq 0$, such that

$$d^T \nabla g_i(\hat{x},\hat{\varepsilon}) \geq 0, \quad \text{for all } i \in B,$$

and

$$d^T \nabla g_i(\hat{x},\hat{\varepsilon}) = 0, \quad \text{for all } i \in D.$$

Then $\hat{x}$ is a strict local minimizer for $P(\hat{\varepsilon})$.

The earliest known reference to these conditions is Pennisi (1953), although it was almost 15 years (see McCormick (1967), and Fiacco and McCormick (1968)) before they were more fully developed and exploited.

The following theorem can be viewed as the basic result in nonlinear programming sensitivity analysis.

Theorem 6. (First-Order Sensitivity)

Let $\hat{x}$ be a feasible point for $P(\hat{\varepsilon})$ and assume that

i.) the functions in (3.1) are twice-continuously differentiable in x and the cross-partial derivatives exist and are jointly continuous in x and ε in a neighborhood of $(\hat{x},\hat{\varepsilon})$;

ii.) the second-order sufficient conditions (Theorem 5) for a local minimum of $P(\hat{\varepsilon})$ hold at $\hat{x}$, with associated Lagrange multipliers $\hat{u}$,

iii.) the gradients of the binding constraints, i.e., $\nabla g_i(\hat{x},\hat{\varepsilon})$, $i \in B$, are linearly independent; and

iv.) $\hat{u}_i > 0$ for all $i \in B$.

Then, for ε in asufficiently small neighborhood of $\hat{\varepsilon}$, there exists a unique, once-continuously differentiable vector function

$$y(\varepsilon) = [x(\varepsilon)^T, u(\varepsilon)^T]^T,$$

satisfying the KKT conditions for $P(\varepsilon)$, with

$$y(\hat{\varepsilon}) = [x(\hat{\varepsilon})^T, u(\hat{\varepsilon})^T]^T,$$

such that $x(\varepsilon)$ is a locally unique isolated minimizer for $P(\varepsilon)$. Furthermore, the first partial derivatives of $x(\varepsilon)$ and $u(\varepsilon)$ with respect to ε can be obtained from the equation

$$Y = -M^{-1}N, \tag{3.3}$$

where

$$Y = \nabla_{\varepsilon} y = \begin{bmatrix} \frac{\partial x_1}{\partial \varepsilon_1} & \cdots & \frac{\partial x_1}{\partial \varepsilon_r} \\ \vdots & & \vdots \\ \frac{\partial x_n}{\partial \varepsilon_1} & \cdots & \frac{\partial x_n}{\partial \varepsilon_r} \\ \\ \frac{\partial u_1}{\partial \varepsilon_1} & \cdots & \frac{\partial u_1}{\partial \varepsilon_r} \\ \vdots & & \vdots \\ \frac{\partial u_m}{\partial \varepsilon_1} & \cdots & \frac{\partial u_m}{\partial \varepsilon_r} \end{bmatrix}$$

$$M = \begin{bmatrix} \nabla^2 L & -\nabla g_1 & -\nabla g_2 & \cdots & -\nabla g_m \\ u_1 \nabla^T g_1 & g_1 & 0 & \cdots & 0 \\ u_2 \nabla^T g_2 & 0 & g_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ u_m \nabla^T g_m & 0 & 0 & \cdots & g_m \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} \nabla^2_{\varepsilon x} L \\ u_1 \nabla^T_{\varepsilon} g_1 \\ u_2 \nabla^T_{\varepsilon} g_2 \\ \vdots \\ u_m \nabla^T_{\varepsilon} g_m \end{bmatrix}$$

with all quantities in M and N evaluated at $(\hat{x}^T, \hat{u}^T)^T$.

Proof. See Fiacco (1983).

3.2 Development of the Second-Order Equation

The result in (3.3) provides a direct way of calculating first-order sensitivity information and is the point of departure for the work given here, which begins with the following theorem due to Fiacco (1983).

Theorem 7. (Higher-Order Sensitivity)

Let $\hat{x}$ be a feasible point for $P(\hat{\varepsilon})$ and assume conditions (i) through (iv) in Theorem 6. Assume also that all $(k+1)^{st}$-order partial derivatives in x and all $(k+1)^{st}$-order cross partial derivatives in x and ε exist and are jointly continuous in x and ε in a neighborhood of $(\hat{x},\hat{\varepsilon})$. Then in a sufficiently small neighborhood of $\hat{\varepsilon}$, the vector function $y(\varepsilon)$ is k times continuously differentiable.

Proof. The proof follows directly from the fact that if the Jacobian, M, of the KKT conditions in (3.2) is nonsingular, and if the functions involved possess the appropriate degree of differentiability, the Implicit Function Theorem (see Bliss (1946), p. 270) guarantees the existence of the higher-order partial derivatives. Nonsingularity of M for (3.2) was shown in Fiacco and McCormick (1968). Thus the theorem is proved by direct application of the Implicit Function Theorem.

If this high-order sensitivity information is to be used, it is necessary also to develop a convenient mechanism for calculating it. This is done next for the case when k = 2. Consider the matrix equation in (3.3) and rewrite it as

$$MY = -N.$$

The $(i,j)^{th}$ element of this matricial equation is

$$\sum_s M_{is} Y_{sj} = -N_{ij},$$

and the total derivative of this equation is obtained using the product rule as follows:

$$\sum_s \left\{ M_{is} \frac{\partial Y_{sj}}{\partial \varepsilon_k} + Y_{sj} \frac{dM_{is}}{d\varepsilon_k} \right\} = - \frac{dN_{ij}}{d\varepsilon_k} ,$$

or

$$- \sum_s M_{is} \frac{\partial Y_{sj}}{\partial \varepsilon_k} = \sum_s \frac{dM_{is}}{d\varepsilon_k} Y_{sj} + \frac{dN_{ij}}{d\varepsilon_k} . \qquad (3.4)$$

Next, the chain rule is required in finding the right-hand-side (rhs) in (3.4) since each element of the M and N matrices is a function of each element of x, u, and ε. Hence,

$$\frac{dM_{is}}{d\varepsilon_k} = \sum_t \frac{\partial M_{is}}{\partial y_t} \frac{\partial y_t}{\partial \varepsilon_k} + \frac{\partial M_{is}}{\partial \varepsilon_k} ,$$

and

$$\frac{dN_{ij}}{d\varepsilon_k} = \sum_t \frac{\partial N_{ij}}{\partial y_t} \frac{\partial y_t}{\partial \varepsilon_k} + \frac{\partial N_{ij}}{\partial \varepsilon_k} .$$

Using these and the fact that $Y_{sj} = \partial y_s / \partial \varepsilon_j$, the rhs in (3.4) can thus be written

$$\sum_s \left[\sum_t \frac{\partial M_{is}}{\partial y_t} Y_{tk} + \frac{\partial M_{is}}{\partial \varepsilon_k} \right] Y_{sj} + \sum_t \frac{\partial N_{ij}}{\partial y_t} Y_{tk} + \frac{\partial N_{ij}}{\partial \varepsilon_k} .$$

Now consider the left-hand-side (lhs) in (3.4) more closely. It can be written

$$- \sum_s M_{is} \frac{\partial^2 y_s}{\partial \varepsilon_k \partial \varepsilon_j} ,$$

which is of the form

$$- \sum_s M_{is} A_{sjk},$$

which in turn gives the $(i,j,k)^{th}$ element of the three-dimensional array that results from the matrix multiplication of M and the k^{th} (counting from front to rear) two-dimensional matrix of A that is parallel to the xz-plane in R^3.

Observe that $\partial^2 y_i / \partial\varepsilon_k \partial\varepsilon_j$ can be obtained by premultiplying both sides of (3.4) by $H = M^{-1}$. Then

$$\frac{\partial^2 y_i}{\partial\varepsilon_j \partial\varepsilon_k} = -\sum_r H_{ir} \left\{ \sum_s \left[\sum_t \frac{\partial M_{rs}}{\partial y_t} \frac{\partial y_t}{\partial\varepsilon_k} + \frac{\partial M_{rs}}{\partial\varepsilon_k} \right] \frac{\partial y_s}{\partial\varepsilon_j} + \sum_t \frac{\partial N_{rj}}{\partial y_t} \frac{\partial y_t}{\partial\varepsilon_k} + \frac{\partial N_{rj}}{\partial\varepsilon_k} \right\}, \quad (3.5)$$

which is the $(i,j,k)^{th}$ element of the three-dimensional array of second partial derivatives of $y = [x(\varepsilon)^T, u(\varepsilon)^T]^T$ with respect to ε.

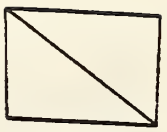
It is desired next to write (3.5) using array notation. Because these operations are performed in three-space, however, some new notation must be developed before this is rewritten. In order to motivate the new notation, consider a multivariable function $f(x)$. It is perhaps clear in this case what is meant by ∇f, $\nabla^2 f$ and $\nabla^3 f$; i.e., ∇f is a vector that is written down the page, $\nabla^2 f$ requires taking the gradient of each element of ∇f and writing that result across the page to form a two-dimensional matrix , and thus to form $\nabla^3 f$ one would take the gradient of each element of $\nabla^2 f$ and write that result into the third dimension (into the page, say). Hence if M is a matrix, it should be clear what is meant by ∇M; i.e., take the gradient of each element of M and write the result into the third dimension.

But, if y is a vector in R^2, there are three possible orientations parallel to the coordinate planes in R^3 for the matrix usually notated as ∇y. These are shown* in Figure 1. In order clearly to differentiate among these, the notation "$\nabla\{\cdot\}$" will be used. Thus $\nabla\{\cdot\}$ operates on the argument by taking its gradient into the third dimension. Note that

$$\nabla y \neq \nabla\{y\} \neq \nabla\{y^T\}.$$

Whereas the elements of these vectors are the same, their orientations in

*All graphs were produced using DATAPLOT as described in Filliben (1981).



∇y

$\nabla\{y\}$

$\nabla\{y^T\}$

FIGURE 1
POSSIBLE ORIENTATIONS OF A MATRIX IN THREE-SPACE

three-space are not. These, too, are shown in Figure 1. The same comments apply with regard to total differentiation, for which the notation $D\{\cdot\}$ will be used. This notation, of course, is only used for an operation from R^2 to R^3; analogs exist for higher dimensions.

Now with this notation, the matrix form of (3.5) is written:

$$\overset{2}{\nabla}_{\varepsilon\varepsilon y} = -M^{-1}\left[\nabla_{y}M\ \nabla_{\varepsilon}\{y\}\ \nabla_{\varepsilon}y + \nabla_{\varepsilon}M\ \nabla_{\varepsilon}y + \nabla_{y}N\ \nabla_{\varepsilon}\{y\} + \nabla_{\varepsilon}N\right], \qquad (3.6)$$

where, letting $p = (n + m)$,

$\overset{2}{\nabla}_{\varepsilon\varepsilon y}$	is (p x r x r)	$\nabla_{\varepsilon}y$	is (p x r x 1)
M^{-1}	is (p x p x 1)	$\nabla_{\varepsilon}M$	is (p x p x r)
$\nabla_{y}M$	is (p x p x p)	$\nabla_{y}N$	is (p x r x p)
$\nabla_{\varepsilon}\{y\}$	is (p x 1 x r)	$\nabla_{\varepsilon}N$	is (p x r x r)

and the multiplication in three-space is carried out so that each array within the brackets on the rhs of (3.6) is (p x r x r). This (p x r x r) array must

then be premultiplied by M^{-1}, a (p x p x 1) array. This multiplication is effected by premultiplying each of the r matrices of dimension (p x r) by M^{-1}, resulting in r matrices of dimension (p x r), or a (p x r x r) array again, as required by $\overset{2}{\nabla}_{\varepsilon\varepsilon}y$. More on three-dimensional array multiplication (including graphical depictions) is given in Section 3.5.

Observe that (3.3) could have been differentiated directly using the techniques for differentiation of matrices as given in Marlow (1978). He uses Kronecker products and a special two-dimensional representation of three-dimensional arrays for performing this kind of differentiation. A disadvantage of this approach is that whatever special structure that exists in the three-dimensional arrays is lost in the process. This "structural integrity" is maintained in the approach given here, allowing the more detailed analysis given in the next section.

There has been some previous work in second-order sensitivity analysis. Dembo (1982) derived a computationally efficient technique for getting an approximation to the second derivative with respect to ε of the vector function y, correct to terms of order two. By contrast, (3.6) is exact. Also, a result for the geometric programming problem for the case where ε is a scalar was provided by Kyparisis (1983).

The material in the remainder of this section addresses the issue of computational efficiency when the problem functions are factorable. Sections 3.3 and 3.4 are admittedly rather detailed. The reason for including them is twofold: to illustrate the special polyadic structure of the tensors in nonlinear programming sensitivity analysis, and to stimulate more research in these areas. It is not necessary to wade through this material for each

second-order sensitivity calculation. Ultimately this will be performed automatically by the Factorable Programming system of programs being developed at the National Bureau of Standards.

3.3 Structure of the Three-Dimensional Arrays

While the formula in (3.6) may be mathematically succinct, it may not be obvious how the polyadic structure of derivatives of factorable functions can assist in its calculation. To understand how these calculations are performed, it is necessary to investigate further the structure of the three-dimensional arrays involved. These are: $\nabla_y M$, $\nabla_\varepsilon M$, $\nabla_y N$, and $\nabla_\varepsilon N$. Since $\nabla_\varepsilon N$ is the simplest of these, it is considered first. Strictly speaking, the $(i,j,k)^{th}$ element of $\nabla_\varepsilon N$ is

$$(\nabla_\varepsilon N)_{ijk} = \begin{cases} \dfrac{\partial^3 L}{\partial\varepsilon_k \partial\varepsilon_j \partial x_i}, & i \leq n \\ u_{i-n} \dfrac{\partial^2 g_{i-n}}{\partial\varepsilon_k \partial\varepsilon_j}, & i > n. \end{cases}$$

This can be thought of as taking the gradient with respect to ε of each element of N into the third dimension, and thus can be pictured as a partitioned rectangular parallelepiped as shown in Figure 2. The partitioning, which is due to the parts in y, separates $\nabla_\varepsilon N$ into an 3 "upper" part which is $\nabla_{\varepsilon\varepsilon x} L$ and a "lower" part which is just a stack of constraint Hessians with respect to ε. This more detailed structure is shown in the exploded view of $\nabla_\varepsilon N$ given in Figure 3.

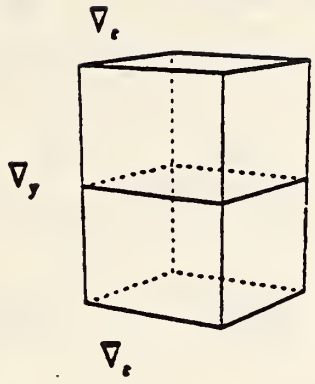


FIGURE 2
BASIC PARTITIONED STRUCTURE OF $\nabla_c N$

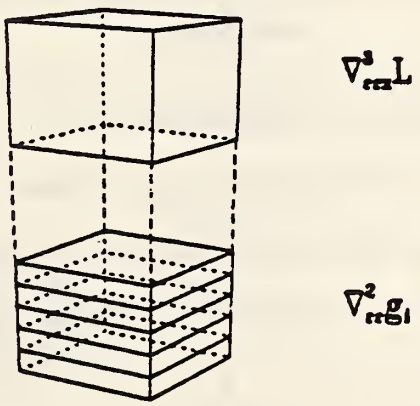


FIGURE 3
EXPLODED VIEW OF STRUCTURAL DETAILS
OF $\nabla_c N$

The next simplest array to portray is $\nabla_y N$. This is a three-dimensional array that has four parts, again a result of the two parts in y, arranged as shown in Figure 4. These partitions are described mathematically as:

$$(\nabla_y N)_{ijk} = \begin{cases} -\dfrac{\partial^3 L}{\partial x_k \partial \varepsilon_j \partial x_i}, & i<n,\ k<n \\ u_{i-n}\dfrac{\partial^2 g_{i-n}}{\partial x_k \partial \varepsilon_j}, & i>n,\ k<n \\ -\dfrac{\partial^2 g_{k-n}}{\partial \varepsilon_j \partial x_i}, & i<n,\ k>n \\ \dfrac{\partial g_{i-n}}{\partial x_j}, & i>n,\ k>n,\ i=k \\ 0, & \text{otherwise.} \end{cases}$$

These are depicted graphically in the exploded view in Figure 5, where the different orientations of the various matrices and vectors in three-space are more easily grasped.

The next three-dimensional array to consider is $\nabla_\varepsilon M$. This too is partitioned into four smaller three-dimensional arrays, but with an orientation that differs from $\nabla_y N$. The basic structure of the parts is shown in Figure 6, and each element of the array is defined in the following equations:

$$(\nabla_\varepsilon M)_{ijk} = \begin{cases} -\dfrac{\partial^3 L}{\partial \varepsilon_k \partial x_j \partial x_i}, & i<n,\ j<n \\ -\dfrac{\partial^2 g_{j-n}}{\partial \varepsilon_k \partial x_i}, & i<n,\ j>n \\ u_{i-n}\dfrac{\partial^2 g_{i-n}}{\partial \varepsilon_k \partial x_j}, & i>n,\ j<n \\ \dfrac{\partial g_{i-n}}{\partial \varepsilon_k}, & i>n,\ j>n,\ i=j \\ 0, & \text{otherwise.} \end{cases}$$

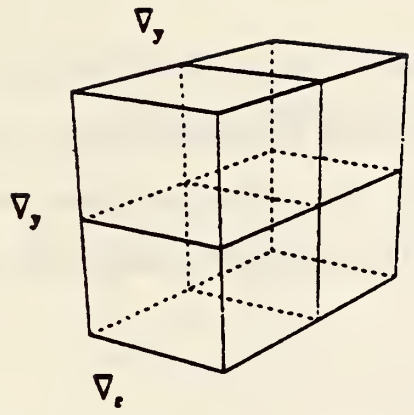


FIGURE 4
BASIC PARTITIONED STRUCTURE OF $\nabla_y N$

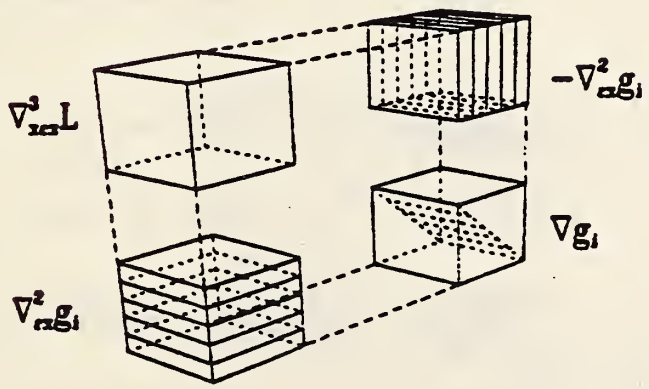


FIGURE 5
EXPLODED VIEW OF STRUCTURAL DETAILS
OF $\nabla_y N$

The three-dimensional orientation of these matrices and vectors is shown in more detail in Figure 7.

The last three-dimensional array to be depicted is $\nabla_y M$, the most complicated, with eight parts that result from differentiating three times with respect to the partitioned vector y. The eight parts are shown in Figure 8, and the mathematical statement of the $(i,j,k)^{th}$ element for each case is given below.

$$(\nabla_y M)_{ijk} = \begin{cases} -\dfrac{\partial^3 L}{\partial x_k \partial x_j \partial x_i}, & i \le n,\ j \le n,\ k \le n \\ -\dfrac{\partial^2 g_{j-n}}{\partial x_k \partial x_i}, & i \le n,\ j > n,\ k \le n \\ u_{i-n} \dfrac{\partial^2 g_{i-n}}{\partial x_k \partial x_j}, & i > n,\ j \le n,\ k \le n \\ \dfrac{\partial g_{i-n}}{\partial x_k}, & i > n,\ j > n,\ k \le n,\ i = j \\ -\dfrac{\partial^2 g_{k-n}}{\partial x_j \partial x_i}, & i \le n,\ j \le n,\ k > n \\ 0, & i \le n,\ j > n,\ k > n \\ \dfrac{\partial g_{i-n}}{\partial x_j}, & i > n,\ j \le n,\ k > n,\ i = k \\ 0, & i > n,\ j > n,\ k > n \\ 0, & \text{otherwise.} \end{cases}$$

And finally, these structures are shown in detail in Figure 9.

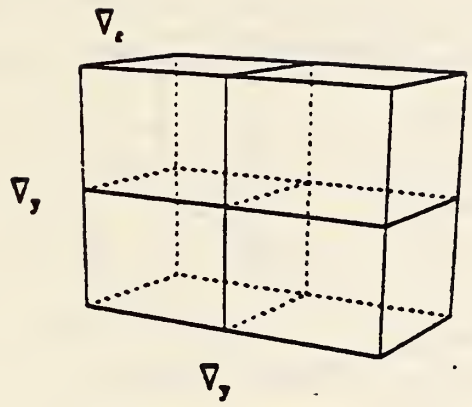


FIGURE 6
BASIC PARTITIONED STRUCTURE OF $\nabla_x M$

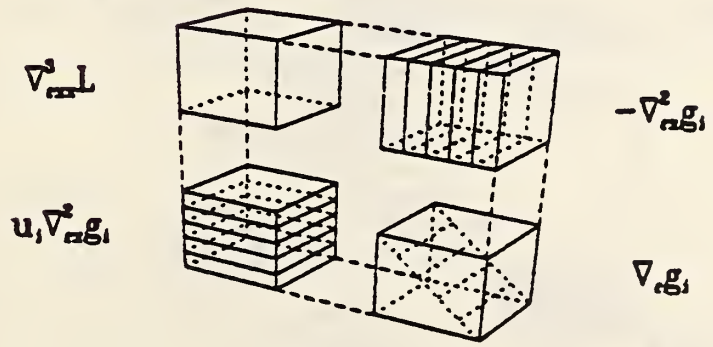


FIGURE 7
EXPLODED VIEW OF STRUCTURAL DETAILS
OF $\nabla_x M$

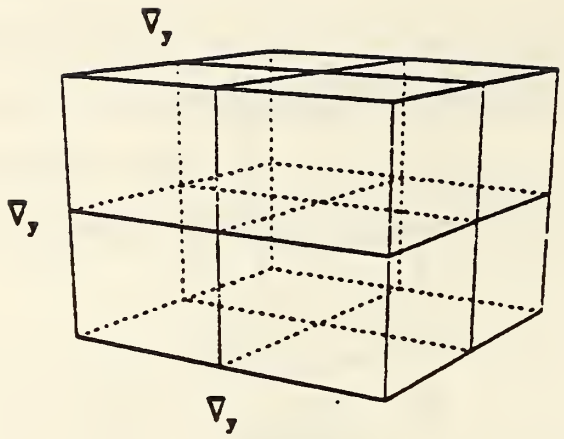


FIGURE 8
BASIC PARTITIONED STRUCTURE OF $\nabla_y M$

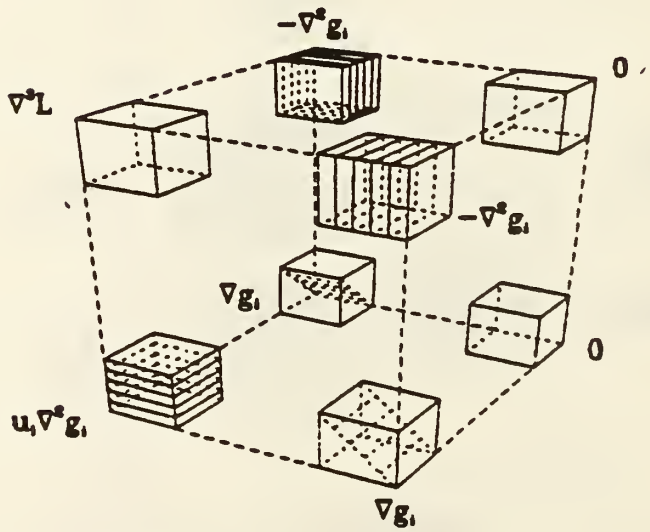


FIGURE 9
EXPLODED VIEW OF STRUCTURAL DETAILS
OF $\nabla_y M$

3.4 Polyadics in Second-Order Sensitivity Analysis

3.4.1 The Dyadics in the Second-Order Terms

Although the material in the previous section provided insight into the three-dimensional structure of the second-order sensitivity analysis formula in (3.6), it still may not be clear how the natural polyadic structure of tensors of factorable functions can assist in the evaluation of the formula. To see this, it is first necessary to show that each substructure exhibited in Figures 3, 5, 7, and 9, is a polyadic, and then to demonstrate how the multiplication with $\nabla_\varepsilon y$ (or $\nabla_\varepsilon\{y\}$) is to be carried out. This section addresses the former of these activities by providing proofs that factorable functions of (x,ε) have polyadic derivatives. Section 3.5 addresses the latter.

First notice that if the functions of the problem given in (3.1) are factorable in x and ε, so too is the Lagrangian of that problem factorable in x and ε. Then the task of this section reduces to considering some function $f(x,\varepsilon) = f_L(x,\varepsilon)$, with factored sequence $[f_1(x,\varepsilon), f_2(x,\varepsilon), \ldots, f_L(x,\varepsilon)]$ formed using the following modification to the rules given in (1.2).

Rule 1. For $i \leq n$,

$$f_i(x,\varepsilon) = x_i.$$

Rule 2. For $i > n$, either

$$\begin{aligned} &\text{a.)}\quad f_i(x,\varepsilon) = f_{j(i)}(x,\varepsilon) + f_{k(i)}(x,\varepsilon); \text{ or} \\ &\text{b.)}\quad f_i(x,\varepsilon) = f_{j(i)}(x,\varepsilon) \cdot f_{k(i)}(x,\varepsilon); \text{ or} \\ &\text{c.)}\quad f_i(x,\varepsilon) = T_i[f_{j(i)}(x,\varepsilon),\varepsilon]; \end{aligned} \tag{3.7}$$

where $j(i) < i$, and $k(i) < i$, and the derivatives with respect to ε of the T_i are themselves factorable functions in ε. For convenience, in the rest of this section, let $j(i) = j$, $k(i) = k$, and drop the arguments (x,ε) and $[f_{j(i)}(x,\varepsilon)]$. Thus, e.g., $T_i[f_{j(i)}(x,\varepsilon),\varepsilon] \to T_i$.

The gradients with respect to x of the forms in (3.7) are given in Table 5. Calculation of the Hessians with respect to x of these forms is straightforward and the result is given in Table 6. Notice that there is no difference between Table 6 and Table 2. Therefore Theorem 2 applies here and it is possible to write $\nabla^2 f(x,\varepsilon)$ as a sum of dyads of the appropriate form.

The calculation of the matrix of second partial derivatives of $f_i(x,\varepsilon)$ with respect to x and ε is slightly more complicated by the fact that T_i is a composite function of $f(x,\varepsilon)$ and ε. This requires the chain rule to obtain the second derivative matrix. Hence,

$$D_{\varepsilon x} T_i = D_\varepsilon [\dot{T}_i \nabla f_j] = \dot{T}_i \nabla^2_{\varepsilon x} f_j + \nabla f_j \ddot{T}_i \nabla_\varepsilon f_j^T + \nabla f_j \nabla_\varepsilon \dot{T}_i^T .$$

With this, the formulae for the matrix of second partials of f_i with respect to x and ε can be written as in Table 7. The first appearance of these formulae was in deSilva and McCormick (1978).

It should be apparent from Table 7 that an inductive argument paralleling that used in the proof of Theorem 2 would yield the fact that $\nabla^2_{\varepsilon x} f$ can also be written as the sum of dyads. The difference is that for $\nabla^2_{\varepsilon x} f$, the first vector in the dyads is a gradient with respect to x of a factored-sequence function, and the second vector is a gradient with respect to ε of a factored-sequence function <u>or</u> a derivative of a single-variable transformation.

This result implies that the submatrices of <u>second</u>-order derivatives which appear in the arrays $\nabla_y M$, $\nabla_\varepsilon M$, $\nabla_y N$, and $\nabla_\varepsilon N$ on the rhs of (3.6) and in Figures 3, 5, 7, and 9, are all dyadics. What remains is to show that the subarrays of <u>third</u>-order derivatives, in these same four arrays, are triadics. Those subarrays, arising from the block $\nabla^2 L$ in M and the block $\nabla^2_{\varepsilon x} L$ in N, are $\nabla^3 L$, $\nabla^3_{x\varepsilon x} L$, $\nabla^3_{\varepsilon x x} L$, and $\nabla^3_{\varepsilon\varepsilon x} L$. The first case, $\nabla^3 L$, is uncomplicated by

TABLE 5

GRADIENTS OF FACTORABLE FUNCTION FORMS
IN SENSITIVITY ANALYSIS

Rule	f_i	∇f_i
1	x_i	e_i
2a	$f_j + f_k$	$\nabla f_j + \nabla f_k$
2b	$f_j \cdot f_k$	$f_j \nabla f_k + f_k \nabla f_j$
2c	T_i	$\dot{T}_i \nabla f_j$

TABLE 6

HESSIANS OF FACTORABLE FUNCTION FORMS
IN SENSITIVITY ANALYSIS

Rule	f_i	$\nabla^2 f_i$
1	x_i	0_{nxn}
2a	$f_j + f_k$	$\nabla^2 f_j + \nabla^2 f_k$
2b	$f_j \cdot f_k$	$f_j \nabla^2 f_k + \nabla f_k \nabla f_j^T + f_k \nabla^2 f_j + \nabla f_j \nabla f_k^T$
2c	T_i	$\dot{T}_i \nabla^2 f_j + \nabla f_j \ddot{T}_i \nabla f_j^T$

TABLE 7

HESSIANS WITH RESPECT TO x AND ε OF FACTORABLE FUNCTION FORMS IN SENSITIVITY ANALYSIS

Rule	f_i	$\nabla^2_{\varepsilon x} f_i$
1	x_i	0_{nxr}
2a	$f_j + f_k$	$\nabla^2_{\varepsilon x} f_j + \nabla^2_{\varepsilon x} f_k$
2b	$f_j \cdot f_k$	$f_j \nabla^2_{\varepsilon x} f_k + \nabla f_k \nabla^T_{\varepsilon} f_j + f_k \nabla^2_{\varepsilon x} f_j + \nabla f_j \nabla^T_{\varepsilon} f_k$
2c	$T_i[f_j, \varepsilon]$	$\dot{T}_i \nabla^2_{\varepsilon x} f_j + \nabla f_j \ddot{T}_i \nabla^T_{\varepsilon} f_j + \nabla f_j \nabla^{\cdot T}_{\varepsilon} T_i$

derivatives with respect to ε, and thus is a triadic by Theorem 4. The proofs for the other cases are in the same vein as the proof of Theorem 2 and require that the formulae for the third derivatives with respect to x and ε respectively be derived for the forms in Tables 6 and 7. These are developed next.

3.4.2 The Triadic Form of $\nabla^3_{\varepsilon xx} f$

The first step in showing that $\nabla^3_{\varepsilon xx} f$ is triadic is to apply the operator $D_\varepsilon\{\cdot\}$ to each factorable function form in Table 6. This is straightforward for cases 1 and 2a. It is also straightforward for 2b, but since this case represents the first use of the new notation, it is developed below.

$$D_\varepsilon\{f_j \nabla^2 f_k + \nabla f_k \nabla^T f_j + f_k \nabla^2 f_j + \nabla f_j \nabla^T f_k\} = f_j \nabla^3_{\varepsilon xx} f_k + \nabla^2 f_k \nabla_\varepsilon\{f_j\} + \nabla f_k \nabla_\varepsilon\{\nabla^T f_j\}$$
$$+ \nabla f_j \nabla_\varepsilon\{\nabla^T f_k\} + f_k \nabla^3_{\varepsilon xx} f_j + \nabla^2 f_j \nabla_\varepsilon\{f_k\}$$
$$+ \nabla f_j \nabla_\varepsilon\{\nabla^T f_k\} + \nabla f_k \nabla_\varepsilon\{\nabla^T f_j\}.$$

The more complicated case is 2c, which is a result of the fact that, as noted earlier, each T_i (and hence $\dot{T}_i$ and $\ddot{T}_i$) is a composite function of $f(x,\varepsilon)$ and ε, requiring the chain rule to calculate the total derivative. This is shown below.

$$D_\varepsilon\{\dot{T}_i\nabla^2 f_j + \nabla f_j \ddot{T}_i \nabla f_j^T\} = \dot{T}_i \nabla^3_{\varepsilon xx} f_j + \nabla^2 f_j D_\varepsilon\{\dot{T}_i\} + \nabla_\varepsilon\{\nabla f_j\}\ddot{T}_i\nabla f_j^T$$

$$+ \nabla f_j D_\varepsilon\{\ddot{T}_i\}\nabla f_j^T + \nabla f_j \ddot{T}_i \nabla_\varepsilon\{\nabla f_j\}^T. \qquad (3.8)$$

However, using the chain rule,

$$D_\varepsilon\{\dot{T}_i\} = \ddot{T}_i\nabla_\varepsilon\{f_j\} + \nabla_\varepsilon\{\dot{T}_i\},$$

and (3.9)

$$D_\varepsilon\{\ddot{T}_i\} = \dddot{T}_i\nabla_\varepsilon\{f_j\} + \nabla_\varepsilon\{\ddot{T}_i\}.$$

After using (3.9), the rhs of (3.8) becomes:

$$\dot{T}_i\nabla^3_{\varepsilon xx} f_j + \ddot{T}_i\nabla^2 f_j \nabla_\varepsilon\{f_j\} + \nabla^2 f_j \nabla_\varepsilon\{\dot{T}_i\}$$

$$+ \nabla_\varepsilon\{\nabla f_j\}\ddot{T}_i\nabla f_j^T + \nabla f_j \dddot{T}_i \nabla_\varepsilon\{f_j\}\nabla f_j^T$$

$$+ \nabla f_j \nabla_\varepsilon\{\ddot{T}_i\}\nabla f_j^T + \nabla f_j \ddot{T}_i \nabla_\varepsilon\{\nabla f_j\}^T.$$

These third-derivative formulae for $\nabla^3_{\varepsilon xx} f$ are collected in Table 8. The proof that $\nabla^3_{\varepsilon xx} f$ is triadic is a straightforward application of induction to the factored sequence and mimics the argument used in the proof of Theorem 2. The result is that $\nabla^3_{\varepsilon xx} f$ can be written as the sum of triads of the form

$$(\alpha{:}a_1a_2a_3),$$

where

a_1 is $(n \times 1)$,

a_2 is $(n \times 1)$,

a_3 is $(r \times 1)$,

TABLE 8

THIRD-ORDER TENSORS WITH RESPECT TO x, x, AND ε OF FACTORABLE FUNCTION FORMS IN SENSITIVITY ANALYSIS

Rule	f_i	$\overset{3}{\nabla}_{\varepsilon xx} f_i$
1	x_i	0_{nxnxr}
2a	$f_j + f_k$	$\overset{3}{\nabla}_{\varepsilon xx} f_j + \overset{3}{\nabla}_{\varepsilon xx} f_k$
2b	$f_i \cdot f_j$	$f_j \overset{3}{\nabla}_{\varepsilon xx} f_k + \overset{2}{\nabla} f_k \nabla_\varepsilon \{f_j\} + \nabla f_k \nabla_\varepsilon \{\nabla f_j\}^T$ $+ \nabla f_j \nabla_\varepsilon \{\nabla f_k\}^T + f_k \overset{3}{\nabla}_{\varepsilon xx} f_j + \overset{2}{\nabla} f_j \nabla_\varepsilon \{f_k\}$ $+ \nabla f_k \nabla_\varepsilon \{\nabla f_k\}^T + \nabla f_k \nabla_\varepsilon \{\nabla f_j\}^T$
2c	$T_i[f_j, \varepsilon]$	$\dot{T}_i \overset{3}{\nabla}_{\varepsilon xx} f_j + \ddot{T}_i \overset{2}{\nabla} f_j \nabla_\varepsilon \{f_j\} + \overset{2}{\nabla} f_j \nabla_\varepsilon \{\dot{T}_i\}$ $+ \nabla_\varepsilon \{\nabla f_j\} \ddot{T}_i \nabla f_j^T + \nabla f_j \dddot{T}_i \nabla_\varepsilon \{f_j\} \nabla f_j^T$ $+ \nabla f_j \nabla_\varepsilon \{\ddot{T}_i\} \nabla f_j^T + \nabla f_j \ddot{T}_i \nabla_\varepsilon \{\nabla f_j\}^T$

and α is a product of factored sequence functions and first, second, and third derivatives of the single-variable transformations used in forming the factored sequence. Also, the vector factors a_1 and a_2 are gradients with respect to x of a factored sequence function, and a_3 is a gradient with respect to ε of a factored-sequence function or of a first or second derivative with respect to f of one of the single-variable transformations in the factored sequence.

3.4.3 The Triadic Form of $\nabla^3_{x\varepsilon x}f$

Just as in the previous section, the key step in the proof here is to compute the derivatives with respect to x of the forms in Table 7. Again this process is straightforward, if tedious, for cases 1, 2a, and 2b, but case 2c presents some complications, and is therefore developed in detail. Thus,

$$\begin{aligned} D\{\dot{T}_i\nabla^2_{\varepsilon x}f_j + \nabla f_j\ddot{T}_i\nabla_\varepsilon f_j^T + \nabla f_j\nabla_\varepsilon\dot{T}_i^T\} &= \dot{T}_i\nabla^3_{x\varepsilon x}f_j + \nabla^2_{\varepsilon x}f_j D\{\dot{T}_i\} + \nabla\{\nabla f_j\}\ddot{T}_i\nabla_\varepsilon f_j^T \\ &\quad + \nabla f_j D\{\dddot{T}_i\}\nabla_\varepsilon f_j^T + \nabla f_j\dddot{T}_i\nabla\{\nabla_\varepsilon f_j^T\} \qquad (3.10) \\ &\quad + \nabla f_j D\{\nabla_\varepsilon\dot{T}_i^T\} + \nabla_\varepsilon\dot{T}_i^T\nabla\{\nabla f_j\}. \end{aligned}$$

But, using the chain rule,

$$\begin{aligned} D\{\dot{T}_i\} &= \ddot{T}_i\nabla\{f_j\} \\ D\{\ddot{T}_i\} &= \dddot{T}_i\nabla\{f_j\} \qquad (3.11) \\ D\{\nabla_\varepsilon\dot{T}_i^T\} &= \nabla_\varepsilon\ddot{T}_i^T\nabla\{f_j\}. \end{aligned}$$

Substituting (3.11) into (3.10) yields the final form given in Table 9, which also contains the formulae for the other cases. Here too it is easy to see that the same inductive argument used in Theorem 2 results in a triadic form for $\nabla^3_{x\varepsilon x}f$ made of triads of the form

$$(\alpha{:}a_1a_2a_3),$$

where

a_1 is (n x 1),

a_2 is (r x 1),

a_3 is (n x 1),

and α is a product of factored-sequence functions and first, second, and third derivatives of the single-variable transformations. In this case, a_1 and a_3

TABLE 9

THIRD-ORDER TENSORS WITH RESPECT TO x, ε, AND x OF FACTORABLE FUNCTION FORMS IN SENSITIVITY ANALYSIS

Rule	f_i	$\overset{3}{\nabla}_{x\varepsilon x} f_i$
1	x_i	0_{nxrxn}
2a	$f_j + f_k$	$\overset{3}{\nabla}_{x\varepsilon x} f_j + \overset{3}{\nabla}_{x\varepsilon x} f_k$
2b	$f_j \cdot f_k$	$f_j \overset{3}{\nabla}_{x\varepsilon x} f_k + \overset{2}{\nabla}_{\varepsilon x} f_k \nabla\{f_j\} + \nabla f_k \nabla\{\overset{T}{\nabla}_{\varepsilon} f_j\}$ $+ \overset{T}{\nabla}_{\varepsilon} f_j \nabla\{\nabla f_k\} + f_k \overset{3}{\nabla}_{x\varepsilon x} f_j + \overset{2}{\nabla}_{\varepsilon x} f_j \nabla\{f_k\}$ $+ \nabla f_j \nabla\{\overset{T}{\nabla}_{\varepsilon} f_k\} + \overset{T}{\nabla}_{\varepsilon} f_k \nabla\{\nabla f_j\}$
2c	$T_i[f_j, \varepsilon]$	$\dot{T}_i \overset{3}{\nabla}_{x\varepsilon x} f_j + \overset{2}{\nabla}_{\varepsilon x} f_j \ddot{T}_i \nabla\{f_j\} + \nabla\{\nabla f_j\} \ddot{T}_i \overset{T}{\nabla}_{\varepsilon} f_j$ $+ \nabla f_j \dddot{T}_i \nabla\{f_j\} \overset{T}{\nabla}_{\varepsilon} f_j + \nabla f_j \ddot{T}_i \nabla\{\overset{T}{\nabla}_{\varepsilon} f_j\}$ $+ \nabla f_j \overset{\cdot\cdot T}{\nabla}_{\varepsilon} \dot{T}_i \nabla\{f_j\} + \overset{T}{\nabla}_{\varepsilon} \dot{T}_i \nabla\{\nabla f_j\}$

are gradients with respect to x of factored-sequence functions and a_2 is the gradient with respect to ε of a factored-sequence function or of a first or second derivative of a T_i.

It is instructive to note that the result in this section could also be obtained more directly by utilizing the triadic structure exhibited in Section 3.4.2 for $\overset{3}{\nabla}_{\varepsilon xx} f$. Because of the symmetric nature of partial derivatives, that array and the one of this section, $\overset{3}{\nabla}_{x\varepsilon x} f$, contain the same entries, in different arrangements. In fact, $\overset{3}{\nabla}_{x\varepsilon x} f$ can be obtained from the former by interchanging the first two vector factors of each triad in its triadic expansion, thereby exhibiting the triadic form for $\overset{3}{\nabla}_{x\varepsilon x} f$ immediately.

3.4.4 The Triadic Form of $\nabla^3_{\varepsilon\varepsilon x}f$

This is the last case to be considered, and also the most difficult of them. As in the previous sections, the formulae for forms 1, 2a, and 2b are straightforward, but for 2c,

$$D_\varepsilon\{\dot{T}_i\nabla^2_{\varepsilon x}f_j + \nabla f_j\ddot{T}_i\nabla_\varepsilon f_j^T + \nabla f_j\ \nabla_\varepsilon\dot{T}_i^T\} = \dot{T}_i\nabla^3_{\varepsilon\varepsilon x}f_j + \nabla^2_{\varepsilon x}f_jD_\varepsilon\{\dot{T}_i\} + \nabla_\varepsilon\{\nabla f_j\}\ddot{T}_i\nabla_\varepsilon f_j^T$$
$$+ \nabla f_jD_\varepsilon\{\ddot{T}_i\}\nabla_\varepsilon f_j^T + \nabla f_j\ddot{T}_i\nabla_\varepsilon\{\nabla_\varepsilon f_j\}^T$$
$$+ \nabla f_jD_\varepsilon\{\nabla_\varepsilon\dot{T}_i^T\} + \nabla_\varepsilon\dot{T}_i^T\nabla_\varepsilon\{\nabla f_j\}.$$

Both $D_\varepsilon\{\dot{T}_i\}$ and $D_\varepsilon\{\ddot{T}_i\}$ were calculated in (3.9). The third special computation needed is, using the chain rule again,

$$D_\varepsilon\{\nabla_\varepsilon\dot{T}_i^T\} = \nabla_\varepsilon\dot{T}_i^T\nabla_\varepsilon\{f_j\} + \nabla_\varepsilon\{\nabla_\varepsilon\dot{T}_i\}.$$

Substitution gives the formulae for the third partials in Table 10.

The inductive argument demonstrating the triadic nature of $\nabla^3_{\varepsilon\varepsilon x}f$ can now go through as before, bearing in mind the assumption in (3.7) that the derivatives with respect to f of the single-variable transformations, T_i, used in forming the factored sequence are themselves factorable functions in ε. This is required since

$$\nabla_\varepsilon\{\nabla_\varepsilon\dot{T}_i^T\}$$

appears in the last term of Table 10. Except for orientation in three-space, this is just the Hessian with respect to ε of the function $\dot{T}_i[f_j(x,\varepsilon),\varepsilon]$. So long as T_i is factorable in ε, its Hessian is a dyadic in gradients with respect to ε of its factored sequence, and the statements regarding the triadic nature of $\nabla^3_{\varepsilon\varepsilon x}f$ go through as before. Thus it is true that $\nabla^3_{\varepsilon\varepsilon x}f$ can

TABLE 10

THIRD-ORDER TENSORS WITH RESPECT TO x, ε, AND ε OF FACTORABLE FUNCTION FORMS IN SENSITIVITY ANALYSIS

Rule	f_i	$\overset{3}{\nabla}_{\varepsilon\varepsilon x} f_i$
1	x_i	0_{nxrxr}
2a	$f_j + f_k$	$\overset{3}{\nabla}_{\varepsilon\varepsilon x} f_j + \overset{3}{\nabla}_{\varepsilon\varepsilon x} f_k$
2b	$f_j \cdot f_k$	$f_j \overset{3}{\nabla}_{\varepsilon\varepsilon x} f_k + \overset{2}{\nabla}_{\varepsilon x} f_k \nabla_\varepsilon \{f_j\} + \nabla_x f_k \nabla_\varepsilon \{\overset{T}{\nabla}_\varepsilon f_j\}$ $+ \overset{T}{\nabla}_\varepsilon f_j \nabla_\varepsilon \{\nabla_x f_k\} + f_k \overset{3}{\nabla}_{\varepsilon\varepsilon x} f_j + \overset{2}{\nabla}_{\varepsilon x} f_j \nabla_\varepsilon \{f_k\}$ $+ \nabla f_j \nabla_\varepsilon \{\overset{T}{\nabla}_\varepsilon f_k\} + \overset{T}{\nabla}_\varepsilon f_k \nabla_\varepsilon \{\nabla f_j\}$
2c	$T_i[f_j, \varepsilon]$	$\dot{T}_i \overset{3}{\nabla}_{\varepsilon\varepsilon x} f_j + \overset{2}{\nabla}_{\varepsilon x} f_j \ddot{T}_i \nabla_\varepsilon \{f_j\} + \overset{2}{\nabla}_{\varepsilon x} f_j \nabla_\varepsilon \{\dot{T}_i\}$ $+ \nabla_\varepsilon \{\nabla f_j\} \ddot{T}_i \overset{T}{\nabla}_\varepsilon f_j + \nabla f_j \dddot{T}_i \nabla_\varepsilon \{f_j\} \overset{T}{\nabla}_\varepsilon f_j$ $+ \nabla f_j \nabla_\varepsilon \{\ddot{T}_i\} \overset{T}{\nabla}_\varepsilon f_j + \nabla f_j \ddot{T}_i \nabla_\varepsilon \{\overset{T}{\nabla}_\varepsilon f_j\}$ $+ \nabla f_j \nabla_\varepsilon \dot{\overset{T}{T}}_i \nabla_\varepsilon \{f_j\} + \nabla f_j \nabla_\varepsilon \{\nabla_\varepsilon \dot{\overset{T}{T}}_i\} + \nabla_\varepsilon \dot{\overset{T}{T}}_i \nabla_\varepsilon \{\nabla f_j\}$

be written as the sum of triads of the form

$$(\alpha : a_1 a_2 a_3),$$

where

a_1 is $(n \times 1)$,

a_2 is $(r \times 1)$,

a_3 is $(r \times 1)$,

and α is a product of factors as before, with additional factors included from the factored sequences for the $\dot{T}_i$'s. The last item of note is that here a_1 is

a gradient with respect to x of a factored-sequence function for f(x), and a_2 and a_3 are gradients with respect to ε of a factored-sequence function for f(x) or a gradient with respect to ε of a derivative of a T_i, or a gradient with respect to ε of a factored-sequence function or transformation for one of the single-variable transformations in the factored sequence for f(x).

3.5 Array Multiplication with Generalized Outer Product Matrices

In the previous section, the polyadic nature of the arrays in (3.6) was exhibited. This section takes up the notion of multiplying these generalized outer product matrices by $Y = \nabla_\varepsilon y$, which appears in (3.6) with two different three-space orientations; viz., $\nabla_\varepsilon y$ which is (p x r x 1) and $\nabla_\varepsilon\{y\}$ which is (p x 1 x r). Multiplication is one area wherein Factorable Programming, through the natural polyadic nature of the function derivatives involved, offers a potentially great computational saving over alternative approaches.

Consider for instance the case of multiplication of a dyad and a matrix,

$$(\alpha:ab) * F = (a\alpha b^T)F,$$

where a is (n x 1), b is (m x 1), F is (m x n) and α is a scalar. Of course one method of computing this is to form $(\alpha a)b^T$ which requires n(m+1) multiplications, then to multiply the result by F, requiring nm multiplications and n(m-1) additions. A far more efficient way, however, is to form $c = b^T F$, requiring the same nm multiplications and n(m-1) additions, but now store the result in the dyadic form as

$$(\alpha:ac) = a\alpha c^T,$$

thus saving the n(m+1) multiplications required to form the dyad explicitly. In fact, one of the basic tenets of Factorable Programming is that certain

matrices need never be formed explicitly, since all required calculations can be performed with the dyadic structures. This of course offers a potentially great computational saving.

It only needs demonstrating, therefore, how to perform the multiplications in (3.6). Consider then multiplication between a generalized outer product matrix of order 3, a triad, and a two-dimensional matrix in three-space. Since there are three possible orientations for a two-dimensional matrix in three-space, one could guess that there are six ways to perform the multiplication depending on whether the matrix is pre- or post-multiplying the three dimensional array. This is of course the case, and these multiplications are illustrated in Figure 10, where in each case the matrix post-multiplies each similarly oriented matrix-slice of the three-dimensional array. A similar situation obtains for pre-multiplication.

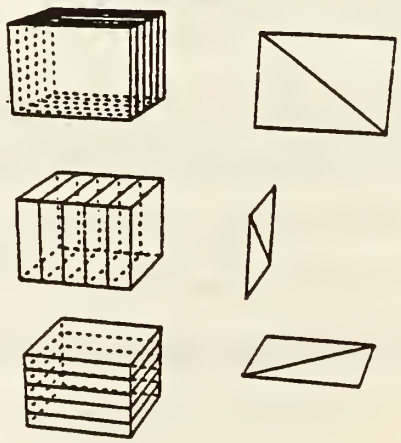


FIGURE 10
THREE WAYS TO MULTIPLY A THREE-DIMENSIONAL ARRAY BY A MATRIX

Just as with the multiplication of a dyad and a matrix, these three multiplications are simplified when the three-dimensional array is stored as a triadic and it is desired to store the result as a triadic also. Let one of the triads be $(\alpha:abc)$ and the matrix be F, and consider the post-multiplication of each each matrix-slice by F. Then the result of the multiplications in Figure 10 is just

i.) $(\alpha:a[b^TF]^Tc)$,

ii.) $(\alpha:ab[c^TF]^T)$, and

iii.) $(\alpha:ab[c^TF]^T)$.

(That which appears at first to be an error in (iii) is in fact a result of the simple fact that the vector in the third position, associated with the third dimension, is involved in defining two sets of slabs: one set with the vector a in the first dimension, and one set with the vector b in the second dimension).

One use of this ability to multiply polyads and matrices is in further exploiting the polyadic structure of the matrices in (3.6). It has been shown (McCormick (1983)) that both M and M^{-1} in (3.6) can be written in dyadic form. Let, therefore, $-M^{-1}$ in (3.6) be written in symmetric dyadic form as

$$-M^{-1} = \sum_{i=1}^{m} a_i\alpha_i a_i^T = \sum_{i=1}^{m} (\alpha_i:a_i a_i).$$

In the <u>unconstrained</u> case, (3.6) becomes

$$\nabla^2_{\varepsilon\varepsilon}x = -M^{-1}\,\nabla^3 L\nabla_\varepsilon\{x\}\nabla_\varepsilon x,$$

and since it has been shown that $\nabla^3 L$ is triadic, let

$$\nabla^3 L = \sum_{j=1}^{n} (\beta_j:b_jb_jb_j).$$

Then, for this unconstrained case,

$$\nabla^2_{\varepsilon\varepsilon}x = \sum_{i=1}^{m} (\alpha_i : a_i a_i) \sum_{j=1}^{m} (\beta_j : b_j b_j b_j)\nabla_\varepsilon\{x\}\nabla_\varepsilon x.$$

Application of (ii) yields

$$\nabla^2_{\varepsilon\varepsilon}x = \sum_{i=1}^{m} (\alpha_i : a_i a_i) \sum_{j=1}^{m} (\beta_j : b_j [b_j^T \nabla_\varepsilon x] b_j)\nabla_\varepsilon\{x\},$$

and application of (iii) yields

$$\nabla^2_{\varepsilon\varepsilon}x = \sum_{i=1}^{m} (\alpha_i : a_i a_i) \sum_{j=1}^{m} (\beta_j : b_j [b_j^T \nabla_\varepsilon x][b_j^T \nabla_\varepsilon x]).$$

Similar rules obtain for premultiplication and yield, using the associative law also,

$$\nabla^2_{\varepsilon\varepsilon}x = \sum_{i=1}^{m} \sum_{j=1}^{m} (\alpha_i\beta_j [a_i b_j]^T : a_i [b_j^T \nabla_\varepsilon x][b_j^T \nabla_\varepsilon x]),$$

the efficiency of which should be apparent. As an aside, if it were desired, as it was in a recent application of this technique to a problem for the Department of Energy, to produce each frontal slab of this in turn, the k^{th} of these is given by

$$\sum_{i=1}^{m} \sum_{j=1}^{n} (\alpha_i\beta_j [a_i b_j]^T a_{i,k} : [b_j^T \nabla_\varepsilon x][b_j^T \nabla_\varepsilon x]),$$

where, as before, $a_{i,k}$ denotes the k^{th} element of the vector a_i.

3.6 Parameter Tensors of the Optimal Value Function

In this section, formulae are developed for the tensors (through order 3) of the optimal value function $f^*(\varepsilon) = f[x(\varepsilon),\varepsilon]$ of problem $P(\varepsilon)$ given in (3.1). Armacost and Fiacco (1975) were the first to extend basic first-order results for the right-hand-side perturbation problem to the general parametric problem in (3.1), and also developed the second-order results given below in Theorem 8. Fiacco (1983) gives a complete treatment of all cases for all the variations of (3.1). Our interest is in providing results for the third-order tensor of $f^*(\varepsilon)$ and we begin with the first and second-order cases. Theorem 8 is due to Armacost and Fiacco (1975).

Theorem 8. (First- and Second-Order Changes in $f^*(\varepsilon)$ for $P(\varepsilon)$.)

Let x be a feasible point for $P(\hat{\varepsilon})$ and assume conditions (i) through (iv) in Theorem 6. Then, for ε in a sufficiently small neighborhood of $\hat{\varepsilon}$

$$\text{a) } f^*(\varepsilon) = L^*,$$

$$\text{b) } \nabla_\varepsilon f^*(\varepsilon) = \nabla_\varepsilon L$$

$$= \nabla_\varepsilon f - \sum_{i=1}^{m} u_i \nabla_\varepsilon g_i$$

$$= \nabla_\varepsilon f - u^T \nabla_\varepsilon g, \text{ and}$$

$$\text{c) } \nabla^2_{\varepsilon\varepsilon} f^*(\varepsilon) = \nabla^2_{y\varepsilon} L \nabla_\varepsilon y + \nabla^2_{\varepsilon\varepsilon} L$$

$$= \nabla^2_{x\varepsilon} L \nabla_\varepsilon x - \nabla_\varepsilon g^T \nabla_\varepsilon u + \nabla^2_{\varepsilon\varepsilon} L$$

Proof. See Fiacco (1983).

It is easy to see from their forms and from the results given in the previous section that $\nabla_{\varepsilon} f^*(\varepsilon)$ is monadic and $\overset{2}{\nabla}_{\varepsilon\varepsilon} f^*(\varepsilon)$ is dyadic. The result for the third-order tensor of $f^*(\varepsilon)$ is given in the following.

Theorem 9. (Third-Order Changes in $f^*(\varepsilon)$ for $P(\varepsilon)$.)

If the conditions of Theorem 8 hold and all third-order partial derivatives in x and third-order partial derivatives in x and ε exist and are continuous in x and ε in a neighborhood of $(\hat{x},\hat{\varepsilon})$, then

$$\begin{aligned}\overset{3}{\nabla}_{\varepsilon\varepsilon\varepsilon} f^*(\varepsilon) &= \overset{2}{\nabla}_{y\varepsilon}L\overset{2}{\nabla}_{\varepsilon\varepsilon}y + \overset{3}{\nabla}_{yy\varepsilon}L\nabla_{\varepsilon}\{y\}\nabla_{\varepsilon}y + \overset{3}{\nabla}_{\varepsilon y\varepsilon}L\nabla_{\varepsilon}y + \overset{3}{\nabla}_{\varepsilon\varepsilon\varepsilon}L + \nabla_{y\varepsilon\varepsilon}L\nabla_{\varepsilon}y \\ &= \overset{2}{\nabla}_{x\varepsilon}L\overset{2}{\nabla}_{\varepsilon\varepsilon}x + \overset{3}{\nabla}_{xx\varepsilon}L\nabla_{\varepsilon}\{x\}\nabla_{\varepsilon}x + \overset{3}{\nabla}_{ux\varepsilon}L\nabla_{\varepsilon}\{u\}\nabla_{\varepsilon}x + \overset{3}{\nabla}_{\varepsilon x\varepsilon}L\nabla_{\varepsilon}x \\ &\quad - \nabla_{\varepsilon}\overset{T}{g}\overset{2}{\nabla}_{\varepsilon\varepsilon}u - \nabla\{\nabla_{\varepsilon}\overset{T}{g}\}\nabla_{\varepsilon}\{x\}\nabla_{\varepsilon}u - \nabla_{\varepsilon}\{\nabla_{\varepsilon}\overset{T}{g}\}\nabla_{\varepsilon}u + \overset{3}{\nabla}_{\varepsilon\varepsilon\varepsilon}L,\end{aligned}$$

and $\overset{3}{\nabla}_{\varepsilon\varepsilon\varepsilon} f^*(\varepsilon)$ is a triadic.

Proof. Straightforward differentiation of (c) in Theorem 8 gives the formula for $\overset{3}{\nabla}_{\varepsilon\varepsilon\varepsilon} f^*(\varepsilon)$. The proof of its triadic structure is also straightforward using the results of the previous sections.

3.7 Second-Order Sensitivity Results in Use

The direct use of first- and second-order sensitivity results is in estimating the solution and multiplier vectors for $P(\varepsilon)$ as the problem is perturbed away from $P(\hat{\varepsilon})$. This estimation is done using the Taylor series approximations to two and three terms:

$$y(\varepsilon) \approx y(\hat{\varepsilon}) + \nabla_{\varepsilon} y(\hat{\varepsilon})(\varepsilon-\hat{\varepsilon}), \tag{3.12}$$

$$y(\varepsilon) \approx y(\hat{\varepsilon}) + \nabla_{\varepsilon} y(\hat{\varepsilon})(\varepsilon-\hat{\varepsilon}) + \frac{1}{2}\nabla^2_{\varepsilon\varepsilon} y(\hat{\varepsilon})(\varepsilon-\hat{\varepsilon})(\varepsilon-\hat{\varepsilon}), \tag{3.13}$$

where the multiplication in the third term on the rhs of (3.13) is understood to be inner product multiplication that reduces the dimension of $\nabla^2_{\varepsilon\varepsilon}y$.

To illustrate this idea as well as some of the other ideas in this section, consider the following parameterized nonlinear programming problem:

$$\underset{x\ \varepsilon\ R^n}{\text{minimize}}\ f(x,\varepsilon) = x^Tx + \varepsilon_1\exp(\varepsilon_2 a^Tx).$$

At $\hat{\varepsilon} = 0$, the solution by inspection is at $\hat{x} = 0$. Since $L(x,u,\varepsilon) = f(x,\varepsilon)$ for this problem, the first-order sensitivity equation, $Y = -M^{-1}N$, reduces to

$$\nabla_{\varepsilon x}(\varepsilon) = -(\nabla^2 f)^{-1}\nabla^2_{\varepsilon x}f.$$

Also

$$\nabla f = 2x + [\varepsilon_1\varepsilon_2\exp(\varepsilon_2 a^Tx)]a,$$

$$\nabla^2 f = 2I + a[\varepsilon_1\varepsilon_2^2\exp(\varepsilon_2 a^Tx)]a^T,$$

and

$$\nabla^2_{\varepsilon x}f = a[\exp(\varepsilon_2 a^Tx)][\varepsilon_2,\ \varepsilon_1\varepsilon_2 a^Tx + \varepsilon_1].$$

Since $(2I)^{-1} = \frac{1}{2}I$, $(\nabla^2 f)^{-1}$ can readily be obtained using the Sherman-Woodbury-Morrison formula (see McCormick (1983), p. 70) that gives the inverse of a matrix perturbed by a dyad. This formula is

$$(A + ucv^T)^{-1} = A^{-1} - A^{-1}u[c(1 + v^TA^{-1}uc)^{-1}]v^TA^{-1}.$$

Using this, and letting $c = \varepsilon_1\varepsilon_2\exp(\varepsilon_2 a^Tx)$,

$$(\nabla^2 f)^{-1} = (2I + aca^T)^{-1} = \tfrac{1}{2}I - a[(\tfrac{4}{c} + 2a^Ta)^{-1}]a^T.$$

Evaluating these at $\hat{x} = 0$ and $\hat{\varepsilon} = 0$ yields

$$(\nabla^2 f)^{-1} = \frac{1}{2}I,$$

and

$$\nabla^2_{\varepsilon x} f = 0.$$

Therefore

$$\nabla_\varepsilon x(\varepsilon) = 0,$$

and the first-order sensitivity analysis approximation (3.12) gives no additional information as the problem is perturbed away from $\hat{x} = 0$. This problem is ideally suited then for a second-order sensitivity analysis using equations (3.13) and (3.6). But since $\nabla_\varepsilon x(\varepsilon)=0$, (3.6) reduces to

$$\nabla^2_{\varepsilon\varepsilon} x(\varepsilon) = -(\nabla^2 f)^{-1} \nabla^3_{\varepsilon\varepsilon x} f.$$

Using the formulae developed in the proof of Theorem 3,

$$\begin{aligned}
\nabla^3_{\varepsilon\varepsilon x} f &= \nabla_\varepsilon (\exp[\varepsilon_2 a^T x] : a[0,\ \varepsilon_1 \varepsilon_2 a^T x]^T) + \nabla_\varepsilon (\exp[\varepsilon_2 a^T x] : a[\varepsilon_2,\ \varepsilon_1]^T) \\
&= (\exp[\varepsilon_2 a^T x] : a[\varepsilon_2,\ \varepsilon_1 \varepsilon_2 a^T x]^T [0,\ a^T x]^T) \\
&\quad + (\exp[\varepsilon_2 a^T x] : a[0, a^T x]^T [\varepsilon_2, \varepsilon_1]^T) \\
&\quad + (\exp[\varepsilon_2 a^T x] : a[\varepsilon_2,\ \varepsilon_1 \varepsilon_2 a^T x]^T [0,\ a^T x]^T) \\
&\quad + (\exp[\varepsilon_2 a^T x] : a[0,1]^T [1,0]^T) \\
&\quad + (\exp[\varepsilon_2 a^T x] : a[1,0]^T [0,1]^T).
\end{aligned}$$

Notice the triadic nature of this tensor and the frequency of occurrence of certain terms. Evaluating these at $\hat{x} = 0$ and $\hat{\varepsilon} = 0$ yields

$$\nabla^3_{\varepsilon\varepsilon x} f = (1 : a e_2 e_1) + (1 : a e_1 e_2),$$

where e_i is the i^{th} unit vector in R^n. The second partial derivatives with respect to ε of the solution vector are given by

$$\begin{aligned} \nabla^2_{\varepsilon\varepsilon} x(\varepsilon) &= -(\nabla^2 f(\hat{x}))^{-1} \nabla^3_{\varepsilon\varepsilon x} f(\hat{x}) \\ &= -\tfrac{1}{2} I[(1 : a e_2 e_1) + (1 : a e_1 e_2)] \\ &= (-\tfrac{1}{2} : a e_2 e_1) + (-\tfrac{1}{2} : a e_1 e_2). \end{aligned}$$

Substituting this into (3.13) gives the second-order estimation

$$x(\varepsilon) \approx x(\hat{\varepsilon}) + \nabla_\varepsilon x(\hat{\varepsilon})(\varepsilon - \hat{\varepsilon}) + \tfrac{1}{2} \nabla^2_{\varepsilon\varepsilon} x(\hat{\varepsilon})(\varepsilon - \hat{\varepsilon})(\varepsilon - \hat{\varepsilon}),$$

which at $\hat{\varepsilon} = 0$ becomes

$$x(\varepsilon) \approx x(0) + \nabla_\varepsilon x(0)\varepsilon + \tfrac{1}{2} \nabla^2_{\varepsilon\varepsilon} x(0)\varepsilon\varepsilon.$$

Since $x(0)$ is the solution to the original problem, $x(0) = \hat{x} = 0$. Furthermore, $\nabla_\varepsilon x(0)$ was shown above to vanish also. Thus

$$\begin{aligned} x(\varepsilon) &\approx \tfrac{1}{2} \nabla^2_{\varepsilon\varepsilon} x(0)\varepsilon\varepsilon \\ &= \tfrac{1}{2}[(-\tfrac{1}{2} : a e_2 e_1) + (-\tfrac{1}{2} : a e_1 e_2)]\varepsilon\varepsilon \\ &= (-\tfrac{1}{4} \varepsilon_2 \varepsilon_1 : a) + (-\tfrac{1}{4} \varepsilon_1 \varepsilon_2 : a) \\ &= (-\tfrac{1}{2} \varepsilon_1 \varepsilon_2 : a). \end{aligned} \tag{3.14}$$

For a more concrete example of this technique, let $a = (1,2)^T$ and perturb the problem by $\varepsilon = (-1.902, .1)^T$. The solution to this new problem is calculated in Jackson and McCormick (1984) using Halley's third-order method of tangent hyperbolas,and is $(.1, .2)^T$. However, an estimate is given by (3.14), without having to solve the new nonlinear programming problem. The approximation is

$$x(\varepsilon) \approx (-\tfrac{1}{2}(-1.902)(.1):[1,2]^T)$$

$$= (.0951, .1902)^T,$$

which of course is much better than the first-order approximation, $x(\varepsilon) = 0$.

Another use of the second-order sensitivity formulae is in solving implicitly defined optimization problems. Consider for example the optimization problem

$$\max_{(p,q)} \; F(p,q) = x^*(p,q) + y^*(p,q) + pq$$

where (x^*,y^*) is defined implicitly as the solution of

$$\min_{(x,y)} \; G(x,y) = (x - y + 3pq)^2 + (x - (p - q)^2)^2.$$

The analytic solution of this problem is easily obtained as

$$x^*(p,q) = (p - q)^2,$$

$$y^*(p,q) = (p - q) = (p - q)^2 + 3pq.$$

Substituting, the other problem becomes

$$\max_{(p,q)} \; 2p^2 + 2q^2,$$

with solution $(p^*,q^*) = (0,0)$.

Let $\varepsilon = (p,q)^T$. The solution of the maximization problem will be accomplished in one iteration of Newton's method using the second-order sensitivity formulas.

It is required to compute

$$\varepsilon_o - [\nabla^2_{\varepsilon\varepsilon} F(\varepsilon_o)]^{-1} \nabla_\varepsilon F(\varepsilon_o).$$

Now

$$\nabla_\varepsilon F = \nabla_\varepsilon x^* + \nabla y^* + [q,p]^T,$$

And

$$\nabla^2_{\varepsilon\varepsilon} F = \nabla^2_{\varepsilon\varepsilon} x^* + \nabla^2_{\varepsilon\varepsilon} y^* + \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$$

Let z denote $[x,y]^T$. Then from first-order sensitivity analysis

$$\nabla_\varepsilon z^* = -(\nabla^2_{\varepsilon\varepsilon} G)^{-1} \nabla_\varepsilon z G .$$

Now

$$\nabla^2_{zz} G = \begin{vmatrix} 1 & \\ & -1 \end{vmatrix} (2)[1,-1] + \begin{vmatrix} 1 & \\ & 0 \end{vmatrix} (2)[1,0]$$

and

$$\nabla^2_{\varepsilon z} G = \begin{vmatrix} 1 & \\ & -1 \end{vmatrix} (2)[3q,3p] + \begin{vmatrix} 1 & \\ & 0 \end{vmatrix} (4)(pq)[1,-1].$$

The most natural representation of $(\nabla^2_{zz} G)^{-1}$ is in dyadic form and is

$$\begin{vmatrix} 0 & \\ & -1 \end{vmatrix} (1/2)[0,-1] + \begin{vmatrix} 1 & \\ & 1 \end{vmatrix} (1/2)[1,1].$$

The second-order sensitivity formula (3.6) can be rewritten (conceptually) in this case as

$$\nabla^2_{\varepsilon\varepsilon} z^* = -(\nabla^2_{\varepsilon\varepsilon} G)^{-1} \left[(\nabla^3_{\varepsilon zz} G)(\nabla_\varepsilon z^*) + (\nabla^3_{zzz} G)(\nabla_\varepsilon z^*)^2 + \nabla^3_{\varepsilon\varepsilon z} G + (\nabla^3_{z\varepsilon z} G)(\nabla_\varepsilon z^*) \right].$$

For this problem the only term multiplying the inverse which does not vanish is $\nabla^3_{\varepsilon\varepsilon z} G$, a (2 x 2 x 2) matrix.

Its triadic form is

$$\nabla^3_{\varepsilon\varepsilon z}G = (6{:}[1,-1]^T[1,0]^T[0,1]^T) + (6{:}[1,-1]^T[0,1]^T[1,0]^T) + (-4{:}[1,0]^T[1,-1]^T[1,-1]^T).$$

Suppose $[p_o,q_o] = [2,1]$. Then

$$\nabla_\varepsilon z_o^* = -\begin{vmatrix} 0 \\ -1 \end{vmatrix}(1/2)[0,-1] + \begin{vmatrix} 1 \\ 1 \end{vmatrix}(1/2)[1,1] + \begin{vmatrix} 1 \\ -1 \end{vmatrix}(2)[3,6] + \begin{vmatrix} 1 \\ 1 \end{vmatrix}(4)[1,-1].$$

Therefore

$$\nabla_\varepsilon x_o^* = [2,-2]^T,\ \nabla_\varepsilon y_o^* = [5,4]^T,$$

and thus

$$\nabla_\varepsilon F_o = [2,-2]^T + [5,4]^T + [1,2]^T = [8,4]^T.$$

The second-order sensitivity formulas yield

$$-[\nabla^2_{zz}G]^{-1}[\nabla^3_{\varepsilon\varepsilon z}G] = -[(1/2{:}[0,-1]^T[0,-1]^T) + (1/2{:}[1,1]^T[1,1]^T)] * [(6{:}[1,-1]^T[1,0]^T[0,1]^T) + (6{:}[1,-1]^T[0,1]^T[1,0]^T) + (-4{:}[1,0]^T[1,-1]^T[1,-1]^T)].$$

To illustrate the computation, one of the six product terms will be computed:

$$(-1/2{:}[0,1]^T[0,-1]^T) * (6{:}[1,-1]^T[1,0]^T[0,1]^T) = (-(1)(1/2)(6)[0,-1][1,-1]^T{:}[0,-1]^T[1,0]^T[0,1]^T) = (-3{:}[0,1]^T[1,0]^T[0,1]^T.$$

In all, the resulting triadic form has the following terms:

$$(-3{:}[0,-1]^T[1,0]^T[0,1]^T) + (-3{:}[0,-1]^T[0,1]^T[1,0]^T) + (2{:}[1,1]^T[1,-1]^T[1,-1]^T).$$

From this,

$$\nabla^2_{\varepsilon\varepsilon} x_o^* = (2:[1,-1]^T[1,-1]^T)$$

and

$$\nabla^2_{\varepsilon\varepsilon} y_o^* = (2:[1,-1]^T[1,-1]^T) + (3:[0,1]^T[1,0]^T) + (3:[1,0]^T[0,1]^T)$$

Thus

$$\nabla^2_{\varepsilon\varepsilon} F_o = \nabla^2_{\varepsilon\varepsilon} x_o^* + \nabla^2_{\varepsilon\varepsilon} y_o^* + \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} = \begin{vmatrix} 4 & 0 \\ 0 & 4 \end{vmatrix}$$

Combining,

$$\varepsilon_o - (\nabla^2_{\varepsilon\varepsilon} F_o)^{-1} (\nabla_\varepsilon F_o) = \begin{vmatrix} 2 \\ 1 \end{vmatrix} - \begin{vmatrix} 4 & 0 \\ 0 & 4 \end{vmatrix}^{-1} \begin{vmatrix} 8 \\ 4 \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$$

as desired.

BIBLIOGRAPHY

R.L. Armacost and A.V. Fiacco, "Computational experience in sensitivity analysis for nonlinear programming," Mathematical Programming 6, (1974), 301-326.

R.L. Armacost and A.V. Fiacco, "Sensitivity analysis for parametric nonlinear programming using penalty methods," In Computers and Mathematical Programming, National Bureau of Standards Special Publication 502, (1978) 261-269.

G.A. Bliss, Lectures on the calculus of variations, (University of Chicago Press, Chicago, 1946).

R.S. Dembo, "Sensitivity analysis in geometric programming," JOTA, 37 (1982) 1-21.

A. DeSilva and G. P. McCormick, "Sensitivity analysis in nonlinear programming using factorable symbolic input," Technical Report T-365, The George Washington University, Institute for Management Science and Engineering, (Washington, DC, 1978).

G. Emami, "Evaluating strategies for Newton's method using a numerically stable generalized inverse algorithm," Dissertation, The George Washington University, Department of Operations Research, (Washington, DC, 1978).

A.V. Fiacco and G.P. McCormick, Nonlinear Programming: Sequential Unconstrained Minimzation Techniques, (Wiley, New York, 1968).

A.V. Fiacco, Sensitivity analysis for nonlinear programming using penalty methods," Math. Prog., 10 (1976) 287-311.

A.V. Fiacco, "Nonlinear programming sensitivity analysis results using strong second order assumptions," In Numerical Optimization of Dynamic Systems (L.C.W. Dixon and G.P. Szego, eds.), (North-Holland, Amsterdam, 1980 327-348.

A.V. Fiacco, Introduction to Sensitivity and Stability Analysis in Nonlinear Programming, (Academic Press, New York, 1983).

J.J. Filliben, "DATAPLOT--an interactive high level language for graphics, nonlinear fitting, data analysis and mathematics," Computer Graphics, 15 (1981) 199-213.

A. Ghaemi, and G.P. McCormick, "Factorable symbolic SUMT: What is it? How is it used?", Technical Report T-402, The George Washington University Institute for Management Science and Engineering (Washington, DC, 1979).

F. Ghotb, "Newton's method for linearly constrained optimization problems," Dissertation, The George Washington University, Department of Operations Research, (Washington, DC, 1980).

A. Graham, Kronecker Products and Matrix Calculus with Applications, (Wiley, New York, 1981).

K.E. Hillstrom, "JAKEF - a portable symbolic differentiator of functions given by algorithms," Technical report ANL-82-48, Argonne National Laboratory, (Argonne , IL, 1982).

R.H.F. Jackson, "Tensors, Polyads, and High-Order Methods in Factorable Programming," Dissertation, The George Washington University, Department of Operations Research, (Washington, DC, 1983).

R.H.F. Jackson and G.P. McCormick, "The polyadic structure of factorable function tensors with application to high-order minimization techniques," JOTA, to appear, (1984).

J. Kyparisis, "Sensitivity and stability for nonlinear and geometric programming: theory and applications. Dissertation, The George Washington University, Department of Operations Research (Washington, DC, 1983).

A. Linneman, "Higher-order necessary conditions for infinite and semi-infinite optimization," JOTA, 38 (1982) 483-511.

W.H. Marlow, Mathematics for Operations Research, (Wiley, New York, 1978).

G.P. McCormick, "Second order conditions for constrained minima," SIAM J. Appl. Math., 15 (1967) 37-47.

G.P. McCormick, "A mini-manual for use of SUMT computer program and the factorable programming language," Technical report SOL-74-15, Stanford University Department of Operations Research, (Stanford, CA, 1974).

G.P. McCormick, Nonlinear Programming: Theory, Algorithms and Applications, (Wiley, New York, 1983).

A. Miele and S. Gonzalez, "On the comparative evaluation of algorithms for mathematical programming problems," in O.L. Mangasarian, et al., (eds.): Nonlinear Programming 3, (Academic Press, New York, 1978) 337-359.

W.C. Mylander, R. Holmes, and G.P. McCormick, "A guide to SUMT-Version 4: The computer program implementing the sequential unconstrained minimization technique for nonlinear programming," Technical Report RAC-P-63, Research Analysis Corporation (McLean, VA, 1971).

L. Pennisi, "An indirect proof of the problem of Lagrange with differential inequalities as added side conditions," Trans. Am. Math. Soc., 74 (1953), 177-198.

R.E. Pugh, "A language for nonlinear programming problems" Math. Prog., 2 (1972) 176-206.

M.E. Shayan, "A methodology for comparing algorithms and a method for computing m^{th} order directional derivatives based on factorable programming," Dissertation, The George Washington University, Department of Operations Research, (Washington, DC, 1978).

A. Sofer, "Computationally efficient techniques for generalized inversion in nonlinear programming," Dissertation, The George Washington University, Department of Operations Research, (Washington, DC, 1983).

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 85-3222	2. Performing Organ. Report No.	3. Publication Date AUGUST 1985

4. TITLE AND SUBTITLE

Polyadic Third-Order Lagrangian Tensor Structure and Second-Order Sensitivity Analysis with Factorable Functions

5. AUTHOR(S)

Richard H. F. Jackson and Garth P. McCormick

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Operations Research Division
Center for Applied Mathematics
National Bureau of Standards
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Second-order sensitivity analysis methods are developed for analyzing the behavior of a local solution to a constrained nonlinear optimization problem when the problem functions are perturbed slightly. Specifically, formulas involving third-order tensors are given to compute second derivatives of components of the local solution with respect to the problem parameters. When in addition, the problem functions are factorable, it is shown that the resulting tensors are polyadic in nature.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Second-order sensitivity analysis; high-order methods; N^{th} derivatives; polyads; tensors

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

67

15. Price

$10.00