Low Operational Order Analytic Sensitivity Analysis
for Tree-type Multibody Dynamic Systems

YuHung Hsu*          Kurt S. Anderson†
Department of Mechanical Engineering, Aeronautical Engineering, and Mechanics
Rensselaer Polytechnic Institute, Troy, NY 12180-3590
(Accepted for publication in the AIAA Journal of Guidance, Control and Dynamics)

Abstract
Computing of first-order sensitivity information is crucial for many gradient-base optimization strategies where the algorithms employed plays a key role on determining the computational efficiency of an optimization process. For complex multibody system optimization problems, the optimization and simulation methodologies used along with their numerical accuracy, stability, and efficiency in performing the first-order sensitivity analysis has become an important subject in properly managing these design problems under time and computational resources constraints. This paper presents an algorithm which is able to determine the state derivatives in a fully recursive manner so to significantly reduce the cost of determining analytic first-order sensitivity information for large scale tree-type dynamic systems. Qualitative and quantitative validation on the operational requirement of the present method are made through analytical means and empirical studies.

1 Introduction
Seeking a best design solution from a group of feasible alternatives has long been a practical subject exhibited in various engineering exercises. These practical exercises have led to the introduction of optimization strategies to formally and systematically perform the search procedures and study design trade-offs through mathematical means. In many contemporary engineering problems, multibody dynamic systems form one of the essential components and represent a very large family of substructures. Thus, the underlying formalism associated with dynamic system modeling as well as numerical efficiency become critical issues for a design optimization process to be effective. However, optimizing the behavior and/or performance of a dynamic system requires a substantial amount of computational effort due to the necessary procedures of solving for state variable derivatives, first-order sensitivities at each integration step, and the subsequent temporal integration of these quantities. To minimize these computational demands when a large scale multibody system is presented, an optimization strategy with low cost per function evaluation and good convergence properties is highly desirable. Within this context, gradient-base optimization methodologies are likely contenders due to their good local convergence characteristics.

A primary task involved in gradient-base optimization strategies is to compute the state sensitivities with respect to each design variable in order to construct the gradient information of objective and constraint functions. Producing these analytic gradients however possess a certain level of computational complexities because of the inherent highly nonlinearity and implicit relationships

*Doctoral Graduate Student. hsuy@rpi.edu
†Assistant Professor. AIAA member. anderk5@rpi.edu
exhibited between state and design variables. Resorting to the numerical finite difference approximation, which perhaps is the most broadly used technique, may be an alternative to yield/estimate the essential gradients. Despite its simplicity, the gradient computed by the finite difference is an approximation which may suffer from sensitivity to magnitude in parameter variations, integration scheme characteristics, and system stiffness. These issues have to be resolved prior the adaptation of the finite difference to a particular multibody system design problem. Moreover, this approach may be numerically expensive due to the necessary additional function evaluations required to compute the perturbed effect of each variable [6]. Based upon the goal of computational efficiency which manifests itself in the form of reduced overall computation time, the finite difference approach may not be suitable for the emerging class of problems with a combination of many degrees of freedom, constraints, and design variables.

The direct differentiation [8][20][22] and the adjoint variable methods [5][12][15][21] are two other competitive analytical techniques with which the gradient information can be determined. From a mathematical perspective, the direct method may be the most straightforward technique due to the usage of chain rule of differentiation to explicitly calculate state sensitivities [8]. Since the procedures to produce sensitivity governing equations in this approach is in many regards similar to those in forming system equations of motion [8][22], a mixed type of analysis routine can be implemented through the combination of both analyses leading to the simultaneous solutions of state derivatives and state sensitivities. Such a combination is advantageous for practical implementation, but its efficiency in yielding key partial derivative quantities greatly depends on the underlying formulation used in the dynamic forward problems [3]. The adjoint variable method, based on variation principles, avoids explicit state sensitivity calculations involved in the direct differentiation approach through introducing a set of so-called adjoint variables [4][13]. Solving a sequence of adjoint relationships yields the design sensitivity vector which directly corresponds to the variation (sensitivity) of performance measures in terms of design variable variations [12].

Both of the aforementioned methodologies are capable of producing the desired derivatives, but they each require that their associated set of sensitivity governing equations be solved explicitly. By direct means, the procedure may be an $O(n^3)$ operation where $n$ is the total number of system degrees of freedom. For a system with $p$ design variables and $m$ independent algebraic constraint equations, the adjoint variable method consequently produces a smaller system of $(n+m+p)$ differential algebraic equations, whereas the direct differentiation counterpart involves $(n+m)(p+1)$ differential algebraic equations (DAE). Thus in many applications the former approach is considered less numerically costly than the latter. Nevertheless, the adjoint variable method requires a significant amount of storage of all state information during dynamic forward problem analysis to allow performing the backward temporal integration in the latter process [8][16]. The solution accuracy may also be affected by the interpolation model used in the backward integration step [9]. Another essential, but major, issue associated with the adjoint variable method is its numerical stability, which has yet to be proven and remains an open question [5][10]. Additionally, a portion of the first-order information, specifically that indicating the derivative of generalized accelerations with respect to system state variables is extremely valuable in the use of implicit integration schemes necessary when dealing with stiff systems. This required explicit information is in generally not directly available in the adjoint variable methods. Due to these aspects, the direct differentiation method is an attractive for producing the necessary first-order gradient information.

In the current applications of direct differentiation methods, Newton-Euler with Cartesian coordinate formulations are often encountered in forming the sensitivity governing equations. Although these formulations can handle constrained dynamic systems quite easily and produce a sparse form of system inertia matrix, they generally produce a maximal set of differential equations [22]. Thus, the dimension of sensitivity governing equations is markedly increased and, if not performed intel-
ligently, the involved differentiation calculation may require an $O(n^4)$ operation. Also, this set of
equations represents a set of DAE of index three which may pose a variety of numerical problems
[7]. Another undesirable phenomena, repeated differentiation, may also arise which is primarily
due to the non-exploitation of the inherent recursive relationships on producing the derivatives of
many kinematic quantities and forming the equations [3]. The advantages provided by the Newton-
Euler formulation unfortunately can be offset by the presented shortcomings, and as a result, the
efficiency of direct differentiation methodologies based upon these formulations can be significantly
degraded on handling large scale design problems.

An innovative first-order sensitivity analysis algorithm is developed in this paper as a means of
circumventing many of the drawbacks exhibited in the direct differentiation methods. Its foundation
in part can be traced to a recursive state space dynamic formulation [1], which has demonstrated
numerical efficiency for large scale dynamic forward problem analysis and simulation. The combina-
tion of the state space representations with implicit equation manipulations results in an extremely
efficient fully recursive procedure on solving state sensitivities as the sensitivity governing equations
are being formed. The studies described herein indicate that, by using local representations of kine-
nematic and kinetic quantities and recursive relationships, a significant reduction on computational
effort of differentiation operation is achievable and the performance of the method depends linearly
on the system degrees of freedom.

2 Recursive First-Order Sensitivity

2.1 Mathematical Preliminaries

The behavior of a multibody dynamic system is determined by the selection of a set of design
variables and its initial consistent states. A quantitative performance measuring index $J$ is then
formulated to transform such a dynamic behavior into a mathematical description during optimization.
In order to facilitate an optimization process, the derivatives of $J$ with respect to a nominal
design variable $p_j$ has to be computed. Mathematically, this derivative is expressed as

$$
\frac{\partial J}{\partial p_j} = J_{p_j} + J_{q_i} \frac{dq_i}{dp_j} + J_{u_i} \frac{du_i}{dp_j} + J_{\dot{u_i}} \frac{d\dot{u}_i}{dp_j},
$$

(1)

where $q_i$s are the generalized coordinates; $u_i$s are the generalized speed; $\dot{u}_i$s are the first
time derivative of $u_i$s. The summation over all repeated indices within each term is implied ($i = 1, \ldots, n;
 j = 1, \ldots, p$). As is with standard indicial notation the “,” used in Eq. (1) signifies a partial
derivative operation. Since $dq_i/dp_j$, $du_i/dp_j$, and $d\dot{u}_i/dp_j$ specify the implicit relationships between
state and design variables in which $dq_i/dp_j$ and $du_i/dp_j$ can be obtained from the first and second
temporal integration, respectively, of $d\dot{u}_i/dp_j$, Eq. (1) clearly indicates that $d\dot{u}_i/dp_j$ is the essential
element to fully determined the value of $dJ/dp_j$. Therefore, computing the first-order gradient
information is now reduced to the problem of acquiring the key values of $d\dot{u}_i/dp_j$.

Conceptually, obtaining these quantities can be achieved by taking total derivatives of the
equations of motion with respect to $p_j$. For an open-loop system, this differential operation can be
proceeded by following equations.

$$
\frac{d}{dp_j} (M \dot{u}) = \frac{d}{dp_j} K_u,
$$

(2)

$$
\left( M_{, p_j} + M_{, q_r} \frac{dq_r}{dp_j} \right) \dot{u} + M_{, u_r} \frac{du_r}{dp_j} = K_{, p_j} + K_{, q_r} \frac{dq_r}{dp_j} + K_{, u_r} \frac{du_r}{dp_j},
$$

(3)

$$
\frac{d\dot{u}_r}{dp_j} = M^{-1} \left[ K_{, p_j} + K_{, q_r} \frac{dq_r}{dp_j} + K_{, u_r} \frac{du_r}{dp_j} - \left( M_{, p_j} + M_{, q_r} \frac{dq_r}{dp_j} \right) \dot{u} \right].
$$

(4)
When an $O(n^3)$ dynamic analysis is the underlying formulation methodology, the greatest computational order, $O(n^4)$, is inevitable to produce $\mathcal{M}_{i}q_r$ for all $q_r$ ($r = 1, \ldots, n$) explicitly. An $O(n^3)$ computational order may further be necessary to yield quantities of $\mathcal{K}_{i}q_r$ and $\mathcal{K}_{i}u_r$. Combining with an $O(n^3)$ for mass matrix inversion and $O(n^2)$ for matrix manipulations, explicitly forming Eq. (4) will results in an $O(n^4)$ method. This cost may be acceptable for a system involving small to modest $n$, but will lead to a drastic increase in computational cost for even a modest increase on $n$. However, such a prohibitive operation order can be reduced significantly through the usage of state space representation in conjunction with a specialize low-order dynamic formulation.

The use of relative coordinates and recursive relations has been shown in many situations to be numerical efficient for producing system equations of motion when using state space (particularly recursive) formulation [1][2][17]. These references demonstrated that generalized acceleration $\ddot{u}$ can be solved for in $O(n)$ operations overall for general multibody dynamic systems in sequential implementations. The very same concept is implemented into sensitivity analysis developed herein which leads to an effective $O(n)$ methodology where the computational cost for yielding total derivatives $\frac{d\hat{i}_i}{dp_j}$ depends linearly on the system degrees of freedom $n$.

### 2.2 Recursive Outward Procedure

The generalized speeds $u_i$ [14] used in recursive relations can be defined as

$$ u_k = \dot{q}_k \quad (k = 1, \ldots, n) \quad \text{(5)} $$

From kinematic relations previously determined for various $O(n)$ algorithms, the following recursive kinematical relationships between body $k$ and its proximal body $Pr[k]$ for angular velocity $N_\omega^k$, velocity $N_v^k$, partial angular velocity $N_\omega_r^k$, and partial velocity $N_v_r^k$ exist:

$$ Pr[k]_\omega^k = \begin{cases} Pr[k]_\omega^k u_r^k & \text{if joint } k \text{ is revolute} \\ 0 & \text{if joint } k \text{ is prismatic} \end{cases} \quad \text{(6)} $$

$$ N_\omega^k = \begin{cases} N_\omega Pr[k] + Pr[k]_\omega^k u_r^k & \text{if joint } k \text{ is revolute} \\ N_\omega Pr[k] & \text{if joint } k \text{ is prismatic} \end{cases} \quad \text{(7)} $$

$$ N_v^k = \begin{cases} N_v Pr[k] + Pr[k]_\omega^k u_r^k \times r^k & \text{if joint } k \text{ is revolute} \\ N_v Pr[k] + k_i v_r^k + u_r^k & \text{if joint } k \text{ is prismatic} \end{cases} \quad \text{(8)} $$

$$ N_\omega_r^k = \begin{cases} Pr[k]_\omega_r^k & \text{if } r = k \text{ and joint } k \text{ is revolute} \\ N_\omega_r Pr[k] & \text{if } r < k \\ 0 & \text{if } r > k \end{cases} \quad \text{(9)} $$

$$ N_v_r^k = \begin{cases} Pr[k]_\omega_r^k \times r^k & \text{if } r = k \text{ and joint } k \text{ is revolute} \\ Pr[k]_\omega_r^k & \text{if } r = k \text{ and joint } k \text{ is prismatic} \\ N_v_r Pr[k] + N_\omega_r Pr[k] \times \gamma^k & \text{if } r < k \\ 0 & \text{if } r > k \end{cases} \quad \text{(10)} $$

In most $O(n)$ formulations, a key goal is to segregate all terms which are explicit in the unknown generalized accelerations $\ddot{u}$ as equations are being formed. To aid in this the angular acceleration
of a body \( k \) or the acceleration of a point, e.g., center of mass of \( k \), may be expressed as 
\[
N \ddot{\alpha}^k = N \ddot{\alpha} + N \dddot{\alpha}^k + N \omega^k \times \omega^k \times u^k_r 
\]
and 
\[
N \ddot{\alpha}_t^k = \frac{N \ddot{\alpha}_t + N \dddot{\alpha}_t^k}{2} + \left\{ \begin{array}{ll}
N \omega^k \times \omega^k \times \omega^k \times u^k_r & \text{if joint } k \text{ is revolute} \\
0 & \text{if joint } k \text{ is prismatic},
\end{array} \right.
\]
\[
N \ddot{\alpha}^k = N \ddot{\alpha} + N \dddot{\alpha}^k + N \omega^k \times \omega^k \times u^k_r + \frac{Pr[k]}{Pr[r]} \omega^k \times \omega^k \times \omega^k \times r^k, 
\]
\[
N \ddot{a}_t^k = N \ddot{a}_t + N \dddot{a}_t^k + N \omega^k \times \omega^k \times \omega^k \times u^k_r + \frac{Pr[k]}{Pr[r]} \omega^k \times \omega^k \times \omega^k \times u^k_r \times r^k 
\]
\[
+ \left\{ \begin{array}{ll}
2 N \omega^k \times \omega^k \times u^k_r & \text{if joint } k \text{ is revolute} \\
0 & \text{if joint } k \text{ is prismatic},
\end{array} \right.
\]

For discussing the differentiation operation, it is more convenient to express quantities on the basis of scalar matrix representations. If one defines the scalar matrices 
\[
N \mathcal{P}^k, \text{ partial velocities or free mode of motion} \] [19]. 
\[
N \mathcal{M}_1^k, \text{ the local central inertia matrix for body } k; \]
\[
N \mathcal{A}_1^k, \text{ generalized acceleration remainder term, as}
\]
\[
N \mathcal{P}^k = \begin{bmatrix}
N \omega^k_r \\
N \omega^k_r \\
\end{bmatrix} 
\]
\[
N \mathcal{M}_1^k = \begin{bmatrix}
N \mathcal{T}^{k/*} + N \mathcal{R}^k \\
\end{bmatrix},
\]
\[
N \mathcal{A}_1^k = \begin{bmatrix}
N \alpha^k_r \\
N \alpha^k_r \\
\end{bmatrix},
\]

where 
\[
N \mathcal{T}^{k/*} \text{ and } N \mathcal{R}^{k/*} \text{ are expressed as}
\]
\[
N \mathcal{T}^{k/*} = -M^k N \alpha^k_r, 
\]
\[
N \mathcal{R}^{k/*} = -I^k/k^* N \alpha^k_r - N \omega \times k^* I^k/k^* N \omega^k, 
\]

then one simply obtains
\[
\frac{d}{dp_j} N \mathcal{P}^k = \begin{bmatrix}
\frac{d}{dp_j} N \omega^k_r \\
\frac{d}{dp_j} N \omega^k_r \\
\end{bmatrix}^T, 
\]
\[
\frac{d}{dp_j} N \mathcal{A}_1^k = \begin{bmatrix}
\frac{d}{dp_j} N \alpha^k_r \\
\frac{d}{dp_j} N \alpha^k_r \\
\end{bmatrix}^T, 
\]
\[
\frac{d}{dp_j} N \mathcal{T}^k = \begin{bmatrix}
\frac{d}{dp_j} N \mathcal{T}^{k/*} \\
0 \\
\end{bmatrix},
\]

5
\[
\frac{d}{dp_j} N \mathcal{F}_1^k = \left[ \frac{d}{dp_j} \left( N \mathcal{T}_t^k + N \mathcal{R}_t^k \right) \right].
\]  

(24)

When the recursive relationships are enforced, Eqs. (21), (22), and (24) can further be written as

\[
\frac{d}{dp_j} N \mathcal{D}_t^{Pr[k]} = \frac{d}{dp_j} \left( \left( Pr[k] \mathcal{S}_k^k \right)^T N \mathcal{D}_t^{Pr[k]} + \left( Pr[k] \mathcal{S}_k^k \right)^T \frac{d}{dp_j} N \mathcal{D}_t^{Pr[k]} \right),
\]

(25)

\[
\frac{d}{dp_j} N \mathcal{A}_t^k = \frac{d}{dp_j} \left( \left( Pr[k] \mathcal{S}_k^k \right)^T N \mathcal{A}_t^k + \left( Pr[k] \mathcal{S}_k^k \right)^T \frac{d}{dp_j} N \mathcal{A}_t^k \right) + \frac{d}{dp_j} Pr[k] \mathcal{A}_t^k,
\]

(26)

\[
\frac{d}{dp_j} N \mathcal{F}_1^{Pr[k]} = \frac{d}{dp_j} \left( \left( Pr[k] \mathcal{S}_k^k \right)^T N \mathcal{F}_1^{Pr[k]} + \left( Pr[k] \mathcal{S}_k^k \right)^T \frac{d}{dp_j} N \mathcal{F}_1^{Pr[k]} \right),
\]

(27)

with \( Pr[k] \mathcal{A}_t^k \) and basis consistent shifting matrix \( Pr[k] \mathcal{S}_k^k \) defined as

\[
Pr[k] \mathcal{A}_t^k \triangleq \left[ \begin{array}{c}
N \omega^{Pr[k]} \times Pr[k] \omega_k^k u_r^k \\
N \omega^{Pr[k]} \times \left( N \omega^{Pr[k]} \times \gamma^k \right) + Pr[k] \omega_k^k u_r^k \times \left( Pr[k] \omega_k^k u_r^k \times \mathbf{r}^k \right) \\
+ 2N \omega^{Pr[k]} \times \gamma^k \times \mathbf{v}_k^k + u_r^k
\end{array} \right],
\]

(28)

and

\[
Pr[k] \mathcal{S}_k^k \triangleq \left[ \begin{array}{c}
\mathbf{U} \\
\gamma^k \times \mathbf{U}
\end{array} \right]_{6 \times 6}.
\]

(29)

In above equations, \( \mathbf{U} \) is simply an identity matrix, \( \gamma^k \) defines the spatial vector from body \( Pr[k] \)'s center of mass to body \( k \)'s center of mass, and \( \gamma^k \times \) is a skew matrix, which is the matrix representation of the vector cross product operator. \( Pr[k] \mathcal{S}_k^k \) is the composite block diagonal direction cosine matrix, relating the \( k \) to the \( Pr[k] \) basis.

Equations (23), (25)-(27) provide a compact form to recursively derive the total derivatives of essential kinematic quantities from the base body to terminal bodies. Based on the definition of relative coordinate and the fixed dimension of each matrix, computing these equations can be performed in a fixed cost which ideally is an \( O(n^0) \) operation for each body. Therefore, the above derivations represent an \( O(n) \) procedure involving from a base-to-tip recursion sequence to compute the velocities, accelerations, and active forces with their associate total derivatives.

### 2.3 Recursive Inward Procedure

When the procedure reaches the terminal bodies, the topological direction in which the calculation proceed is effectively reversed and the procedure works to produce the total derivatives of articulated body inertias [11], generalized active forces, and triangularization the system equations. At terminal body \( k = b \), Kane’s dynamics formulation yields

\[
F_b + F_b^* = \left( \mathcal{P}_b^b \right)^T \left[ \mathcal{L}_b^b \left( b \mathcal{S}^{Pr[b]} \right)^T \mathcal{A}^{Pr[b]} - \mathcal{F}_b^b \right] + \mathcal{M}_{bb} \ddot{u}_b = 0,
\]

(31)
with
\[
\mathcal{A}^{Pr[b]} = \begin{bmatrix}
N \mathcal{A}^{Pr[b]} \\
N \mathcal{A}^{Pr[b]} \\
\end{bmatrix}_{6 \times 1},
\]
(32)
which contains all terms that are explicit in \( \dot{u}'s \). Proceeding inward to body \( Pr[b] \), one has
\[
F_{Pr[b]} + F_{Pr*b} = 0
\]
\[
= (Pr_{Pr[b]})^T \begin{bmatrix}
\mathcal{I}_{1}^{Pr[b]} \mathcal{A}^{Pr[b]} - \mathcal{F}_{1}^{Pr[b]} \\
\end{bmatrix} + (Pr_{Pr[b]})^T \begin{bmatrix}
\mathcal{I}_{1}^{b} \mathcal{A}^{b} - \mathcal{F}_{1}^{b} \\
\end{bmatrix}
\]
\[
= (Pr_{Pr[b]})^T \begin{bmatrix}
\mathcal{I}_{1}^{Pr[b]} \mathcal{A}^{Pr[b]} - \mathcal{F}_{1}^{Pr[b]} \\
\end{bmatrix}
\]
\[
+ (Pr_{Pr[b]})^T Pr_{[b]} S_{b} \left( \mathcal{I}_{1}^{b} \begin{bmatrix}
\mathcal{I}_{1}^{Pr[b]} S_{b} \mathcal{A}^{Pr[b]} + Pr_{b} \mathcal{A}_{b} \\
\end{bmatrix} - \mathcal{F}_{3}^{b} \right)
\]
\[
= (Pr_{Pr[b]})^T \begin{bmatrix}
\mathcal{I}_{3}^{Pr[b]} \mathcal{A}^{Pr[b]} - \mathcal{F}_{3}^{Pr[b]} \\
\end{bmatrix},
\]
(33)
or for a general body \( k \)
\[
F_{k} + F_{k}^{*} = \left( Pr_{k}^T \mathcal{I}_{k}^{k} \mathcal{A}^{k} - \mathcal{F}_{k}^{k} \right) = 0,
\]
(34)
where the matrices \( \mathcal{I}_{3}^{k} \) and \( \mathcal{F}_{3}^{k} \) are defined as
\[
\mathcal{I}_{3}^{k} = \mathcal{I}_{1}^{k} + \sum_{b \in Dist[k]} \left( k \mathcal{S}_{b} \mathcal{I}_{3}^{b} \left( k \mathcal{S}_{b} \right)^T - \frac{1}{M_{bb}} \left[ k \mathcal{S}_{b} \mathcal{I}_{3}^{b} Pr_{b} \left( k \mathcal{S}_{b} \mathcal{I}_{3}^{b} Pr_{b}^T \right) \right] \right),
\]
(35)
\[
\mathcal{F}_{3}^{k} = \mathcal{F}_{1}^{b} + \sum_{b \in Dist[k]} \left( k \mathcal{S}_{b} \mathcal{F}_{3}^{b} - \frac{1}{M_{bb}} \left[ k \mathcal{S}_{b} \mathcal{I}_{3}^{b} Pr_{b} \left( Pr_{b}^T \mathcal{F}_{3}^{b} \right) \right] \right),
\]
(36)
and \( M_{bb} \) is given as
\[
M_{bb} = \left( Pr_{b}^T \right) \mathcal{I}_{3}^{b} Pr_{b},
\]
(37)
In Eq. (36) and (37), the notation \( Dist[k] \) is called the distal body set which contains all outboard bodies that are immediately connected to the body \( k \). Due to the summation appeared in Eqs. (35) and (36), the computational requirements for evaluating matrices \( \mathcal{I}_{3}^{k} \) and \( \mathcal{I}_{3}^{k} \) may vary, but are bounded by the total number of distal bodies which are adjacent and outboard to body \( k \). If for the sake of this example one restricts body \( Pr[k] \) to have only a single distal body, (i.e., \( Dist[Pr[n]] = \{ n \} \)), then the summation in both equations contains only one term. For a general open-loop system, above recursive triangularization process is still able to perform an \( O(1) \) operation for each body. That is, matrices \( \mathcal{I}_{3}^{k} \) and \( \mathcal{F}_{3}^{k} \) are needed to be computed only once for each body, regardless the total number of distal bodies appeared.

The determination of the total derivatives of all kinematic quantities in the previous section allows the procedures of recursive triangularization to be carried out to compute the total derivatives of \( \mathcal{I}_{3}^{k}, \mathcal{F}_{3}^{k}, \) and \( M_{kk} \). Using the generalized expressions of \( \mathcal{I}_{3}^{k}, \mathcal{F}_{3}^{k}, \) and \( M_{kk} \) described in Eqs. (35)-(37) gives their total derivatives. Namely
\[
\frac{d}{dp_{j}} \mathcal{I}_{3}^{k} = \frac{d}{dp_{j}} \mathcal{I}_{1}^{k} + \sum_{b \in Dist[k]} \frac{d}{dp_{j}} \left( k \mathcal{S}_{b} \mathcal{I}_{3}^{b} \left( k \mathcal{S}_{b} \right)^T \right)
\]
\[
- \frac{1}{\mathcal{M}_{bb}} \left[ k^2 \tilde{S}_b^b \mathcal{I}_3^b \mathcal{P}_b^b \left( k^2 \tilde{S}_b^b \mathcal{I}_3^b \mathcal{P}_b^b \right)^T \right] , 
\]
(38)
\[
\frac{d}{dp_j} \mathcal{F}_3^k = \frac{d}{dp_j} \mathcal{F}_1^k + \sum_{b \in \text{Dist}[k]} \frac{d}{dp_j} \left( k^2 \tilde{S}_b^b \mathcal{F}_3^k \right) 
- \frac{1}{\mathcal{M}_{bb}} \left[ k^2 \tilde{S}_b^b \mathcal{I}_3^b \mathcal{P}_b^b \left( \mathcal{P}_b^b \right)^T \mathcal{F}_3^b \right] , 
\]
(39)
\[
\frac{d}{dp_j} \mathcal{M}_{bb} = \frac{d}{dp_j} \left[ \left( \mathcal{P}_b^b \right)^T \mathcal{I}_3^b \mathcal{P}_b^b \right] . 
\]
(40)

Note that computing Eqs. (38)-(40) does not require additional differentiation operations. They can be fully determined through matrix manipulations of known quantities acquired from the forward dynamic analysis procedures and the total derivatives computed in the last section.

2.4 Back-Substitution

The solution scheme of the back-substitution is to recursively solve for generalized acceleration once the equations of motion are effectively put in a lower triangular form in the triangularization procedure. At the base body, information associated with an entire set of outboard bodies have all been accumulated and explicitly available such that the equation \( \mathcal{M}_{11} \dot{\mathbf{u}}_1 = \mathcal{K}_1 \) can be isolated and yields the solution of \( \dot{\mathbf{u}}_1 \). A generalized function expression for the solution of each generalized acceleration \( \dot{\mathbf{u}}_k \) in this triangularization process is given as follows

\[
\dot{\mathbf{u}}_k = \left( \mathcal{P}_k^k \right)^T \left[ \mathcal{F}_3^k - \mathcal{I}_3^k \left( k^{-1} \tilde{S}_k^k \right)^T \mathcal{A}^{-1} \right] . 
\]
(41)

Equation (41) clearly indicates that solving for each of \( \dot{\mathbf{u}}_i \) (\( i = 1, \ldots, n \)) can be performed at a fixed cost due to the recursive manipulations. Such a solution scheme does not require forming the entire system matrices \( \mathcal{M} \) and \( \mathcal{K} \) explicitly. Instead, each element is implicitly manipulated in parallel with solving system equations which makes it possible to eliminate many operations. This recursive solution scheme can be applied to the sensitivity analysis equally well. Specifically, taking the total derivative of Eq. (41) yields the essential state derivative values.

At the base body, the derivative of \( \dot{\mathbf{u}}_1 \) with respect to \( p_j \) is expressed as

\[
\frac{d}{dp_j} \frac{d}{dp_j} = \frac{d}{dp_j} \left[ \left( \mathcal{P}_1^1 \right)^T \mathcal{F}_1^1 \right] , 
\]
(42)
from which the generalized acceleration matrix of the base body can be determined as

\[
\frac{d}{dp_j} \mathcal{A}^{-1} = \left[ \frac{d}{dp_j} \mathcal{P}_1^1 \right] \dot{\mathbf{u}}_1 + \mathcal{P}_1^1 \left[ \frac{d}{dp_j} \dot{\mathbf{u}}_1 \right] . 
\]
(43)

Properly substituting the result obtained from Eq. (43) into the differentiated form of Eq. (42) yields the desired \( d\dot{\mathbf{u}}_2 / dp_j \). Therefore, a generalized procedure for the back-substitution is carried out in an orderly manner following two equations

\[
\frac{d}{dp_j} \dot{\mathbf{u}}_k = \frac{d}{dp_j} \left[ \left( \mathcal{P}_k^k \right)^T \left[ \mathcal{F}_3^k - \mathcal{I}_3^k \left( k^{-1} \tilde{S}_k^k \right)^T \mathcal{A}^{-1} \right] \right] , 
\]
(44)
\[
\frac{d}{dp_j} \tilde{A}^k = \frac{d}{dp_j} \left[ \left( k^{-1 \rightarrow k} \hat{S}^k \right)^T \tilde{A}^{k-1} + \mathcal{P}^k \hat{u}_k \right].
\] 

(45)

To this end, the key quantities \( \frac{d \hat{u}_k}{dp_j} \) in the first-order sensitivity analysis are determined in a fully recursive manner. Consequently, the triangularization and back-substitution processes yields an \( O(n) \) operation overall to solve for all \( \frac{d \hat{u}_k}{dp_j} \) \( (k = 1, \ldots, n) \) for one design variable. For an entire \( p \) design variables, the proposed solution scheme is able to achieve an \( O(pn) \) operation overall for obtaining all solutions associated with the \( p \) sets of sensitivity coefficient equations.

3 Computational Structure of New Algorithm

3.1 Derivative Primitives

The \( O(n) \) first-order sensitivity analysis derived previously indicates that its overall computational structure is in many regards analogous to its dynamic forward problem analysis counterpart. Specifically, the computational structure of this specialized method is divided into three major steps:

1. working from the base body outward toward the terminal bodies, recursively generate the total derivatives of kinematic quantities;

2. working recursively inward from the terminal bodies to the base body, yield the total derivatives of articulated body inertias, generalized active forces, and triangularize the resulting equations;

3. working outward from base body to the terminal bodies, recursively back substitute to obtain the desired total derivatives of relative joint acceleration values.

These procedures are indeed highly similar to those in the recursive dynamic forward problem analysis as illustrated in Fig. 1. Due to their similarities, a mixed type of analysis routine can be developed that allows both dynamic forward problem analysis and sensitivity analysis to be performed simultaneously. Many needed kinematic as well as kinetic information used in the presented sensitivity algorithm can be extracted directly from the known quantities computed in the dynamic forward problem analysis. Implementing both analysis in such a mixed type fashion not only significantly reduces the numbers of intermediate quantities used in the sensitivity analysis routines, but also induces a concurrent procedure to yield simultaneous solutions of state derivatives and state sensitivities. Both circumstances contribute to the improvement of computational efficiency.

The primary task in the presented algorithm is to efficiently compute various total derivatives associated with kinematic quantities. In the current development, these operations are required only in the first outward recursive procedure because forming total derivatives of articulated body inertias and generalized inertia forces can be achieved through the proper utilization of known differentiated quantities. As has been seen in Fig. 1, there are 8 matrices which need to be differentiated for each body which can in general be characterized as the properties of generalized speeds, lengths, masses, orientation angles, and force elements. Since design variables are the quantities that specify local properties of particular elements, the derivatives of these matrices with respect to a specific design variable lead to many null matrices. Combining the intelligent usage of relative state space and local representations to describe these matrices enables a great simplification on the entire differentiation procedure. The outcome is the additional computational saving which is especially beneficial for processing a high order of design variables when the number of system
degrees of freedom is large \((n \gg 1)\). Due to the significance of these fundamental derivatives in current sensitivity algorithm, they are termed \textit{derivative primitives}.

For a general tree system, the computational requirement to yield these derivative primitives in the presented method is significantly less than those needed in the \(O(n^3)\) direct differentiation approach. As mentioned earlier, a more traditional direct differentiation approach requires an \(O(n^4)\) operation to evaluate all essential derivatives in matrices \(A\) and \(K\). The empirical results illustrated in Fig. 2, which is obtained from the commercially available symbolic dynamic simulation and analysis software AUTOLEV [18], clearly demonstrate that the current development results in an extremely efficient performance to obtain values of derivative primitives. The empirical CPU time required for the current method is seen to be linear in\( n\), whereas the traditional direct \(O(n^3)\) implementation is of quartic order. Note the dramatic ratio between both approaches. The MAPLE implementation of these derivative operations also validates the required computational orders for both cases.

### 3.2 Theoretical Operation Count

Theoretical derivation of the new algorithm as has been shown in the previous sections provides qualitative information of its operational order, and the efficiency comparison between current approach with a direct \(O(n^3)\) approach is made based on their underlying formulation methodologies. To validate these arguments and for practical implementation purposes, it is highly desirable to give a more precise examination of its operation count and computational performance. However, different system topology, joint types, and design variable may contribute to very different overall operation count. In order to circumventing such situations, a chain system with uniform body structure is used as a representative case for determining the actual operational order. In this instance, a special type of design variable, length, is chosen and assumed that it forms various properties of each body. This particular setup allows a generalized formulation to be produced so counting the required operations of each body can be performed systematically. The selection of length as the main design variable is primarily based on the fact that, due to the change on length, the computation of kinematic (e.g., linkage length, angles, velocities, etc.) as well as kinetic (e.g., mass, inertia tensor, C.M. location, etc.) and each of their derivative is considered more involved than other types of design variables. Although this representative case may not resemble a realistic dynamic system and its associated operation order, it does provide advantages to reveal the information of the \textit{maximal} computational requirement of current development.

Table 1 gives required theoretical operations of each recursive step for the \(O(n)\) sensitivity algorithm as well as the \(O(n)\) dynamic algorithm. The total operation cost given for the sensitivity analysis reflects those costs beyond what was necessary to determine necessary quantities from the dynamic forward problem analysis. The comparison of operations for both cases indicates that a notable feature of the current method is it maintains nearly a constant ratio on computation relative to the \(O(n)\) dynamic forward problem. This ratio can be roughly estimated by

\[
\frac{O(n) \text{ sensitivity analysis}}{O(n) \text{ forward problem analysis}} \approx \begin{cases} 
 2.138 & \text{for } n = 2 \\
 2.318 & \text{for } n \gg 1
\end{cases}.
\]  

\[(46)\]

The assumptions of approximately the same computational cost for multiplication \(m\) and addition \(a\), and the differentiation \(d\) is approximated by \(m + a\), are made in calculating Eq. (46). It is this nearly constant ratio that provides a competitive edge of this new algorithm on optimization process when applied to many complex systems. Firstly, the computational requirements of this algorithm is bilinear in the number of design variable, and number of degrees of freedom. Specifically, for a set
of p design variables, the total cost to evaluate first-order sensitivity information is \( \sim 2p \times O(n_d) \) where \( O(n_d) \) is the cost associated with underlying dynamic analysis. Secondly, this new algorithm will be more efficient than the finite difference approach using an \( O(n^3) \) underlying formulation for the cases of large \( n \) and/or \( p \). For a finite difference approach utilizing any of the formulations method, one expects a factor of \( p + 1 \) times the cost of the forward problem to estimate the sensitivity information. Although this ratio seems to be smaller than the methodology presented, if considering the shortcomings of accuracy and stability exhibited by finite difference approach, the current method becomes a strong candidate given its “exact” analytic solution. In general, the ratio computed in Eq. (46) will be greatly reduced when applied to more realistic physical systems.

3.3 Empirical Simulation Performance

Quantitatively examining the efficiency of this specialized algorithm by means of analytical operation count has provided a certain insight to its computational advantages over the \( O(n^3) \) direct differentiation method. To further demonstrate the efficiency of the presented methodology on sequential computing machines, this section focuses on its fundamental performance compared with a traditional state space \( O(n^3) \) direct differentiation approach and the broadly used finite difference approach.

Among these numerical comparison, the only difference in the simulation/analysis codes is in the underlying formulation used such that their raw performances can be isolated from other I/O operations leading to an accurate comparison. Figure 3 plots the chain system simulation timing results for all cases and each of their associated least square polynomial function fit is tabulated in Table 2 so to empirically identify the order of different formulation. The computational advantages of \( O(n) \) analytic algorithm over \( O(n^3) \) analytic algorithm and finite difference on sensitivity analysis can be approximated from following calculations

\[
\frac{O(n^3) \text{ analytic sensitivity}}{O(n) \text{ analytic sensitivity}} \approx 0.0619n^3 - 2.0719n^2 + 24.1198n - 95.5904 . \tag{47}
\]

\[
\frac{O(n^3) \text{ finite difference}}{O(n) \text{ analytic sensitivity}} \approx 0.0225n^2 - 0.4053n + 2.7147 . \tag{48}
\]

\[
\frac{O(n) \text{ finite difference}}{O(n) \text{ analytic sensitivity}} \approx 0.4972 , \tag{49}
\]

When \( n \) is small (\( < 6 \)), the desired approaches to perform sensitivity analysis may belong to the group of \( O(n^3) \) methodologies. Due to the small system degrees of freedom in this instance, the differences in computational loads may not be sufficiently large to differentiate the advantages of using an \( O(n^3) \) approach from other \( O(n) \) methodologies. This may also be seen from Eqs. (47)-(48) in which the highest order of \( n^3 \) and \( n^2 \) in the respective polynomial function is not a dominant factor for small \( n \). However, as \( n \) becoming large, the approaches based upon the \( O(n) \) formulation are significantly faster than the \( O(n^3) \) approaches on performing the first-order sensitivity analysis because of their performance rates being dependent linearly in \( n \) as opposed to a cubic or quartic order in the \( O(n^3) \) approaches. Consequently, one can conclude that these \( O(n) \) algorithms for generating the first-order sensitivities are extremely efficient for the representative problem simulated.

Comparing the presented \( O(n) \) based analytical sensitivity method with the \( O(n) \) finite difference showed that the former approach seems to be slower than the latter. The difference between these approaches arises from the fact that, in this extreme case (change in geometric and inertia properties of every body), the analytical sensitivity calculation is more costly than the associated
forward problems. Although this empirical result is consistent with aforementioned theoretical operation count, one should keep in mind that the assumption made previously is the main cause for the performance degradation of the presented method. Indeed in situations where the change of a single design parameter does not so pervasively affect so many terms in the sensitivity calculation (e.g., simple changes in stiffness, damping, mass) the analytical sensitivity can be shown to be less costly than that generated via finite difference.

4 Numerical Example

To further illustrate the practical performance of the presented method and to demonstrate method’s validity, a vehicle suspension system, which is similar to the example presented in reference [13], is provided as a numerical example. The model shown in Fig. 4 possesses 7 degrees of freedom and forms a tree system. The motion of the system is characterized by the generalized coordinates \( q_i \) \((i = 1, \ldots, 7)\) and is restricted to the planar motion. The design variables involve all types of system properties and constitutes by the force elements, \( k_1, k_2, k_3, c_1, c_2, \) and \( c_3, \) mass, \( m_4, \) and the length, \( L. \) These design variables have to be selected such that the acceleration of \( m_1 \) is minimized. This particular design problem can be formulated as a nonlinear programming problem to which many gradient-base optimization techniques are applicable [13]. However, rather than formally performing a design optimization, the objective of this numerical example is to provide more insight of solution accuracy and computational speed of the methodology developed.

Some numerical data and system properties associated with this model are listed as follows: \( m_{1g} = 290 \text{ lb}, \ m_{2g} = 4, \ 500 \text{ lb}, \ m_{4g} = m_{7g} = 48.3 \text{ lb}, \ I = 41,000 \text{ lb-in-sec}^2, \ k_1 = k_5 = 1,500 \text{ lb/in}, \) and \( c_4 = c_5 = 20 \text{ lb-sec/in}. \) The initial data of the design variables are also given: \( m_{4g} = m_{5g} = 96.6 \text{ lb}, \ L = 120 \text{ in}, \ k_2 = k_3 = 1,000 \text{ lb/in}, \) and \( c_2 = c_3 = 50 \text{ lb-sec/in}. \)

4.1 Solution Validity

The first-order sensitivity information associated with each variable is calculated by means of direct differentiation approach, Eq. (4), an \( O(n) \) finite difference approximation, and finally the presented recursive \( O(n) \) analytical sensitivity procedure. The state sensitivities obtained from solving Eq. (4) are taken as analytical exact solutions. For representative purpose, Fig. 5 depicts the computed sensitivity of \( d\dot{u}_1/dL \) by these methods. It is clear that a high degree of solution agreement between the proposed method and the other methodologies. For other design variables, degrees of solution agreements are almost identical to the case presented.

The design variable perturbation employed in the finite difference approach should be chosen carefully due to approximation errors. Figure 6 illustrates that parameter perturbations \( \delta P \) has considerable influence on the computed sensitivity errors in finite difference implementations. The mean absolute errors between the analytical solutions and finite difference plotted in Fig. 6 are functions of relative parameter perturbation \( \delta p \). Small perturbation may lead to error amplification while large perturbation results in approximation errors [5]. Complicating matters, the optimal \( \delta p \) for which the errors are minimal may differ from one design variable to another. There is a narrow range of optimal perturbation associated with each design variable, but the range is usually not known a priori. For a large scale design problem, searching for optimal parameter perturbations to reduce approximation errors requires error screening, a computational cumbersome and expensive procedure. For these reasons, the simplicity of the finite difference approach may not necessarily imply a cheap computation [6]. The recursive \( O(n) \) base analytical sensitivity algorithm, because it is “exact”, is not subject to such sensitivities. The solution accuracy is up to machine precision and is independent from the magnitude of \( \delta p \), a similar result that has also been demonstrated in
the case of adjoint variable methods [4]. Such independency is observed throughout all ranges of \( \delta p \) simulated.

Another difficulty that may arise is associated with determining the optimal parameter perturbation in the finite difference approach is due to the fact that the optimal perturbation range is also state variable dependent. The optimal perturbation found for one state variable is not generally globally applicable for all other state variables, or even the same variable over an extended time. For instance, as indicated in Fig. 6, identical parameter perturbation of \( k_2 \) yields different error magnitude on \( d\hat{u}_1/dk_2 \) and \( d\hat{u}_3/dk_2 \). The optimal perturbation range for \( d\hat{u}_3/dk_2 \) differs significantly from that in the case of \( d\hat{u}_1/dk_2 \). For a large scale dynamic system involving a great number of \( n \), it becomes even more difficult to keep track of each individual optimal perturbation range of all state variables. Therefore, the general approach of using a fixed magnitude of parameter perturbation may lead to error accumulation on computing state sensitivities. By comparison, the presented method is not subject to such tendencies and maintains a very high solution accuracy. Insensitive to the parameter perturbation makes the present algorithm a superior method than the finite difference approach and greatly reduces the efforts on error monitoring.

### 4.2 Computational Performance

The theoretical worst case operation count aforementioned has indicated that a factor \( \sim 2 \) may be necessary for the presented \( O(n) \) analytical method to perform sensitivity analysis with respect to the \( O(n) \) dynamic forward analysis. However, this worst case is based upon the assumption that the design variable length affects both geometric and mass properties of each body. For a more realistic problem presented herein, one expects the reduction of this factor since design variables are independent and local variables. Thus, the change on one design variable only affects the properties of a particular body, but not all bodies as previously assumed. As a result, one observes the following facts: (i) the total differentiation operations are reduced, and (ii) numbers of null matrices and the sparseness of other matrices are increased. Both situations eliminate many operations and lead to an increase on computational performance of the proposed method. Consequently, the simulation results of the various techniques present here better reveal their actual performance characteristics for realistic problem than what has been demonstrated previously when they are applied to practical sensitivity analysis.

The state derivatives of the vehicle suspension model has to be obtained for a set of design variables \([k_1, k_2, k_3, c_1, c_2, c_3, m_4, L]\). The comparison of computational speed is made for four cases: \( O(n^3) \) direct differentiation approach, \( O(n^3) \) finite difference, \( O(n) \) finite difference, and the presented method. In general, the overall simulation cost can be heavily influenced by the temporal integration scheme used (i.e., its accuracy, step size, stability, etc.). In an effort to remove this performance dependency on integration characteristics, all simulations were performed using the same temporal integration software, with the same prescribed fixed step size. The simulation implemented based on this basis leads to an accurate result of computational speed/cost per function evaluation for all approaches. The obtained simulation results are normalized by the greatest value and tabulated in Table 3. Figure 7 provides the accumulated simulation time to obtain the state derivative of a complete set design variable for each case.

From the listed results, the fully recursive \( O(n) \) base analytic sensitivity algorithm is the most efficient approach among all methodologies. Even for this simple system considered, the presented method is able to achieve an 85% reduction on computational efforts compared with the \( O(n^3) \) direct approach, and is about 5 times faster than the \( O(n^3) \) finite difference. Comparing the method proposed with the \( O(n) \) finite difference approach also demonstrates the performance superiority of the former case than the latter case in most of the design variable evaluations, and as a result,
obtaining an overall of 6\% computational speedup. Such a superior performance of the $O(n)$ base analytic sensitivity method implies that the theoretical factor of 2 may not necessarily transfer to practical implementation. Therefore, for a large scale design problem with a set of $p$ design variables, the proposed method is able to yield first-order analytical sensitivity information in an extremely efficient $O(pn)$ operation overall.

5 Conclusions

A fully recursive sensitivity analysis method is presented for the determination of first-order sensitivity information of general dynamic tree-systems. The intelligent use of recursive relationships significantly eliminates many matrix manipulation and derivative operations exhibited in traditional direct differentiation approaches. The provided numerical example demonstrates the solution accuracy, method validity, and procedure stability of the method. Qualitative and quantitative examination of method’s computational performance provide solid evidence of its usefulness and efficiency. The method is able to achieve the same computational order as in the case of $O(n)$ finite difference approach allowing it to be a competitive alternative. Computational performance offered by this method has offered many competitive advantages and established a foundation for it to become a viable sensitivity analysis tool in practical design optimization problems.

Acknowledgments

Support for this work received under National Science Foundation through award No. 9733684 is gratefully acknowledged.

References


Figure 1: Computational Structure of $O(n)$ Dynamic and Sensitivity Algorithms
Figure 2: CPU Time Required for Derivative Operation by $O(n)$ and $O(n^3)$ Methodologies.
Figure 3: Computational Performance of Various Techniques
Figure 4: A Schematic of 7 DOF Vehicle Suspension Model
Figure 5: Comparing $\frac{d\theta}{d\ell}$ Computed by $O(n)$ Algorithm to the Exact Solution
Figure 6: Dependency Between Solution Accuracy and Parameter Perturbations
Figure 7: Comparison of Accumulated Simulation Time for Various Approaches
Table 1: Operation Count for the $O(n)$ Forward Dynamics and Sensitivity Analysis

<table>
<thead>
<tr>
<th></th>
<th>Forward Problem Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outward Recursive</td>
<td>$(102m + 77a)(n - 2) + (115m + 77a)$</td>
</tr>
<tr>
<td>Inward Recursive</td>
<td>$(582m + 476a)(n - 2) + (414m + 291a)$</td>
</tr>
<tr>
<td>Back-Substitution</td>
<td>$(70m + 62a)(n - 2) + (65m + 52a)$</td>
</tr>
<tr>
<td>Total Operation Cost</td>
<td>$(754m + 615a)(n - 2) + (594m + 420a)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Sensitivity Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outward Recursive</td>
<td>$(217m + 165a + 23d)(n - 2) + (141m + 115a + 34d)$</td>
</tr>
<tr>
<td>Inward Recursive</td>
<td>$(1215m + 1079a)(n - 2) + (881m + 722a)$</td>
</tr>
<tr>
<td>Back-Substitution</td>
<td>$(140m + 132a)(n - 2) + (130m + 111a)$</td>
</tr>
<tr>
<td>Total Operation Cost</td>
<td>$(1572m + 1376a + 23d)(n - 2) + (1152m + 948a + 34d)$</td>
</tr>
</tbody>
</table>

$m$: multiplication       $a$: addition           $d$: differentiation          $n$: number of degrees of freedom
<table>
<thead>
<tr>
<th>Method</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n^3)$ direct differentiation</td>
<td>$0.0245n^4 - 0.8454n^3 + 10.4039n^2 - 47.8518n + 65.125$</td>
</tr>
<tr>
<td>$O(n^3)$ finite difference</td>
<td>$0.0089n^3 - 0.1697n^2 + 1.2426n - 2.1752$</td>
</tr>
<tr>
<td>$O(n)$ finite difference</td>
<td>$0.1967n - 0.1261$</td>
</tr>
<tr>
<td>$O(n)$ analytic full recursive</td>
<td>$0.3956n - 0.4161$</td>
</tr>
</tbody>
</table>
Table 3: Normalized Computing Speed on Producing State Sensitivities

<table>
<thead>
<tr>
<th></th>
<th>$O(n^3)$ DD</th>
<th>$O(n^3)$ FD</th>
<th>$O(n)$ FD</th>
<th>$O(n)$ FR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>0.997</td>
<td>0.428</td>
<td>0.085</td>
<td>0.077</td>
</tr>
<tr>
<td>$k_2$</td>
<td>0.986</td>
<td>0.415</td>
<td>0.085</td>
<td>0.083</td>
</tr>
<tr>
<td>$k_3$</td>
<td>0.986</td>
<td>0.405</td>
<td>0.089</td>
<td>0.073</td>
</tr>
<tr>
<td>$c_1$</td>
<td>0.983</td>
<td>0.405</td>
<td>0.083</td>
<td>0.067</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0.945</td>
<td>0.410</td>
<td>0.089</td>
<td>0.078</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0.990</td>
<td>0.419</td>
<td>0.085</td>
<td>0.076</td>
</tr>
<tr>
<td>$m_4$</td>
<td>0.998</td>
<td>0.410</td>
<td>0.083</td>
<td>0.091</td>
</tr>
<tr>
<td>$L$</td>
<td>1.000</td>
<td>0.410</td>
<td>0.083</td>
<td>0.093</td>
</tr>
</tbody>
</table>

DD: direct differentiation  FD: finite difference
FR: fully recursive

10 seconds simulation with 0.005 seconds fixed step size