

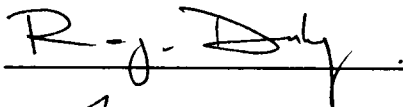
To the Graduate Council:

I am submitting herewith a thesis written by Irawan Malik entitled "Comparison Between Variational and Bilinear Time Finite Element Methods Performance in Nonlinear Dynamical Systems Analysis." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of master of science, with a major in Mechanical Engineering.



Dr. Grzegorz Kawiecki, Major Professor

We have read this thesis and
recommend its acceptance:





Accepted for the Council:



Associate Vice Chancellor
and Dean of The Graduate School

**COMPARISON BETWEEN VARIATIONAL AND BILINEAR
TIME FINITE ELEMENT METHODS PERFORMANCE IN
NONLINEAR DYNAMICAL SYSTEM ANALYSIS**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Irawan Malik

December 1995

Copyright © Irawan Malik, 1995
All rights reserved

DEDICATION

This thesis is dedicated to my parents:

Mr. Abdul Malek (alm.)
Mrs. Rahma Rachman

and

my family:

Triana Hasbir
Fajar Novriawan
Fradisa Mariana
Fahmi Ramadhana

who have given me invaluable supports and encouragements.

ACKNOWLEDGMENTS

I am particularly indebted to my thesis advisor, Dr. Grzegorz Kawiecki, for his valuable guidance, help, patience and continuous support. I would like to thank the other committee members, Dr. Jay Frankel and Dr. Rajiv Dubey, for their comments and assistance. I would also like to thank the government of the Republic of Indonesia, for their financial support. Finally, the members of my family in Indonesia, Triana Hasbir, Fajar Novriawan, Fradisa Mariana and Fahmi Ramadhana, deserve my thanks and love for patiently waiting for, and subtly encouraging me in the completion of this work and my studies.

ABSTRACT

The purpose of this thesis is to compare the performance of two finite element formulations when applied to nonlinear dynamic system analysis. One of these formulations is a new approach to the finite element method based on Hamilton's law of varying action. That approach will be called a bilinear approach or method in this thesis. Its performance will be compared against that of a widely used in nonlinear systems dynamics variational approach to the finite element method.

This study evaluates the convergence characteristics and speed (CPU time) for these two approaches in application to such single and two-degree-of-freedom dynamical systems as mass-damper-spring systems and a helicopter blade modeled as a rigid body with flap and lead-lag degrees of freedom. It is shown that the bilinear approach is slower than the variational approach in terms of CPU time. On the other hand, the bilinear approach produces steady-state solutions more often than the variational approach, particularly for systems with strong nonlinearities.

Also, the program based on the bilinear approach is easier to use because the only change necessary to analyze different dynamical systems is a modification of subroutines computing inertia, damping, stiffness and forcing function arrays. This is not the case for the computer implementation of the variational method. That approach requires defining separate linear and nonlinear mass, damping and stiffness matrices. That may be very difficult for complex dynamical systems.

TABLE OF CONTENTS

CHAPTER	Page
1. INTRODUCTION	1
Objective	2
Literature Review	2
Application of the Variational Approach to Helicopter	
Rotor Blade Dynamics Analysis	2
Application of the Variational Approach to Wind Turbine	
Rotor Blade Dynamics Analysis	4
Application of the Bilinear Approach to Helicopter	
Rotor Blade Dynamics Analysis	5
2. DERIVATION OF VARIATIONAL AND BILINEAR METHODS	7
The Variational Method	7
Discretization in Time	8
The Bilinear Method	11
Discretization in Time	11
3. APPLICATION OF VARIATIONAL AND BILINEAR METHODS TO A NONLINEAR DYNAMICAL ANALYSIS.....	15
A Single Degree of Freedom System	15
The Variational Method	16
The Bilinear Method	17
A Two Degrees of Freedom System	19
The Variational Method	20
The Bilinear Method	22
Application to a Rigid Blade Model	23
The Variational Method	26
The Bilinear Method	27
4. VALIDATION	28
Duffing's Equation	28
Verification	29

5. RESULTS	34
A Single Degree of Freedom System	34
A Two-Degree-of-Freedom System	35
A Mass-Damper-Spring System	38
A Rigid Blade Model	47
6. CONCLUSIONS	59
BIBLIOGRAPHY	60
APPENDIX	62
Program based on the Variational Algorithm	62
VITA	99

LIST OF FIGURES

Figure	Page
1. A Schematic of a Spring-Mass-Damper System (Single Degree of Freedom)	16
2. A Schematic of a Spring-Mass-Damper System (Two Degrees of Freedom)	19
3. A Schematic of a Flapping and Lead-Lag Degree-of-Freedom System (A Rigid Blade Model)	24
4. Solutions of Duffing's Equation by Thompson and Stewart (1986).....	28
5. Solution of Duffing's Equation Using the Variational Algorithm (1).....	30
6. Solution of Duffing's Equation Using the Bilinear Algorithm (1).....	31
7. Solution of Duffing's Equation Using the Variational Algorithm (2).....	32
8. Solution of Duffing's Equation Using the Bilinear Algorithm (2).....	33
9. Comparison of Results for Initial Conditions $q = -0.67$ $\dot{q} = 0.02$	36
10. Comparison of Results for Initial Conditions $q = -0.46$ $\dot{q} = 0.30$	37
11. Comparison of Results for a Two-Degree-of-Freedom System.....	39
12. Comparison of Results for Displacements q_1	40
13. Comparison of Results for Displacements q_2	41
14. Variational and Bilinear Results vs. Transient Runge-Kutta and Gear Solutions for Case 2 of Table 4	44
15. Variational and Bilinear Solutions vs. Converged Runge-Kutta Solutions for Case 2 of Table 4	45
16. Bilinear Results vs. Converged Runge-Kutta and Gear Solutions for Case 1 of Table 4	46

17. Bilinear Solutions with Initial Conditions from Converged Runge-Kutta Solutions for Case 1 of Table 4	48
18. Bilinear Solutions vs. Runge-Kutta Solutions for Case 3 of Table 4.....	49
19. Solutions of Investigated Methods vs. Runge-Kutta Solutions for Case 10 of Table 4	50
20. Variational Solutions with Initial Conditions from Converged Runge-Kutta for Case 10 of Table 4	51
21. Runge-Kutta vs. Bilinear for Beta (Flapping) Displacements (Advance Ratio 0.1)	53
22. Runge-Kutta vs. Bilinear for Zeta (Lead-Lag) Displacements (Advance Ratio 0.1)	54
23. Runge-Kutta vs. Bilinear for Beta (Flapping) Displacements (Advance Ratio 0.2)	55
24. Runge-Kutta vs. Bilinear for Zeta (Lead-Lag) Displacements (Advance Ratio 0.2)	56
25. Runge-Kutta vs. Bilinear for Beta (Flapping) Displacements (Advance Ratio 0.3)	57
26. Runge-Kutta vs. Bilinear for Zeta (Lead-Lag) Displacements (Advance Ratio 0.3)	58

NOMENCLATURE

$A(v)$	= Linear operator on v
a	= Section lift curve slope
$B(q, v)$	= Bilinear operator on q, v
C	= Damping matrix
C_D	= Drag coefficient
C_L	= Lift coefficient
c	= Blade chord, m
c_0, c_1	= Coefficients in the expression for C_L
d_0, d_1, d_2	= Coefficients in the expression for C_D
E	= Matrix, special error functional
F_0	= Forcing function amplitude, N
Q, Q_{NL}	= Linear, Nonlinear force vector
G	= Test function matrix
H	= Shape function matrix
I	= Blade moment of inertia, kg m^2
K	= Stiffness matrix
k_β, k_ζ	= Spring rate about flap and lead-lag hinges, respectively
M	= Inertia matrix, generalized aerodynamic moments in Equation (51), Nm
P	= Generalized momentum, kg/s
p	= Nodal displacement vector in bilinear method

- q = Displacement vector, meter
 $\tilde{q}$ = Approximation to q
 R = Rotor radius
 r = Radial distance along elastic axis in Equation (52), m, and vector of arbitrary constants in Equations (23) and (25)
 T = Kinetic energy, Nm
 U_p, U_T = Blade cross section air velocity components, m/s
 u = Velocity, m/s
 V = Potential energy function, Joules and blade cross section resultant air velocity, m/s

GREEK SYMBOLS

- α = Angle of attack, rad
 β = Blade flapping angle, rad
 $\delta(.)$ = Variation of $(.)$
 $\delta T, \delta V$ = Variation of kinetic and strain energies
 ζ = Blade lead-lag angle, rad
 ζ_β, ζ_ζ = Viscous damping ratio in flap and lead-lag hinges, respectively
 θ = Blade pitch at 0.75 of blade radius
 λ = Lagrange multiplier and induced inflow in Equation (54)
 μ = Advance ratio

ξ	= Nodal displacement vector in variational method
v	= Vector of test functions
v_{β}	= Dimensionless rotating flap natural frequency
$\tilde{v}$	= Vector of test function approximations
ρ	= Air mass density
τ	= Local time coordinate
ψ	= Nondimensional time, $\omega.t$, and azimuth (position of a rotor blade), degree
Ω	= Rotor blade angular velocity, rad/s
ω	= Natural frequency, rad/s
ω_{β}	= Nondimensional, nonrotating flap frequency
ω_{ζ}	= Nondimensional, nonrotating lead-lag frequency
$(\dot{})$	= $\frac{d}{dt}$

CHAPTER 1

INTRODUCTION

The analysis of rotating blade dynamics in flight is difficult due to the complexity of rotor structure, and variations in inflow velocities and resulting air loads. These air loads are in most cases periodic and affect in-plane, out of plane and torsional blade motions.

The system of equations describing the motion of helicopter or wind turbine rotor blades can be described using a system of nonlinear ordinary differential equations. The stability of such a system can be analyzed using a system of equations linearized about the steady state solution of the nonlinear system.

One of the most popular methods to investigate the dynamic response of rotor blades is the variational time finite element method based on Hamilton's principle (Panda and Chopra, 1986) and the Galerkin-type finite element method (Straub and Friedmann, 1981). Other methods that can be found in recent publications are the variational finite element method based on weak Hamilton formulation (Borri *et.al*, 1985) and the bilinear finite element method based on Hamilton's law of varying action (Izadpanah, 1986 ,and Kawiecki and Sivaneri, 1994). These finite element methods, which have been widely used in the structural analysis problems, have a great potential to solve complex (e.g., rotor blade) dynamics problems.

One of the most promising methods to investigate the response of dynamical systems is the bilinear formulation of Hamilton's law of varying action. Izadpanah (1986), and Peters and Izadpanah (1988) used the bilinear formulation to analyze the response of a single degree

of freedom linear system. Kawiecki and Sivaneri (1994) extended the bilinear formulation to nonlinear systems and applied it to analyze the dynamics of a rotor blade modeled as a nonlinear multiple degree of freedom system.

Objective

The main objective of this thesis is to compare the performance of the variational time finite element approach based on Hamilton's principle with the performance of the bilinear finite element in time approach of Hamilton's law of varying action to analyze nonlinear system dynamics. This study will estimate the convergence characteristics and speed (CPU time) for these two approaches in application to one and two degrees of freedom mass, damper, and spring nonlinear systems. The performance of both investigated methods applied to the rotor blade dynamics problem is also investigated. The rotor blade is modeled as a rigid body with flap and lead-lag degrees of freedom.

Literature Review

The following literature review is limited to studies analyzing applications of the variational approach and the bilinear approach to investigate the dynamic response of rotating blades.

Application of the Variational Approach to Helicopter Rotor Blade Dynamics Analysis

Panda and Chopra (1986) examined the aeroelastic stability of flap bending, lead-lag bending and torsion of hingeless and bearingless rotor blades in forward flight using the finite

element in time formulation. The blade was modeled as an elastic beam and discretized into beam elements. Each element had fifteen degrees of freedom.

The system of nonlinear, periodic differential equations was transformed to the modal space using six or seven (depending on a particular case) most significant eigenvectors. The resulting system was solved iteratively using Floquet transition matrix theory.

The blade stability solution was obtained using the periodic perturbation equations linearized about the steady state equilibrium position. These blade perturbation equations then were transformed to six or seven normal mode equations. The details of the formulation can be found in Sivaneri and Chopra (1982 and 1984).

Results of that study showed that four beam elements were sufficient to obtain satisfactory response and stability solutions for a hingeless rotor blade, but six elements were needed to get satisfactory solutions for a bearingless rotor blade, as shown by the convergence study.

Friedmann and Kottapalli (1982) investigated the aeroelastic stability and response of an elastic-isolated (hingeless) blade in forward flight. The spatial dependence of partial differential nonlinear equations of motion was discretized using a multimodal Galerkin's method. The nonlinear time dependent equilibrium position or steady state response was obtained by solving a sequence of linear periodic response problem (quasilinearization). That investigation showed that converged solutions could be obtained within three revolutions.

Panda and Chopra (1987) investigated the aeroelastic stability and response of flap bending, lead-lag bending and torsion (three-degree of freedom) of a four-blade hingeless

rotor made of composite materials in forward flight using a finite element formulation based on Hamilton's principle. Nonlinear periodic equations were solved for steady blade response (periodic) solution using two different methods. The first method, used to calculate the initial conditions, was an iterative procedure based on Floquet theory. The second method was a time finite element formulation. Cubic and quadratic shape functions were used. It was found that the finite element method was more computationally efficient. That investigation also evaluated the influence of composite plies orientation on rotor dynamics.

Application of the Variational Approach to Wind Turbine Rotor Blade Dynamics Analysis

Friedmann (1976) derived the nonlinear differential equations of motion for a single pretwisted cantilevered wind turbine blade. His formulation included distributed torsional stiffness without root torsional flexibility. He noted that the torsional degree of freedom was less important than in helicopter blade aeroelastic problem because of a high torsional rigidity of a wind turbine configuration.

The system of nonlinear partial differential equations for the coupled flap-lag torsional motion of the blade was derived using Houbolt and Brooks's (1958) approach. Resulting partial differential equations of motion were used to obtain the aeroelastic stability boundaries for a single wind turbine blade. The mathematical formulation of this problem was not given in that paper.

Kottapalli (1979) *et.al.* formulated a system of coupled flap-lag-torsion, periodic, nonlinear differential equations motion of an isolated horizontal axis wind turbine blade. That work was based on the following assumptions: (a) the blade material was isotropic, (b)

the cross-sectional blade shape was unaffected by deformations, (c) deflections, strains, and rotations (slopes) were small, (d) the blade-tower coupling was neglected.

Chopra and Dugundji (1979) used Lagrange's equations to derive nonlinear equations of motion for an isolated three-degree-of-freedom rotor blade. The resulting nonlinear algebraic equations were solved numerically using the Newton-Raphson iterative technique. The limit cycle analysis for forced oscillations due to periodic forces from the gravity field and wind shear was performed.

Application of the Bilinear Approach to Helicopter Rotor Blade Dynamics Analysis

Izadpanah (1986), and Peters and Izadpanah (1988) applied the time finite element to a single-degree-of-freedom flapping motion of a rotor blade in hover and forward flight. That study has demonstrated that the trailing terms in the numerical formulation of Hamilton's law of varying action must be applied in terms of momenta to guarantee the convergence of initial value problems.

Sivaneri and Kawiecki (1989) used the bilinear time-finite-element approach to obtain an approximate solution of the steady response of a rotor blade problem. In that paper, the finite element method based on Hamilton's principle was used to discretize the spatial dependence of the equations of motion and the bilinear approach of Hamilton's law of varying action was used to discretize the temporal dependence of the equations of motion.

Kawiecki and Sivaneri (1994) applied the finite element method in time based on the bilinear approach to analyze a nonlinear multidegree of freedom dynamical model of a rotor blade. Similarly as in their earlier work, they used the bilinear formulation of Hamilton's law

of varying action to discretize the temporal dependence of the equations of motion. The blade was modeled as a rigid body.

Two approaches to discretization of the temporal dependence of the equations of motion were used in that study. The first approach was used to obtain the response of general dynamic system using marching in time. This method was applied to find the Floquet transition matrix to test the stability of the blade motion. The second approach was used to obtain the solution of a helicopter blade steady state periodic-response problem using an assumption that the solution was identical at the beginning and the end of one revolution. Test functions based on the Legendre polynomials were used.

CHAPTER 2

DERIVATION OF VARIATIONAL AND BILINEAR METHODS

In this chapter, both the variational approach and the bilinear approach to the finite element method are outlined.

The Variational Method

The variational time finite element method is based on Hamilton's principle. A mathematical statement of Hamilton's principle is

$$\delta \int_{t_1}^{t_2} (U - T) dt = 0 \quad (1)$$

where U : the potential energy and

T : the kinetic energy.

A more general statement of Hamilton's principle is Hamilton's law that expresses dynamic equations. This Hamilton's law can be expressed as

$$\delta \int_{t_1}^{t_2} (U - T) dt - \int_{t_1}^{t_2} \delta W dt = 0 \quad (2)$$

where δU , δT , δW are the variation of strain energy, the variation of kinetic energy, and the virtual work done, respectively.

Discretization in Time

The equations of motion in the modal space have the following form

$$[M] \{\ddot{q}\} + [C(\psi)] \{\dot{q}\} + [K(\psi)] \{q\} = \{Q(\psi)\} \quad (3)$$

where

- $[M]$: inertia matrix
- $[C]$: damping matrix
- $[K]$: stiffness matrix
- $[Q(\psi)]$: a force vector containing nonlinear and periodic terms.
- ψ : azimuth and
- $\{q\}$: a nodal displacement vector.

Multiplying Equation (3) by $\{\delta q\}$ and integrating it by parts for $\psi \in (\psi_i, \psi_f)$, we arrive at

$$\int_{\psi_i}^{\psi_f} \{\delta q\}^T \{[M] \{\ddot{q}\} + [C] \{\dot{q}\} + [K] \{q\} - \{Q(\psi)\}\} d\psi = 0 \quad (4)$$

Rearrange Equation (4) to obtain

$$\int_{\psi_i}^{\psi_f} \left\{ \begin{matrix} \{\delta q\} \\ \{\delta \dot{q}\} \end{matrix} \right\}^T \left\{ \begin{matrix} \{Q(\psi)\} - [C] \{\dot{q}\} - [K] \{q\} \\ [M] \{\dot{q}\} \end{matrix} \right\} d\psi = \{0\} \quad (5)$$

where

$$\{Q\} = \left\{ \begin{matrix} \{Q(\psi)\} - [C] \{\dot{q}\} - [K] \{q\} \\ [M] \{\dot{q}\} \end{matrix} \right\}$$

and

$$\{ \delta Y \} = \begin{Bmatrix} \{ \delta q \} \\ \{ \delta \dot{q} \} \end{Bmatrix}$$

which reduces to

$$\int_{\psi_i}^{\psi_f} \{ \delta Y \}^T \{ Q \} d\psi = 0 \quad (6)$$

In the linearization process, Q function can be expressed using a first order Taylor series expansion of $\{ Q \}$ about $\{ \bar{Q} \}$, as follows

$$\{ Q \} = \{ \bar{Q} \} + \left(\frac{\delta}{\delta \{ q \}} \{ \bar{Q} \} \right) \{ \Delta q \} + \left(\frac{\delta}{\delta \{ \dot{q} \}} \{ \bar{Q} \} \right) \{ \Delta \dot{q} \} + \dots \quad (7)$$

where $\{ \bar{Q} \}$ is the value of the Q function as well as all its derivatives at point $\psi = \psi_0$.

Equation (6) can then be written in the incremental form

$$\int_{\psi_i}^{\psi_f} \{ \delta Y \}^T \{ \bar{Q} \} d\psi + \int_{\psi_i}^{\psi_f} \{ \delta Y \}^T [\bar{k}_1] \{ \Delta Y \} d\psi = 0 \quad (8)$$

This is an energy expression for the entire system, where

$$[\bar{k}_1] = \begin{bmatrix} -[K] + \frac{\delta \{ Q \}}{\delta \{ q \}} & -[C] + \frac{\delta \{ Q \}}{\delta \{ \dot{q} \}} \\ 0 & [M] \end{bmatrix}$$

A time period corresponding to one revolution is discretized into N time elements and the displacement vector in each element is expressed in terms of shape functions and nodal displacements, namely

$$\{ \mathbf{q} \} = [\mathbf{H}(\psi)] \{ \xi \}. \quad (9)$$

The velocity vector can be written as

$$\{ \dot{\mathbf{q}} \} = [\dot{\mathbf{H}}(\psi)] \{ \xi \}. \quad (10)$$

Substituting these two equations into equation (8), we will arrive at

$$\{ \bar{\mathbf{Q}}^G \} + [\bar{\mathbf{K}}_t^G] \{ \bar{\Delta} \xi^G \} = 0 \quad (11)$$

where

$$\{ \bar{\mathbf{Q}}^G \} = \sum_{i=1}^N \int_{\psi_i}^{\psi_{i+1}} \begin{bmatrix} \mathbf{H}(\psi) \\ \dot{\mathbf{H}}(\psi) \end{bmatrix}^T \{ \bar{\mathbf{Q}} \}^i \begin{bmatrix} \mathbf{H}(\psi) \\ \dot{\mathbf{H}}(\psi) \end{bmatrix} d\psi \quad (12)$$

$$\{ \bar{\mathbf{K}}_t^G \} = \sum_{i=1}^N \int_{\psi_i}^{\psi_{i+1}} \begin{bmatrix} \mathbf{H}(\psi) \\ \dot{\mathbf{H}}(\psi) \end{bmatrix}^T \{ \bar{\mathbf{k}}_t \}^i \begin{bmatrix} \mathbf{H}(\psi) \\ \dot{\mathbf{H}}(\psi) \end{bmatrix} d\psi \quad (13)$$

and ψ_i and ψ_{i+1} are lower and upper time integration limits for element i , respectively, and $\{ \bar{\Delta} \xi^G \}$ is the incremental global vector of normal coordinates. Then, we apply the periodic boundary condition $\{ \xi^G(\psi = 0) \} = \{ \xi^G(\psi = 2\pi) \}$ (one revolution from 0 to 2π), so

$$\{Q^G\} + \{K_t^G\} \{\Delta \xi^G\} = 0 \quad (14)$$

The steady-state response vector $\{\xi^G\}$ is obtained iteratively. The details of this procedure can be found in Sivaneri and Chopra (1982), and Panda and Chopra (1987).

The Bilinear Method

This formulations is based on the work by Peters and Izadpanah (1988). Details are given in Izadpanah (1986), Sivaneri and Kawiecki (1989), and Kawiecki and Sivaneri (1994).

Discretization in Time

The discretization in time is performed using Hamilton's law of varying action. Consider the equation of motion given in Equation 3 which is multiplied by a test function and integrated over domain $(0, \psi)$, so that

$$\int_0^{\psi} \{v\} ([M]\{\ddot{q}\} + [C]\{\dot{q}\} + [K]\{q\}) d\psi = \int_0^{\psi} \{v\} \{Q(\psi)\} d\psi \quad (15)$$

Integrating by parts and rearranging yields

$$\begin{aligned} \int_0^{\psi} \begin{Bmatrix} v \\ \dot{v} \end{Bmatrix}^T \begin{Bmatrix} [C]\{\dot{q}\} + [K]\{q\} \\ -[M]\{\ddot{q}\} \end{Bmatrix} d\psi = \\ - \begin{Bmatrix} v \\ \dot{v} \end{Bmatrix}^T \left[\begin{Bmatrix} [M]\{\dot{q}\} \\ \{0\} \end{Bmatrix} \right]_0^{\psi} + \int_0^{\psi} \begin{Bmatrix} v \\ \dot{v} \end{Bmatrix}^T \begin{Bmatrix} Q(\psi) \\ \{0\} \end{Bmatrix} d\psi \quad (16) \end{aligned}$$

The momenta at the beginning and end of the time period are

$$[M] \{ \dot{q}(0) \} = \{ P_0 \} \quad (17)$$

$$[M] \{ \dot{q}(\psi) \} = \{ P_\psi \}$$

In the bilinear form, Equation (16) has the form of $B(q, v) = A(v)$ where

$$B(q, v) = \int_0^\psi \begin{Bmatrix} [v] \\ [\dot{v}] \end{Bmatrix}^T \begin{Bmatrix} [C] \{ \dot{q} \} + [K] \{ q \} \\ - [M] \{ \dot{q} \} \end{Bmatrix} d\psi \quad (18)$$

$$A(v) = \left[- \begin{Bmatrix} [v] \\ [\dot{v}] \end{Bmatrix}^T \begin{Bmatrix} \{ P \} \\ \{ 0 \} \end{Bmatrix} \right]_0^\psi + \int_0^\psi \begin{Bmatrix} [v] \\ [\dot{v}] \end{Bmatrix}^T \begin{Bmatrix} Q(\psi) \\ \{ 0 \} \end{Bmatrix} d\psi \quad (19)$$

The above results show that Equations (18) and (19) are equivalent to Equation (3) or

$$\begin{aligned} B(q, v) - A(v) &= \int_0^\psi \{ v \} ([M] \{ \ddot{q} \} + [C] \{ \dot{q} \} + [K] \{ q \} - \{ Q(\psi) \}) d\psi \\ &- \{ v(0) \} \{ \{ P_0 \} - [M] \{ \dot{q}(0) \} \} + \{ v(\psi) \} \{ \{ P_\psi \} - [M] \{ \dot{q}(\psi) \} \} = 0 \end{aligned} \quad (20)$$

The error equation for this formulation can be found if the exact solution $\{q\}$ is substituted by a trial function vector $\{\tilde{q}\}$, so that

$$E(\tilde{q}, v) = B(\tilde{q}, v) - A(v) \quad (21)$$

For initial value problems with $\{P_0\}$ known and $\{P_\psi\}$ unknown $(\tilde{v}(\psi)) = 0$ must be chosen to eliminate the unknown from the error equation and $(\tilde{v}(0)) \neq 0$ should be chosen to allow the natural convergence of $\{P_0\} = [M] \{\dot{q}(0)\}$ to the initial value condition. The details are given in Izadpanah (1986).

The displacement and test functions are next discretized in the time domain,

$$\{q\} = [H(\psi)] \{p\} \quad (22)$$

$$\{v\} = [G(\psi)] \{r\} \quad (23)$$

where $[G(\psi)]$ and $[H(\psi)]$: the shape function matrices such that $G_{ij}(\psi) \neq H_{ij}(\psi)$

$\{p\}$: the nodal displacement vectors

$\{r\}$: the vector of arbitrary constants.

Because this discretization deals with an initial value problem, the displacement vector $\{q\}$ and the generalized momenta vector $\{P\}$ are known only at the first mode of the time element. Therefore the constraints are

$$\tilde{q}(0) = [H(0)] \{p\} = q_0 \quad (24)$$

$$\tilde{v}(\psi) = [G(\psi)] \{r\} = 0 \quad (25)$$

Substituting Equations (24) and (25) to Equation (20) and introducing a vector of the Lagrange multipliers, λ , we obtain

$$\begin{bmatrix} [\mathbf{B}] & [\mathbf{G}(\Psi)] \\ [\mathbf{H}(0)] & [\mathbf{0}] \end{bmatrix} \begin{Bmatrix} \{\mathbf{p}\} \\ \{\lambda\} \end{Bmatrix} = \begin{Bmatrix} \{\mathbf{A}\} \\ \{\mathbf{q}_0\} \end{Bmatrix} \quad (26)$$

This equation is solved over several revolutions in order to obtain a steady-state periodical solution. Further details are given in Kawiecki and Sivaneri (1994).

CHAPTER 3

APPLICATION OF VARIATIONAL AND BILINEAR METHODS TO A NONLINEAR DYNAMICAL ANALYSIS

The performance of these two methods are compared by applying them to obtain the response of two different spring-mass-damper nonlinear systems and of a helicopter rigid blade system. Existing programs are used. The programs are modified by writing new subroutines generating linear mass, damping, stiffness and force matrices and nonlinear damping, stiffness and force matrices for the variational algorithm. Programs based on fifth order Runge-Kutta and Gear's algorithms available in IMSL FORTRAN libraries are used to validate the investigated imposed periodicity methods. The CPU time used by both investigated algorithms is measured using calls to the processor clock inserted at the beginning and at the end of main programs. The objectives of this study involve

- (a) the comparison of the efficiency of both algorithms based on their speed (CPU time), and
- (b) the comparison of their convergence characteristics when applied to the analysis of nonlinear systems dynamics.

A Single Degree of Freedom System

The algorithms described above are used to analyze a simple one degree of freedom system. This is a spring-mass-damper system shown in Figure 1. The equation of motion describing this system is

$$[m] \{\ddot{q}\} + [c] \{\dot{q}\} + [k] \{q\} = \{Q(\psi)\} \quad (27)$$

where $Q(\psi) = F_0 \cos(\omega \cdot \psi)$. Next, the spring is made nonlinear by defining $k = q^2$ (N/m) and Equation (27) becomes Duffing's equation, see Thompson and Stewart (1986). The numerical values for the problem are as follows

$$\begin{array}{lll} m = 1.0 \text{ kg} & F_0 = 0.2 \text{ N} & \psi_{\text{step}} = 6^\circ \\ \omega = 1.0 \text{ rad/s} & c = 0.08 \text{ Ns/m} & \end{array}$$

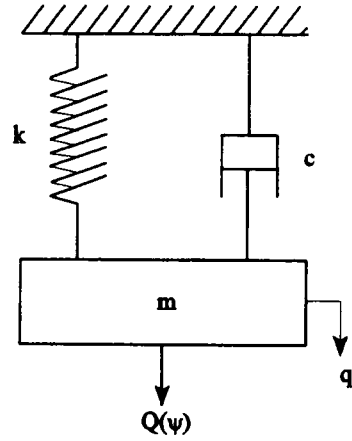


Figure 1. A Schematic of a Spring-Mass-Damper System (Single Degree of Freedom)

The Variational Method

If the nonlinear term is moved to the right-hand side, the Equation (27) becomes

$$[m] \{\ddot{q}\} + [c] \{\dot{q}\} = \{F_0 \cos(\omega \cdot \psi)\} - \{q^3\} \quad (28)$$

From Equation (8), one obtains

$$\{Q\} = \begin{Bmatrix} F_0 \cdot \cos(\omega \cdot \psi) - \{q^3\} - [c] \{\dot{q}\} \\ [m] \{\dot{q}\} \end{Bmatrix} \quad (29)$$

$$[k_t] = \begin{bmatrix} -\{3q^2\} & -[c] \\ [0] & [m] \end{bmatrix} \quad (30)$$

$$\{\delta Y\} = \begin{Bmatrix} \delta \{H(\psi) \{\xi\}\} \\ \delta \{\dot{H}(\psi) \{\xi\}\} \end{Bmatrix} \quad (31)$$

where

$$\begin{aligned} H_1 &= 1 - \frac{\psi}{\Delta}, \\ H_2 &= \frac{\psi}{\Delta}. \end{aligned} \quad (32)$$

By substituting Equations (29)-(32) into Equation (11), we obtain a system of linear equations that can be solved iteratively for nodal displacements $\{q\}$ and velocities $\{\dot{q}\}$.

The Bilinear Method

For one degree of freedom, two shape functions, and one element, the equations for a spring-mass-damper system can be presented as

$$\begin{bmatrix} \int_0^* \begin{bmatrix} G_1(\psi) & \dot{G}_1(\psi) \\ G_2(\psi) & \dot{G}_2(\psi) \end{bmatrix} \begin{bmatrix} k & c \\ 0 & -m \end{bmatrix} \begin{bmatrix} H_1(\psi) & H_2(\psi) \\ \dot{H}_1(\psi) & \dot{H}_2(\psi) \end{bmatrix} d\psi & \begin{Bmatrix} G_1(\psi) \\ G_2(\psi) \end{Bmatrix} \\ [H_1(0) & H_2(0)] & 0 & 0 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ \lambda \end{Bmatrix} = \quad (33)$$

$$\left\{ \int_0^{\Psi} \begin{bmatrix} G_1(\Psi) & \dot{G}_1(\Psi) \\ G_2(\Psi) & \dot{G}_2(\Psi) \end{bmatrix} \begin{Bmatrix} Q \\ 0 \end{Bmatrix} d\Psi + \begin{bmatrix} G_1(0) & \dot{G}_1(0) \\ G_2(0) & \dot{G}_2(0) \end{bmatrix} \begin{Bmatrix} [m] \cdot \dot{q}_1 \\ 0 \end{Bmatrix} \right\} [p_1]$$

For $H_1(0) = G_1(\psi) = 1$ and $H_2(0) = G_2(\psi) = 0$, the above equation can be rewritten as

$$\begin{bmatrix} B_{11} & B_{12} & 1 \\ B_{21} & B_{22} & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ \lambda \end{Bmatrix} = \begin{Bmatrix} A_1 \\ A_2 \end{Bmatrix} \quad (34)$$

where

$$B = \int_0^{\Psi} \begin{Bmatrix} v_1 \\ v_2 \end{Bmatrix}^T \begin{Bmatrix} k & q + c \dot{q} \\ -m & \dot{q} \end{Bmatrix} d\Psi, \quad (35)$$

$$A_1 = \int_0^{\Psi} \begin{Bmatrix} v_1 \\ v_2 \end{Bmatrix}^T [Q] d\Psi + \begin{Bmatrix} v_1(0) \\ v_2(0) \end{Bmatrix}^T [m] \dot{q}_0, \quad (36)$$

$$A_2 = p_1, \quad (37)$$

$$\lambda = [m] \{ \dot{q}_0 \}.$$

By considering Equations (22) and (23), the shape functions become

$$H_1(\Psi) = 1 - \frac{\Psi}{\Delta} \text{ and } H_2(\Psi) = \frac{\Psi}{\Delta} \quad (38)$$

$$G_1(\Psi) = \frac{\Psi}{\Delta} \text{ and } G_2(\Psi) = 1 - \frac{\Psi}{\Delta} \quad (39)$$

In the above equations p_1 and $\dot{p}_1$ are known, while p_2 , λ and $\dot{p}_2$ are unknown.

A Two Degrees of Freedom System

The previously outlined methods are also used to investigate the dynamics of the two-degree-of-freedom mass-spring-damper system shown in Figure 2.

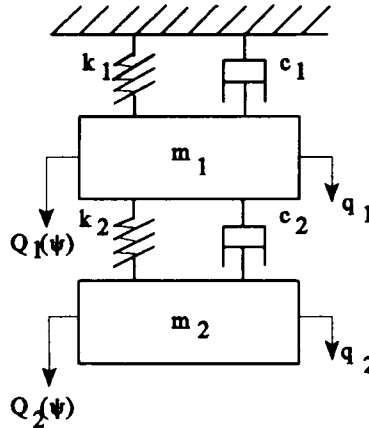


Figure 2. A Schematic of a Spring-Mass-Damper System (Two Degrees of Freedom)

The stiffness coefficients of the springs are defined as

$$k_1 = a_1 q_1 + b_1,$$

$$k_2 = a_2 q_2 + b_2,$$

where

$$a_1 = 2.0 \text{ N/m}^2, \quad b_1 = 5.0 \text{ N/m}$$

$$a_2 = 5.0 \text{ N/m}^2, \quad b_2 = 2.0 \text{ N/m}$$

The equations of motion for this system are given by

$$[m_1] \{\ddot{q}_1\} + [c_1 + c_2] \{\dot{q}_1\} - [c_2] \{\dot{q}_2\} + [k_1 + k_2] \{q_1\} - [k_2] \{q_2\} = \{Q_1(\psi)\} \quad (40)$$

$$[m_2] \{\ddot{q}_2\} - [c_2] \{\dot{q}_1\} + [c_2] \{\dot{q}_2\} - [k_2] \{q_1\} + [k_2] \{q_2\} = \{Q_2(\psi)\} \quad (41)$$

The numerical values for this system are chosen as

$$m_1 = 5.0 \text{ kg}$$

$$m_2 = 8.0 \text{ kg}$$

$$F_0 = 1.0 \text{ N}$$

$$c_1 = 5.0 \text{ Ns/m}$$

$$c_2 = 8.0 \text{ Ns/m}$$

$$\omega = 1.0 \text{ rad/s}$$

$$\psi_{\text{step}} = 6^\circ$$

$$Q_1(\psi) = 0.0$$

$$Q_2(\psi) = F_0 \cos(\omega t)$$

The Variational Method

Equations (40) and (41) can be written in the matrix form

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{Bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{Bmatrix} + \begin{bmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 \end{bmatrix} \begin{Bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{Bmatrix} + \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ Q_2(\psi) \end{Bmatrix}$$

or

$$[M] \{\ddot{q}\} + [C] \{\dot{q}\} + [K] \{q\} = \{Q(\psi)\} \quad (42)$$

where

$$[M] = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} : \text{inertia matrix,}$$

$$[C] = \begin{bmatrix} c_1 + c_2 & -c_2 \\ -c_2 & c_2 \end{bmatrix} : \text{damping matrix,}$$

$$[K] = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} : \text{stiffness matrix,}$$

(43)

$$[Q(\psi)] = \begin{Bmatrix} Q_1(\psi) \\ Q_2(\psi) \end{Bmatrix} \quad : \text{ force matrix.}$$

After expanding and simplifying the nonlinear and linear terms, $\{Q_{NL}\}$ can be represented as

$$\{Q_{NL}\} = \begin{Bmatrix} 0 \\ F_0 \cos(\omega t) \end{Bmatrix} - \begin{bmatrix} a_1 q_1^2 + a_2 q_2 q_1 - a_2 q_1^2 - a_2 q_2^2 + a_2 q_1 q_2 \\ a_2 q_1^2 - a_2 q_2 q_1 + a_2 q_2^2 - a_2 q_1 q_2 \end{bmatrix} \quad (44)$$

where the last term of Equation (44) is defined as $[K]_{NL}$, while $[K]_L = \begin{bmatrix} b_1 + b_2 & -b_2 \\ -b_2 & b_2 \end{bmatrix}$

From Equation (44), we get

$$\left\{ \frac{\delta Q_{NL}}{\delta q} \right\} = \begin{bmatrix} -(2a_1 q_1 - 2a_2 q_1 + 2a_2 q_2) & -(2a_2 q_1 - 2a_2 q_2) \\ -(2a_2 q_1 - 2a_2 q_2) & -(2a_2 q_2 - 2a_2 q_1) \end{bmatrix} \quad (45)$$

where

$$\begin{aligned} \{q\} &= [H_1(\psi) \quad H_2(\psi)] \{\xi\} \\ \{\dot{q}\} &= [\dot{H}_1(\psi) \quad \dot{H}_2(\psi)] \{\dot{\xi}\} \end{aligned}$$

By substituting these equations into Equation (11), we obtain a system of linear equations which can be solved iteratively to obtain the vectors of nodal displacements $\{q\}$ and velocities $\{\dot{q}\}$ for the above system.

The Bilinear Method

From Equation (42) and using Equations (16), (22), and (23), we obtain

$$\int_0^{\psi} \begin{bmatrix} G_1(\psi) & \dot{G}_2(\psi) \\ G_2(\psi) & \dot{G}_2(\psi) \end{bmatrix} \begin{bmatrix} [K] & [C] \\ 0 & -[M] \end{bmatrix} \begin{bmatrix} H_1(\psi) & H_2(\psi) \\ \dot{H}_1(\psi) & \dot{H}_2(\psi) \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \end{Bmatrix} d\psi = \quad (46)$$

$$\int_0^{\psi} \begin{bmatrix} G_1(\psi) & \dot{G}_1(\psi) \\ G_2(\psi) & \dot{G}_2(\psi) \end{bmatrix} \begin{Bmatrix} Q_1(\psi) \\ Q_2(\psi) \end{Bmatrix} d\psi + \begin{bmatrix} G_1(0) & \dot{G}_1(0) \\ G_2(0) & \dot{G}_2(0) \end{bmatrix} \begin{Bmatrix} [M] \dot{q}(0) \\ 0 \end{Bmatrix}$$

where $[M]$, $[C]$, and $[K]$ are defined by Equations (43). Let

$$[B] = \int_0^{\psi} \begin{bmatrix} G_1(\psi) & \dot{G}_1(\psi) \\ G_2(\psi) & \dot{G}_2(\psi) \end{bmatrix} \begin{bmatrix} [K] & [C] \\ 0 & -[M] \end{bmatrix} \begin{bmatrix} H_1(\psi) & H_2(\psi) \\ \dot{H}_1(\psi) & \dot{H}_2(\psi) \end{bmatrix} d\psi \quad (47)$$

$$[A] = \int_0^{\psi} \begin{bmatrix} G_1(\psi) & \dot{G}_1(\psi) \\ G_2(\psi) & \dot{G}_2(\psi) \end{bmatrix} \begin{Bmatrix} Q_1(\psi) \\ Q_2(\psi) \end{Bmatrix} d\psi + \begin{bmatrix} G_1(0) & \dot{G}_1(0) \\ G_2(0) & \dot{G}_2(0) \end{bmatrix} \begin{Bmatrix} [M] \dot{q}(0) \\ 0 \end{Bmatrix}$$

then the bilinear formulation for a two-degree-of-freedom system results in the following system of equations

$$\begin{bmatrix} B_{22} & B_{24} \\ B_{42} & B_{44} \end{bmatrix} \begin{Bmatrix} p_2 \\ p_4 \end{Bmatrix} = \begin{Bmatrix} A_2 \\ A_4 \end{Bmatrix} - \begin{bmatrix} B_{21} & B_{23} \\ B_{41} & B_{43} \end{bmatrix} \begin{Bmatrix} p_1 \\ p_3 \end{Bmatrix} \quad (48)$$

From the above equation, we find the nodal displacements p_2 and p_4 . The Lagrange

multipliers are

$$\begin{aligned}\lambda_1 &= [A_1] - [B_{11} \ B_{13}] \begin{Bmatrix} p_1 \\ p_3 \end{Bmatrix} - [B_{12} \ B_{14}] \begin{Bmatrix} p_2 \\ p_4 \end{Bmatrix} \\ \lambda_2 &= [A_3] - [B_{31} \ B_{33}] \begin{Bmatrix} p_1 \\ p_3 \end{Bmatrix} - [B_{32} \ B_{34}] \begin{Bmatrix} p_2 \\ p_4 \end{Bmatrix}\end{aligned}\quad (49)$$

with

$$\begin{Bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{Bmatrix} = \int_0^{2\pi} \begin{Bmatrix} G_1(\psi) Q_1(\psi) \\ 0 \\ G_2(\psi) Q_2(\psi) \\ 0 \end{Bmatrix} d\psi + \begin{Bmatrix} m_{11} \dot{q}_1(0) \\ 0 \\ m_{22} \dot{q}_3(0) \\ 0 \end{Bmatrix} \quad (50)$$

where $H_1(0) = 1$, $H_2(0) = 0$, $G_1(\psi) = 1$, and $G_2(\psi) = 0$. Nodal velocities are $\dot{p}_2(\psi) = \frac{\lambda_1}{m_{11}}$ and $\dot{p}_4(\psi) = \frac{\lambda_2}{m_{22}}$.

Application to a Rigid Blade Model

A comparative study of both methods performance when applied to the analysis of rotor blade dynamics is performed. The blade model used in this study was developed by Kaza and Kvaternik (1979). That model had flap and lead-lag degrees of freedom, as shown in Figure 3.

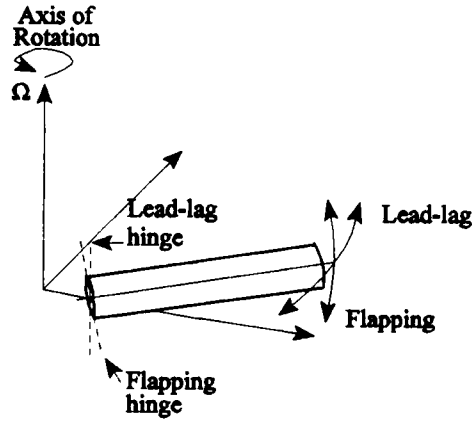


Figure 3. A Schematic of a Flapping and Lead-Lag Degree-of-Freedom System (A Rigid Blade Model)

Equations of motion for this flap-lag blade model are

$$\begin{aligned} I \ddot{\beta} + 2\Omega I \dot{\zeta} \beta + 2\zeta_{\beta} v_{\beta} \Omega I \dot{\beta} + v_{\beta}^2 \Omega^2 I \beta &= M_{\beta} \\ I \ddot{\zeta} - 2\Omega I \dot{\beta} \beta + 2\zeta_{\zeta} \omega_{\zeta} \Omega I \dot{\zeta} + \omega_{\zeta}^2 \Omega^2 I \zeta &= M_{\zeta} \end{aligned} \quad (51)$$

where

$$M_{\beta} = \frac{1}{2} \rho c \cos(\zeta) \int_0^R r (C_L V U_T - C_D V U_P) dr$$

$$M_{\zeta} = -\frac{1}{2} \rho c \int_0^R r (C_L V U_P + C_D V U_T) dr$$

and

$$C_L = c_0 + c_1 \alpha$$

$$C_D = d_0 + d_1 \alpha + d_2 \alpha^2$$

$$v_{\beta}^2 = \omega_{\beta}^2 + 1, \omega_{\beta}^2 = k_{\beta}/\Omega^2 I, \text{ and } \omega_{\zeta}^2 = k_{\zeta}/\Omega^2 I$$

Dividing both sides of Equation (51) by $\Omega^2 I$ produces the nondimensionalized equations of motion

$$\ddot{\beta} + 2 \dot{\zeta} \beta + 2 \zeta_{\beta} v_{\beta} \dot{\beta} + (1 + \omega_{\beta}^2) \beta = \frac{\gamma \cos(\zeta)}{2 a} \int_0^1 r V (C_L U_T - C_D U_P) dr \quad (52a)$$

$$\ddot{\zeta} - 2 \dot{\beta} \beta + 2 \zeta_{\zeta} \omega_{\zeta} \dot{\zeta} + \omega_{\zeta}^2 \zeta = - \frac{\gamma}{2 a} \int_0^1 r V (C_L U_P + C_D U_T) dr \quad (52b)$$

where

$$\begin{aligned} U_T &= r (\dot{\zeta} + \cos \beta) + \mu (\cos \psi \cos \beta \sin \zeta + \sin \psi \cos \zeta) + \lambda \sin \beta \sin \zeta \\ U_P &= r (\dot{\beta} \cos \zeta + \sin \beta \sin \zeta) + \mu \cos \psi \sin \beta - \lambda \cos \beta \\ V &= \sqrt{U_T^2 + U_P^2} \end{aligned} \quad (53)$$

and

$$\lambda = \mu \sin \alpha - \lambda_i \quad (54)$$

$$\alpha = \theta - \frac{U_P}{U_T} \quad \text{where}$$

$$\theta = \theta_0 - \theta_1 \cos \psi - \theta_2 \sin \psi.$$

Further details on the derivations of Equations 52 a and 52 b are given in Kaza and Kvaternik (1979).

The steady-state response of the rigid blade model is obtained for the following blade properties:

$$c_0 = 0.0 \quad c_1 = 5.7 \quad d_0 = 0.01 \quad d_1 = d_2 = 0.0$$

$\gamma = 5.0$	$a = 5.7$	$v_\beta = 1.15$	$\zeta_\beta = 0.0$
$\zeta_\zeta = 0.0$	$\omega_\beta = 1.14$	$\omega_\zeta = 1.5$	$\lambda_i = 0.03744$
$\mu = 0.3$	$\theta_0 = 0.1884$	$\theta_1 = 0.01723$	$\theta_1 = -0.116$

In matrix form, Equation (52) becomes

$$\begin{bmatrix} m_{11} & 0 \\ 0 & m_{22} \end{bmatrix} \begin{Bmatrix} \ddot{\beta} \\ \ddot{\zeta} \end{Bmatrix} + \begin{bmatrix} 2 \zeta_\beta v_\beta & 2 \beta \\ 2 \beta & 2 \zeta_\zeta \omega_\zeta \end{bmatrix} \begin{Bmatrix} \dot{\beta} \\ \dot{\zeta} \end{Bmatrix} + \begin{bmatrix} (1 + \omega_\beta^2) & 0 \\ 0 & \omega_\zeta^2 \end{bmatrix} \begin{Bmatrix} \beta \\ \zeta \end{Bmatrix} = \begin{Bmatrix} \bar{M}_\beta \\ \bar{M}_\zeta \end{Bmatrix} \quad (55)$$

where $m_{11} = m_{22} = 1$

$\bar{M}_\beta$: the right-hand-side term of Equation (52a),

$\bar{M}_\zeta$: the right-hand-side term of Equation (52b).

The Variational Method

Linear and nonlinear terms in Equation (55) have the following forms

$$\begin{aligned} [C]_L &= \begin{bmatrix} 2 \zeta_\beta v_\beta & 0 \\ 0 & 2 \zeta_\zeta \omega_\zeta \end{bmatrix} & [C]_{NL} &= \begin{bmatrix} 0 & 2 \beta \\ 2 \beta & 0 \end{bmatrix} \\ [K]_L &= \begin{bmatrix} (1 + \omega_\beta^2) & 0 \\ 0 & \omega_\zeta^2 \end{bmatrix} & [K]_{NL} &= [0] \\ [Q(\psi)]_L &= \begin{Bmatrix} \bar{M}_\beta \\ \bar{M}_\zeta \end{Bmatrix}_L & [Q(\psi)]_{NL} &= \begin{Bmatrix} \bar{M}_\beta \\ \bar{M}_\zeta \end{Bmatrix}_{NL} - [C]_{NL} \end{aligned} \quad (56)$$

$$\begin{bmatrix} \frac{\delta Q_{NL}}{\delta \dot{q}} \end{bmatrix} = \begin{bmatrix} \frac{\delta \bar{M}_\beta}{\delta \dot{\beta}} & \frac{\delta \bar{M}_\beta}{\delta \dot{\zeta}} - 2\beta \\ \frac{\delta \bar{M}_\zeta}{\delta \dot{\beta}} - 2\beta & \frac{\delta \bar{M}_\zeta}{\delta \dot{\zeta}} \end{bmatrix} \quad \begin{bmatrix} \frac{\delta Q_{NL}}{\delta q} \end{bmatrix} = \begin{bmatrix} \frac{\delta \bar{M}_\beta}{\delta \beta} & \frac{\delta \bar{M}_\beta}{\delta \zeta} \\ \frac{\delta \bar{M}_\zeta}{\delta \beta} & \frac{\delta \bar{M}_\zeta}{\delta \zeta} \end{bmatrix} \quad (57)$$

Equations (53) and (54) show that terms U_T , U_P and λ depend on each other. To simplify the problem of calculating derivatives of nonlinear terms as shown in Equations (57), it is assumed that $\lambda = \mu \sin \theta - \lambda_i$. The term U_P/U_T in Equation (54) is also neglected. That means that the effect of inflow velocity components on the angle of attack is neglected.

The Bilinear Method

Matrices $[M]$, $[C]$, $[K]$ and $[Q(\psi)]$ can be defined using Equation (55). These matrices have the following form

$$[M] = \begin{bmatrix} m_{11} & 0 \\ 0 & m_{22} \end{bmatrix} \quad [C] = \begin{bmatrix} 2 \zeta_\beta v_\beta & 2\beta \\ 2\beta & 2 \zeta_\zeta \omega_\zeta \end{bmatrix}$$

$$[K] = \begin{bmatrix} (1 + \omega_\beta^2) & 0 \\ 0 & \omega_\zeta^2 \end{bmatrix} \quad [Q(\psi)] = \begin{Bmatrix} \bar{M}_\beta \\ \bar{M}_\zeta \end{Bmatrix}$$

The bilinear method does not require separating the nonlinear terms from the linear terms. That is usually a very tedious task and, thus, the bilinear algorithm is much easier to use.

CHAPTER 4

VALIDATION

In this chapter, the computer programs based on variational and bilinear algorithms will be validated against known solutions of nonlinear problems. The known solutions used in this study have been published by Thompson and Stewart (1986).

Duffing's Equation

Thompson and Stewart (1986) have produced several solutions of Duffing's equation with different sets of initial conditions. Solutions of the following equation $\ddot{q} + c \dot{q} + q^3 = F_0 \cos(t)$ where $c = 0.08$ and $F_0 = 0.2$ are shown in Figure 4.

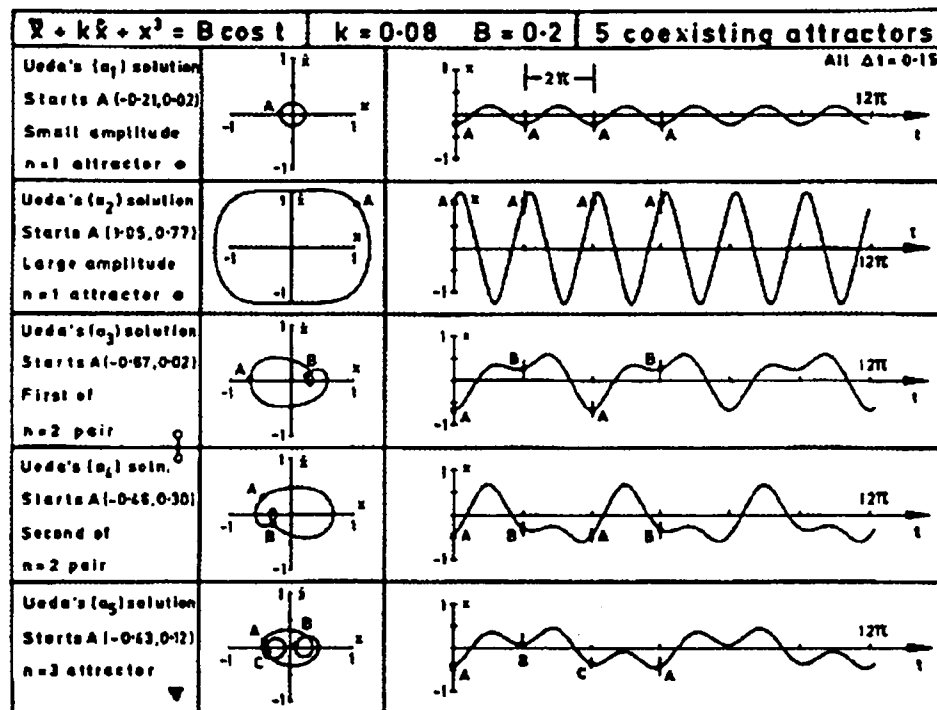


Figure 4. Solutions of Duffing's Equation by Thompson and Stewart (1986)

The results shown in Figure 4 indicate clearly that the choice of initial conditions can affect both the amplitude of a solution of a nonlinear equation and its period. Rows 1 and 2 show solutions with the same period but different amplitudes. Rows 3, 4, 5 show solutions with period equal to two and three period lengths of the forcing function.

Verification

Selected solutions shown in Figure 4 are reproduced using variational and bilinear algorithms. Figures 5 and 6 demonstrate the steady-state solution of Duffing's equation shown in the second row of Figure 4 obtained using variational and bilinear algorithms, respectively. Similarly, Figures 7 and 8 show the solution of Duffing's equation displayed in the fourth row of Figure 4 obtained using the two investigated algorithms.

The agreement between solutions obtained by the imposed period algorithms and the solutions presented by Thompson and Stewart (1986) is very good. Therefore, the bilinear and variational algorithms are considered to be correct. Next, the speed (CPU time) and convergence characteristics of both algorithms as applied to nonlinear dynamical systems analysis are investigated.

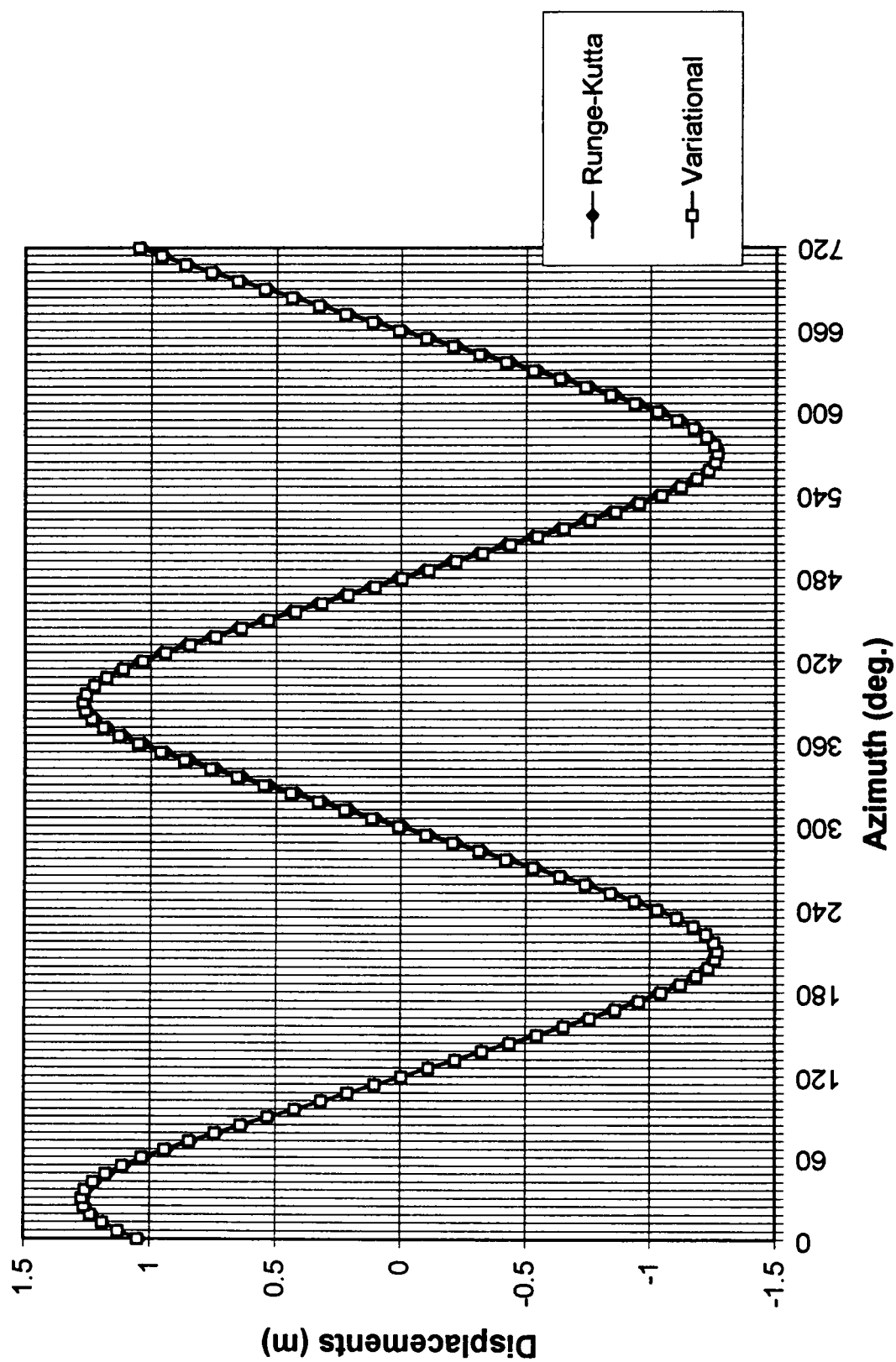


Figure 5. Solution of Duffing's Equation Using the Variational Algorithm (1)

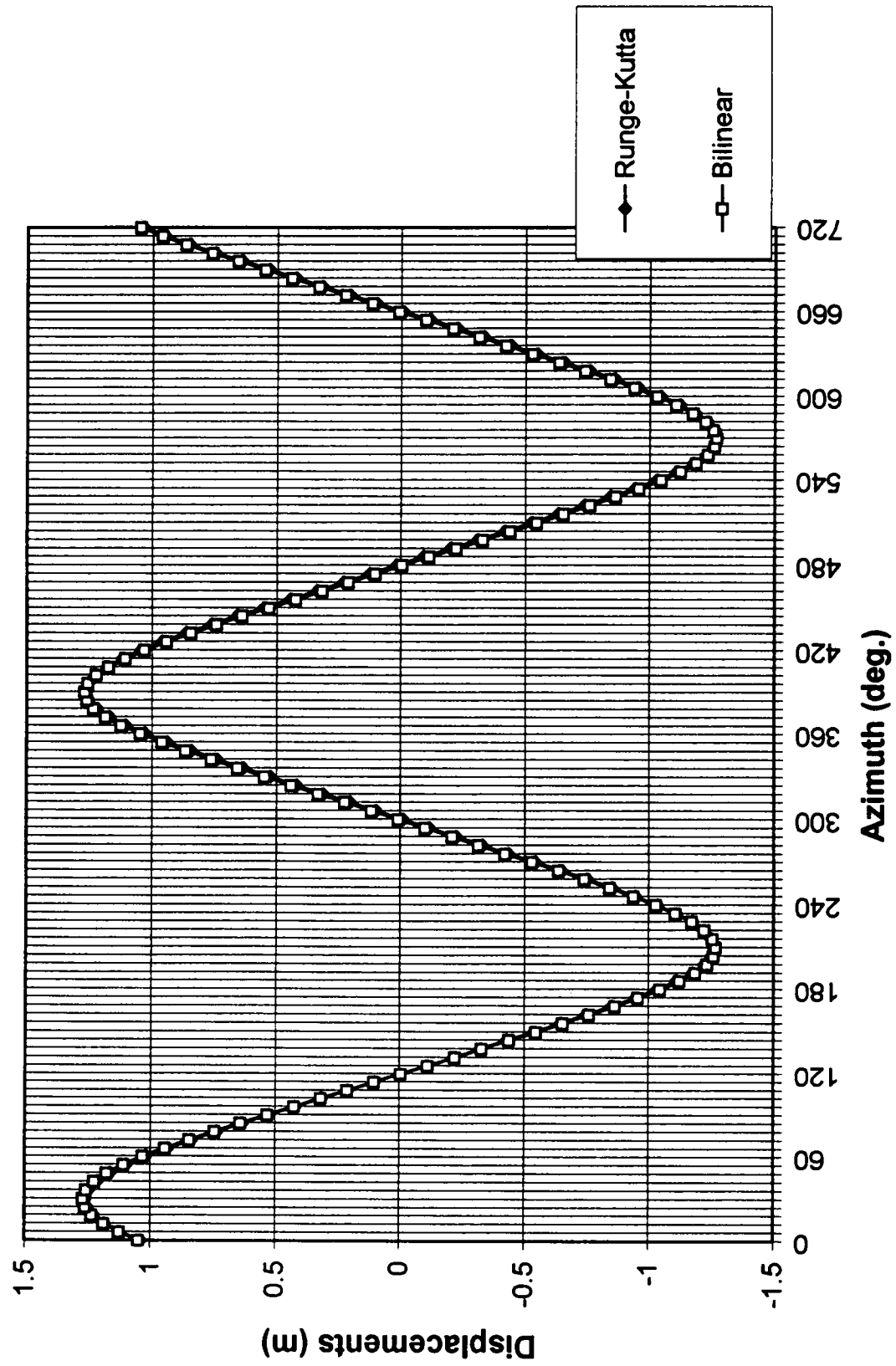


Figure 6. Solution of Duffing's Equation Using the Bilinear Algorithm (1)

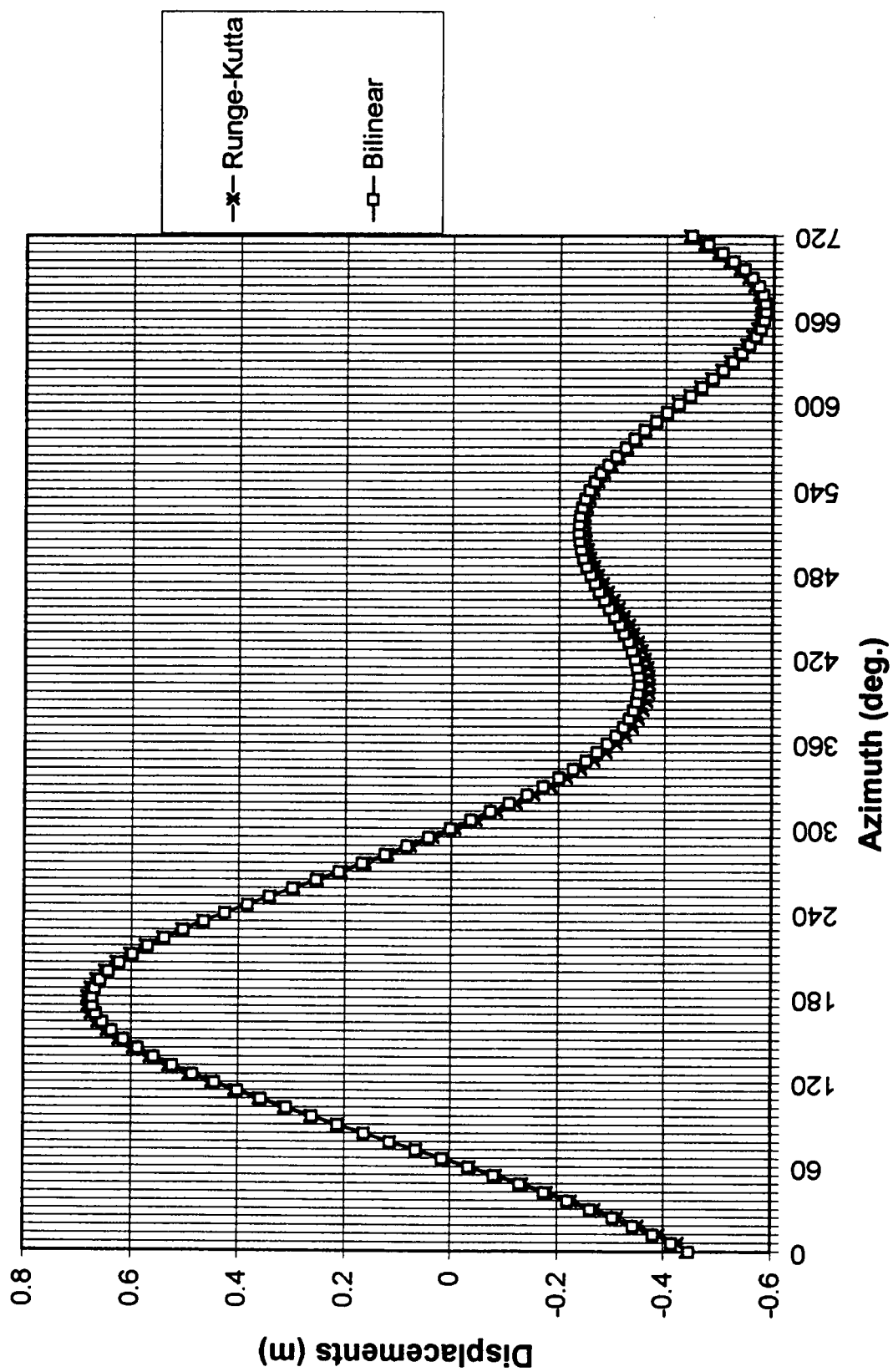


Figure 7. Solution of Duffing's Equation Using the Variational Algorithm (2)

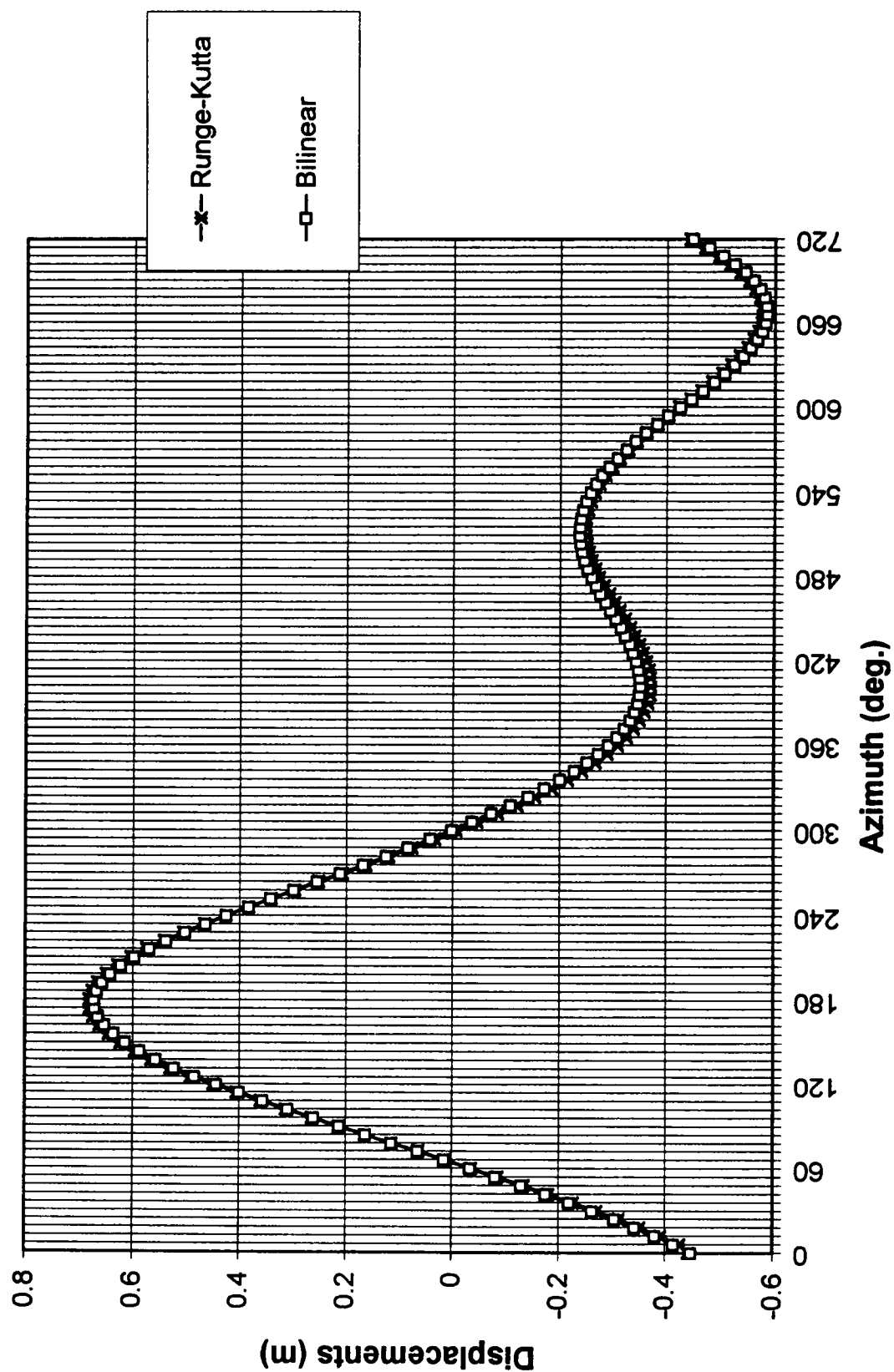


Figure 8. Solution of Duffing's Equation Using the Bilinear Algorithm (2)

CHAPTER 5

RESULTS

In this chapter, both the variational and bilinear approaches to the time finite element method are applied to single and two degrees of freedom nonlinear dynamical systems analysis. Numerical results obtained using both methods as well as CPU times and convergence characteristics are compared. All computations were performed on a Digital vxt 2000 station.

A Single Degree of Freedom System

Algorithms based on variational and bilinear formulations are used to solve Equation (27). This is Duffing's Equation with two initial conditions (a) $q = -0.67$, $\dot{q} = 0.02$ and (b) $q = -0.46$, $\dot{q} = 0.30$. Solutions of this equation for selected initial conditions can be found in Thompson and Stewart (1986). First, Thompson and Stewart's (1986) solutions are reproduced by using a fifth order Runge-Kutta algorithm available in IMSL library. Second, the last two of the five revolutions of these solutions are used as initial conditions for both investigated approaches. Next, it is shown that for constant coefficients of Duffing's Equation, the convergence characteristics of solutions for these nonlinear dynamical mass-damper-spring systems depend on their initial conditions. Comparisons between results obtained using the two investigated approaches are shown in Figures 9 and 10, and Tables 1 and 2.

a. Initial Conditions: $q = -0.67$ and $\dot{q} = 0.02$

Figure 9 shows the comparison between the results obtained using the two investigated methods. Runge-Kutta solution computed for initial conditions $q = -0.67$, $\dot{q} = 0.02$ was used as the initial condition. The length of the solution period equals to the length of two forcing-function periods. Table 1 shows the comparison of their CPU time and convergence characteristics.

Table 1. Solution Characteristics for Initial Conditions $q = -0.67$ and $\dot{q} = 0.02$

Methods	CPU Time (seconds)	Convergence
Bilinear	84. 638	Good agreement with the accurate steady -state solution shown in Figure 4.
Variational	9. 449	

b. Initial Conditions: $q = -0.46$ and $\dot{q} = 0.30$

Figure 10 shows the comparison between variational and bilinear algorithm results computed using the Runge-Kutta solution obtained for initial conditions $q = -0.46$ and $\dot{q} = 0.30$ as the initial condition. Table 2 shows the comparison of their CPU time and convergence characteristics.

Table 2. Solution Characteristics for Initial Conditions $q = -0.46$ and $\dot{q} = 0.30$

Methods	CPU Time (seconds)	Convergence
Bilinear	115. 642	Good agreement with the accurate steady -state solution shown in Figure 4.
Variational	9. 606	

A Two-Degree-of-Freedom System

Both approaches are applied to solve a system of equations used in this study to model the following two-degree-of-freedom nonlinear dynamical systems: a mass-damper-

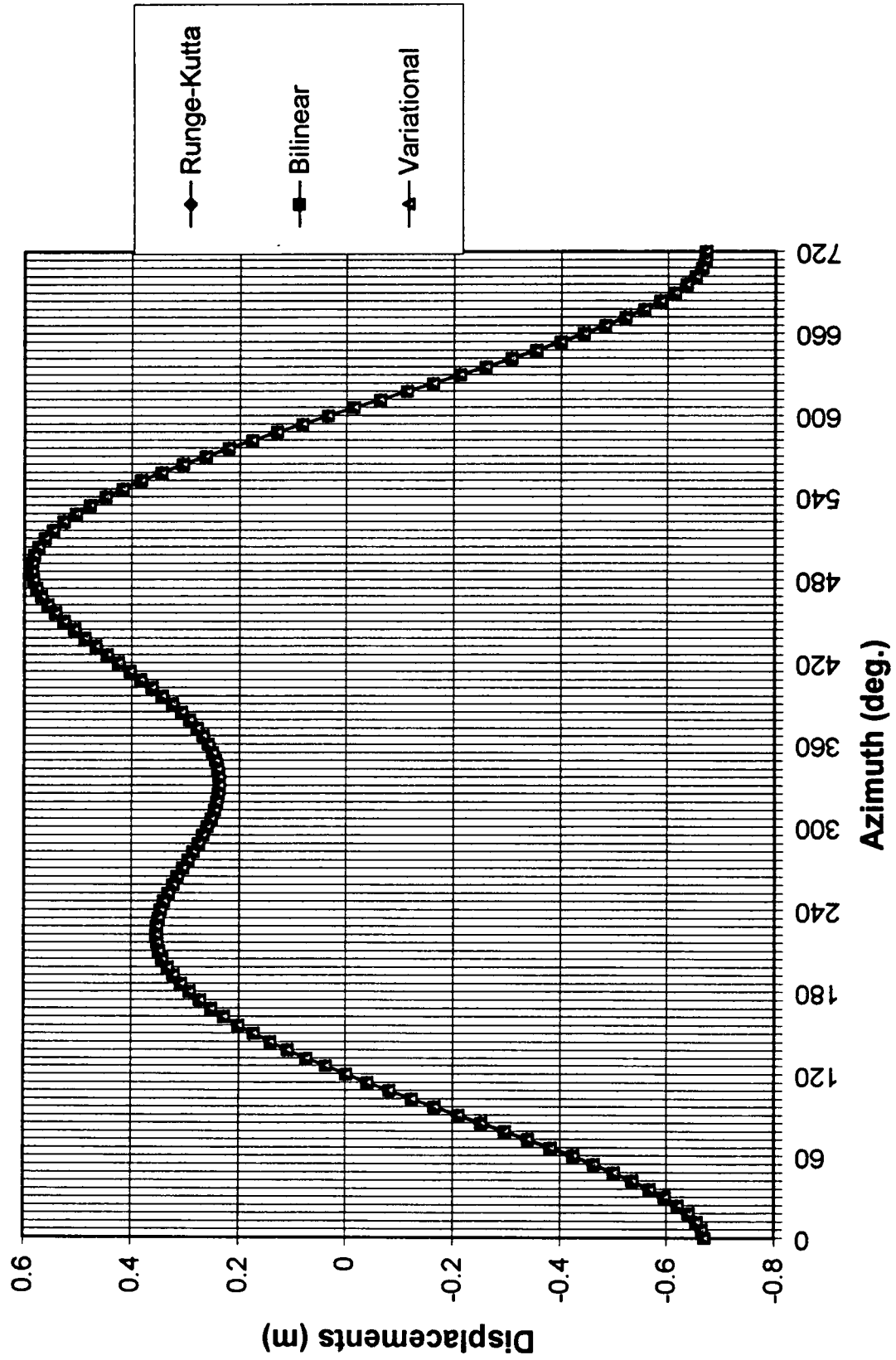


Figure 9. Comparison of Results for Initial Conditions $q = -0.67$ and $q = 0.02$

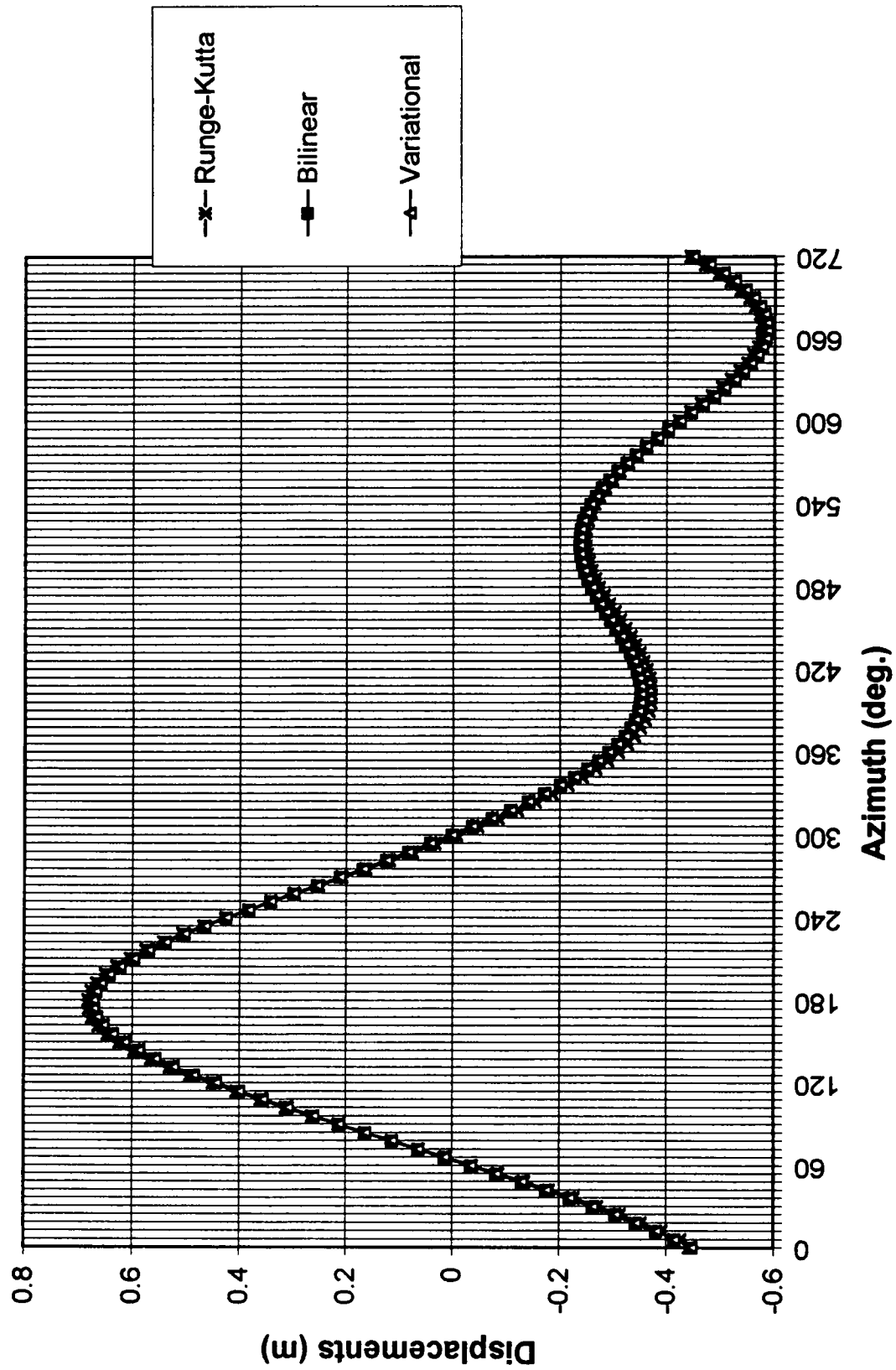


Figure 10. Comparison of Results for Initial Conditions $q = -0.46$ and $q = 0.30$

spring system and a rigid body representation of a helicopter rotor blade.

A Mass-Damper-Spring System

The two investigated algorithms are used to obtain a solution for Equations (40) and (41). First, the solution of a linearized system is obtained using the bilinear algorithm. This was done by neglecting nonlinear terms in inertia, damping, stiffness and forcing function matrices. The displacement-velocity solution for the end of one revolution is used as the set of initial conditions for the Runge-Kutta algorithm. Runge-Kutta solution over the first revolution (the transient) solution was used as the initial condition for both investigated algorithms. The comparison between resulting solutions is shown in Figure 11 and Table 3. Both algorithms show a good agreement with a high precision Runge-Kutta solution (tolerance for error control = 0.1 E-09).

Table 3. Comparison between Results Obtained Using Variational and Bilinear Approaches for Constant Coefficients a and b in Equations (40) and (41)

Methods	CPU Time (seconds)	Convergence
Bilinear	8.345	Good agreement with Runge-Kutta solutions.
Variational	1.659	Good agreement with Runge-Kutta solutions.

Figures 12 and 13 show steady-state solutions obtained using the Runge-Kutta and both investigated algorithms plotted over five revolutions. To gain an insight into the convergence characteristics of both investigated algorithms, they are used to solve Equations (40) and (41) for different stiffness-modifying coefficients a and b . The parameters used in Equations (40) and (41) are as follows :

$$\begin{array}{lll}
 m_1 = 10.0 \text{ kg} & Q_1(\psi) = F_0 \cos(q_1) \cos(\omega t) \text{ N} & k_1 = a_1 q_1 + b_1 \text{ N/m} \\
 m_2 = 15.0 \text{ kg} & Q_2(\psi) = 0.0 & k_2 = a_2 q_2 + b_2 \text{ N/m}
 \end{array}$$

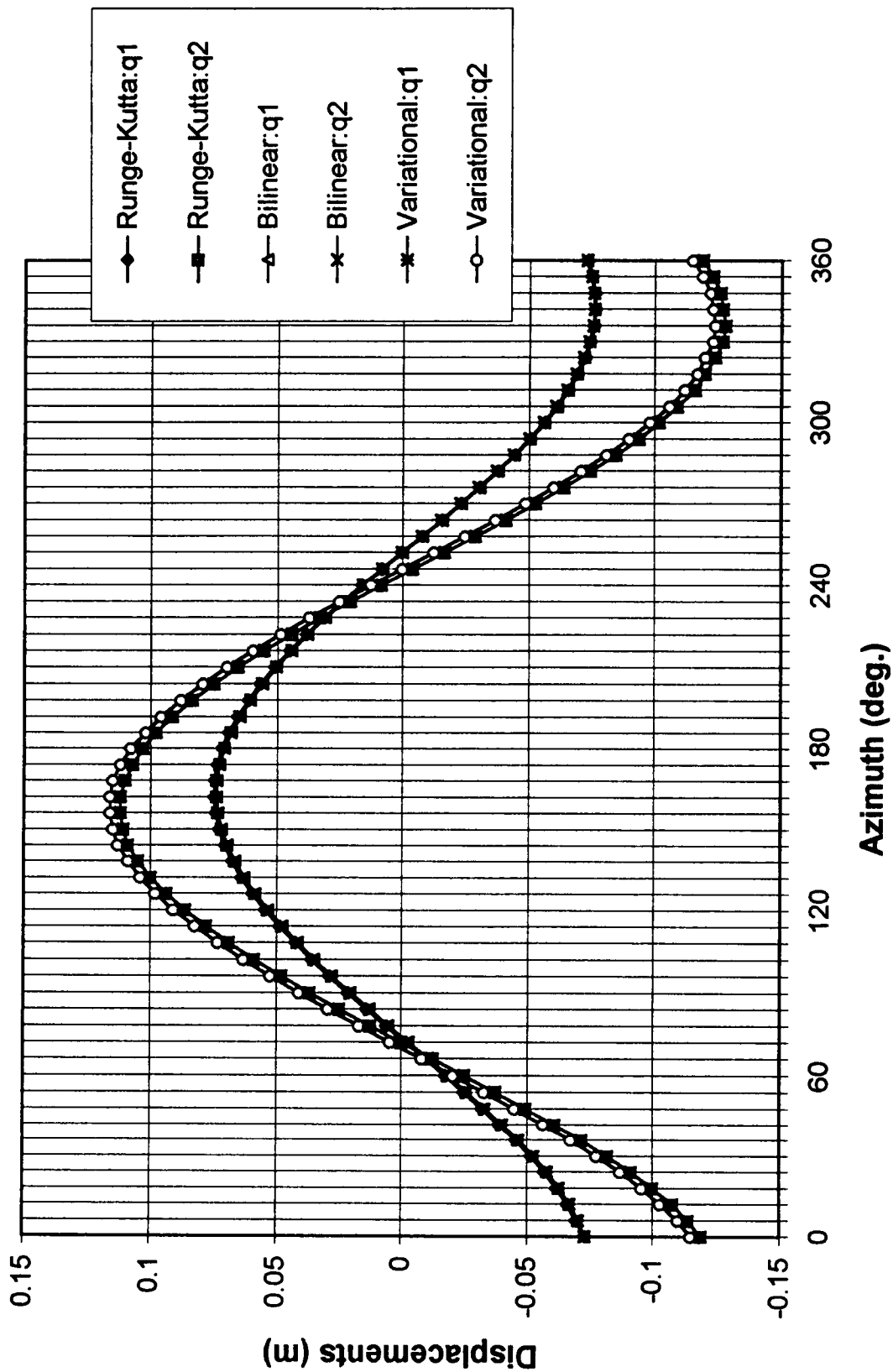


Figure 11. Comparison of Results for a Two-Degree-of-Freedom System

Displacements vs. Azimuth over 5 Revolutions

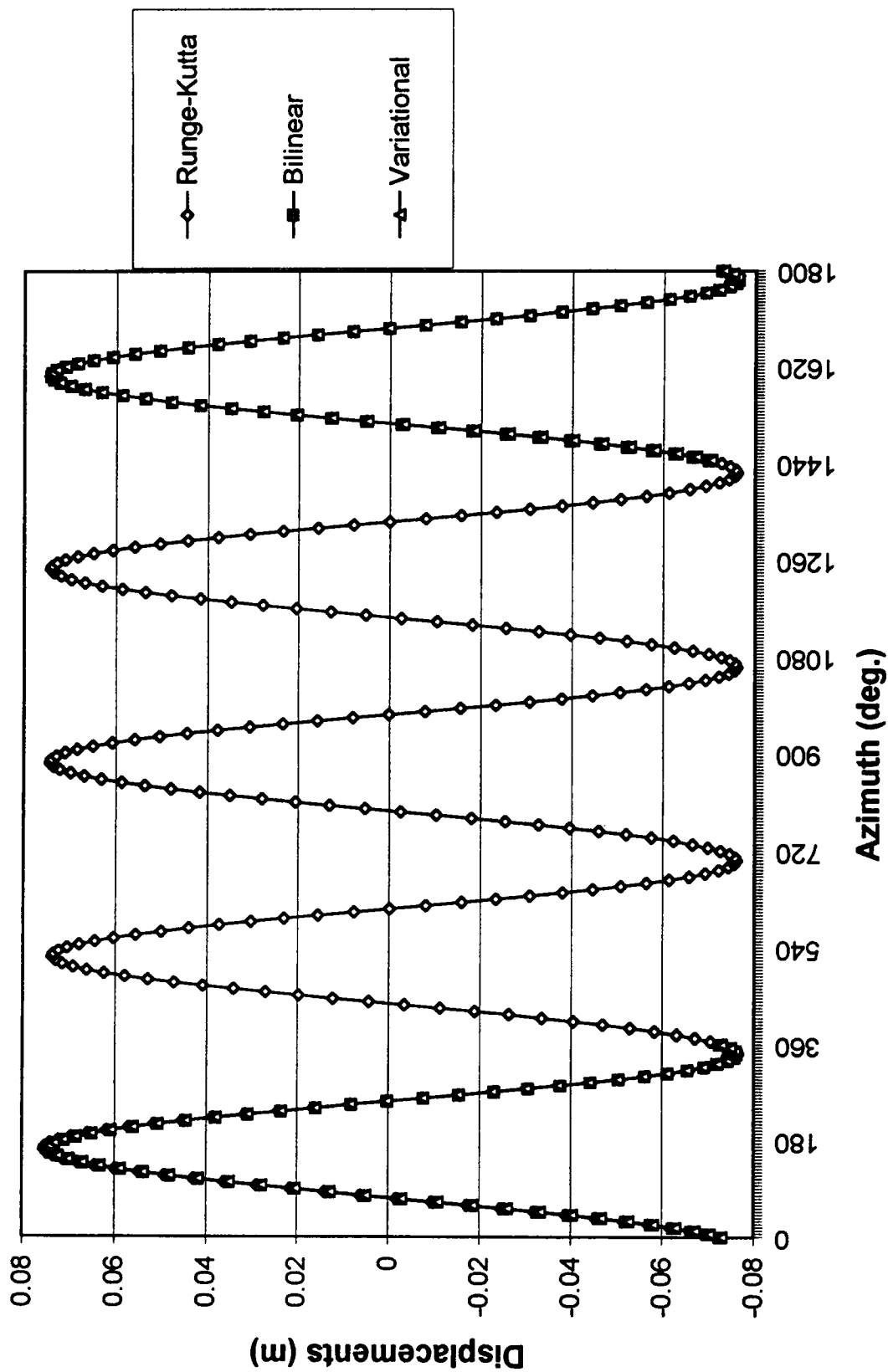


Figure 12. Comparison of Results for Displacements q1

Displacements vs. Azimuth over 5 Revolutions

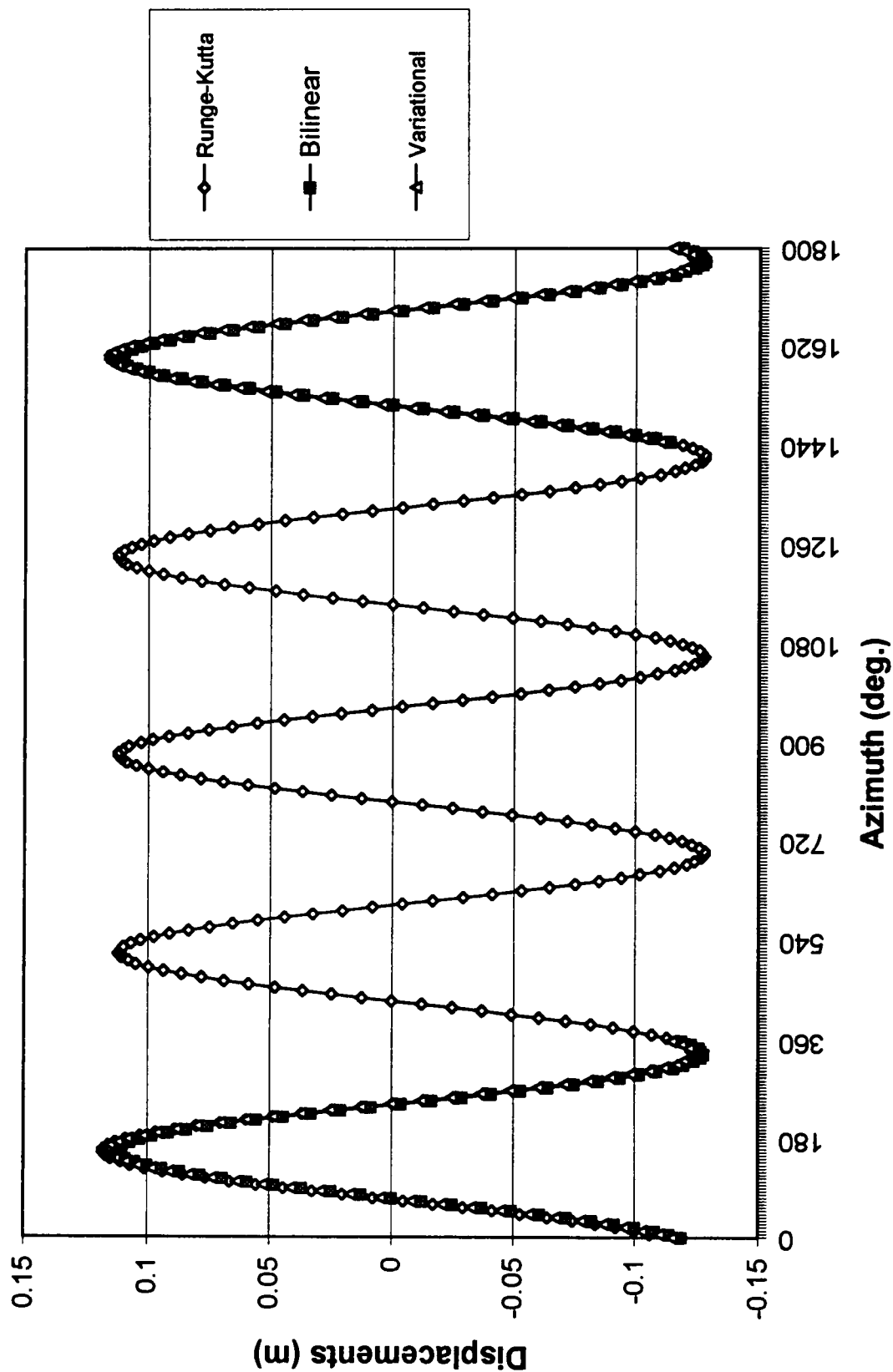


Figure 13. Comparison of Results for Displacements q2

$$c_1 = 0.0 \text{ Ns/m}$$

$$\omega = 1.0 \text{ rad/s}$$

$$c_2 = 10.0 \text{ Ns/m}$$

where a_1 , b_1 , a_2 , b_2 and F_0 for different runs are given in Table 4. Results of this study are also given in Table 4.

Table 4. Comparison between Results Obtained Using Variational and Bilinear Approaches for Variable Coefficients a , b and F_0 in Equations (40) and (41)

No	a_1	b_1	a_2	b_2	F_0	CPU Time (seconds)		Comments
						Bilinear	Variational	
1	0.2	10	1	2	80	121.95	N. C.	N. C. = Not Converged ¹⁾
2	0.2	10	1	2	10	37.159	4.764	
3	5	20	5	50	40	58.89	N. C.	
4	5	20	5	3	40	168.086	N. C.	
5	5	20	5	4	40	59.064	N. C.	
6	5	20	5	60	40	85.426	N. C.	
7	5	20	5	80	40	58.474	N. C.	
8	5	20	4	5	40	57.673	9.533	
9	5	20	3	50	40	58.312	29.340*	*Converged to different steady-state solutions.
10	5	20	1	50	40	246.141	44.931*	
11	5	40	3	50	40	N. C.	N. C.	
12	5	40	3	25	40	161.365	N. C.	
13	5	20	3	25	40	49.397	8.27	
14	10	20	5	25	40	87.609	8.386	
15	10	20	10	25	40	156.749	10.679	
16	13	20	10	25	40	N. C.	N. C.	
17	13	20	7	25	40	122.886	13.012	

1) The solution is assumed to be converged if $\sqrt{\sum_{i=1}^{60} q_i^2} < 0.1\text{E-}03$ for less than 500 iterations.

Table 4 shows that the parameters of Equations (40) and (41) affect strongly convergence characteristics for both algorithms. The Runge-Kutta method usually produces a converged solution after marching through four to five periods. Figure 14 shows the results determined using the variational method and bilinear method compared against the transient solution (over the first period) obtained using the Runge-Kutta algorithm for case number 2 of Table 4. Imposed periodicity algorithms seem to converge to a steady-state solution different than the solution obtained using the Runge-Kutta algorithm. Figure 15 shows a comparison between results determined using both investigated methods and the converged results obtained using the Runge-Kutta algorithm. The agreement is excellent. In this case, the results obtained using the Runge-Kutta algorithm over the fifth revolution are shown.

Now, let us look at cases 1 and 3 of Table 4. Figure 16 and 18 show a comparison between bilinear solutions and solutions obtained using the Runge-Kutta algorithm for the cases number 1 and 3 of Table 4. Note that for those sets of parameters the variational method did not converge at all. Figure 16 shows the comparison among the solutions obtained using the bilinear algorithm the steady-state solutions obtained using Runge-Kutta and Gear's algorithms. Although the steady-state solutions obtained using the bilinear method and Runge-Kutta and Gear's algorithm are different, all of them are correct. According to Thompson and Stewart (1986), the solution of a nonlinear, ordinary differential equation with a linear, periodic forcing function depends on the initial conditions. The different solutions shown in Figure 16 correspond, therefore, to different initial conditions. Since we have no way to find out what initial conditions would give a Runge-Kutta solution identical to the

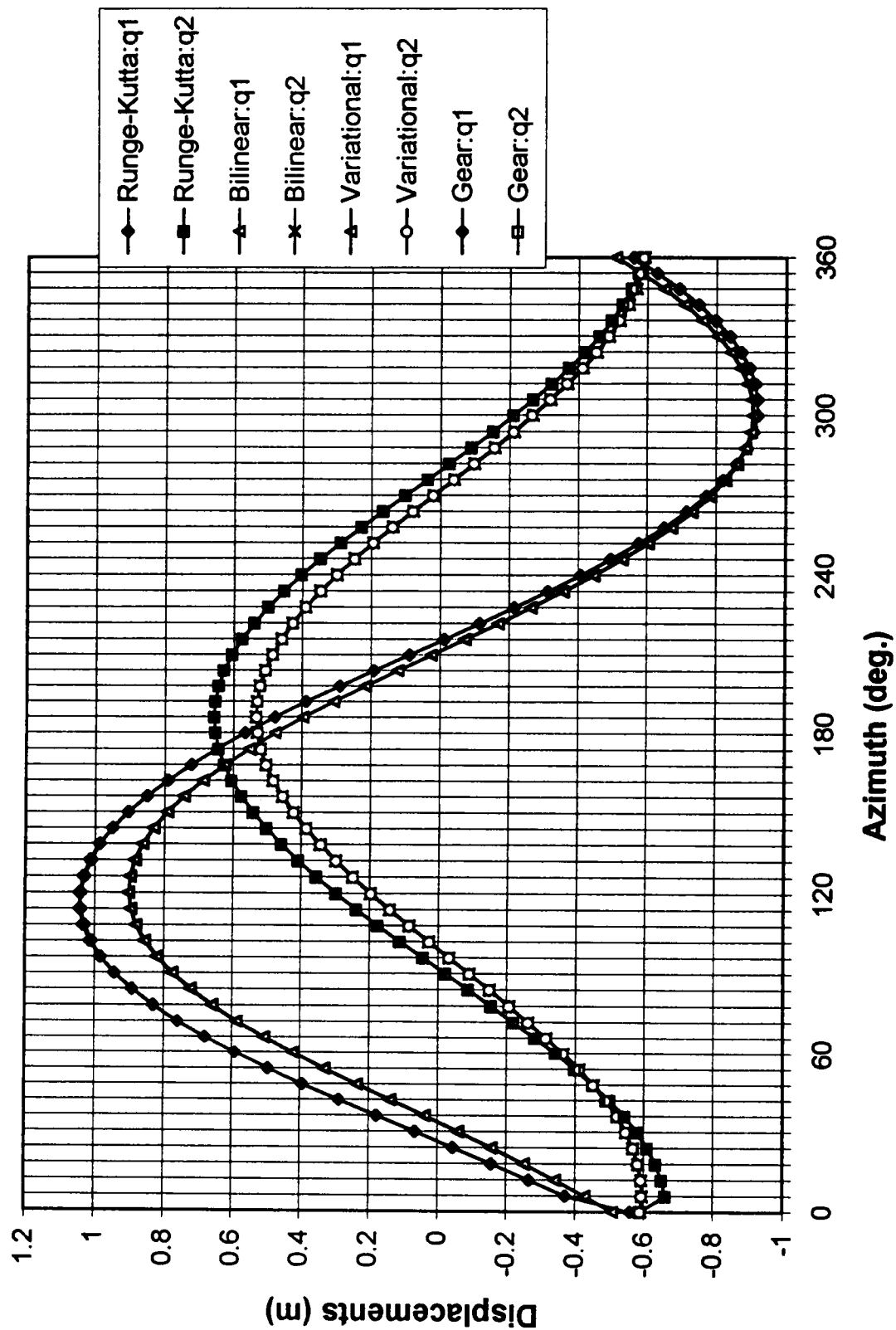


Figure 14. Variational and Bilinear Results vs. Transient Runge-Kutta and Gear Solutions for Case 2 of Table 4

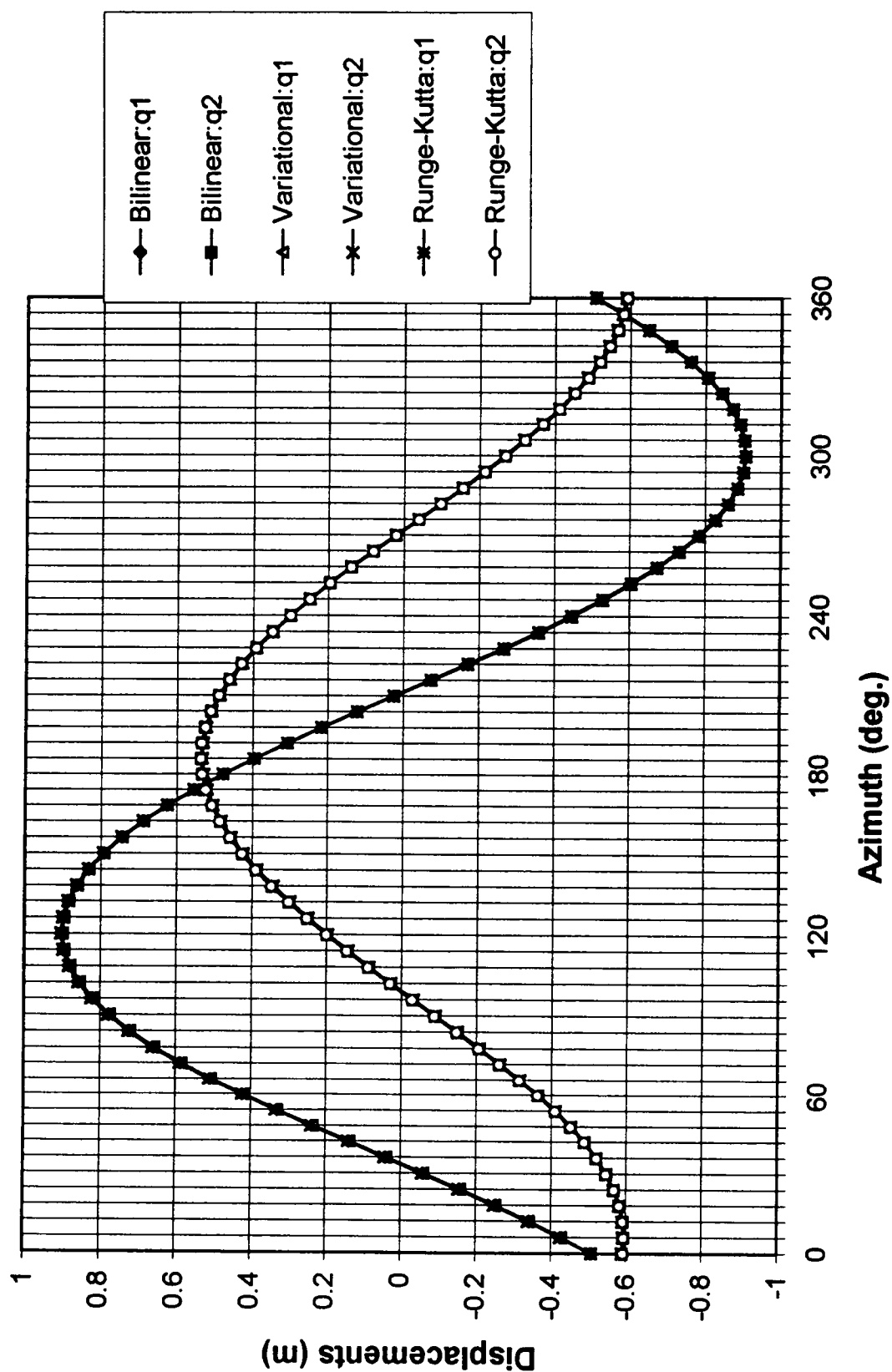


Figure 15. Variational and Bilinear Solutions vs. Converged Runge-Kutta Solutions for Case 2 of Table 4

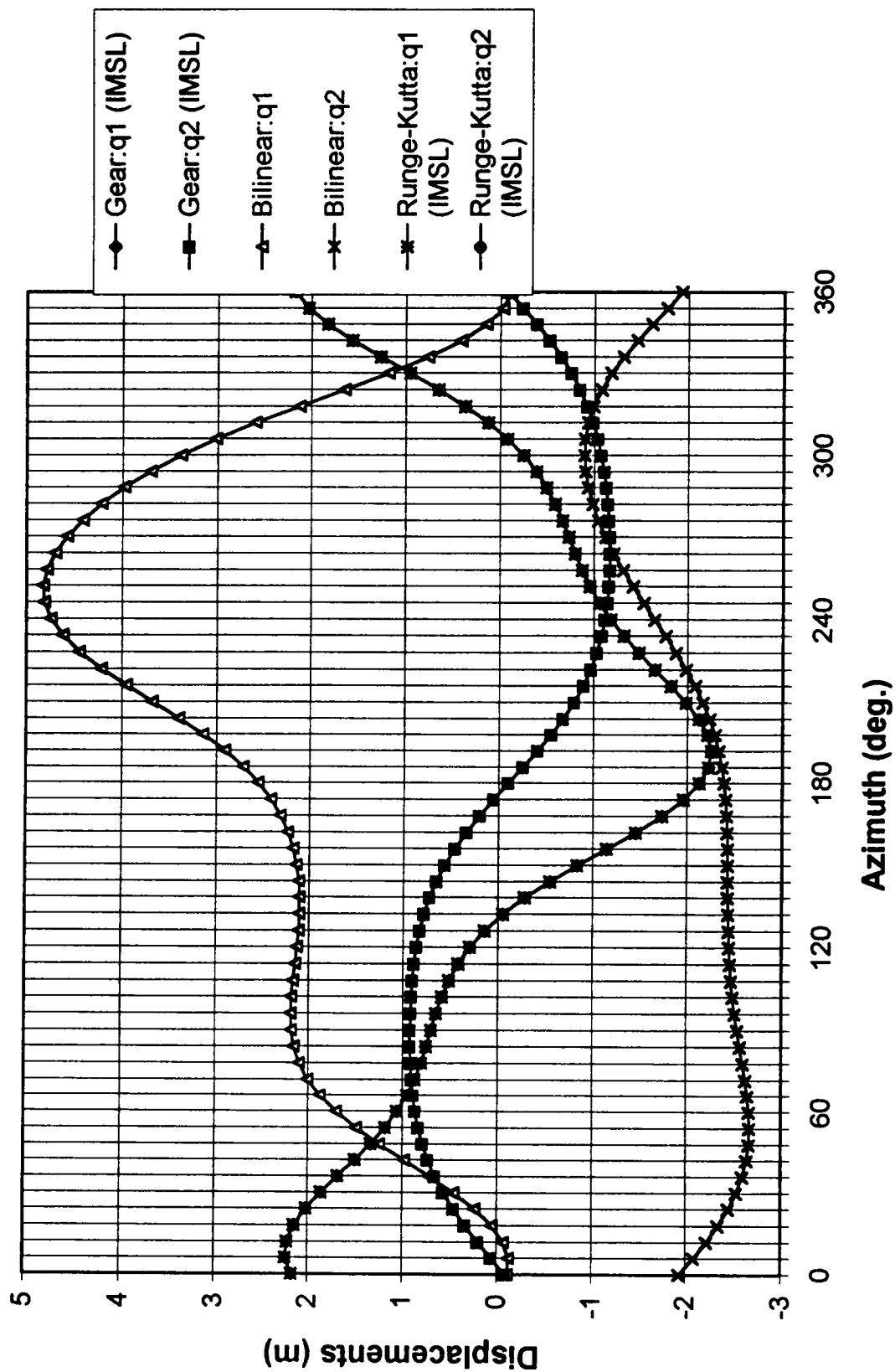


Figure 16. Bilinear Results vs. Converged Runge-Kutta and Gear Solutions for Case 1 of Table 4

bilinear solution, we reverse the problem and we use the converged Runge-Kutta solution as the initial condition for the bilinear algorithm. The steady-state solution obtained in this way was identical with the converged Runge-Kutta solution, as shown in Figure 17. We can notice that in this case the bilinear algorithm initially converged to a different steady-state solution because it was fed with a different initial guess. Figure 18 shows that the solutions obtained by the bilinear and Runge-Kutta algorithms are identical for case number 3 of Table 4. In this case, apparently, the initial conditions used for the Runge-Kutta program correspond to the steady-state solution obtained using the bilinear algorithm with the solution of a linearized system as an initial guess. Figure 19 shows results obtained using both investigated methods against results obtained using the Runge-Kutta algorithm for the case number 10 of Table 4. The results obtained by the variational algorithm converged to a steady-state solution different than the results obtained by the bilinear and Runge-Kutta algorithms. The bilinear solution was in good agreement with the results obtained using the Runge-Kutta algorithm. This is another case of different, but equally legitimate steady-state solutions. In order to show that the variational algorithm could also produce the converged Runge-Kutta solution, the latter was used as an initial guess for the program based on the variational method. The converged results obtained in this way were in excellent agreement with the Runge-Kutta solution, as shown in Figure 20.

A Rigid Blade Model

A solution of a system of nonlinear differential equations modeling a rigid helicopter blade

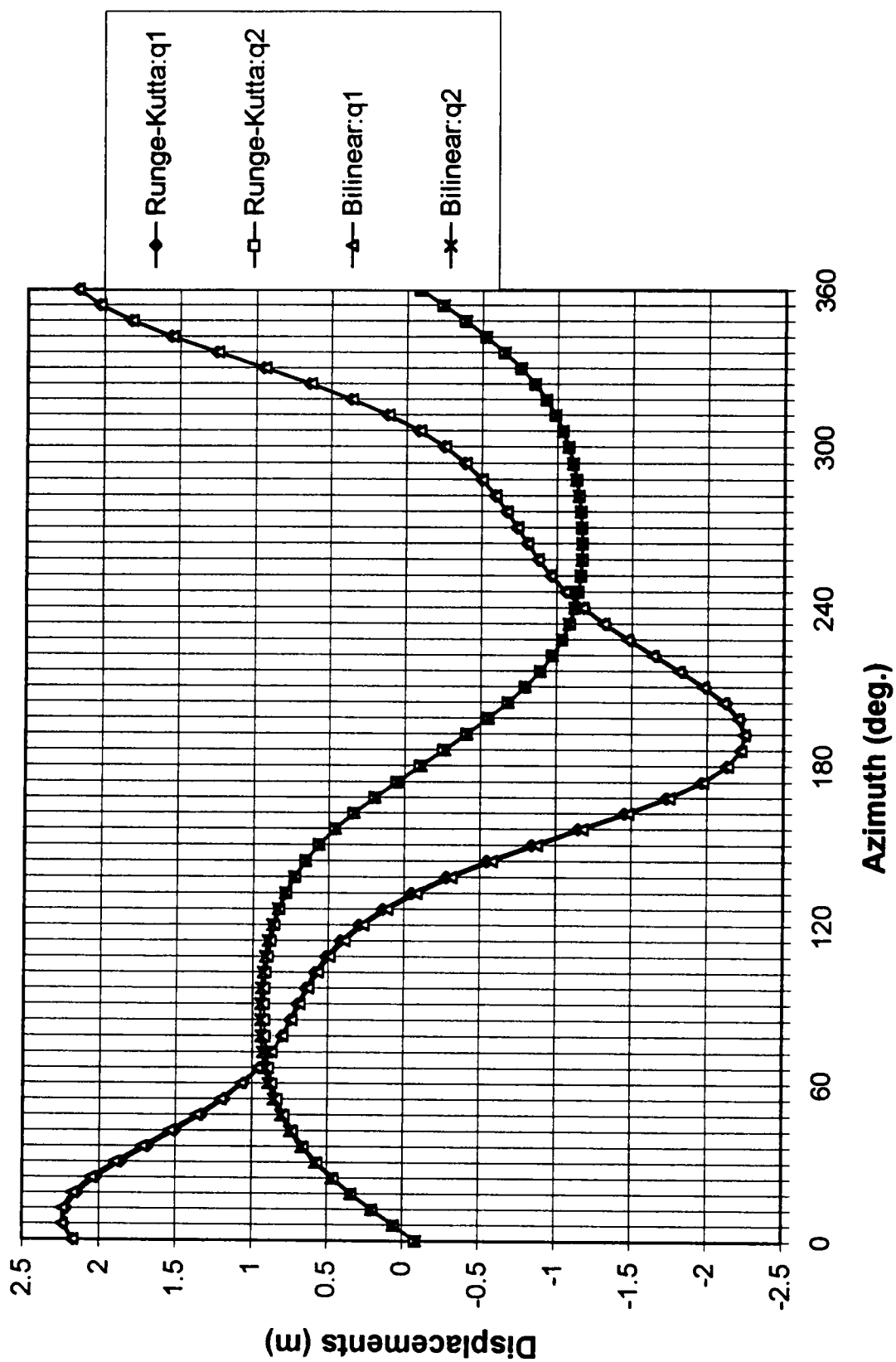


Figure 17. Bilinear Solutions with Initial Conditions from Converged Runge-Kutta Solutions for Case 1 of Table 4

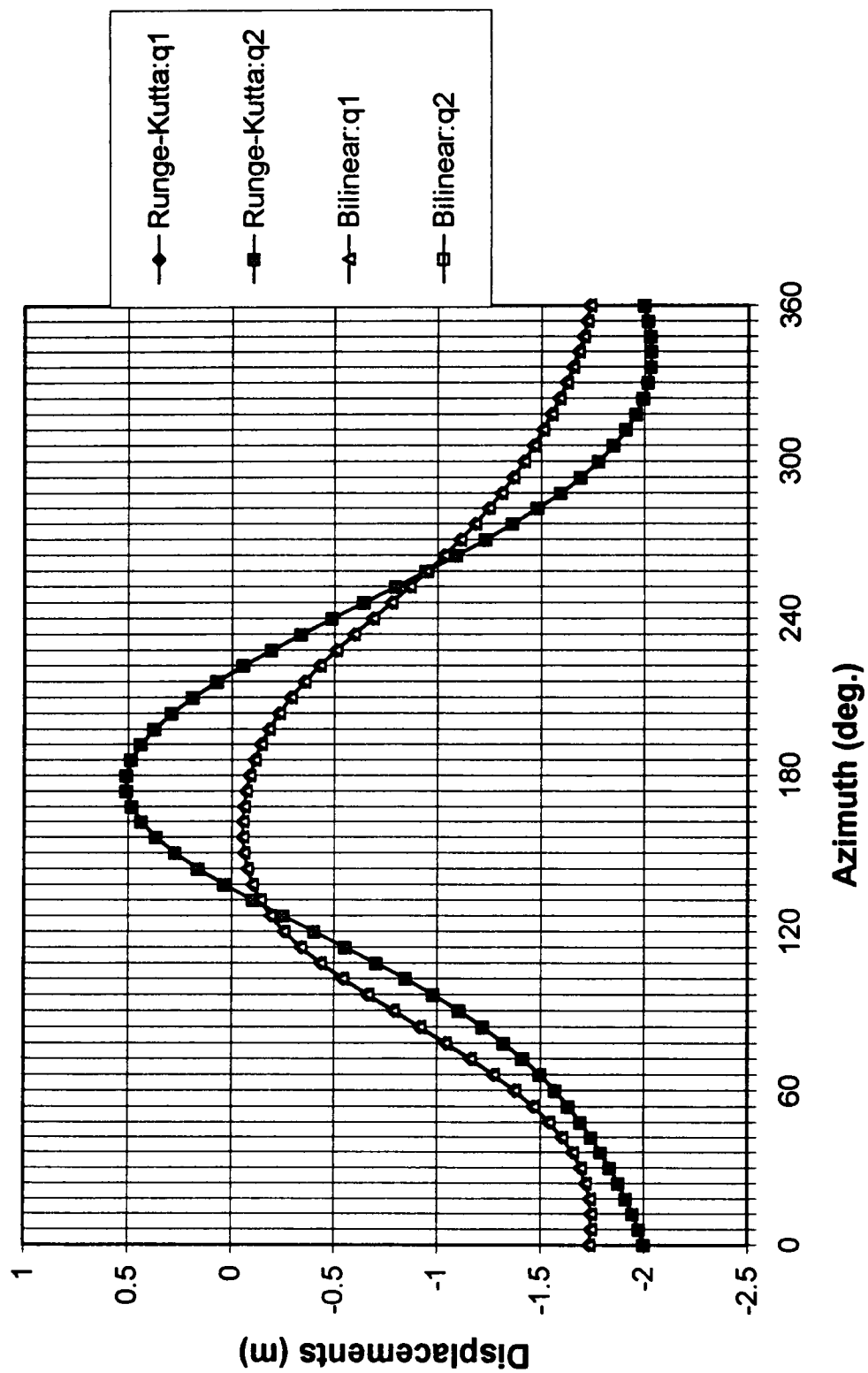


Figure 18. Bilinear Solutions vs. Runge-Kutta Solutions for Case 3 of Table 4

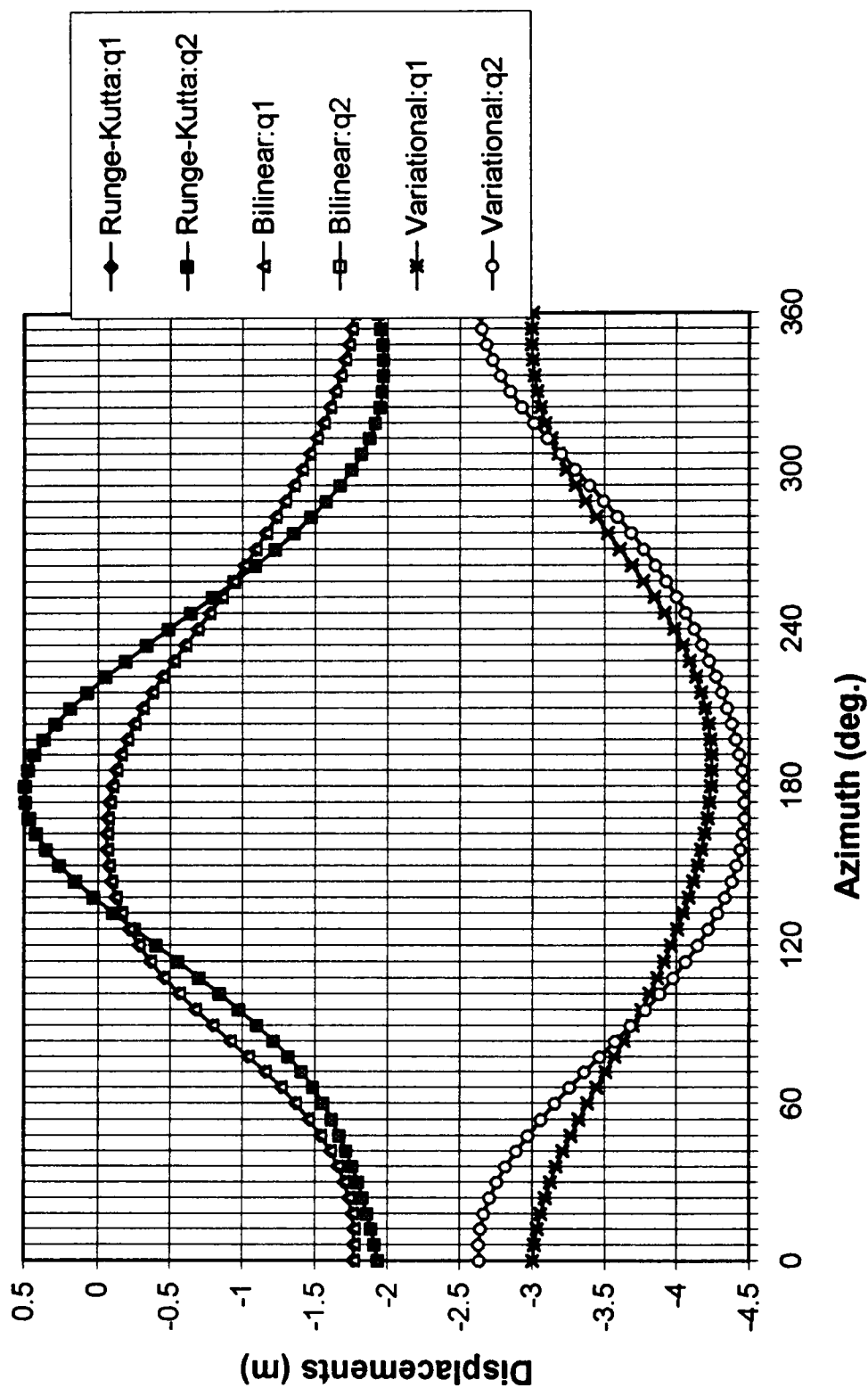


Figure 19. Solutions of Investigated Methods vs. Runge-Kutta Solutions for Case 10 of Table 4

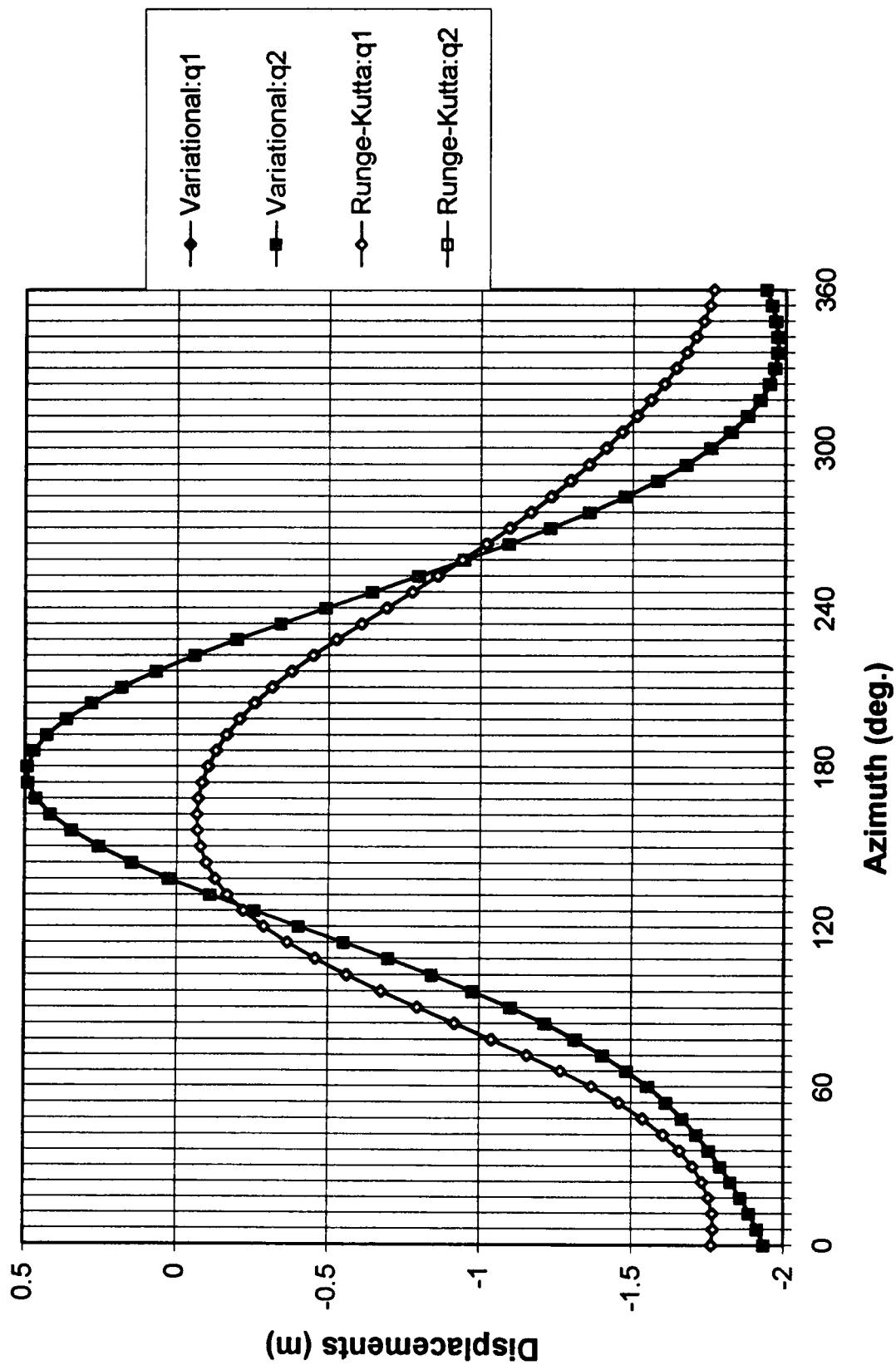


Figure 20. Variational Solutions with Initial Conditions from Converged Runge-Kutta for Case 10 of Table 4

is attempted using both the bilinear and variational methods. A set of initial conditions for the Runge-Kutta algorithm is obtained using the linearized bilinear algorithm. Then, the transient Runge-Kutta solution is used as the initial condition for both methods.

Solutions were produced for $\mu = 0.1$, $\mu = 0.2$ and $\mu = 0.3$. The program based on the variational method failed to produce converged results. That is why Figures 21 to 26 show comparisons between only two solutions: one obtained using the Runge-Kutta method and the other produced using the bilinear algorithm. The corresponding CPU times for $\mu = 0.3$ were 14.457 seconds for Runge-Kutta algorithm and 330.230 seconds for the bilinear algorithm. CPU times for $\mu = 0.1$ and $\mu = 0.2$ were very similar to those observed for $\mu = 0.3$.

The results obtained using the Runge-Kutta method and the bilinear algorithm are different, but considering our experience with the mass-damper-spring system analysis, we know that they represent two correct solutions for this problem. However, it is evident that the Runge-Kutta solution has a period equal to two period lengths of the forcing function whereas the period of the bilinear solution is equal to the period of the forcing function. The practical experience shows, see Friedmann and Kottapalli (1982), that the period of a helicopter blade motion is equal to the period of the forcing function-aerodynamical disturbance due to the rotation of the rotor. Therefore, the bilinear solution is closer to the physical reality.

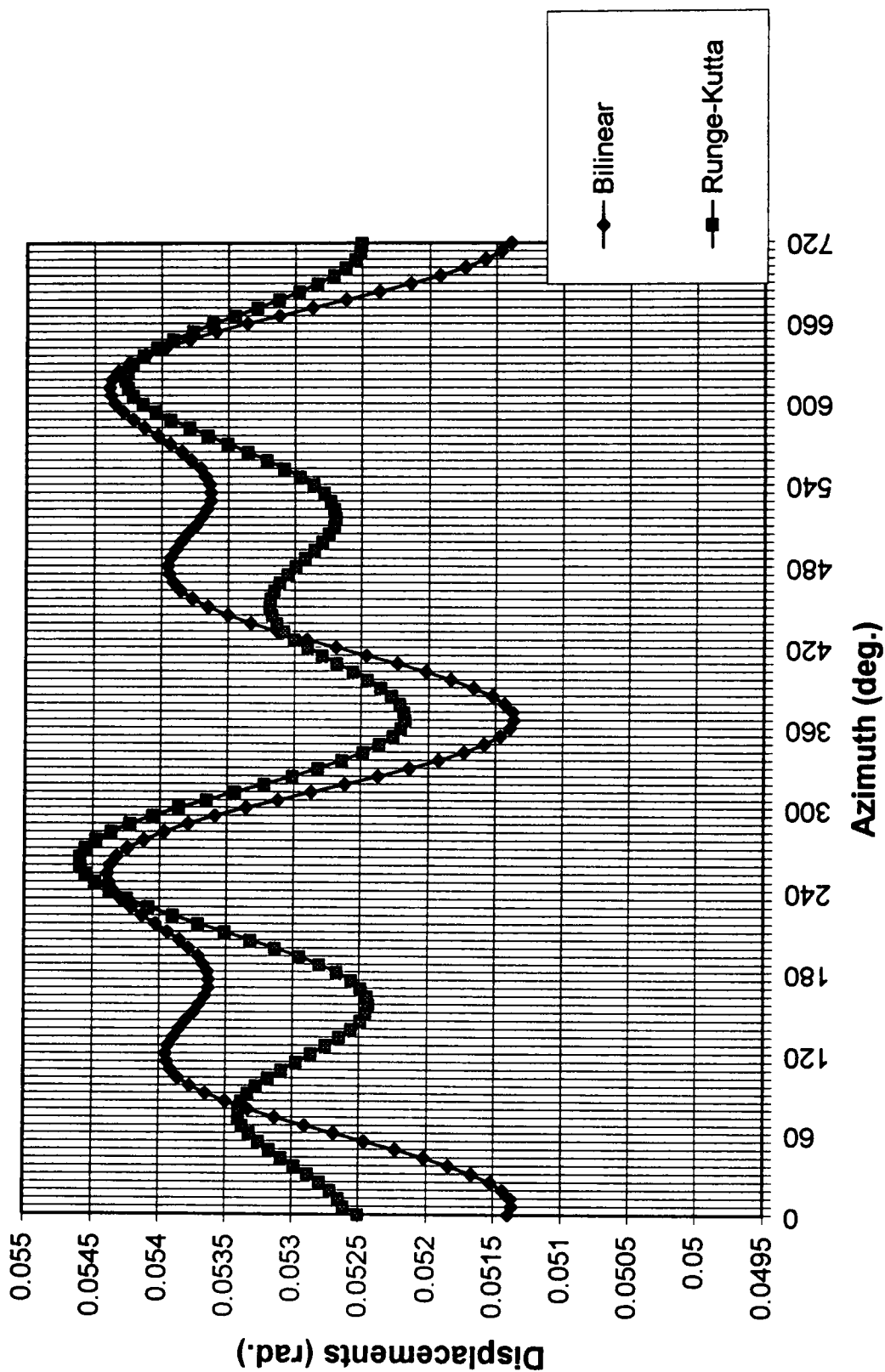


Figure 21. Runge-Kutta vs. Bilinear Solutions for Beta (Flapping) Displacements
(Advance Ratio 0.1)

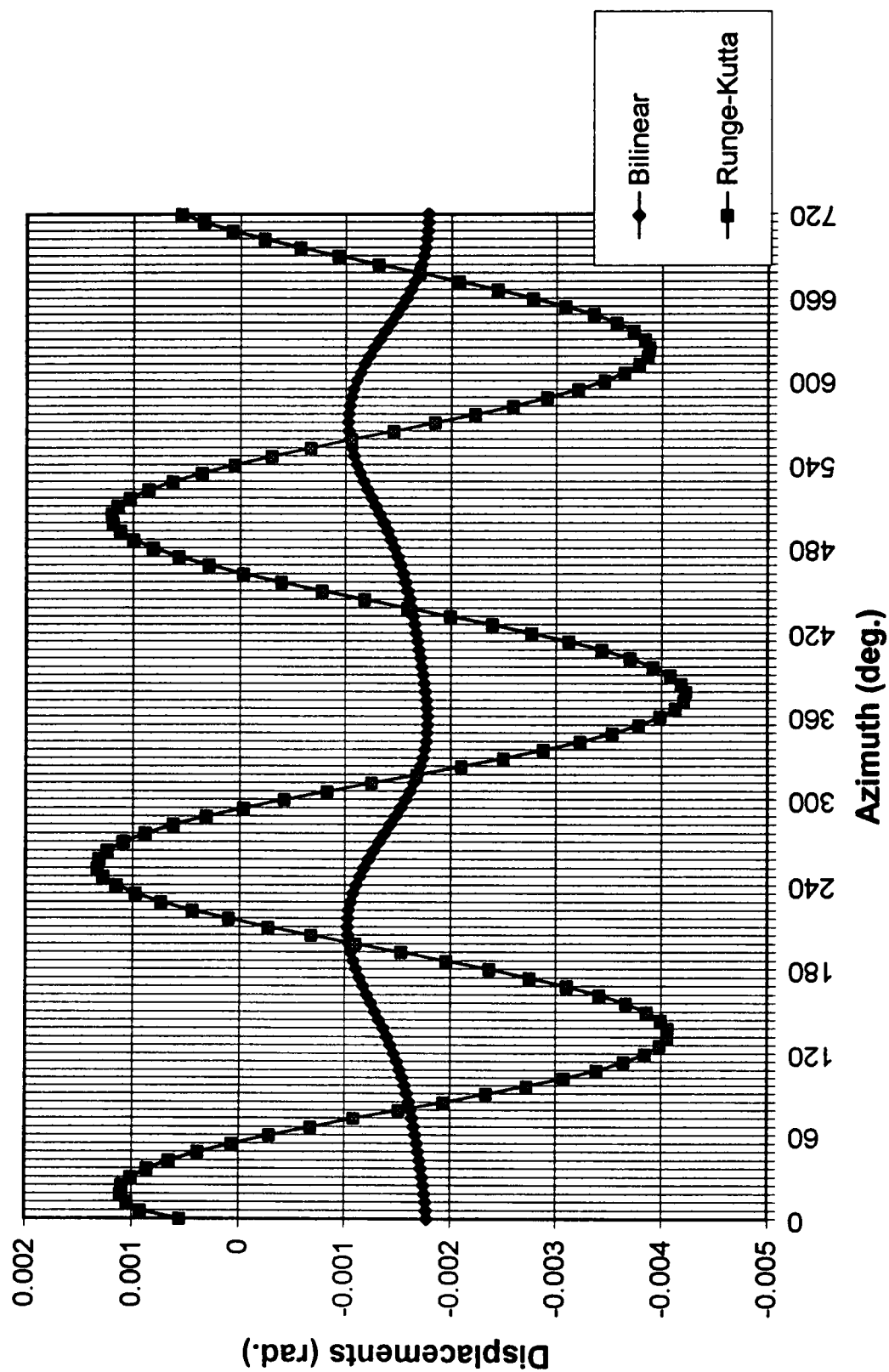


Figure 22. Runge-Kutta vs. Bilinear Solutions for Zeta (Lead-Lag) Displacements
(Advance Ratio 0.1)

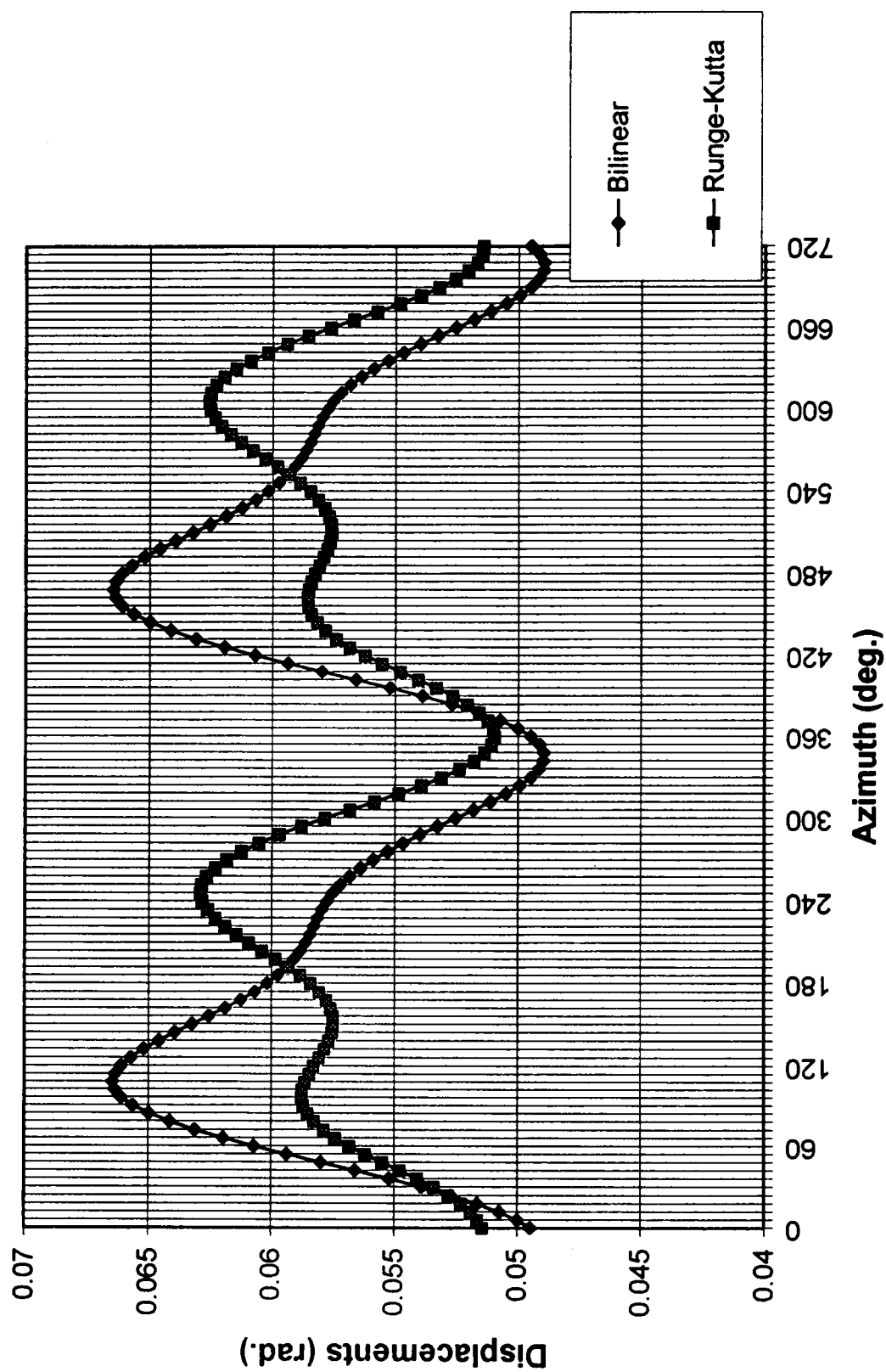


Figure 23. Runge-Kutta vs. Bilinear Solutions for Beta (Flapping) Displacements
(Advance Ratio 0.2)

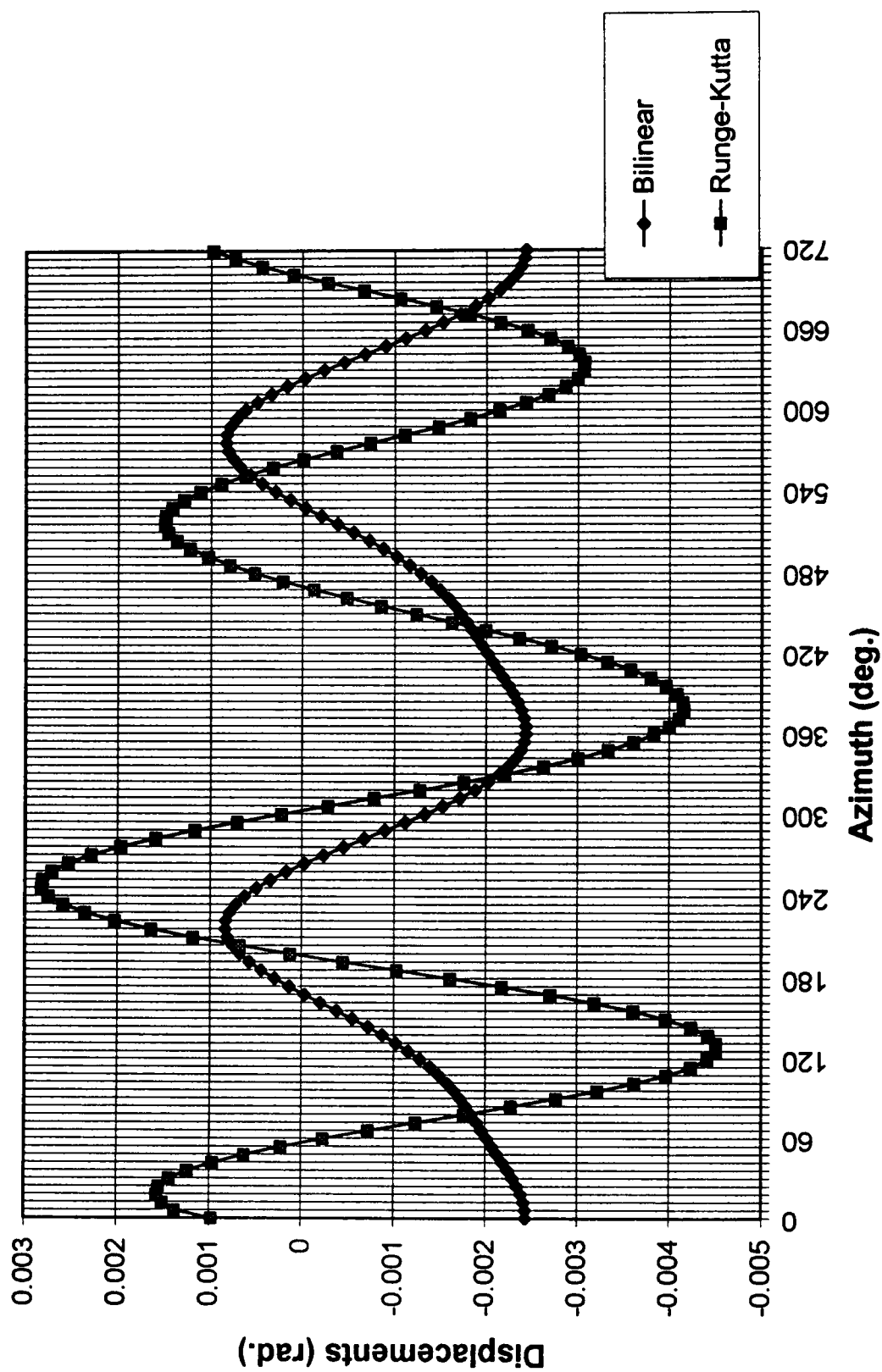


Figure 24. Runge-Kutta vs. Bilinear Solutions for Zeta (Lead-Lag) Displacements
(Advance Ratio 0.2)

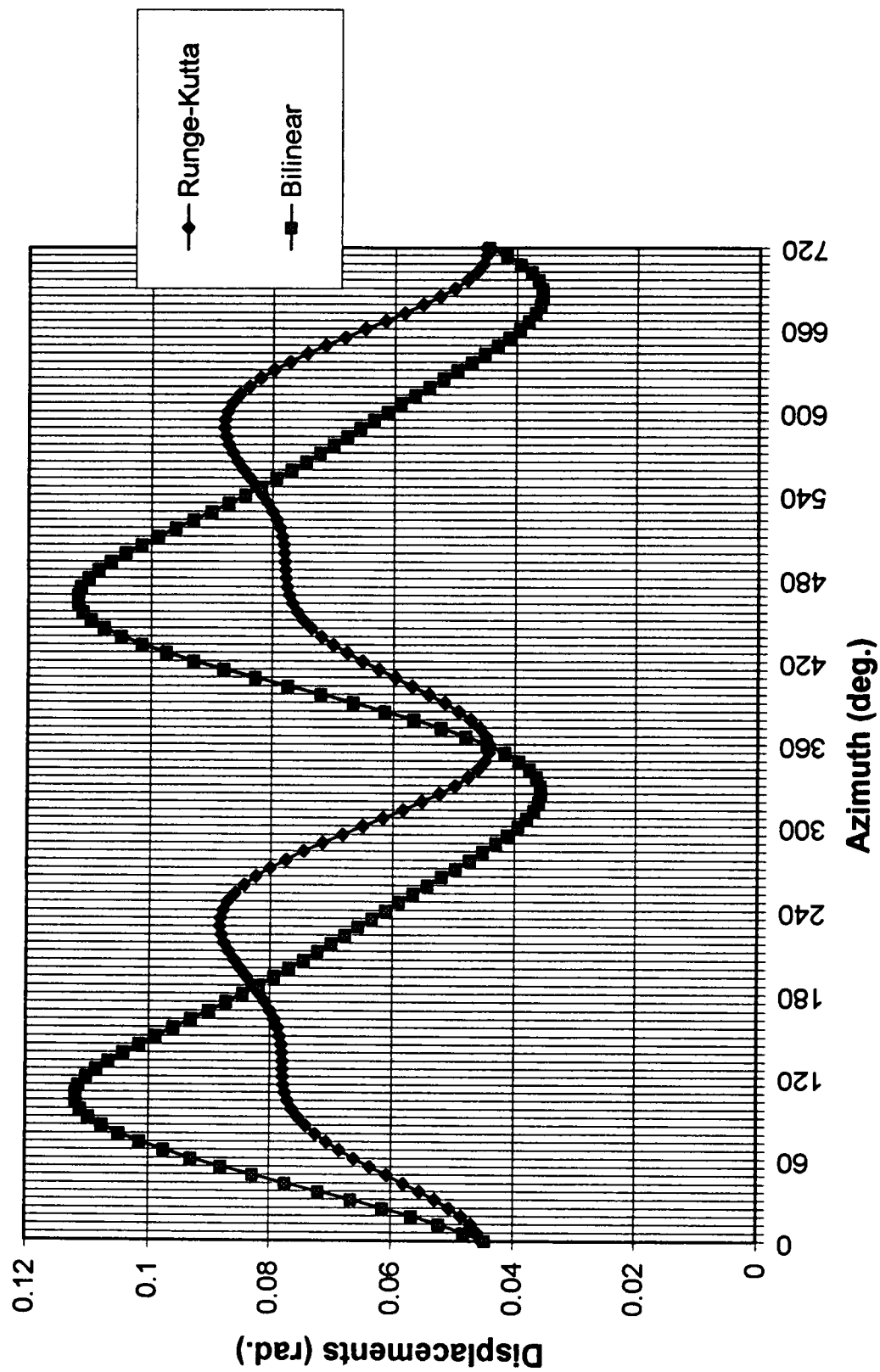


Figure 25. Runge-Kutta vs. Bilinear Solutions for Beta (Flapping) Displacements
(Advance Ratio 0.3)

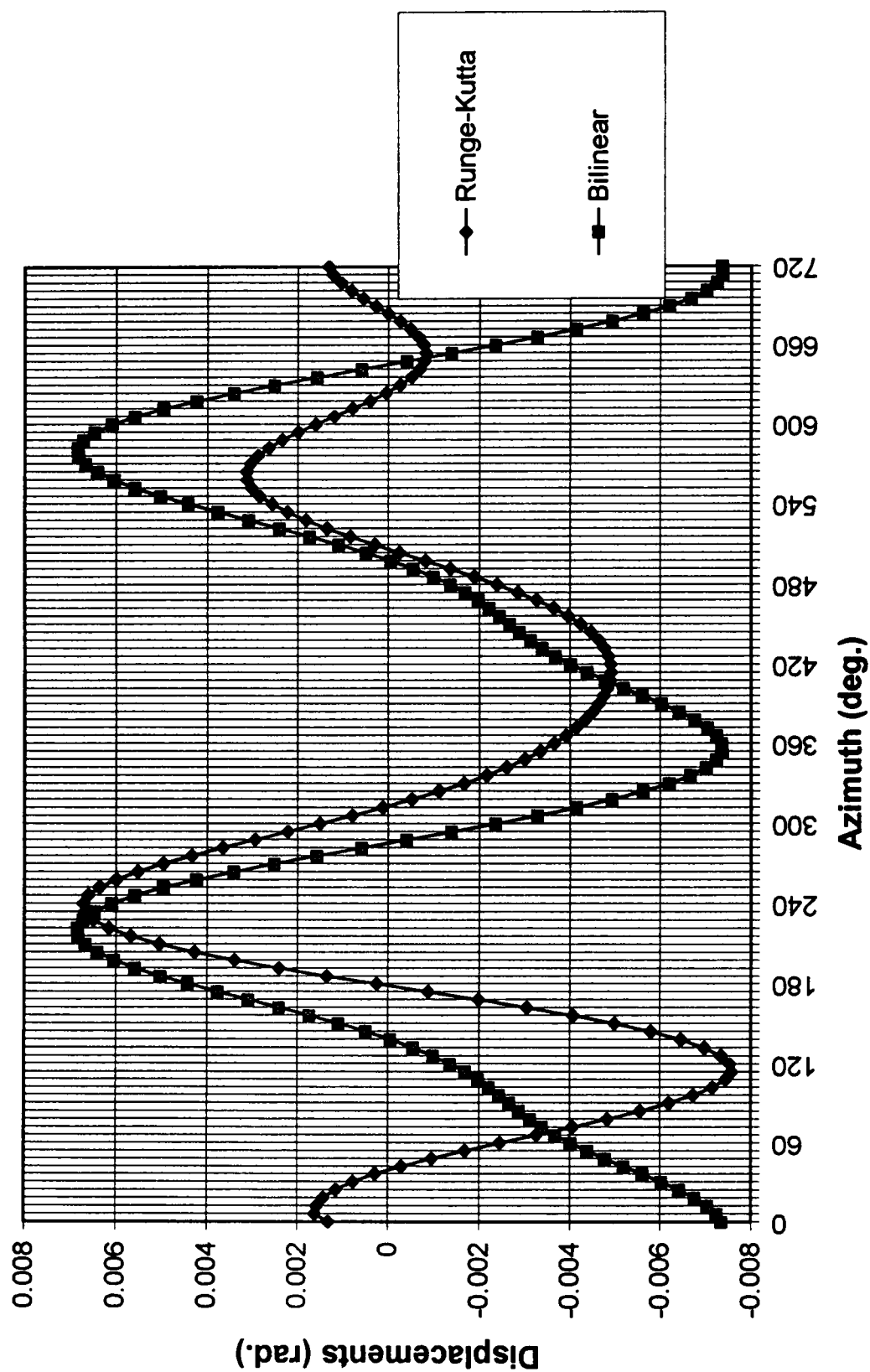


Figure 26. Runge-Kutta vs. Bilinear Solutions for Zeta (Lead-Lag) Displacements
(Advance Ratio 0.3)

CHAPTER 6

CONCLUSIONS

The conclusions of this thesis are as follows:

1. Both the time finite element method which is based on the variational approach and the time finite element method based on Hamilton's Law of varying action (bilinear approach) can be used to solve systems of ordinary differential equations of motion modelling nonlinear dynamical systems.
2. The bilinear algorithm is a convenient-to-use tool to analyze both linear and nonlinear systems because it does not require the generation of two separate subroutines for both nonlinear and linear terms as in the case for the variational algorithm.
3. Considering the results shown in Tables 1, 2, 3 and 4, the variational algorithm is faster than the bilinear algorithm but it fails to converge to a steady-state solution more often than the bilinear algorithm.
4. To summarize, the bilinear algorithm is slower than the variational algorithm but it is more convenient to use and it seems to be more reliable for problems with strong nonlinearities.

BIBLIOGRAPHY

BIBLIOGRAPHY

Borri, M., Ghirringhelli, G., L., Lauz, M., Mantegazza, P., and Merlini, T., 1985, *Dynamic Response of Mechanical Systems by a Weak Hamiltonian Formulation*, Computers and Structures, Vol. 20, No. 1-3, pp. 495 - 508.

Chopra, I and Dugundji, J., 1979, *Non-linear Dynamics Response of a Wind Turbine Blade*, Journal of Sound and Vibration, Vol. 63, 2, pp. 265-286.

Friedmann, P.P., 1976, *Aeroelastic Modelling of Large Wind Turbines*, Journal of the American Helicopter Society, pp. 17-27.

Friedmann, P., P. and Kottapalli, S., B., R., 1982, *Coupled Flag-Lag-Torsional Dynamics of Hingeless Rotor Blades in Forward Flight*, Journal of the American Helicopter Society, Vol. 27, pp. 28 - 36.

Houbolt, J. C., and Brooks, G. W., 1958, *Differential Equation of Motion for Combined Flapwise Bending Chordwise Bending, and Torsion of Twist Nonuniform Rotor Blades*, NACA Report 1346.

Izadpanah, A. 1986, *p-Version Finite Elements for the Space-Time Domain with Application to Floquet Theory*, Ph. D. Thesis, Aerospace Engineering Dept., Georgia Institute of Technology, Atlanta, GA.

Kawiecki, G., and Sivaneri, T., N., 1994, *Bilinear Formulation Applied to the Response and Stability of Helicopter Rotor Blade*, AIAA Journal, Vol. 32, No. 10, pp. 2036 - 2043.

Kaza, K. R. V., and Kvaternik, R. G., 1979, *Examining of the Flap-Lag Stability of Rigid Articulated Blades*, Journal of Aircraft, Vol. 16, No. 12, pp. 876 - 884.

Kottapalli, S. B. R., Friedmann, P. P., and Rosen, A., 1979, *Aeroelastic Stability and Response of Horizontal Axis Wind Turbine Blades*, AIAA Journal, Vol. 17, No. 12, pp. 1381-1389.

Panda, B., and Chopra, I., 1986, *Dynamics Stability of Hingeless and Bearingless Rotors in Forward Flight*, Comp. and Math. with Applications, Vol. 12A, No. 01, pp. 111 - 130.

Panda, B., and Chopra, I., 1987, *Dynamics of Composite Rotor Blades in Forward Flight*, Vertica, Vol. 11, No. 1/2, pp. 187 - 209.

Peters, D., A., and Izadpanah, A., P., 1988, *hp-Version Finite Element for the Space-Time*

Domain, Computational Mechanics, Vol. 3, pp. 73 - 88.

Straub, F., K., and Friedmann, P., P., 1981, *A Galerkin-type Finite Element Method for Rotary Wing Aeroelasticity*, Vertica, Vol. 5, pp. 75 - 98.

Sivaneri, N., T., and Chopra, I., 1982, *Dynamic Stability of a Rotor Blade Using Finite Element Analysis*, AIAA Journal, Vol. 20, No. 5, pp. 716 - 723.

Sivaneri, N. T., and Chopra, I., 1984, *Finite Element Analysis for Bearingless Rotor Blade Aeroelasticity*, Journal of the American Helicopter Society, Vol. 29, pp. 42 - 51.

Sivaneri, T., N., and Kawiecki, G., 1989, *Forward Flight Aeroelasticity of a Hingeless Rotor Blade by Bilinear Formulation*, AIAA/ASME/ASCE/AHS/ASc 30th Structures, Structural Dynamics, and Material Conference, AIAA Paper 89-1233, Mobile, AL.

Thompson, J.M.T., and Stewart H.B., 1986, *Nonlinear Dynamics and Chaos*, John Wiley and Sons, New York.

APPENDIX

PROGRAM VARIATIONAL

```

C*****
C  THIS PROGRAM IS SOLVING A SYSTEM OF ORDINARY, NONLINEAR DIFF.
C  EQUATIONS MODELLING A DYNAMICAL SYSTEM REPRESENTED BY INERTIA,
C  DAMPING, STIFFNESS AND NONLINEAR FORCING FUNCTION, VARIATIONAL
C  TIME FINITE ELEMENT IS USED.
C  IT CAN READ THE SOLUTION VECTOR FROM
C  OTHER PROGRAMS AND USE IT AS THE INITIAL GUESS.
C*****
C      DECLARE VARIABLES
C*****
      INTEGER IRCODE_B
      INTEGER IPATH ! PATH INDICATOR IN DLSARG, 1=>AX=B SOLVED
      INTEGER ITERNU ! ITERATION NUMBER
      INTEGER LA1 ! DO LOOP 105 COUNTER
      INTEGER LA1VEL ! CURVEL VECTOR ELEMENT PROPER NUMBER (LOOP 105)
      INTEGER LIMLOW ! MULTIPLIERS TO GET THE LENGTH OF ONE PERIOD
      INTEGER LIMUPP ! -//-
      INTEGER MAXITR ! MAXIMUM NUMBER OF ITERATIONS
      INTEGER NEDSIZ !=NUSHFU*NUMELE: NUM. OF ROWS IN ELEMENT DISPL. MX
      INTEGER NELCSF ! NUMBER OF TIME ELEMENTS CONSIDERED SO FAR
      INTEGER NELROW !=NLCSIZ*NUMELE : NUMBER OF ROWS IN STACKED ARRAYS
      INTEGER NFINSL !=(NUSHFU-1)*NUMELE:NUM. OF ROWS IN FIN. SOL. MX.
      INTEGER NGLSIZ !=NUMDOF*NUMELE*(NUSHFU-1): SIZE OF THE GLOBAL MX.
      INTEGER NGLSPO !=NGLSIZ+1 : NUMBER OF COLUMNS IN THE AUGMENT. MX.
      INTEGER NLCSIZ !=NUMDOF*NUSHFU: SIZE OF ELEMENT COEFF. MATRIX
      INTEGER NRELAS ! LAST ROW OF ELEM. COEF. MX. FOR GIVEN TIME EL.
      INTEGER NRELFI ! FIRST ROW OF ELEM. COEF. MX. FOR GIVEN TIME EL.
      INTEGER NRELMC ! NUMBER OF ROWS OF ELEM. COEF. MX. CONSID. SO FAR
      INTEGER NRFSMX ! ROW NUMBER IN THE FINAL SOLUTION MATRIX
      INTEGER NRWCSF ! NUMBER OF STACKED MATRIX ROWS CONSIDERED SO FAR
      INTEGER NRWEFV ! ROW NUMBER OF THE EFFICIENT SOLUTION VECTOR
      INTEGER NRWSMX ! ROW NUMBER IN THE STACKED MATRIX
      INTEGER NUDOFD !=NUMDOF*2: DOUBLE NUMBER OF TIME ELEMENTS
      INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
      INTEGER NUMELE ! NUMBER OF TIME ELEMENTS
      INTEGER NURCSF ! NUMBER OF ELEMENT DISPL. MX. CONSIDERED SO FAR
      INTEGER NURODV ! ROW NUMBER IN THE DISPLACEMENT VECTOR
      INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
      INTEGER NUSHM1 !=NUSHFU-1 : NUMBER OF SHAPE FUNCTIONS MINUS ONE
      INTEGER I, J, K, L, M, N, IA, JA, KA, LA, MA, NA! DO LOOP COUNT.
      INTEGER IB, JB, KB, LB, MB, NB, IC, JC, KC, LC! DO LOOP COUNTERS
C
C*****
      PARAMETER ( NUMDOF = 1, NUMELE =120, NUSHFU = 2, NEDSIZ =240)
      PARAMETER ( NGLSIZ =120, NLCSIZ = 2, NUDOFD = 2, NELROW =240)
      PARAMETER ( NGLSPO =121, NFINSL =120)

```

C*****

C

```

REAL*8 ACCUM_A, ACCUM_B, USED_TIME
REAL*8 AMXCEM( NELROW, NLCSIZ) ! MX. CONTAINING STACKED ELEMENT MX
REAL*8 DACURM( NUMDOF, NUMDOF) ! CURRENT DAMP. MX. FOR GAUSS PT.
REAL*8 DAMPMX( NUMDOF, NUMDOF) ! DAMPING MATRIX
REAL*8 DANLMX( NUMDOF, NUMDOF) ! - FORCING FUN. DER. W.R.T. VEL.
REAL*8 DANLPT( NUMDOF, NUMDOF) ! NONLINEAR PART OF THE DAMPING MX.
REAL*8 DISPIN( NFINSL, NUMDOF) ! VECTOR OF INITIAL DISPLACEMENTS
REAL*8 ELCFMA( NLCSIZ, NLCSIZ) ! ELEMENT COEF. MX USED IN ASSEMBLY
REAL*8 ELCFMX( NLCSIZ, NLCSIZ) ! ELEMENT COEFFICIENT MATRIX
REAL*8 ELDIMX( NEDSIZ, NUMDOF) ! ELEMENT DISPLACEMENT MATRIX
REAL*8 EMCKFI( NLCSIZ, NLCSIZ) ! FINAL COEFFICIENT MATRIX
REAL*8 EMCKMX( NUDOFD, NUDOFD) ! SYSTEM COEFFICIENT MATRIX
REAL*8 EMCKTR( NLCSIZ, NUDOFD) !=SHDTRM*EMCKMX
REAL*8 ETANMX( NUDOFD, NUDOFD) ! TANGENTIAL MATRIX (K SUB T)
REAL*8 FINSLM( NFINSL, NUMDOF) ! FINAL SOLUTION MATRIX
REAL*8 GLCFAU( NGLSIZ, NGLSPO) ! AUGMENT. MX. USED IN GAUSS-JORDAN
REAL*8 GLCFMX( NGLSIZ, NGLSIZ) ! GLOBAL COEFFICIENT MATRIX
REAL*8 MASSMX( NUMDOF, NUMDOF) ! INERTIA MATRIX
REAL*8 SHDERM( NUDOFD, NLCSIZ) ! SHAPE FUNCT. AND DERIVAT. MATRIX
REAL*8 SHDTRM( NLCSIZ, NUDOFD) ! TRANSPOSED SHAPE FUNCT. DERIV. MX
REAL*8 STCURM( NUMDOF, NUMDOF) ! CURRENT STIF. MX. FOR GAUSS PT.
REAL*8 STIFMX( NUMDOF, NUMDOF) ! STIFFNESS MATRIX
REAL*8 STNLMX( NUMDOF, NUMDOF) ! - FORCING FUN. DER. W.R.T. DISPL.
REAL*8 STNLPT( NUMDOF, NUMDOF) ! NONLINEAR PART OF STIFFNESS MX.
REAL*8 TEMEDM( NEDSIZ, NUMDOF) ! TEMPORARY ELEMENT DISPLAC. MX.
REAL*8 TEMPOR( NGLSIZ, NGLSIZ) ! TEMPORARY MX. USED IN GAUSS-JOR.
REAL*8 VELOIN( NFINSL, NUMDOF) ! INITIAL VELOCITY VECTOR

```

C

```

REAL*8 _____( _____, _____) !
REAL*8 AVCCEV( NELROW) ! VECTOR CONTAINING STACKED RH SIDES
REAL*8 CURDIS( NUMDOF) ! DISPLACEMENT VECTOR FOR GIVEN GAUSS PT.
REAL*8 CURVEL( NUMDOF) ! VELOCITY VECTOR FOR GIVEN GAUSS PT.
REAL*8 CURRHS( NUDOFD) ! VECTOR OF FORCES IN THE SYSTEM
REAL*8 DANLVC( NUMDOF) ! NONLIN. CONTRIBUTION FROM DAMPING TO F(T)
REAL*8 DEGVAL( NUMELE) ! VECTOR OF TIME INTERVAL ENDS
REAL*8 DISLOC( NUDOFD) ! DISPLACEMENT AT LOCAL GAUSS POINT VECTOR
REAL*8 ELDIVE( NLCSIZ) ! ELEMENT DISPLACEMENT VECTOR
REAL*8 ELRHVA( NLCSIZ) ! ELEMENT DISPL. VEC. USED IN ASSEMBLY
REAL*8 ERHSVC( NLCSIZ) ! ELEMENT RIGHT HAND SIDE VECTOR
REAL*8 FIRHSV( NLCSIZ) ! FINAL RIGHT HAND SIDE VECTOR
REAL*8 FORCEV( NUMDOF) ! FORCE FUNCTION VECTOR
REAL*8 GAQPNT( 6) ! GAUSS QUADRATURE POINTS VECTOR
REAL*8 GAQWGT( 6) ! GAUSS QUADRATURE WEIGHTS VECTOR
REAL*8 GRHSVC( NGLSIZ) ! GLOBAL RIGHT HAND SIDE VECTOR
REAL*8 SHDERV( NUSHFU) ! SHAPE FUNCTION DERIVATIVE VECTOR
REAL*8 SHFUNV( NUSHFU) ! SHAPE FUNCTION VECTOR

```

```

REAL*8 SOLVEC( NGLSIZ) ! SOLUTION VECTOR
REAL*8 STNLVC( NUMDOF) ! NONLIN. CONTRIBUTION FROM STIFF. TO F(T)
REAL*8 UNDIVC( NGLSIZ) ! UNKNOWN DISPL. V. IN MORE EFFICIENT FORM
REAL*8 VECFOR( NUDOFD) ! RHS VECTOR CONTAINING FORCE VECTOR
REAL*8 VECMOD( NUDOFD) !=VECFOR + CURRHS: FORCES RESIDUA VECTOR
C REAL*8 _____( _____) !
REAL*8 AMPLIT ! FORCING FUNCTION AMPLITUDE
REAL*8 AZGLOB ! GLOBAL AZIMUTH COORDINATE IN TIME ELEMENT
REAL*8 AZLOCA ! LOCAL AZIMUTH COORDINATE IN TIME ELEMENT
REAL*8 CA1 ! NONLIN. SPRING #1 CHARACTERISTICS SLOPE
REAL*8 CB1 ! NONLIN. SPRING #1 CHARACTERISTICS INTERCEPT
REAL*8 CA2 ! NONLIN. SPRING #2 CHARACTERISTICS SLOPE
REAL*8 CB2 ! NONLIN. SPRING #2 CHARACTERISTICS INTERCEPT
REAL*8 COEFFI ! NOT USED
REAL*8 CONINI ! INITIAL DISPL. CONST. OVER AZIMUTH
REAL*8 CURAZI ! NOT USED
REAL*8 DUMMYV ! DUMMY VARIABLE
REAL*8 ENDNOD ! END NODE COORDINATE
REAL*8 EPSILN ! ACCEPTABLE DIFFERENCE BETWEEN TWO SOLUTIONS
REAL*8 FLNGLS ! FLOATING POINT NGLSIZ=NUMDOF*NUMELE*(NUSHFU-1)
REAL*8 FLNUEL ! FLOATING POINT NUMBER OF ELEMENTS
REAL*8 FOURTH ! ONE FOURTH OF PI RADIANS
REAL*8 FREQUE ! FORCING FUNCTION FREQUENCY
REAL*8 GAULOC ! GAUSSIAN POINT IN LOCAL COORDINATES
REAL*8 LINTLI ! LOWER ELEMENT INTEGRATION LIMIT
REAL*8 LOWLIM ! LOWER INTEGRATION LIMIT IN GLOBAL COORDINATES
REAL*8 PINUMB ! PI NUMBER
REAL*8 SONORM ! SOLUTION VECTOR NORM
REAL*8 TIELEN ! TIME ELEMENT LENGTH
REAL*8 UINTLI ! UPPER ELEMENT INTEGRATION LIMIT
REAL*8 UPLUSL ! SCALING FACTOR FOR TRANSFORMATION TO GLOB. COOR.
REAL*8 UPMINL ! SCALING FACTOR FOR TRANSFORMATION TO GLOB. COOR.
REAL*8 UPPLIM ! UPPER INTEGRATION LIMIT IN GLOBAL COORDINATES
C REAL*8 _____ !
C*****
COMMON /STIFCF/ CA1, CB1, CA2, CB2
C*****
COMMON /WORKSP/ RWKSP
REAL RWKSP(180930)
CALL IWKIN(180930)
C*****
C
CALL CPUTIME (ACCUM_A, IRCODE_A) ! Call "A"
C
READ ( 5, 1040) AMPLIT, FREQUE, EPSILN, CONINI, COEFFI
READ ( 5, 1010) LOWLIM, UPPLIM, MAXITR
READ ( 5, 1030) CA1, CB1, CA2, CB2

```

```

      READ ( 5, 1050) ( DEGVAL( I), DISPIN( I, 1),
&      VELOIN( I, 1),      I = 1, NFINSL)  VAR01440
      WRITE( 6, 2040) AMPLIT, FREQUE, EPSILN, CONINI, COEFFI
      WRITE( 6, 2010) LOWLIM, UPPLIM, MAXITR
      WRITE( 6, 2030) CA1, CB1, CA2, CB2
      WRITE( 6, 2050) ( DEGVAL( I), DISPIN( I, 1),
&      VELOIN( I, 1),      I = 1, NFINSL)
C
      AMPLIT = AMPLIT * 1.0D+00
      FREQUE = FREQUE * 1.0D+00
      EPSILN = EPSILN * 1.0D+00
      CONINI = CONINI * 1.0D+00
      LOWLIM = LOWLIM * 1.0D+00
      UPPLIM = UPPLIM * 1.0D+00
      CA1  = CA1  * 1.0D+00
      CB1  = CB1  * 1.0D+00
      CA2  = CA2  * 1.0D+00
      CB2  = CB2  * 1.0D+00
C
C*****
C      DEFINE THE VECTOR OF GAUSSIAN QUADRATURE POINTS
C*****
      CALL GAQPAW ( GAQPNT, GAQWGT)
C*****
C      DEFINE SYSTEM ARRAYS
C*****
      CALL INERTI ( NUMDOF, MASSMX)
      CALL DAMPIN ( NUMDOF, CURAZI, CURDIS, CURVEL, COEFFI, DAMPMX)
      CALL STIFFN ( NUMDOF, CURAZI, CURDIS, CURVEL, COEFFI, STIFMX)
C*****
C      DEFINE CONSTANTS
C*****
      ITERNU = 0
      NUSHM1 = NUSHFU - 1
      FLNGLS = DFLOAT( NGLSIZ)
      FLNUEL = DFLOAT( NUMELE)
      FOURTH = DATAN( 1.0D0)
      PINUMB = 4.0D0 * FOURTH
      SONORM = 10.0D0
      UINTLI = UPPLIM
      TIELEN = ( UINTLI - LINTLI) / DFLOAT( NUMELE)
C*****
C      TEST THE CONDITIONS
C*****
      IF ( NUSHFU .NE. 2) THEN
        PRINT *, 'CHECK NUSHFU. STOPPED AT THE BEGINNING'
        PRINT *, ' THIS VERSION IS DESIGNED FOR NUSHFU=2'

```

```

      STOP
    ENDIF
C
    IF ( SONORM .LE. EPSILN) THEN
      PRINT *, ' CHECK SONORM AND EPSILN. STOPPED AT THE BEGINNING'
      PRINT *, ' SONORM MUST BE .GT. EPSILN AT THE BEGINNING'
      STOP
    ENDIF
C*****
C      DEFINE THE VECTOR OF TIME ELEMENT END POSITIONS AR02000
C      IN DEGREES
C*****
      CALL DEGVEC ( NUMELE, TIELEN, PINUMB, DEGVAL)
C*****
C      GENERATE THE COEFFICIENT MATRIX
C*****
      CALL GENECDM ( NUDOFD, NUMDOF, MASSMX, DAMPMX, STIFMX, EMCKMX)
C*****
      CALL GICEDM ( NUMELE, NUSHFU, NUMDOF, NEDSIZ, NFINSL, DISPIN,
&      ELDIMX)
C*****
C      ENTER THE ITERATION LOOP
C*****
      DO 30 WHILE ( SONORM .GT. EPSILN)
        ITERNU = ITERNU + 1
        UPPLIM = LINTLI
C*****
C      INITIALIZE GLOBAL ARRAYS
C*****
      CALL INITMX ( NGLSIZ, NGLSIZ, GLCFMX)
      CALL INITVE ( NGLSIZ, GRHSVC)
C*****
C      ENTER THE LOOP OVER THE NUMBER OF TIME ELEMENTS
C*****
      DO 40 L = 1, NUMELE
        NELCSF = L - 1
        NRWCSF = NELCSF * NLCSIZ
        LOWLIM = UPPLIM
        UPPLIM = LOWLIM + TIELEN
C*****
C      INITIALIZE ELEMENT ARRAYS
C*****
      CALL INITMX ( NLCSIZ, NLCSIZ, ELCFMA)
      CALL INITMX ( NLCSIZ, NLCSIZ, ELCFMX)
      CALL INITVE ( NLCSIZ, ERHSVC)
      CALL INITVE ( NLCSIZ, ELRHVA)
C*****

```

```

C          DEFINE THE SCALING FACTORS FOR THE
C          TRANSFORMATION FROM GLOBAL TO LOCAL COORDINATES
C*****
      UPLUSL = ( UPPLIM + LOWLIM) * 0.5D0
      UPMINL = ( UPPLIM - LOWLIM) * 0.5D0
C*****
C          EXTRACT THE DISPLACEMENT VECTOR CORRESPONDING
C          TO GIVEN TIME ELEMENT
C*****
      CALL EXTEDV( NUSHFU, NUMDOF, NUDOFD, NELCSF, NLCSIZ, NEDSIZ,
&                ELDIMX,ELDIVE)
C*****
C          ENTER THE INTEGRATION LOOP
C*****
      DO 70 IA = 1, 6
        GAULOC = GAQPNT( IA)
        AZGLOB = UPLUSL + UPMINL * GAULOC
        AZLOCA = AZGLOB - LOWLIM
C*****
C          GENERATE THE FORCING VECTOR FOR THIS GAUSS PT.
C*****
      CALL FORCIN ( NUMDOF, AZGLOB, CURDIS, CURVEL, AMPLIT,
&                FREQUE, FORCEV)
C*****
C          GENERATE THE MATRIX CONTAINING THE SHAPE FUNCT.
C          AND SHAPE FUNCT. DERIVATIVES
C*****
      CALL GENSHD ( NUSHFU, AZLOCA, TIELEN, SHFUNV, SHDERV)
      CALL SHFDRM ( NLCSIZ, NUDOFD, NUMDOF, NUSHFU, SHFUNV,
&                SHDERV, SHDERM)
C*****
C          TRANSPOSE THE SHAPE FUNCTION AND DERIVATIVE MX.
C*****
      DO 80 JA = 1, NUDOFD
        DO 90 KA = 1, NLCSIZ
          SHDTRM( KA, JA) = SHDERM( JA, KA)
        90    CONTINUE
      80    CONTINUE
C*****
C          GENERATE THE INTEGRANDS OF ELEMENT COEFFICIENT
C          MATRIX AND RHS VECTOR
C*****
      CALL MATMUL ( SHDERM, ELDIVE, DISLOC, NUDOFD, NLCSIZ, 1)
C*****
C          EXTRACT CURRENT DISPLACEMENT VECTOR AND CURRENT
C          VELOCITY VECTOR FROM DISLOC VECTOR
C*****

```

```

DO 105 LA1 = 1, NUMDOF
  LA1VEL = LA1 + NUMDOF
  CURDIS( LA1) = DISLOC( LA1)
  CURVEL( LA1) = DISLOC( LA1VEL)
105  CONTINUE
C*****
C      GENERATE THE NONLINEAR CONTRIBUTION TO THE
C      FORCING VECTOR
C*****
      CALL STMXNL ( NUMDOF, AZGLOB, CURDIS, CURVEL, COEFFI,
&      AMPLIT, FREQUE, STNLPT )
      CALL DAMXNL ( NUMDOF, AZGLOB, CURDIS, CURVEL, COEFFI,
&      DANLPT )
      CALL MATMUL ( STNLPT, CURDIS, STNLVC, NUMDOF, NUMDOF, 1)
C*****
C      INITIALIZE RIGHT HAND SIDE VECTOR CONTAINING
C      FORCE VECTOR
C*****
      CALL INITVE ( NUDOFD, VECFOR)
C*****
C      FILL THE UPPER PART OF RHS VECTOR WITH FORCING
C      FUNCTION VECTOR
C*****
      DO 100 LA = 1, NUMDOF
        VECFOR( LA) = FORCECV( LA) - DANLVC( LA) - STNLVC( LA)
100    CONTINUE
C*****
      CALL MATMUL ( EMCKMX, DISLOC, CURRHS, NUDOFD, NUDOFD, 1)
      DO 110 MA = 1, NUDOFD
        VECMOD( MA) = -1.0D0*(VECFOR( MA) + CURRHS( MA))
110    CONTINUE
      CALL MATMUL ( SHDTRM, VECMOD, FIRHSV, NLCSIZ, NUDOFD, 1)
C*****
C      GENERATE THE TANGENTIAL MATRIX ETANMX FOR GIVEN
C      GAUSSIAN POINT
C*****
      CALL STIFNL ( NUMDOF, AZGLOB, CURDIS, CURVEL, COEFFI,
&      AMPLIT, FREQUE, STNLMX )
      CALL DAMPNL ( NUMDOF, AZGLOB, CURDIS, CURVEL, COEFFI,
&      DANLMX )
      CALL MATAD ( STIFMX, STNLMX, STCURM, NUMDOF, NUMDOF)
      CALL MATAD ( DAMPMX, DANLMX, DACURM, NUMDOF, NUMDOF)
      CALL GENECM ( NUDOFD, NUMDOF, MASSMX, DACURM, STCURM,
&      ETANMX)
      CALL MATMUL ( SHDTRM, ETANMX, EMCKTR, NLCSIZ, NUDOFD,
&      NUDOFD)
      CALL MATMUL ( EMCKTR, SHDERM, EMCKFI, NLCSIZ, NUDOFD,

```

```

&          NLCSIZ)
C*****
C          INTEGRATE
C*****
      DO 120 NA = 1, NLCSIZ
        ERHSVC( NA) = ERHSVC( NA) + FIRHSV( NA) * GAQWGT( IA)
&          * UPMINL
      DO 130 IB = 1, NLCSIZ
        ELCFMX( NA, IB) = ELCFMX( NA, IB) + EMCKFI( NA, IB)
&          * GAQWGT( IA) * UPMINL
130      CONTINUE
120      CONTINUE
70      CONTINUE
C*****
C          ADD THE INTEGRATED COEFFICIENTS MATRICES AND
C          RIGHT HAND SIDES TO THE STACKED ARRAYS
C*****
      DO 140 JB = 1, NLCSIZ
        NRWSMX = NRWCSF + JB
        AVCCEV( NRWSMX) = ERHSVC( JB)
      DO 150 KB = 1, NLCSIZ
        AMXCEM( NRWSMX, KB) = ELCFMX( JB, KB)
150      CONTINUE
140      CONTINUE
40      CONTINUE
C*****
C          ASSEMBLE THE GLOBAL SYSTEM AND SOLVE
C*****
      CALL ASSEMB ( NUMDOF, NUSHFU, NUMELE, NELROW, NGLSIZ, NLCSIZ,
&          AVCCEV, AMXCEM, ELRHVA, ELCFMA, GLCFMX, GRHSVC)
      IPATH = 1
      CALL DLSARG ( NGLSIZ, GLCFMX, NGLSIZ, GRHSVC, IPATH , SOLVEC)
C*****
C          EVALUATE THE NORM OF SOLUTION
C*****
      SONORM = 0.0D0
      DO 160 LB = 1, NGLSIZ
        SONORM = SONORM + DABS( SOLVEC( LB))
160      CONTINUE
      SONORM = SONORM/FLNGLS
C*****
C          CONVERT THE SOLUTION VECTOR TO THE MORE
C          EFFICIENT FORMAT
C*****
      CALL CUTFUT ( NGLSIZ, NUSHFU, NUMDOF, NUMELE, SOLVEC, UNDIV)
C*****
C          CONVERT THE EFFICIENT SOLUTION VECTOR TO THE

```

```

C          FORMAT OF DISPLACEMENT MATRIX
C*****
      CALL UDVEDM ( NUMELE, NUSHFU, NUMDOF, NEDSIZ, NGLSIZ, UNDIVI,
&          TEMEDM)
C*****
C          ADD THE CURRENT MODIFICATION TO PREVIOUS
C          APPROXIMATION
C*****
      DO 170 MB = 1, NUMDOF
        DO 180 NB = 1, NEDSIZ
          ELDIMX( NB, MB) = ELDIMX( NB, MB) + TEMEDM( NB, MB)
180      CONTINUE                                VAR03910
170     CONTINUE                                VAR03920
C*****
C          TEST THE NUMBER OF ITERATIONS
C*****
      PRINT *,      ITERNU,      SONORM
      IF ( ITERNU .GE. MAXITR) THEN
        PRINT *, ' TO MANY ITERATIONS, STOPPED IN LOOP 30'
        STOP
      ENDIF
30 END DO
C*****
C          GENERATE THE FINAL SOLUTION VECTOR
C*****
      CALL GEFSOV ( NUMELE, NUSHFU, NUMDOF, NEDSIZ, NFINSL, ELDIMX,
&          FINSLM )
C*****
C
      PRINT *, ' _____ '
      PRINT *, 'NUMBER OF TIME ELEMENTS =', NUMELE
      PRINT *, 'NUMBER OF ITERATIONS =', ITERNU
      PRINT *, 'FINAL NORM = ', SONORM
C
      DO 220 KC = 1, NUMELE
        WRITE (6, 1022) DEGVAL(KC), FINSLM(KC, 1)
220     CONTINUE
C*****
1000 FORMAT (3F10.0)
1010 FORMAT (2F10.0,I5)
1020 FORMAT (1X, I5, 1X, E10.3, 1X, E13.6)
1021 FORMAT (1X, E10.3, 1X, E13.6, 1X, E13.6)
1022 FORMAT (1X, E10.3, 1X, E13.6)
1030 FORMAT (4F10.0)
1040 FORMAT (2F10.0, 1F10.0, 2F10.0)
1050 FORMAT ( 3( 1X, E13.6))
2040 FORMAT (1X, ' AMPLITUDE =', E10.3, 1X, 'FREQUENCY =', E10.3/

```

```

& 1X, 'MAX. NORM =', E10.3, 1X, 'IN. DISPL. =', E10.3/
& 1X, 'DAMPING COEFFICIENT C =', E10.3)
2010 FORMAT (1X, 'LOWER LIMIT =', E13.6, 1X, 'UPPER LIMIT =', E13.6/
& 1X, 'MAX. NUMBER OF ITERATIONS =', I5)
2030 FORMAT (1X, 'A(1) =', E13.6, ' B(1) =', E13.6/
& 1X, 'A(2) =', E13.6, ' B(2) =', E13.6//)
2050 FORMAT ( 3( 1X, E13.6))
C
CALL CPUTIME(ACCUM_B, IRCODE_B) ! Call "A"
C*****
C CALCULATING THE CPU TIME USED
C*****
IF (IRCODE_A.NE.8.AND.IRCODE_B.EQ.0) THEN
    USED_TIME = ACCUM_B - ACCUM_A
    PRINT *
    PRINT *, USED_TIME, 'Microseconds of CPU time were used.'
END IF
STOP
END
C
C*****
SUBROUTINE ASLAST (NUMDOF, NUSHFU, NUMELE, NELROW, NGLSIZ, NLCSIZ,
&                 AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
C*****
C THIS SUBROUTINE INSERTS THE LAST ELEMENT MATRIX IN THE GLOBAL
C COEFFICIENT MATRIX
C*****
C DECLARE VARIABLES
C*****
INTEGER NLCSIZ ! =NUMDOF*NUSHFU, SIZE OF ELEMENT MATRIX
INTEGER NACTRW ! POSITION OF THE ROW OF THE GLOBAL COEFFICIENT
C & MATRIX TO WHICH AN ELEMENT OF THE ELEMENT
C & COEFFICIENT MATRIX IS ADDED
INTEGER NGLSIZ ! NUMDOF*(NUMELE-1)
INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
INTEGER NCOLM1 ! CONSIDERED DEGREE OF FREEDOM MINUS ONE (IN
C & INSERTION COLUMN CALCULATION)
INTEGER NROWM1 ! CONSIDERED DEGREE OF FREEDOM MINUS ONE (IN
C & INSERTION ROW CALCULATION)
INTEGER NGLCOL ! INSERTION COLUMN NUMBER FOR RIGHT SUBMATRIX
INTEGER NLCCOL ! INSERTION COLUMN NUMBER FOR LEFT SUBMATRIX AND
C & INSERTED COLUMN NUMBER FOR RIGHT ELEMENT
C & SUBMATRIX
INTEGER LELCOC ! INSERTED COLUMN NUMBER OF LEFT ELEMENT
C & SUBMATRIX
INTEGER NGLROW ! INSERTION ROW NUMBER FOR LOWER SUBMATRIX

```

```

      INTEGER NLCROW ! INSERTION ROW NUMBER FOR UPPER SUBMATRIX AND
C &      INSERTED ROW NUMBER FOR LOWER ELEMENT
C &      SUBMATRIX
      INTEGER LELROC ! INSERTED COLUMN NUMBER OF UPPER ELEMENT
C &      SUBMATRIX
      INTEGER NELROW !=NLCSIZ*NUMELE, NUMBER OF ROWS IN THE MATRIX
C &      CONTAINING STACKED ELEMENT COEFFICIENT MATRICES
      INTEGER I,J,K,L ! DO LOOP COUNTERS
C
      REAL*8 GRHSVC(NGLSIZ) ! GLOBAL RIGHT HAND SIDE VECTOR
      REAL*8 AVCCEV(NELROW) ! VECTOR CONTAINING STACKED RIGHT HAND
C &      SIDE VECTORS
      REAL*8 ELRHVC(NLCSIZ) ! ELEMENT RIGHT HAND SIDE VECTOR
      REAL*8 AMXCEM(NELROW, NLCSIZ) ! MATRIX CONTAINING STACKED
C &      ELEMENT COEFFICIENT MATRICES
      REAL*8 GLCFMX(NGLSIZ, NGLSIZ) ! GLOBAL COEFFICIENT MATRIX
      REAL*8 ELCFMX(NLCSIZ, NLCSIZ) ! ELEMENT COEFFICIENT MATRIX
C*****
C      EXTRACT THE LAST ELEMENT MATRIX FROM
C      THE ASSEMBLED MATRIX CONTAINING ALL ELEMENT MATRICES
C*****
      DO 10 I = 1, NLCSIZ
        NACTRW = NLCSIZ*(NUMELE-1) + I
        ELRHVC(I) = AVCCEV(NACTRW)
        DO 20 J = 1, NLCSIZ
          ELCFMX(I, J) = AMXCEM(NACTRW, J)
        20 CONTINUE
      10 CONTINUE
C*****
C      INSERT THE LAST ELEMENT MATRIX INTO
C      THE GLOBAL COEFFICIENT MATRIX
C*****
      DO 30 K = 1, NUMDOF
        NROWM1 = K - 1
        NGLROW = NGLSIZ - NUMDOF + K
        NLCROW = NROWM1 * NUSHFU + 1
        LELROC = NUSHFU * K
        GRHSVC(NLCROW) = ELRHVC(LELROC) + GRHSVC(NLCROW)
        GRHSVC(NGLROW) = ELRHVC(NLCROW) + GRHSVC(NGLROW)
        DO 40 L = 1, NUMDOF
          NCOLM1 = L - 1
          NGLCOL = NGLSIZ - NUMDOF + L
          NLCCOL = NCOLM1 * NUSHFU + 1
          LELCOC = NUSHFU * L
          GLCFMX(NLCROW, NLCCOL) = ELCFMX(LELROC, LELCOC) +
&          GLCFMX(NLCROW, NLCCOL)
          GLCFMX(NGLROW, NLCCOL) = ELCFMX(NLCROW, LELCOC) +

```

```

&          GLCFMX( NGLROW, NLCCOL)
GLCFMX( NLCROW, NGLCOL) = ELCFMX( LELROC, NLCCOL) +
&          GLCFMX( NLCROW, NGLCOL)
GLCFMX( NGLROW, NGLCOL) = ELCFMX( NLCROW, NLCCOL) +
&          GLCFMX( NGLROW, NGLCOL)
40  CONTINUE
30  CONTINUE
C
  RETURN
  END
C
C*****
  SUBROUTINE ASECON (NUMDOF, NUSHFU, NUMELE, NELROW, NGLSIZ, NLCSIZ,
&          AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
C*****
C  THIS SUBROUTINE INSERTS SECOND ELEMENT MATRIX IN THE GLOBAL
C  COEFFICIENT MATRIX
C*****
C          DECLARE VARIABLES
C*****
  INTEGER NLCSIZ ! =NUMDOF*NUSHFU, SIZE OF ELEMENT MATRIX
  INTEGER NACTRW ! POSITION OF THE ROW OF THE GLOBAL COEFFICIENT
C &          MATRIX TO WHICH AN ELEMENT OF THE ELEMENT
C &          COEFFICIENT MATRIX IS ADDED
  INTEGER NGLSIZ ! NUMDOF*(NUMELE-1)
  INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
  INTEGER NUMELE ! NUMBER OF ELEMENTS
  INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
  INTEGER NCOLM1 ! CONSIDERED DEGREE OF FREEDOM MINUS ONE (IN
C &          INSERTION COLUMN CALCULATION)
  INTEGER NROWM1 ! CONSIDERED DEGREE OF FREEDOM MINUS ONE (IN
C &          INSERTION ROW CALCULATION)
  INTEGER NGCCNT ! INSERTION COLUMN NUMBER FOR RIGHT SUBMATRIX
  INTEGER NGLCCN ! INSERTION COLUMN NUMBER FOR LEFT SUBMATRIX
  INTEGER NLCCOL ! INSERTED COLUMN NUMBER OF LEFT ELEMENT
C &          SUBMATRIX
  INTEGER LELCOC ! INSERTED COLUMN NUMBER OF RIGHT ELEMENT
C &          SUBMATRIX
  INTEGER NGRCNT ! INSERTION ROW NUMBER FOR LOWER SUBMATRIX
  INTEGER NGLRCN ! INSERTION ROW NUMBER FOR UPPER SUBMATRIX
  INTEGER NLCROW ! INSERTED ROW NUMBER OF UPPER ELEMENT
C &          SUBMATRIX
  INTEGER LELROC ! INSERTED ROW NUMBER OF LOWER ELEMENT
C &          SUBMATRIX
  INTEGER NELROW ! =NLCSIZ*NUMELE, NUMBER OF ROWS IN THE MATRIX
C &          CONTAINING STACKED ELEMENT COEFFICIENT MATRICES
  INTEGER I,J,K,L ! DO LOOP COUNTERS

```

```

C
  REAL*8 GRHSVC( NGLSIZ) ! GLOBAL RIGHT HAND SIDE VECTOR
  REAL*8 AVCCEV( NELROW) ! VECTOR CONTAINING STACKED RIGHT HAND
C &          SIDE VECTORS
  REAL*8 ELRHVC( NLCSIZ) ! ELEMENT RIGHT HAND SIDE VECTOR
  REAL*8 AMXCEM( NELROW, NLCSIZ) ! MATRIX CONTAINING STACKED
C &          ELEMENT COEFFICIENT MATRICES
  REAL*8 GLCFMX( NGLSIZ, NGLSIZ) ! GLOBAL COEFFICIENT MATRIX
  REAL*8 ELCFMX( NLCSIZ, NLCSIZ) ! ELEMENT COEFFICIENT MATRIX
C*****
C          EXTRACT THE LAST ELEMENT MATRIX FROM
C          THE ASSEMBLED MATRIX CONTAINING
C          ALL ELEMENT MATRICES
C*****
  DO 10 I = 1, NLCSIZ
    NACTRW = NLCSIZ + I
    ELRHVC( I) = AVCCEV( NACTRW)
    DO 20 J = 1, NLCSIZ
      ELCFMX( I, J) = AMXCEM( NACTRW, J)
20  CONTINUE
10  CONTINUE
C*****
C          INSERT SECOND ELEMENT MATRIX INTO
C          THE GLOBAL COEFFICIENT MATRIX
C*****
  DO 30 K = 1, NUMDOF
    NROWM1 = K - 1
    NGLRCN = K * NUSHFU
    NLCROW = NROWM1 * NUSHFU + 1
    NGRCNT = NLCSIZ + K
    LELROC = NUSHFU * K
    GRHSVC( NGLRCN) = ELRHVC( NLCROW) + GRHSVC( NGLRCN)
    GRHSVC( NGRCNT) = ELRHVC( LELROC) + GRHSVC( NGRCNT)
    DO 40 L = 1, NUMDOF
      NCOLM1 = L - 1
      NGLCCN = L * NUSHFU
      NLCCOL = NCOLM1 * NUSHFU + 1
      NGCCNT = NLCSIZ + L
      LELCOC = NUSHFU * L
      GLCFMX( NGLRCN, NGLCCN) = ELCFMX( NLCROW, NLCCOL) +
&          GLCFMX( NGLRCN, NGLCCN)
      GLCFMX( NGRCNT, NGLCCN) = ELCFMX( LELROC, NLCCOL) +
&          GLCFMX( NGRCNT, NGLCCN)
      GLCFMX( NGLRCN, NGCCNT) = ELCFMX( NLCROW, LELCOC) +
&          GLCFMX( NGLRCN, NGCCNT)
      GLCFMX( NGRCNT, NGCCNT) = ELCFMX( LELROC, LELCOC) +
&          GLCFMX( NGRCNT, NGCCNT)

```

```

40  CONTINUE
30  CONTINUE
C
  RETURN
  END
C
C*****
  SUBROUTINE ASTHIR (NUMDOF, NUSHFU, NUMELE, NELROW, NGLSIZ, NLCSIZ,
    &                AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
C*****
C  THIS SUBROUTINE INSERTS SECOND ELEMENT MATRIX IN THE GLOBAL
C  COEFFICIENT MATRIX
C*****
C  DECLARE VARIABLES
C*****
  INTEGER NLCSIZ ! =NUMDOF*NUSHFU, SIZE OF ELEMENT MATRIX
  INTEGER NACTRW ! POSITION OF THE ROW OF THE GLOBAL COEFFICIENT
C &    MATRIX TO WHICH AN ELEMENT OF THE ELEMENT
C &    COEFFICIENT MATRIX IS ADDED
  INTEGER NGLSIZ ! NUMDOF*(NUMELE-1)
  INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
  INTEGER NUMELE ! NUMBER OF ELEMENTS
  INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
  INTEGER NCOLM1 ! CONSIDERED DEGREE OF FREEDOM MINUS ONE (IN
C &    INSERTION COLUMN CALCULATION)
  INTEGER NROWM1 ! CONSIDERED DEGREE OF FREEDOM MINUS ONE (IN
C &    INSERTION ROW CALCULATION)
  INTEGER NGLRWC ! INSERTION COLUMN NUMBER FOR RIGHT SUBMATRIX
  INTEGER NGLCLC ! INSERTION COLUMN NUMBER FOR LEFT SUBMATRIX
  INTEGER NLCCOL ! INSERTED COLUMN NUMBER OF LEFT ELEMENT
C &    SUBMATRIX
  INTEGER LELCOC ! INSERTED COLUMN NUMBER OF RIGHT ELEMENT
C &    SUBMATRIX
  INTEGER NGL21C ! INSERTION ROW NUMBER FOR LOWER SUBMATRIX
  INTEGER NGL11C ! INSERTION ROW NUMBER FOR UPPER SUBMATRIX
  INTEGER NLCROW ! INSERTED ROW NUMBER OF UPPER ELEMENT
C &    SUBMATRIX
  INTEGER NELROW ! =NLCSIZ*NUMELE, NUMBER OF ROWS IN THE MATRIX
C &    CONTAINING STACKED ELEMENT COEFFICIENT MATRICES
  INTEGER LELROC ! INSERTED ROW NUMBER OF LOWER ELEMENT
C &    SUBMATRIX
  INTEGER NUMEM1 ! NUMBER OF ELEMENTS MINUS ONE
  INTEGER I,J,K,L,M ! DO LOOP COUNTERS
C
  REAL*8 GRHSVC(NGLSIZ) ! GLOBAL RIGHT HAND SIDE VECTOR
  REAL*8 AVCCEV(NELROW) ! VECTOR CONTAINING STACKED RIGHT HAND
C &    SIDE VECTORS

```

```

REAL*8 ELRHVC( NLCSIZ) ! ELEMENT RIGHT HAND SIDE VECTOR
REAL*8 AMXCEM( NELROW, NLCSIZ) ! MATRIX CONTAINING STACKED
C & ELEMENT COEFFICIENT MATRICES
REAL*8 GLCFMX( NGLSIZ, NGLSIZ) ! GLOBAL COEFFICIENT MATRIX
REAL*8 ELCFMX( NLCSIZ, NLCSIZ) ! ELEMENT COEFFICIENT MATRIX
C
NUMEM1 = NUMELE - 1
C*****
C LOOP OVER THE TIME ELEMENTS
C BETWEEN THE SECOND AND THE LAST
C*****
DO 10 I = 3, NUMEM1
C*****
C EXTRACT THE ELEMENT MATRIX FROM
C THE ASSEMBLED MATRIX CONTAINING
C ALL ELEMENT MATRICES
C*****
DO 20 J = 1, NLCSIZ
NACTRW = NLCSIZ * ( I - 1) + J
ELRHVC( J) = AVCCEV( NACTRW)
DO 30 K = 1, NLCSIZ
ELCFMX( J, K) = AMXCEM( NACTRW, K)
30 CONTINUE
20 CONTINUE
C*****
C INSERT THE ELEMENT MATRIX INTO
C THE GLOBAL COEFFICIENT MATRIX
C*****
DO 40 L = 1, NUMDOF
NROWM1 = L - 1
NGL11C = NLCSIZ + (I - 3) * NUMDOF + L
NLCROW = NROWM1 * NUSHFU + 1
NGL21C = NGL11C + NUMDOF
LELROC = NUSHFU * L
GRHSVC( NGL11C) = ELRHVC( NLCROW) + GRHSVC( NGL11C)
GRHSVC( NGL21C) = ELRHVC( LELROC) + GRHSVC( NGL21C)
DO 50 M = 1, NUMDOF
NCOLM1 = M - 1
NGLCLC = NLCSIZ + (I - 3) * NUMDOF + M
NLCCOL = NCOLM1 * NUSHFU + 1
NGLRWC = NGLCLC + NUMDOF
LELCOC = NUSHFU * M
GLCFMX( NGL11C, NGLCLC) = ELCFMX( NLCROW, NLCCOL) +
& GLCFMX( NGL11C, NGLCLC)
GLCFMX( NGL21C, NGLCLC) = ELCFMX( LELROC, NLCCOL) +
& GLCFMX( NGL21C, NGLCLC) VAR02640
GLCFMX( NGL11C, NGLRWC) = ELCFMX( NLCROW, LELCOC) +

```

```

&          GLCFMX( NGL11C, NGLRWC)    VAR02660
      GLCFMX( NGL21C, NGLRWC) = ELCFMX( LELROC, LELCOC) +
&          GLCFMX( NGL21C, NGLRWC)
50    CONTINUE
40    CONTINUE
10    CONTINUE
C
      RETURN
      END
C
C*****
      SUBROUTINE ASSEMB( NUMDOF, NUSHFU, NUMELE, NELROW, NGLSIZ, NLCSIZ,
&          AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
C*****
C      THIS SUBROUTINE ASSEMBLES THE GLOBAL COEFFICIENT MATRIX AND
C      GLOBAL RIGHT HAND SIDE VECTOR
C*****
C          DECLARE VARIABLES
C*****
      INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
      INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
      INTEGER NUMELE ! NUMBER OF TIME ELEMENTS
      INTEGER NGLSIZ ! =NUMDOF*(NUMELE-1) , SIZE OF GLOBAL ARRAYS
      INTEGER NLCSIZ ! =NUMDOF*NUSHFU , ELEMENT MATRIX SIZE
      INTEGER NELROW ! =NLCSIZ*NUMELE , NUMBER OF ROWS IN THE MATRIX
C &          CONTAINING ELEMENT MATRICES
      INTEGER NACTRW ! ACTUAL ROW NUMBER IN ASSEMBLED MATRIX
C &          CONTAINING ELEMENT MATRICES
      INTEGER NUMEM1 ! NUMBER OF TIME ELEMENTS MINUS ONE
      INTEGER I,J,K,L ! DO LOOP COUNTERS
C
      REAL*8 GRHSVC( NGLSIZ) ! GLOBAL RIGHT HAND SIDE VECTOR
      REAL*8 AVCCEV( NELROW) ! VECTOR CONTAINING STACKED RIGHT HAND
C &          SIDE VECTORS
      REAL*8 ELRHVC( NLCSIZ) ! ELEMENT RIGHT HAND SIDE VECTOR
      REAL*8 AMXCEM( NELROW, NLCSIZ) ! MATRIX CONTAINING STACKED
C &          ELEMENT COEFFICIENT MATRICES
      REAL*8 GLCFMX( NGLSIZ, NGLSIZ) ! GLOBAL COEFFICIENT MATRIX
      REAL*8 ELCFMX( NLCSIZ, NLCSIZ) ! ELEMENT COEFFICIENT MATRIX
C*****
      EXTERNAL ASLAST, ASECON, ASTHIR
C*****
C          ZERO THE GLOBAL COEFFICIENT MATRIX AND
C          THE GLOBAL RIGHT HAND SIDE VECTOR
C*****
      DO 10 I = 1, NGLSIZ
          GRHSVC( I) = 0.0D+00

```

```

        DO 20 J = 1, NGLSIZ
          GLCFMX( I, J) = 0.0D+00
20    CONTINUE
10    CONTINUE
C*****
C          INSERT THE FIRST ELEMENT MATRIX INTO
C          THE GLOBAL MATRIX
C*****
        DO 30 K = 1, NLCSIZ
          GRHSVC( K) = AVCCEV( K)
          DO 40 L = 1, NLCSIZ
            GLCFMX( K, L) = AMXCEM( K, L)
40    CONTINUE
30    CONTINUE
C*****
C          INSERT THE REMAINING ELEMENT MATRICES
C          INTO THE GLOBAL MATRIX
C*****
        IF (NUMELE .EQ. 2) THEN
          CALL ASLAST (NUMDOF, NUSHFU, NUMELE,NELROW, NGLSIZ, NLCSIZ,
&          AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
        ELSE
          IF (NUMELE .EQ. 3) THEN
            CALL ASECON (NUMDOF, NUSHFU, NUMELE,NELROW, NGLSIZ, NLCSIZ,
&            AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
            CALL ASLAST (NUMDOF, NUSHFU, NUMELE,NELROW, NGLSIZ, NLCSIZ,
&            AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
          ELSE
            CALL ASECON (NUMDOF, NUSHFU, NUMELE,NELROW, NGLSIZ, NLCSIZ,
&            AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
            CALL ASLAST (NUMDOF, NUSHFU, NUMELE,NELROW, NGLSIZ, NLCSIZ,
&            AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
            CALL ASTHIR (NUMDOF, NUSHFU, NUMELE,NELROW, NGLSIZ, NLCSIZ,
&            AVCCEV, AMXCEM, ELRHVC, ELCFMX, GLCFMX, GRHSVC)
          ENDIF
        ENDIF
C
        RETURN
        END
C
C*****
C          SUBROUTINE CUTFUT ( NUNDIS, NUSHFU, NUMDOF, NUMELE, SOLVEC,
&          UNDIVE)
C*****
C
C          THIS SUBROUTINE CONVERTS THE SOLUTION VECTOR (SOLVEC) FROM
C          CURRENT FORMAT TO MORE EFFICIENT FORMAT

```

```

C*****
C      DECLARE VARIABLES
C*****
C      INTEGER NUNDIS !=(NUSHFU-1)*NUMDOF*NUMELE: NUMBER OF UNKNOWN
      DISPLACEMENTS (INPUT)
C      INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
      INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
      INTEGER NUMELE ! NUMBER OF TIME ELEMENTS
      INTEGER NRFESV ! ROW NUMBER FOR FIRST ELEMENT IN SOLUTION VECTOR
      INTEGER NRUDIV ! ROW NUMBER IN UNKNOWN DISPLACEMENTS VECTOR
      INTEGER NRUDCT ! UNKNOWN DISPLACEMENTS VECTOR ROW COUNTER
      INTEGER NRSLVC ! ROW NUMBER IN SOLUTION VECTOR
      INTEGER NUSFM1 ! NUMBER OF SHAPE FUNCTIONS MINUS ONE
      INTEGER NUSFM2 ! NUMBER OF SHAPE FUNCTIONS MINUS TWO
      INTEGER NSTRSV ! NUMBER OF STARTING ROW IN SOLUTION VECTOR
      INTEGER NSTRUD ! NUMBER OF STARTING ROW IN UNKNOWN DISPL. VECTOR
      INTEGER NUDCSF ! NUMBER OF UNKNOWN DISP. VEC. ROWS CONSID. SO FAR
      INTEGER NSVCSF ! NUMBER OF SOLUTION. VEC. ROWS CONSIDERED SO FAR
      INTEGER NRUDVC ! ROW NUMBER IN THE UNKNOWN DISPLACEMENT VECTOR
      INTEGER NRSVEC ! ROW NUMBER IN THE SOLUTION VECTOR
      INTEGER NUSTSV ! NUMBER OF STARTING ROW IN SOLUTION VECTOR
      INTEGER NUSTUD ! NUMBER OF STARTING ROW IN UNKNOWN DISPL. VECTOR
      INTEGER NUNTE ! NUMBER OF INNER TIME ELEMENTS
      INTEGER NURWTC ! NUMBER OF ROWS TO CONSIDER
      INTEGER NSVERW ! ROW NUMBER IN SOLUTION VECTOR
      INTEGER NUDVRW ! ROW NUMBER IN UNKNOWN DISPLACEMENTS VECTOR
      INTEGER I, J, K ! DO LOOP COUNTERS
      INTEGER L, M, N ! DO LOOP COUNTERS
C
C      REAL*8 SOLVEC ( NUNDIS) ! SOLUTION VECTOR IN CURRENT FORMAT
      REAL*8 UNDIVC ( NUNDIS) ! SOLUTION VECTOR IN MORE EFFICIENT FORM.
C
C      NUSFM1 = NUSHFU - 1
      NUSFM2 = NUSHFU - 2
      NUNTE = NUMELE - 2
C*****
C      CONVERT VECTOR ELEMENTS CORRESPONDING TO FIRST
C      TIME ELEMENT
C      CONVERT FIRST NODES FOR EACH TIME ELEMENT
C*****
      DO 10 I = 1, NUMDOF
        NRFESV = ( I - 1 ) * NUSHFU + 1
        NRUDIV = NUNDIS - ( NUMDOF - I ) * NUSFM1
        UNDIVC( NRUDIV ) = SOLVEC( NRFESV )
      10 CONTINUE
C*****
C      CONVERT REMAINING NODES FOR EACH TIME ELEMENT

```

```

C*****
NRUDCT = 0
C
DO 20 J = 1, NUMDOF
  NUDCSF = ( J - 1 ) * NUSHFU
  DO 30 K = 2, NUSHFU
    NRSLVC = NUDCSF + K
    NRUDCT = NRUDCT + 1
    UNDIV( NRUDCT ) = SOLVEC( NRSLVC )
  30 CONTINUE
20 CONTINUE
C*****
C          CONVERT VECTOR ELEMENTS CORRESPONDING TO INNER
C          TIME ELEMENTS
C*****
IF ( NUNTE .GT. 0 ) THEN
  NUSTSV = NUSHFU * NUMDOF
  NUSTUD = NUSFM1 * NUMDOF
  NURWTC = NUNTE * NUSFM1 * NUMDOF
C
DO 60 N = 1, NURWTC
  NSVERW = NUSTSV + N
  NUDVRW = NUSTUD + N
  UNDIV( NUDVRW ) = SOLVEC( NSVERW )
60 CONTINUE
ENDIF
C
RETURN
END
C
C*****
SUBROUTINE DAMPIN ( NUMDOF, CURAZI, CURDIS, CURVEL,
&                 DZCOEF, DAMPMX )
C*****
C  THIS SUBROUTINE CALCULATES THE DAMPING MATRIX.
C*****
C  VARIABLES:
C  NUMDOF : NUMBER OF DEGREES OF FREEDOM (INPUT).
C  CURAZI : CURRENT AZIMUTH - TIME (INPUT).
C  CURDIS : CURRENT DISPLACEMENT VECTOR (INPUT).
C  CURVEL : CURRENT VELOCITY VECTOR (INPUT).
C  PCOEFF : DIMENSIONLESS ROTATING FLAP FREQUENCY. (INPUT).
C  DAMPMX : DAMPING MATRIX (OUTPUT).
C  DBCOEF : VISCOUS DAMPING RATIO IN FLAP.
C  DZCOEF : VISCOUS DAMPING RATIO IN LEAD-LAG.
C  FBNROT : NONDIMENSIONAL, NONROTATING FLAP NATURAL FREQUENCY.
C  FZNROT : NONDIMENSIONAL, NONROTATING LEAD-LAG NATURAL FREQUENCY.

```

```

C*****
C      DECLARE VARIABLES.
C*****
      INTEGER NUMDOF
      REAL*8 CURAZI, CURDIS(NUMDOF), CURVEL(NUMDOF), FBNROT, FZNROT
      REAL*8 DBCOEF, DZCOEF, PCOEF
      REAL*8 DAMPMX( NUMDOF, NUMDOF)
C*****
C      DUFFING'S EQN., J.M.T. THOMPSON,
C      "NONLINEAR DYNAMICS AND CHAOS",P.8,10.
C*****
      DAMPMX( 1, 1) = 0.08D+00
C
      RETURN
      END
C
C*****
      SUBROUTINE DAMPNL ( NUMDOF, CURAZI, CURDIS, CURVEL,
&      DZCOEF, DAMPMX )
C*****
C
C      THIS SUBROUTINE CALCULATES THE DERIVATIVE W.R.T. VELOCITY
C      VECTOR OF THE NONLINEAR FORCING VECTOR.
C*****
C      REMEMBER THAT A NEGATIVE VALUE OF THE DERIVATIVE MUST BE
C      GENERATED. THE REASON IS THAT THE DERIVATIVE IS ADDED TO
C      THE DAMPING MATRIX, NOT SUBTRACTED AS IT SHOULD !!!
C      CURAZI : CURRENT AZIMUTH - TIME (INPUT).
C      CURDIS : CURRENT DISPLACEMENT VECTOR (INPUT).
C      CURVEL : CURRENT VELOCITY VECTOR (INPUT).
C      PCOEF : DIMENSIONLESS ROTATING FLAP FREQUENCY. (INPUT).
C      DAMPMX : DAMPING MATRIX (OUTPUT).
C      DBCOEF : VISCOUS DAMPING RATIO IN FLAP.
C      DZCOEF : VISCOUS DAMPING RATIO IN LEAD-LAG.
C      FBNROT : NONDIMENSIONAL, NONROTATING FLAP NATURAL FREQUENCY.
C      FZNROT : NONDIMENSIONAL, NONROTATING LEAD-LAG NATURAL FREQUENCY.
C*****
C      DECLARE VARIABLES.
C*****
      INTEGER NUMDOF
      REAL*8 CURAZI, CURDIS(NUMDOF), CURVEL(NUMDOF), FBNROT, FZNROT
      REAL*8 DBCOEF, DZCOEF, PCOEF
      REAL*8 DAMPMX( NUMDOF, NUMDOF)
C*****
C      DUFFING'S EQN., J.M.T. THOMPSON,

```

```

C          "NONLINEAR DYNAMICS AND CHAOS",P.8,10.
C*****
C      DAMPMX( 1, 1) = 0.00D+00
C
C      END
C
C*****
C      SUBROUTINE DAMXNL ( NUMDOF, CURAZI, CURDIS, CURVEL,
C      &          DZCOEF, DAMPMX )
C*****
C
C      THIS SUBROUTINE CALCULATES THE NONLINEAR PART OF THE DAMPING
C      MATRIX
C*****
C      VARIABLE
C      NUMDOF : NUMBER OF DEGREES OF FREEDOM (INPUT).
C      CURAZI : CURRENT AZIMUTH - TIME (INPUT).
C      CURDIS : CURRENT DISPLACEMENT VECTOR (INPUT).
C      CURVEL : CURRENT VELOCITY VECTOR (INPUT).
C      PCOEFF : DIMENSIONLESS ROTATING FLAP FREQUENCY. (INPUT).
C      DAMPMX : DAMPING MATRIX (OUTPUT).
C      DBCOEF : VISCOUS DAMPING RATIO IN FLAP.
C      DZCOEF : VISCOUS DAMPING RATIO IN LEAD-LAG.
C      FBNROT : NONDIMENSIONAL, NONROTATING FLAP NATURAL FREQUENCY.
C      FZNROT : NONDIMENSIONAL, NONROTATING LEAD-LAG NATURAL FREQUENCY.
C*****
C      DECLARE VARIABLES.
C*****
C      INTEGER NUMDOF
C      REAL*8 CURAZI, CURDIS(NUMDOF), CURVEL(NUMDOF), FBNROT, FZNROT
C      REAL*8 DBCOEF, DZCOEF, PCOEFF
C      REAL*8 DAMPMX( NUMDOF, NUMDOF)
C*****
C      DUFFING'S EQN., J.M.T. THOMPSON,
C      "NONLINEAR DYNAMICS AND CHAOS",P.8,10.
C*****
C      DAMPMX( 1, 1) = 0.00D+00
C
C      RETURN
C      END
C*****
C      SUBROUTINE DEGVEC ( NUMELE, TIELEN, PINUMB, DEGVAL)
C*****
C      THIS SUBROUTINE DEFINES THE VECTOR OF TIME INTERVAL END
C      POSITIONS IN DEGREES
C*****

```

```

C          DECLARE VARIABLES
C*****
C      INTEGER NUMELE ! NUMBER OF TIME INTERVALS
C      INTEGER I      ! DO LOOP 10 COUNTER
C
C      REAL*8 PINUMB ! PI NUMBER
C      REAL*8 TIELEN ! TIME ELEMENT LENGTH
C      REAL*8 ANGRAD ! TIME ELEMENT END POSITION IN RADIANS
C      REAL*8 DEGVAL( NUMELE) ! VECTOR OF TIME INERVAL END POSITIONS
C
C      ANGRAD = TIELEN
C*****
C          CALCULATE THE VECTOR OF END POSITIONS
C*****
C      DO 10 I = 1, NUMELE
C          DEGVAL( I) = ANGRAD/PINUMB * 180.0D0
C          ANGRAD = ANGRAD + TIELEN
C      10 CONTINUE
C
C      RETURN
C      END
C
C*****
C      SUBROUTINE EXTEDV( NUSHFU, NUMDOF, NUDOFD, NELCSF, NLCSIZ, NEDSIZ,
C      &          ELDIMX, ELDIVE)
C*****
C
C      THIS SUBROUTINE EXTRACTS THE FULL DISPLACEMENT VECTOR
C      CORRESPONDING TO GIVEN TIME ELEMENT FROM THE ELEMENT
C      DISPLACEMENT MATRIX
C*****
C          DECLARE VARIABLES
C*****
C      INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
C      INTEGER NURODV ! ROW NUMBER IN THE DISPLACEMENT VECTOR
C      INTEGER NUDOFD !=NUMDOF*2: DOUBLE NUMBER OF TIME ELEMENTS
C      INTEGER NELCSF ! NUMBER OF TIME ELEMENTS CONSIDERED SO FAR
C      INTEGER NLCSIZ !=NUMDOF*NUSHFU: SIZE OF ELEMENT COEFF. MATRIX
C      INTEGER NRELMC ! NUMBER OF ROWS OF ELEM. COEF. MX. CONSID. SO FAR
C      INTEGER NRELF1 ! FIRST ROW OF ELEM. COEF. MX. FOR GIVEN TIME EL.
C      INTEGER NRELAS ! LAST ROW OF ELEM. COEF. MX. FOR GIVEN TIME EL.
C      INTEGER NEDSIZ !=NUSHFU*NUMELE: NUM. OF ROWS IN ELEMENT DISPL. MX
C      INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
C      INTEGER I, J ! DO LOOP COUNTERS
C      REAL*8 ELDIMX( NEDSIZ, NUMDOF) ! ELEMENT DISPLACEMENT MATRIX
C      REAL*8 ELDIVE( NLCSIZ) ! ELEMENT DISPLACEMENT VECTOR
C*****

```

```

C          EXTRACT THE DISPLACEMENT VECTOR CORRESPONDING
C          TO THE GIVEN TIME ELEMENT
C*****
      NURODV = 0
      NRELMC = NELCSF * NUSHFU
      NRELFI = NRELMC + 1
      NRELAS = NRELMC + NUSHFU
      DO 10 I = 1, NUMDOF
        DO 20 J = NRELFI, NRELAS
          NURODV = NURODV + 1
          ELDIVE( NURODV ) = ELDIMX( J, I)
20      CONTINUE
10      CONTINUE
C
      RETURN
      END
C
C*****
      DOUBLE PRECISION FUNCTION FICDF1 ( VARIAB)
C*****
C      THIS FUNCTION GENERATES THE INITIAL VALUES AT TIME NODES (IN
C      GLOBAL COORDINATES) FOR DOF #1
C*****
C      DECLARE VARIABLES
C*****
C      REAL*8 VARIAB ! FUNCTION VARIABLE
C*****
C      GENERATE THE NODAL VALUE
C*****
      FICDF1 = 1.0D0
C
      RETURN
      END
C
C*****
      SUBROUTINE FORCIN ( NUMDOF, CURAZI, CURDIS, CURVEL, AMPLIT, FREQ,
      +      FORCEV)
C*****
C      THIS SUBROUTINE CALCULATES THE FORCING FUNCTION VECTOR.
C      VARIABLES:
C      NUMDOF : NUMBER OF DEGREES OF FREEDOM (INPUT).
C      CURAZI : CURRENT AZIMUTH - TIME (INPUT).
C      CURDIS : CURRENT DISPLACEMENT VECTOR (INPUT).
C      CURVEL : CURRENT VELOCITY VECTOR (INPUT).
C      AMPLIT : FORCING FUNCTION AMPLITUDE (INPUT).
C      FREQ   : FORCING FUNCTION FREQUENCY (INPUT).
C      FORCEV  : FORCING FUNCTION VECTOR (OUTPUT).

```

```

C  GAM  : GAMMA (LOCK NUMBER).
C  TET0  : TETA 0 (PRETWIST).
C  XMU   : MU (ADVANCE RATIO).
C  XLAM  : LAMBDA (NONDIMENSIONAL INFLOW).
C  XLAMI : LAMDA I (NONDIMENSIONAL INDUCED VELOCITY).
C  XNU   : NU (FLAPPING FREQUENCY).
C *****
C
C          DECLARE VARIABLES.
C *****
C  INTEGER NUMDOF
C  REAL*8 CURAZI, CURDIS(NUMDOF), CURVEL(NUMDOF), AMPLIT, FREQ,
C  +   FORCEV( NUMDOF), GAM, TET0, XMU, XLAM, XLAMI, XNU, F1, F2,
C  +   F3, F4, F5
C *****
C
C          DUFFING'S EQN., J.M.T. THOMPSON,
C          "NONLINEAR DYNAMICS AND CHAOS", P.8,10.
C *****
C  FORCEV( 1) = AMPLIT * DCOS( CURAZI * FREQ)
C
C  RETURN
C  END
C
C *****
C  SUBROUTINE GAQPAW ( GAQPNT, GAQWGT)
C *****
C  THIS SUBROUTINE DEFINES VECTORS OF GAUSS QUADRATURE POINTS AND
C  WEIGHTS
C *****
C
C          DECLARE VARIABLES
C *****
C  INTEGER I ! DO LOOP 10 COUNTER
C  INTEGER J ! SYMMETRIC GAUSS WEIGHTS COUNTER
C  INTEGER I3 ! GAUSS WEIGHT DATA SPACER
C
C  REAL*8 GAUVEC( 6) ! VECTOR OF DATA FOR GAUSS POINTS & WEIGHTS
C  REAL*8 GAQPNT( 6) ! VECTOR OF GAUSS POINTS
C  REAL*8 GAQWGT( 6) ! VECTOR OF GAUSS WEIGHTS
C  DATA GAUVEC/.93246951420315D0, .66120938646627D0,
C  &      .23861918608320D0, .17132449237917D0,
C  &      .36076157304814D0, .46791393457269D0/
C
C  DO 10 I=1, 3
C  J = 7 - I
C  I3 = I + 3
C  GAQPNT(I) = - GAUVEC(I)
C  GAQPNT(J) = GAUVEC(I)
C  GAQWGT(I) = GAUVEC(I3)

```

```

      GAQWGT(J) = GAQWGT(I)
10 CONTINUE
C
      RETURN
      END
C
C*****
      SUBROUTINE GAUJOR ( NGLSIZ, NGLSPO, COEFMX, RHSVEC, A, B, X)
C*****
C      DECLARE VARIABLES
C*****
      INTEGER NGLSIZ ! GLOBAL MATRIX SIZE
      INTEGER NGLSPO ! GLOBAL MATRIX SIZE PLUS ONE, NUMBER OF COLUMNS
C      &          IN THE AUGMENTED MATRIX
      INTEGER M, K, I, J
      INTEGER JJ, IA, JA
C
      REAL*8 A( NGLSIZ, NGLSPO) ! AUGMENTED MATRIX
      REAL*8 COEFMX( NGLSIZ, NGLSIZ) ! GLOBAL COEFFICIENT MATRIX
      REAL*8 B( NGLSIZ, NGLSIZ) ! TEMPORARY MATRIX
      REAL*8 RHSVEC( NGLSIZ) ! RIGHT HAND SIDE VECTOR
      REAL*8 X( NGLSIZ) ! THE SOLUTION VECTOR
      REAL*8 AB
      REAL*8 BIG
      REAL*8 TEMP
C*****
C      GENERATE THE AUGMENTED MATRIX
C*****
      DO 20 IA = 1, NGLSIZ
        A( IA, NGLSPO) = RHSVEC( IA)
        DO 30 JA = 1, NGLSIZ
          A( IA, JA) = COEFMX( IA, JA)
        30 CONTINUE
      20 CONTINUE
C
      M = NGLSIZ + 1
      6 K = M - 1
      IF ( K.EQ. 1) GO TO 11
      JJ = 1
      BIG = DABS( A( 1, 1))
      DO 8 I = 2, K
        AB = DABS( A( I, 1))
        IF ( BIG - AB) 7,8,8
      7 BIG = AB
      JJ = I
      8 CONTINUE
      IF ( JJ - 1) 9,11,9

```

```

9 DO 10 J = 1, M
  TEMP = A( JJ, J)
  A( JJ, J) = A( 1, J)
10 A( 1, J) = TEMP
11 DO 12 J = 2, M
  DO 12 I = 2, NGLSIZ
12 B( I-1, J-1) = A( I, J) - A( 1, J) * A( I, 1) / A( 1, 1)
  DO 13 J = 2, M
13 B( NGLSIZ, J-1) = A( 1, J) / A( 1, 1)
  M = M - 1
  DO 14 J = 1, M
  DO 14 I = 1, NGLSIZ
14 A( I, J) = B( I, J)
  IF ( M - 1) 6,15,6
15 CONTINUE
  DO 90 I = 1, NGLSIZ
  X( I) = A( I, 1)
90 CONTINUE
  RETURN
  END
C
C*****
C  SUBROUTINE GEFSOV ( NUMELE, NUSHFU, NUMDOF, NEDSIZ, NFINSL,
C    &                ELDIMX, FINSLM)
C*****
C  THIS SUBROUTINE GENERATES THE FINAL SOLUTION VECTOR FROM THE
C  STACKED MATRIX OF ELEMENT DISPLACEMENTS.
C*****
C    DECLARE VARIABLES
C*****
C  INTEGER I, J, K! DO LOOP COUNTERS
C  INTEGER NEDSIZ !=NUSHFU*NUMELE: NUM. OF ROWS IN ELEMENT DISPL. MX
C  INTEGER NFINSL !=(NUSHFU -1)*NUMELE:NUM. OF ROWS IN FIN. SOL. MX.
C  INTEGER NRFSMX ! ROW NUMBER IN THE FINAL SOLUTION MATRIX
C  INTEGER NRWEFV ! ROW NUMBER OF THE EFFICIENT SOLUTION VECTOR
C  INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
C  INTEGER NUMELE ! NUMBER OF TIME ELEMENTS
C  INTEGER NURCSF ! NUMBER OF ELEMENT DISPL. MX. CONSIDERED SO FAR
C  INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
C
C  REAL*8 ELDIMX( NEDSIZ, NUMDOF) ! ELEMENT DISPLACEMENT MATRIX
C  REAL*8 FINSLM( NFINSL, NUMDOF) ! FINAL SOLUTION MATRIX
C*****
C    GENERATE THE FINAL SOLUTION VECTOR
C*****
C  NRWEFV = 0
C  DO 10 I = 1, NUMELE

```

```

      NURCSF = ( I - 1 ) * NUSHFU
      DO 20 J = 2, NUSHFU
        NRFSMX = NURCSF + J
        NRWEFV = NRWEFV + 1
        DO 30 K = 1, NUMDOF
          FINSLM( NRWEFV, K) = ELDIMX( NRFSMX, K)
30    CONTINUE
20    CONTINUE
10    CONTINUE
C
      RETURN
      END
C
C*****
      SUBROUTINE GENECM ( NUDOFD, NUMDOF, SINERM, SDAMPM, SSTIFM,
&      EMCKMX)
C*****
C      THIS SUBROUTINE GENERATES THE COEFFICIENT MATRIX
C*****
C      DECLARE VARIABLES
C*****
      INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
      INTEGER NUDOFD ! DOUBLE NUMBER OF DOF'S (INPUT)
      INTEGER I, J ! DO LOOP COUNTERS
      INTEGER ILOWER ! COUNTER FOR THE LOWER PART OF THE COEFF. MATRIX
      INTEGER JRIGHT ! COUNTER FOR THE UPPER PART OF THE COEFF. MATRIX
C
      REAL*8 SINERM( NUMDOF, NUMDOF) ! SYSTEM INERTIA MATRIX
      REAL*8 SDAMPM( NUMDOF, NUMDOF) ! SYSTEM DAMPING MATRIX
      REAL*8 SSTIFM( NUMDOF, NUMDOF) ! SYSTEM STIFFNESS MATRIX
      REAL*8 EMCKMX( NUDOFD, NUDOFD) ! COEFFICIENT MATRIX
C*****
C      GENERATE THE COEFFICIENT MATRIX
C*****
      DO 10 I = 1, NUMDOF
        ILOWER = NUMDOF + I
C
        DO 20 J = 1, NUMDOF
          JRIGHT = NUMDOF + J
          EMCKMX( I, J) = -SSTIFM( I, J)
          EMCKMX( ILOWER, J) = 0.0D0
          EMCKMX( I, JRIGHT) = -SDAMPM( I, J)
          EMCKMX( ILOWER, JRIGHT) = SINERM( I, J)
20    CONTINUE
C
      10 CONTINUE
C

```

```

RETURN
END
C
C*****
SUBROUTINE GENSHD ( NUSHFU, GPLCRD, TIELEN, SHFUNV, SHDERV)
C*****
C   THIS SUBROUTINE GENERATES THE VECTORS OF SHAPE FUNCTIONS
C   AND SHAPE FUNCTION DERIVATIVES
C*****
C           DECLARE VARIABLES
C*****
C   INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
C
C   REAL*8 GPLCRD ! GAUSSIAN POINT IN LOCAL COORDINATES
C   REAL*8 TIELEN ! TIME ELEMENT LENGTH
C   REAL*8 SHFUNV( NUSHFU) ! SHAPE FUNCTION VECTOR
C   REAL*8 SHDERV( NUSHFU) ! SHAPE FUNCTION DERIVATIVES VECTOR
C*****
C           VERIFY THE NUMBER OF SHAPE FUNCTIONS
C*****
C   IF ( NUSHFU .NE. 2) THEN
C       PRINT *, 'NUMBER OF SHAPE FUNCTIONS MUST BE NUSHFU = 2 !'
C       PRINT *, 'STOPPED IN SUBROUTINE GENSHD'
C       STOP
C   ENDIF
C*****
C           GENERATE SHAPE FUNCTION VECTORS AND
C           SHAPE FUNCTION DERIVATIVES VECTORS
C*****
C   SHFUNV( 2) = GPLCRD/TIELEN
C   SHFUNV( 1) = 1.0D0 - SHFUNV( 2)
C   SHDERV( 2) = 1.0D0/TIELEN
C   SHDERV( 1) = -SHDERV( 2)
C
C   RETURN
C   END
C
C*****
SUBROUTINE GICEDM ( NUMELE, NUSHFU, NUMDOF, NEDSIZ, NFINSL,
& FINSLM, ELDIMX)
C*****
C
C   THIS SUBROUTINE GENERATES THE STACKED MATRIX OF ELEMENT
C   DISPLACEMENTS FROM THE FINAL SOLUTION VECTOR.
C*****
C           DECLARE VARIABLES
C*****

```

```

INTEGER I, J, K! DO LOOP COUNTERS
INTEGER L, M, N! DO LOOP COUNTERS
INTEGER I1 ! DO LOOP COUNTER
INTEGER NEDSIZ !=NUSHFU*NUMELE: NUM. OF ROWS IN ELEMENT DISPL. MX
INTEGER NFINSL !=(NUSHFU -1)*NUMELE:NUM. OF ROWS IN FIN. SOL. MX.
INTEGER NRFSMX ! ROW NUMBER IN THE FINAL SOLUTION MATRIX
INTEGER NRFSP1 ! NRFSMX + 1
INTEGER NRWEFV ! ROW NUMBER OF THE EFFICIENT SOLUTION VECTOR
INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
INTEGER NUMELE ! NUMBER OF TIME ELEMENTS
INTEGER NUMEM1 ! NUMELE - 1
INTEGER NURCSF ! NUMBER OF ELEMENT DISPL. MX. CONSIDERED SO FAR
INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS

C
REAL*8 ELDIMX( NEDSIZ, NUMDOF) ! ELEMENT DISPLACEMENT MATRIX
REAL*8 FINSLM( NFINSL, NUMDOF) ! FINAL SOLUTION MATRIX
C*****
C      GENERATE THE FINAL SOLUTION VECTOR
C*****
NRWEFV = 0
NUMEM1 = NUMELE - 1
DO 10 I = 1, NUMEM1
  NURCSF = (I - 1) * NUSHFU
  DO 20 J = 2, NUSHFU
    NRFSMX = NURCSF + J
    NRWEFV = NRWEFV + 1
    DO 30 K = 1, NUMDOF
      ELDIMX( NRFSMX, K) = FINSLM( NRWEFV, K)
30  CONTINUE
20  CONTINUE
C*****
C      FILL THE EXTRA ROW OF ELDIMX
C*****
NRFSMP1 = NRFSMX + 1
DO 40 L = 1, NUMDOF
  ELDIMX( NRFSMP1, L) = FINSLM( NRWEFV, L)
40  CONTINUE
10 CONTINUE
C*****
C      FILL THE ROWS IN ELDIMX CORRESPONDING TO THE
C      LAST TIME ELEMENT
C*****
NURCSF = NUMEM1 * NUSHFU
DO 50 M = 2, NUSHFU
  NRFSMX = NURCSF + M
  NRWEFV = NRWEFV + 1
  DO 60 N = 1, NUMDOF

```

```

        ELDIMX( NRFSMX, N) = FINSLM( NRWEFV, N)
60  CONTINUE
50  CONTINUE
C*****
C          FILL THE FIRST ROW OF ELDIMX
C*****
        NRFSPI = NRFSMX + 1
        DO 70 I1 = 1, NUMDOF
            ELDIMX( 1, I1) = FINSLM( NRWEFV, I1)
70  CONTINUE
C
        RETURN
        END
C
C*****
C          SUBROUTINE INITMX ( NUMROW, NUMCOL, TINMAT)
C*****
C          THIS SUBROUTINE INITIALIZES A MATRIX
C          I, J : DO LOOP COUNTERS
C          NUMROW : NUMBER OF ROWS
C          NUMCOL : NUMBER OF COLUMNS
C          TINMAT : INITIALIZED MATRIX
C*****
C          DECLARE VARIABLES
C*****
        INTEGER I, J, NUMROW, NUMCOL
        REAL*8 TINMAT( NUMROW, NUMCOL)
C
        DO 10 I = 1, NUMROW
C
            DO 20 J = 1, NUMCOL
                TINMAT( I, J) = 0.0D+00
            20  CONTINUE
        10  CONTINUE
C
        RETURN
        END
C
C*****
C          SUBROUTINE INITVE ( NUMDOF, DISTN)
C*****
C          THIS SUBROUTINE INITIALIZES VECTOR OF DISPLACEMENTS.
C          VARIABLES:
C          I : DO LOOP 10 COUNTER.
C          NUMDOF : NUMBER OF DOF'S.
C          DISTN : INITIALIZED VECTOR.

```

```

C*****
C          DECLARE VARIABLES.
C*****
C      INTEGER I, NUMDOF
C      REAL*8 DISTN( NUMDOF)
C
C      DO 10 I = 1, NUMDOF
C          DISTN( I) = 0.0D+00
C      10 CONTINUE
C
C      RETURN
C      END
C
C*****
C      SUBROUTINE INERTI ( NUMDOF, MASSMX)
C*****
C      THIS SUBROUTINE CALCULATES THE INERTIA MATRIX.
C*****
C      VARIABLES:
C      NUMDOF : NUMBER OF DEGREES OF FREEDOM (INPUT).
C      MASSMX : MASS MATRIX (OUTPUT).
C*****
C          DECLARE VARIABLES.
C*****
C      INTEGER NUMDOF
C      REAL*8 MASSMX( NUMDOF, NUMDOF)
C*****
C          DUFFING'S EQN., J.M.T. THOMPSON,
C          "NONLINEAR DYNAMICS AND CHAOS", P.8,10
C*****
C      MASSMX( 1, 1) = 1.0D+00
C
C      RETURN
C      END
C
C*****
C      SUBROUTINE MATAD (A, B, C, N, M)
C      IMPLICIT REAL*8(A-H, O-Z)
C      DIMENSION A(N, M), B(N, M), C(N, M)
C      DO 10 IR = 1, N
C      DO 10 IC = 1, M
C          C(IR, IC) = A(IR, IC) + B(IR, IC)
C      10 CONTINUE
C      RETURN
C      END
C*****
C      SUBROUTINE MATSUB (A, B, C, N, M)

```

```

      IMPLICIT REAL*8(A-H, O-Z)
      DIMENSION A(N,M), B(N,M), C(N,M)
      DO 10 IR = 1, N
      DO 10 IC = 1, M
      C(IR, IC) = A(IR, IC) - B(IR, IC)
10 CONTINUE
      RETURN
      END
C
C*****
      SUBROUTINE MATMUL (A, B, C, L, M, N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(L,M), B(M,N), C(L,N)
      DO 10 I=1, L
      DO 10 J=1, N
      C(I, J) = 0.0D0
      DO 10 K = 1, M
      C(I, J) = C(I, J) + A(I, K) * B(K, J)
10 CONTINUE
      RETURN
      END
C
C*****
      SUBROUTINE SHFDRM ( NLCSIZ, NUDOFD, NUMDOF, NUSHFU, SHFUNV,
&          SHDERV, SHDERM)
C*****
C      THIS SUBROUTINE GENERATES THE MATRIX CONTAINING THE SHAPE
C      FUNCTIONS AND THE DERIVATIVES OF THE SHAPE FUNCTIONS
C*****
C          DECLARE VARIABLES
C*****
      INTEGER NLCSIZ !=NUMDOF*NUSHFU, COEFFICIENT MATRIX SIZE
      INTEGER NUDOFD ! DOUBLE NUMBER OF DOF'S (INPUT)
      INTEGER NUMDOF ! NUMBER OF DOF'S
      INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
      INTEGER NMDFM1 ! NUMBER OF CONSIDERED DOF'S
      INTEGER ILOWER ! ROW NUMBER IN THE LOWER PART OF COEFF. MATRIX
      INTEGER I, J ! DO LOOP COUNTERS
      INTEGER NUMCOL ! PROPER NUMBER OF A COLUMN
C
      REAL*8 SHDERM( NUDOFD, NLCSIZ) ! MATRIX WITH SHAPE FUNCTIONS AND
C          DERIVATIVES OF SHAPE FUNCTIONS
      REAL*8 SHFUNV( NUSHFU) ! SHAPE FUNCTION VECTORS
      REAL*8 SHDERV( NUSHFU) ! SHAPE FUNCTION DERIVATIVES VECTOR
C*****
C          INITIALIZE THE MATRIX CONTAINING
C          SHAPE FUNCTIONS AND DERIVATIVES OF

```

```

C          SHAPE FUNCTIONS
C*****
C          CALL INITMX ( NUMDOF, NLCSIZ, SHDERM)
C*****
C          GENERATE THE MATRIX CONTAINING
C          SHAPE FUNCTIONS AND DERIVATIVES OF
C          SHAPE FUNCTIONS
C*****
C          DO 10 I = 1, NUMDOF
C             NMDFM1 = I - 1
C             ILOWER = NUMDOF + I
C
C             DO 20 J = 1, NUSHFU
C                NUMCOL = NMDFM1 * NUSHFU + J
C                SHDERM( I, NUMCOL) = SHFUNV( J)
C                SHDERM( ILOWER, NUMCOL) = SHDERV( J)
C             20 CONTINUE
C          10 CONTINUE
C          RETURN
C          END
C
C*****
C          SUBROUTINE STIFFN ( NUMDOF, CURAZI, CURDIS, CURVEL,
C          &          FZNROT, STIFMX )
C*****
C          THIS SUBROUTINE CALCULATES THE LINEAR PART OF THE STIFFNESS
C          MATRIX.
C*****
C          VARIABLES:
C          NUMDOF : NUMBER OF DEGREES OF FREEDOM (INPUT).
C          CURAZI : CURRENT AZIMUTH - TIME (INPUT).
C          CURDIS : CURRENT DISPLACEMENT VECTOR (INPUT).
C          CURVEL : CURRENT VELOCITY VECTOR (INPUT).
C          COEFF : STIFFNESS TERM COEFFICIENT - USED FOR 1 DOF (INPUT).
C          STIFMX : STIFFNESS MATRIX (OUTPUT).
C          SPRIN1 : SPRING #1 CONSTANT.
C          SPRIN2 : SPRING #2 CONSTANT.
C          CA1   : COEFFICIENT OF NONLINEARITY,  $K_1 = CA1 * X + CB1$ 
C          CB1   : COEFF. OF NONLIN..
C          CA2   : COEFFICIENT OF NONLINEARITY,  $K_2 = CA2 * X + CB2$ 
C          CB2   : COEFF. OF NONLIN..
C          FBNROT : NONDIMENSIONAL, NONROTATING FLAP NATURAL FREQUENCY.
C          FZNROT : NONDIMENSIONAL, NONROTATING LEAD-LAG NATURAL FREQUENCY.
C

```

```

COMMON /STIFCF/ CA1, CB1, CA2, CB2
C
C*****
C
C          DECLARE VARIABLES.
C*****
C      INTEGER NUMDOF
C      REAL*8 CURAZI, CURDIS(NUMDOF), CURVEL(NUMDOF), COEFF
C      REAL*8 FBNROT, FZNROT
C      REAL*8 STIFMX( NUMDOF, NUMDOF), SPRIN1, SPRIN2, CA1, CB1, CA2, CB2
C*****
C          DUFFING'S EQN., J.M.T. THOMPSON,
C          "NONLINEAR DYNAMICS AND CHAOS", P.8,10.VAR11250
C*****
C
C      STIFMX( 1, 1) = 0.0D+00
C
C      RETURN
C      END
C
C*****
C      SUBROUTINE STIFNL ( NUMDOF, CURAZI, CURDIS, CURVEL,
C      &          FZNROT, AMPLIT, FREQUE, STIFMX )
C*****
C
C      THIS SUBROUTINE CALCULATES THE DERIVATIVE W.R.T. DISPLACEMENT
C      VECTOR OF THE NONLINEAR FORCING FUNCTION VECTOR.
C*****
C      REMEMBER THAT A NEGATIVE VALUE OF DERIVATIVE MUST BE GENERATED.
C      THE REASON IS THAT THE DERIVATIVE IS ADDED TO STIFFNESS
C      MATRIX, NOT SUBTRACTED AS IT SHOULD !!!
C*****
C      VARIABLES:
C      NUMDOF : NUMBER OF DEGREES OF FREEDOM (INPUT).
C      CURAZI : CURRENT AZIMUTH - TIME (INPUT).
C      CURDIS : CURRENT DISPLACEMENT VECTOR (INPUT).
C      CURVEL : CURRENT VELOCITY VECTOR (INPUT).
C      COEFF : STIFFNESS TERM COEFFICIENT - USED FOR 1 DOF (INPUT).
C      STIFMX : STIFFNESS MATRIX (OUTPUT).
C      SPRIN1 : SPRING #1 CONSTANT.
C      SPRIN2 : SPRING #2 CONSTANT.
C      CA1 : COEFFICIENT OF NONLINEARITY,  $K_1 = CA1 * X + CB1$ 
C      CB1 : COEFF. OF NONLIN..
C      CA2 : COEFFICIENT OF NONLINEARITY,  $K_2 = CA2 * X + CB2$ 
C      CB2 : COEFF. OF NONLIN..
C      FBNROT : NONDIMENSIONAL, NONROTATING FLAP NATURAL FREQUENCY.
C      FZNROT : NONDIMENSIONAL, NONROTATING LEAD-LAG NATURAL FREQUENCY.
C

```

```

COMMON /STIFCF/ CA1, CB1, CA2, CB2
C
C*****
C
C      DECLARE VARIABLES.
C*****
C      INTEGER NUMDOF
C      REAL*8 CURAZI, CURDIS(NUMDOF), CURVEL(NUMDOF), COEFF
C      REAL*8 FBNROT, FZNROT, AMPLIT, FREQUE
C      REAL*8 STIFMX( NUMDOF, NUMDOF), SPRIN1, SPRIN2, CA1, CB1, CA2, CB2
C*****
C      DUFFING'S EQN., J.M.T. THOMPSON,
C      "NONLINEAR DYNAMICS AND CHAOS", P.8,10.
C*****
C
C      STIFMX( 1, 1) = 1D+00 * (3.0D+00 * CURDIS( 1)**2)
C
C      RETURN
C      END
C
C*****
C      SUBROUTINE STMXNL ( NUMDOF, CURAZI, CURDIS, CURVEL,
C      &      FZNROT,AMPLIT, FREQUE, STIFMX )
C*****
C
C      THIS SUBROUTINE CALCULATES THE NONLINEAR PART OF STIFFNESS
C      MATRIX.
C*****
C      VARIABLES:
C      NUMDOF : NUMBER OF DEGREES OF FREEDOM (INPUT).
C      CURAZI : CURRENT AZIMUTH - TIME (INPUT).
C      CURDIS : CURRENT DISPLACEMENT VECTOR (INPUT).
C      CURVEL : CURRENT VELOCITY VECTOR (INPUT).
C      COEFF : STIFFNESS TERM COEFFICIENT - USED FOR 1 DOF (INPUT).
C      STIFMX : STIFFNESS MATRIX (OUTPUT).
C      SPRIN1 : SPRING #1 CONSTANT.
C      SPRIN2 : SPRING #2 CONSTANT.
C      CA1 : COEFFICIENT OF NONLINEARITY,  $K_1 = CA1 * X + CB1$ 
C      CB1 : COEFF. OF NONLIN..
C      CA2 : COEFFICIENT OF NONLINEARITY,  $K_2 = CA2 * X + CB2$ 
C      CB2 : COEFF. OF NONLIN..
C      FBNROT : NONDIMENSIONAL, NONROTATING FLAP NATURAL FREQUENCY.
C      FZNROT : NONDIMENSIONAL, NONROTATING LEAD-LAG NATURAL FREQUENCY.
C*****
COMMON /STIFCF/ CA1, CB1, CA2, CB2
C*****
C      DECLARE VARIABLES.
C*****

```

```

INTEGER NUMDOF
REAL*8 CURAZI, CURDIS(NUMDOF), CURVEL(NUMDOF), COEFF
REAL*8 FBNROT, FZNROT, AMPLIT, FREQUE
REAL*8 STIFMX( NUMDOF, NUMDOF), SPRIN1, SPRIN2, CA1, CB1, CA2, CB2
C*****
C          DUFFING'S EQN., J.M.T. THOMPSON,
C          "NONLINEAR DYNAMICS AND CHAOS", P.8,10.
C*****
      STIFMX( 1, 1) = CURDIS( 1) ** 2
C
      RETURN
      END
C
C*****
      SUBROUTINE UDVEDM ( NUMELE, NUSHFU, NUMDOF, NEDSIZ, NUNDIS,
&          UNDIVE, ELDIMX)
C*****
C      THIS SUBROUTINE CONVERTS THE UNKNOWN DISPLACEMENT VECTOR IN THE
C      MORE EFFICIENT FORM TO ELEMENT DISPLACEMENT MATRIX (ELDIMX)
C*****
C          DECLARE VARIABLES
C*****
      INTEGER NEDSIZ !=NUSHFU*NUMELE, NUMBER OF ROWS IN ELEMENT DISPL.
      MATRIX (INPUT)
      INTEGER NUNDIS !=(NUSHFU-1)*NUMDOF*NUMELE: NUMBER OF UNKNOWN
      DISPLACEMENTS (INPUT)
      INTEGER NUSHFU ! NUMBER OF SHAPE FUNCTIONS
      INTEGER NUMDOF ! NUMBER OF DEGREES OF FREEDOM
      INTEGER NUMELE ! NUMBER OF TIME ELEMENTS
      INTEGER NUMEM1 ! NUMBER OF TIME ELEMENTS MINUS ONE
      INTEGER NUSHM1 ! NUMBER OF SHAPE FUNCTIONS MINUS ONE
      INTEGER NREDMX ! ROW NUMBER IN ELEMENT DISPLACEMENT MATRIX
      INTEGER NURCSF ! NUMBER OF ROWS CONSIDERED SO FAR
      INTEGER NELCSF ! NUMBER OF ELEMENTS CONSIDERED SO FAR
      INTEGER NDFCSF ! NUMBER OF DOF'S CONSIDERED SO FAR
      INTEGER NRUDVE ! ROW NUMBER IN UNKNOWN DISPLACEMENTS VECTOR
      INTEGER I, J, K! DO LOOP COUNTERS
      INTEGER L, M, N! DO LOOP COUNTERS
C
      REAL*8 ELDIMX( NEDSIZ, NUMDOF) ! ELEMENT DISPLACEMENT MATRIX
      REAL*8 UNDIVE( NUNDIS) ! UNKNOWN DISPLACEMENT VECTOR
C*****
C          TRANSFER ELEMENTS FROM THE UNKNOWN DISPLACEMENT
C          VECTOR UP TO THE BEFORE-LAST TIME ELEMENT
C*****
      NUMEM1 = NUMELE - 1
      NUSHM1 = NUSHFU - 1

```

```

NUNDIS = NUSHM1 * NUMDOF * NUMELE
C
DO 10 I = 1, NUMEM1
  NURCSF = ( I - 1 ) * NUSHM1 * NUMDOF
  NELCSF = I - 1
  DO 20 J = 1, NUMDOF
    NDFCSF = ( J - 1 ) * NUSHM1
    NREDMX = NELCSF * NUSHFU + 1
    DO 30 K = 1, NUSHM1
      NRUDVE = NURCSF + K + NDFCSF
      NREDMX = NREDMX + 1
      ELDIMX( NREDMX, J ) = UNDIVE( NRUDVE )
    30 CONTINUE
    NREDMX = NREDMX + 1
    ELDIMX( NREDMX, J ) = UNDIVE( NRUDVE )
  20 CONTINUE
10 CONTINUE
C*****
C      TRANSFER ELEMENTS FROM THE UNKNOWN DISPLACEMENT
C      VECTOR FOR THE LAST TIME ELEMENT
C*****
  NURCSF = NUMEM1 * NUSHM1 * NUMDOF
  DO 40 L = 1, NUMDOF
    NDFCSF = ( L - 1 ) * NUSHM1
    NREDMX = NUMEM1 * NUSHFU + 1
    DO 50 M = 1, NUSHM1
      NRUDVE = NURCSF + M + NDFCSF
      NREDMX = NREDMX + 1
      ELDIMX( NREDMX, L ) = UNDIVE( NRUDVE )
    50 CONTINUE
    ELDIMX( 1, L ) = UNDIVE( NRUDVE )
  40 CONTINUE
C
  RETURN
END

```

VITA

Irawan Malik was born on October 15, 1958, in Belitung Island, South Sumatra, Indonesia. He graduated from Technology High School Pk. Pinang, Bangka Island, South Sumatra, in December 1977 majoring in mechanical engineering. The following January he entered Bandung Institute of Teachers' Training and Education in Bandung, West Java and in January 1983 received his Bachelor of Science degree in Mechanical Engineering Education. Between 1984 and 1985 he received additional training in mechanical engineering at the Polytechnic of Educational Development Center (PEDC) in Bandung. Since then he has been an instructor in the mechanical engineering department of Polytechnic of Sriwijaya University, Palembang, South Sumatra. He entered the department of mechanical and aerospace engineering of the University of Tennessee, Knoxville, in August 1993 and received his Master of Science degree in Mechanical Engineering in July 1995.