AIAA JOURNAL
Vol. 41, No. 8, August 2003

Automated Generation of High-Order Partial Derivative Models

James D. Turner*
Amdyn Systems, White, Georgia 30184

A standard problem in science and engineering consists of developing mathematics and sensitivity models for complex applications for optimizing a candidate design. A chain rule-based evaluation technique is presented for analytically evaluating partial derivatives of nonlinear functions and differential equations defined by a high-level language. A coordinate embedding strategy is introduced that replaces all scalar variables with higher-dimensional objects. The higher-dimensional objects are defined by a concatenation of the original scalar and its Jacobian, Hessian, and higher-order partials. Exact sensitivity models are recovered for arbitrarily complex mathematical models. An operator-overloading technique is used to define generalized operators for basic and standard library functions. The generalized operators encode the chain rule of calculus and store the results of partial derivative calculations in the artificial dimensions used to redefine the scalar operations. Hidden operations automatically generate and evaluate exact first- through fourth-order partial derivative models, which are accurate to the working precision of the machine. The new algorithm replaces a normally complex, error-prone, time-consuming, and labor-intensive process for producing the partials with an automatic procedure. Module functions encapsulate new data types, and extended mathematic and library functions for handling vector, matrix, and tensor operations. Matrix operations are shown to generalize easily. The algorithm has broad potential for impacting the design and use of mathematical programming tools for applications in science and engineering. Several applications are presented that demonstrate the effectiveness of the methods.

Introduction

THE calculation of sensitivity partial derivatives is a frequently occurring task during the engineering design and control process.[1–27] Sensitivity models are derived from an underlying mathematical model of the physical system. Analytical models are preferred. Unfortunately, for many large problems, the time and labor costs required for code development and testing are prohibitive. Numerical methods are conceptually straightforward; however, many algorithms are very sensitive to assumed perturbation step sizes, and a function must be sampled two or more times to estimate reliably the sensitivity. This paper presents a third alternative approach for generating the sensitivity data: an object-oriented Cartesian embedding algorithm (OCEA).[1,2] OCEA combines the accuracy of the analytical method with the simplicity of a numerical method, where the user makes no decisions about how the partials are generated. Operator-overloading techniques[28–30] are used to automate the partial derivative calculations. A single OCEA function evaluation generates exact numerical values for the function, as well as hidden values for the Jacobian and higher-order partial derivatives. The partial derivatives are extracted as a postprocessing step by using utility routines. FORTRAN 90 (F90) and Macsyma 2.4 (Ref. 31) OCEA prototype codes have been developed.

OCEA belongs to a class of computational methods known as automatic differentiation (AD).[1–19] AD has existed as a research topic since the 1980s (Ref. 3). AD is a chain rule-based evaluation technique for building partial derivative models with respect to user-defined sets of independent variables. The methodology works for any high-level language.[1,2] A common strategy has been to preprocess a programmed function, identify the unary and binary functional operations,[28–30] and build up forward and backward computational sequences,[3–9] where the sequence data are used for generating an output source code for the partial derivative calculations. This approach has proven to be extremely successful for large-scale applications with highly sparse system models.[5–11] For example, ADIFOR[4,5] represents a successful implementation of this approach for first-order partials. Other related codes include AD01,[8] which uses F90 operator-overloading techniques[28–30] and computational graphs.[9,19] ADOL-C[9] is a widely used graph-based C/C++ code for first-order partials. ADMIT-1[11] is a MATLAB® interface toolbox for linking different AD codes to the MATLAB environment. AUTO_Deriv[21] handles transformations of FORTRAN codes. The availability of AD tools has sparked the investigation of many emerging applications in optimization,[1,2,7–11,16] robotics,[17] multibody,[1,2,18] algorithms,[1,2,15,19] and molecular dynamics.[13]

In contrast with computational graph approaches, OCEA's operator-overloading algorithms[28–30] build models on the fly: No application-specific source code is written for computing the partials. Arbitrarily complex systems and matrix algebra are handled because OCEA operates at the elementary operator and function level. Many algorithms only require the insertion of a USE Module statement and some minor changes of variable types. OCEA enabled partial derivative capabilities represent a language extension for high-level languages, that is, F90/F95,[28–30] C++, and Matlab. For large problems, OCEA has the potential for replacing months of analyst time with days of computer time. The automated nature of OCEA produces a computational overhead for all calculations. Some calculations can be 100 times slower than equivalent hand-optimized codes. Nevertheless, for complex problems, it is likely that the reductions in analyst development time will significantly offset any increased computer costs.

Previous AD approaches have been limited to second-order capabilities for Hessians and directionally projected partials. This paper significantly extends the range of application areas for AD technologies by presenting algorithms and applications that exploit though fourth-order capabilities.

Computationally, OCEA replaces each scalar object with a scalar-like compound data object that consists of a concatenation of the scalar variable and its associated partial derivatives. OCEA variables are created in software by defining a derived data type. The hidden artificial dimensions provide storage and workspace for the partial derivative calculations. The scalarlike nature of OCEA variables allows conventional mathematical programming algorithms to remain essentially unchanged when enhanced for OCEA capabilities.[28–30] For example, the 1×1 scalar g, in a second-order OCEA algorithm,

Received 24 October 2002; revision received 26 February 2003; accepted for publication 17 March 2003.

*President, Research and Development, 36 Windrush Drive; jamesdanielturner@hotmail.com. Member AIAA.

is transformed to become the following data structure[28–30]:

$$\underbrace{g}_{1\times(1+m+m^2)} := \left[\underbrace{g}_{1\times 1} \quad \underbrace{\nabla g}_{m\times 1} \quad \underbrace{\nabla\nabla g}_{m\times m}\right];$$

$$\nabla = \hat{\boldsymbol{n}}_1 \frac{\partial}{\partial x_1} + \cdots + \hat{n}_m \frac{\partial}{\partial \boldsymbol{x}_m}$$

where $x_i, i = 1, \ldots, m$, is the vector of independent variables, $\hat{\boldsymbol{n}}_i = (\delta_{i1}, \ldots, \delta_{im})$ is a unit vector in the ith coordinate direction, δ_{ij} is the standard Kronecker delta, and the transformed version of g has dimension $1 \times (1 + m + m^2)$. The new compound data object consists of a concatenation of the variable and two-orders of partial derivatives. Generalizations for higher-dimensional versions of OCEA are obvious.

The development of an OCEA-based AD capability requires three elements: 1) generalized intrinsic binary operators $\{+, -, *, **, /\}$, 2) generalized unary functions $\{\cos(x), \sin(x), \tan(x), \log(x), \ldots.\}$, and 3) encoded chain rules for all new operators and functions. Derived data types and interface operators[28–30] are used to manage the OCEA binary operators and unary functions. Expressing these requirements in F90 leads to 50+ module-hidden routines for redefining the intrinsic and mathematical library functions. Operator overloading manages the definitions for interface operators that allow the compiler to recognize 1) the mathematical functions and 2) the argument list data types (including user-defined data types) for automatically building links to the hidden routines at compile time.

The mechanics of operator overloading is best understood by considering a simple example. To this end, assume one wants to evaluate $f = xy/\sqrt{z}$, where (x, y, z) are independent variables. As shown in Table 1, the standard and OCEA-enhanced FORTRAN models for f are identical. The compiler recognizes the operator-overloaded operators for $*$ and $/$, as well as a generalized library function in the square root (sqrt) calculation. This information allows the compiler to build links to the hidden OCEA functions for automatically computing f and its partials.

During an OCEA partial derivative calculation, four factors impact the efficiency of the calculations: 1) partial derivative order, 2) exploitation of symmetry (Table 2), 3) exploitation of sparse structure, and 4) level of optimization for the OCEA generalized intrinsic and mathematical functions. A full exploitation of these factors can dramatically impact the computational performance of OCEA-based tools, that is, 10–100 times).

Table 1 Comparison of FORTRAN and a second-order OCEA FORTRAN model

Mathematical model	$f = xy/\sqrt{z}$
FORTRAN $f :=$ (scalar)	$f = x * y/\text{sqrt}(z)$
OCEA-enhanced FORTRAN $f :=$ (scalar, vector, tensor)	$f = x * y/\text{sqrt}(z)$ $= \left[\begin{array}{c} f, \\ \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \quad \frac{\partial f}{\partial z}\right], \\ \left[\frac{\partial^2 f}{\partial x \partial x} \quad \frac{\partial^2 f}{\partial x \partial y} \quad \cdots \quad \frac{\partial^2 f}{\partial z \partial z}\right] \end{array}\right]$

Table 2 Number of terms for a symmetric and nonsymmetric OCEA variable

OCEA data storage strategy	Total number of OCEA variables
Nonsymmetric OCEA	$\sum_{n=0}^{q} m^n = \frac{(1-m^{q+1})}{(1-m)}$
Symmetric OCEA	$\sum_{n=0}^{q} \binom{n+m-1}{m-1} = \binom{q+m}{m}$

OCEA methodology can be expected to have a broad impact on the design and use of mathematical programming software for science and engineering applications and support a wide spectrum of sensitivity-based knowledge discovery applications. The paper presents algorithm, software, and application formulation issues.

Mathematical Formulation

OCEA methodology can be viewed as a transformational process that changes OCEA data, that is, functional and partial derivative information, into different forms during calculations. For example, when an OCEA sine function is computed, one obtains $\sin(x) := \sin([x, \nabla x, \nabla\nabla x, \ldots]) := [\sin(x), \cos(x)\nabla x, \cos(x)\nabla\nabla x - \sin(x)\nabla x\nabla x, \ldots]$, where OCEA has transformed all of the information provided in the input OCEA variable x. All intrinsic operators and library functions similarly encode the transformational process.

OCEA tool development addresses the following six software issues: 1) defining how independent variables are transformed to OCEA form; 2) developing derived data types for vectors, tensors, and embedded variables; 3) defining interface operators for supporting generalized operations; 4) using module functions to hide OCEA computational resources; 5) defining OCEA-enhanced library routines that encode chain rule models; and 6) providing utility routines provide to access the OCEA partial derivative calculations.

Initialization of OCEA Independent Variables

Independent variables are identified for each application and transformed to OCEA form. For example, given the following set of independent variables $x_1, x_2, \ldots, x_n$, the second-order OCEA form of x_i is given by

$$\underbrace{\boldsymbol{x}_i}_{1\times(1+n+n^2)} := \left[\underbrace{\boldsymbol{x}_i}_{1\times 1}, \underbrace{\begin{bmatrix}\delta_{1i}\\ \vdots \\ \delta_{ni}\end{bmatrix}}_{n\times 1}, \underbrace{\begin{bmatrix}0 & \cdots & 0\\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0\end{bmatrix}}_{n\times n}\right]$$

where δ_{ij} denotes the standard Kronecker delta function and $i = 1, \ldots, n$. The vector part represents the Jacobian and the matrix part represents the Hessian for x_i. The nonvanishing part of the Jacobian is the essential element for enabling the numerical partial derivative calculations. During a calculation, the partial derivatives are accumulated so that a general OCEA variable assumes the form

$$v := \left[v, \begin{bmatrix}x\\ \vdots \\ x\end{bmatrix}, \begin{bmatrix}y & \cdots & y\\ \vdots & \ddots & \vdots \\ y & \cdots & y\end{bmatrix}\right]$$

where x and y are generic filling elements.

The elements of an OCEA-scalar compound data object are referenced by defining $f := [f, \nabla f, \nabla\nabla f]$, where the individual objects are extracted as

$$f = f\%E, \qquad \nabla f = f\%V, \qquad \nabla\nabla f = f\%T$$

where $\%E$ is the scalar part of the embedded variable, $\%V$ is the vector part of the embedded variable, and $\%T$ is the tensor part of the embedded variable. At a finer level of detail, the individual components are extracted by defining

$$f = f\%E, \qquad (\nabla f)_i = f\%V\%V\text{PART}(i)$$

$$(\nabla f)_{ij} = f\%T\%T\text{PART}(i, j)$$

where $(\,)_i$ is the ith component, $(\,)_{ij}$ is the i–jth component, and $V\text{PART}(i)$ and $T\text{PART}(i, j)$ are structure constructor variables.[28–30] The computational advantage of this approach is that high-level variable assignments can be made for updating vector, matrix, and tensor computations.

Intrinsic Operators and Functions

Operator-overloading[28–30] methodologies are used to redefine the computers operational rules for processing numerical calculations. The new operational rules are managed in Module functions. Generalizations for the intrinsic mathematical operators and functions are presented for addition, subtraction, multiplication, division, and composite functions. Advanced partial derivative capabilities are enabled because multiple levels of the chain rule are encoded in each OCEA operator and function.

Fourth-order OCEA variables are used to define the math models, where a and b are the OCEA variables defined by

$$a := [a, \nabla a, \nabla\nabla a, \nabla\nabla\nabla a, \nabla\nabla\nabla\nabla a\,]$$

$$b := [\,b, \nabla b, \nabla\nabla b, \nabla\nabla\nabla b, \nabla\nabla\nabla\nabla b\,]$$

The third- and fourth-order results represent nontrivial extensions of previously published results.[1–20] Addition and subtraction are straightforward. Generalizations for the product, division, and composite function calculations are complex and best expressed in tensor index form.

Addition

Adding two OCEA variables yields

$$a + b := [a + b, \nabla a + \nabla b, \nabla\nabla a + \nabla\nabla b, \nabla\nabla\nabla a + \nabla\nabla\nabla b,$$
$$\nabla\nabla\nabla\nabla a + \nabla\nabla\nabla\nabla b]$$

Subtraction

Subtracting two OCEA variables yields

$$a - b := [a - b, \nabla a - \nabla b, \nabla\nabla a - \nabla\nabla b, \nabla\nabla\nabla a - \nabla\nabla\nabla b,$$
$$\nabla\nabla\nabla\nabla a - \nabla\nabla\nabla\nabla b]$$

Product Rule

The product of two OCEA variables yields

$$a^*b := \lfloor a^*b, \partial_i(a^*b), \partial_j\partial_i(a^*b), \partial_k\partial_j\partial_i(a^*b), \partial_r\partial_k\partial_j\partial_i(a^*b)\rfloor$$

where

$$\partial_i(a^*b) = ab_{,i} + ba_{,i}$$

$$\partial_j\partial_i(a^*b) = a_{,j}b_{,i} + b_{,j}a_{,i} + ab_{,i,j} + ba_{,i,j}$$

$$\partial_k\partial_j\partial_i(a^*b) = a_{,j,k}b_{,i} + b_{,j,k}a_{,i} + a_{,k}b_{,i,j} + b_{,k}a_{,i,j}$$
$$+ a_{,j}b_{,i,k} + b_{,j}a_{,i,k} + ab_{,i,j,k} + ba_{,i,j,k}$$

$$\partial_r\partial_k\partial_j\partial_i(a^*b) = a_{,j,k,r}b_{,i} + b_{,j,k,r}a_{,i} + a_{,k,r}b_{,i,j} + b_{,k,r}a_{,i,j}$$
$$+ a_{,j,r}b_{,i,k} + b_{,j,r}a_{,i,k} + a_{,r}b_{,i,j,k} + b_{,r}a_{,i,j,k}$$
$$+ a_{,j,k}b_{,i,r} + b_{,j,k}a_{,i,r} + a_{,k}b_{,i,j,r} + b_{,k}a_{,i,j,r}$$
$$+ a_{,j}b_{,i,k,r} + b_{,j}a_{,i,k,r} + ab_{,i,j,k,r} + ba_{,i,j,k,r}$$

where ∂_i is the partial derivative with respect to the ith variable, $i, j, k, r = 1, \ldots, m$, and m is the number of independent variables. The preceding index notation is very useful for identifying the symmetry properties of the objects.

Composite Function Rule

The composite function transformation evaluates $b = b(a)$, where $b \in \{\sin, \cos, \tan, \exp, \ell n, \cosh, \sinh, \tanh, a\sin, a\cos, a\tan, abs,$ etc.$\}$. The transformation follows as $b := \lfloor b, \partial_i b, \partial_j\partial_i b, \partial_k\partial_j\partial_i b, \partial_r\partial_k\partial_j\partial_i b\rfloor$, where

$$\partial_i b = b' a_{,i}$$

$$\partial_j\partial_i b = b'' a_{,i}a_{,j} + b' a_{,i,j}$$

$$\partial_k\partial_j\partial_i b = b''' a_{,i}a_{,j}a_{,k} + b''(a_{,i,k}a_{,j} + a_{,i}a_{,j,k} + a_{,i,j}a_{,k}) + b' a_{,i,j,k}$$

$$\partial_r\partial_k\partial_j\partial_i b = b'''' a_{,i}a_{,j}a_{,k}a_{,r} + b'''(a_{,i,k}a_{,j} + a_{,i}a_{,j,k} + a_{,i,j}a_{,k})a_{,r}$$
$$+ b'''(a_{,i,r}a_{,j}a_{,k} + a_{,i}a_{,j,r}a_{,k} + a_{,i}a_{,j}a_{,k,r})$$
$$+ b''(a_{,i,k,r}a_{,j} + a_{,i,r}a_{,j,k} + a_{,i,j,r}a_{,k} + a_{,i,k}a_{,j,r} + a_{,i}a_{,j,k,r}$$
$$+ a_{,i,j}a_{,k,r}) + b''a_{,i,j,k}a_{,r} + b' a_{,i,j,k,r}$$

$$[b', b'', b''', b''''] = \left[\frac{db}{da}, \frac{d^2b}{da^2}, \frac{d^3b}{da^3}, \frac{d^4b}{da^4}\right]$$

The structure of the a tensors is independent of the library function being processed. The complexities of the transformations arise because of the encoded chain rule operations. For example, if $b = \ell n(a)$, then the composite function transformational partials are given by

$$[b', b'', b''', b''''] = \left[\frac{1}{a}, \frac{-2}{a^2}, \frac{6}{a^3}, \frac{-24}{a^4}\right]$$

Division Rule

A two-step strategy is presented for developing the division rule. The goal is to replace the operation b/a with $b \times h$, where $h = a^{-1}$. Numerical experiments have demonstrated that the two-stage approach is ~30% faster than using a direct OCEA division operator. The first step uses the composite function transformation to generate the reciprocal h variable, where $h', h'', h''', h''''] = [-2/a^2, 6/a^3, -24/a^4, 120/a^5]$. The second step forms the product $b \times h$ using the product operator, which completes the definition of the division rule.

Simple Multiplication Example

A second-order OCEA method is used to multiply xy^2, where x and y are the only OCEA variables, given by

$$x := \left[x, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right], \qquad y := \left[y, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right]$$

Recalling the second-order product rule given by

$$a \times b = \lfloor ab, ab_{,i} + ba_{,i}, a_{,j}b_{,i} + b_{,j}a_{,i} + ab_{,i,j} + ba_{,i,j}\rfloor$$

one computes the OCEA product for y^2 as

$$y^2 = \left[y, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right] \times \left[y, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right]$$
$$= \left[y^2, y\begin{bmatrix}0\\1\end{bmatrix} + y\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}^T + \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}^T\right.$$
$$\left. + y\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix} + y\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right]$$
$$= \left[y^2, \begin{bmatrix}0\\2y\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 2\end{bmatrix}\right]$$

Next, completing the product xy^2 yields

$$xy^2 = \left[x, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right] \times \left[y^2, \begin{bmatrix}0\\2y\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 2\end{bmatrix}\right]$$
$$= \left[xy^2, x\begin{bmatrix}0\\2y\end{bmatrix} + y^2\begin{bmatrix}1\\0\end{bmatrix}, x\begin{bmatrix}0 & 0\\0 & 2\end{bmatrix} + y^2\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right.$$

$$+\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}0\\2y\end{bmatrix}^T+\begin{bmatrix}0\\2y\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}^T\Bigg]$$

$$=\left[xy^2,\begin{bmatrix}y^2\\2xy\end{bmatrix},\begin{bmatrix}0&2y\\2y&2x\end{bmatrix}\right]$$

The calculations are straightforward and performed by hidden routines in the OCEA library Module function.

Software Architecture Issues

Module functions and derived data types handle OCEA's core capabilities. Derived data types allow the compiler to detect the data types involved in a calculation and invoke the correct subroutine or function without user intervention. Providing typed data subroutines and functions for all possible data type mathematical operations enables the automated compiler detection capabilities. Three derived data types are required, namely, vector, tensor, and embedded. Table 3 provides a partial list of the hidden capabilities incorporated in Modules PV_Handling, PT_Handling, and EB_Handling for generalizing the assignment (=) operator. The embedded variable Module provides the libraries of chain rule encoded generalized mathematical operators and functions.

Three examples are provided for clarifying the software design issues. The first example adds two vectors. The second example computes an embedded sine function. The third example presents a utility routine for extracting the function, Jacobian, and Hessian parts of a second-order OCEA variable. In the software fragment of Fig. 1, PV denotes the user-defined vector data type. The variable : denotes an array assignment that assigns all components.[28–30] Explicit data typing is required for all variables. Rigid data typing is very useful because the compiler can find the correct function by 1) identifying the operator, 2) checking the interface operator definitions for generalized names for the operator, and 3) matching the input data types for the calculation.

Adding Vector Data Types

At compile time, when the compiler encounters a statement that adds two user-defined vector data types, it automatically builds a link that calls the function routine FUNCTION ADD_PV in Module PV_HANDLING (Fig. 1). Symbolically this equation looks like

$$\underbrace{\text{PV\_TYPE} = \text{PV\_TYPE} + \text{PV\_TYPE}}_{\Uparrow \quad \Downarrow \quad \Downarrow}$$

Detected by Interface Operator

Detected by Function Input Data Types

To identify the correct hidden function call, the compiler first identifies the operator, in this case the addition operator (+). Next, the compiler identifies the data type(s) involved in the operation. Because user-defined data types are detected, that is, vector data types, enabled by operator-overloading techniques, the PV_Handling Module is called to resolve the specific routine to be invoked. In PV_Handling, an interface operator definition is encountered for addition that handles two vector data types, that is, Module procedure ADD_PV. This information enables the compiler to write a direct link to function ADD_PV at compile time for evaluating this operation during calculations.

Table 3 Operator overloading for user-defined data-type operations

Data types (A and B)	Operations	Mathematical model
Embedded, embedded	+, −, =	$A+B$, $A-B$
Embedded, vector	=	$A=B$
Embedded, vector	=	$A=B$
Vector, scalar	×, =	$A=b\times A$, $A_i=b$, $i=1,\ldots,n$
Tensor, tensor	=	$A=B$
Embedded, embedded	sin	$A=\sin(B)$

```
MODULE PV_HANDLING   !Vector Operations
                .
                INTEFACE OPERATOR (+)
                    MODULE PROCEDURE ADD_PV
                END INTERFACE
                .
                FUNCTION ADD_PV(A,B)
                    TYPE(PV)  ADD_PV
                    TYPE(PV), INTENT(IN)::A,B
                    ADD_PV%VPART(:) = A%VPART(:) + B%VPART(:)
                END FUNCTION ADD_PV
                .
END MODULE PV_HANDLING
```

Fig. 1 Sofware fragment for vector module PV_Handling.

```
FUNCTION EB_SIN( A )
TYPE(EB)      EB_SIN
TYPE(EB), INTENT(IN)::A
TYPE(PT)::AAT
REAL(KIND=8)::F,DF,DDF
INTEGER::I
!VECTOR OUTTER PRODUCT FOR A*A^` FOR TENSOR OPERATOR
DO I=1,NV
   AAT%TPART(:,I) = A%V%VPART(:)*A%V%VPART(I)
END DO
!FUNCTION AND DERIVATIVE DEFINITIONS
F     = SIN( A%E )              !FUNCTION
DF    = COS( A%E )              !FIRST  DERIVATIVE
DDF =-SIN( A%E )                !SECOND DERIVATIVE
!EMBEDDED FUNCTION DEFINITIONS
EB_SIN%E =      F;
EB_SIN%V =    DF*A%V;
EB_SIN%T = DDF*AAT + DF*A%T
END FUNCTION EB_SIN
```

Fig. 2 Embedded sine function.

Evaluating a Second-Order OCEA Sine Function

An OCEA sine function is a mathematical library object that is included in Module EB_Handling (Fig. 2). A composite function calculation for the sine function yields

$$\sin(A)=\lfloor \sin(A),\ \partial_i \sin(A),\ \partial_j\partial_i \sin(A)\rfloor$$

$$=[F,\ DF\times A_{,i},\ DDF\times A_{,i}A_{,j}+DF\times A_{,i,j}]$$

where $F=\sin(A\%E)$, $DF=\cos(A\%E)$, $DDF=-\sin(A\%E)$, $A_{,i}=A\%V$, $A_{,i}A_{,j}=AAT\%T$, that is, vector outer product, and $A_{,i,j}=A\%T$. Very compact models are developed because user-defined data types for scalar, vector, and tensor objects have been defined. For example, the product $DF\times A_{,i,j}$ is evaluated in the Module PT_Handling, where a scalar and tensor product is defined.

Extracting Partial Derivative Numerical Data

Function and partial derivative values are recovered by using utility routines, such as the partition–vector–data subroutine presented in Fig. 3. The input for the utility routine consists of an OCEA variable. The utility routine unpacks the OCEA compound data object by using the *%E*, *%V*, and *%T* operators. The output consists of 1) the vector part of the data, that is, VEC, 2) the Jacobian part of the data, that is, JAC, and 3) the Hessian part of the data, that is, HES.

OCEA Applications

Three applications are presented for exploiting OCEA algorithms. First, a fourth-order perturbation technique is presented

```
      SUBROUTINE PARTITION_VECTOR_DATA( VAR, VEC, JAC, HES )
!  THIS PROGRAM ACCEPTS AN EMBEDDED VECTOR-VALUED VARIABLE
!  AND RETURNS F(X), JACOBIAN( F(X) ), AND SYMMETRIC HESSIAN( F(X) ).
!
!..INPUT
!        VAR   VECTOR VARIABLE CONTAINING FUNCTION, 1ST, AND 2ND PARTIALS
!..OUTPUT
!      [VEC, JAC, HES ] = [ F(X),  JACOBIAN( F(X) ), HESSIAN  ( F(X) ) ]
       USE EB_HANDLING  !OCEA MODULE
       TYPE(EB),        DIMENSION(NF),        INTENT(IN):: VAR
       REAL(KIND=8), DIMENSION(NF),           INTENT(OUT)::VEC
       REAL(KIND=8), DIMENSION(NF,NF),        INTENT(OUT)::JAC
       REAL(KIND=8), DIMENSION(NF,NF,NF), INTENT(OUT)::HES
       INTEGER::P1, P2
!...FETCH SCALAR PART OF INPUT VECTOR
       VEC = VAR%E
!...FETCH JACOBIAN PART OF INPUT VECTOR
       DO P1 = 1, NV
          JAC( :, P1 ) = VAR%V%VPART( P1 )
!...FETCH HESSIAN PART OF INPUT VECTOR (Symmetric Part)
          DO P2 = P1, NV
             HES( :, P1, P2 ) = VAR%T%TPART( P1, P2 )
          END DO
       END DO
       RETURN
       END SUBROUTINE PARTITION_VECTOR_DATA
```

Fig. 3 Utility routine for extracting the scalar, Jacobian, and Hessian.

for approximately solving nonlinear vector-valued sets of algebraic equations. Second, equation of motion models for linked mechanical systems are presented using OCEA generated partials for Lagrange's method.[20] Third, OCEA is used to obtain the sensitivity partial derivatives for a first-order linear matrix–vector differential equation and its associated state and parameter transition matrices.

Vector-Valued Algebraic Perturbation Method

A fourth-order power series is assumed for the solution of a vector set of algebraic equations. The unknowns in the power series are the vector-valued derivatives. The analytical calculations for the implicit derivatives are simplified by embedding the nonlinear algebraic equations into a space of higher dimension.[23] OCEA is used to evaluate the transformed set of algebraic equations. This calculation provides numerical values for the gradientlike terms appearing in the implicit derivative expressions. After computing the derivative rates and introducing the results into the assumed power series, the complete perturbation solution is obtained.

Perturbation Mathematical Model

The mathematical model for a nonlinear perturbation problem is given by $\boldsymbol{f}[\boldsymbol{x}(p), p] = 0$, where $\boldsymbol{f}[\,]$ is an $n \times 1$ nonlinear vector function, $\boldsymbol{x}(\,)$ is an $n \times 1$ unknown solution vector, and p is a scalar perturbation parameter. A starting solution is assumed available for solving $\boldsymbol{f}[\boldsymbol{x}(0), 0] = 0$ for $\boldsymbol{x}(0)$. A Taylor series is assumed for analytically continuing the starting guess for $\boldsymbol{x}(0)$ as a function of p, as

$$x(p) = x|_{p=0} + \frac{p}{1!}\frac{\mathrm{d}x}{\mathrm{d}p}\bigg|_{p=0} + \cdots$$

where the unknowns are the rates for $\boldsymbol{x}(p)$, evaluated about $p = 0$. The implicit nature of $\boldsymbol{f}(\,)$ makes the problem challenging, particularly for expansion orders greater than two.

A two-step transformation process yields the implicit rates. First, the original nonlinear algebraic problem is embedded into a higher-dimensional model[23]:

$$H(y) = q, \qquad H = \begin{pmatrix} f \\ p \end{pmatrix}, \qquad q = \begin{pmatrix} 0 \\ p \end{pmatrix}, \qquad y = \begin{pmatrix} x \\ p \end{pmatrix}$$

where the new equation is obtained by adding the trivial equation $p = p$ to $\boldsymbol{f}(\,)$. Second, both x and p become OCEA variables. The benefit of this transformation is that the original inequality constraint becomes an equality constraint. Assuming a fourth-order OCEA version of the preceding equation, one obtains

$$H = [H, \nabla H, \nabla\nabla H, \nabla\nabla\nabla H, \nabla\nabla\nabla\nabla H]$$

$$q = [q, \nabla q, \nabla\nabla q, \nabla\nabla\nabla q, \nabla\nabla\nabla\nabla q]$$

where the dimension of $H[\,]$ is $m = n + 1$. The higher-order gradientlike structures of $H[\,]$ are exploited for evaluating the implicit rates. The following fourth-order Taylor series defines the solution for the perturbation problem

$$y(p) = y|_{p=0} + \frac{p}{1!}\frac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} + \frac{p^2}{2!}\frac{\mathrm{d}^2y}{\mathrm{d}p^2}\bigg|_{p=0} + \frac{p^3}{3!}\frac{\mathrm{d}^3y}{\mathrm{d}p^3}\bigg|_{p=0} + \frac{p^4}{4!}\frac{\mathrm{d}^4y}{\mathrm{d}p^4}\bigg|_{p=0} + \cdots \tag{1}$$

where the unknowns are the vector-valued derivatives. The OCEA version of p is evaluated at $p = 0$ by defining

$$p = \left[0_{1\times 1} \quad \begin{pmatrix} 0_{n\times 1} \\ 1 \end{pmatrix} \quad 0_{m\times m} \quad 0_{m\times m\times m} \quad 0_{m\times m\times m\times m}\right]$$

The partial derivatives for $\boldsymbol{y}$ are obtained from the augmented equation for $\boldsymbol{H}(\,)$, by repeatedly partially differentiating $\boldsymbol{H}(\,)$ with respect to p, yielding

$$\frac{\partial H}{\partial p} = \frac{\partial q}{\partial p}, \qquad \frac{\partial^2 H}{\partial p^2} = 0, \qquad \frac{\partial^3 H}{\partial p^3} = 0, \qquad \frac{\partial^4 H}{\partial p^4} = 0$$

When it is recalled that $\boldsymbol{H} = \boldsymbol{H}[\boldsymbol{y}(p)]$, these implicit equations yield

$$\frac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} = (\nabla H)^{-1}\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\frac{\mathrm{d}^2y}{\mathrm{d}p^2}\bigg|_{p=0} = -(\nabla H)^{-1}\nabla\nabla H \cdot \frac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \frac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0}$$

$$\frac{\mathrm{d}^3y}{\mathrm{d}p^3}\bigg|_{p=0} = -(\nabla H)^{-1}\left\{\begin{array}{l} \nabla\nabla\nabla H \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \\ +2\nabla\nabla H \cdot \dfrac{\mathrm{d}^2y}{\mathrm{d}p^2}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \\ +\nabla\nabla H \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}^2y}{\mathrm{d}p^2}\bigg|_{p=0} \end{array}\right\}$$

$$\frac{\mathrm{d}^4y}{\mathrm{d}p^4}\bigg|_{p=0} =$$

$$-(\nabla H)^{-1}\left\{\begin{array}{l} \nabla\nabla\nabla\nabla H \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \\ +3\nabla\nabla\nabla H \cdot \dfrac{d^2y}{\mathrm{d}p^2}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \\ +2\nabla\nabla\nabla H \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{d^2y}{\mathrm{d}p^2}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \\ +\nabla\nabla\nabla H \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{d^2y}{\mathrm{d}p^2}\bigg|_{p=0} \\ +3\nabla\nabla H \cdot \dfrac{d^3y}{\mathrm{d}p^3}\bigg|_{p=0} \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \\ +3\nabla\nabla H \cdot \dfrac{d^2y}{\mathrm{d}p^2}\bigg|_{p=0} \cdot \dfrac{d^2y}{\mathrm{d}p^2}\bigg|_{p=0} \\ +\nabla\nabla H \cdot \dfrac{\mathrm{d}y}{\mathrm{d}p}\bigg|_{p=0} \cdot \dfrac{d^3y}{\mathrm{d}p^3}\bigg|_{p=0} \end{array}\right\}$$

where the ( ) denote a dot product for $m \times 1$ vectors. The solution is sequential in nature, because higher-order calculations make use of lower-order y-rate solutions. These equations represent a general solution for algebraic vector-valued problems through fourth order. Higher-order extensions for are easily defined. Using the Macsyma 2.4 computer algebra system[31] has validated the basic algorithm. The implicit rate calculations are readily handled by parallel methods.

Vector Perturbation Example Problem

A simple 2×1 nonlinear system of coupled quadraticlike equations is solved using the methods presented in the preceding section. The governing nonlinear equations are given by

$$x^2 - p/z^2 - 1/100 = 0, \qquad z^2 + 2z - px^2/z - 3 = 0$$

Setting $p = 0$ in the preceding equations yields four starting solutions:

$$x^2 - 1/100 = 0 = > x(0) = \pm 1/10$$

$$z^2 + 2z - 3 = 0 = > z(0) = 1, -3$$

Evaluating the y-rate equations, using the starting values just computed, and introducing the results into the Taylor series expansion of Eq. (1) leads to

$$x \approx x_0 + \frac{p}{2x_0 z_0^2} - \frac{p^2\left(z_0 + 4x_0^4 + 1\right)}{4x_0^3 z_0^4 (z_0 + 1)} + \mathcal{O}(p^3, p^4)$$

$$z \approx z_0 + \frac{px_0^2}{2z_0(z_0+1)} + \frac{p^2\left(4z_0^2 - 3x_0^4 z_0 + 8z_0 - 2x_0^4 + 4\right)}{4z_0^3(z_0+1)^3} + \mathcal{O}(p^3, p^4)$$

Macsyma 2.4 has been used to validate the OCEA generated expansions.[31] An F90 version of the solution only provides numerical results: no symbolic solutions. A convergence history for the perturbation solution is presented in Table 4, where the starting solution is $[x(0), z(0)] = (0.1, -3.0)$ and $p = 0.005$. As shown in Table 4, as the order of the perturbation expansion increases, the x and z solutions rapidly converge.

OCEA Lagrangian Dynamics

A standard problem in analytical dynamics consists of using Lagrange's equations[20] for generating equations of motion for linked mechanical systems.[21–24] The mathematical model for the physical system is obtained by mechanically differentiating the scalar Lagrangian function and assembling first- and second-order partial derivatives of the system kinetic energy. An OCEA version of Lagrange's method is obtained by defining the generalized coordinates for q and $\dot{q}$ as OCEA variables. All of the first and second-order partial derivatives required by Lagrange's equations are obtained by introducing OCEA processed velocities into the system kinetic energy. A general OCEA-based Lagrangian mathematical model and a simple planer example are presented.

Lagrangian Mathematical Model

The mathematical modeling technique used in this section is based on the work of Lagrange (1736–1813), who published his analytical dynamics method in 1788.[20] His methods remain attractive today for two reasons. First, the process of generating the equations of motion is reduced to performing mechanical differentiation of a scalar function. Second, his method provides an automated way to eliminate topology-based constraint forces and torques. Lagrange's equation is obtained by applying the calculus of variations for Hamilton's principle (see Refs. 21–24):

$$\delta \int_{t_1}^{t_2} (T - V)\, \mathrm{d}t = 0, \qquad \delta(t_1) = 0, \qquad \delta(t_2) = 0$$

where $\boldsymbol{q}$ denotes the $m \times 1$ vector of generalized coordinates, the time interval endpoints are assumed to be fixed, and δ is the usual variational symbol. Application of Hamilton's principle leads to the Lagrangian equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i, \qquad i = 1, \ldots, m$$

where $L = T - V$ denotes the scalar Lagrangian function, $T = T(q, \dot{q})$ denotes the kinetic energy, $V = V(q)$ denotes the potential energy, and from the virtual work

$$\delta W = \sum_{i=1}^{N} Q_i \delta q_i$$

the generalized force is defined by Q_i. Lagrange's equation is valid for independent generalized coordinates. OCEA is ideally suited for this application because it handles all required mechanical differentiations of the scalar Lagrange function.

The generalized coordinates and coordinate rates are given by q and $\dot{q}$, where both are treated as OCEA variables. The body reference point position and velocity vectors are defined by $r_i = r_i(q)$ and $\boldsymbol{v}_i = \nabla r_i(q)\dot{q}$, where the gradient of the position vector is obtained from an OCEA calculation for the position vector. The OCEA kinetic energy (KE) is computed as

$$\mathrm{KE} = \sum_{i=1}^{N_B} m_i v_i \cdot v_i = \sum_{i=1}^{N_B} m_i (\boldsymbol{v}_{ix}\boldsymbol{v}_{ix} + \boldsymbol{v}_{iy}\boldsymbol{v}_{iy} + \boldsymbol{v}_{iz}\boldsymbol{v}_{iz})$$

where N_B is the number of linked bodies and the OCEA product rule is used for computing the products of the velocity components. After evaluating the OCEA kinetic energy, one obtains

$$\mathrm{KE} := \left[\mathrm{KE}, \begin{bmatrix} \mathrm{KE}_{,q} \\ \mathrm{KE}_{,\dot{q}} \end{bmatrix}, \begin{bmatrix} \mathrm{KE}_{,q,q} & \mathrm{KE}_{,q,\dot{q}} \\ \mathrm{KE}_{,\dot{q},q} & \mathrm{KE}_{,\dot{q},\dot{q}} \end{bmatrix}\right]$$

where all of the partial derivative partitions have been automatically assembled.

When it is assumed that the potential energy vanishes, the equations of motion are obtained by evaluating the following form of Lagrange's equation (see Refs. 1, 2, and 21–25):

$$\mathrm{KE}_{,\dot{q},\dot{q}} \cdot \ddot{q} = Q + \mathrm{KE}_{,q} - \mathrm{KE}_{,\dot{q},q} \cdot \dot{q}$$

where the generalized force is defined by

$$Q = \sum_{i=1}^{N_B} F_i \cdot \boldsymbol{v}_{i,q}$$

The OCEA form of the KE clearly provides all of the vector and matrix partitions required in Lagrange's equation. Accordingly, one can model arbitrarily complex mechanical systems by letting OCEA carryout the mechanical first- and second-order differentiations for the KE.

Planar Dynamics Problem

A planar example problem is considered in this section. The mechanical system consists of two bodies. The first body consists of a cart whose attached wheels interact with a rigid frictionless surface, where only translational motion is allowed. The motion of the body is restrained by a linear spring that is attached to rigid wall (Fig. 4). The second body is attached to the first body by a rigid massless

Table 4 Summary of perturbation solutions and necessary condition errors

	X and Y solutions		Equation errors	
Order	X	Y	$E1$	$E2$
0	0.100000000	−3.00000000	−5.5(−4)	1.9(−5)
1	0.102777777	−2.99999583	7.7(−6)	3.7(−6)
2	0.102739205	−2.99999560	2.1(−7)	−3.3(−10)
3	0.102740277	−2.99999560	7.3(−9)	1.6(−11)
4	0.102740240	−2.99999560	2.9(−10)	−4.8(−13)

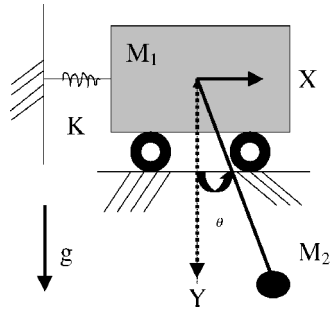

Fig. 4 Planar cart–pendulum dynamics problem.

link, where only rotational motion is allowed about the common hinge that connects both bodies. Both bodies experience gravity loads, which are assumed to act in the y direction. The motion of the system is completely defined by two generalized coordinates, x and θ, where the position and velocity vectors are defined by

$$r_1 = x\hat{\boldsymbol{i}}, \qquad r_2 = [x + l\sin(\theta)]\hat{\boldsymbol{i}} + l\cos(\theta)\hat{\boldsymbol{j}}$$

$$\boldsymbol{v}_1 = \dot{x}\hat{\boldsymbol{i}}, \qquad v_2 = [\dot{x} + l\dot{\theta}\cos(\theta)]\hat{\boldsymbol{i}} - l\dot{\theta}\sin(\theta)\hat{\boldsymbol{j}}$$

where $\hat{i}$ is the unit vector in the x direction, $\hat{j}$ is the unit vector in the y direction, and the overdot denotes the time derivative.

The KE for the system is given by

$$\mathrm{KE} = m_1(\boldsymbol{v}_{1x}\boldsymbol{v}_{1x} + \boldsymbol{v}_{1y}\boldsymbol{v}_{1y})/2 + m_2(\boldsymbol{v}_{2x}\boldsymbol{v}_{2x} + \boldsymbol{v}_{2y}\boldsymbol{v}_{2y})/2$$

$$= (m_1 + m_2)\dot{x}^2/2 + lm_2\cos(\theta)\dot{\theta}\dot{x} + l^2m_2\dot{\theta}^2/2$$

The equation of motion is obtained by evaluating the following Lagrangian equations:

$$\frac{\mathrm{d}}{\mathrm{dt}}\frac{\partial \mathrm{KE}}{\partial \dot{x}} - \frac{\partial \mathrm{KE}}{\partial x} = Q_x, \qquad \frac{\mathrm{d}}{\mathrm{dt}}\frac{\partial \mathrm{KE}}{\partial \dot{\theta}} - \frac{\partial \mathrm{KE}}{\partial \theta} = Q_\theta$$

Lagrange's method requires a second-order OCEA method for building the required first and second-order partials. Higher-order OCEA methods yield both the equation of motion and sensitivity models for the equations of motion. The analyst only builds the KE. Hidden OCEA tools automatically generate the required partial derivatives. For the planar cart–pendulum problem, one defines x, θ, $\dot{x}$, and $\dot{\theta}$ to be OCEA variables, where the velocity components follow as

$$\boldsymbol{v}_{1x} := \left[\dot{x}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\right],$$

$$\boldsymbol{v}_{1y} := \left[0, \begin{bmatrix}0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\right]$$

$$\boldsymbol{v}_{2x} := \left[\dot{x} + l\dot{\theta}\cos(\theta), \begin{bmatrix}0\\-l\dot{\theta}\sin(\theta)\\1\\l\cos(\theta)\end{bmatrix}, \begin{bmatrix}0&0&0&0\\0&-l\dot{\theta}\cos(\theta)&0&-l\sin(\theta)\\0&0&0&0\\0&-l\sin(\theta)&0&0\end{bmatrix}\right]$$

$$\boldsymbol{v}_{2y} := \left[-l\dot{\theta}\sin(\theta), \begin{bmatrix}0\\-l\dot{\theta}\cos(\theta)\\0\\-l\sin(\theta)\end{bmatrix}, \begin{bmatrix}0&0&0&0\\0&l\dot{\theta}\sin(\theta)&0&-l\cos(\theta)\\0&0&0&0\\0&-l\cos(\theta)&0&0\end{bmatrix}\right]$$

Introducing the OCEA velocities into the KE, one obtains KE := [KE, ∇KE, $\nabla\nabla$KE], where

$$\mathrm{KE}\%E = \mathrm{KE} = (m_1 + m_2)\dot{x}^2/2 + lm_2\cos(\theta)\dot{\theta}\dot{x} + l^2m_2\dot{\theta}^2/2$$

$$\mathrm{KE}\%V = \nabla\,\mathrm{KE} = \begin{bmatrix}\mathrm{KE}_{,q}\\ \mathrm{KE}_{,\dot{q}}\end{bmatrix} = \begin{bmatrix}0\\ -lm_2\sin(\theta)\dot{\theta}\dot{x}\\ (m_1+m_2)\dot{x} + lm_2\cos(\theta)\dot{\theta}\\ m_2[\cos(\theta)\dot{x} + l^2\theta]\end{bmatrix}$$

$$\mathrm{KE}\%T = \nabla\nabla\,\mathrm{KE} = \begin{bmatrix}\mathrm{KE}_{,q,q} & \mathrm{KE}_{,q,\dot{q}}\\ \mathrm{KE}_{,\dot{q},q} & \mathrm{KE}_{,\dot{q},\dot{q}}\end{bmatrix}$$

$$= \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & -lm_2\cos(\theta)\dot{\theta}\dot{x} & -lm_2\sin(\theta)\dot{\theta} & -lm_2\sin(\theta)\dot{x}\\ 0 & -lm_2\sin(\theta)\dot{\theta} & m_1+m_2 & lm_2\cos(\theta)\\ 0 & -lm_2\sin(\theta)\dot{x} & lm_2\cos(\theta) & l^2m_2\end{bmatrix}$$

where $\%E$, $\%V$, and $\%T$ have been used to unpack the OCEA version of the KE. Introducing these partitions into the equation of motion, $\mathrm{KE}_{,\dot{q},\dot{q}}\cdot\ddot{q} = Q + \mathrm{KE}_{,q} - \mathrm{KE}_{,\dot{q},q}\cdot\dot{q}$, leads to

$$\begin{bmatrix}m_1+m_2 & lm_2\cos(\theta)\\ lm_2\cos(\theta) & l^2m_2\end{bmatrix}\cdot\ddot{q}$$

$$= \begin{bmatrix}Q_x\\ Q_\theta\end{bmatrix} + \begin{bmatrix}0\\ -lm_2\sin(\theta)\dot{\theta}\dot{x}\end{bmatrix} - \begin{bmatrix}0 & -lm_2\sin(\theta)\dot{\theta}\\ 0 & -lm_2\sin(\theta)\dot{x}\end{bmatrix}\cdot\dot{q}$$

which reduces to

$$\begin{bmatrix}m_1+m_2 & lm_2\cos(\theta)\\ lm_2\cos(\theta) & l^2m_2\end{bmatrix}\cdot\ddot{q} = \begin{bmatrix}Q_x + lm_2\sin(\theta)\dot{\theta}^2\\ Q_\theta\end{bmatrix}$$

In real-world OCEA applications, the analytic variable expressions are replaced with numerical values.

Differential Equation Sensitivity

A first-order differential equation is cast in the form $M(\boldsymbol{x})\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t; \boldsymbol{p})$, where $\boldsymbol{x}$ is the $n \times 1$ state vector and $\boldsymbol{p}$ is a $m \times 1$ vector of modeling parameters. The goal of the analysis is to obtain partial derivatives of the system response with respect to $\boldsymbol{x}$ and $\boldsymbol{p}$. The system parameters can consist of initial conditions for the state vector $\boldsymbol{x}$, control and/or estimator gains, manufacturing misalignment data, structural stiffness and damping data, etc.

Standard First-Order Matrix Sensitivity Model

The standard solution procedure consists of inverting $\boldsymbol{M}(\boldsymbol{x})$, yielding $\dot{x} = M(x)^{-1}f(x, t, p)$, where the state sensitivity partial derivative follows as

$$\dot{x}_{,x} = -M(x)^{-1}M(x)_{,x}M(x)^{-1}f(x, t, p) + M(x)^{-1}f_{,x}$$

and the parameter sensitivity partial derivative follows as

$$\dot{x}_{,p} = -M(x)^{-1}\{M(x)_{,x}x_{,p}\}M(x)^{-1}f(x, t, p) + M(x)^{-1}(f_{,x}x_{,p} + f_{,p})$$

where the inverse matrix partials are complicated and state transition matrix are obtained by assuming appropriate initial conditions for integrating the matrix differential equations.

Standard Second-Order Calculation

The first-order matrix–vector partial derivative models are extended to second-order partial derivative models. Analytically

computing the second-order partial derivatives leads to

$$\dot{x}_{,p,q} = (M^{-1})_{,p,q} f + (M^{-1})_{,p} f_{,q} + (M^{-1})_{,q} f_{,p} + M^{-1} f_{,p,q}$$

where $(M^{-1})_{,p} = -M^{-1} M_{,p} M^{-1}$, and

$$(M^{-1})_{,p,q} = M^{-1} M_{,p} M^{-1} M_{,q} M^{-1} + M^{-1} M_{,q} M^{-1} M_{,p} M^{-1} - M^{-1} M_{,p,q} M^{-1}$$

By counting the operations for the second-order inverse matrix partials, one finds that $(5n^2 + 11n)/2$ $n \times n$ matrix products must be formed (fully accounting for symmetry). Consequently, for a medium-sized system where $n = 100$, one must evaluate 25,550 100×100 matrix products. Alternatively, OCEA generates the first- and second-order rates without ever explicitly forming any of the complicated matrix products.

OCEA Matrix Sensitivity Model

The analytical sensitivity models presented earlier are replaced with much simpler OCEA-based models. Here, x, p, $M(x)$, and $f(x, t; p)$ are developed using OCEA algebra. An OCEA version of Gaussian elimination (or other related matrix solver that has been extended for OCEA capabilities) is used for solving the linear equation, thereby avoiding the explicit calculation of $\boldsymbol{M}(\boldsymbol{x})^{-1}$. For a second-order version of OCEA, the linear matrix equation is defined by $M(x)\dot{x} = f(x, t; p)$, where $M(x) := [M, \nabla M, \nabla\nabla M]$, $f(x, p) := [f, \nabla f, \nabla\nabla f]$, and $\dot{x} := [\dot{x}, \nabla\dot{x}, \nabla\nabla\dot{x}]$. The solution is $\dot{x} := [M^{-1} f, \vec{\nabla}(M^{-1} f), \vec{\nabla}\vec{\nabla}(M^{-1} f)]$, where the matrix partials are implicitly defined.

An OCEA-enhanced classical Gaussian elimination algorithm has been developed and tested that only requires the introduction of a single Use EB_Handling Module command and typing variables as needed. All data are passed through the procedure argument list, where the original argument list has been retained. The generalized intrinsic operators and functions handle all of the details. The resulting solutions have been verified by using symbolic methods by analytically inverting $M(x)$, analytically computing the first- and higher-order partials of $M^{-1} f$, and comparing double precision numerical results for OCEA and the analytically derived models, as shown next.

OCEA supports general matrix operations for linear equations solutions, eigenproblems, inversion, etc., because are all matrix algorithms are ultimately defined in terms of elementary computational steps involving scalar operations, which are managed by OCEA operators and functions.

Example problem for state and parameter sensitivity calculations. Assume a first-order linear matrix differential equation of the form $M(x)\dot{x} = f(x, t; p)$, where $M(x)$ and $f(x, t; p)$ follow as

$$M(x) = \begin{bmatrix} 20x_1^2 & 5x_1x_2 \\ 5x_1x_2 & 10x_1/x_2 \end{bmatrix}, \qquad f(x, t; p) = \begin{pmatrix} px_1x_2^2 \\ \sqrt{5x_2} \end{pmatrix}$$

and $x = (x_1, x_2)$. The analytical solution for the differential equation can be shown to be

$$\dot{x} = M(x)^{-1} f(x, t, p), \quad = \frac{1}{5x_1x_2^3 - 40x_1^2} \begin{pmatrix} \sqrt{5}x_2^{\frac{5}{2}} - 2px_1x_2^2 \\ px_1x_2^4 - 4\sqrt{5}x_1x_2^{\frac{3}{2}} \end{pmatrix}$$

This model is used for computing analytic sensitivity models.

Analytical state and parameter partial derivative models. The analytical state partial derivatives can be shown to be

$$\frac{\partial\dot{x}}{\partial x_1} = -\begin{pmatrix} \sqrt{x_2}\left(\sqrt{5}x_2^{\frac{5}{2}} - 16\sqrt{5}x_1x_2^2\right) + 16x_1^2x_2^2 \\ 32\sqrt{5}x_2^{\frac{7}{2}} - 8px_2^6 \end{pmatrix} \Big/ \left(5x_1^2x_2^6 - 80x_1^3x_2^3 + 320x_1^4\right)$$

$$\frac{\partial\dot{x}}{\partial x_2} = \begin{bmatrix} -\sqrt{5}x_2\left(x_2^4 + 40x_1x_2\right) + 4px_1x_2^4 + 64px_1^2x_2 \\ 2px_1x_2^6 + \sqrt{5x_2}\left(12x_1x_2^3 + 96x_1^2\right) - 64px_1^2x_2^3 \end{bmatrix} \Big/ \left(10x_1x_2^6 - 160x_1^2x_2^3 + 640x_1^3\right)$$

and parameter derivative is given by

$$\frac{\partial\dot{x}}{\partial p} = \begin{pmatrix} -2x_2^2 \\ x_2^4 \end{pmatrix} \Big/ \left(5x_2^3 - 40x_1\right)$$

These equations are used to validate an OCEA-based solution.

OCEA state and parameter partial derivative models. For a first-order OCEA method for modeling the differential equation, the OCEA variables are x_1, x_2, and p, and the OCEA forms for $\boldsymbol{M}$ and $\boldsymbol{f}$ follow as

$$m_{11} := \left[20x_1^2, \begin{pmatrix} 40x_1 \\ 0 \\ 0 \end{pmatrix}\right], \qquad m_{12} := \left[5x_1x_2, \begin{pmatrix} 5x_2 \\ 5x_1 \\ 0 \end{pmatrix}\right]$$

$$m_{22} := \left[\frac{10x_1}{x_2}, \begin{pmatrix} 10/x_2 \\ -10x_1/x_2^2 \\ 0 \end{pmatrix}\right]$$

$$f_1 := \left[px_1x_2^2, \begin{pmatrix} px_2^2 \\ 2px_1x_2 \\ x_1x_2^2 \end{pmatrix}\right], \qquad f_2 := \left[\sqrt{5x_2}, \begin{pmatrix} 0 \\ \sqrt{5}/2\sqrt{x_2} \\ 0 \end{pmatrix}\right]$$

The analytic solution for the differential equation rate equation is given by

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} := \begin{pmatrix} m_{22}f_1 - m_{12}f_2 \\ -m_{12}f_1 + m_{11}f_2 \end{pmatrix} \Big/ (m_{11}m_{22} - m_{12}m_{12})$$

where OCEA operations are implied. The OCEA and analytical results are compared by introducing the parameter values $x_1 = 1.5$, $x_2 = 2.3$, and $p = 5$, leading to

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} \left[-49.0305, \begin{pmatrix} -2358.32 \\ 4619.85 \\ -12.6707 \end{pmatrix}\right] \\ \left[130.206, \begin{pmatrix} 6237.4 \\ -12106.4 \\ 33.5139 \end{pmatrix}\right] \end{pmatrix}$$

which reproduces the numerical results above to 12+ digits, thereby validating the OCEA approach. One should observe that the solution for the linear matrix equation is identical to the classical solution.

State and Parameter Transition Matrix Algorithms

The governing equations for a first-order state and parameter state transition matrix differential equations are defined by

$$\frac{\partial\dot{x}}{\partial x_0} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial x_0}, \qquad \left.\frac{\partial x}{\partial x_0}\right|_{t=t_0} = I_{n\times n}$$

$$\frac{\partial\dot{x}}{\partial p} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial f}{\partial p}, \qquad \left.\frac{\partial x}{\partial p}\right|_{t=t_0} = 0_{n\times n}$$

The partial derivatives are easily identified as the equations computed in the "Analytical State and Parameter Partial Derivative Models" section. These calculations are easily automated by defining new derived data types or a generalized composite function operator.

Conclusions

An object-oriented operator overloading technique, OCEA, has been presented that introduces hidden artificial problem dimensions to store and evaluate partial derivative models for arbitrarily complex engineering and scientific systems. OCEA provides a rational process for generating sensitivity models by creating a new level of scientific and engineering software data abstraction. OCEA operator-overloading tools allow analysts to develop and code mathematical models in a familiar high-level computer language. Several essential principles drive the agenda for this development effort: 1) identifying the model development barriers for rapidly developing exact high-order application sensitivity models, and working out solutions for these barriers; 2) developing methods and techniques for providing commanded level of accuracy results, especially for high-dimensional models; 3) working with end-user application specialists for defining research directions and software solution packaging requirements; and 4) focusing on methods and systems for deploying integrated solution approaches.

OCEA has broad potential use for impacting the design and use of mathematical programming tools for applications in science and engineering. OCEA software replaces a time-consuming, error-prone, labor-intensive, and costly endeavor, with an automated tool that is capable of generating exact high-order partial derivatives models for arbitrarily complex systems. Future generalizations will extend the current algorithm for handling large-scale sparse applications.

References

[1]Turner, J., "Object Oriented Coordinate Embedding Algorithm For Automatically Generating the Jacobian and Hessian Partial Derivatives of Nonlinear Vector Function," Invention Disclosure, Univ. of Iowa, Iowa City, IA, May 2002.

[2]Turner, J., "The Application of Clifford Algebras for Computing the Sensitivity Partial Derivatives of Linked Mechanical Systems," Nonlinear Dynamics and Control, USNCTAM14: Fourteenth U.S. National Congress of Theoretical and Applied Mechanics, Blacksburg, VA, June 2002.

[3]Griewank, A., "On Automatic Differentiation" *Mathematical Programming: Recent Developments and Applications*, edited by M. Iri and K. Tanabe, Kluwer Academic, Amsterdam, 1989, pp. 83–108.

[4]Bischof, C., Carle, A., Corliss, G., Griewank, A., and Hovland, P., "ADIFOR: Generating Derivative Codes from Fortran Programs," *Scientific Programming*, Vol. 1, 1992, pp. 1–29.

[5]Bishchof, C., Carle, A., Khademi, P., Mauer, A., and Hovland, P., "ADIFOR 2.0 User's Guide (Revision C)," Mathematics and Computer Science Div., Technical Rept. ANL/MCS-TM-192, Argonne National Lab., Argonne, IL, 1995.

[6]Eberhard, P., and Bischof, C., "Automatic Differentiation of Numerical Integration Algorithms," Mathematics and Computer Science Div., Technical Rept. ANL/MCS-P621-1196, Argonne National Lab., Argonne, IL, 1996.

[7]Hovland, P., and Heath, M., "Adaptive SOR: A Case Study in Automatic Differentiation of Algorithm Parameters," Mathematics and Computer Science Div., Technical Rept. ANL/MCS-P673-0797, Argonne National Lab., Argonne, IL, 1997.

[8]Pryce, J. D., and Reid, J. K., "AD01, A Fortran 90 code for Automatic Differentiation," Rept. RAL-TR-1998-057, Rutherford Appleton Lab., Oxfordshire, U.K., 1998.

[9]Griewank, A., Juedes, D., and Utke, J., "Algorithm 755; ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++," *ACM Transactions on Mathematical Software*, Vol. 22, No.1, 1996, pp. 131–167.

[10]Coleman, T. F., and Verma, A., "The Efficient Computation of Sparse Jacobian Matrices Using Automatic Differentiation," *SIAM Journal on Scientific Computing*, Vol. 19, No. 4, 1998, pp. 1210–1233.

[11]Coleman, T. F., and Verma, A., "ADMIT-1: Automatic Differentiation and MATLAB Interface Toolbox," *ACM Transactions on Mathematical Software*, Vol. 26, No. 1, 2000, pp. 150–175.

[12]Tolsma, J. E., and Barton, P. I., "Hidden Discontinuities and Parametric Sensitivity Calculations," *SIAM Journal on Scientific Computing*, Vol. 23, No. 6, 2002, pp. 1861–1874.

[13]Stamatiadis, S., Prosmiti, R., and Farantos, S. C., "AUTO_Deriv: Tool for Automatic Differentiation of a FORTRAN Code," *Computer Physics Communication*, Vol. 127, 2000, pp. 343–355.

[14]Chinchalkar, S., "The Application of Automatic Differentiation to Problems in Engineering Analysis," *Computer Methods in Applied Mechanics and Engineering*, Vol. 118, 1994, pp. 197–207.

[15]Karczmarczuk, J., "Functional Differentiation of Computer Programs," *Higher-Order and Symbolic Computation*, Vol. 14, 2001, pp. 35–57.

[16]Elizondo, D., Cappelaere, B., and Faure, C., "Automatic Versus Manual Model Differentiation to Compute Sensitivities and Solve Non-Linear Inverse Problems," *Computers and Geosciences*, Vol. 28, No. 3, 2002, pp. 309–326.

[17]Stadler, W., and Eberhard, P., "Jacobian Motion and Its Derivatives," *Mechatronics*, Vol. 11, 2001, pp. 563–591.

[18]Li, S., and Petzold, L., "Software and Algorithms for Sensitivity Analysis of Large-Scale Differential Algebraic Systems," *Journal of Computational and Applied Mathematics*, Vol. 125, 2000, pp. 131–145.

[19]Biggs, M. B., Brown, S., Christianson, B., and Dixon, L., "Automatic Differentiation of Algorithms," *Journal of Computational and Applied Mathematics*, Vol. 124, 2002, pp. 171–190.

[20]Lagrange, J. L., *Méchanique Analytique*, Nouvelle Édition, M^{me} V^{e} Courcier, Imprimeur-Libraire pour les Mathématiques, Paris, 1811.

[21]Whittaker, E. T., *Analytical Dynamics of Particles and Rigid Bodies*, Dover, New York, 1944, pp. 41–44.

[22]Pars, L. A., *A Treatise on Analytical Dynamics*, Ox Bow, Woodbridge, CT, 1979, Chap. 5.

[23]Junkins, J. L., and Turner, J. D., *Optimal Spacecraft Rotational Maneuvers: Studies in Astronautics*, Vol. 3, Elsevier Science, Amsterdam, 1986.

[24]Meirovitch, L., *Methods of Analytical Dynamics*, McGraw-Hill, New York, 1970, pp. 157–162.

[25]Likins, P. W., "Analytical Dynamics and Nonrigid Spacecraft Simulation," Jet Propulsion Lab., Technical Rept. 32-1593, California Inst. of Technology, Pasadena, CA, July 1974.

[26]Frisch, H. P., Chun, H. M., and Turner, J. D., "NDISCOS—Users & Programmers Manual," PRA, Technical Rept., Cambridge, MA, Dec. 1992, pp. 1–215.

[27]Chun, H. M., and Turner, J. D., "DISCOS Upgrade for Recursive Dynamics, Final Report," Contract Y03525, Univ. of Iowa, Iowa City, IA, Feb. 1991, pp. 1–50.

[28]Nyhoff, L. R., *Introduction to FORTRAN 90 for Engineers and Scientists*, Prentice–Hall, New York, 1996.

[29]Press, W. H., and Vetterling, W. T., *Numerical Recipes in FORTRAN 90*, Vol. 2, Cambridge Univ. Press, Cambridge, England, U.K., 1996.

[30]Meissner, L. P., *FORTRAN 90*, PWS Publ. Co., 1995.

[31]Symbolic/Numeric/Graphical Mathematics Software, Macsyma®, Release 2.4 for Windows, 1998, Chap. 8.

A. Messac
Associate Editor