205

SUPERCOMPUTER BASED SECOND ORDER DESIGN SENSITIVITY ANALYSIS OF CONSTRAINED DYNAMIC SYSTEMS

by

SITARAM RAMASWAMY

B.Tech., Indian Institute of Technology, Madras, India, 1978

P.G.D.I.E, National Institute for Training in Industrial Engineering, Bombay, India, 1980

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Mechanical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:

K. Prakash

Major Professor

TABLE OF CONTENTS

		Page
LIST OF FIGURES		iv
CHAPTER		
I.	INTRODUCTION	1
	1.1 Applications of Second Order Sensitivity	4
	1.2 Scope of the Present Work	5
II.	DYNAMIC ANALYSIS	8
	2.1 System Description and Kinetic Energy	10
	2.2 Equations of Constraint	12
	2.3 Generalized Forces	18
	2.4 System Equations of Motion and Constraint	22
III.	FIRST ORDER DESIGN SENSITIVITY ANALYSIS	25
	3.1 First Order Design Sensitivity Analysis by Direct Differentiation	28
IV.	SECOND ORDER DESIGN SENSITIVITY ANALYSIS	35
	4.1 Second Order Design Sensitivity Analysis by Direct Differentiation	35
	4.2 Comparison with the Adjoint Variable Method	41
V.	IMPLEMENTATION	47
	5.1 Symbolic Computing in Design Sensitivity Analysis	49
	5.2 The REDUCE Preprocessor	51
	5.3 Advantages of Symbolic Computing	55
	5.4 Solution of the System Equations of Motion	59
	5.5 Solution of State Sensitivity Equations	64
	5.6 Selection of Numerical Integration Algorithm	66
	5.7 Supercomputer Implementation	68
	5.8 Algorithm for Combined Dynamic Analysis and First/Second Order Design Sensitivity Analysis	74
VI.	NUMERICAL RESULTS	77
	6.1 Verification of Results	77
	6.2 Numerical Examples	82

VII. CONCLUSIONS 104

7.1 Recommendations for Future Research and Development . 105

LIST OF REFERENCES 106

APPENDIX I 108

APPENDIX II 111

APPENDIX III. 112

LIST OF FIGURES

Figure		Page
2.1	Description of a Rigid Body	11
2.2	Revolute Joint	14
2.3	Translational Joint	16
2.4	Conversion of Applied Forces to Generalized Forces . . .	19
2.5	Linear Spring-Damper-Actuator Combination	21
6.1	Example 1 : Double Slider Mechanism	85
6.2	State sensitivity check for Ex. 1 with 1% perturbation. .	86
6.3	State sensitivity check for Ex. 1 with 5% perturbation. .	87
6.4	Example 2 : Slider-crank Mechanism.	91
6.5	State sensitivity check for Ex. 2 with 1% perturbation. .	92
6.6	State sensitivity check for Ex. 2 with 10% perturbation .	93
6.7	Example 3 : Two Degree-of-freedom Vibration Absorber. . .	97
6.8	State sensitivity check for Ex. 3 with 1% perturbation. .	98
6.9	Example 4 : Peaucellier-Lipkin Straight Line Generator. .	102
6.10	State sensitivity check for Ex. 4 with 1% perturbation. .	103

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Prakash Krishnaswami for all the help, guidance, support and encouragement that he provided me during the entire Masters program.

I would also like to thank Prof. C. L. D. Huang and Prof. K. H. Carpenter for serving on my committee and for their valuable suggestions.

I am grateful to Prof. A. C. Cogley, Chairman of the Department of Mechanical Engineering and ex-chairman Prof. P. L. Miller, Jr., as well as the Department of Mechanical Engineering for the financial assistance received during my stay with the department.

I would also like to thank the Mechanical Engineering faculty and my fellow graduate students, not only for all the help I have received from them, but also for providing an atmosphere conducive to learning. The staff at the Mechanical Engineering Department office deserve thanks for having been helpful at all times despite their heavy work load.

Thanks are also due to the National Science Foundation for having provided computing time on the CRAY X-MP/48 supercomputer at the Pittsburg Supercomputing Center, under their Grant No. ECS-8515718.

Finally, I would like to thank my wife Suvasini for making my return to college possible and worthwhile. Her cheerful support, encouragement and understanding have been of invaluable help in my work.

CHAPTER I

INTRODUCTION

Computers have been among the prime-movers of change in all branches of engineering during the last three decades. Developments in the field of engineering design resulting directly from advances in computer hardware and software have been quite dramatic. While early computer programs merely automated straight forward design calculations, present day computers are capable of providing a sophisticated design environment with interactive computing, graphics, communication facilities, and a staggering variety of ready-to-use software packages.

Although the availability of computational aids and better hardware has considerably eased the task of producing a good design, in several applications it is fast becoming important to produce the best possible design, not just one that is good enough. In many cases, optimization methods are being used to do this. To use the techniques of optimal design, it is necessary to formulate the design problem in the form of a standard optimal design problem. This standard problem is usually stated in the following manner:

Suppose the design of the system under consideration is specified by a vector $\underline{b}$ of r design variables, i.e.,

$$\underline{b} = [b_1, b_2, \ldots, b_r]^T \tag{1.1}$$

Find that design $\underline{b}$ which minimizes a specified cost function $\Psi_0(\underline{b})$, subject to the constraints

$$\Psi_i(\underline{b}) = 0, \; i = 1, 2, \ldots, m \qquad (1.2)$$
$$\Psi_i(\underline{b}) \leq 0, \; i = m+1, \ldots, n$$

The optimal design process attempts to solve this standard problem by iterative improvement of the design vector until the solution meets a predefined convergence criterion. Each iteration of the solution process involves the following three distinct steps:

1. Analysis: The behavior of the system at the present design is analyzed in this step; the cost and constraint function values are calculated.

2. Design Sensitivity Analysis: The derivatives of the cost and constraint functions with respect to the design variables are evaluated in this step. Generally, this step is likely to be quite expensive in terms of the amount of computation required; as a result, only first derivatives are evaluated for most applications.

3. Design Update/Optimization: In this step, the cost and constraint function values along with their respective derivatives are supplied to an optimization algorithm which calculates the required change in design. Most of the commonly used optimization algorithms require and use only first derivative information. Depending on how the optimization algorithm is implemented, it either returns control of the optimization process to the designer or it calls for reanalysis and fresh derivative information at the improved design and repeats the iterative design improvement process till convergence is reached.

In order to carry out the entire optimal design process on a computer, it is evident that the software should be capable of performing all the three steps mentioned above. In the case of large scale constrained dynamic mechanical systems, several general purpose programs are available for analysing the system response [1,2,3]. General purpose optimization packages have also been available for many years. However, attempts to integrate all the three steps listed above in a general computation scheme have been quite recent [4,5]. These attempts were based on first order design sensitivity and first order optimization methods.

Most of the work done so far on design sensitivity analysis of constrained dynamic systems has rested with finding first order sensitivity, with the exception of References 6 and 7, both of which were attempts to evaluate second order design sensitivity of particular systems. A generalized computation scheme for obtaining second order design sensitivity of a broad class of systems is yet to be realized. This is possibly due to the fact that the process of design sensitivity analysis is computationally intensive. The computation effort rises exponentially with the order of the sensitivity evaluated [5], making the evaluation of higher order sensitivities very expensive or computationally infeasible. However, there are application areas where it is necessary to use second order design sensitivity information. Some of the classes of problems which require second order design sensitivity information in the solution process are discussed in the following section.

1.1 Applications of Second Order Sensitivity

Second order design sensitivity information can be used in one of the following four application areas:

i) Second Order Optimization: In the case of certain highly non-linear systems, optimization algorithms using only first order sensitivity may not work very well. The problems that could be encountered include slow convergence, oscillatory behavior, large number of function evaluations, etc. In such situations, a second order optimization method will prove to be more effective. Even for well-behaved systems, second order optimization usually converges faster. Second order methods can be expected to use lesser number of function evaluations, since they avoid the costly line searches which are characteristic of most first order optimization methods. Further, second order optimization methods are usually direct methods in which the required change in design is obtained as a solution to a set of linear equations with the Hessian as the coefficient matrix, making the optimization process much faster.

ii) Minimum Sensitivity Design: The cost of manufacturing any component of a mechanical system is determined to some extent by the dimensional tolerances that are specified for it. These tolerances, in turn are governed by the sensitivity of the system to manufacturing errors. It follows, therefore, that a system whose performance does not deteriorate even for a relatively large change in the dimensions of its components would be cheaper to produce. The search for such a design can be formulated as an optimal design problem for minimizing

the sensitivity of the system with respect to design. To solve this minimization problem using a gradient based method, we need the gradient of the sensitivity, which is a second order design sensitivity.

iii) Reliability Design: To design a system with reliability as a criterion, we need to find a design that cannot be dislodged from a region of safe operation into an unsafe region by relatively small changes in design parameters. An optimal design formulation of this requirement also leads to a sensitivity minimization problem. To solve this problem, we once again need second order design sensitivity information.

iv) Approximating system behavior: When both first and second order design sensitivity information are available, the behavior of the system can be predicted for relatively larger changes in design parameters without repeating the system analysis, than what would be possible if only first order sensitivity information were available. This fact can be used in doing the line searches required in most gradient based optimization procedures and also as an efficient means of answering certain 'what if' queries related to moderately large design changes.

1.2 Scope of the Present Work

The intent of the present work is to extend the design sensitivity analysis of constrained dynamic mechanical systems to the evaluation of second order design sensitivities and to implement the same in an integrated scheme of computation. The computational

intensity of the solution process and the fact that most of the quantities dealt with in design sensitivity analysis of constrained dynamic systems are naturally representable as vectors and matrices led to the idea of using a supercomputer to solve the problem. This is on account of the fact that in addition to being capable of handling heavy computation loads, supercomputers are particularly adept at vector oriented computations.

The first requirement of a scheme for doing design sensitivity analysis is that it should have the capability to perform dynamic analysis. Most computerized schemes for dynamic analysis of mechanical systems follow the methodology of Lagrangian dynamics. However, they differ in their implementation. The present work uses the constrained multi-element formulation as presented in Reference 3, since this approach is simple and easy to use for analyzing the class of dynamic mechanical systems under investigation. The method used for system description and dynamic analysis is discussed in Chapter 2.

Design sensitivity analysis of constrained dynamic mechanical systems has been performed by the method of adjoint variables [4,8] and by the technique of direct differentiation [5,9]. The adjoint variable method in general requires the solution of fewer differential equations as compared to the direct differentiation technique. However, the adjoint variable method requires a large number of intermediate input/output operations and suffers from an inability to provide positive error control. In view of the better error control it affords and its suitability for implementation on a supercomputer, the

direct differentiation technique has been chosen as the method for performing design sensitivity analysis in the present work. A fuller comparison of the two methods is provided in Chapter 4. The equations required for evaluating first and second order design sensitivities by the technique of direct differentiation are derived in Chapters 3 and 4 respectively.

The computational scheme devised in the present work makes extensive use of symbolic computing for enhancing the generality and flexibility of the implementation. The use of symbolic computing considerably reduces the amount of input required from the user and makes it possible to carry out the entire design sensitivity analysis without any user written subroutines. The advantages of using symbolic computing are discussed in Chapter 5. The technique for solving the system equations of motion and the equations for first and second order design sensitivity analysis as well as some specific details of the implementation on a supercomputer are also described in Chapter 5.

Finally, some representative numerical examples have been solved in Chapter 8. The validity of the computed second order sensitivities was verified through perturbation analysis.

The present implementation has convincingly demonstrated the feasibility of carrying out second order design sensitivity analysis of constrained dynamic systems. It is hoped that second order design sensitivity analysis will attract more investigators in the near future and that some of the areas for future research indicated in the concluding chapter will be pursued.

CHAPTER II

DYNAMIC ANALYSIS

In order to analyze the transient response of a dynamic mechanical system, it is necessary to formulate the equations of motion that govern its behavior. These equations, which are usually second order ordinary differential equations, can then be integrated to obtain the system response. To implement the solution process on a computer, it is necessary to device a scheme which is capable of formulating the equations of motion and carrying out the integration. The method should also be capable of solving a reasonably wide range of problems.

The equations of motion for any system of interconnected rigid bodies can be obtained from Newtonian mechanics or from Lagrangian mechanics. Most of the existing methods for computer-aided dynamic analysis are based on the Lagrangian formulation. This is due to the fact that in Newtonian mechanics, the equations are vectorial in nature and forces and accelerations are often difficult to determine [10]. Furthermore, each problem seems to require its own insights and there are no general procedures for obtaining the differential equations of motion; which makes a computer implementation more difficult. The Lagrangian approach on the other

hand is algebraic in nature and can be implemented more easily on a computer.

The Lagrangian formulation for a general constrained dynamic system with holonomic, workless constraints can be stated in the following manner.

Consider a system whose state is completely described by a vector of generalized coordinates, $\underline{q}$, and a vector of corresponding generalized velocities, $\dot{\underline{q}}$. Let the constraints which may depend on the state of the system and the design variables be denoted by the column vector

$$\Phi(\underline{q}, \underline{b}; t) = 0 \tag{2.1}$$

where $\underline{b}$ is the design vector defined in Equation 1.1.

If all these constraints are assumed to be workless, then the Lagrangian equations of motion for the system are given by

$$\frac{d}{dt}[T_{\dot{\underline{q}}}]^T - [T_{\underline{q}}]^T - [\Phi_{\underline{q}}]^T \underline{\lambda} - \underline{Q} = 0 \tag{2.2}$$

where

T is the total kinetic energy of the system,

$\underline{Q}$ is the vector of generalized forces acting on the system,

$\underline{\lambda}$ is the vector of Lagrange multipliers associated with the system constraints

and all subscripts denote partial differentiation with respect to the subscript.

In order to obtain the Lagrangian equations of motion, therefore, the kinetic energy, constraint equations and generalized forces must

be known. An elegant method for deriving these quantities for a general planar dynamic mechanical system is presented in Reference 3. The formulation presented here is largely based on this technique and is known as the constrained multi-element formulation.

2.1 System Description and Kinetic Energy

In the constrained multi element formulation, the position of a planar rigid body is specified in terms of the location and orientation of a centroidal body-fixed local coordinate system with respect to the global cartesian coordinate system. The local coordinate system is represented by the ζ_i-η_i axes, and the global coordinate system by the X-Y axes as shown in Figure 2.1. The position of the body can be completely specified by giving the coordinates of the body centroid in the global system and the angle which the ζ_i-axis makes with the global X-axis. Thus, for body i, we can define a generalized coordinate vector $\underline{q}^i$ as:

$$\underline{q}^i = [x_i \; y_i \; \phi_i]^T \tag{2.3}$$

The vector of system generalized coordinates can be written as:

$$\underline{q} = \left[\underline{q}^{1^T} \quad \underline{q}^{2^T} \quad \ldots\ldots \quad \underline{q}^{n^T}\right]^T \tag{2.4}$$

where the $\underline{q}^i$ are the body generalized coordinate vectors and n is the number of bodies in the system.

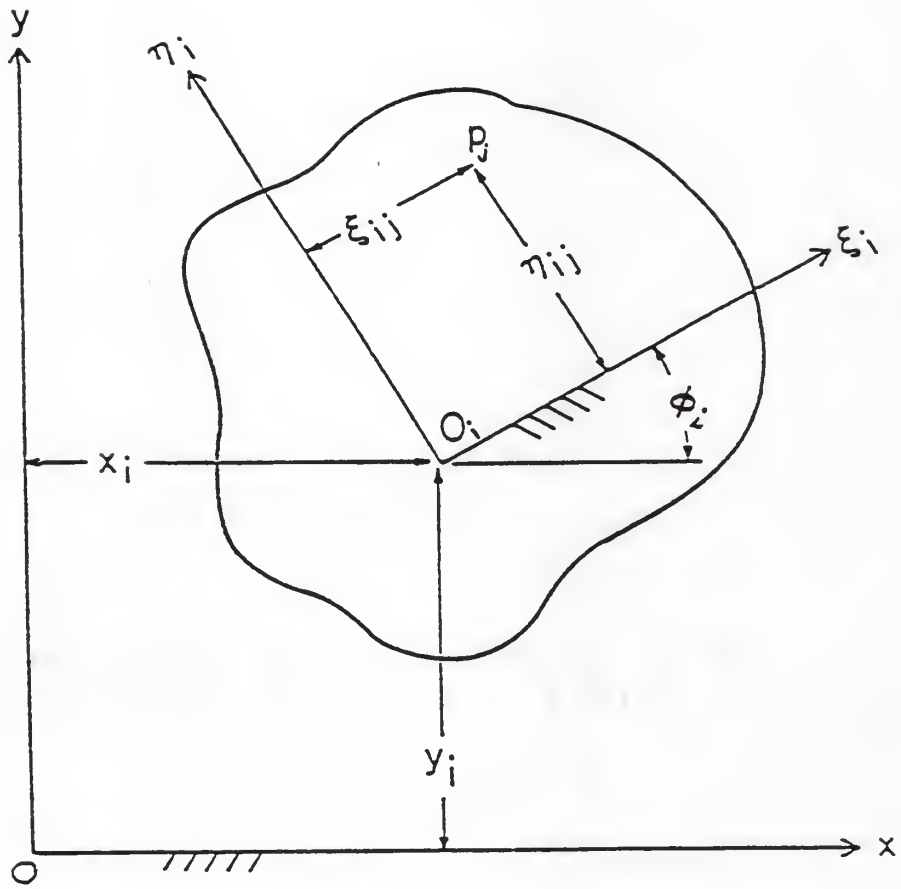

Figure 2.1. Description of a Rigid Body

In terms of these generalized coordinates and the corresponding generalized velocities, the kinetic energy of body i can be written as:

$$T^i = (1/2)\, m_i\, \dot{x}_i^2 + (1/2)\, m_i\, \dot{y}_i^2 + (1/2)\, J_i\, \dot{\phi}_i^2 \qquad (2.5)$$

where m_i and J_i are the mass and moment of inertia of body i respectively. Equation 2.5 can be rewritten in the following quadratic form:

$$T^i = (1/2)\, [\dot{x}_i\ \dot{y}_i\ \dot{\phi}_i] \begin{bmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & J_i \end{bmatrix} \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{\phi}_i \end{bmatrix} \qquad (2.6)$$

From Equation 2.6, it may be seen that the element mass matrix is

$$M^i = \begin{bmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & J_i \end{bmatrix} \qquad (2.7)$$

The kinetic energy of the entire system is then obtained by summing the kinetic energies of the individual bodies as

$$T = (1/2)\, [\dot{q}]^T\, [M]\, [\dot{q}] \qquad (2.8)$$

where

$$[M] = \text{diag}\, [M^1, M^2, \ldots, M^n] \qquad (2.9)$$

2.2 Equations of Constraint

The system description scheme detailed above employs a maximal set of generalized coordinates to describe the system. A body in planar motion can have three degrees of freedom at most, representing

its ability to translate along the X and Y axes and rotate about the Z-axis. The scheme described earlier employs three generalized coordinates to fully describe each body and would require 3n generalized coordinates to describe an entire system of n rigid bodies. The bodies in the system are connected to each other by joints and hence constrained in the manner in which they can move relative to each other. The joints, which place restrictions on the relative movement between connected bodies take away some degrees of freedom from the system and can be represented mathematically by a set of kinematic constraint equations.

The two types of joints commonly encountered in planar mechanical systems are revolute joints and translational joints. Both joints represent the constraining of two degrees of freedom. Therefore, each of these joints requires two equations of constraint to model the effects of the joint. The constraint equations for these two types of joints will now be derived.

a) Revolute Joint: A typical revolute joint between bodies i and j is shown in Figure 2.2 . The vectors $\underline{R}_i$ and $\underline{R}_j$ locate the centroids of bodies i and j in the global coordinate system. The points P_{ij} on body i and P_{ji} on body j are located by the vectors $\underline{r}_{ij}$ and $\underline{r}_{ji}$, which are expressed in their respective body-fixed coordinate systems.

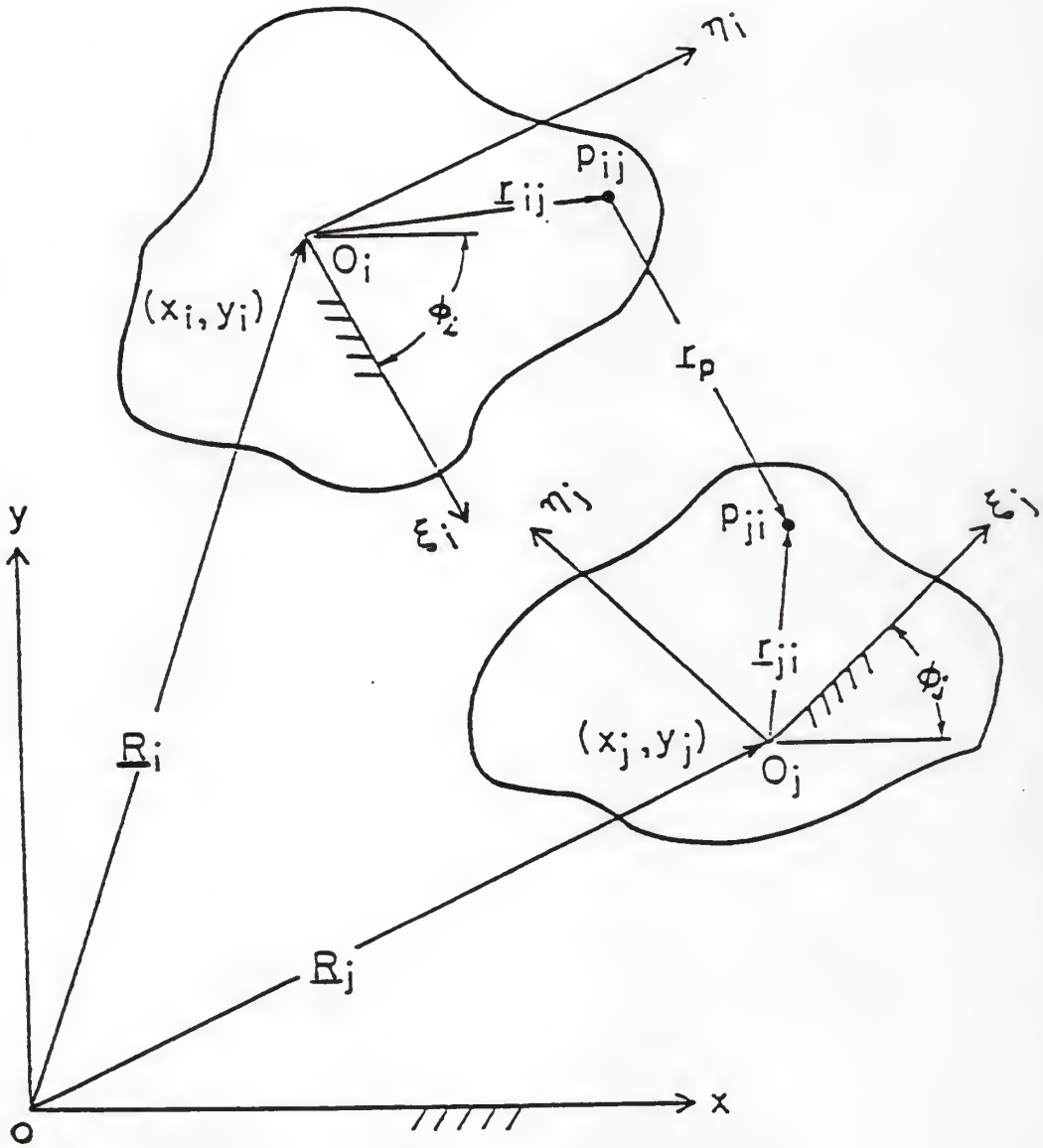

Figure 2.2. Revolute Joint

The vector $\underline{r}_p$ connecting P_{ij} to P_{ji} is given by the vector equation

$$\underline{r}_p = \underline{R}_i + \underline{r}_{ij} - \underline{R}_j - \underline{r}_{ji} \tag{2.10}$$

In order to have a revolute joint between bodies i and j through the points P_{ij} and P_{ji}, it is required that

$$\underline{r}_p = 0 \tag{2.11}$$

This vector equation can be translated into the following pair of scalar equations:

$$\left[\underline{r}_p\right]_x = 0 \tag{2.12}$$

$$\left[\underline{r}_p\right]_y = 0 \tag{2.13}$$

These scalar equations can be expanded to yield

$$x_i + \zeta_{ij}\cos\phi_i - \eta_{ij}\sin\phi_i - x_j - \zeta_{ji}\cos\phi_j + \eta_{ji}\sin\phi_j = 0 \tag{2.14}$$

$$y_i + \zeta_{ij}\sin\phi_i + \eta_{ij}\cos\phi_i - y_j - \zeta_{ji}\sin\phi_j - \eta_{ji}\cos\phi_j = 0 \tag{2.15}$$

where (ζ_{ij},η_{ij}) are the coordinates of P_{ij} in the ζ_i-η_i coordinate system and (ζ_{ji},η_{ji}) are the coordinates of P_{ji} in the ζ_j-η_j coordinate system. Equations 2.14 and 2.15 are the required equations of constraint for a revolute joint.

b) Translational Joint: A typical translational joint between bodies i and j is shown in Figure 2.3. The joint is completely specified by specifying the vectors $\underline{r}_i$ and $\underline{r}_j$ in bodies i and j respectively such that the two vectors are collinear and parallel to

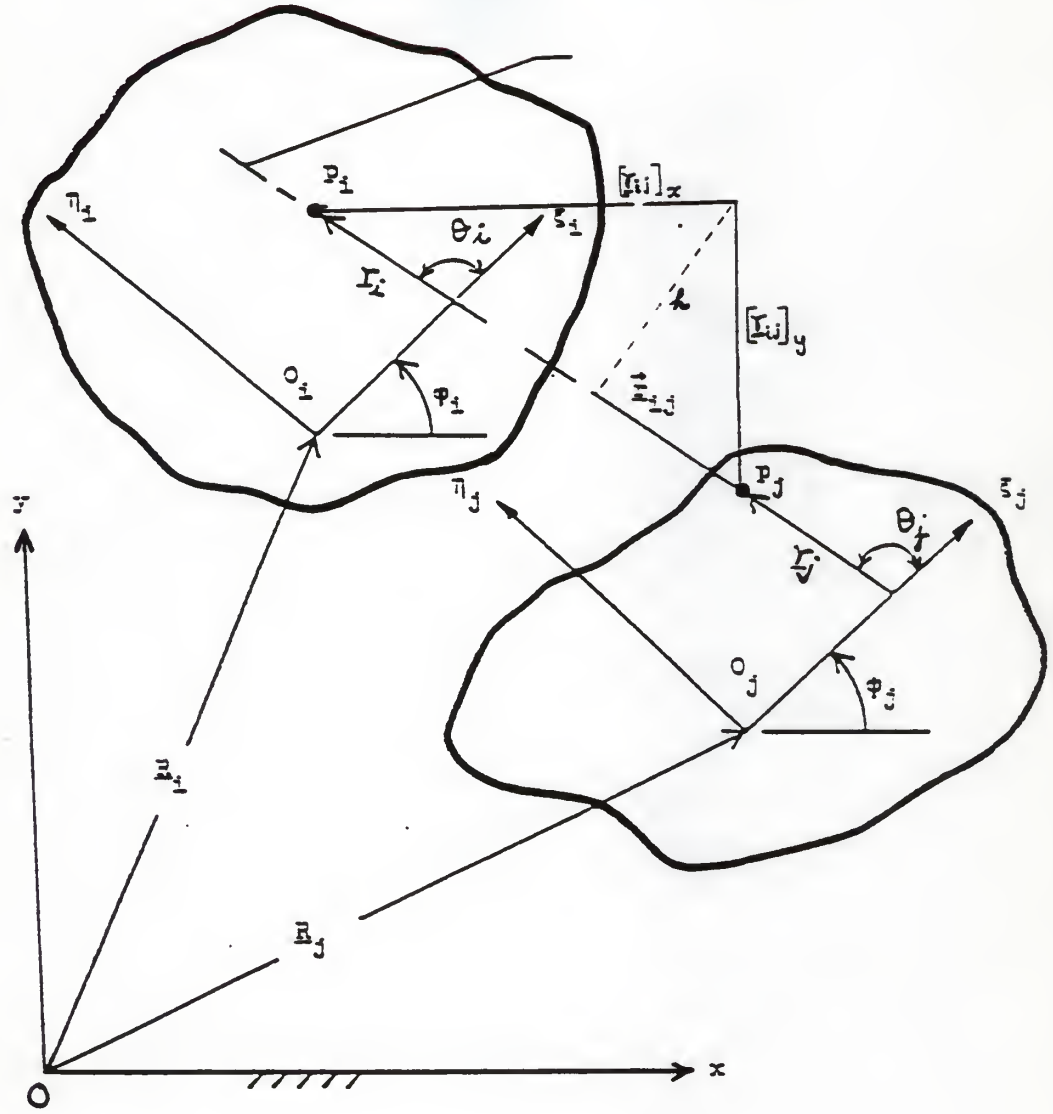

Figure 2.3 Translational Joint

the line of relative motion between the two bodies. The vectors $\underline{r}_i$ and $\underline{r}_j$ are specified by specifying the coordinates (ζ_i,η_i) and (ζ_j,η_j) of points P_i and P_j in terms of their respective body-fixed coordinate systems, and the angles θ_i and θ_j made by these vectors with respect to their respective ζ-axes. Further, let $\underline{r}_{ij}$ be the vector connecting P_j to P_i.

The equations of kinematic constraint for the joint are derived from the fact that the following conditions must be satisfied:

a) $\underline{r}_i$ and $\underline{r}_j$ must make the same angle with respect to the global X-axis.

b) The height h of triangle $P_i\ P_j\ T_{ij}$ must be the same when either $\left[\underline{r}_{ij}\right]_x$ or $\left[\underline{r}_{ij}\right]_y$ is projected on a line perpendicular to $\underline{r}_{ij}$.

The first condition translates into the equation

$$\phi_i + \theta_i - \phi_j - \theta_j = 0 \tag{2.16}$$

The second constraint equation is derived as follows:

$$\left[\underline{r}_{ij}\right]_x = (x_i + \zeta_i\cos\phi_i - \eta_i\sin\phi_i) - (x_j + \zeta_j\cos\phi_j - \eta_j\sin\phi_j) \tag{2.17}$$

$$\left[\underline{r}_{ij}\right]_y = (y_i + \zeta_i\sin\phi_i + \eta_i\cos\phi_i) - (y_j + \zeta_j\sin\phi_j + \eta_j\cos\phi_j) \tag{2.18}$$

$$h = \left[\underline{r}_{ij}\right]_x \sin(\pi - \phi_i - \theta_i) = \left[\underline{r}_{ij}\right]_y \cos(\pi - \phi_i - \theta_i) \tag{2.19}$$

combining the above equations yields

$$[x_i + \zeta_i \cos\phi_i - \eta_i \sin\phi_i - x_j - \zeta_j \cos\phi_j + \eta_j \sin\phi_j] \sin(\phi_i + \theta_i) -$$
$$[y_i + \zeta_i \sin\phi_i + \eta_i \cos\phi_i - y_j - \zeta_j \sin\phi_j - \eta_j \cos\phi_j] \cos(\phi_i + \theta_i) = 0 \quad (2.20)$$

Equations 2.16 and 2.20 represent the kinematic constraint equations for a translational joint.

2.3 Generalized Forces

The Lagrangian formulation requires that all forces acting on the system be written in terms of generalized forces, i.e., they must be expressed in terms of their components along the generalized coordinate directions. Thus, for the system of generalized coordinates described in Section 2.1, we can write the vector of generalized forces for body i as

$$\underline{Q} = \begin{bmatrix} Q_x^i & Q_y^i & Q_z^i \end{bmatrix} \quad (2.21)$$

The system generalized force vector then becomes

$$\underline{Q} = \begin{bmatrix} \underline{Q}^{1T} & \underline{Q}^{2T} & \ldots\ldots & \underline{Q}^{nT} \end{bmatrix}^T \quad (2.22)$$

where n is the number of bodies in the system.

The conversion of a given force $\underline{F}$ acting on body i, as shown in Figure 2.4, into generalized forces is done by resolving the force along the along the X and Y axes and finding the equivalent moment of the force about the origin of body-fixed ζ_i–η_i coordinate system.

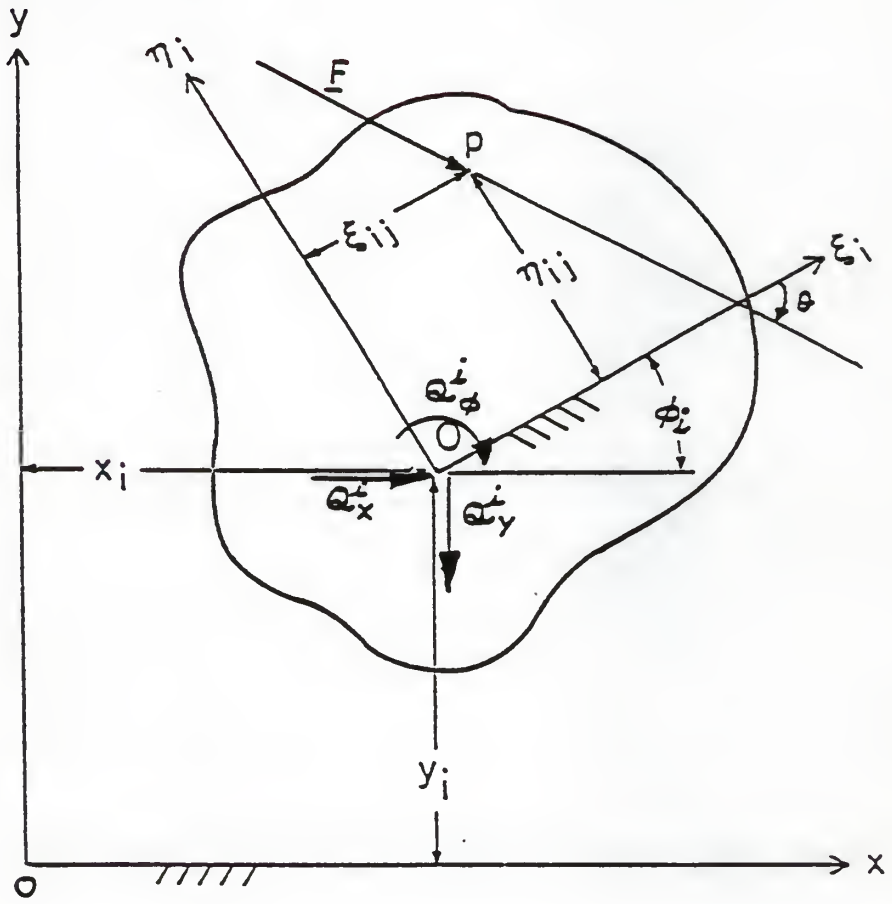

Figure 2.4. Conversion of Applied Forces to Generalized Forces

Thus, the contribution of this force to the body generalized force vector is given by

$$\left[Q^i\right]_x = \underline{F}_x = F \cos(\phi_i + \theta) \tag{2.23}$$

$$\left[Q^i\right]_y = \underline{F}_y = F \sin(\phi_i + \theta) \tag{2.24}$$

$$\left[Q^i\right]_\phi = \underline{F}_x \left(-\zeta_i \sin \phi_i - \eta_i \cos \phi_i \right) + \underline{F}_y \left(\zeta_i \cos \phi_i - \eta_i \sin \phi_i \right) \tag{2.25}$$

The above relationship does not depend on the nature of the force. The force could be conservative or non-conservative; it could be a constant or a function of design, position, velocity, time or any combination of these factors.

One particular type of force that is of special interest is the force generated by a linear spring-damper-actuator combination within the system. Such a combination between bodies i and j is shown in Figure 2.5. The effect of this combination is accounted for by calculating the force developed in the combination and treating it in the manner described earlier, i.e., determining its contribution to the body generalized forces. For the configuration shown, the spring-damper-actuator force is given by

$$\underline{F}_{ij} = \left[K_{ij} \left(\ell_{ij} - \ell_{0_{ij}}\right) + C_{ij} \dot{\ell}_{ij} + \underline{F}_{0_{ij}}\right] \hat{\underline{R}}_{ij} \tag{2.26}$$

where

K_{ij} is the spring constant,

C_{ij} is the damping coefficient,

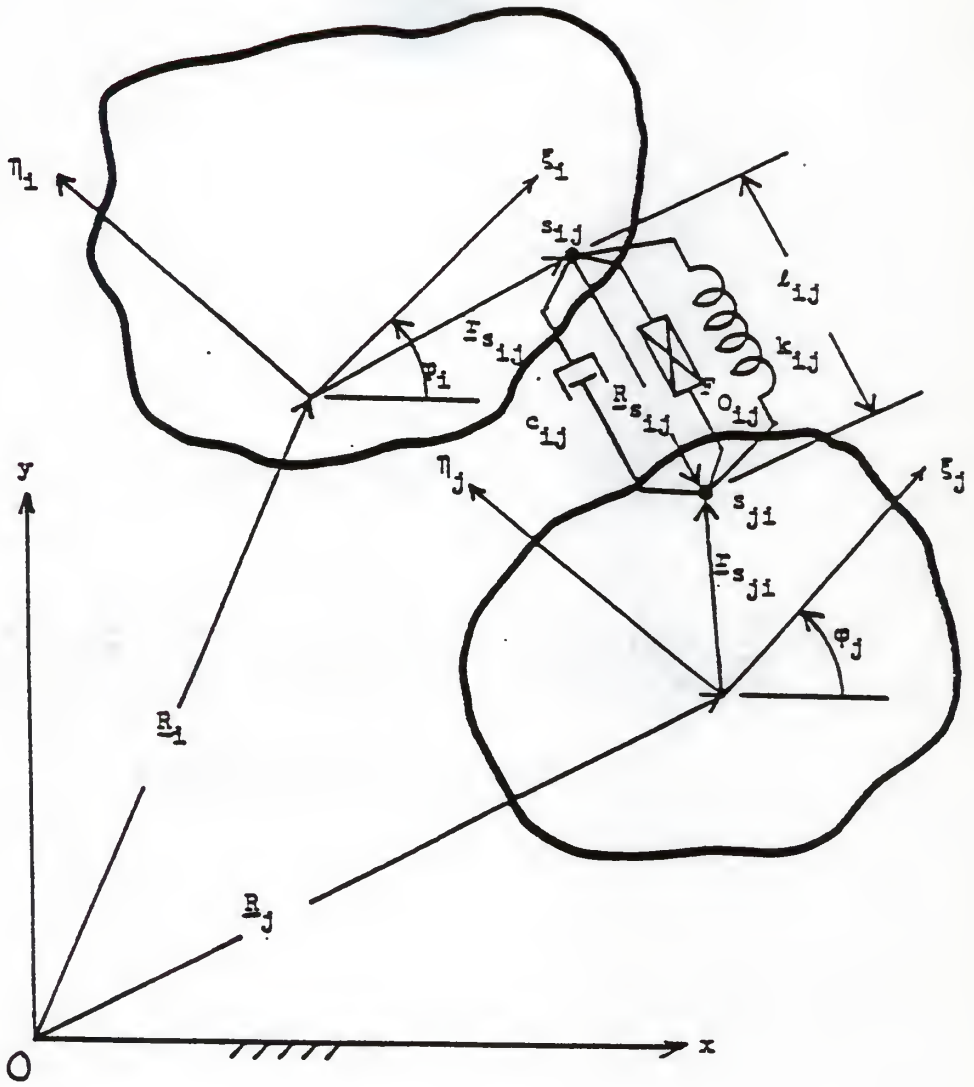

Figure 2.5. Linear Spring-Damper-Actuator Combination

$F_{0_{ij}}$ is the actuator force,

$\ell_{0_{ij}}$ is the unstretched length of the spring, and

$\hat{R}_{ij}$ is the unit vector along the line joining the connecting points on the two bodies.

Once F_{ij} is known, its contributions to the body generalized forces can be obtained using Equations 2.23 through 2.25.

2.4 System Equations of Motion and Constraint

The expression for kinetic energy in Equation 2.8 can be substituted into the Lagrangian equations of motion of Equation 2.2 to yield the system equations of motion in the form

$$[M]\,\ddot{q} - [\Phi_q]^T \lambda = Q \tag{2.27}$$

The initial condition for these differential equations are specified by specifying the initial positions and velocities of as many generalized coordinates as the number of degrees of freedom in the system.

These equations and the equations of kinematic constraint of Equation 2.1 together constitute a mixed system of algebraic and differential equations that has to be solved simultaneously to obtain the dynamic response of the system. However, the equations of kinematic constraint represent a set of non-linear algebraic equations which may not yield an easy solution in the form in which they appear

in Equation 2.1. It is, therefore, advantageous to convert the constraint equations to a more convenient form as follows.

Equation 2.1 is first differentiated with respect to time to yield

$$\Phi_{\underline{q}} \dot{\underline{q}} = - \Phi_t \tag{2.28}$$

Differentiating Equation 2.28 once again with respect to time and transposing some terms to the right, we get

$$\Phi_{\underline{q}} \ddot{\underline{q}} = - 2 (\Phi_{t\underline{q}}) \dot{\underline{q}} - \Phi_{tt} - (\Phi_{\underline{q}} \tilde{\dot{\underline{q}}})_{\underline{q}} \dot{\underline{q}} \tag{2.29}$$

where subscripts indicate partial differentiation and a tilde (~) over a quantity indicates that it is held constant during partial differentiation.

The last term in Equation 2.29 represents a quantity which can also be written as $(\Phi_{\underline{qq}} \dot{\underline{q}}) \dot{\underline{q}}$, i.e., the vector of kinematic constraints differentiated twice with respect to the position vector and subsequently multiplied twice with the velocity vector. Although the two terms are equivalent, the term with the tilde (~) is easier to evaluate. This is due to the fact that the quantity $\Phi_{\underline{qq}}$ in the term $(\Phi_{\underline{qq}} \dot{\underline{q}}) \dot{\underline{q}}$ is a third order tensor while all the quantities in the term $(\Phi_{\underline{q}} \tilde{\dot{\underline{q}}})_{\underline{q}} \dot{\underline{q}}$ are at most second order terms (matrices). Therefore, using the tilde to perform the multiplication of $\Phi_{\underline{q}}$ with $\dot{\underline{q}}$ before taking the partial derivative with respect to $\underline{q}$ a second time reduces the order

of the of the intermediate quantity by one. The use of the tilde to reduce the order of intermediate quantities encountered during partial differentiation is also resorted to in the derivation of equations related to first and second order design sensitivity analyses.

Equation 2.29 represents the equations of constraint in their well-known "Gaussian form". These constraint equations can be combined with the equations of motion of Equation 2.27 to obtain a combined set of equations that can be written using matrix notation as

$$\begin{bmatrix} M & \Phi_{\underline{q}}^T \\ \Phi_{\underline{q}} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\underline{q}} \\ -\underline{\lambda} \end{bmatrix} = \begin{bmatrix} \underline{Q} \\ -2\,(\Phi_{t\underline{q}})\,\dot{\underline{q}} - \Phi_{tt} - (\Phi_{\underline{q}}\,\dot{\underline{q}})_{\underline{q}}\,\dot{\underline{q}} \end{bmatrix} \tag{2.30}$$

The combined system of equations of motion and kinematic constraint represented by Equation 2.30 along with appropriate initial conditions completely determines the dynamic response of the mechanical system under investigation. The numerical method employed for the solution of these equations is presented in Chapter 5.

CHAPTER III

FIRST ORDER DESIGN SENSITIVITY ANALYSIS

The equations of motion and the equations of kinematic constraint derived in the preceding chapter are in general dependent on the vector of design variables $\underline{b}$. The objective of optimal design is to choose this vector of design variables in a manner that will minimize the cost function subject to some performance constraints. It must be emphasized that these performance constraints refer to conditions imposed on the response of the dynamic system and are not to be confused with the kinematic constraints of Equation 2.1.

It is assumed that the cost and constraint functions are dependent on time, position, velocity, acceleration, Lagrange multipliers and design, i.e., they are of the form

$$\Psi^i = \psi^i(t, \underline{q}, \dot{\underline{q}}, \ddot{\underline{q}}, \underline{\lambda}, \underline{b}) \tag{3.1}$$

The Lagrange multipliers are included because they represent the joint forces in the mechanical system and it is expected that any cost or constraint function involving joint forces, joint stresses, etc. will have to be written in terms of $\underline{\lambda}$.

Most optimization algorithms require the derivatives of cost and constraint functions with respect to design in order to calculate the next improved design. The aim of first order design sensitivity analysis is to evaluate these derivatives upto the first order, i.e.,

to find the matrix of design sensitivity coefficients, denoted by 1, that relates a small change in design parameters to the resulting change in cost/constraint function value through the equation

$$d\Psi = [1]^T \, d\underline{b} \tag{3.2}$$

The cost and constraint functions can be classified into three categories, each of which has to be treated differently when design sensitivity is calculated:

a) Integral Constraints: These are constraints of the form

$$\Psi^i = \int_{t_0}^{t_1} \psi^i(t, \underline{q}, \dot{\underline{q}}, \ddot{\underline{q}}, \underline{\lambda}, \underline{b}) \, dt \tag{3.3}$$

where t_0 and t_1 are the initial and final times between which the system response is to be studied. A constraint of this type arises when we deal with quantities like integrated square error, total work input, etc. In addition, pointwise constraints, i.e., constraints that must be satisfied at each point within the interval from t_0 to t_1 can also be formulated as integral constraints. A general pointwise constraint of the form

$$\phi^i(t, \underline{q}, \dot{\underline{q}}, \ddot{\underline{q}}, \underline{\lambda}, \underline{b}) \leq 0 \ , \quad t_0 \leq t \leq t_1 \tag{3.4}$$

can be formulated as the equivalent integral constraint

$$\Psi^i = \int_{t_0}^{t_1} [\phi^i + |\phi^i|]^2 \, dt = 0 \tag{3.5}$$

The squaring of the integrand is necessary to ensure that the constraint function is continuously differentiable throughout the time interval of interest.

Although this formulation of pointwise constraints has been used with success in Reference 4, it has the drawback of increasing the non-linearity of the problem owing to the squaring of the integrand in Equation 3.5. This could slow down the rate of convergence. Further, since the minimum value of the functional in Equation 3.5 is zero, this causes the integral constraint to appear ϵ-active even when the corresponding pointwise constraint is comfortably satisfied.

b) Grid-point Constraints: This is an alternate method of imposing pointwise constraints. In this approach, the simulation interval from t_0 to t_1 is subdivided into a suitable number of subintervals with the end points of these intervals being considered as "Grid-points". The given pointwise constraint is then imposed independently at these grid-points. Thus, the pointwise constraint of Equation 3.4 becomes

$$\psi_1^i = \psi_1^i(t_1, \underline{q}, \dot{\underline{q}}, \ddot{\underline{q}}, \underline{\lambda}, \underline{b}) \leq 0 \ , \quad 1 = 1, 2, \ldots, m \qquad (3.6)$$

where $\underline{q}$, $\dot{\underline{q}}$, $\ddot{\underline{q}}$ and $\underline{\lambda}$ represent the instantaneous values of the respective quantities at time t_1 and 1 represents the grid-point number and m is the total number of grid-points in the interval from t_0 to t_1.

Although the grid-point formulation has the disadvantage of substituting a single pointwise constraint by several separate constraints, it has been shown to be an effective and feasible method of imposing pointwise constraints [5]. The effective use of grid-point constraints depends on a proper balance being achieved between

accuracy and efficiency by choosing a subinterval size that is neither too large nor too small.

c) Purely Design-dependent Constraints: These are constraints that have an explicit dependence only on the design variables and are of the form

$$\Psi^i = \psi^i(\underline{b}) \tag{3.7}$$

Since these constraints donot depend on position, velocity, acceleration or Lagrange multipliers, the sensitivity of these constraints can be obtained by simply differentiating Equation 3.7 with respect to design. Owing to their simplicity, design-dependent constraints are almost always treated separately during design sensitivity analysis.

3.1 First Order Design Sensitivity Analysis by Direct Differentiation

The total differential $d\Psi$ on the left hand side of Equation 3.2 can be written as a sum of partial differentials, with each of the partial differentials accounting for the dependence of the cost/constraint function Ψ on design or state variables. Based on this idea, the following expressions can be obtained for the three types of constraint functions described by Equations 3.3, 3.6 and 3.7.

a) For integral constraints:

$$d\Psi = \int_{t_0}^{t_1} [\psi^i_{\underline{q}}\, d\underline{q} + \psi^i_{\dot{\underline{q}}}\, d\dot{\underline{q}} + \psi^i_{\ddot{\underline{q}}}\, d\ddot{\underline{q}} + \psi^i_{\underline{\lambda}}\, d\underline{\lambda} + \psi^i_{\underline{b}}\, d\underline{b}\ ]\, dt \tag{3.8}$$

b) For grid-point constraints:

$$d\Psi = \psi^i_{\underline{q}}\, d\underline{q} + \psi^i_{\dot{\underline{q}}}\, d\dot{\underline{q}} + \psi^i_{\ddot{\underline{q}}}\, d\ddot{\underline{q}} + \psi^i_{\underline{\lambda}}\, d\underline{\lambda} + \psi^i_{\underline{b}}\, d\underline{b} \qquad i = 1, \ldots, m \quad (3.9)$$

c) For purely design-dependent constraints:

$$d\Psi = \psi^i_{\underline{b}}\, d\underline{b} \quad (3.10)$$

where the subscripts denote partial differentiation.

Comparing Equation 3.10 with Equation 3.2, it is evident that for purely design-dependent constraints, the sensitivity matrix 1 is given by:

$$1 = \left[\psi^i_{\underline{b}}\right]^T \quad (3.11)$$

Thus, if the constraint function is known, it can be directly differentiated to yield the first order design sensitivity matrix. Hence, further discussion on first order design sensitivity will not refer to purely design dependent constraints.

Equations 3.8 and 3.9 contain terms representing differential changes in position, velocity, acceleration and Lagrange multipliers, in addition to the differential change in design, thereby complicating the process of evaluating the sensitivity matrix. The adjoint variable method [4,9] avoids computing these differential changes explicitly; instead, an expression for the sensitivity is derived in terms of a set of adjoint variables which are then evaluated and used to find numerical values of the sensitivity matrix coefficients.

The method of direct differentiation [5,10], seeks to evaluate the differential changes in position, velocity, acceleration and Lagrange multipliers in terms of the differential change in design.

This information, which is the design sensitivity of state variables, is then used to calculate the design sensitivity matrix for cost and constraint functions in the following manner:

Since position, velocity, acceleration and Lagrange multipliers are all dependent on design, it follows that

$$d\underline{q} = \underline{q}_{\underline{b}}\, d\underline{b} \tag{3.12}$$

$$d\dot{\underline{q}} = \dot{\underline{q}}_{\underline{b}}\, d\underline{b} \tag{3.13}$$

$$d\ddot{\underline{q}} = \ddot{\underline{q}}_{\underline{b}}\, d\underline{b} \tag{3.14}$$

$$d\underline{\lambda} = \underline{\lambda}_{\underline{b}}\, d\underline{b} \tag{3.15}$$

where the subscripts denote partial differentiation.

Based on the above four equations, it is possible to rewrite Equations 3.8 and 3.9 as

$$d\Psi = \left\{ \int_{t_0}^{t_1} [\psi^i_{\underline{q}}\, \underline{q}_{\underline{b}} + \psi^i_{\dot{\underline{q}}}\, \dot{\underline{q}}_{\underline{b}} + \psi^i_{\ddot{\underline{q}}}\, \ddot{\underline{q}}_{\underline{b}} + \psi^i_{\underline{\lambda}}\, \underline{\lambda}_{\underline{b}} + \psi^i_{\underline{b}}\,]\, dt \right\} d\underline{b} \tag{3.16}$$

for integral constraints, and

$$d\Psi = [\psi^i_{\underline{q}}\, \underline{q}_{\underline{b}} + \psi^i_{\dot{\underline{q}}}\, \dot{\underline{q}}_{\underline{b}} + \psi^i_{\ddot{\underline{q}}}\, \ddot{\underline{q}}_{\underline{b}} + \psi^i_{\underline{\lambda}}\, \underline{\lambda}_{\underline{b}} + \psi^i_{\underline{b}}\,]\, d\underline{b} \tag{3.17}$$

for grid-point constraints.

Comparing Equations 3.8 with 3.16 and 3.9 with 3.17, it follows that

$$1 = \int_{t_0}^{t_1} [\psi^i_{\underline{q}}\, \underline{q}_{\underline{b}} + \psi^i_{\dot{\underline{q}}}\, \dot{\underline{q}}_{\underline{b}} + \psi^i_{\ddot{\underline{q}}}\, \ddot{\underline{q}}_{\underline{b}} + \psi^i_{\underline{\lambda}}\, \underline{\lambda}_{\underline{b}} + \psi^i_{\underline{b}}\,]^T\, dt \tag{3.18}$$

for integral constraints, and

$$1 = [\psi^i_{\underline{q}}\, \underline{q}_{\underline{b}} + \psi^i_{\dot{\underline{q}}}\, \dot{\underline{q}}_{\underline{b}} + \psi^i_{\ddot{\underline{q}}}\, \ddot{\underline{q}}_{\underline{b}} + \psi^i_{\underline{\lambda}}\, \underline{\lambda}_{\underline{b}} + \psi^i_{\underline{b}}\,]^T \tag{3.19}$$

for grid-point constraints.

Therefore, the first order sensitivity matrices in Equations 3.18 and 3.19 can be calculated once the quantities $\underline{q}_{\underline{b}}$, $\dot{\underline{q}}_{\underline{b}}$, $\ddot{\underline{q}}_{\underline{b}}$ and $\underline{\lambda}_{\underline{b}}$ are known. Thus, the relationships expressed in Equations 3.18 and 3.19 convert the problem of finding design sensitivity of cost and constraint functions into a problem of finding design sensitivity of state variables.

Equations for the design sensitivity of state variables can be obtained from the system equations of motion and the equations of kinematic constraint by direct differentiation in the following manner:

It is assumed that the masses and generalized forces in the system are of the form

$$M = M(\underline{b}) \tag{3.20}$$

and

$$\underline{Q} = Q(t, \underline{q}, \dot{\underline{q}}, \underline{b}) \tag{3.21}$$

respectively.

Under these assumptions, the equations of motion of Equation 2.27 can be differentiated with respect to the design variable b_j to yield the following set of equations:

$$M \ddot{\underline{q}}_{b_j} - \Phi_{\underline{q}}^T \underline{\lambda}_{b_j} = - (M \tilde{\ddot{\underline{q}}})_{b_j} + (\Phi_{\underline{q}}^T \tilde{\underline{\lambda}})_{\underline{b}} + (\Phi_{\underline{q}}^T \tilde{\underline{\lambda}})_{\underline{q}} \underline{q}_{b_j} \tag{3.22}$$

$$+ \underline{Q}_{b_j} + \underline{Q}_{\underline{q}} \underline{q}_{b_j} + \underline{Q}_{\dot{\underline{q}}} \dot{\underline{q}}_{b_j}$$

where the subscripts indicate partial differentiation and a tilde (~) above a quantity indicates that it is held constant during partial differentiation. It must be noted that there will be r such sets of equations, where r is the number of design variables.

Equation 3.22 represents a set of linear second order ordinary differential equations for $\underset{\sim}{q}_{b_j}$. The initial conditions for these differential equations are obtained from the fact that at the initial time, the sensitivities corresponding to the independent generalized coordinates must be zero. It is interesting to note that the differential equations for design sensitivity of state are linear, even though the differential equations governing the dynamic response of the system may be non-linear.

However, the set of equations in Equation 3.22 is not sufficient to completely solve for the design sensitivities of all the state variables. The additional equations required to completely determine the first order design sensitivities of the state variables can be obtained by differentiating the equations of kinematic constraint in the following manner:

Differentiating Equation 2.1, the equation of kinematic constraint with respect to design variable b_j yields

$$\Phi_{b_j} + \Phi_{\underset{\sim}{q}} \underset{\sim}{q}_{b_j} = 0 \tag{3.23}$$

which can be rewritten as

$$\Phi_{\underset{\sim}{q}} \underset{\sim}{q}_{b_j} = - \Phi_{b_j} \tag{3.24}$$

Taking the total time derivative of Equation 3.24 and noting that for a scleronomic system, Φ does not have an explicit dependence on time, we get

$$\Phi_{\underline{q}} \dot{\underline{q}}_{b_j} = - \dot{\Phi}_{\underline{q}} \underline{q}_{b_j} - \dot{\Phi}_b \tag{3.25}$$

Taking the total time derivative of Equation 3.25, we get

$$\Phi_{\underline{q}} \ddot{\underline{q}}_{b_j} = - 2 \dot{\Phi}_{\underline{q}} \dot{\underline{q}}_{b_j} - \ddot{\Phi}_{\underline{q}} \underline{q}_{b_j} - \ddot{\Phi}_{b_j} \tag{3.26}$$

Equations 3.22 and 3.26 can be combined and written in matrix form as

$$\begin{bmatrix} M & \Phi_{\underline{q}}^T \\ \Phi_{\underline{q}} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\underline{q}}_{b_j} \\ -\underline{\lambda}_{b_j} \end{bmatrix} = \begin{bmatrix} -(M\ddot{\underline{q}})_{b_j} + (\Phi_{\underline{q}}^T \tilde{\underline{\lambda}})_{b_j} + (\Phi_{\underline{q}}^T \tilde{\underline{\lambda}})_{\underline{q}} \underline{q}_{b_j} + Q_{b_j} + Q_{\underline{q}} \underline{q}_{b_j} + Q_{\dot{\underline{q}}} \dot{\underline{q}}_{b_j} \\ -2 \dot{\Phi}_{\underline{q}} \dot{\underline{q}}_{b_j} - \ddot{\Phi}_q \underline{q}_{b_j} - \ddot{\Phi}_{b_j} \end{bmatrix} \tag{3.27}$$

This system of equations and the combined equations of motion and kinematic constraint of Equation 2.30 can be integrated simultaneously to obtain the dynamic response of the system and the design sensitivity of state variables at the same time. Once the design sensitivity of the state variables is known, the design sensitivity of cost and constraint functions can be obtained using the relationship between the two as given by Equations 3.18 and 3.19 for the two types of performance constraints. The technique employed for solving the above mixed system of algebraic and differential equations is discussed in Chapter 5.

A comparison of Equation 3.27 with Equation 2.30 shows that the combined equations of motion and kinematic constraint as well as the

equations for first order sensitivity analysis have the same coefficient matrix. In fact, from the manner in which the equations for first order sensitivity analysis were derived, it can be inferred that the coefficient matrix would in fact remain the same for any higher order design sensitivity analysis. This is on account of the fact that additional terms in the left hand side resulting from the differentiation of a lower order sensitivity equation, with the exception of the required design sensitivity, would be transferred to the right hand side. This ensures that the coefficient matrix is preserved. In fact, it is even possible to extend this line of thought backward and think of the dynamic analysis of the system as zeroth order design sensitivity analysis.

CHAPTER IV

SECOND ORDER DESIGN SENSITIVITY ANALYSIS

4.1 Second Order Design Sensitivity Analysis by Direct Differentiation

It was seen in the previous chapter that the problem of finding the first order design sensitivity of the cost and constraint functions could be reduced to a problem of finding the first order design sensitivity of the state variables. It was also seen that a combined set of equations for first order state sensitivity could be obtained by differentiating the system equations of motion and the equations of kinematic constraint with respect to design. The same approach can be extended to the problem of evaluating the second order design sensitivity of the cost and constraint functions. For the three types of performance constraints identified in Chapter 3, the second order design sensitivity can be evaluated as follows:

a) Integral Constraints: Equation 3.16, which gives an expression for the differential change $d\Psi^i$ in integral constraint Ψ^i, can be rewritten as an expression for the partial derivative of Ψ^i with respect to design variable b_j as

$$\frac{\partial \Psi^i}{\partial b_j} = \int_{t_0}^{t_f} [\psi^i_{\underline{q}} \underline{q}_{b_j} + \psi^i_{\dot{\underline{q}}} \dot{\underline{q}}_{b_j} + \psi^i_{\ddot{\underline{q}}} \ddot{\underline{q}}_{b_j} + \psi^i_{\underline{\lambda}} \underline{\lambda}_{b_j} + \psi^i_{b_j}] \, dt \qquad j = 1, \ldots r \tag{4.1}$$

Equation 4.1 can be differentiated with respect to design variable b_k to yield an expression for the partial derivative $\frac{\partial^2 \Psi^i}{\partial b_j \partial b_k}$, which is one term in the r × r square matrix of second derivatives of Ψ^i with respect to design. In carrying out this differentiation, it is necessary to bear in mind the fact that the partial dervatives of Ψ^i with respect to $\underline{q}$, $\dot{\underline{q}}$, $\ddot{\underline{q}}$, $\underline{\lambda}$ and $\underline{b}$ are once again dependent on these quantities. On carrying out the differentiation, we get

$$\frac{\partial^2 \Psi^i}{\partial b_j \partial b_k} = \int_{t_0}^{t_1} \Bigg[\psi^i_{\underline{q}} \underline{q}_{b_j b_k} + \psi^i_{\underline{q} b_k} \underline{q}_{b_j} + \left[\psi^i_{\underline{q}} \tilde{\underline{q}}_{b_j}\right]_{\underline{q}} \underline{q}_{b_k} + \left[\psi^i_{\underline{q}} \tilde{\underline{q}}_{b_j}\right]_{\dot{\underline{q}}} \dot{\underline{q}}_{b_k}$$

$$+ \left[\psi^i_{\underline{q}} \tilde{\underline{q}}_{b_j}\right]_{\ddot{\underline{q}}} \ddot{\underline{q}}_{b_k} + \left[\psi^i_{\underline{q}} \tilde{\underline{q}}_{b_j}\right]_{\underline{\lambda}} \underline{\lambda}_{b_k} + \psi^i_{\dot{\underline{q}}} \dot{\underline{q}}_{b_j b_k} + \psi^i_{\dot{\underline{q}} b_k} \dot{\underline{q}}_{b_j}$$

$$+ \left[\psi^i_{\dot{\underline{q}}} \tilde{\dot{\underline{q}}}_{b_j}\right]_{\underline{q}} \underline{q}_{b_k} + \left[\psi^i_{\dot{\underline{q}}} \tilde{\dot{\underline{q}}}_{b_j}\right]_{\dot{\underline{q}}} \dot{\underline{q}}_{b_k} + \left[\psi^i_{\dot{\underline{q}}} \tilde{\dot{\underline{q}}}_{b_j}\right]_{\ddot{\underline{q}}} \ddot{\underline{q}}_{b_k} + \left[\psi^i_{\dot{\underline{q}}} \tilde{\dot{\underline{q}}}_{b_j}\right]_{\underline{\lambda}} \underline{\lambda}_{b_k}$$

$$+ \psi^i_{\ddot{\underline{q}}} \ddot{\underline{q}}_{b_j b_k} + \psi^i_{\ddot{\underline{q}} b_k} \ddot{\underline{q}}_{b_j} + \left[\psi^i_{\ddot{\underline{q}}} \tilde{\ddot{\underline{q}}}_{b_j}\right]_{\underline{q}} \underline{q}_{b_k} + \left[\psi^i_{\ddot{\underline{q}}} \tilde{\ddot{\underline{q}}}_{b_j}\right]_{\dot{\underline{q}}} \dot{\underline{q}}_{b_k}$$

$$+ \left[\psi^i_{\ddot{\underline{q}}} \tilde{\ddot{\underline{q}}}_{b_j}\right]_{\ddot{\underline{q}}} \ddot{\underline{q}}_{b_k} + \left[\psi^i_{\ddot{\underline{q}}} \tilde{\ddot{\underline{q}}}_{b_j}\right]_{\underline{\lambda}} \underline{\lambda}_{b_k} + \psi^i_{\underline{\lambda}} \underline{\lambda}_{b_j b_k} + \psi^i_{\underline{\lambda} b_k} \underline{\lambda}_{b_j}$$

$$+ \left[\psi^i_{\underline{\lambda}} \tilde{\underline{\lambda}}_{b_j}\right]_{\underline{q}} \underline{q}_{b_k} + \left[\psi^i_{\underline{\lambda}} \tilde{\underline{\lambda}}_{b_j}\right]_{\dot{\underline{q}}} \dot{\underline{q}}_{b_k} + \left[\psi^i_{\underline{\lambda}} \tilde{\underline{\lambda}}_{b_j}\right]_{\ddot{\underline{q}}} \ddot{\underline{q}}_{b_k} + \left[\psi^i_{\underline{\lambda}} \tilde{\underline{\lambda}}_{b_j}\right]_{\underline{\lambda}} \underline{\lambda}_{b_k}$$

$$+ \psi^i_{b_j b_k} + \psi^i_{b_j \underline{q}} \underline{q}_{b_k} + \psi^i_{b_j \dot{\underline{q}}} \dot{\underline{q}}_{b_k} + \psi^i_{b_j \ddot{\underline{q}}} \ddot{\underline{q}}_{b_k} + \psi^i_{b_j \underline{\lambda}} \underline{\lambda}_{b_k} \Bigg] \, dt \quad (4.2)$$

b) Grid-point Constraints : Equation 3.17 for grid-point constraints can be rewritten as

$$\frac{\partial \Psi}{\partial b_j} = [\psi^i_{\underline{q}}\, q_{b_j} + \psi^i_{\dot{\underline{q}}}\, \dot{q}_{b_j} + \psi^i_{\ddot{\underline{q}}}\, \ddot{q}_{b_j} + \psi^i_{\underline{\lambda}}\, \underline{\lambda}_{b_j} + \psi^i_{b_j}]\, dt \qquad \begin{matrix} i = 1, \ldots m \\ j = 1, \ldots r \end{matrix} \tag{4.3}$$

where m is the total number of grid-points.

Differentiating Equation 4.3 with respect to design variable b_k, we get

$$\begin{aligned}
\frac{\partial^2 \Psi}{\partial b_j \partial b_k} = \Bigg[& \psi^i_{\underline{q}}\, q_{b_j b_k} + \psi^i_{\underline{q} b_k}\, q_{b_j} + \left[\psi^i_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\underline{q}}\, q_{b_k} + \left[\psi^i_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\dot{\underline{q}}}\, \dot{q}_{b_k} \\
& + \left[\psi^i_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\ddot{\underline{q}}}\, \ddot{q}_{b_k} + \left[\psi^i_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\underline{\lambda}}\, \underline{\lambda}_{b_k} + \psi^i_{\dot{\underline{q}}}\, \dot{q}_{b_j b_k} + \psi^i_{\dot{\underline{q}} b_k}\, \dot{q}_{b_j} \\
& + \left[\psi^i_{\dot{\underline{q}}}\, \tilde{\dot{q}}_{b_j}\right]_{\underline{q}}\, q_{b_k} + \left[\psi^i_{\dot{\underline{q}}}\, \tilde{\dot{q}}_{b_j}\right]_{\dot{\underline{q}}}\, \dot{q}_{b_k} + \left[\psi^i_{\dot{\underline{q}}}\, \tilde{\dot{q}}_{b_j}\right]_{\ddot{\underline{q}}}\, \ddot{q}_{b_k} + \left[\psi^i_{\dot{\underline{q}}}\, \tilde{\dot{q}}_{b_j}\right]_{\underline{\lambda}}\, \underline{\lambda}_{b_k} \\
& + \psi^i_{\ddot{\underline{q}}}\, \ddot{q}_{b_j b_k} + \psi^i_{\ddot{\underline{q}} b_k}\, \ddot{q}_{b_j} + \left[\psi^i_{\ddot{\underline{q}}}\, \tilde{\ddot{q}}_{b_j}\right]_{\underline{q}}\, q_{b_k} + \left[\psi^i_{\ddot{\underline{q}}}\, \tilde{\ddot{q}}_{b_j}\right]_{\dot{\underline{q}}}\, \dot{q}_{b_k} \\
& + \left[\psi^i_{\ddot{\underline{q}}}\, \tilde{\ddot{q}}_{b_j}\right]_{\ddot{\underline{q}}}\, \ddot{q}_{b_k} + \left[\psi^i_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\underline{\lambda}}\, \underline{\lambda}_{b_k} + \psi^i_{\underline{\lambda}}\, \underline{\lambda}_{b_j b_k} + \psi^i_{\underline{\lambda} b_k}\, \underline{\lambda}_{b_j} \\
& + \left[\psi^i_{\underline{\lambda}}\, \tilde{\underline{\lambda}}_{b_j}\right]_{\underline{q}}\, q_{b_k} + \left[\psi^i_{\underline{\lambda}}\, \tilde{\underline{\lambda}}_{b_j}\right]_{\dot{\underline{q}}}\, \dot{q}_{b_k} + \left[\psi^i_{\underline{\lambda}}\, \tilde{\underline{\lambda}}_{b_j}\right]_{\ddot{\underline{q}}}\, \ddot{q}_{b_k} + \left[\psi^i_{\underline{\lambda}}\, \tilde{\underline{\lambda}}_{b_j}\right]_{\underline{\lambda}}\, \underline{\lambda}_{b_k} \\
& + \psi^i_{b_j b_k} + \psi^i_{b_j \underline{q}}\, q_{b_k} + \psi^i_{b_j \dot{\underline{q}}}\, \dot{q}_{b_k} + \psi^i_{b_j \ddot{\underline{q}}}\, \ddot{q}_{b_k} + \psi^i_{b_j \underline{\lambda}}\, \underline{\lambda}_{b_k} \Bigg]
\end{aligned} \tag{4.4}$$

c) Purely Design-dependent Constraints: Since these constraints have a dependence only on the design variables, they can be directly differentiated twice to yield the second derivative of the constraint with respect to design, i.e.,

$$\frac{\partial^2 \Psi^i}{\partial b_j \partial b_k} = \frac{\partial}{\partial b_k}\left[\frac{\partial}{\partial b_j}(\psi^i)\right] \tag{4.5}$$

The second order design sensitivity of purely design dependent constraints does not depend on the design sensitivity of the state variables. They can be evaluated directly and, therefore, such constraints will be omitted from future discussions on second order design sensitivity.

From Equations 4.2 and 4.4, it can be seen that for integral constraints and grid-point constraints, second order design sensitivity can be evaluated once the second order design sensitivities of position, velocity, acceleration and Lagrange multipliers are known. Equations for the second order design sensitivity of the state variables can be derived by differentiating the corresponding equations already derived for first order state sensitivity in the following manner:

Differentiating Equation 3.26, which is itself the system equations of motion differentiated with respect to b_j, with respect to design variable b_k and keeping track of all the dependancies, we get

$$
\begin{aligned}
M\,\ddot{q}_{b_j b_k} - \Phi_q^T\,\lambda_{b_j b_k} = &- \left[M\,\tilde{\ddot{q}}_{b_j}\right]_{b_k} - \left[M\,\tilde{\ddot{q}}\right]_{b_j b_k} - \left[M\,\tilde{\ddot{q}}_{b_k}\right]_{b_j} + \Phi^T_{q b_k}\,\lambda_{b_j} \\
&+ \left[\Phi_q^T\,\tilde{\lambda}_{b_j}\right]_q\,q_{b_k} + \left[\Phi_q^T\,\lambda\right]_{b_j b_k} + \left[\Phi_q^T\,\tilde{\lambda}\right]_{b_j q}\,q_{b_k} \\
&+ \left[\Phi_q^T\,\tilde{\lambda}_{b_j}\right]_{b_k} + \left[\Phi_q^T\,\tilde{\lambda}\right]_{q b_k}\,q_{b_j} \\
&+ \left[\left[\Phi_q^T\,\tilde{\lambda}\right]_q\,\tilde{q}_{b_j}\right]_q\,q_{b_k} + \left[\Phi_q^T\,\tilde{\lambda}_{b_j}\right]_q\,q_{b_k} \\
&+ \left[\Phi_q^T\,\tilde{\lambda}\right]_q\,q_{b_j b_k} + Q_{b_j b_k} + Q_{b_j q}\,q_{b_k} + Q_{b_j \dot{q}}\,\dot{q}_{b_k} \\
&+ Q_{q b_k}\,q_{b_j} + \left[Q_q\,\tilde{q}_{b_j}\right]_q\,q_{b_k} + \left[Q_q\,\tilde{q}_{b_j}\right]_{\dot{q}}\,\dot{q}_{b_k} \\
&+ Q_q\,q_{b_j b_k} + Q_{\dot{q} b_k}\,\dot{q}_{b_j} + \left[Q_{\dot{q}}\,\tilde{\dot{q}}_{b_j}\right]_q\,q_{b_k} \\
&+ \left[Q_{\dot{q}}\,\tilde{\dot{q}}_{b_j}\right]_{\dot{q}}\,\dot{q}_{b_k} + Q_{\dot{q}}\,\dot{q}_{b_j b_k} \qquad (4.6)
\end{aligned}
$$

where the subscripts again indicate partial differentiation and a tilde (~) above a quantity indicates that it is held constant during partial differentiation. Equation 4.6 represents a set of linear second order ordinary differential equations in $q_{b_j b_k}$. The initial conditions for these differential equations follow from the fact that at the initial time, the second order design sensitivity corresponding to the independent generalized coordinates must be zero. It may be noted that there will be r^2 such equations, where r is the number of design variables. However, as in the case of first order design

sensitivity analysis, these equations by themselves are insufficient to completely determine the second order design sensitivities of all the state variables. Additional equations required to carry out the solution are derived from the equations of kinematic constraint in the following manner:

Differentiating Equation 3.24 with respect design variable b_k and transposing some terms to the right hand side, we get

$$\Phi_{\underline{q}}\, \underline{q}_{b_j b_k} = -\Phi_{\underline{q} b_k}\, \underline{q}_{b_j} - \left[\Phi^{i}_{\underline{q}}\, \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\, \underline{q}_{b_k} - \Phi_{b_j \underline{q}}\, \underline{q}_{b_k} - \underline{\Phi}_{b_j b_k} \tag{4.7}$$

Taking the total derivative of Equation 4.7 with respect to time and transposing some terms to the right hand side, we get

$$\begin{aligned}\Phi_{\underline{q}}\, \dot{\underline{q}}_{b_j b_k} = &-\dot{\Phi}_{\underline{q}}\, \underline{q}_{b_j b_k} - \dot{\underline{\Phi}}_{b_j b_k} - \dot{\Phi}_{\underline{q} b_k}\, \underline{q}_{b_j} - \Phi_{\underline{q} b_k}\, \dot{\underline{q}}_{b_j} - \left[\dot{\Phi}^{i}_{\underline{q}}\, \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\, \underline{q}_{b_k} \\ &- \left[\Phi^{i}_{\underline{q}}\, \tilde{\dot{\underline{q}}}_{b_j}\right]_{\underline{q}}\, \underline{q}_{b_k} - \left[\Phi^{i}_{\underline{q}}\, \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\, \dot{\underline{q}}_{b_k} - \dot{\Phi}_{b_j \underline{q}}\, \underline{q}_{b_k} - \Phi_{b_j \underline{q}}\, \dot{\underline{q}}_{b_k}\end{aligned} \tag{4.8}$$

Taking the total derivative of Equation 4.8 with respect to time and transposing some terms to the right hand side, we get

$$\begin{aligned}\Phi_{\underline{q}}\, \ddot{\underline{q}}_{b_j b_k} = &-\Phi_{\underline{q} b_k}\, \ddot{\underline{q}}_{b_j} - \left[\Phi^{i}_{\underline{q}}\, \tilde{\ddot{\underline{q}}}_{b_j}\right]_{\underline{q}}\, \underline{q}_{b_k} - 2\,\dot{\Phi}_{\underline{q} b_k}\, \dot{\underline{q}}_{b_j} - 2\left[\dot{\Phi}^{i}_{\underline{q}}\, \tilde{\dot{\underline{q}}}_{b_j}\right]_{\underline{q}}\, \underline{q}_{b_k} \\ &- 2\left[\Phi^{i}_{\underline{q}}\, \tilde{\dot{\underline{q}}}_{b_j}\right]_{\underline{q}}\, \dot{\underline{q}}_{b_k} - 2\,\dot{\Phi}_{\underline{q}}\, \dot{\underline{q}}_{b_j b_k} - \Phi_{\underline{q} b_k}\, \ddot{\underline{q}}_{b_j} - \left[\ddot{\Phi}^{i}_{\underline{q}}\, \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\, \underline{q}_{b_k} \\ &- \ddot{\Phi}_{\underline{q}}\, \underline{q}_{b_k b_j} - 2\left[\dot{\Phi}^{i}_{\underline{q}}\, \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\, \dot{\underline{q}}_{b_k} - \left[\Phi^{i}_{\underline{q}}\, \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\, \ddot{\underline{q}}_{b_k} - \ddot{\underline{\Phi}}_{b_j b_k} \\ &- \Phi_{b_j \underline{q}}\, \ddot{\underline{q}}_{b_k} - 2\,\dot{\Phi}_{b_j \underline{q}}\, \dot{\underline{q}}_{b_k} - \ddot{\underline{\Phi}}_{b_j \underline{q}} \underline{q}_{b_k}\end{aligned} \tag{4.9}$$

Equation 4.9 was obtained by twice differentiating the equations of kinematic constraint of Equation 2.1 with respect to design and then taking the total time derivative of the result two times. In order to verify the above result, the same equation was derived with a different order of differentiation, with the time derivatives being taken before the derivatives with respect to design. The alternate derivation, which yielded the same result as Equation 4.9 is presented in Appendix I.

Equations 4.6 and 4.9 can be combined and written in matrix form as

$$\begin{bmatrix} M & \Phi_{\underline{q}}^T \\ \Phi_{\underline{q}} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\underline{q}}_{b_j b_k} \\ -\underline{\lambda}_{b_j b_k} \end{bmatrix} = \begin{bmatrix} \text{Right hand side of Equation 4.6} \\ \text{Right hand side of Equation 4.9} \end{bmatrix} \qquad (4.10)$$

Equation 4.10 represents a combined system of equations for the second order design sensitivity of the state variables. These equations, along with similar equations for first order sensitivity, can be integrated simultaneously with the system equations of motion and kinematic constraint to give the dynamic response as well as first and second order design sensitivities. The method used for solving these equations is discussed in Chapter 5.

4.2 Comparison with the Adjoint Variable Method

A general method for computing first and second order design sensitivity by direct differentiation was presented in Chapter 3 and in the preceding section. An alternate method for performing design

sensitivity analysis for this class of problems using an adjoint variable technique is described in Reference 8. Although a detailed comparative study of the two methods is needed to establish their relative merits and demerits, it is possible to make the following general remarks on some important aspects, based purely on the theory on which the methods are founded:

a) Computational Effort: In the adjoint variable method, the number of differential equations that will have to be solved for obtaining first order design sensitivity is proportional to the number of active constraints, while in the case of the direct differentiation method, it is proportional to the number of design variables. For second order design sensitivity by the adjoint variable method, the number of differential equations to be solved is proportional to the square of the number of active constraints, while in the case of direct differentiation, the number of equations is proportional to the square of the number of design variables. In general, one could expect the number of design variables to be greater than the number of active performance constraints. Thus, the computational effort could be expected to be greater in the direct differentiation method. However, a part of the additional computation required by the direct differentiation method is likely to be offset by the advantages it enjoys over the adjoint variable method in a supercomputer implementation. Specific advantages are brought out in some of the points that follow.

b) Memory Requirement: In view of the larger number of differential equations that may be expected to be solved in the direct differentiation method, memory requirements for this method would be greater than for the adjoint variable method. On a supercomputer, the requirement of additional memory is not likely to significantly increase the overall cost of computation.

c) Error Control: The essence of any numerical technique is its ability to maintain control over the errors inherent in the method. In the absence of effective error control, numerical errors can increase rapidly to the point where the very results of the computation could be rendered meaningless. In the direct differentiation method, the state and sensitivity equations are integrated forward in time simultaneously. Therefore, by using a numerical algorithm with automatic error control, it is possible to control the error in both state and sensitivity directly.

In the adjoint variable method, the integration for state proceeds forward in time, whereas the integration for adjoint variables must be done backward in time. It is difficult to estimate how the errors in the forward integration will affect the solution during the backward integration. Further, the adjoint variable method needs to store state variables on disk during the forward integration, since this information is required during the backward integration and solution for the adjoint variables. This storing of information during the forward integration is possible only at discrete intervals of time. During backward integration, if values of state variables are

required at some intermediate point for which values were not stored, numerical interpolation is employed to generate the required values. This leads to additional computation and introduces a probable source of error, the effect of which is difficult to estimate. These considerations indicate that error control in the direct differentiation method is more positive and reliable than in the adjoint variable method.

There is another factor which favours the direct differentiation method over the adjoint variable method in terms of accuracy. In determining the design sensitivity of cost and constraint functions by direct differentiation, the only unintegrated quantities that come into play are the first and second order design sensitivities of accelerations and Lagrange multipliers. The design sensitivity of velocity is an integrated quantity and the design sensitivity of position is a twice integrated quantity. In the adjoint variable method, the sensitivity almost always depends on a non-integrated adjoint variable. Since the process of integration tends to reduce the effect of localized and random errors, the direct differentiation method will probably be less sensitive to these errors than the adjoint variable method.

d) Input/Output Requirements: The adjoint variable method requires a large number of intermediate input/output operations which could be a disadvantage in a supercomputer implementation. Supercomputers are designed for intense uninterrupted computation and frequent interrupts for input/output operations are likely to

adversely affect the running time and also the processor time due to input/output overhead. The time spent in the actual input/output operations could come down considerably if the supercomputer supports a solid-state storage device (SSD) which is usually a bank of high-speed semiconductor memory configured to function like disk storage. Thus, for the problem of computing second order design sensitivity, which has to be solved on a supercomputer because of its computational intensity, the direct differentiation method appears to be the better choice.

e) Vectorization: Vectorization is the technique by which a supercomputer does similar calculations on the components of an array variable (vector) simultaneously in the multiple processing units of the machine. The direct differentiation method is highly matrix oriented and is capable of utilizing the vectorizing capabilities of a supercomputer better than the adjoint variable method. The input/output operations and operations like numerical interpolation which are specific to the adjoint variable method are not likely to vectorize.

f) Interpretability of Intermediate Variables: The intermediate variables computed in the direct differentiation method are the first and second order design sensitivities of position, velocity, acceleration and Lagrange multipliers. These quantities have obvious physical interpretations and can be understood quite easily by the designer. Such an understanding can be of help in understanding the system behavior and in checking results. It may even be possible

to make design decisions based on the information contained in these variables. Adjoint variables on the other hand are abstract mathematical quantities with no clearly defined physical significance. Consequently, they are almost impossible to interpret and are of no practical help to the designer.

g) Formulation of Pointwise Constraints: It was pointed out in Chapter 3 that a pointwise constraint may be formulated as an equivalent integral constraint or as a grid-point constraint. The choice between the two formulations would be dictated by the method used in design sensitivity analysis. This is on account of the fact that the integral constraint formulation results in one integral constraint for each pointwise constraint, while the grid-point constraint formulation may replace a single pointwise constraint by several grid-point constraints. If the adjoint variable method is used in design sensitivity analysis, the integral constraint formulation would be used, since the computational effort in this method is proportional to the number of active constraints. On the other hand, if the direct differentiation method is used, the grid-point formulation would be preferred since adding more constraints does not increase the computation load in this method.

CHAPTER V

IMPLEMENTATION

The theoretical basis and governing equations for dynamic analysis as well as first and second order design sensitivity analyses were developed in the last three chapters. The systems of equations derived for dynamic analysis, first and second order design sensitivity analyses are obviously too complex to be solved analytically. It is therefore necessary to devise a suitable numerical technique to solve these equations. However, before any numerical technique can be applied, it is necessary to formulate the equations governing the dynamic response and design sensitivity of the particular system being investigated.

Several approaches are possible in the formulation of the equations governing response and sensistivity. The simplest approach would be to build up expressions for kinematic constraints, cost/constraint functions, etc. and work out by hand the coefficient matrices and right hand sides for the governing equations, devise a numerical solution scheme and code these into a computer program. Such an approach would work for any given system but would necessitate rewriting the entire program for analyzing a different system.

A better approach would be to separate the computation required into two parts; a fixed part consisting of subprograms to solve the

equations governing the system response and design sensitivity, and a problem-dependent part consisting of subprograms to formulate the expressions for the governing equations of the particular system under investigation. By designing the fixed solution subprograms to be driven by external inputs like number of equations, number of unknowns, size of coefficient matrix, etc. , this portion of the software can be kept reusable. The problem-dependent subprograms would then have to be rewritten for each new system that is investigated. While this seems to be an acceptable proposition, a closer look will reveal the enormous amount of work this involves. For dynamic analysis and first order design sensitivity, this involves writing about thirty-two subroutines. At least fifty to sixty more subroutines will be required to program the terms required for second order design sensitivity analysis. This estimate may seem excessive at first, but considering the fact that the expressions for second derivatives of integral and grid-point constraints have twenty-nine terms each on the right hand side and that the system of equations for second order design sensitivity has thirty-eight terms on the right hand side, the estimates are in fact quite conservative. Further, the fact that these expressions are to be coded into subroutines implies that they must at first be worked out by hand. The probability of error is quite high in such an exercise, since the work of developing the governing equations involves a good deal of partial differentiation as well as addition and multiplication of vectors and matrices of algebraic expressions. The programming of such a solution scheme is bound to be very tedious

and precludes the possibility of making the scheme general-purpose or capable of handling a wide range of problems. However, all these problems are eliminated if the task of carrying out the symbolic manipulations related to the formulation of the governing equations is also done on a computer, as described in the following section.

5.1 Symbolic Computing in Design Sensitivity Analysis

Computers were at one time thought to be capable of handling only numerical data with ease. However, the discovery that they could also handle character data with equal facility led to the realization that if they could be programmed to do arithmetic, they should be capable of doing algebra as well. The area of computer science that deals with the computer's symbolic manipulation abilities is generally known as "symbolic computing". Various symbolic manipulation languages have been developed in the last three decades. One of the best known of these languages is REDUCE-2, which was developed in the early seventies by A.C.Hearn for "general algebraic computations of interest to mathematicians, physicists and engineers" [11].

REDUCE is a programming system for carrying out algebraic operations accurately, no matter how complicated the expressions become. It can manipulate polynomials in a variety of forms, both expanding and factoring them, and extracting various parts of them as required. REDUCE can also handle vectors and matrices of algebraic expressions and has features built into it for performing matrix operations like multiplication, inversion, transposition, etc. REDUCE can also do differentiation and integration of algebraic expressions.

User defined procedures, functions and operators are also permitted. REDUCE has the capability to both read in and write out files containing algebraic expressions. It also has all the usual control structures like IF-THEN-ELSE blocks and FOR, WHILE and REPEAT loops, which are considered essential in any procedural language. In fact, there is nothing unusual about REDUCE except the fact that it is designed to primarily deal with symbols instead of numbers. Thus, the reduce statements

```
X:=A+B;
Y:=X*X;
```

would assign the value $A^2 + 2*A*B + B^2$ to Y, where A, B, X and Y are arbitrary user defined symbols which may or may not be assigned numerical values within the REDUCE program in which they occur. Similarly the statement

```
Z:=DF(Y,A)
```

following the two earlier statements would assign the value 2*A + 2*B to Z, i.e, Z is equated to the partial derivative of Y with respect to A. A complete explanation of all REDUCE commands and syntax is given in Reference 11.

Although REDUCE is capable of performing arithmetic operations using numerical data, it is very slow in these operations when compared to a language like FORTRAN. Therefore, it is generally advisable to use REDUCE for applications which require mainly algebraic manipulation and switch over to a language like FORTRAN whenever there is a lot of numeric computation to be done. In order to

facilitate such a change, REDUCE has a built-in feature that enables it to write out the results of the algebraic computations in the form of FORTRAN program statements. These statements can then be compiled and linked with other FORTRAN program segments for execution.

It may not always be possible to separate a given algorithm into two distinct parts, one involving mostly algebraic manipulations and the other involving mostly numerical calculations. Fortunately, the method presented earlier for design sensitivity analysis of dynamic systems has a natural division into two such parts. The first step of formulating the equations governing the dynamic response and design sensitivity is ideally suited for symbolic computing. Accordingly a REDUCE program was developed to generate all the necessary problem-dependent information in the form of FORTRAN subroutines, which were then linked with problem-independent FORTRAN subroutines that carried out the iterative numerical calculations. The REDUCE program may, therefore, be thought of as a preprocessor for the input data for a particular problem. Details of how this preprocessor was built are given in the following section.

5.2 The REDUCE Preprocessor

In order to understand the functioning of the preprocessor, it is first necessary to understand the required input data. This input data must be in the form of REDUCE statements stored in two input files. The first input file contains information required by the preprocessor to set up internal arrays of the required size. The second input file

gives the expressions that are used to formulate the terms in the equations governing dynamic response and sensitivity.

The first input file consists of information about the number of bodies in the system, number of grounded revolute joints, number of non-grounded revolute joints, number of grounded translational joints, number of non-grounded translational joints, number of design variables, number of degrees of freedom of the system, number of non-standard kinematic constraints, number of performance constraints of each type (integral, grid-point and design-dependent), body number of grounded body, etc. The second input file contains information such as the masses and moments of inertia of the various bodies, expressions for cost/constraint functions, information describing each joint in the system, expressions for non-standard forces and kinematic constraints, etc.

Once the first input file is read in, the preprocessor can ascertain the size of the problem and assign arrays of required size. Then the second input file is read in and the evaluation of the terms that go into the governing equation for system response and design sensitivity begins.

The preprocessor first constructs the equations of kinematic constraint of Equation 2.1 by building up two equations for each revolute or translational joint and taking in the non-standard kinematic constraints exactly as fed in. If a grounded body is indicated, then three additional constraint equations, restricting the

X, Y and ϕ coordinates of the grounded body to fixed values are added to the vector of equations of kinematic constraint.

The exact sequence of operations followed by the preprocessor is too long to be described in stepwise fashion and is, therefore, descibed very briefly. First, the vector of kinematic constraints is differentiated with respect to the position vector to yield the Jacobian matrix $\Phi_{\underline{q}}$. The Jacobian matrix is then differentiated with respect to time twice to yield the $\dot{\Phi}_{\underline{q}}$ and $\ddot{\Phi}_{\underline{q}}$ matrices. A similar sequence of operations with the vector of kinematic constraints and the vector of design variables leads to the quantities $\Phi_{\underline{b}}$, $\dot{\Phi}_{\underline{b}}$ and $\ddot{\Phi}_{\underline{b}}$. These quantities are what may be considered as the building blocks for terms relevant to dynamic analysis and first order design sensitivity analysis. Compound terms like $(\Phi_{\underline{b}}\ \dot{\underline{q}})_{\underline{q}}\ \dot{\underline{q}}$, $(M\ \tilde{\underline{q}})_{\underline{b}}$, $(\Phi_{\underline{q}}^{T}\ \tilde{\underline{\lambda}})_{\underline{b}}$ and $(\Phi_{\underline{q}}^{T}\ \tilde{\underline{\lambda}})_{\underline{q}}$ which are required for dynamic analysis and first order design sensitivity analysis are then evaluated by suitable multiplication and partial differentiation.

The next part of the preprocessor computes all the derivatives of the performance constraints required for first order design sensitivity analysis. For purely design dependent constraints, only the derivative with respect to design is calculated. For the other two types of constraints, derivatives with respect to position, velocity, acceleration and Lagrange multiplier are also calculated. Thirty-two FORTRAN subroutines are then written out to cover the terms that occur

in the governing equations for dynamic analysis and first order design sensitivity analysis.

Compound terms like $(M\,\ddot{\underline{q}})_{\underline{b}}$, $(\Phi_{\underline{q}}^{T}\,\tilde{\underline{\lambda}})_{\underline{b}}$ and $(\Phi_{\underline{q}}^{T}\,\tilde{\underline{\lambda}})_{\underline{q}}$ are the building blocks for the terms related to second order design sensitivity. In dynamic analysis and first order design sensitivity analysis, all the terms that are needed were first calculated and then the FORTRAN subroutines were written out enbloc. The approach in the case of the second order sensitivity terms is slightly different for the following reason. All the terms in second order sensitivity analysis are third order tensors and are likely to occupy a lot of space if all of them are simultaneously stored in memory before being written out in suitable FORTRAN subroutines. Fortunately, REDUCE permits the user to declare space for a new array and clear up the space for an array no longer required, all in the midst of ongoing computation. Therefore, in obtaining the expressions for the second order related terms, the terms are calculated individually or in some cases in pairs when the terms are almost the same but differ in only one multiplicand, and the space occupied by each term is cleared immediately after its FORTRAN subroutine is written out. As far as possible intermediate quantities required repeatedly are kept in work arrays and recalculation of terms is avoided.

The third order quantities involved in these computations can be pictured as square prisms. However, only half of each square prism needs to be calculated since the terms involved are symmetric by themselves or have other complementary quantities such that their sums

are symmetric. This is on account of the fact that the second order sensitivies, which are derivatives with respect to b_j and b_k, are themselves symmetric and hence the right hand sides of the linear equations from which these quantities are calculated must also be symmetric. This symmetry is also used in the numeric solution to cut the amount of computation by almost half. It is also worth noting that the coefficient matrix is the same for dynamic analysis, first and second order design sensitivity analyses and hence, these terms need to be calculated only once. Finally, the terms related to second order sensitivity of performance constraints are calculated and written out in the form of FORTRAN subroutines. A total of forty FORTRAN subroutines are written out for the terms related to second order design sensitivity analysis. A simple example showing the kind of terms evaluated by the preprocessor for a particular constraint is given in Appendix II.

5.3 Advantages of Symbolic Computing

Although symbolic computing has been used successfully in several applications in mathematics and physics for many years, its use in the area of analysis and design of dynamic systems has been quite recent [5]. The preprocessor developed in the present work has demonstrated the feasibility of using symbolic computing in this area for highr order sensitivity analysis. In fact, it would not be too much of an exaggeration to say that the solution of the second order design sensitivity problem using the direct differentiation method

would be practically impossible without preprocessor. The use of symbolic computing in the analysis of the dynamic response and design sensitivity of constrained dynamic systems affords the following benefits:

1. Simplified treatment of non-standard forces and kinematic constraints: Non-standard forces and constraints are simply entered through the data input to the preprocessor. All the required derivatives of these quantities are then calculated by the preprocessor so that no further action is required. In contrast to this, in an all-FORTRAN implementation, the user may have to write upto a dozen subroutines for each non-standard constraint or force.

2) Simplified treatment of performance constraints: Performance constraints can vary widely and cannot usually be standardized. Therefore, in most implementations that do not use symbolic computing, these have to be input as separate subroutines. Subroutines are also needed for the derivatives of these constraints, which is not necessary if the preprocessor is used.

3) Streamlining of the program: The formulation of the state and sensitivity equations is basically an algebraic operation that can be coded quite naturally and elegantly in REDUCE or any other symbolic manipulation language. This formulation can be coded in FORTRAN too, but the implementation becomes much more difficult and complex. In order to use FORTRAN, a standard set of possible design variables must be assumed, and any non-standard design

variable would necessitate separate user-written subroutines. In the REDUCE implementation, there is no need for predefining any choice of design variables and the user is not required to write even a single subroutine.

4) Easier program development: The capabilities of the design sensitivity program can be enhanced by introducing new types of forces, joints, etc. Such changes are more easily absorbed in an implementation using symbolic computation.

5) Reliability of software: Most of the errors that occur in the development and use of software arise from mistakes made either by the programmer or the user. By streamlining the program flow and simplifying the process of program development, the use of symbolic computing considerably reduces the chances of programmer error. Further, by minimizing the amount of input required from the user and simplifying even the little input that is required, the possibility of user error is also kept low. These factors add a great deal of reliability to the software.

6) Efficiency: The FORTRAN subroutines that are generated by the REDUCE preprocessor are problem-dependent routines tailor-made for each problem that is being solved. Consequently they will be much more efficient than equivalent general purpose routines, as a lot of conditional logic (IF..the joint is revolute.. etc.) and branching will be avoided. This is particularly important in a supercomputer implementation since run-through code executes much faster than code that has to be executed non-sequentially.

Symbolic computing also allows a lot of source code editing which can be used to advantage in a supercomputer implementation. This point is discussed in the section dealing with the details of the supercomputer implementation.

In most problem solving situations, there is usually a trade-off between efficiency and generality of the solution - the greater the generality, the lower is the efficiency and vice-versa. This, however, does not apply when the REDUCE preprocessor is used instead of an equivalent FORTRAN program. The superior handling of non-standard forces and constraints by the REDUCE preprocessor makes the REDUCE based software more general and yet it performs more efficiently. Thus both generality and efficiency are simultaneously enhanced through the use of symbolic computing.

The next step after formulating expressions for the terms in the equations governing the response and sensitivity is to solve these equations using a suitable numerical technique. The techniques for solving the equations governing dynamic response and design sensitivity are discussed in the next two sections. This is followed by a discussion of some of the considerations that go into the choice of a numerical integration algorithm and details of some features built into the software specially for the supercomputer implementation. A detailed stepwise algorithm for obtaining the

dynamic response and design sensitivity is also presented at the end of the Chapter.

5.4 Solution of System Equations of Motion

Several solution schemes have been presented in the literature for integrating the equations of motion. Generally, the methods either involve integrating all components of the system generalized acceleration vector or integrating only a minimal set. Methods that integrate all the components of the generalized acceleration vector impose the kinematic constraints in the Gaussian form of Equation 2.29. Methods that integrate only some of the components of the generalized acceleration vector use some technique for identifying the particular components that can be integrated and impose the kinematic constraints in the algebraic form of Equation 2.1. Features of some of the methods mentioned in the literature are briefly mentioned in the following paragraphs. Following this, the method used in the present work for integrating the equations of motion is presented in detail.

The dynamic analysis program ADAMS [12] follows the first of the two approaches mentioned above. All components of the acceleration vector are integrated to obtain the position and the velocity vectors. Although this method has the virtue of simplicity, it involves integrating more equations than what is strictly necessary. Further, as the method does not impose the equations of kinematic constraint except in the Gaussian form of Equation 2.29, integration errors lead

to solutions for position and velocity that do not satisfy the kinematic constraints. These constraint violations become worse as the simulation time increases.

Baumgarte has suggested a variation of the above method [13] in which two correction terms are added to the Gaussian form of the kinematic constraint equation. The introduction of these terms practically eliminates the growth of constraint violations. Although this method also calls for the integration of more differential equations, a part of this additional computation is offset by the fact that the method eliminates the need to solve for the equations of kinematic constraint in the algebraic form. This method has been found to be quite acceptable for analyzing the dynamic response of mechanical systems.

Generalized coordinate partitioning, which is discussed in detail in Reference 3, is a technique that makes use of the idea that all generalized coordinates are not independent. They are in fact related by the kinematic constraint equations of Equation 2.1 and can be partitioned into dependent and independent generalized coordinates. Suppose we have N generalized coordinates and M constraint equations for a particular dynamic system, we then have only (N - M) independent generalized coordinates, which is also the same as the number of independent degrees of freedom of the system. To solve for the system response, therefore, it is necessary to integrate the equations of motion of only (N - M) generalized coordinates. The remaining M generalized coordinates can be calculated from Equation 2.1 once the

values of the independent coordinates are known. Since the constraint equations are non-linear, an iterative process - usually a Newton-Raphson iteration - must be used to obtain the solution. The dependent velocities can then be directly calculated from the linear equations of Equation 2.28, once the independent velocities are known; and finally, all the components of generalized acceleration as well as Lagrange multipliers can be calculated from the system of equations of Equation 2.30.

The generalized coordinate partitioning approach offers a considerable reduction in the number of differential equations to be solved. For a simple four-bar linkage, this method requires the solution of one second order differential equation as compared to twelve for the two methods described earlier. Although some of this saving in effort is lost in doing the Newton-Raphson iteration and in selection of independent coordinates, the generalized coordinate partitioning technique on the whole performs very efficiently and reliably. This method is, therefore, chosen as the basis for solving the state and sensitivity equations.

In order to ensure satisfactory performance of this method, it is necessary to have:

1. An effective method for determining the set of independent coordinates;
2. A sufficiently accurate predictor for the dependent variables so that the Newton-Raphson iteration has a good initial estimate.

A good choice of the set of independent coordinates is critical to the performance of the solution algorithm. A poor choice could lead to the divergence of the Newton-Raphson iteration or cause numerical errors in the solution to increase very rapidly. There are no clear-cut guidelines for choosing the independent coordinates. A strategy that has been found to work quite well is to choose the independent coordinates such that the integration errors in the calculation of the independent coordinates do not get magnified when the dependent coordinates are calculated from the independent ones.

A method for determining the set of independent coordinates based on a full row and column pivoting is presented in Reference 3. It is shown that the independent set chosen by this method does prevent the integration errors from getting amplified in the calculation of the dependent variables. However, this selection algorithm involves a lot of computation and hence the alternative approach of Reference 5, which is described in the following paragraphs, is chosen as the method for implementing generalized coordinate partitioning in the present work.

Consider the state of the system at time t. The solution for the state must now be advanced to time $t+\Delta t$. It is required to find a set of independent generalized coordinates that will be a reasonable choice over the entire interval $[t,t+\Delta t]$. In order to do this, the expected average velocities, $\dot{q}_{av}$, over the interval are estimated from the generalized velocities and accelerations at time t using the relationship

$$\dot{q}_{av} = \dot{q}(t) + (1/2)\,\ddot{q}(t)\,\Delta t \qquad (5.1)$$

The coordinates corresponding to the highest estimated average velocities are then chosen to be the independent ones.

Errors in the independent coordinates may be interpreted as perturbations from their true positions. Since the independent coordinates are those that have the highest estimated average velocities over the interval $[t,t+\Delta t]$, the velocity ratio of any dependent variable to an independent variable is less than unity. This implies that for any small perturbation of the independent coordinates, the resulting perturbation in the dependent coordinates will be smaller in magnitude, thus tending to damp out the effects of errors in calculating the independent coordinates during calculations for the dependent coordinates. The computing effort required for this selection scheme is far less than that required for the Gaussian elimination and is almost negligible when compared to the overall effort required for solving the system equations.

In addition to integrating for the independent coordinates to advance the solution from time t to time $t+\Delta t$, it is also necessary to predict the values of the dependent coordinates at time $t+\Delta t$. This is necessary in order to have a reasonable initial guess for the Newton-Raphson iteration and thus guard against divergence. The method used in Reference 3 uses a sophisticated variable-order polynomial predictor based on the time history of the generalized coordinates to calculate these initial estimates. Although this method yields very accurate estimates, it is not clear that the accuracy of the

predictions is worth the effort of additional storage and computation. Therefore, the simpler quadratic predictor of Reference 5 is used in the present work. The dependent coordinate value at time t+Δt is obtained using the relationship

$$q(t+\Delta t) = q(t) + \dot{q}(t)\,\Delta t + (1/2)\,\ddot{q}(t)\,(\Delta t)^2 \qquad (5.2)$$

This predictor can always be made to yield as high a degree of accuracy as desired by choosing Δt sufficiently small. This predictor worked quite efficiently and accurately in all the example problems presented in the next chapter. It is possible that in the case of exceptionally non-linear systems, the method may require such small time steps that it becomes prohibitively expensive. In such cases, it would be advisable to switch to a higher order predictor. Such a situation was, however, not encountered in any of the examples that were solved using this method.

5.5 Solution of State Sensitivity Equations

The similarities between the first and second order sensitivity equations of Equations 3.27 and 4.10, and the system equations of motion of Equation 2.30 suggest that the same techniques that were used for integrating the system equations of motion could be applied to integrate the equations for first and second order design sensitivity as well. The same idea of integrating only for the independent quantities and using these to solve for the dependent quantities algebraically can be directly applied. Only the rows of $\underline{q}_b$

and $\dot{q}_b$ corresponding to the independent generalized coordinates need to be obtained by integration. Once these are known, the dependent rows of q_b and $\dot{q}_b$ can be calculated from Equations 3.24 and 3.25 respectively. Unlike the constraint equations of Equation 2.1, these equations are linear equations and can be solved directly. Further, as the Jacobian matrix Φ_q is once again the coefficient matrix in these equations, and it has already been factored while solving for dependent velocities, the only additional effort required to compute q_b and $\dot{q}_b$ is in doing the forward-and-back substitution. The choice of independent coordinates has already been made earlier and need not be repeated. The fact that these equations are linear makes the calculation of dependent rows of q_b and $\dot{q}_b$ quite simple.

The same idea can be used to evaluate the second order design sensitivities of dependent coordinates. Only about half the terms in $q_{b_j b_k}$ and $\dot{q}_{b_j b_k}$ corresponding to the independent coordinates need to be obtained by integration. This is on account of the fact that the terms $q_{b_j b_k}$ and $\dot{q}_{b_j b_k}$ are symmetric. As in the case of first order sensitivity, the values of $q_{b_j b_k}$ and $\dot{q}_{b_j b_k}$ corresponding to the dependent coordinates can be calculated by solving the linear systems of equations of Equations 4.7 and 4.8 respectively. Once again, the work of solving these equations involves only forward-and-back

substitutions since the coefficient matrix, $\Phi_{\underline{q}}$, is already available in factored form. In fact, a major portion of the work in solving for first and second order design sensitivities of dependent coordinates is in assembling the right hand sides, since this involves a lot of matrix products which are costly to evaluate.

Finally $\ddot{\underline{q}}_{\underline{b}}$, $\underline{\lambda}_{\underline{b}}$, $\ddot{\underline{q}}_{b_j b_k}$ and $\underline{\lambda}_{b_j b_k}$ can be evaluated from Equations 3.27 and 4.10. The coefficient matrix of these equations is the same as the coefficient matrix of the combined equations of motion and kinematic constraint of Equation 2.30. This coefficient matrix was factored while solving for accelerations and Lagrange multipliers and hence the only work required to solve for first and second order sensitivities of accelerations and Lagrange multipliers is the work involved in performing forward-and-back substitution. However, in this case the computational effort required to assemble the right hand sides is quite extensive on account of the matrix multiplications involved.

5.6 Selection of Numerical Integration Algorithm

The discussion in the preceding sections dealt with methods for setting up the systems of differential equations to be integrated and the methods for performing the function evaluations for the derivatives to be supplied to the integration algorithm. The actual integration algorithm itself was not mentioned. This is because several well-developed numerical integration algorithms are already in

existence and many of these have been programmed into efficient, reliable computer programs. Therefore, it only remains to choose an appropriate program from among the existing ones. The choice must be based on the special characteristics of the systems of differential equations to be solved.

In the present case, function evaluation for the derivatives is quite expensive. Therefore, single step methods such as Runge-Kutta formulas are not preferred because these require many more function evaluations than comparable multi-step methods. The equations to be solved are generally expected to be non-stiff and hence stiff differential equation solvers like Gear's algorithm are not necessary. From these considerations, it appears that a multi-step method designed for non-stiff systems of non-linear ordinary differential equations must be chosen. Accordingly, the package DE/STEP/INTRP based on the method devised by Shampine and Gordon [14] was selected. This is a variable order, variable step size, multi-step explicit/implicit predictor/corrector method with automatic error control. The order of integration and step size are automatically selected by the program and varied whenever necessary, to achieve the specified error tolerance. This is important because the behavior of the system and its sensitivity can change drastically within the simulation period. A fixed order or fixed step size method could lead to ineffective error control or inefficient computation. A complete discussion of the theory and implementation of the method is given in Reference 14.

In addition to integrating the state and sensitivity, it is also necessary to carry out integrations to compute the values of integral functional constraints and their first and second order design sensitivities. The DE/STEP/INTRP package is used for performing these integrations as well.

5.7 Supercomputer Implementation

Supercomputers are characterized by their extremely fast hardware and special architecture which makes them much faster than mainframe computers. They usually have more than one central processing unit (CPU) and the programs running on these machines take advantage of the availability of multiple CPUs. Vectorization is one of the ways by which a supercomputer speeds up the execution of programs. The CRAY X-MP/48 has eight vector processing registers and high-speed vector and floating-point functional units which can be driven by special vector instructions. Vectorization is a technique by which an iterative piece of code like a DO loop is executed in parallel in the multiple processing units of the supercomputer. All DO loops cannot be vectorized. This is on account of the fact that a subsequent pass through the loop may rely on the result of a previous pass. Such a condition is referred to as a vector dependency. In such a case, parallel execution would produce the wrong result. Normally DO loops on the CRAY are broken into 64-at-a-time blocks for vectorization. Therefore, vector dependencies resulting from backward references to array items which are farther away than the reach of the vectorized

instruction can be ignored. Although, the decision to vectorize a DO loop is usually taken by the compiler but in certain cases, the programmer can decide to force vecrization if the absence of vector dependency can be ascertained. Vectorized code can run 3 to 6 times faster than equivalent non-vectorized (scalar) code.

In the software developed for second order design sensitivity analysis, there are two areas where it is possible to build in features that make use of the special capabilities of a supercomputer: in the preprocessor written in the symbolic computing language REDUCE-2 and in the general-purpose solution routine written in FORTRAN. These features are discussed in the following paragraphs.

The ability of REDUCE to perform logic checks has been used to the fullest extent possible for doing effective source code editing of the problem-dependent FORTRAN subroutines written out by the preprocessor. The Jacobian matrix $\Phi_{\underline{q}}$, from which most of the terms related to response and sensitivity analysis arise is expected to be sparse. Therefore, the preprocessor checks to see that any term for which FORTRAN code is generated is a non-zero term, thus avoiding many lines of unnecessary code.

The version of FORTRAN available on the CRAY X-MP/48 has a facility for declaring a DO loop as a short DO loop. A short DO loop is one that will be executed at least once and not more than sixty-four times. The CFT compiler generates special vectorizing object code when it encounters a short loop directive. The REDUCE preprocessor makes use of this facility. Whenever it writes a DO loop,

it checks to see if the particular loop can be declared to be a short loop and adds an extra line of compiler directive when the condition is met. The number of times that a loop is executed in the problem-independent routines depends on the size of the problem and cannot be known beforehand. Since the short loop directive must be in the source code before it is compiled, (a wrongly used short loop directive leads to unpredictable results) the advantage of this directive is available only in the routines generated by the preprocessor. The preprocessor also linearizes DO loops involving multi-dimensional arrays. The concept of linearizing is explained later in this section.

The preprocessor also helps to save code in the problem-independent routines. Some of the terms written out by the preprocessor are required as negatives of the expressions that would result from normal algebraic manipulations. Since most of these quantities are array variables, it would require a DO loop to flip the sign on these quantities. This additional code is saved by having the preprocessor write out a "-(" before the expression and a ")" after the expression, which would make the particular FORTRAN statement evaluate the term with the required negative sign.

The CFT compiler allows the declaration of POINTER variables. A pointer is merely an integer variable which is associated with a particular array and bears the address of the first item in the array. The array associated with a pointer variable can be a dummy array, the size and shape of which can be changed during execution by changing

the contents of its declared subscript variables. It is also possible to change the value contained in the pointer variable itself which is equivalent to changing the address of the array. The entire responsibility of managing pointer variables is left to the programmer who has to ensure that the pointer is pointing to a valid location in memory and that the information retrieved from arrays is based on their correct sizes.

The solution scheme implemented on the CRAY makes full use of pointer variables. All the storage space required for carrying out dynamic analysis as well as first and second order design sensitivity analyses are contained in two large arrays, one for integer variables and one for real variables. These two arrays are partitioned into suitable sized sub-arrays to hold the data items relevant to the particular problem being solved. The partitioning is done by using pointer declarations. The variables that indicate the size of an array accessed via a pointer are carried in COMMON blocks. The pointer declarations are entirely in terms of these common block variables. Thus, by having the same pointer and common block declarations in all the subroutines, it is possible to access the entire data base from any subroutine. This eliminates the need for subroutine call lists which can be quite lengthy and difficult to maintain during program modifications. The same pointer and common block routines are added to each subroutine in the both problem-dependent and problem-independent code by means of a VAX/TPU (text processing utility) program which adds on the lines in the front end VAX/8650 computer before the

program is sent for compilation on the CRAY. The fact that only a single copy of the pointer and common block declarations is maintained assures the integrity of this information and makes modification easy.

In addition to allowing the programmer greater control over array addressing and dimensioning, pointer variables are also a means of writing more easily vectorizable code. One of the ways in which vectorizing is promoted is by linearizing multi-dimensional arrays. Consider for example an array A of size M by N which is to be zero filled (say). The normal FORTRAN code for doing this might look like

```
      DO 10 I = 1,M
         DO 10 J = 1,N
10          A(I,J) = 0.0
```

While the above code would work on a CRAY, only the inner DO loop will vectorize. To get the code to vectorize better, the two-dimensional array A would have to be addressed as a single-dimensioned array using a pointer variable. Let a pointer variable IPOINT be declared to be associated with a dummy array named WORK by the statement.

```
      POINTER (IPOINT,WORK(1))
```

Then the code to zero fill the array can be written as

```
      IPOINT = LOC(A(1,1))
         DO 10 I = 1, N*M
10       WORK(I) = 0.0
```

The first statement assigns the address of A(1,1) to IPOINT and the fully vectorizable DO loop zero fills N * M times starting with

A(1,1). The above is an example of a multi-dimensioned array linearized with the help of a pointer variable.

Pointer variables are also used to save the amount of storage space needed to store the right hand sides of the second order design sensitivity equations. As already mentioned, owing to the symmetry of second order sensitivities, only about half the number of actual terms need be calculated. For example, if NY is the number of generalized coordinates and ND is the number of design variables, then the number of terms in the third order tensor $q_{b_j b_k}$ is NY * ND * ND. But for all values of j and k, $q_{b_j b_k}$ should equal $q_{b_k b_j}$. Therefore, the number of unique terms is only NY * ND * (ND+1)/2 , which can be visualized as a triangular prism shaped solid of terms. No array definition scheme will permit anything other than rectangular shapes to be defined for an array; but using pointer definitions and bypassing the address calculation methods native to the language, it is possible to build the required data structure in a linear array, which will result in space saving and also lead to more easily vectorizable code.

The final step in enhancing the extent of vectorization of the code on a CRAY is to take the compilation listing and follow the compiler generated messages indicating the DO loops which will not vectorize due to dependencies in the indices of the array variables referred to within the loop. Sometimes, these dependencies can be removed by modifying the code. A few ways to get around vector dependencies are given in Reference 15.

In addition to vectorization, great savings in computation can be achieved by micro-tasking and multi-tasking. These are techniques which use the parallelism inherent in most solution algorithms. Micro-tasking results in the spawning of several processes made up of small chunks of code, each of which is equivalent to a few lines of source program which can be executed in parallel, with reliable and repeatable results. Multi-tasking on the other hand identifies entire subroutines or sub-problems which can be computed in parallel. Both these techniques involve a considerable investment in programming and testing effort and have not been attempted in the present work.

5.8 Algorithm for Combined Dynamic Analysis and First/Second Order Design Sensitivity Analysis

A consolidated step-by-step algorithm for dynamic analysis and first/second order design sensitivity analysis based on the methods derived upto this point is presented in the following paragraphs

Step 1: Read in the system description, initial configuration of the system, initial design vector, initial choice of independent coordinates, integration error tolerance, starting/ending time for simulation and time step Δt.

Step 2: Perform Newton-Raphson iterations to determine exact initial values of dependent coordinates. Solve for initial values of dependent velocities from Equation 2.28. Solve for initial values of dependent $\underline{q}_{\underline{b}}$, $\dot{\underline{q}}_{\underline{b}}$, $\underline{q}_{b_j b_k}$ and $\dot{\underline{q}}_{b_j b_k}$ from Equations 3.24, 3.25, 4.7 and

4.8 respectively. Determine initial accelerations and Lagrange multipliers from Equation 2.30. Calculate first and second order sensitivities of accelerations and Lagrange multipliers from Equations 3.27 and 4.10. Determine the set of independent generalized coordinates.

Step 3: Start up the integration algorithm.

Step 4: Have the integration algorithm predict forward in time for all independent quantities.

Step 5: Use the predicted values of independent quantities to calculate all the necessary dependent quantities. Calculate accelerations and Lagrange multipliers from Equation 2.30; calculate their first and second order sensitivities from Equations 3.27 and 4.10 respectively. Evaluate the integrands of all integral functional constraints and their first and second order design sensitivities.

Step 6: Return to the integration algorithm and correct the predicted values for the independent quantities. Repeat Step 5 with these corrected values to obtain correct values for the dependent quantities, for accelerations, Lagrange multipliers and their first and second order sensitivities.

Step 7: Return to the integration algorithm to estimate integration error. If the error is unacceptable, the algorithm will adjust step size and order, reset the time back to the value at the previous successful step and return to Step 4. If the error tolerence is satisfied, proceed to Step 8.

Step 8: Check if the present time is a grid-point. If so, calculate the values of all grid-point constraints as well as their first and second order sensitivities and write them to a disk file. Check if termination time has been reached. If so, proceed to Step 9. If termination time has not been reached, set next output time to lower of $(t+\Delta t)$ and termination time. Determine the set of independent coordinates. If there has been a change in the set of independent coordinates, go to Step 3, else go to Step 4.

Step 9: Write the constraint function value and sensitivities of all integral constraints to a disk file. Calculate the values of all design dependent constraints as well as their first and second order sensitivities and write them to a disk file and exit from program.

Further details of the symbolic preprocessor and vectorization on the supercomputer along with the relevant computer code are presented in Reference 16.

CHAPTER VI

NUMERICAL RESULTS

The techniques developed in the previous chapters were implemented in a computer code and tested on selected numerical problems. The results from the programs were verified by two different methods, both of which are based on perturbation analysis. The main thrust in the verifications was towards validating the second order design sensitivities of the state variables and the cost/constraint functions. The method used in the present work for dynamic analysis and first order design sensitivity analysis is largely based on Reference 5, in which the first order sensitivities have been adequately verified. Therefore, the same numerical examples were used in the present work in order to skip over the verification of first order sensitivities and instead concentrate on verifying second order sensitivities. The test procedures and the results of the test problems are presented in the next two sections.

6.1 Verification of Results

In order to assess the performance of any solution process, it is necessary to devise some scheme by which the results of the process can be verified. Design sensitivity analysis is essentially a method of computing derivatives of the state variables and the

cost/constraint functions with respect to design. Derivatives computed by any process are usually verifiable by methods based on perturbation theory. Two such methods, derived for verifying the second order sensitivity of state and cost/constraint functions are as follows:

1) Verification of Second Order Design Sensitivity: The second order design sensitivity of the cost/constraint functions can be verified in the following manner:

The first and second order design sensitivities of the cost/constraint functions are calculated at the current design $\underline{b}$. The design is then given a small perturbation $\Delta\underline{b}$, so that the new design becomes

$$\underline{b}^* = \underline{b} + \Delta\underline{b} \tag{6.1}$$

The same sensitivities are then recalculated at the perturbed design. The actual change in the value of the first order design sensitivity of a particular constraint is given by

$$\Delta\psi^i_{\underline{b}} = \psi^i_{\underline{b}}(\underline{b}^*) - \psi^i_{\underline{b}}(\underline{b}) \tag{6.2}$$

If $\psi^i_{\underline{bb}}$ is the matrix of second order design sensitivities of ψ^i, and if the change in design $\Delta\underline{b}$ is sufficiently small, then $\delta\psi^i_{\underline{b}}$, the predicted change in the first order sensitivity of ψ^i is given by

$$\delta\psi^i_{\underline{b}} = \psi^i_{\underline{bb}}\ \Delta\underline{b} \tag{6.3}$$

If the matrix of second order design sensitivities is correct and if the perturbation is small enough to justify the linear approximation,

then the actual change in first order sensitivity as measured by $\Delta\psi^i_{\underline{b}}$ should be approximately the same as the predicted change in first order sensitivity, i.e., $\delta\psi^i_{\underline{b}}$.

2) Check on Second Order State Sensitivity: In order to verify the second order state sensitivity, the first and second order state sensitivities are evaluated at the original design and at a slightly perturbed design as before. Consider the variation in the first order sensitivity of the acceleration associated with generalized coordinate i at time t when the design undergoes a small perturbation. The actual change in the first order sensitivity is given by

$$\Delta\ddot{q}^i_{\underline{b}}(t) = \ddot{q}^i_{\underline{b}}(t,\underline{b}^*) - \ddot{q}^i_{\underline{b}}(t,\underline{b}) \tag{6.4}$$

If $\ddot{q}^i_{\underline{bb}}$ is the matrix of second order design sensitivities of q^i, and if the change in design $\Delta\underline{b}$ is sufficiently small, then the change predicted in the first order sensitivity of $\ddot{q}^i$ is given by

$$\delta\ddot{q}^i_{\underline{b}}(t) = \ddot{q}^i_{\underline{bb}}(t,\underline{b})\ \Delta\underline{b} \tag{6.5}$$

If the second order sensitivity matrix is correct and if the perturbation in design is small enough, then $\Delta\ddot{q}^i_{\underline{b}}(t)$ must be approximately equal to $\delta\ddot{q}^i_{\underline{b}}(t)$. Since both $\Delta\ddot{q}^i_{\underline{b}}$ and $\delta\ddot{q}^i_{\underline{b}}$ vary with time, the best way to compare the two quantities is by generating a plot over time of $\Delta\ddot{q}^i_{\underline{b}}$ and $\delta\ddot{q}^i_{\underline{b}}$. Then, the difference between the curves at

any time t is a measure of the error in the second order sensitivity of the component of acceleration under consideration.

The preceding method can also be used to check the values of second order design sensitivities of position and velocity. However, the error in these quantities is usually lower than the errors in second order sensitivity of acceleration. It is, therefore, sufficient to check the values of second order sensitivity of acceleration alone.

The input required to be fed to the preprocessor for the examples presented in this chapter are given in Appendix III.

The following points regarding the implementation of these error tests should be noted:

1) For all problems, the perturbed design was obtained from the original design by adding to each component of the design vector a certain percentage of its original value. Several perturbed designs were tested, usually with changes of 1.0, 5.0 and 10.0 percent, although the results presented here reflect only a single perturbed design.

2) Although the check on second order state sensitivity can be carried out on the partial derivative of any component of the generalized acceleration vector with respect to one of the design variables, the perturbed and predicted changes in the value of only one such derivative is presented here for the sake of brevity. Similarly, the checks on only one of the cost/constraint functions is presented in each example.

3) The computing time reported for the state sensitivity check is based on a time step which represents 1% of the total simulation time. This small interval was chosen in order to get as many points on the graph as possible. Increasing the time step to a larger value will reduce the computation time. In this connection it is also worth noting that about 25% of the time spent by the program was on the Input/Output needed to leave behind the data trail used for verifying the output and to generate the graphs. If the program is used only to compute sensitivity information and feed it to an optimization routine, the computation time can be expected to be even less since there will be no need to leave behind a very big audit trail in such an application.

4) The preprocessor was run on a NAS-6650 which is comparable to an IBM/370 and the FORTRAN programs were executed on a CRAY X-MP/48 supercomputer.

6.2 Numerical Examples

Example 1 : Double Slider Mechanism

The initial assembly of a double slider mechanism and the design variables of the problem are shown in Figure 6.1. The only loading is the weight of the sliding bodies. The link connecting the two sliders is massless. The simulation time is from 0 to 1 second.

The input data for the problem, in MKS units is as follows:

Masses:

$$m_1 = 8.0;\ m_2 = 8.0;$$

Moments of Inertia:

$$J_1 = 8.0;\ J_2 = 8.0;$$

The vector of design variables is

$$\underline{b} = [-0.707107 \quad 0.707107]^T$$

The cost functional was defined as:

$$\psi_0 = \int_0^1 (y_2)^2 \, dt$$

Results

The first order design sensitivity vector at the base design was found to be

$$1 = [-0.7130899 \quad 0.7130899]^T$$

The second order design sensitivity matrix was found to be

$$H = \begin{bmatrix} 0.36777898 & -0.36777898 \\ -0.36777898 & 0.36777898 \end{bmatrix}$$

After a 1% change in the design vector, the perturbed design was

$$\underline{b}^* = [-0.714178 \quad 0.714178]^T$$

The first order design sensitivity at the perturbed design was found to be

$$1^* = [-0.7180667 \quad 0.7180667]^T$$

Therefore, the actual change in first order design sensitivity is

$$\Delta[\psi_0]_{\underline{b}} = [-4.976764E\text{-}3 \quad 4.976764E\text{-}3]$$

The predicted change in first order sensitivity is

$$\delta[\psi_0]_{\underline{b}} = [-5.201182E\text{-}3 \quad 5.201182E\text{-}3]$$

which shows a reasonably good agreement with the actual change. The numerical algorithm as it is implemented now necessarily calculates both first and second order design sensitivities. Hence the algorithm also calculated the second order design sensitivity at the perturbed design and when this information was used to project back to the first order design sensitivity at the base design, the predicted difference in first order design sensitivity between the two designs was found to be

$$\delta[\psi_0]_{\underline{b}} = [-4.754088E\text{-}3 \quad 4.754088E\text{-}3]$$

Thus, the average of the two predicted changes in first order design sensitivity is

$$\text{Average } \delta[\psi_0]_{\underline{b}} = [-4.977635E\text{-}3 \quad 4.977635E\text{-}3]$$

Therefore, the prediction of change in first order sensitivity based on the average second order sensitivity over the perturbed range shows excellent agreement.

The state sensitivity check for this problem was performed on $\frac{\partial \ddot{x}_1}{\partial b_1}$ and the results are plotted in Figure 6.2. It can be seen that the predicted change in the first order sensitivity of $\ddot{x}_1$ with respect to design variable b_1 follows the actual change quite closely. Figure 6.3 shows the same state sensitivity when the design is perturbed by 5%. The predicted change in sensitivity is seen to follow the actual change in sensitivity reasonably well even for this larger perturbation.

The computation time for a single evaluation of first and second sensitivities for this problem was 0.588 cpu-seconds on the CRAY X-MP/48.

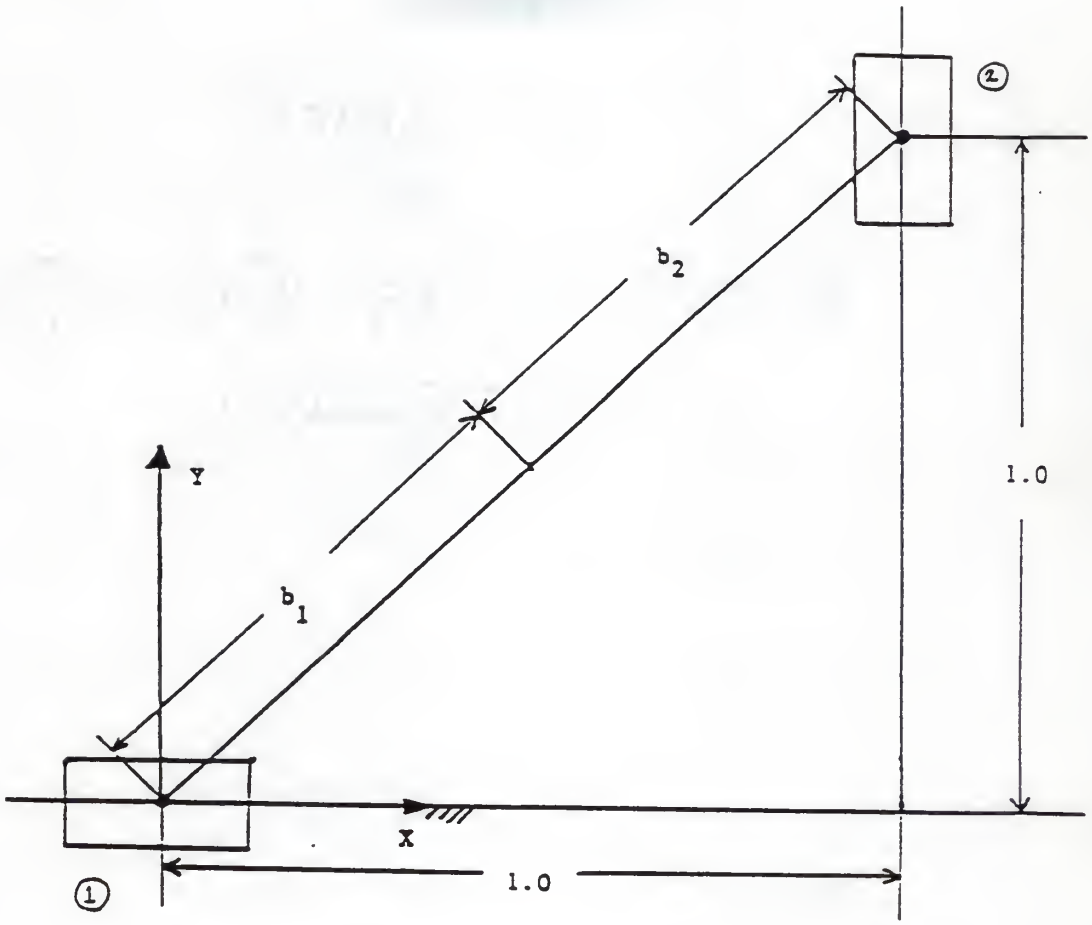

Figure 6.1. Example 1:
Double Slider Mechanism.

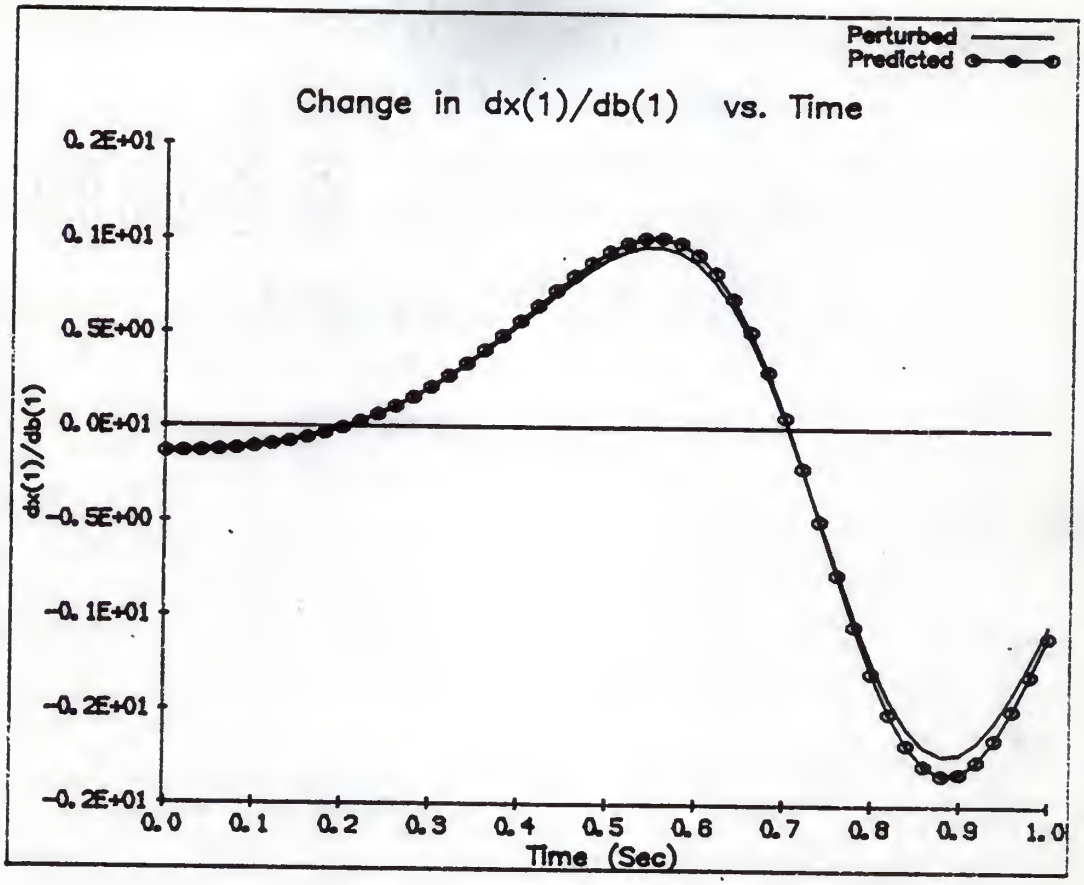

Figure 6.2. State sensitivity check for Example 1 with 1% perturbation

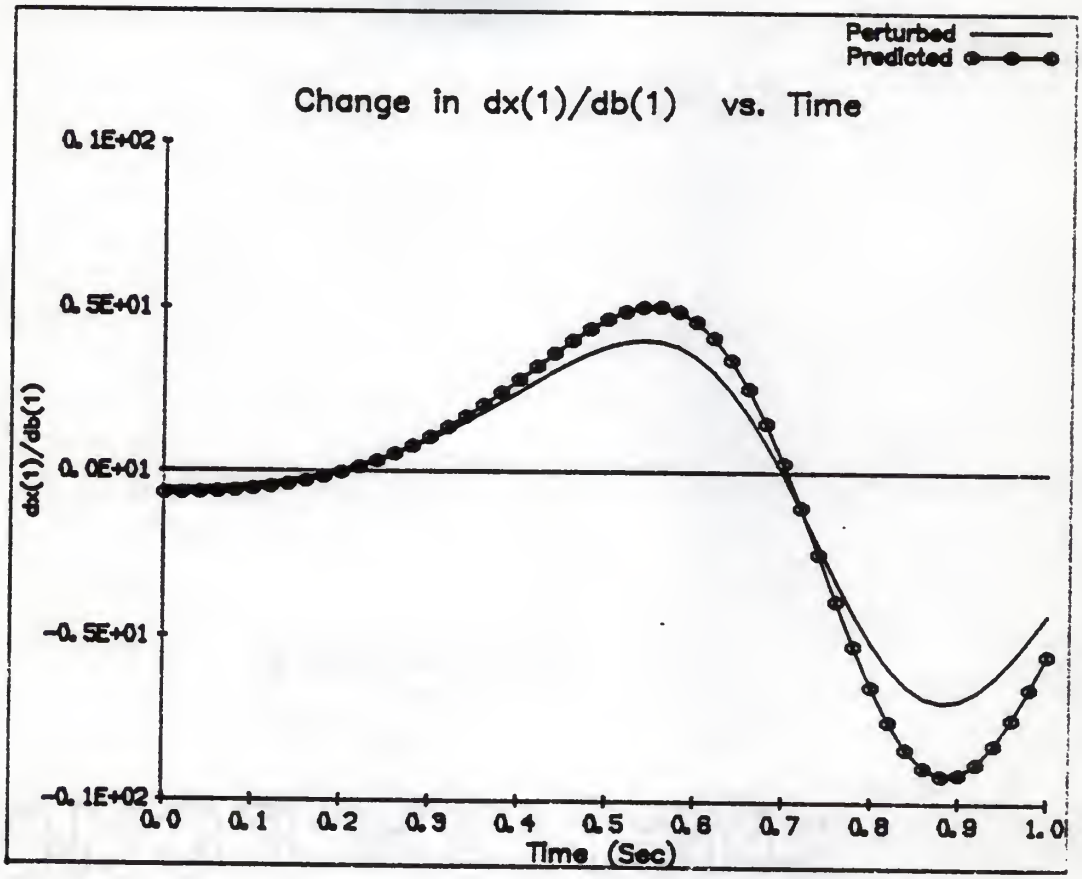

Figure 6.3. State sensitivity check for Example 1 with 5% perturbation

Example 2 : Slider-Crank Mechanism

The initial assembly of a slider-crank mechanism and the design variables of the problem are shown in Figure 6.4. The loading in this case is the weight of the members and a constant force of 125 lbs which acts on the piston in the direction indicated. The simulation time is from 0 to 1 second.

The input data for the problem, in foot-pound units is as follows:

Masses:

$$m_1 = 5.0;\ m_2 = 15.0;\ m_3 = 8.0;$$

Moments of Inertia:

$$J_1 = 0.5;\ J_2 = 2.5;\ J_3 = 8.0;$$

The vector of design variables is

$$\underline{b} = [-0.25 \quad 0.25]^T$$

The cost functional was defined as:

$$\psi_0 = \int_0^1 (\dot{x}_3 - 20)^2 \, dt$$

Results

The first order design sensitivity vector at the base design was

$$1 = [\ 46.73852 \quad -34.30223]^T$$

The second order design sensitivity matrix was found to be

$$H = \begin{bmatrix} 36.81818 & -9.517410 \\ -9.517410 & 32.47064 \end{bmatrix}$$

After a 1% change in the design vector, the perturbed design was

$$\underline{b}^* = [-0.2525 \quad 0.2525]^T$$

The first order design sensitivity at the perturbed design was found to be

$$1^* = [\ 46.62628 \ -34.20157]^T$$

Therefore, the actual change in first order design sensitivity is

$$\Delta[\psi_0]_{\underline{b}} = [-1.11224E\text{-}1 \quad 1.00657E\text{-}1]$$

The predicted change in first order sensitivity was calculated to be

$$\delta[\psi_0]_{\underline{b}} = [-1.15839E\text{-}1 \quad 1.04970E\text{-}1]$$

The change in first order sensitivity as predicted by the second order design sensitivity at $\underline{b}^*$ was found to be

$$\delta[\psi_0]_{\underline{b}} = [-1.08774E\text{-}1 \quad 9.64827E\text{-}2]$$

Therefore, the average of the two predicted changes in first order sensitivity is

$$\text{Average } \delta[\psi_0]_{\underline{b}} = [-1.12307E\text{-}1 \quad 1.00721E\text{-}1]$$

The above figures show very good agreement between the predicted and actual change in first order sensitivity.

The state sensitivity check for this problem was performed on $\frac{\partial \ddot{x}_3}{\partial b_1}$ and the results are plotted in Figure 6.5. It can be seen that the predicted change in the first order sensitivity of $\ddot{x}_3$ with respect to design variable b_1 follows the actual change quite closely. Figure 6.6

shows the same sensitivity with a 10% perturbation in design. It is evident that the predicted change in first order design sensitivity is not as close in this case due to the large perturbation.

The computation time for a single evaluation of first and second sensitivities for this problem was 2.790 cpu-seconds on the CRAY X-MP/48.

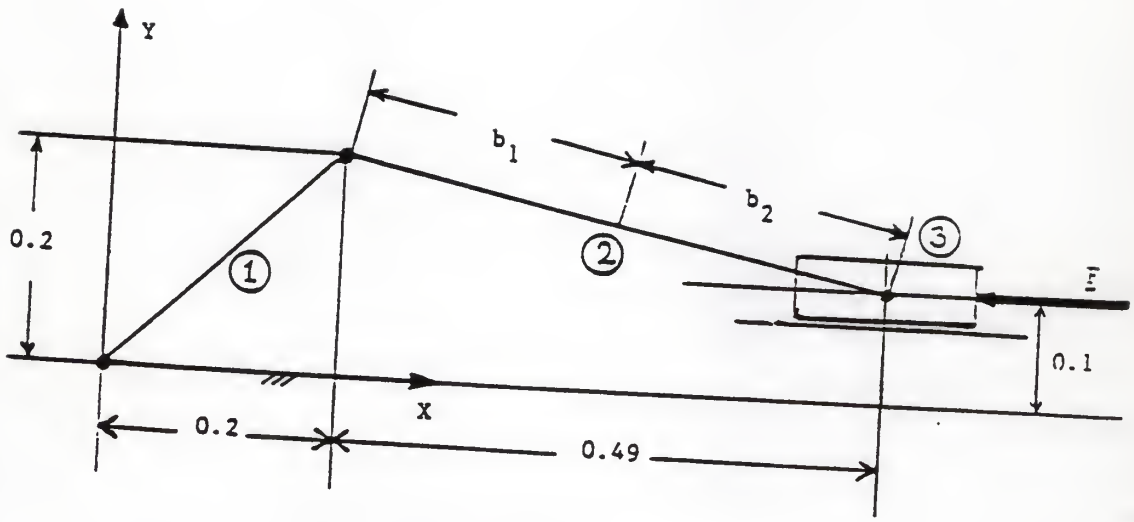

Figure 6.4. Example 2:
Slider-crank Mechanism.

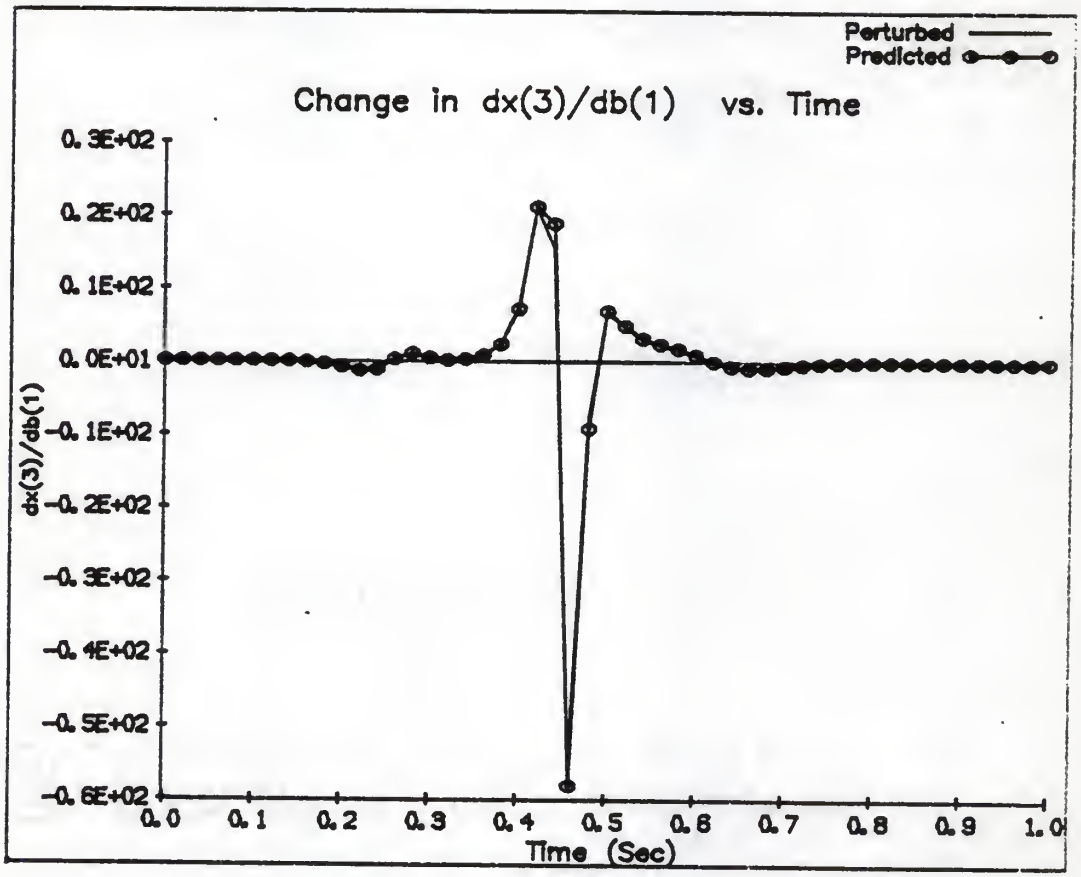

Figure 6.5. State sensitivity check for Example 2 with 1% perturbation

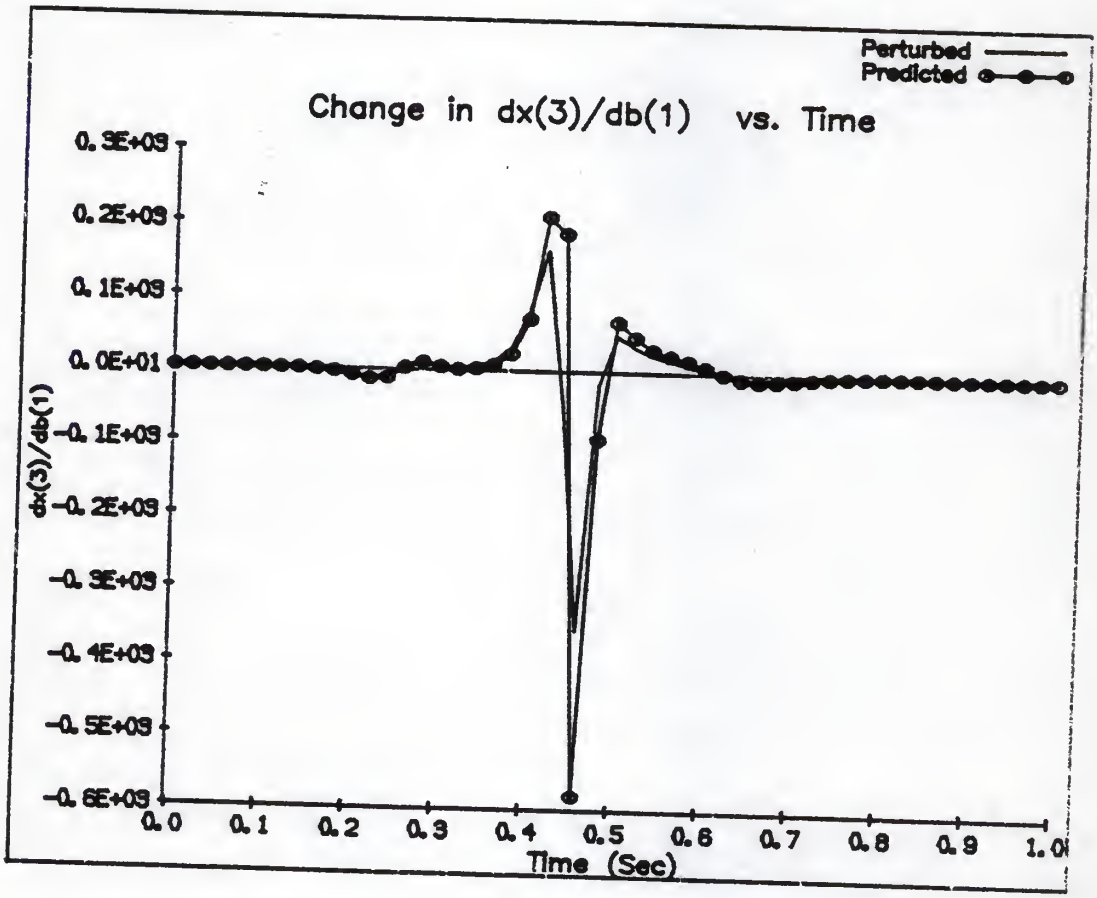

Figure 6.6. State sensitivity check for Example 2 with 10% perturbation

Example 3 : Two Degree-of-freedom Vibration Absorber:

The vibration absorber shown in Figure 6.7 has an exciting force F = 1000(Sin(20t)) acting on the upper mass. The spring constants and the damping coefficients are the design variables and the simulation time is from 0 to 1 second.

The input data for the problem, in MKS units is as follows:

Masses:

$m_1 = 20.0$; $m_2 = 20.0$;

Moments of Inertia:

$J_1 = 125.0$; $J_2 = 125.0$;

The vector of design variables is

$$\underline{b} = [\ 3920.0 \quad 10.0 \quad 3920.0 \quad 10.0]^T$$

The objective of the vibration absorber is to minimize the travel of the upper mass and accordingly the cost functional is defined as:

$$\psi_0 = \int_0^1 (y_1 - 5)^2 \, dt$$

Results

The first order design sensitivity vector at the base design was found to be

$$1 = [-8.09486E\text{-}3 \quad -3.57658E\text{-}2 \quad 7.39875E\text{-}4 \quad -1.04082E\text{-}2]^T$$

The second order design sensitivity matrix was found to be

$$H = \begin{bmatrix} 7.00654E\text{-}6 & 5.05381E\text{-}6 & 5.68522E\text{-}7 & -2.95837E\text{-}6 \\ 5.05381E\text{-}6 & 3.57432E\text{-}4 & -1.04606E\text{-}5 & 7.51916E\text{-}5 \\ 5.68522E\text{-}7 & -1.04606E\text{-}5 & -3.16935E\text{-}7 & 5.82948E\text{-}6 \\ -2.95837E\text{-}6 & 7.51916E\text{-}5 & 5.82948E\text{-}6 & 1.88878E\text{-}4 \end{bmatrix}$$

After a 1% change in the design vector, the perturbed design is

$$\underline{b}^* = [\ 3959.2 \quad 10.1 \quad 3959.2 \quad 10.1]^T$$

The first order design sensitivity at the perturbed design was found to be

$$1^* = [-7.80745E\text{-}3 \quad -3.59392E\text{-}2 \quad 7.482816E\text{-}4 \quad -1.02688E\text{-}2]^T$$

Therefore, the actual change in first order design sensitivity is

$$\Delta[\psi_0]_{\underline{b}} = [\ 2.87413E\text{-}4 \quad -1.73391E\text{-}4 \quad 8.40561E\text{-}6 \quad 1.39464E\text{-}4]$$

The predicted change in first order sensitivity was found to be

$$\delta[\psi_0]_{\underline{b}} = [\ 2.97152E\text{-}4 \quad -1.68684E\text{-}4 \quad 9.39910E\text{-}6 \quad 1.38954E\text{-}4]$$

The change in first order sensitivity as predicted by the second order design sensitivity at $\underline{b}^*$ was

$$\delta[\psi_0]_{\underline{b}} = [2.77861E\text{-}4 \quad -1.77406E\text{-}4 \quad 7.43558E\text{-}6 \quad 1.39946E\text{-}4]$$

Therefore, the average of the two predicted changes in first order sensitivity is

$$\text{Average } \delta[\psi_0]_{\underline{b}} = [2.87507E\text{-}4 \quad -1.73045E\text{-}4 \quad 8.41734E\text{-}6 \quad 1.39450E\text{-}4]$$

The above figures show a very good agreement between the predicted and actual change in first order sensitivity.

The state sensitivity check for this problem was performed on $\frac{\partial \ddot{y}_1}{\partial b_3}$ and the results are plotted in Figure 6.8. It can be seen that the predicted change in the first order sensitivity of $\ddot{y}_1$ with respect to design variable b_3 follows the actual change quite closely.

The computation time for a single evaluation of first and second sensitivities for this problem was 1.374 cpu-seconds on the CRAY X-MP/48.

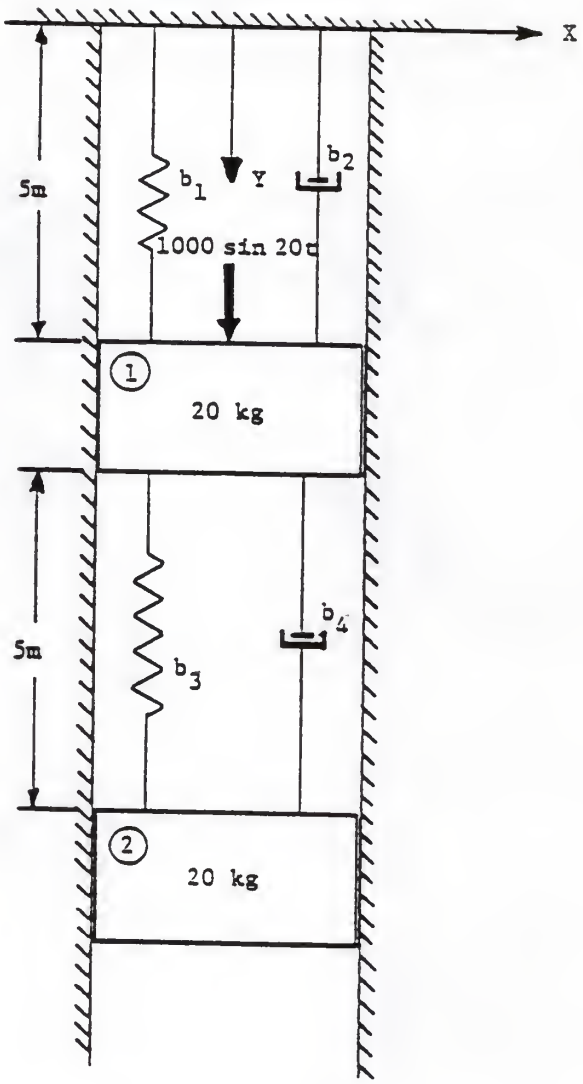

Figure 6.7. Example 3: Two Degree-of-freedom Vibration Absorber.

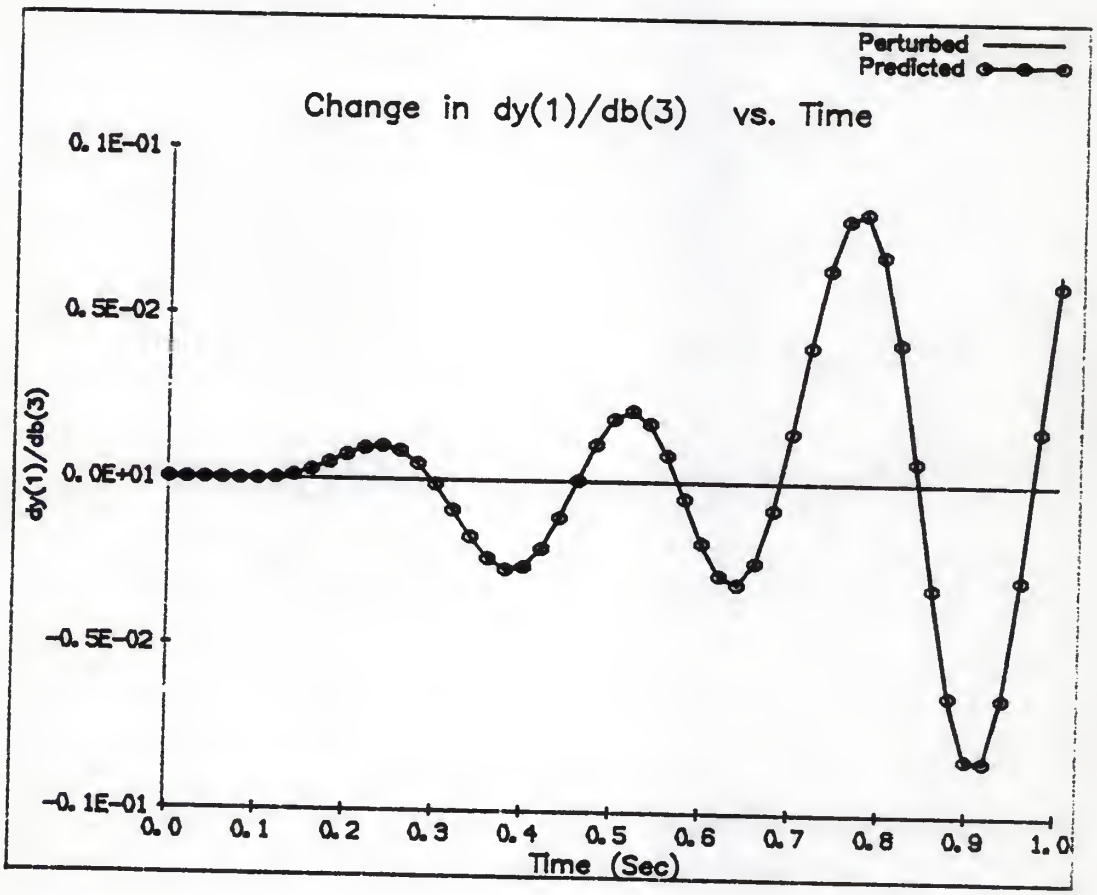

Figure 6.8. State sensitivity check for Example 3 with 1% perturbation

Example 4 : Peaucellier-Lipkin Straight Line Generator:

Figure 6.9 shows the initial assembled position of the mechanism. The external forces on the system are the self-weight of the members and a torque applied to link 1. This torque and the link lengths shown in the figure are the design variables in this problem. The simulation period is from 0 to 0.7 seconds.

The input data for the problem, in FPS units is as follows:

Masses:

$m_1 = 0.075$; $m_2 = 0.250$; $m_3 = 0.250$; $m_4 = 0.150$;

$m_5 = 0.150$; $m_6 = 0.250$; $m_7 = 0.250$;

Moments of Inertia:

$J_1 = 0.00075$; $J_2 = 0.00150$; $J_3 = 0.00150$; $J_4 = 0.00100$;

$J_5 = 0.00100$; $J_6 = 0.00150$; $J_7 = 0.00150$;

The vector of design variables is

$$\underline{b} = [\ 0.075 \quad 0.250 \quad 0.150 \quad 0.250 \quad 2.000]^T$$

The cost function in the problem is the applied torque, i.e., b_5. Since the cost function is purely design dependent and hence easy to analyse for sensitivity, we can instead consider the integral constraint

$$\psi_1 = \int_0^{0.5} (\phi_1 - 2\pi)\ dt$$

The above is equivalent to constraining link 1 to rotate through an angle of π radians in the simulation time.

Results

The first order design sensitivity vector at the base design was

$$\underline{1} = [2.61956E+0 \quad -3.49850E+0 \quad 2.56192E+0 \quad -6.00479E-1 \quad -4.351017E-2]^T$$

The second order design sensitivity matrix was found to be

$$H = \begin{bmatrix} 4.41025E+2 & -4.03689E+2 & 4.10180E+2 & -6.84574E+1 & -1.72144E+1 \\ -4.03689E+2 & 4.92034E+2 & -4.26735E+2 & 9.49733E+1 & 1.43794E+1 \\ 4.10180E+2 & -4.26735E+2 & 4.34601E+2 & -7.56105E+1 & -1.76551E+1 \\ -6.84574E+1 & 9.49733E+1 & -7.56105E+1 & 2.16974E+1 & 2.24636E+0 \\ -1.72144E+1 & 1.43794E+1 & -1.76551E+1 & 2.24636E+0 & 8.66466E-1 \end{bmatrix}$$

After a 1% change in the design vector, the perturbed design was

$$\underline{b}^* = [0.07575 \quad 0.25250 \quad 0.15150 \quad 0.25250 \quad 2.02000]^T$$

The first order design sensitivity at the perturbed design was found to be

$$\underline{1}^* = [1.98171E+0 \quad -2.92680E+0 \quad 1.98529E+0 \quad -5.08195E-1 \quad -1.65114E-2]^T$$

Therefore, the actual change in first order design sensitivity was

$$\Delta[\psi_0]_{\underline{b}} = [-6.3784E-1 \quad 5.7169E-1 \quad -5.7663E-1 \quad 9.2284E-2 \quad 2.6998E-2]$$

The predicted change in first order sensitivity was

$$\delta[\psi_0]_{\underline{b}} = [-5.7861E-1 \quad 8.1223E-1 \quad -6.4942E-1 \quad 1.7184E-1 \quad 1.9500E-2]$$

The change in first order sensitivity as predicted by the second order design sensitivity at $\underline{b}^*$ was

$$\delta[\psi_0]_{\underline{b}} = [-5.4127E-1 \quad 8.2315E-1 \quad -6.2036E-1 \quad 1.7881E-1 \quad 1.6178E-2]$$

Therefore, the average of the two predicted changes in first order sensitivity is

Avg. $\delta[\psi_0]_{\underline{b}}$= [-5.5494E-1 8.1769E-1 -6.3489E-1 1.7533E-1 1.7839E-2]

The above figures do not show a good agreement between the predicted and actual change in first order sensitivity. The exact reason for the discrepancy is not clear. The variation is even more surprising in view of the fact that the agreement seen in the state sensitivity check is quite good. The behavior is probably indicative of the fact that problem size brings in its own complexities. Hence the task of testing the method developed in the present work on truly large scale problems should be taken up in the near future.

The state sensitivity check for this problem was performed on $\frac{\partial \ddot{x}_1}{\partial b_1}$ and the results are plotted in Figure 6.10. It can be seen that the predicted change in the first order sensitivity of $\ddot{x}_1$ with respect to design variable b_1 follows the actual change quite closely.

The computation time for a single evaluation of first and second sensitivities for this problem was 6.400 cpu-seconds on the CRAY X-MP/48.

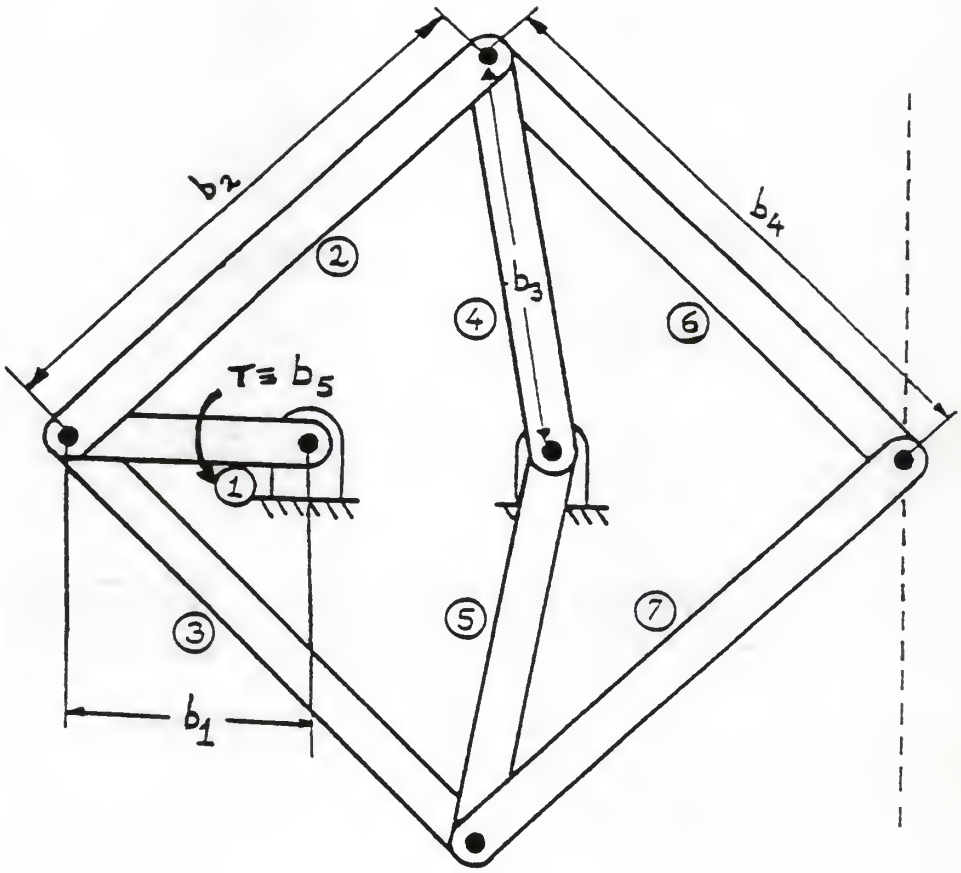

Figure 6.9. Example 4:
Peaucellier-Lipkin
Straight Line Generator

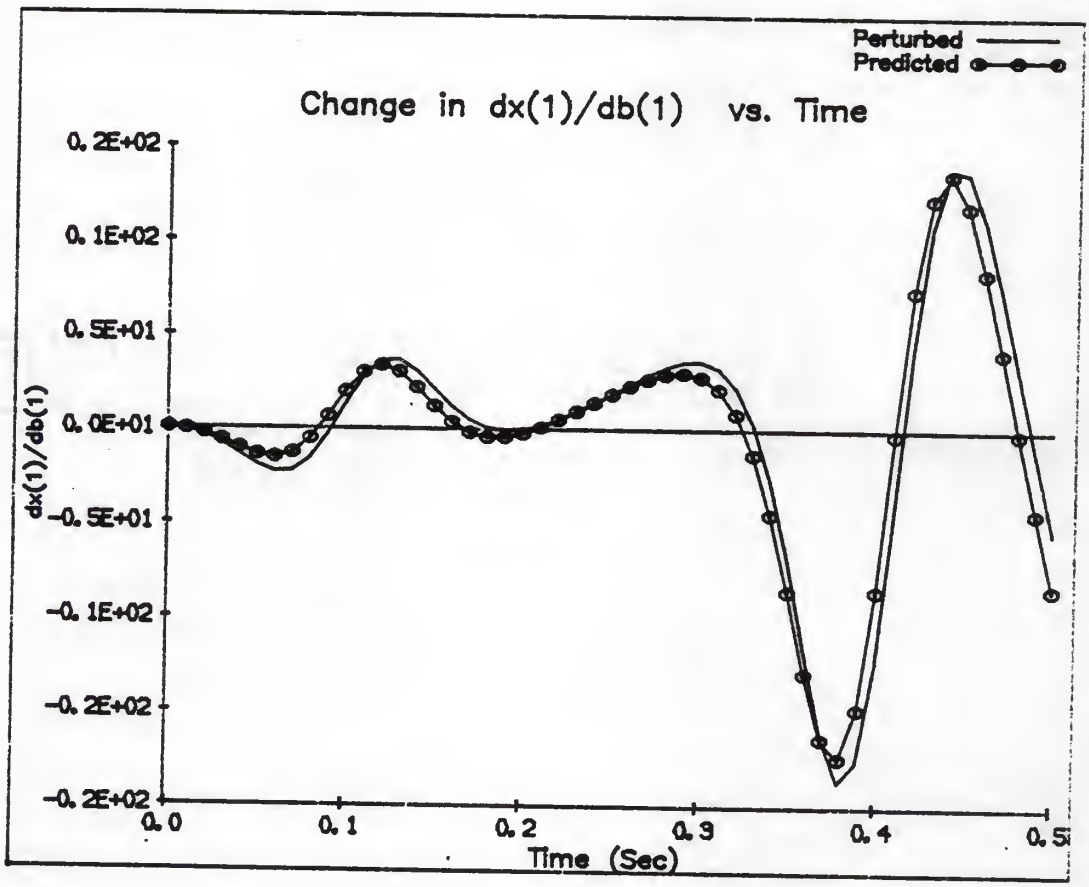

Figure 6.10. State sensitivity check for Example 4 with 1% perturbation

CHAPTER VII

CONCLUSION

A general method is developed for second order design sensitivity analysis of constrained dynamic systems based on a direct differentiation technique. The results obtained in the previous chapter indicate that the method is computationally feasible and yields results that are consistent with those obtained by perturbation analysis. The earlier work [6,7] in the area of second order design sensitivity analysis was oriented towards the evaluation of second order design sensitivities of the particular systems investigated. The present work has resulted in a more general method which can be applied to an entire class of systems. Also the earlier attempts were based entirely on the adjoint variable method [7] or a combination of the direct differentiation method and the adjoint variable method [6]. A pure direct differentiation approach was not attempted earlier as the computation required for this was considered to be very high. Hence the present work is implemented on a supercomputer.

The use of symbolic computing in design sensitivity analysis has resulted in an implementation in which the amount of user input required is kept to a bare minimum. The resulting software is more efficient, reliable and versatile. The use of symbolic computing to

generate the problem-specific code is also an effective way of improving vectorization on a supercomputer.

7.1 Recommendations for Future Research and Development

The software developed in the present work should be tested on several more problems. Numerical studies and tests are needed to gauge the performance of the software. The accuracy of the program can be estimated by solving a variety of problems with known analytical solutions. The numerical techniques used in the present work can be improved upon. Better algorithms for choosing independent coordinates, size of time step for numerical integration, error tolerances, etc. need to be developed.

The software for evaluating second order design sensitivity must be coupled with a suitable optimization routine to test its performance in actual optimization work. A comparative study of optimization using first and second order information will help determine the cost-benefit ratio for the computation work involved in evaluating second order design sensitivity.

LIST OF REFERENCES

1. Sheth, P. N. and Uicker, J. J., Jr., "IMP (Integrated Mechanisms Program), A Computer Aided Design Analysis System for Mechanisms and Linkages", ASME Journal of Engineering for Industry, Ser. B, Vol. 94, 1972.

2. Orlandea, N., Chace, M. A., and Calhan D. A., "A Sparsity Oriented Approach to the Dynamic Analysis and Design of Mechanical Systems, Part I and II", ASME Journal of Engineering for Industry, Ser. B, Vol. 99, 1977.

3. Wehage, R. A., and Haug, E. J., "Generalized Coordinate Partitioning in Dynamic Analysis of Mechanical Systems", Technical Report No. 75, Division of Materials Engineering, University of Iowa, December 1980.

4. Haug, E. J., Wehage, R. A., and Mani, N. K., "Design Sensitivity Analysis of Large Scale Constrained Dynamic Mechanical Systems", ASME Journal of Mechanisms, Transmissions and Automation in Design, Vol. 106, June 1984.

5. Krishnaswami, P., "Computer-Aided Optimal Design of Constrained Dynamic Systems", Ph.D. Thesis, Department of Mechanical Engineering, University of Iowa, 1983.

6. Haug, E. J., Mani, N. K., and Krishnaswami, P., "Design Sensitivity Analysis and Optimization of Dynamically Driven Systems", Computer-Aided Analysis and Optimization of Mechanical System Dynamics, Springer-Verlag, 1984

7. Haug, E. J., and Ehle, P., "Second Order Design Sensitivity Analysis for Mechanical System Dynamics", International Journal for Numerical Methods in Engineering, Vol. 18, 1982.

8. Haug, E. J., and Arora, J. S., Applied Optimal Design, Wiley-Interscience, New York, 1979.

9. Tomovic, R., Sensitivity Analysis of Dynamic Systems, McGraw-Hill, New York, 1963.

10. Greenwood, D. T., Principles of Dynamics, Prentice-Hall, Englewood Cliffs, NJ, 1965.

11. Hearn, A. C., "REDUCE User's Manual", The Rand Corporation, Santa Monica, CA, 1985.

12. Orlandea, N., Wiley, J. C., and Wehage, R. A., "ADAMS2 : A Sparse Matrix Approach to the Dynamic Simulation of Two-Dimensional Mechanical Systems", SAE Technical Paper Series 780486, 1978.

13. Baumgarte, J., "Stabilisation of Constraints and Integrals of Motion in Dynamic Systems", Computer Methods in Applied Mechanics and Engineering 1, 1972.

14. Shampine, L. F., and Gordon, M. K., Computer Solution of Ordinary Differential Equations : The Initial Value Problem, W. J. Freeman, San Francisco, CA, 1975.

15. Cray Reasearch Inc., "FORTRAN (CFT) Reference Manual", Publication No. SR-0009, 1984.

16. Ramaswamy, S., "Symbolic Computing and Vectorization for Second Order Design Sensitivity Analysis of Constrained Dynamic Systems", Internal Report, Department of Mechanical Engineering, Kansas State University, Manhattan, KS, 1988.

APPENDIX I

The following is an alternate way of deriving Equation 4.9. The equations of kinematic constraint of Equation 2.1 are differentiated twice with respect to time and then twice with respect to design.

Taking the total time derivative of Equation 2.1, the equations of kinematic constraint, we get

$$\dot{\Phi}(\underline{q}, \dot{\underline{q}}, \underline{b}) = 0 \tag{A1.1}$$

Taking the total time derivative of Equation A1.1, we get

$$\ddot{\Phi}(\underline{q}, \dot{\underline{q}}, \ddot{\underline{q}}, \underline{b}) = 0 \tag{A1.2}$$

It may be noted from the above that as each time derivative is taken, it results in an additional dependence on a higher derivative of the generalized coordinate vector $\underline{q}$. Thus, $\dot{\Phi}$ has an additional dependence on $\dot{\underline{q}}$ and $\ddot{\Phi}$ has an additional dependence on $\ddot{\underline{q}}$. The following relationship can also be deduced from the above equations:

$$\Phi_{\underline{q}} = \dot{\Phi}_{\dot{\underline{q}}} = \ddot{\Phi}_{\ddot{\underline{q}}} \tag{A1.3}$$

Differentiating Equation A1.2 with respect to design variable b_j, we get

$$\ddot{\Phi}_{b_j} + \ddot{\Phi}_{\underline{q}}\, \underline{q}_{b_j} + \ddot{\Phi}_{\dot{\underline{q}}}\, \dot{\underline{q}}_{b_j} + \ddot{\Phi}_{\ddot{\underline{q}}}\, \ddot{\underline{q}}_{b_j} = 0 \tag{A1.4}$$

The result of Equation A1.4 can also be obtained by the following steps:

Differentiating Equation 2.1, the equations of kinematic constraint with respect to design variable b_j, we get

$$\Phi_{b_j} + \Phi_{\underline{q}}\, q_{b_j} = 0 \tag{A1.5}$$

Taking the total time derivative of Equation A1.5, we get

$$\dot{\Phi}_{b_j} + \Phi_{\underline{q}}\, \dot{q}_{b_j} + \dot{\Phi}_{\underline{q}}\, q_{b_j} = 0 \tag{A1.6}$$

Taking the total time derivative of Equation A1.6, we get

$$\ddot{\Phi}_{b_j} + \Phi_{\underline{q}}\, \ddot{q}_{b_j} + 2\,\dot{\Phi}_{\underline{q}}\, \dot{q}_{b_j} + \ddot{\Phi}_{\underline{q}}\, q_{b_j} = 0 \tag{A1.7}$$

Equations A1.4 and A1.7 are equivalent since they both resulted from taking the time derivative of the equations of kinematic constraint twice and differentiating once with respect to design variable b_j, although the order of taking these derivatives differed. Cancelling out the terms that are identical in the two equations, we get the following equality relationship

$$\ddot{\Phi}_{\dot{\underline{q}}} = 2\,\dot{\Phi}_{\underline{q}} \tag{A1.8}$$

Differentiating Equation A1.2 with respect to design variable b_k and keeping track of all the dependencies, we get

$$\ddot{\Phi}_{b_j b_k} + \ddot{\Phi}_{b_j \underline{q}}\, q_{b_k} + \ddot{\Phi}_{b_j \dot{\underline{q}}}\, \dot{q}_{b_k} + \ddot{\Phi}_{b_j \ddot{\underline{q}}}\, \ddot{q}_{b_k} +$$

$$\ddot{\Phi}_{\underline{q} b_k}\, q_{b_j} + \left[\ddot{\Phi}_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\underline{q}}\, q_{b_k} + \left[\ddot{\Phi}_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\dot{\underline{q}}}\, \dot{q}_{b_k} + \left[\ddot{\Phi}_{\underline{q}}\, \tilde{q}_{b_j}\right]_{\ddot{\underline{q}}}\, \ddot{q}_{b_k} + \ddot{\Phi}_{\underline{q}}\, q_{b_j b_k} +$$

$$\ddot{\Phi}_{\dot{\underline{q}} b_k}\, \dot{q}_{b_j} + \left[\ddot{\Phi}_{\dot{\underline{q}}}\, \tilde{\dot{q}}_{b_j}\right]_{\underline{q}}\, q_{b_k} + \left[\ddot{\Phi}_{\dot{\underline{q}}}\, \tilde{\dot{q}}_{b_j}\right]_{\dot{\underline{q}}}\, \dot{q}_{b_k} + \left[\ddot{\Phi}_{\dot{\underline{q}}}\, \tilde{\dot{q}}_{b_j}\right]_{\ddot{\underline{q}}}\, \ddot{q}_{b_k} + \ddot{\Phi}_{\dot{\underline{q}}}\, \dot{q}_{b_j b_k} +$$

$$\ddot{\Phi}_{\ddot{q}b_k}\, q_{b_j} + \left[\ddot{\Phi}_{\ddot{q}}\, \tilde{q}_{b_j}\right]_{q}\, q_{b_k} + \left[\ddot{\Phi}_{\ddot{q}}\, \tilde{q}_{b_j}\right]_{\dot{q}}\, \dot{q}_{b_k} + \left[\ddot{\Phi}_{\ddot{q}}\, \tilde{q}_{b_j}\right]_{\ddot{q}}\, \ddot{q}_{b_k} + \ddot{\Phi}_{\ddot{q}}\, \ddot{q}_{b_j b_k} = 0 \qquad \text{(A1.9)}$$

Making the substitutions indicated by Equations A1.3 and A1.8 and grouping like terms, we get

$$\begin{aligned}
\Phi_q\, \ddot{q}_{b_j b_k} = &- \Phi_{qb_k}\, \ddot{q}_{b_j} - \left[\Phi^{i}_{q}\, \tilde{\ddot{q}}_{b_j}\right]_{q}\, q_{b_k} - 2\, \dot{\Phi}_{qb_k}\, \dot{q}_{b_j} - 2\left[\dot{\Phi}^{i}_{q}\, \tilde{\dot{q}}_{b_j}\right]_{q}\, q_{b_k} \\
&- 2\left[\Phi^{i}_{q}\, \tilde{\dot{q}}_{b_j}\right]_{q}\, \dot{q}_{b_k} - 2\, \dot{\Phi}_{q}\, \dot{q}_{b_j b_k} - \Phi_{qb_k}\, \ddot{q}_{b_j} - \left[\ddot{\Phi}^{i}_{q}\, \tilde{q}_{b_j}\right]_{q}\, q_{b_k} \\
&- \ddot{\Phi}_{q}\, q_{b_k b_j} - 2\left[\dot{\Phi}^{i}_{q}\, \tilde{q}_{b_j}\right]_{q}\, \dot{q}_{b_k} - \left[\Phi^{i}_{q}\, \tilde{q}_{b_j}\right]_{q}\, \ddot{q}_{b_k} - \ddot{\Phi}_{b_j b_k} \\
&- \Phi_{b_j q}\, \ddot{q}_{b_k} - 2\, \dot{\Phi}_{b_j q}\, \dot{q}_{b_k} - \ddot{\Phi}_{b_j q} q_{b_k}
\end{aligned} \qquad \text{(A1.10)}$$

Equation A1.10 is the same as Equation 4.9, thereby proving that the differentiations have been carried out correctly. In the process of carrying out this derivation, the useful relationships of Equations A1.3 and A1.8 were also established.

APPENDIX II

The following is a simple example of the kind of terms calculated by the symbolic preprocessor. This example considers the fifth non-standard kinematic constraint of the double slider example problem of Chapter 6. All the terms worked out by the symbolic preprocessor and the relevant lines of FORTRAN code written out by the preprocesor are listed in the following paragraphs:

The constraint considered here is based on the fact that the length of the massless link connecting the two sliding bodies can be determined from the X-coordinate of body 1 and the Y-coordinate of body 2 as well as from the two design variables. The length of the link should be the same in both cases and the constraint equation that expresses this condition is

```
CST(5):=(Q(1)-1)**2+Q(4)**2-(B(2)-B(1))**2;
```

where

CST(5) is the fifth non-standard kinematic constraint;

Q(1) is the X-coordinate of body 1;

Q(4) is the Y-coordinate of body 2 and

B(1) and B(2) are the two design variables as shown in Figure 6.1.

Since non-standard kinematic constraints are taken in by the preprocessor exactly in the form they are fed in, the relevant

equation in the vector of equations of kinematic constraint, $\Phi(\underline{q}, \underline{b})$, is written out in FORTRAN as

```
PHI(5)=(Q(1)-1)**2+Q(4)**2-(B(2)-B(1))**2
```

The relevant terms in the jacobian matrix $\Phi_{\underline{q}}$ denoted by the array AJAC are written out in FORTRAN as

```
AJAC(5,1)=2*Q(1)-2
AJAC(5,4)=2*Q(4)
```

For the remaining terms only the terms and the relevant FORTRAN code generated by the preprocessor are listed below and these should by themselves be self-explanatory.

$\Phi_{\underline{q}}$ - Time derivative of jacobian (negative term written out):

```
AJACT(5,1)=-2*QD(1)
AJACT(5,4)=-2*QD(4)
```

where QD is the vector of generalized velocities.

$\Phi_{\underline{q}}$ - Second time derivative of jacobian (negative term written out):

```
AJACTT(5,1)=-2*QDD(1)
AJACTT(5,4)=-2*QDD(4)
```

where QDD is the vector of generalized accelerations.

$\Phi_{\underline{b}}$ - Derivative of constraint equation with respect to design (negative term written out):

```
DPHIDB(5,1)=-(2*B(2)-2*B(1))
DPHIDB(5,2)=-(-2*B(2)+2*B(1))
```

All the terms related to second order design sensitivity analysis are directly absorbed in the relevant right hand sides of the

equations for which the terms are calculated. Terms required for solving for the $q_{b_j b_k}$ and $\dot{q}_{b_j b_k}$ of the dependent coordinates using Equations 4.7 and 4.8 are added to the relevant components of matrix RHSA. RHSA is defined to have a size of NY by (ND)*(ND+1)/2 where NY is the number of generalized coordinates (3 * no. of bodies in the system) and ND is the number of design variables.

Similarly, matrix RHSB is used to store the right hand sides required to solve for $\lambda_{b_j b_k}$ and $q_{b_j b_k}$ from Equation 4.10. RHSB is defined to have a size of (NY+NF) by (ND)*(ND+1)/2 where NF is the number of kinematic constraints and the other terms have bee explained in the earlier paragraph. Terms in the top NY planes of RHSB are those obtained by differentiating the equations of motion and the terms in the bottom NF planes are those obtained by differentiating the equations of kinematic constraint. Therefore, the terms derived from the equations of kinematic constraint will have a positive offset of NY added to the constraint number when they add on to the relevant component of RHSB. Further, the symbolic preprocessor recognizes the fact that certain terms have interchanged indices as in the case of terms $\left[M\ \tilde{q}_{b_j}\right]_{b_k}$ and $\left[M\ \tilde{q}_{b_k}\right]_{b_j}$ both of which occur in the right hand side of Equation 4.6. The preprocessor does not calculate the algebraic expressions for both these terms; instead, the first term is calculated and the expressions for the second term are obtained from the expressions for the first term by switching the indices. In fact,

the preprocessor does not write out two separate subroutines for the two terms. A single subroutine reflecting the combined effect of both the terms is written out. The following paragraphs list out the terms related to second order sensitivity analysis written out by the preprocessor for the particular constraint under consideration. In each case, the term(s) and the relevant equation as well as the corresponding lines of FORTRAN code are listed. The names of the arrays used for the terms related to dynamic analysis and first as well as second order design sensitivity analysis are as follows:

EL - vector of Lagrange multipliers;

DELQ - $\underline{q}_{b_j}$;

DELQD - $\dot{\underline{q}}_{b_j}$;

DELQDD - $\ddot{\underline{q}}_{b_j}$;

DELEL - $\underline{\lambda}_{b_j}$;

DQBB - $\underline{q}_{b_j b_k}$;

DVBB - $\dot{\underline{q}}_{b_j b_k}$;

DABB - $\ddot{\underline{q}}_{b_j b_k}$;

DELBB - $\underline{\lambda}_{b_j b_k}$;

The relevant terms and the lines of FORTRAN code generated by the preprocessor for second order sensitivity analysis are as follows:

$\left[\Phi_{\underline{q}}\ \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\ \underline{q}_{b_k}$ - Equation 4.7 (term subtracted from RHSA):

```
      RHSA(5,1)=RHSA(5,1)-(2*DELQ(4,1)**2+2*DELQ(1,1)**2)
      RHSA(5,2)=RHSA(5,2)-(2*DELQ(4,2)*DELQ(4,1)+2*DELQ(1,2)*
     . DELQ(1,1))
      RHSA(5,3)=RHSA(5,3)-(2*DELQ(4,2)**2+2*DELQ(1,2)**2)
```

$\Phi_{b_j b_k}$ - Equation 4.7 (term subtracted from RHSA):

```
      RHSA(5,1)=RHSA(5,1)-((-2))
      RHSA(5,2)=RHSA(5,2)-(2)
      RHSA(5,3)=RHSA(5,3)-((-2))
```

$\Phi_{\underline{q}\ \underline{q}_{b_j b_k}}$ - Equation 4.8:

```
      RHSA(5,1)=RHSA(5,1)-(2*DQBB(4,1,1)*QD(4)+2*DQBB(1,1,1)*
     . QD(1))
      RHSA(5,2)=RHSA(5,2)-(2*DQBB(4,2,1)*QD(4)+2*DQBB(1,2,1)*
     . QD(1))
      RHSA(5,3)=RHSA(5,3)-(2*DQBB(4,2,2)*QD(4)+2*DQBB(1,2,2)*
     . QD(1))
```

$\left[\Phi_{\underline{q}}\ \tilde{\dot{\underline{q}}}_{b_j}\right]_{\underline{q}}\ \underline{q}_{b_k}$ and $\left[\Phi_{\underline{q}}\ \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\ \dot{\underline{q}}_{b_k}$ - Equation 4.8:

```
      RHSA(5,1)=RHSA(5,1)-(4*DELQD(4,1)*DELQ(4,1)+4*DELQD(1,1)
     . *DELQ(1,1))
```

```
      RHSA(5,2)=RHSA(5,2)-(2*DELQD(4,1)*DELQ(4,2)+2*DELQD(1,1)
     . *DELQ(1,2))
      RHSA(5,2)=RHSA(5,2)-(2*DELQD(4,2)*DELQ(4,1)+2*DELQD(1,2)
     . *DELQ(1,1))
      RHSA(5,3)=RHSA(5,3)-(4*DELQD(4,2)*DELQ(4,2)+4*DELQD(1,2)
     . *DELQ(1,2))
```

$\left[\Phi_q^T \tilde{\lambda}_{b_j}\right]_{b_k}$ and $\left[\Phi_q^T \tilde{\lambda}_{b_k}\right]_{b_j}$ - Equation 4.10:

```
      RHSB(1,1)=RHSB(1,1)+(4*DELEL(5,1)*DELQ(1,1))
      RHSB(1,2)=RHSB(1,2)+(2*DELEL(5,1)*DELQ(1,2))
      RHSB(1,2)=RHSB(1,2)+(2*DELEL(5,2)*DELQ(1,1))
      RHSB(1,3)=RHSB(1,3)+(4*DELEL(5,2)*DELQ(1,2))
      RHSB(4,1)=RHSB(4,1)+(4*DELEL(5,1)*DELQ(4,1))
      RHSB(4,2)=RHSB(4,2)+(2*DELEL(5,1)*DELQ(4,2))
      RHSB(4,2)=RHSB(4,2)+(2*DELEL(5,2)*DELQ(4,1))
      RHSB(4,3)=RHSB(4,3)+(4*DELEL(5,2)*DELQ(4,2))
```

$\left[\Phi_q^T \lambda\right]_{q_{b_j b_k}}$ - Equation 4.10 :

```
      RHSB(1,1)=RHSB(1,1)+(2*DQBB(1,1,1)*EL(5))
      RHSB(1,2)=RHSB(1,2)+(2*DQBB(1,2,1)*EL(5))
      RHSB(1,3)=RHSB(1,3)+(2*DQBB(1,2,2)*EL(5))
      RHSB(4,1)=RHSB(4,1)+(2*DQBB(4,1,1)*EL(5))
      RHSB(4,2)=RHSB(4,2)+(2*DQBB(4,2,1)*EL(5))
      RHSB(4,3)=RHSB(4,3)+(2*DQBB(4,2,2)*EL(5))
```

$\left[\Phi_{\underline{q}}\ \tilde{\underline{q}}_{b_j}\right]_{\underline{q}}\ \underline{q}_{b_k}$ and $\left[\Phi_{\underline{q}}\ \tilde{\underline{q}}_{b_k}\right]_{\underline{q}}\ \underline{q}_{b_j}$ - Equation 4.10:

```
      RHSB(11,1)=RHSB(11,1)-(4*DELQDD(4,1)*DELQ(4,1)+4*DELQDD(1,
     . 1)*DELQ(1,1))
      RHSB(11,2)=RHSB(11,2)-(2*DELQDD(4,2)*DELQ(4,1)+2*DELQDD(1,
     . 2)*DELQ(1,1))
      RHSB(11,2)=RHSB(11,2)-(2*DELQDD(4,1)*DELQ(4,2)+2*DELQDD(1,
     . 1)*DELQ(1,2))
      RHSB(11,3)=RHSB(11,3)-(4*DELQDD(4,2)*DELQ(4,2)+4*DELQDD(1,
     . 2)*DELQ(1,2))
```

$\Phi_{\underline{q}}\ \underline{q}_{b_j b_k}$ - Equation 4.10:

```
      RHSB(11,1)=RHSB(11,1)-(2*DQBB(4,1,1)*QDD(4)+2*DQBB(1,1,1)*
     . QDD(1))
      RHSB(11,2)=RHSB(11,2)-(2*DQBB(4,2,1)*QDD(4)+2*DQBB(1,2,1)*
     . QDD(1))
      RHSB(11,3)=RHSB(11,3)-(2*DQBB(4,2,2)*QDD(4)+2*DQBB(1,2,2)*
     . QDD(1))
```

$2\left[\Phi_{\underline{q}}\ \tilde{\dot{\underline{q}}}_{b_j}\right]_{\underline{q}}\ \dot{\underline{q}}_{b_k}$ - Equation 4.10 :

```
      RHSB(11,1)=RHSB(11,1)-(4*DELQD(4,1)**2+4*DELQD(1,1)**2)
      RHSB(11,2)=RHSB(11,2)-(4*DELQD(4,2)*DELQD(4,1)+4*DELQD(1,2
     . )*DELQD(1,1))
      RHSB(11,3)=RHSB(11,3)-(4*DELQD(4,2)**2+4*DELQD(1,2)**2)
```

$2\,\Phi_{\underline{q}}\,\dot{\underline{q}}_{b_j b_k}$ - Equation 4.10:

```
      RHSB(11,1)=RHSB(11,1)-(4*DVBB(4,1,1)*QD(4)+4*DVBB(1,1,1)*
     . QD(1))
      RHSB(11,2)=RHSB(11,2)-(4*DVBB(4,2,1)*QD(4)+4*DVBB(1,2,1)*
     . QD(1))
      RHSB(11,3)=RHSB(11,3)-(4*DVBB(4,2,2)*QD(4)+4*DVBB(1,2,2)*
     . QD(1))
```

For more complex constraints than what was considered in the present example, the number of lines of FORTRAN code increases significantly. Once the preprocessor is tested successfully on a range of simple problems, it can be expected to yield correct results for the more complicated cases also. Verification by hand working may not be feasible beyond a certain point as the terms tend to get quite unwieldy.

APPENDIX III

The following lines represent the REDUCE-2 code forming the input to the preprocessor for the examples considered in Chapter 6. For each example, the input is in the form of two separate files. The contents of each input file are explained in Chapter 5.

Double Slider Example

File 1:

```
PNAME:=DOUBLE!-SLIDER;
NB:=2;
ND:=2;
NCST:=5;
NCI:=1;
NCG:=1;
NCD:=1;
NDF:=1;
;END;
```

File 2:

```
MAS(1):=8;
MAS(2):=8;
MI(1):=8;
MI(2):=8;
CST(1):=Q(3);
CST(2):=Q(5);
CST(3):=Q(2)-1;
CST(4):=Q(6);
CST(5):=(Q(1)-1)**2+Q(4)**2-(B(2)-B(1))**2;
FCI(1):=Q(4)**2;
FCG(1):=QD(4)**2-16;
FCD(1):=B(2)+B(1);
;END;
```

Slider-crank Example

File 1:

```
PNAME:=SLIDER!-CRANK;
NB:=3;
ND:=2;
NGRVLT:=1;
NGTRAN:=1;
NRVLT:=2;
NCI:=1;
NDF:=1;
;END;
```

File 2:

```
MAS(1):=5;
MAS(2):=15;
MAS(3):=8;
MI(1):=0.5;
MI(2):=2.5;
MI(3):=8;
GRVLT(1,1):=-0.141421;
GRVLT(1,2):=0;
GRVLT(1,3):=0;
GRVLT(1,4):=0;
GRVLT(1,5):=1;
GTRAN(1,1):=0;
GTRAN(1,2):=0;
GTRAN(1,3):=0;
GTRAN(1,4):=0;
GTRAN(1,5):=0;
GTRAN(1,6):=0.1;
GTRAN(1,7):=3;
RVLT(1,1):=0.141421;
RVLT(1,2):=0;
RVLT(1,3):=B(1);
RVLT(1,4):=0;
RVLT(1,5):=1;
RVLT(1,6):=2;
RVLT(2,1):=B(2);
RVLT(2,2):=0;
RVLT(2,3):=0;
RVLT(2,4):=0;
RVLT(2,5):=2;
RVLT(2,6):=3;
GENFRC(3):=-125;
FCI(1):=(Q(3)-20)**2;
;END;
```

Two-degree-of-freedom Vibration Absorber

File 1:

```
PNAME:=VIBRATION!-ABSORBER;
NB:=2;
ND:=4;
NCST:=4;
NCI:=1;
NCD:=1;
NDF:=2;
;END;
```

File 2:

```
MAS(1):=20;
MAS(2):=20;
MI(1):=125;
MI(2):=125;
CST(1):=Q(1);
CST(2):=Q(2);
CST(3):=Q(5);
CST(4):=Q(6);
GENFRC(3):=1000*SIN(20*T)
          +B(1)*(-Q(3)-5)
          -B(2)*QD(3)
          -B(3)*(-Q(4)+Q(3)-5)
          +B(4)*(QD(4)-QD(3));
GENFRC(4):=+B(3)*(-Q(4)+Q(3)-5)
          -B(4)*(QD(4)-QD(3));
FCI(1):=(Q(3)-5)**2;
FCD(1):=B(1)-5000;
;END;
```

Peaucellier-Lipkin Straight Line Generator

File 1:

```
PNAME:=PEAUCELLIER!-LIPKIN;
NB:=7;
ND:=5;
NP:=1;
NGRVLT:=3;
NRVLT:=7;
NCD:=1;
NCG:=3;
NCI:=2;
NDF:=1;
;END;
```

File 2:

```
MAS(1):=0.075;
MAS(2):=0.25;
MAS(3):=0.25;
MAS(4):=0.15;
MAS(5):=0.15;
MAS(6):=0.25;
MAS(7):=0.25;
MI(1):=0.00075;
MI(2):=0.0015;
MI(3):=0.0015;
MI(4):=0.001;
MI(5):=0.001;
MI(6):=0.0015;
MI(7):=0.0015;
GENFRC(15):=B(5);
GRVLT(1,1):=B(1)/2;
GRVLT(1,3):=0.075;
GRVLT(1,5):=1;
GRVLT(2,1):=-B(3)/2;
GRVLT(2,3):=0.15;
GRVLT(2,5):=4;
GRVLT(3,1):=-B(3)/2;
GRVLT(3,3):=0.15;
GRVLT(3,5):=5;
RVLT(1,1):=-B(1)/2;
RVLT(1,3):=-B(2)/2;
RVLT(1,5):=1;
RVLT(1,6):=2;
```

```
RVLT(2,1):=-B(1)/2;
RVLT(2,3):=-B(2)/2;
RVLT(2,5):=1;
RVLT(2,6):=3;
RVLT(3,1):=B(2)/2;
RVLT(3,3):=B(3)/2;
RVLT(3,5):=2;
RVLT(3,6):=4;
RVLT(4,1):=B(2)/2;
RVLT(4,3):=-B(4)/2;
RVLT(4,5):=2;
RVLT(4,6):=6;
RVLT(5,1):=B(2)/2;
RVLT(5,3):=B(3)/2;
RVLT(5,5):=3;
RVLT(5,6):=5;
RVLT(6,1):=B(2)/2;
RVLT(6,3):=-B(4)/2;
RVLT(6,5):=3;
RVLT(6,6):=7;
RVLT(7,1):=B(4)/2;
RVLT(7,3):=B(4)/2;
RVLT(7,5):=6;
RVLT(7,6):=7;
FCD(1):=B(5);
FCG(1):=Q(6)+(B(4)/2)*COS(Q(20))-0.405;
FCG(2):=-Q(6)-(B(4)/2)*COS(Q(20))+0.395;
FCG(3):=EL(1)**2+EL(2)**2-PAR(1);
FCI(1):=(QD(15)-6.2832);
FCI(2):=QDD(15);
;END;
```

SUPERCOMPUTER BASED SECOND ORDER DESIGN SENSITIVITY ANALYSIS OF CONSTRAINED DYNAMIC SYSTEMS

by

SITARAM RAMASWAMY

B.Tech., Indian Institute of Technology, Madras, India, 1978

P.G.D.I.E, National Institute for Training in Industrial Engineering, Bombay, India, 1980

AN ABSTRACT OF A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Mechanical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

ABSTRACT

Second order design sensitivity analysis is an essential component of reliability design, second order optimization and minimum sensitivity design and is a useful tool for estimating the changes in system behavior caused by changes in design. In this thesis, a computer-oriented method is developed for second order design sensitivity analysis of planar, constrained dynamic mechanical systems. The kinematic constraint equations and the Lagrangian equations of motion for the system are derived using a constrained multi-element formulation. These equations of motion and equations of kinematic constraint are differentiated with respect to design to obtain a set of differential equations for the first order state sensitivity coefficients. The first order state sensitivity equations are again differentiated with respect to design to yield the differential equations for the second order state sensitivity coefficients. A minimal set of independent equations is automatically identified. These equations of motion and the corresponding first and second order state sensitivity equations are integrated simultaneously to obtain the dynamic response and first/second order state sensitivities. The first and second order design sensitivities of any cost and performance constraint functions can be obtained from the state sensitivities. All integrations are performed using a multi-step

predictor-corrector method. Since problem standardization is difficult, a preprocessor written in the symbolic manipulation language REDUCE-2 is used for automated generation of problem-specific quantities. Because of the extensive matrix calculations that are required for the numerical solution of the state sensitivity equations, a CRAY X-MP/48 supercomputer is used for this purpose and special measures are taken to ensure a high degree of vectorization. Finally, illustrative numerical examples are solved and the calculated second order sensitivities are verified by perturbation analysis. The results show the method to be accurate, reliable and computationally feasible.