98

0-315-12413-X

National Library of Canada / Bibliothèque nationale du Canada

Canadian Theses Division / Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

60324

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

BERNHARD MICHAEL JATZECK

Date of Birth — Date de naissance

55-10-12

Country of Birth — Lieu de naissance

FEDERAL REPUBLIC OF GERMANY

Permanent Address — Résidence fixe

#305-115 REINDEER RD.
SASKATOON, SASK.
S7K 4W8

Title of Thesis — Titre de la thèse

NUMERICAL SOLUTION OF SOME SECOND-ORDER DIFFERENTIAL EQUATIONS

University — Université

UNIVERSITY OF ALBERTA

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

M.Sc.

Year this degree conferred — Année d'obtention de ce grade

1982

Name of Supervisor — Nom du directeur de thèse

DR. J. B. HADDOW

Date

82-08-06

Signature

Bernhard Michael Jatzeck

NL-91 (4/77)

THE UNIVERSITY OF ALBERTA

NUMERICAL SOLUTION OF SOME SECOND-ORDER DIFFERENTIAL EQUATIONS

by

BERNHARD MICHAEL JATZECK, B. SC., MEC. E. (ALTA.)

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

MECHANICAL ENGINEERING

EDMONTON, ALBERTA

FALL 1982

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR BERNHARD MICHAEL JATZECK, B. SC., MEC. E. (ALTA.)

TITLE OF THESIS NUMERICAL SOLUTION OF SOME SECOND-ORDER DIFFERENTIAL EQUATIONS

DEGREE FOR WHICH THESIS WAS PRESENTED MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED FALL 1982

(SIGNED) Bernhard Michael Jatzeck

PERMANENT ADDRESS:

#305 - 115 REINDEER RD.

SASKATOON, SASK.

S7K 4W8

DATED ...820806.........19...

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled NUMERICAL SOLUTION OF SOME SECOND-ORDER DIFFERENTIAL EQUATIONS submitted by BERNHARD MICHAEL JATZECK, B. SC., MEC. E. (ALTA.) in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE.

J.B. Haddow ....

Supervisor

....

....

Date 20th July/82 .....

ABSTRACT

The solution of some second-order differential equations using the method of piecewise linearization is examined. These are the undamped and damped hard spring equations (linear and nonlinear damping) and the Van der Pol and Mathieu equations.

Two methods of piecewise linearization are considered: solution by chords and solution by tangents. The two sets of results that are obtained are compared to the approximation to the exact solution of the undamped hard spring equation. The former method of linearization is chosen as the solution to the remainder of the equations that are examined.

Generally, the piecewise linearization solution gives a good agreement when compared to either the Runge-Kutta solution or the approximation to the exact solution (if available). Exceptions to this appear in some of the results for the Van der Pol and Mathieu equations.

The programmed piecewise linear methods are generally slower in execution time than either the Runge-Kutta solution or the approximation to the exact solution. The exceptions to this are the undamped hard spring equation and some instances for the Mathieu equation.

ACKNOWLEDGEMENTS

Several individuals contributed to the completion of this thesis. The following names I mention are among those whose efforts are deserving of my special thanks.

My supervisor, Dr. J. B. Haddow, was most helpful in the research and the writing. His guidance and advice in the material that was examined and the methods of solution that were used is greatly appreciated. I also wish to thank the other members of the examining committee for their efforts.

The personnel of Telsec Business Centre, Saskatoon, particularly Mrs. Carole Fontaine, were most patient in the preparation of the manuscript. I also wish to thank Mrs. Faye Magera, also of Saskatoon, for her initial efforts at this task.

The computer programming was a major endeavour and the advice of fellow graduate student Mr. Roger Toogood was a valuable asset. The advice of the consultants at the University of Alberta Computing Services, particularly Dr. Ron Torgerson, was also appreciated.

I am grateful to those department faculty and staff members for their efforts which allowed the completion of my degree. In particular, I thank Dr. D. G. Bellow, department chairman, for making it all possible.

A brief word of thanks also goes to some fellow students and some of the staff at the University of British Columbia, Department of Mechanical Engineering, from whom I learned much while I was there.

Finally, I wish to thank the many authors and/or publishers who were kind to allow me to use or cite material from their works.

TABLE OF CONTENTS

Chapter		Page
	ABSTRACT	iv
	ACKNOWLEDGEMENTS	v
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
	LIST OF SYMBOLS	xv
I.	INTRODUCTION	1
II.	HARD SPRING EQUATION (UNDAMPED)	5
	A. Preliminary Comments	5
	B. Approximation to Exact Solution	5
	C. Derivation of Piecewise Linear Solution by Chords	9
	D. Derivation of Piecewise Linear Solution by Tangents	16
	E. Results	23
III.	HARD SPRING EQUATION (LINEAR DAMPING)	35
	A. Preliminary Comments	35
	B. Derivation of Piecewise Linear Solution	37
	C. Runge-Kutta Solution	39
	D. Results	39
IV.	HARD SPRING EQUATION (NONLINEAR DAMPING)	58
	A. Preliminary Comments	58
	B. Derivation of Piecewise Linear Solution	58
	C. Results	62
V.	VAN DER POL EQUATION	79
	A. Preliminary Comments	79
	B. Derivation of Piecewise Linear Solution	80
	C. Results	87

TABLE OF CONTENTS (continued)

Chapter		Page
VI.	MATHIEU EQUATION	135
	A. Preliminary Comments	135
	B. Generation of Characteristic Number	135
	C. Series Approximation to Exact Solution	142
	D. Derivation of Piecewise Linear Solution	147
	E. Results	150
VII.	DISCUSSION	184
VIII.	CONCLUSIONS	197
IX.	RECOMMENDATIONS FOR FURTHER WORK	199
	A. Mathieu's Equation	199
	B. Two-Degree-of-Freedom Nonlinear Equations	200
	REFERENCES	201
	APPENDIX	206
	A. HSPRING	207
	B. HSPRINGC	214
	C. HSPRINGQ	221
	D. VANDERPOL	228
	E. MATHIEU	237
	VITA	247

LIST OF TABLES

Table		Page
2.1	Undamped Hard Spring Equation Periods for $a = 1.0$, $b = 0.15$	24
2.2	Undamped Hard Spring Equation Periods for $a = 1.0$, $b = 2.0$	26

LIST OF FIGURES

Figure		Page
1	Methods of Piecewise Linearization	4
2.1	Mass-Spring System (Undamped)	6
2.2	Approximation by Chords	9
2.3	Chord Segment Construction	12
2.4	Approximation by Tangents	17
2.5	Tangent Segment Construction	18
2.6	Solutions to Hard Spring Equation (Undamped) for $a = 1.0$, $b = 0.15$, $\Delta t = \tau_{\pi/4}/100.0$, $\Delta x = 0.20$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	27
2.7	Solutions to Hard Spring Equation (Undamped) for $a = 1.0$, $b = 0.15$, $\Delta t = \tau_{\pi/4}/100.0$, $\Delta x = 0.10$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	28
2.8	Solutions to Hard Spring Equation (Undamped) for $a = 1.0$, $b = 2.0$, $\Delta t = \tau_{\pi/4}/100.0$, $\Delta x = 0.20$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	29
2.9	Solutions to Hard Spring Equation (Undamped) for $a = 1.0$, $b = 2.0$, $\Delta t = \tau_{\pi/4}/100.0$, $\Delta x = 0.10$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	30
2.10	Execution Times for Solutions to Hard Spring Equation (Undamped) for $a = 1.0$, $b = 0.15$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	32
2.11	Execution Times for Solutions to Hard Spring Equation (Undamped) for $a = 1.0$, $b = 2.0$, $x(0) = 1.0$ $\dot{x}(0) = 0.0$	33
3.1	Mass-Spring System (Linear Damping)	36
3.2	Solutions to Hard Spring Equation (Linear Damping) for $a = 1.0$, $b = 0.15$, $\frac{C}{M} = 0.0$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	41
3.3	Solutions to Hard Spring Equation (Linear Damping) for $a = 1.0$, $b = 2.0$, $\frac{C}{M} = 0.0$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	43
3.4	Solutions to Hard Spring Equation (Linear Damping) for $a = 1.0$, $b = 0.15$, $\frac{C}{M} = 0.10$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	45

LIST OF FIGURES (cont'd)

Figure		Page
3.5	Solutions to Hard Spring Equation (Linear Damping) for a = 1.0, b = 2.0, $\frac{c}{M}$ = 0.10, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	47
3.6	Solutions to Hard Spring Equation (Linear Damping) for a = 1.0, b = 0.15, $\frac{c}{M}$ = 0.25, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	49
3.7	Solutions to Hard Spring Equation (Linear Damping) for a = 1.0, b = 2.0, $\frac{c}{M}$ = 0.25, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	51
3.8	Execution Times for Solutions to Hard Spring Equation (Linear Damping) for a = 1.0, $\frac{c}{M}$ = 0.0, x(0) = 1.0, $\dot{x}(0)$ = 0.0	53
3.9	Execution Times for Solutions to Hard Spring Equation (Linear Damping) for a = 1.0, $\frac{c}{M}$ = 0.10, x(0) = 1.0, $\dot{x}(0)$ = 0.0	54
3.10	Execution Times for Solutions to Hard Spring Equation (Linear Damping) for a = 1.0, $\frac{c}{M}$ = 0.25, x(0) = 1.0, $\dot{x}(0)$ = 0.0	55
4.1	Mass-Spring System (Nonlinear Damping)	59
4.2	Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, b = 0.15, $\frac{q}{M}$ = 0.0, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	64
4.3	Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, b = 2.0, $\frac{q}{M}$ = 0.0, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	66
4.4	Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, b = 0.15, $\frac{q}{M}$ = 0.10, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	68
4.5	Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, b = 2.0, $\frac{q}{M}$ = 0.10, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	70
4.6	Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, b = 0.15, $\frac{q}{M}$ = 0.25, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	72

LIST OF FIGURES (cont'd)

Figure		Page
4.7	Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, b = 2.0, $\frac{q}{M}$ = 0.25, Δx = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	74
4.8	Execution Times for Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, $\frac{q}{M}$ = 0.0, x(0) = 1.0, $\dot{x}(0)$ = 0.0	75
4.9	Execution Times for Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, $\frac{q}{M}$ = 0.10, x(0) = 1.0, $\dot{x}(0)$ = 0.0	76
4.10	Execution Times for Solutions to Hard Spring Equation (Nonlinear Damping) for a = 1.0, $\frac{q}{M}$ = 0.25, x(0) = 1.0, $\dot{x}(0)$ = 0.0	77
5.1	Typical Interval Layout (Van der Pol Equation)	82
5.2	Layout of Peak or Trough (Van der Pol Equation)	84
5.3	Solutions to Van der Pol Equation for μ = 0.25, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	89
5.4	Phase Trajectories for Van der Pol Equation for μ = 0.25, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0	91
5.5	Solutions to Van der Pol Equation for μ = 0.25, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 2.0, $\dot{x}(0)$ = 0.0	93
5.6	Phase Trajectories for Van der Pol Equation for μ = 0.25, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 2.0, $\dot{x}(0)$ = 0.0	95
5.7	Solutions to Van der Pol Equation for μ = 0.25, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 2.5, $\dot{x}(0)$ = 0.0	97
5.8	Phase Trajectories for Van der Pol Equation for μ = 0.25, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 2.5, $\dot{x}(0)$ = 0.0	99
5.9	Solutions to Van der Pol Equation for μ = 1.0, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 2.5, $\dot{x}(0)$ = 0.0	102
5.10	Phase Trajectories for Van der Pol Equation for μ = 1.0, Δt_P = 0.20, Δt_{RK} = 0.05, x(0) = 2.5, $\dot{x}(0)$ = 0.0	104

Figure		Page
5.11	Solutions to Van der Pol Equation for $\mu = 2.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	106
5.12	Phase Trajectories for Van der Pol Equation for $\mu = 2.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	108
5.13	Solutions to Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	110
5.14	Phase Trajectories for Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$	112
5.15	Solutions to Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.0$, $\dot{x}(0) = 0.0$	114
5.16	Phase Trajectories for Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.0$, $\dot{x}(0) = 0.0$	116
5.17	Solutions to Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	118
5.18	Phase Trajectories for Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	120
5.19	Solutions to Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.05$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	122
5.20	Phase Trajectories for Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.05$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	124
5.21	Solutions to Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.25$, $\Delta t_{RK} = 0.25$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	126
5.22	Phase Trajectories for Van der Pol Equation for $\mu = 5.0$, $\Delta t_P = 0.25$, $\Delta t_{RK} = 0.25$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$	128
5.23	Execution Times for Van der Pol Equation Solutions for $\mu = 0.25$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$	131
5.24	Execution Times for Van der Pol Equation Solutions for $\mu = 1.0$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$	132
5.25	Execution Times for Van der Pol Equation Solutions for $\mu = 2.0$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$	133

LIST OF FIGURES (cont'd)

Figure		Page
5.26	Execution Times for Van der Pol Equation Solutions for $\mu = 5.0$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$	134
6.1	Typical Interval Layout (Mathieu Equation)	149
6.2	Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 1.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	153
6.3	Solutions to Mathieu Equation for $a_{GUESS} = 10.0$, $q = 1.0$, $\Delta z = \pi/200.0$, $y(0) = 1.0$, $y'(0) = 0.0$	155
6.4	Solutions to Mathieu Equation for $a_{GUESS} = 30.0$, $q = 1.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	157
6.5	Solutions to Mathieu Equation for $a_{GUESS} = 10.0$, $q = 10.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	159
6.6	Solutions to Mathieu Equation for $a_{GUESS} = 30.0$, $q = 10.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	161
6.7	Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 10.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	163
6.8	Solutions to Mathieu Equation for $a_{GUESS} = 10.0$, $q = 20.0$, $\Delta z = \pi/200.0$, $y(0) = 1.0$, $y'(0) = 0.0$	165
6.9	Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 20.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	167
6.10	Solutions to Mathieu Equation for $a_{GUESS} = 30.0$, $q = 30.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	169
6.11	Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 30.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	171
6.12	Solutions to Mathieu Equation for $a_{GUESS} = 30.0$, $q = 40.0$, $\Delta z = \pi/200.0$, $y(0) = 1.0$, $y'(0) = 0.0$	173
6.13	Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 40.0$, $\Delta z = \pi/200.0$, $y(0) = 1.0$, $y'(0) = 0.0$	175
6.14	Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 40.0$, $\Delta z = \pi/400.0$, $y(0) = 1.0$, $y'(0) = 0.0$	178
6.15	Execution Times for Solutions to Mathieu Equation for $a_{GUESS} = 10.0$, $q = 1.0$, $y(0) = 1.0$, $y'(0) = 0.0$	180

LIST OF FIGURES (cont'd)

Figure		Page
6.16	Execution Times for Solutions to Mathieu Equation for $a_{GUESS} = 30.0$, $q = 20.0$, $y(0) = 1.0$, $y'(0) = 0.0$	181
6.17	Execution Times for Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 40.0$, $y(0) = 1.0$, $y'(0) = 0.0$	182
7.1	Solutions to Mathieu Equation for $a_{GUESS} = -50.0$, $q = 20.0$, $\Delta z = \pi/200.0$, $y(0) = 0.0$, $y'(0) = 1.0$	193
7.2	Stability Chart for Piecewise Linear Solution to Mathieu Equation	194

LIST OF SYMBOLS

a,b	Hard spring restoring force coefficients
a	Characteristic number (general use and for non-zero initial displacement and zero initial velocity)
a_{GUESS}	Initial guess at characteristic number
a_{IN}	Input value for characteristic number continued fraction (scan phase)
a_{OUT}	Output for preceding continued fraction (scan phase)
a_{MID}	Input value for characteristic number continued fraction (interval halving search)
a_{MIDOUT}	Output value for preceding continued fraction
a_{UP}, a_{LOW}	Upper and lower bounds, respectively (scan phase)
a_{2n}	Characteristic number (even order, non-zero initial displacement, and zero initial velocity)
A_{2r}	Coefficient for series approximation to Mathieu function (even order and preceding initial conditions)
a_{2n+1}	Characteristic number (odd order and preceding initial conditions)
A_{2r+1}	Coefficient for series approximation to Mathieu function (odd order and preceding initial conditions)
b_{2n+1}	Characteristic number (odd order, zero initial displacement and non-zero initial velocity)
B_{2n+1}	Coefficient for series approximation to Mathieu function (odd order and preceding initial conditions)
b_{2n+2}	Characteristic number (even order and preceding initial conditions)
B_{2r+2}	Coefficient for series approximation to Mathieu function (even order and preceding initial conditions)
A, B	General equation coefficients for linear segment solution (trigonometric)
c, p	Damping constant and angular frequency, respectively, for linear approximation
c	Damping constant

C, D	General equation coefficients for $\ddot{x}[(\Delta t)_i]$ (hyperbolic)
$f(a, q, n)$	General term for characteristic number continued fraction
$f(x, \dot{x}, t)$	Generalized term representing damping and restoring forces
F	Generalized restoring force
F_0	Restoring force at beginning of segment (method of chords)
F_1	Restoring force at end of segment (method of chords)
F_0'	Slope of tangent line at x_0 (method of tangents)
F_1'	Slope of tangent line at x_1 (method of tangents)
$F(\lambda, \phi)$	Incomplete elliptic integral of first kind
k_0	Slope of linear segment (method of chords)
$K(\lambda^2)$	Complete elliptic integral of first kind
m	Number of linear segments per quarter-cycle (piecewise linear solution)
M	System mass
n	Number of increments per quarter-cycle (approximation to exact solution of undamped hard spring equation)
n	c/2M
p	Trigonometric angular frequency for linear segment (piecewise linear solution)
p*	Damped natural frequency
p**	$\sqrt{n^2 - p^2}$ (Van der Pol equation)
p_m	Trigonometric angular frequency for mth segment
q	Nonlinear damping constant (hard spring equation, nonlinear damping)
q	Parameter for a (Mathieu equation)
s	ip

s_m	Hyperbolic angular frequency for mth segment (Mathieu equation)
t	Time
t_0	Cumulative time at beginning of linear segment (method of chords)
t_1	Cumulative time at end of linear segment (method of chords)
$(t_1 - t_0)_i$	Interim value for $t_1 - t_0$ for Newton-Raphson relations (damped hard spring equation)
t_{C0}	Cumulative time at beginning of linear segment (method of tangents)
t_{C1}	Cumulative time at end of linear segment (method of tangents)
x	Displacement
$x[(t_1 - t_0)_i]$	Displacement using $(t_1 - t_0)_i$ (damped hard spring equation)
$\dot{x}$	Velocity
$\dot{x}[(t_1 - t_0)_i]$	Velocity using $(t_1 - t_0)_i$ (damped hard spring equation)
$\dot{x}[(\Delta t)_i]$	Velocity using $(\Delta t)_i$
$\ddot{x}$	Acceleration
$\ddot{x}[(t_1 - t_0)_i]$	Acceleration using $(t_1 - t_0)_i$ (damped hard spring equation)
$\ddot{x}[(\Delta t)_i]$	Acceleration using $(\Delta t)_i$
x_0	Displacement at beginning of linear segment
x_1	Displacement at end of linear segment
$\dot{x}_0$	Velocity at beginning of linear segment (method of chords)
$\dot{x}_1$	Velocity at end of linear segment (method of chords)
x_0^*	Relative equilibrium point for linear segment (method of chords)

x_F	Estimated value for x_{11}
x_{PREV}	Previous value of estimated linear segment displacement
x_{C0}	Crossing point tangents from x_0 to x_1 for (method of tangents)
x_{C1}	Crossing point for tangents from x_1 and following linear segment (method of tangents)
x_{11}	Displacement at peak or trough
x_{12}	Displacement at end of linear segment in which peak or trough occurs
$\dot{x}_{AV}$	Average segment velocity (hard spring equation, nonlinear damping)
$\dot{x}_{C0}$	Velocity at x_{C0} (method of tangents)
$\dot{x}_{C1}$	Velocity at x_{C1} (method of tangents)
X_0	Initial system displacement
y	Displacement (Mathieu equation)
y'	Velocity (Mathiur equation)
y	Acceleration (Mathieu equation)
y_{m-1}, y_m	Displacements at end of (m - 1)th and mth segments, respectively
y'_{m-1}, y'_m	Velocities at end of (m - 1)th and mth segments, respectively
y_0	Displacement at beginning of segment
y'_0	Velocity at beginning of segment
Y_0	Initial system displacement (Mathieu equation)
Y'_0	Initial velocity (Mathieu equation)
z	Independent variable (Mathieu equation)
α, β	Coefficients for approximation to exact solution to Mathieu equation
α, β, γ	Trigonometric equation coefficients for finding $t_1 - t_0$ (undamped hard spring equation)

α_ν	Generalized numerator for continued fraction
β_ν	Generalized denominator for continued fraction
γ_{n+1}, γ_n	Sum of characteristic number continued fraction terms prior to (n + 1)th or nth terms
Δt	Time increment (approximation to exact solution to undamped hard spring equation)
Δt_p	Time interval for piecewise linear solution (Van der Pol equation)
Δt_{RK}	Time interval for Runge-Kutta solution (Van der Pol equation)
Δt_1	Interval from t_0 to time at which peak or trough occurs
Δt_2	Interval from peak or trough to t_1
$(\Delta t)_i$	Interim value of Δt_1 as used in Newton-Raphson method
Δx	Displacement increment (piecewise linear solution
Δz	Increment of z
θ	Solution to preceding trigonometric equation
λ^2	Parameter for $F(\lambda, \phi)$
μ	Equation parameter (Van der Pol equation)
ν	Term number for continued fraction
τ	Period of oscillation (Mathieu equation)
τ_π	Period of oscillation (approximation to exact solution of undamped hard spring equation)
$\tau_\pi/4$	Quarter-period of oscillation (approximation to exact solution of undamped hard spring equation)
ϕ	Amplitude for $F(\lambda, \phi)$
$\phi_{(r)}$	Order of approximation to exact solution of Mathieu equation
ϕ_{n+1}	Denominator for characteristic number continued fraction

ω	Angular frequency of oscillation for Jacobian elliptic function

I. INTRODUCTION

Oscillating systems are common in mechanical and electrical engineering, as well as in other areas of science such as chemistry and biology. These can often be described by using differential equations, either nonlinear or linear with time-varying coefficients.

Exact solutions for these equations are often difficult to obtain (if at all), but reasonably good approximations can usually be achieved using a variety of methods. Some of the better-known ones are variation of parameters [1], reversion [2], and perturbation [3].

One method that is mentioned in some textbooks but is often not given as much prominence as the others is that of piecewise linearization, the approximation of an equation by one that is linear. This approach is investigated in this thesis, applying it to a number of second-order differential equations, comparing the results with either an approximation to an exact solution (if available), such as [4], or to a solution obtained from a Runge-Kutta method [5].

What this method does is to approximate the equation being examined:

$$\ddot{x} + f(x,\dot{x},t) = 0 \;, \qquad (1.1)$$

by one that is linear. The terms corresponding to the damping and restoring forces in (1.1) are represented by $f(x,\dot{x},t)$. The linear equation is that for a mass-spring oscillator:

$$\left.\begin{aligned} &\ddot{x} + \frac{c}{M}\dot{x} + \frac{k}{M} = 0 \\ &\quad = \ddot{x} + 2nx + p^2x \;, \end{aligned}\right\} \qquad (1.2)$$

where:

c = Coefficient of damping,

M = System mass,

k = Spring constant,

p = Undamped natural frequency.

The key is to approximate the coefficients of x and $\dot{x}$ in (1.1) by constants that would correspond to n and p in (1.2).

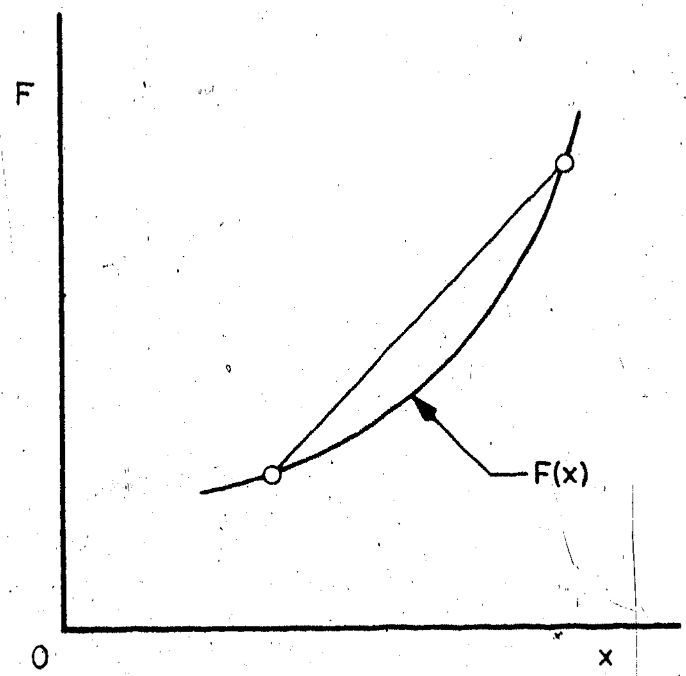
This is done by considering the restoring force-displacement curve of (1.1). For a small section of this curve, a straight line could be used as an approximation to it over the same region in which this section lies [6]. The curve can be thought of consisting of a sequence of these small sections, and each will have its own unique linear approximation.

The approximation of (1.1) by (1.2) will only be effective over a small interval of displacement or time. The initial and final displacements of this interval will determine the length of the straight line approximation of the restoring force-displacement curve just mentioned. The slope of this line will be k in (1.2). If the damping expression in (1.1) is nonlinear, c would have to be found by iteration over the interval of displacement or time, depending upon the nature of the nonlinearity. The solution to (1.2) can now be obtained, using the initial conditions for the interval in question. The end conditions of this interval become the initial values of the one following, and the procedure is repeated with new values of c and k, as well as n and p, being determined.

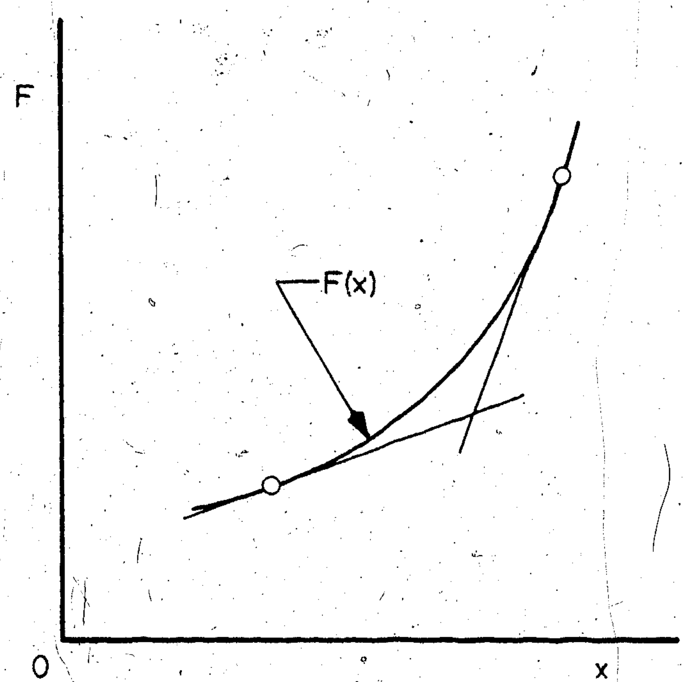
By successfully solving (1.2) for each interval, an approximate solution for (1.1) can be obtained, approaching the actual one as the intervals used become smaller.

The straight line approximation of the restoring force-displacement curve is accomplished by running a chord between the two endpoints of the section (Figure 1A) or tangents at the respective locations (Figure 1B). These points are located on the curve by knowing the initial and final displacements of the interval over which (1.2) is to be used. This force is shown in the figure as F and the displacement as x.

With this in mind, one can now apply this method to specific problems, as shall be seen in the following chapters.



A. CHORDS



B. TANGENTS

FIGURE 1 - METHODS OF PIECEWISE LINEARIZATION

II. HARD SPRING EQUATION (UNDAMPED)

A. Preliminary Comments

One of the simplest examples of the application of piecewise linearization is that of the hard undamped spring, the equation of which is:

$$\ddot{x} + ax + bx^3 = 0, \tag{2.1.1}$$

where:

$$a, b > 0. \tag{2.1.2}$$

See Figure 2.1 for details [7]. This is considered to be the equation for a hard spring since the restoring force increases quicker with respect to deflection than if it were linear [8]. The overall time span to be considered is the first quarter-cycle, since (2.1.1) is an odd function.

The following initial conditions will be used:

$$x(0) = X_0, \qquad \dot{x}(0) = 0. \tag{2.1.3}$$

The cases examined had values for a and b of 1.0 and 0.15 respectively, and 1.0 and 2.0, respectively, to see how the solutions behaved with a low and a high degree of nonlinearity.

B. Approximation to Exact Solution

An exact solution for this equation exists, making use of Jacobian elliptic functions. The derivations of this result can be found in any

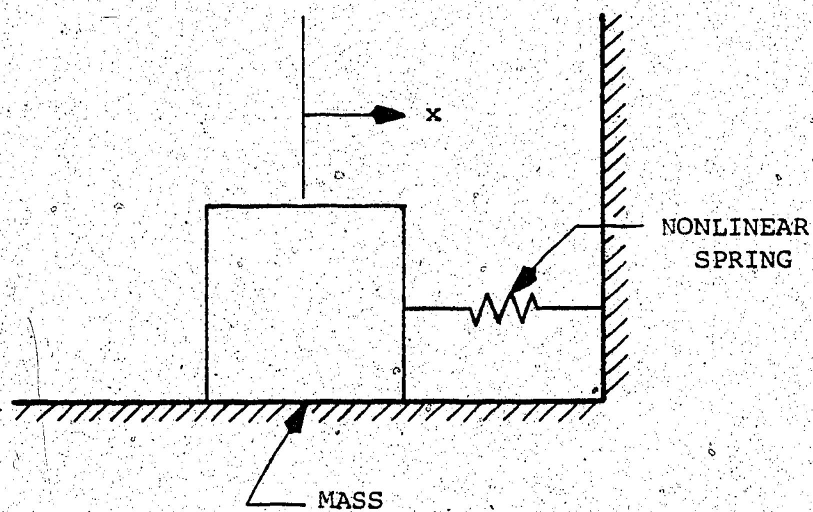


FIGURE 2.1 - MASS-SPRING SYSTEM (UNDAMPED)[7]

one of McLachlan [9], Cunningham [10], or Soudack [11]. This exact solution starts with:

$$t = \frac{1}{(a + bX_0^2)^{1/2}} \int_0^{\phi} \frac{d\psi}{(1 - \lambda^2 \sin^2\psi)^{1/2}}$$

$$= \frac{F(\lambda,\phi)}{(a + bX_0^2)^{1/2}}, \qquad (2.2.1)$$

where:

$$\phi = \cos^{-1}\left(\frac{x}{X_0}\right),$$

$$\lambda^2 = \frac{bX_0^2}{2(a + bX_0^2)},$$

$F(\lambda,\phi)$ = Incomplete elliptical integral of the first kind.

By applying the assumptions (2.1.2) and initial conditions (2.1.3), the solution to (2.1.1) is, together with (2.2.1) [12], [13]:

$$x = X_0 \mathrm{cn}(\lambda,\omega t), \qquad (2.2.2)$$

where:

$$\omega^2 = (a + bX_0^2). \qquad (2.2.3)$$

At the end of the first quarter-cycle, ϕ is $\pi/2$. Using the definition for some of the terms of (2.1.4), $F(\lambda,\phi)$ is known as the complete elliptic integral of the first kind, which is given the designation of $K(\lambda^2)$ [14]. But, it is not always practical to refer to tables each time one varies a and/or b, and so an easier means of obtaining this value should be used. An infinite series does exist for this and for a specific value for λ^2, one can use the following [15]:

$$K(\lambda^2) = \frac{\pi}{2}\Big[1 + \left(\frac{1}{2}\right)^2 (\lambda^2) + \left(\frac{1\cdot 3}{2\cdot 4}\right)^2 (\lambda^2) + \left(\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\right)^2 (\lambda^2)^3 + \ldots\Big], \quad (|\lambda^2| < 1), \tag{2.2.4}$$

substituting this for $F(\lambda,\phi)$ in (2.2.1)[9], [10], [11] for calculating the period, and combining this with the definition for λ^2 to generate the solution. Since only the first 100 terms of (2.2.4) will be calculated, the solution obtained, which will be used as a benchmark for the piecewise linear results, will only be an approximation to (2.2.2) [12], [13] and shall be subsequently be referred to as such.

A method to generate the values for $cn(\lambda,\omega t)$ is required. One does exist, making use of what is known as the "arithmetic-geometric mean" approach [16]. The computer program that was used for solving (2.1.1) accessed a subroutine library devised by the computing centre at the University of British Columbia which uses this method [4].

From McLachlan [17], the period of oscillation is:

$$\tau_\pi = \frac{4K(\lambda^2)}{(a + bX_0^{\,2})^{1/2}} \tag{2.2.5}$$

and so, for the quarter-cycle,

$$\tau_{\pi/4} = \frac{K(\lambda^2)}{(a + bX_0^2)^{1/2}} .$$

It should be noted that it is necessary to know the solution period as a basis for comparing the two piecewise linear methods that are to follow.

The actual solutions obtained by these methods are to be compared as well as with the one obtained by this approximation to the exact solution. The curve for the approximation to (2.2.2)[4], [15], [17] is obtained by dividing the quarter-cycle into n increments of time (where n is an integer) and calculating the displacement by incrementing the value of t by Δt, which is defined as follows:

$$\Delta t = \frac{K(\lambda^2)}{n(a + bX_0^2)^{1/2}} .$$

C. Derivation of Piecewise Linear Solution by Chords

The following method is based on Timoshenko, et. al. [18].

Assume that a restoring force-displacement curve for the system shown in Figure 2.1 can be approximated by a series of linear segments, as shown in Figure 2.2. The force is designated as F, while displacement is x. Assume further that the initial total displacement is divided into m equal sections of length Δx (m being an integer), with the total displacement being X_0, or:

X_0

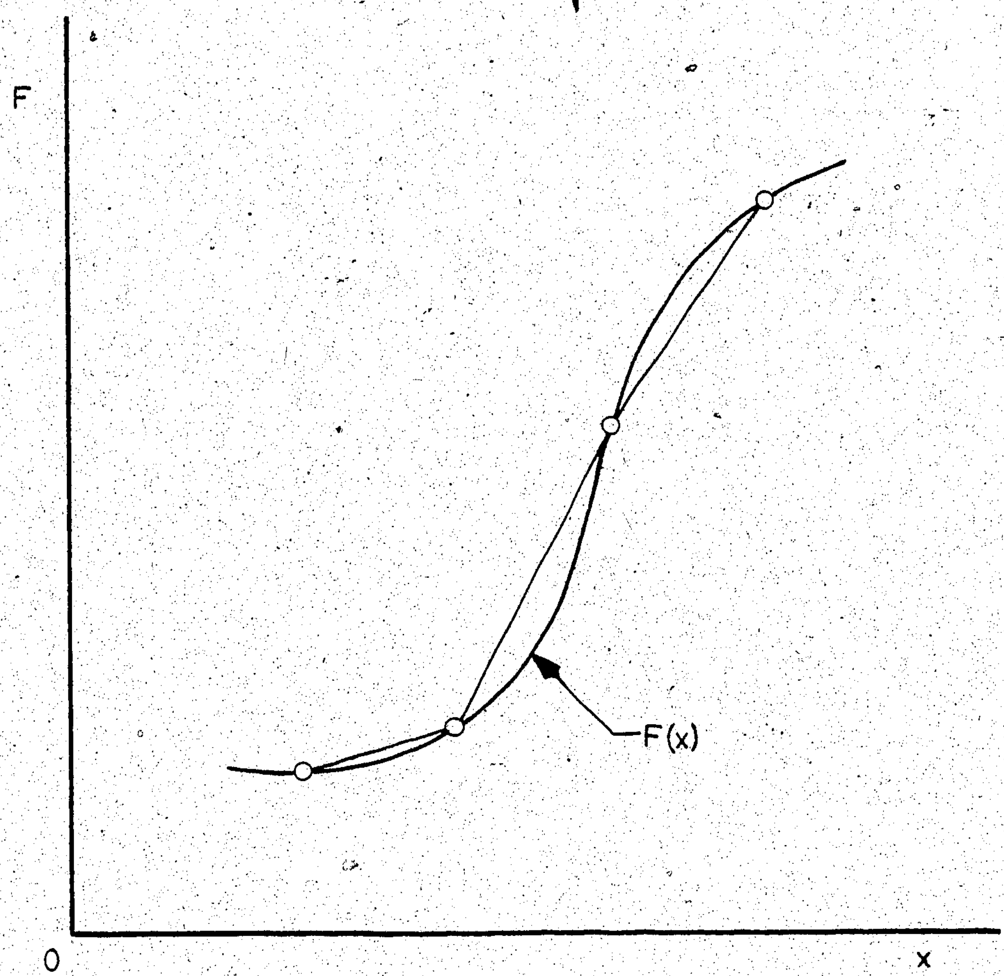


FIGURE 2.2 - APPROXIMATION BY CHORDS

Now consider, specifically for (2.1.1), two adjacent points (A and B) with displacements x_0 and x_1, respectively, and times t_0 and t_1, respectively (Figure 2.3). The system is initially at rest with displacement X_0, and then released. For this:

x_0 = Initial displacement of segment
$= x(t_0)$,

x_1 = Final displacement of segment
$= x(t_1)$,

k_0 = Slope of segment,

F_0 = Restoring force at beginning of segment
$= M(ax_0 + bx_0^3)$,

F_1 = Restoring force and end of segment
$= M(ax_1 + bx_1^3)$,

x_0^* = x - axis crossing point for chord,

M = System mass.

It can be also seen that:

$$\Delta x = x_0 - x_1$$

and:

$$\frac{k_0}{M} = \frac{F_0 - F_1}{M\Delta x}$$
$$= \frac{a(x_0 - x_1) + b(x_0^3 - x_1^3)}{x_0 - x_1} .$$

The equation of motion for line AB is (for $x_1 \leq x \leq x_0$, $t_1 \geq t \geq t_0$) [19]:

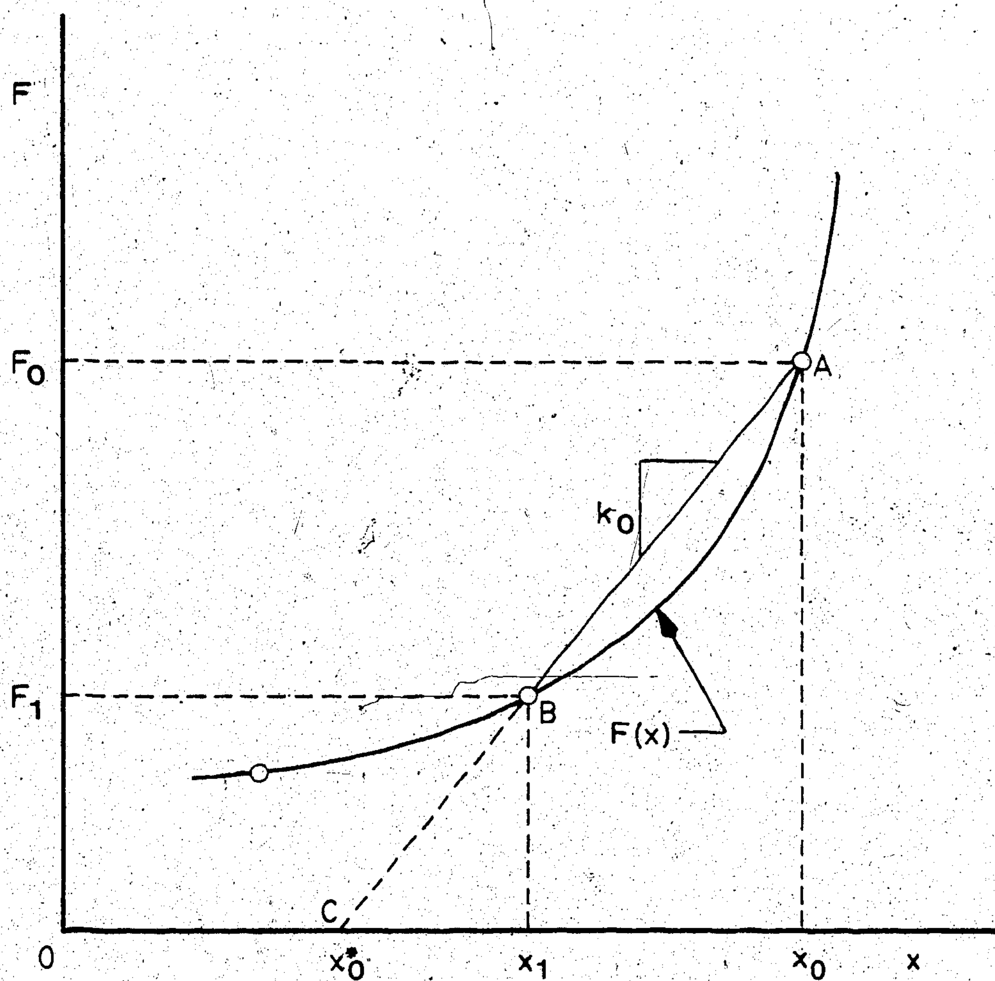


FIGURE 2.3 - CHORD SEGMENT CONSTRUCTION

$$\frac{d^2}{dt^2}(x - x_0^*) + \frac{k_0}{M}(x - x_0^*) = 0. \qquad (2.3.1)$$

Referring to Figure 2.3, the value of x_0^* is determined as follows. AB is extended to the displacement axis (line AC) and the ratio of BC to AC is found. This is done by using the point-of-division methods from plane analytic geometry [20]:

$$r = \frac{BC}{AC}$$

$$= \frac{F_1 - 0}{F_0 - 0}$$

$$= \frac{x_1 - x_0^*}{x_0 - x_0^*} ,$$

$$(x_0 - x_0^*)r = x_1 - x_0^* ,$$

$$x_0^*(1 - r) = x_1 - x_0 r ,$$

$$x_0^* = \frac{x_1 - x_0 r}{1 - r}$$

$$= \left(\frac{1}{1 - F_1/F_0}\right)\left[x_1 - (F_1/F_0)x_0\right] .$$

The solution to (2.2.1) is of the form [21]:

$$x = x_0^* + A\cos p(t - t_0) + B\sin pt(t - t_0) ,$$

where:

$$p = \sqrt{\frac{k_0}{M}} .$$

Differentiating gives the velocity [21]:

$$\dot{x} = -pA \sin p(t - t_0) + pB \cos p(t - t_0) .$$

Consider what happens at the beginning of the segment. The results are:

$$x_0 - x_0^* = A \cos p(0) + B \sin p(0) ,$$

$$A = x_0 - x_0^* ,$$

$$\dot{x}_0 = -pA \sin p(0) + pB \cos p(0) ,$$

$$B = \frac{\dot{x}_0}{p} .$$

Therefore, the general solution for any segment would be [21]:

$$x = x_0^* + (x_0 - x_0^*) \cos p(t - t_0) + \frac{\dot{x}_0}{p} \sin p(t - t_0) , \qquad (2.3.2)$$

with the velocity [21]:

$$\dot{x} = -p(x_0 - x_0^*) \sin p(t - t_0) + \dot{x}_0 \cos p(t - t_0) . \qquad (2.3.3)$$

For any given x_0, the value of t_0 is the total time taken to traverse the preceding individual segments. The question now is what, for any given segment, this time would be. From (2.3.2), it can be seen that for a given segment, the time taken to go from x_0 to x_1 would be $(t_1 - t_0)$, where, in this case, t_1 is the total time taken to reach x_1. Making the substitution of t_1 for t in (2.3.2) yields the following:

$$x_1 = x_0^* + (x_0 - x_0^*) \cos p(t_1 - t_0) + \frac{\dot{x}_0}{p} \sin p(t_1 - t_0) .$$

This takes the form of the trigonometric equation [22]:

$$\gamma = \alpha \cos \theta - \beta \sin \theta ,$$

where:

$$\left.\begin{aligned} \alpha &= x_0 - x_0^* \\ &> 0 , \\ \beta &= \left|\frac{\dot{x}_0}{p}\right| \\ &> 0 , \\ \gamma &= x_1 - x_0^* . \end{aligned}\right\} \qquad (2.3.4)$$

Two possible solutions are possible for this equation [22]:

$$\theta_1 = -\tan^{-1}\left(\frac{\beta}{\alpha}\right) - \cos^{-1}\left(\frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}\right) ,$$

$$\theta_2 = -\tan^{-1}\left(\frac{\beta}{\alpha}\right) + \cos^{-1}\left(\frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}\right) .$$

From $\tan^{-1} \delta = -\tan^{-1}(-\delta)$ [23] and the fact that $\gamma > 0$, plus $\theta > 0$ at a point right after $t = 0$,

$$\theta = p(t_1 - t_0)$$

$$= \tan^{-1}\left[-\frac{\beta}{\alpha}\right] + \cos^{-1}\left[\frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}\right], \qquad (2.3.5)$$

$$t_1 = t_0 + \frac{\theta}{p} .$$

using the definitions of (2.3.4).

Since the various values would have to be found iteratively, the quarter-period for this method would be the final value of t_1.

D. **Derivation of Piecewise Linear Solution by Tangents**

As for the method of chords, the reference for this section is Timoshenko, et. al. [18].

Referring to the oscillator shown in Figure 2.1, consider now the case of the restoring force-displacement curve being approximated by a series of linear segments, but using tangents instead of chords (Figure 2.4). The force is designated as F and displacement as x. Assume further that, as before, the length of the segment is Δx. For two consecutive segments, specifically for (2.1.1), the construction would look like Figure 2.5, with:

x_0 = Displacement at beginning of segment,

x_1 = Displacement at end of segment,

x_0^* = x - axis crossing point for tangent,

x_{C0} = Point where tangents from x_0 and x_1 cross

= $x(t_{C0})$,

x_{C1} = Point where tangent from x_1 crosses one from endpoint of next segment,

= $x(t_{C1})$,

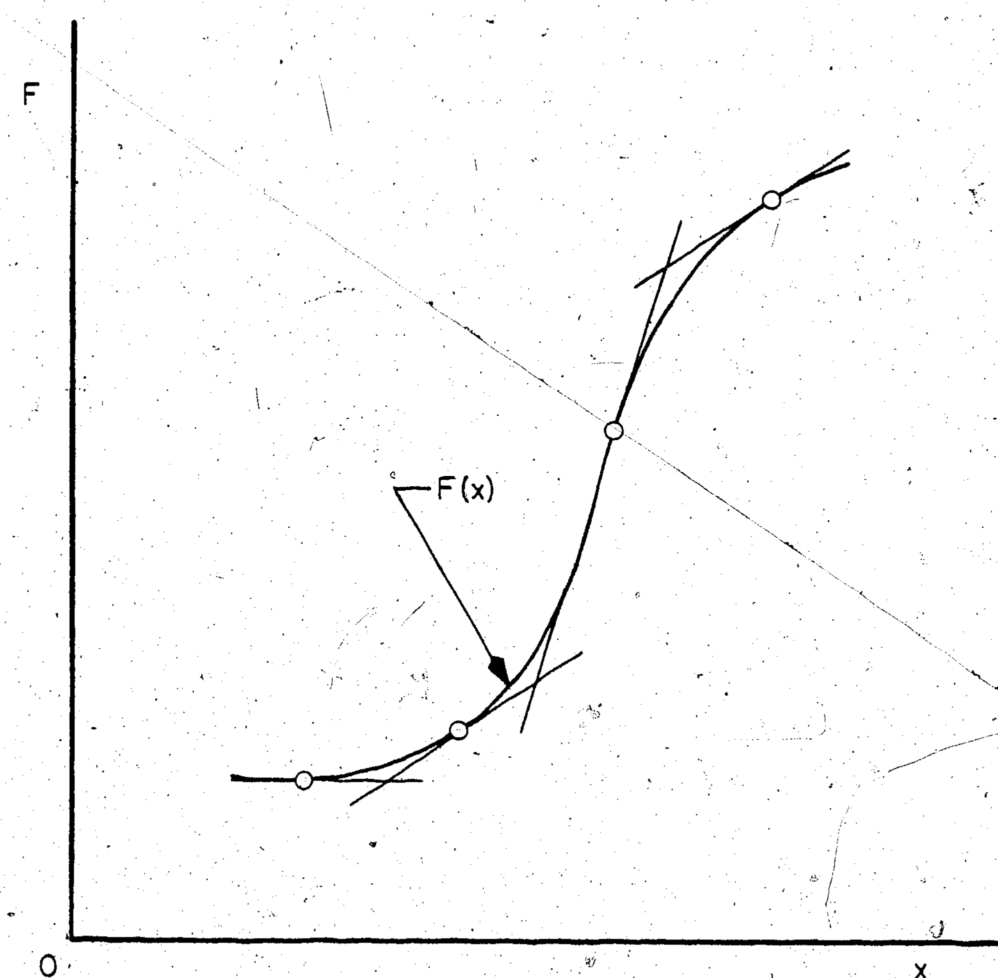


FIGURE 2.4 - APPROXIMATION BY TANGENTS

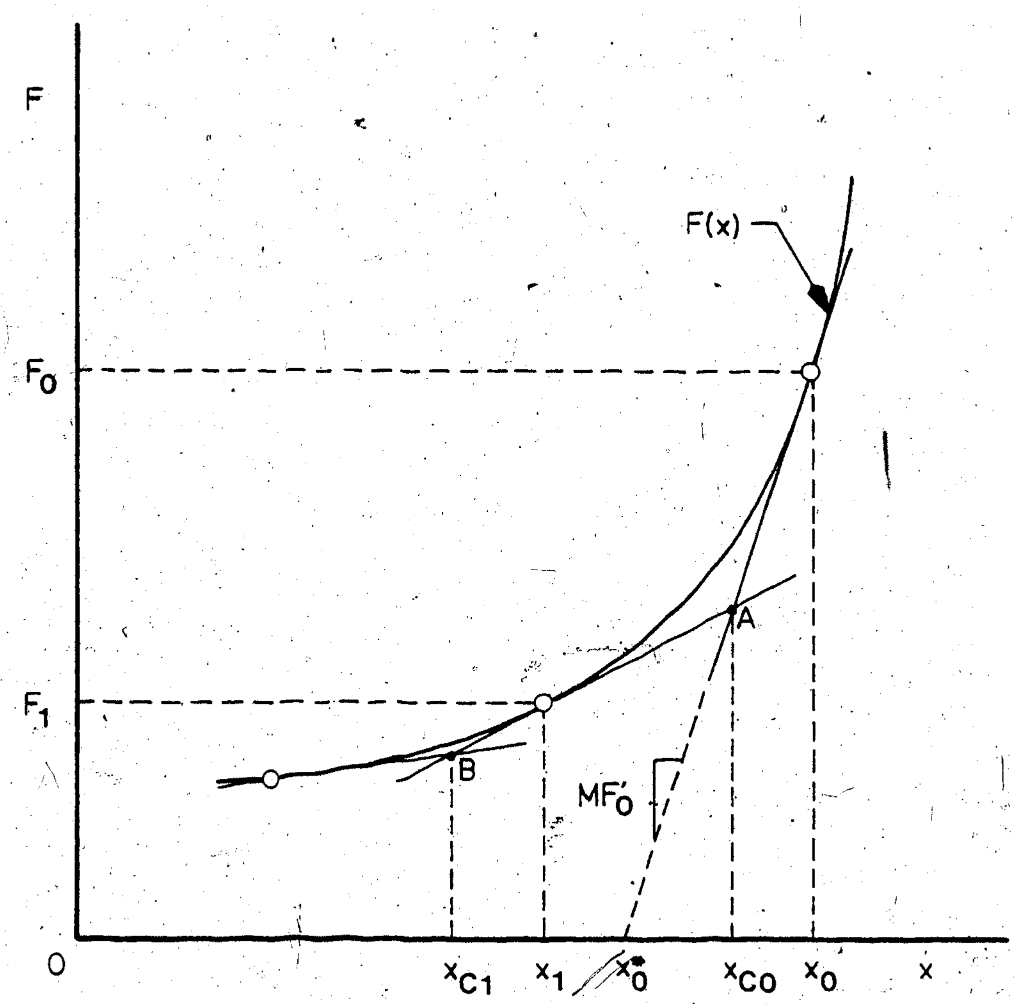


FIGURE 2.5 - TANGENT SEGMENT CONSTRUCTION

MF_0' = Slope of tangent line at x_0,

F_0, F_1 = Same as for method of chords.

Again, as before for m equal sectors of length Δx (with m being an integer):

$$\Delta x = \frac{x_0}{m}$$

$$= x_0 - x_1 .$$

In this case, motion is from one crossing point to the next (that is, from A to B), and so, the equation for line AB is (for $x_{C1} \leq x \leq x_{C0}$ and $t_{C1} \geq t \geq t_{C0}$) [19]:

$$\frac{d^2}{dt^2}(x - x_0^*) + \frac{k_0}{M}(x - x_0^*) = 0, \quad (2.4.1)$$

$$\frac{k_0}{M} = a + 3bx_0^2$$

$$= F_0' .$$

From Figure 2.5 it can be seen that:

$$F_0' = \frac{F_0 - 0}{M(x_0 - x_0^*)}$$

$$= \frac{ax_0 + bx_0^3}{x_0 - x_0^*}$$

$$x_0^* = x_0 - \frac{ax_0 + bx_0^3}{a + 3bx_0^2}$$

The solution to (2.4.1) is on the form:

$$x = x_0^* + A\cos p(t - t_{C0}) + B\cos p(t - t_{C0}) ,$$

where:

$$p = \sqrt{F_0'} .$$

The velocity is then [21]:

$$\dot{x} = -pA\sin p(t - t_{C0}) + pB\cos p(t - t_{C0}) .$$

Using the same method as before, but for $t = t_{C0}$, the solution for line AB is [21]:

$$x = x_0^* + (x_{C0} - x_0^*)\cos p(t - t_{C0}) + \frac{\dot{x}_{C0}}{p}\sin p(t - t_{C0}), \qquad (2.4.2)$$

and the associated velocity [21]:

$$\dot{x} = -p(x_{C0} - x_0^*)\sin p(t - t_{C0}) + \dot{x}_{C0}\cos p(t - t_{C0}) . \qquad (2.4.3)$$

Consider now what happens when the system reaches x_1. In other words:

$$x_{C1} = x_0^* + (x_{C0} - x_0^*)\cos p(t_{C1} - t_{C0}) + \frac{\dot{x}_{C0}}{p}\sin p(t_{C1} - t_{C0}) ,$$

$$\dot{x}_{C1} = -p(x_{C0} - x_0^*) \sin p(t_{C1} - t_{C0}) + \dot{x}_{C0} \cos p(t_{C1} - t_{C0}) \; .$$

The first expression takes on the form of the trigonometric equation referred to in the previous section [22], with the coefficients in this case being:

$$\left.\begin{aligned} \alpha &= x_{C0} - x_0^* \\ &> 0 \, , \\ \beta &= \left|\frac{\dot{x}_{C0}}{p}\right| \\ &> 0 \, , \\ \gamma &= x_{C1} - x_0^* \, . \end{aligned}\right\} \qquad (2.4.4)$$

Again, there are two possibilities for a solution with the same logic as before, such that, in this case, the result is [22]:

$$\begin{aligned} \theta &= \tan^{-1}\left(-\frac{\beta}{\alpha}\right) + \cos^{-1}\left(\frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}\right) \\ &= p(t_{C1} - t_{C0}) \, , \\ t_1 &= t_0 + \frac{\theta}{p} \, . \end{aligned} \qquad (2.4.5)$$

As before, the various segments have their individual values for t_1, and so, the quarter-period is the sum of all of these.

However, there is a slight difference here, owing to the fact that at each point, a tangent is drawn. The endpoints will not have full tangent lines, as can be seen by inspection. For the initial displacement, the equation of motion will be:

$$x = x_0^* + (x_{C0} + x_0^*) \cos pt .$$

For the end of the quarter period (or $x_1 = 0$), the result is:

$$-x_0^* = (x_{C0} - x_0^*) \cos p(t_{C1} - t_{C0})$$

$$+ \frac{\dot{x}_{C0}}{p} \sin p(t_{C1} - t_{C0}).$$

Each x_{C0} is found as follows. From plane analytic geometry [24], for an x-y plot and two points (x_1,y_1) and (x_2,y_2) with the respective slopes being:

$$m_1 = \frac{y - y_1}{x - x_1} ,$$

$$m_2 = \frac{y - y_2}{x - x_2} ,$$

$$y = m_1 x + (-m_1 x_1 + y_1) ,$$

$$= m_2 x + (-m_2 x_2 + y_2) .$$

Extending this to the current situation:

$$F = F_0' x_{C0} - F_0' x_0 + F_0$$

$$= F_1' x_{C0} - F_1' x_1 + F_1 ,$$

$$(F_0' - F_1')\, x_{C0} = F_0' x_0 - F_1' - (F_0 - F_1) ,$$

$$x_{C0} = \frac{F_0' x_0 - F_1' x_1 - (F_0 - F_1)}{F_0' - F_1'} ,$$

where:

$$F_1' = a + 3x_1^2 .$$

The two methods of solution had used different quantities for incrementation: time for the approximation to the exact solution and displacement for the piecewise linearization. For the former, time was the better of the two, as it is one variable of the argument. For the piecewise linearization, it was easier to use displacement, since one can readily find the values for x_0, k_0, and so on, since these are based upon a value for Δx. By using time, this does not become immediately apparent, and it would be difficult to determine the end of an interval. First the end of the segment would have to be guessed, and all the other variables used in the construction of a segment calculated on that basis. A variable, such as the current average segment velocity, is used as a means of determining the end of the interval by comparing it to the previous segment velocity, adjusting the end of the segment until there is an agreement, recalculating all the various variables concerned each time. This method is long and complicated and would increase the cost of the run. Also, the reference of Timoshenko, et. al. [18] already had a form of a solution that could readily be applied to this situation.

E. Results

A computer program was written using the three solutions, and the results will now be examined. This was run on the Amdahl 470V/8 at the University of Alberta [25], [26].

First consider the periods for each solution. It is important to consider this now as one is interested in the accuracy of the values obtained as compared with those from the approximation to the exact

TABLE 2.1 - UNDAMPED HARD SPRING EQUATION PERIODS
FOR a = 1.0, b = 0.15

Approx. to Exact Soln. = 5.958299670

No. of Divisions	PIECEWISE LINEAR CHORDS	PIECEWISE LINEAR TANGENTS
5	5.951629401	5.961481246
10	5.956522358	5.959160535
15	5.957490299	5.958695097
20	5.957838384	5.958526637
25	5.958002069	5.958447100
30	5.958091944	5.958403305
35	5.958146564	5.958376633
40	5.958182232	5.958359186
45	5.958206807	5.958347149
50	5.958224456	5.958338494
55	5.958237560	5.958332062
60	5.958247556	5.958327151
65	5.958255356	5.958323316
70	5.958261559	5.958320265
75	5.958266573	5.958317797
80	5.958270684	5.958315772
85	5.958274097	5.958314090
90	5.958276961	5.958312678
95	5.958279389	5.958311481
100	5.958281464	5.958310457

(which uses the routine described in [4]). The first 100 terms of (2.2.4) were calculated for this. Initially, a low degree of nonlinearity was examined, with the results on Table 2.1. One can see, as was originally stated in the introduction, that as the number of segments gets large, the solution by chords and by tangents slowly approach the same value. Further evidence of this can be seen in Table 2.2, where the nonlinearity is more pronounced.

In theory, then, one could keep decreasing the size of the increments, and the periods would slowly converge to that of the exact solution to as many significant figures as is needed. The only limitation to this accuracy would be the execution time for the program, as each set of calculations will add to the run cost.

It should be noted that the period for the approximation to the exact solution was obtained as explained earlier in this chapter, while for the piecewise methods, the final value of t_1 was multiplied by 4.

An evaluation of the methods will be helped by examining the solution curves. The program that produced the results for Tables 2.1 and 2.2 also drew a plot each time it looped through, using the number of increments that was in effect as a basis.

The plots can be seen in Figures 2.6 through 2.9. A few comments should be made concerning the results. A good agreement to the approximation to the exact solution [4], [15], [17] (for a = 1.0 and b = 0.15) is obtained by both methods for 5 increments (Figure 2.6) and with 10 (Figure 2.7). Here one can hardly detect any difference between the piecewise method results and that of the approximation to (2.2.2) [4], [15], [17]. As one might suspect for a = 1.0 and b = 2.0, Figure 2.8 shows that there is a discernable deviation from the solution being

TABLE 2.2 - UNDAMPED HARD SPRING EQUATION PERIODS FOR a = 1.0, b = 2.0

Approx. to Exact Soln. = 4.004308722

No. of Divisions	PIECEWISE LINEAR CHORDS	PIECEWISE LINEAR TANGENTS
5	3.976017631	4.005261005
10	3.996812433	4.007978008
15	4.000907203	4.005978416
20	4.002378678	4.005261005
25	4.003061288	4.004923996
30	4.003438030	4.004739061
35	4.003666618	4.004626708
40	4.003815695	4.004553357
45	4.003918296	4.004502829
50	4.003991920	4.004466543
55	4.004046539	4.004439606
60	4.004088177	4.004419059
65	4.004120645	4.004403028
70	4.004146464	4.004390280
75	4.004167307	4.004379975
80	4.004184398	4.004371526
85	4.004198581	4.004364513
90	4.004210479	4.004358627
95	4.004220560	4.004353639
100	4.004229175	4.004349375

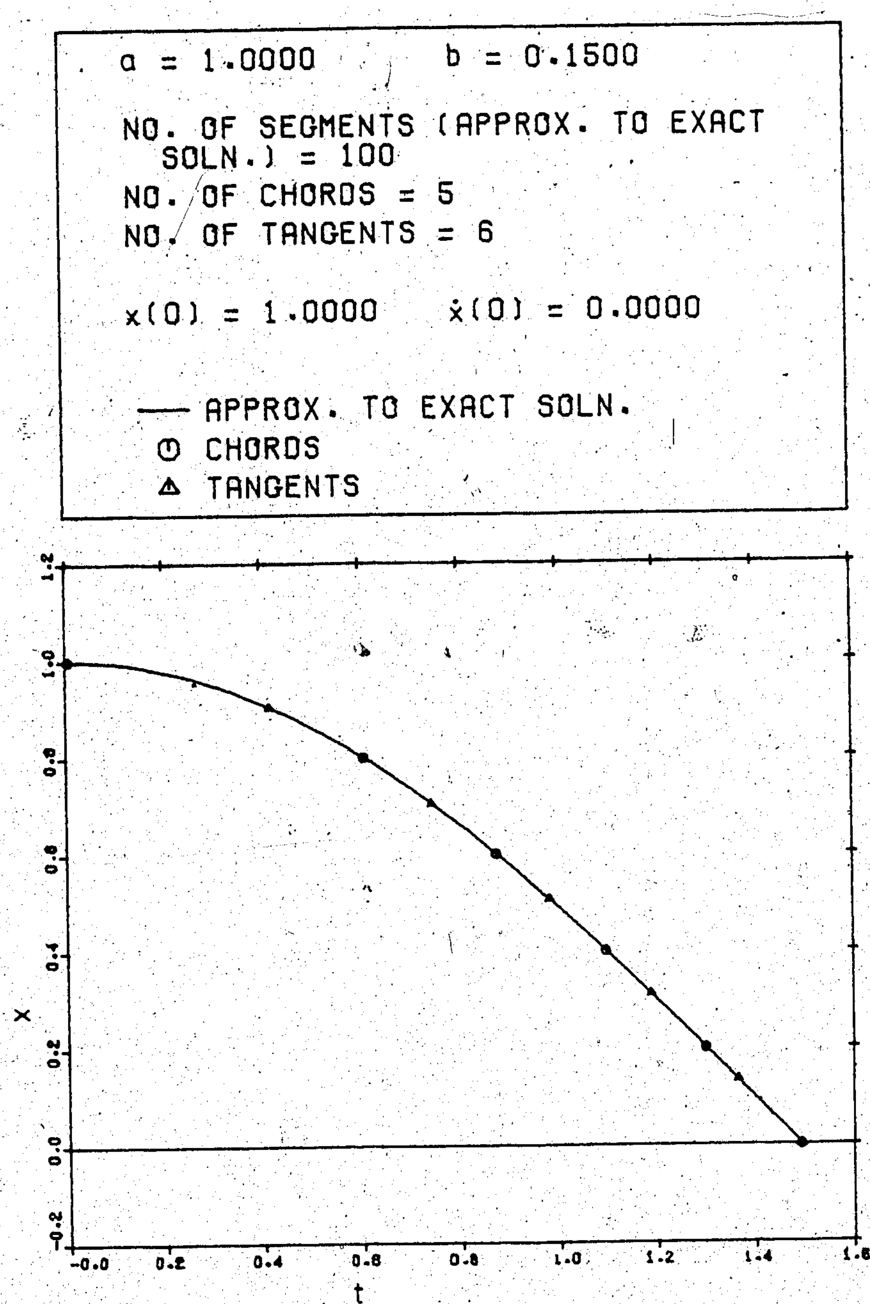


FIGURE 2.6 - SOLUTIONS TO HARD SPRING EQUATION (UNDAMPED) FOR $a = 1.0$, $b = 0.15$, $\Delta t = \tau_{\pi/4}/100.0$, $\Delta x = 0.20$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Approximation to exact solution uses [4], [15], [17].)

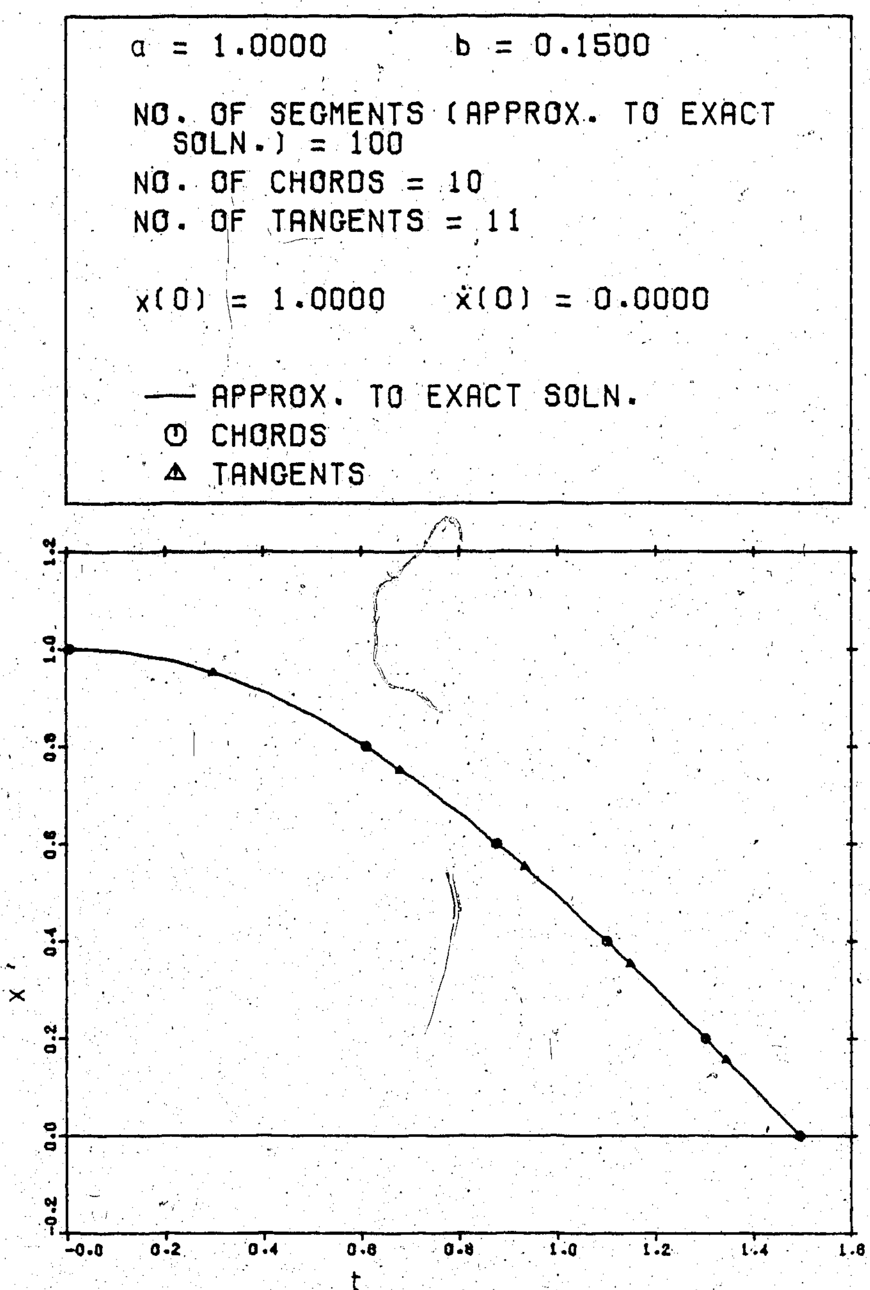


FIGURE 2.7 – SOLUTIONS TO HARD SPRING EQUATION (UNDAMPED) FOR a = 1.0, b = 0.15, $\Delta t = \tau_{\pi/4}/100.0$, $\Delta x = 0.10$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Approximation to exact solution uses [4], [15], [17].)

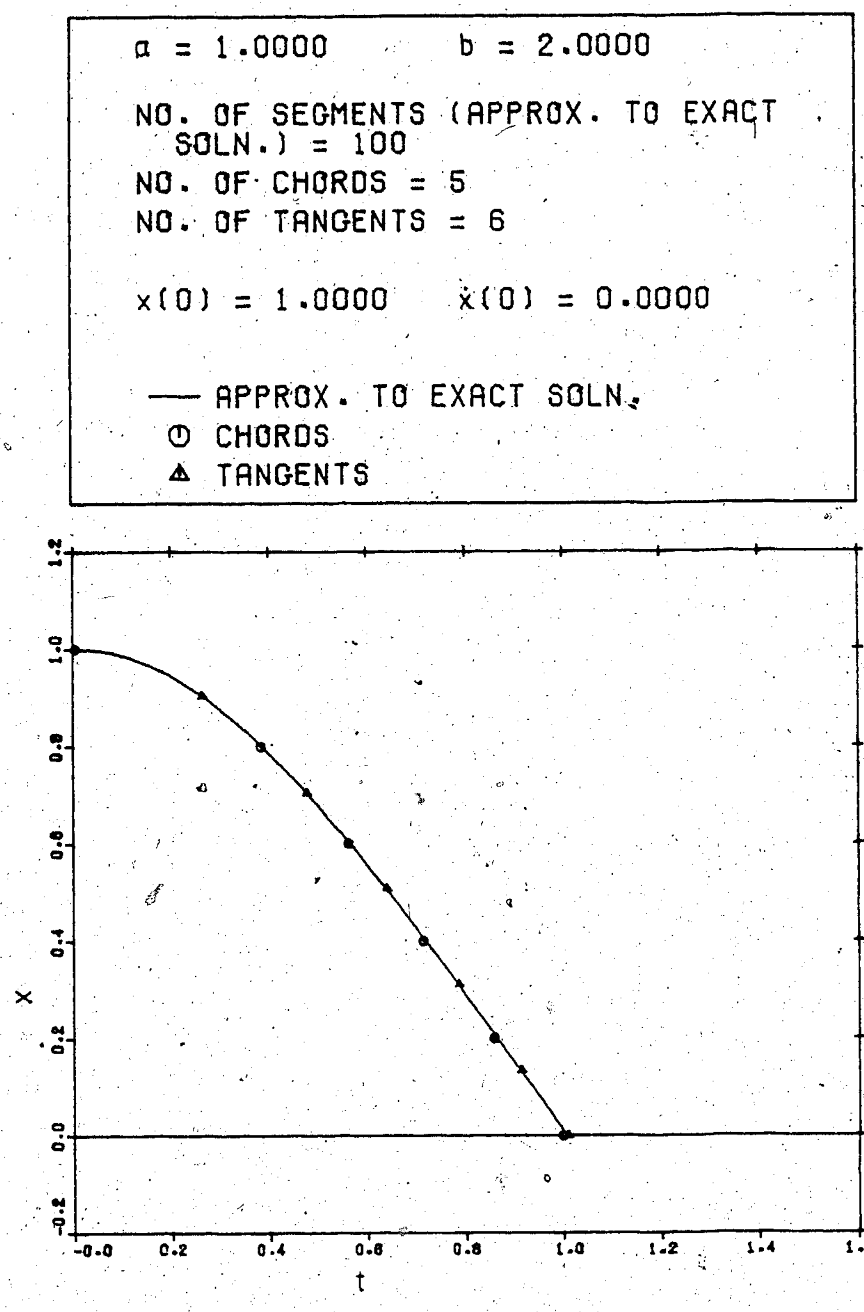


FIGURE 2.8 - SOLUTIONS TO HARD SPRING EQUATION (UNDAMPED) FOR a = 1.0, b = 2.0, $\Delta t = \tau_{\pi/4}/100.0$, $\Delta x = 0.20$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Approximation to exact solution uses [4], [15], [17].)

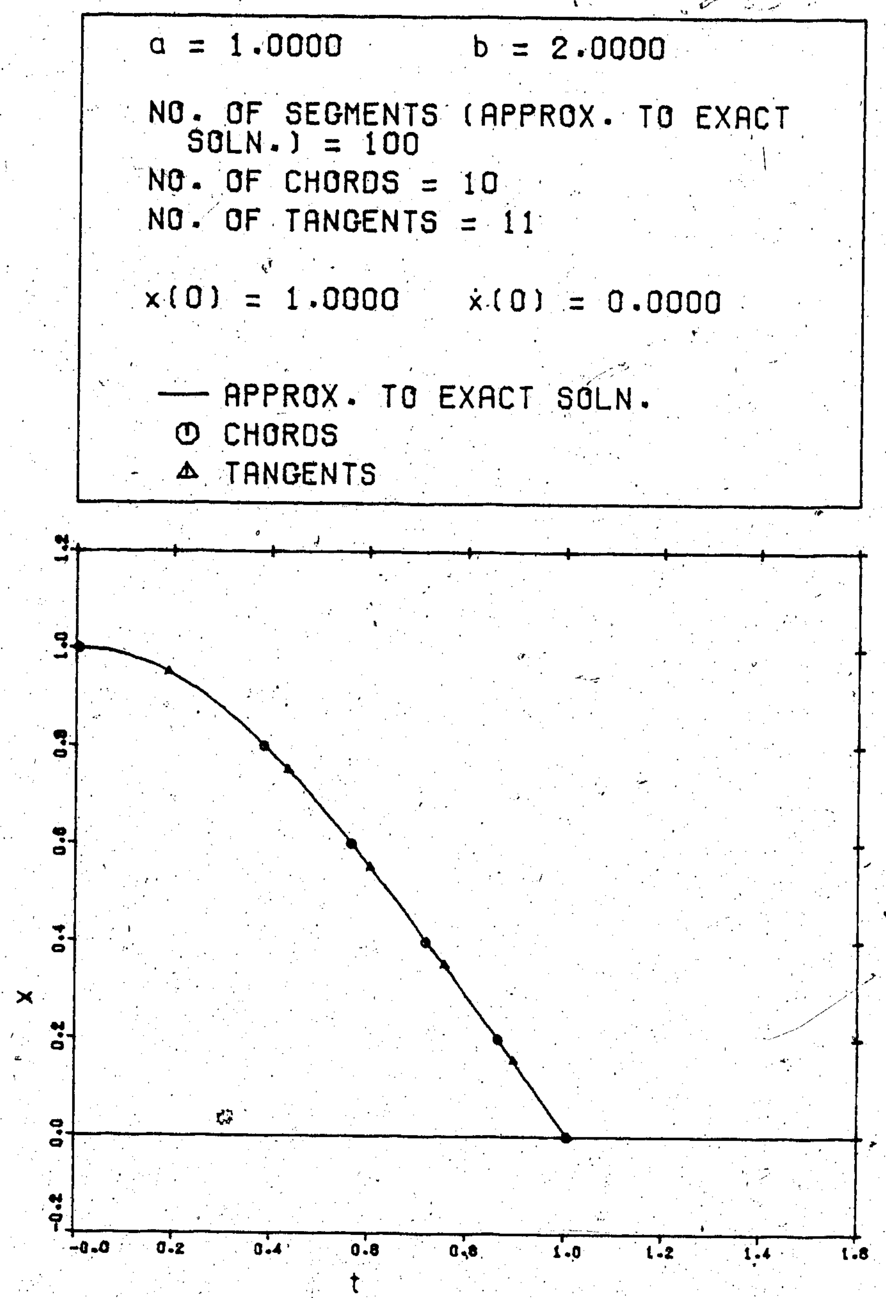


FIGURE 2.9 - SOLUTIONS TO HARD SPRING EQUATION (UNDAMPED) FOR a = 1.0, b = 2.0, $\Delta t = \tau_{\pi/4}/100.0$; $\Delta x = 0.10$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Approximation to exact solution uses [4], [15], [17].)

compared, since the degree of nonlinearity is considerably higher than in the first case. This does not appear in Figure 2.9, where 10 increments were used.

A question that should be raised is concerning the actual execution times involved, since, when choosing a method of solving these sorts of problems, one must strike a balance between required accuracy to the solution being compared and the cost of the run. This was done by taking the original program and modifying it by stripping it of all non-essential operations with only the actual calculations remaining, and then utilizing a timing subroutine available through the library of the computer system, and calling it for each solution [27]. The results are seen on Figures 2.10 and 2.11. It should be noted here that these results were obtained with an older and somewhat slower version of the the program, but they should serve to illustrate the relative execution times. It should also be noted that these results were obtained with an Amdahl 470V/7 [25]. This difference in the computers was due to an upgrading of the original 470V/7 model to a 470V/8 done while the research for this thesis was being carried out [26]. The author reran the timing program and found that the results were about 10 percent faster with the V/8 than the V/7. Also, with some further minor modifications to the program itself, the author estimates about another 10 percent can be taken off the original times.

By inspection, one can see that the actual distance travelled by the system using the method of chords would be somewhat less than that using tangents. Already this implies that the period would be greater using the latter method than the former when comparing it to that obtained by using the results based on the exact solution. That would

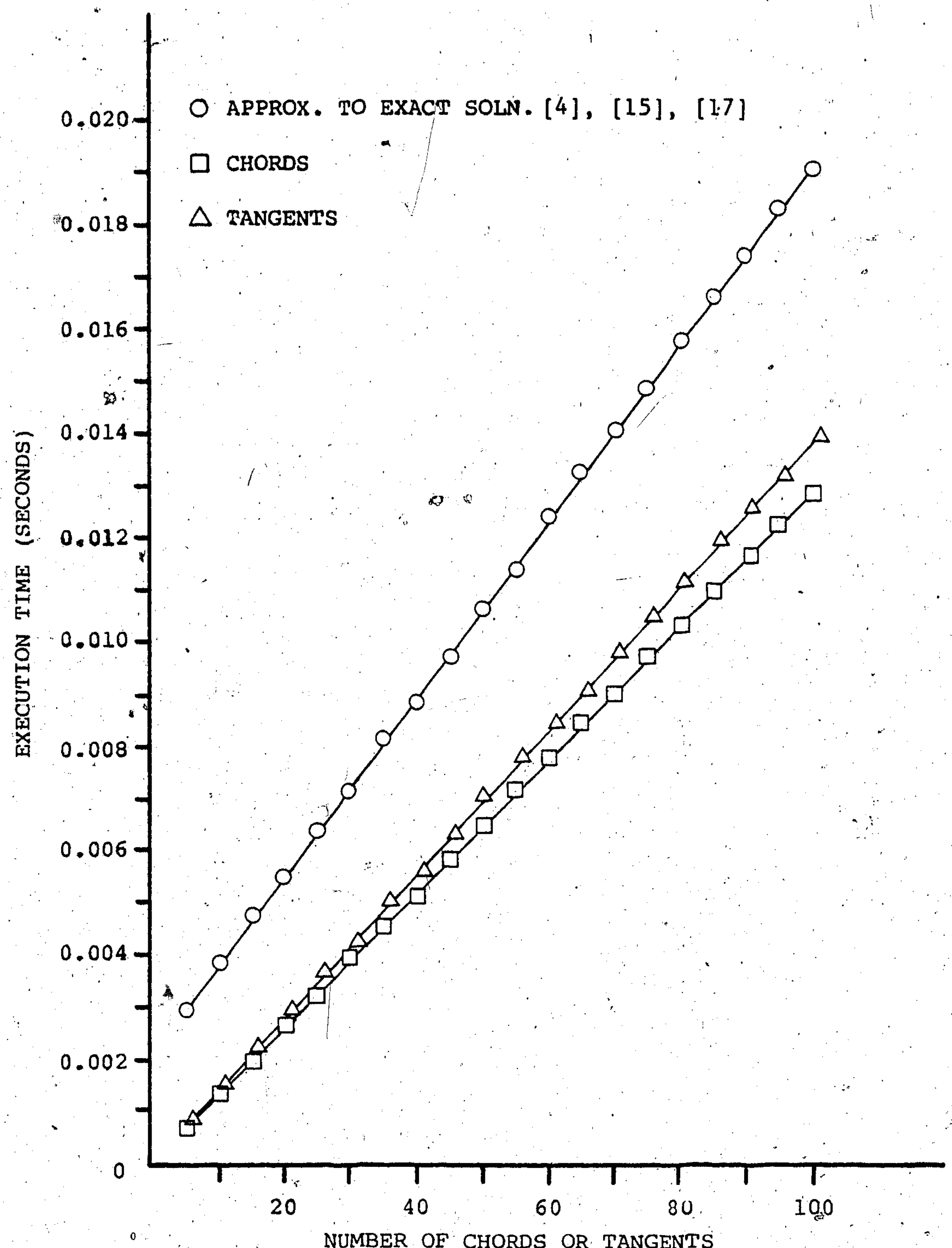


FIGURE 2.10 - EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (UNDAMPED) FOR $a = 1.0$, $b = 0.15$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$ [27]

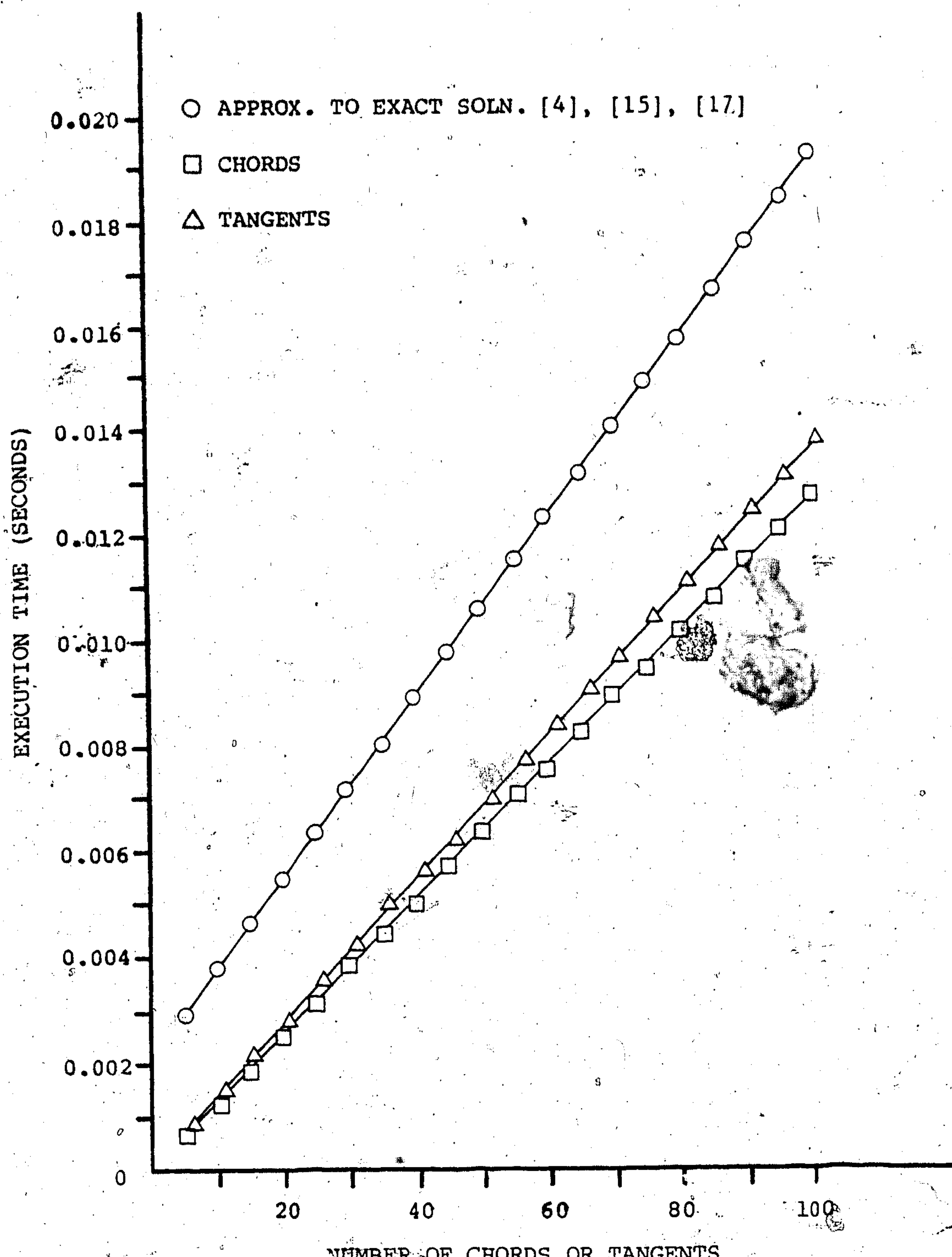


FIGURE 2.11 - EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (UNDAMPED) FOR a = 1.0, b = 2.0, $x(0) = 1.0$, $\dot{x}(0) = 0.0$ [27]

explain why, when b was 2.0, for example, one obtained a difference in the displacement on the higher side using tangents than chords.

Another question is why the timing results for the approximation to the exact solution [4], [15], [17] should be considerably higher than that for the piecewise methods. A word of explanation concerning the program will clarify this point. The execution times were obtained by calculating the first 50 terms of (2.2.4)[15] and then generating the solution results, prior to increasing the number of segments and looping through the calculations once again. This was done to determine the total time required to obtain a solution for a given Δx, thus allowing comparison with the other methods. This required that the subroutine that calculated (2.2.4)[15] had to be called each time.

Since more calculations are necessary for each segment for the method of tangents, it would be expected that the execution times would be greater for this solution than for the method of chords.

In any case, the results for the piecewise methods are impressive when one considers that for this problem, it would appear that an accurate solution can be easily obtained by using either method, with a relatively small number of segments, and that it is faster than the solution that was chosen to be the standard one [4], [15], [17]. One can conclude that for this problem, the piecewise method would be the recommended one to use, particularly the method by chords.

Because the method of chords was easier to manipulate and converged faster to the approximation to (2.2.2)[4], [15], [17], it was chosen to be the method of piecewise linearization for the remainder of the thesis and will subsequently be referred to as such.

III. HARD SPRING EQUATION (LINEAR DAMPING)

A. Preliminary Comments

Most physical systems involve dissipation, and an example of this is the system described in the previous chapter, but with the inclusion of a simple form of damping in which it is linearly related to the velocity:

$$\ddot{x} + \frac{c}{M} + ax + bx^3 = 0 \ , \qquad (3.1.1)$$

where:

c = Damping constant,

M = System mass,

a,b = Restoring force factors, as used in Chapter II.

The system is shown in Figure 3.1 [28].

Since there is no exact solution and approximate methods such as those mentioned in the first chapter are beyond the scope of this thesis, a Runge-Kutta solution [5] has been chosen to be the standard by which to judge the accuracy of the piecewise method.

For convenience, it was decided to restrict the cases examined to underdamped conditions.

The values for a and b are the same as in the previous chapter, while $\frac{c}{M}$ will have the values of 0.0, 0.1, and 0.25 which should give a good representation of an underdamped case.

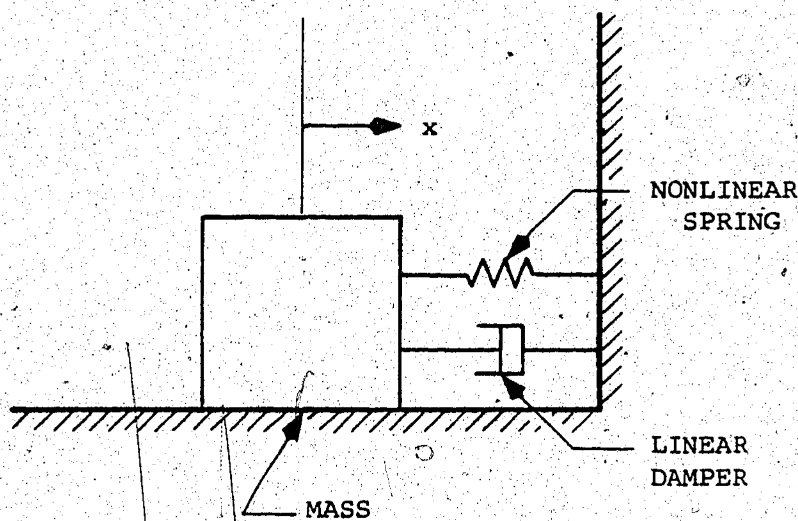


FIGURE 3.1 - MASS-SPRING SYSTEM (LINEAR DAMPING)[28]

B. Derivation of Piecewise Linear Solution

Since the piecewise linearization method by chords had been chosen to be the means of solving the equations for the remainder of the thesis, and much of the background derivation is exactly the same as before, only the results of this plus the derivation of new material will be shown in this and subsequent chapters.

Recalling the development of the method by chords and using the same initial conditions and notation as in the previous chapter, the following equation results for a linear segment [19], [29]:

$$\frac{d^2}{dt^2}(x - x_0^*) + \frac{c}{M}\dot{x} + \frac{k_0}{M}(x - x_0^*) = 0,$$

k_0 = Slope of chord line, as in Chapter II.

where x, x_0, and $\dot{x}$ are the same as before and are found in the manner previously described in Chapter II, Section C. The solution to this system is now the familiar [29]:

$$x = x_0^* + e^{-n(t - t_0)}\Big[A \cos p^*(t - t_0) + B \sin p^*(t - t_0)\Big] , \qquad (3.2.1)$$

where:

$$p^* = \sqrt{\frac{k_0}{M} - n^2} ,$$

t_0 = Time taken for system to reach current x_0,

$$n = \frac{c}{2M},$$

$$B = \frac{\dot{x}_0 + n(x_0 - x_0^*)}{p^*} .$$

Differentiating (3.2.1) yields the velocity [29]:

$$\dot{x} = e^{-n(t - t_0)} \Big[(p^*B - nA) \cos p^*(t - t_0) + (-nB - p^*A) \sin p^*(t - t_0) \Big] , \qquad (3.2.2)$$

with the various coefficients as before.

Solving for $t_1 - t_0$ requires the use of a Newton-Raphson method, since there does not appear to be a simpler way of finding this quantity such as that used in the previous chapter. This would be [30]:

$$(t_1 - t_0)_{i+1} = (t_1 - t_0)_i - \frac{x[(t_1 - t_0)_i]}{\dot{x}[(t_1 - t_0)_i]} .$$

However, as the displacement approaches the first trough, the velocity approaches zero, and consequently the method as defined would break down. But, since this can be set up in a computer program, a check can be run such that when the denominator becomes very small (corresponding to a small speed) the Newton-Raphson relationship can be redefined as [30]:

$$(t_1 - t_0)_{i+1} = (t_1 - t_0)_i - \frac{\dot{x}[(t_1 - t_0)_i]}{\ddot{x}[(t_1 - t_0)_i]} ,$$

$$\ddot{x}\left[(t_1 - t_0)_i\right] = e^{-n\left[(t_1 - t_0)_i\right]}$$

$$\cdot\{\left[(-p^{*2} + n^2)A - 2np^*B\right]\cos p^*\left[(t_1 - t_0)_i\right]$$

$$+ \left[2np^*A + (-p^{*2} + n^2)B\right]\sin p^*\left[(t_1 - t_0)_i\right]\} ,$$

with the coefficients the same as used for (3.2.1) and (3.2.2).

Also, there is the likelihood that the first trough to be encountered bottoms out in the middle of a segment, which poses another problem as to how to determine exactly where this occurs. First $t_1 - t_0$ is found as before (with the second Newton-Raphson relationship being in effect as well as $\dot{x}$ will approach zero). The appropriate value for the displacement is found, and this is compared to what the particular x_1 would have been. If the difference between the two values is small, then the iterations stops. If not, this x becomes the new x_1, and new end conditions are found with the iteration continuing using $t_1 - t_0$.

The half-period, then, would be the final value of t_1.

C. Runge-Kutta Solution

The Runge-Kutta numerical method used throughout this thesis is the routine DVERK from IMSL [5]. Since it was in a system library, it was easily accessed by the program written to solve this situation. The basis of the method is an approach using approximations described in this reference.

D. Results

A computer program was written that included the piecewise linear solution, complete with the necessary subroutines for calculating the

FIGURE 3.2 - SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $b = 0.15$, $\frac{C}{M} = 0.0$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)

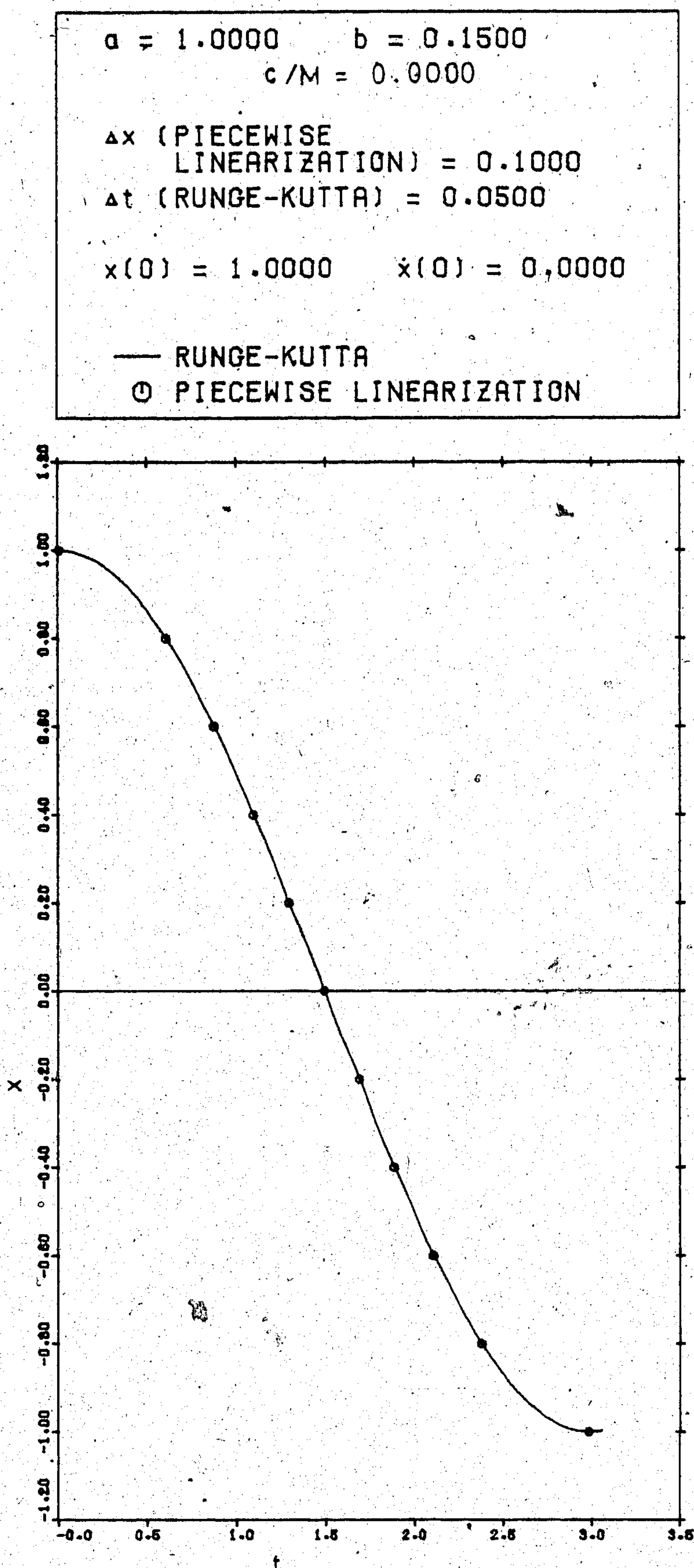

a = 1.0000 b = 0.1500
c/M = 0.0000
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
x
t
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

FIGURE 3.3 - SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $b = 2.0$, $\frac{c}{M} = 0.0$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)

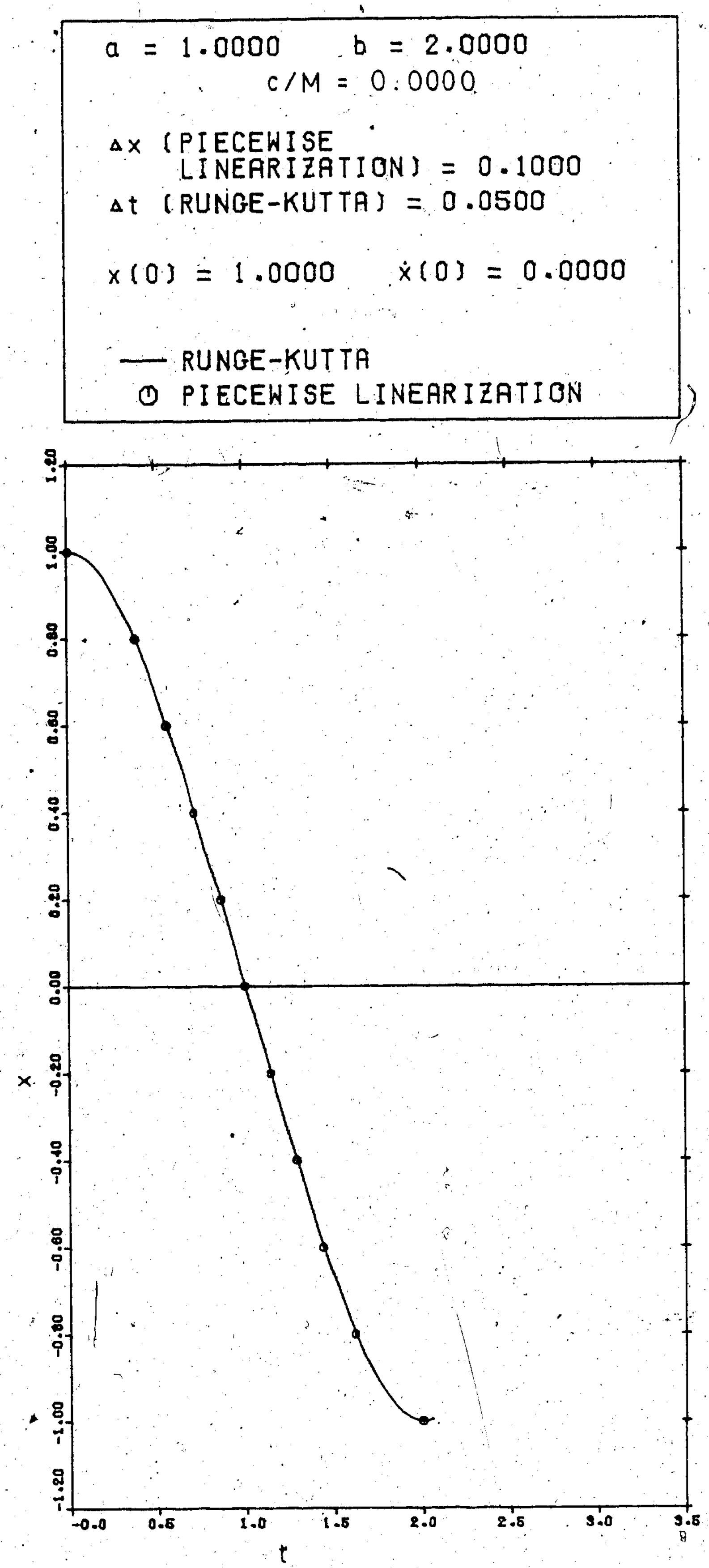

a = 1.0000 b = 2.0000
c/M = 0.0000
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
X
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
t

FIGURE 3.4 - SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $b = 0.15$, $\frac{C}{M} = 0.10$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)

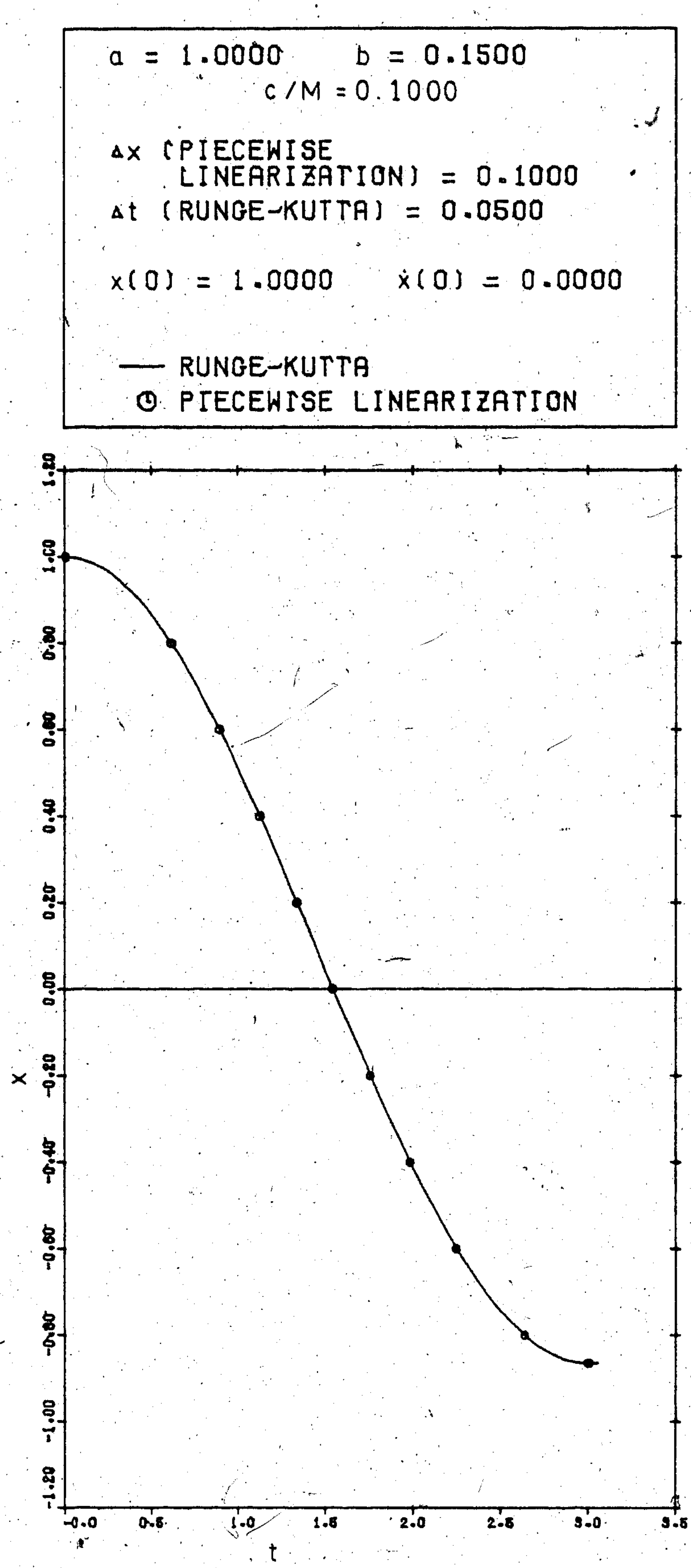
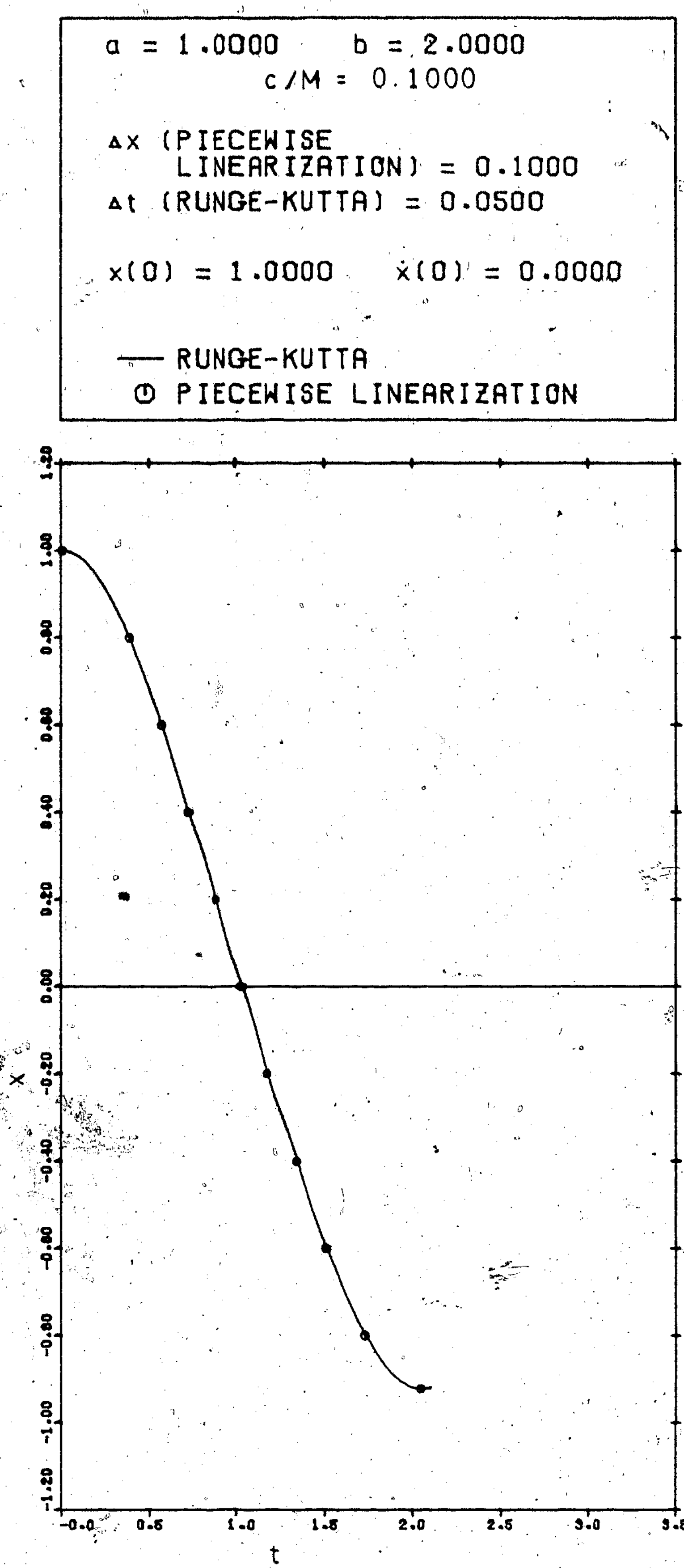

a = 1.0000 b = 0.1500
c/M = 0.1000
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
X
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
t

FIGURE 3.5 - SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $b = 2.0$, $\frac{C}{M} = 0.10$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)



a = 1.0000 b = 2.0000
c/M = 0.1000
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
x
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
t

FIGURE 3.6 - SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $b = 0.15$, $\frac{C}{M} = 0.25$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)

a = 1.0000 b = 0.1500
c/M = 0.2500

Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500

x(0) = 1.0000 ẋ(0) = 0.0000

—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION

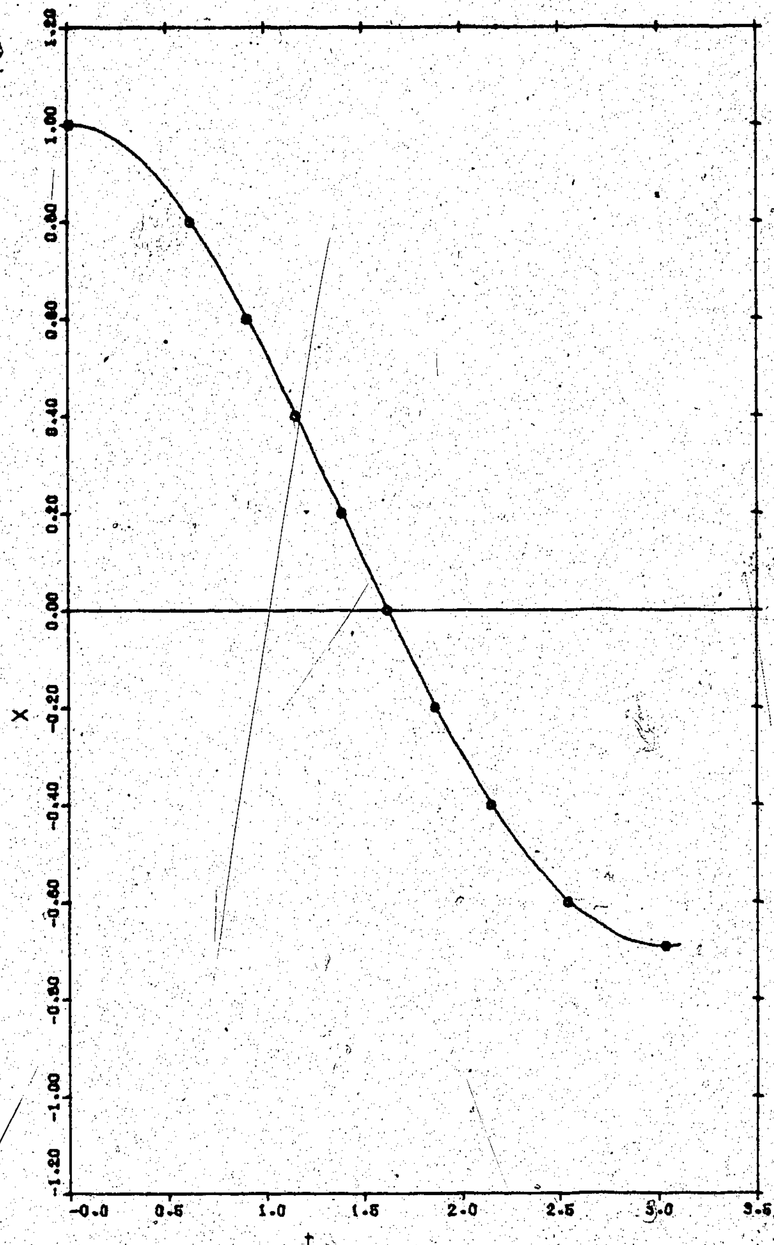
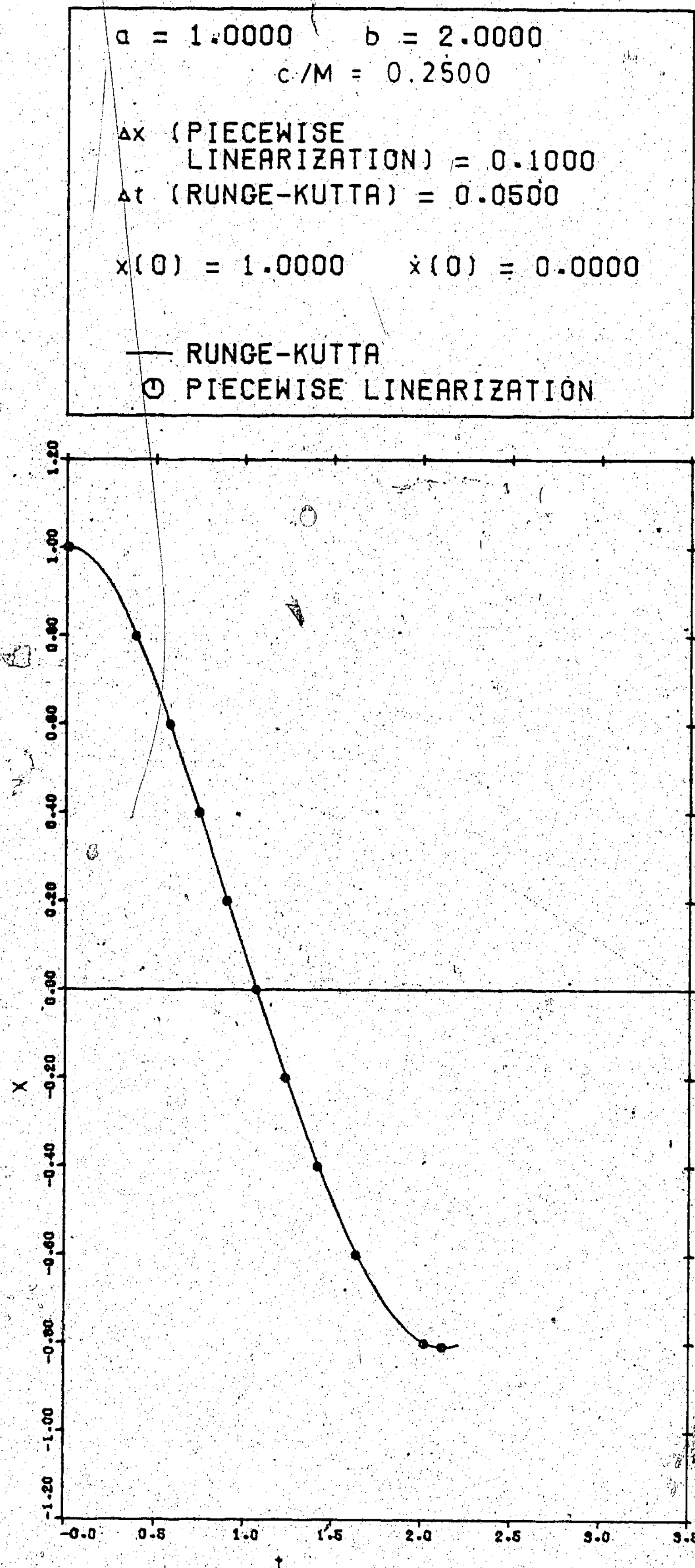


FIGURE 3.7 - SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $b = 2.0$, $\frac{C}{M} = 0.25$, $\Delta x = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)


a = 1.0000 b = 2.0000
c/M = 0.2500
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
x
t

various parameters like x_0^*, as well as the input parameters and call to DVERK [5]. The results were obtained by running the program on the Amdahl 480V/8 [26] described earlier.

As a first trial, to see if the program functioned, it was decided to check if the solution would work for the simple case of $\frac{C}{M}$ being zero for the sets of values for a and b as described earlier in the chapter, and $\bar{X}_0$ being taken to be 1.0. The results of this are seen in Figures 3.2 and 3.3, using a value of Δx of 0.1, and a time increment for the Runge-Kutta solution [5] of 0.05, the sizes being chosen arbitrarily, but still sufficiently large to allow any problems concerning stability arise.

From Figures 3.2 and 3.3, it is obvious that the program works for the undamped case, as the correspondence between the two solutions is good, and there appears to be little problem in the piecewise linear method bottoming out at the first trough.

Now for solving the equation using non-zero values for $\frac{C}{M}$. The results for this can be seen in Figures 3.4 - 3.7. For each plot there seems to be a good correspondence between the results obtained for the piecewise linear solution and the Runge-Kutta procedure [5], which implies two things. One is that the method of solution using the piecewise linearization by chords gives a good agreement with the Runge-Kutta approximation [5] of this equation. The other is that for even a somewhat large increment of displacement, the piecewise linear method gives a good result, and that it is not necessary to go to a smaller value to get an accurate solution. This has an advantage as far as the economics of the runs are concerned.

As before, the program was stripped down to the essential

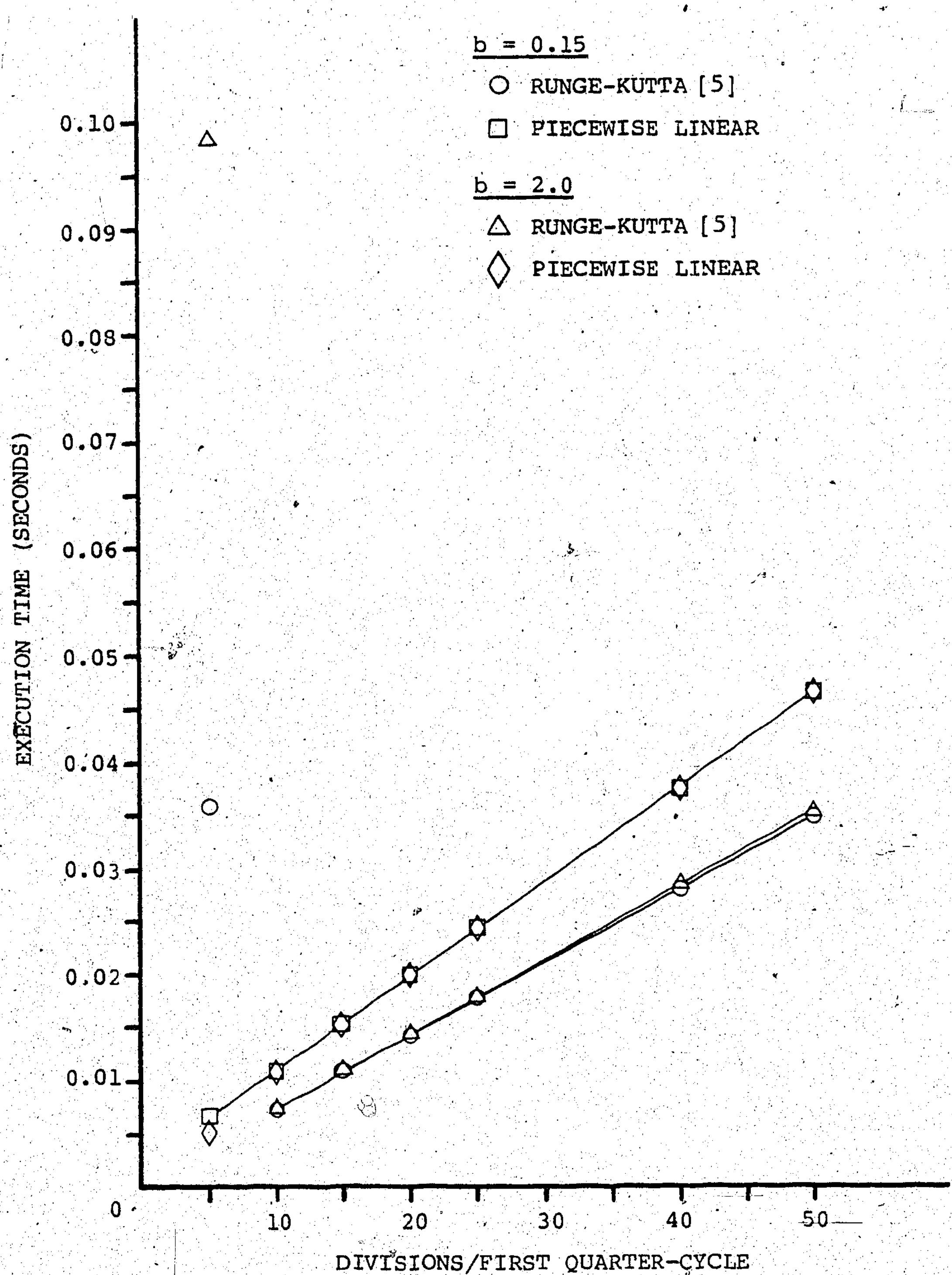


FIGURE 3.8 – EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $\frac{C}{M} = 0.0$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$ [27]

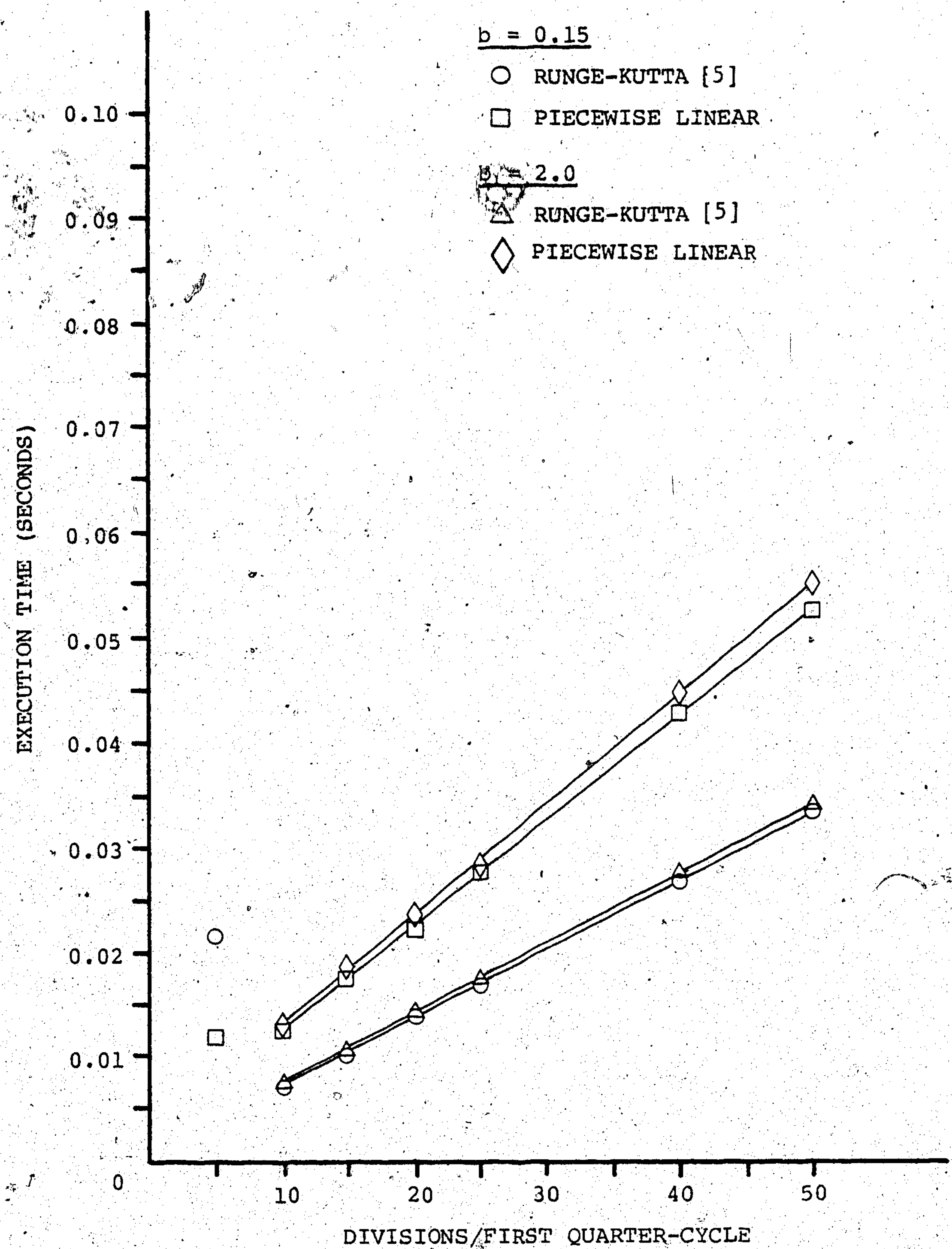


FIGURE 3.9 - EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0$, $\frac{C}{M} = 0.10$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$ [27]

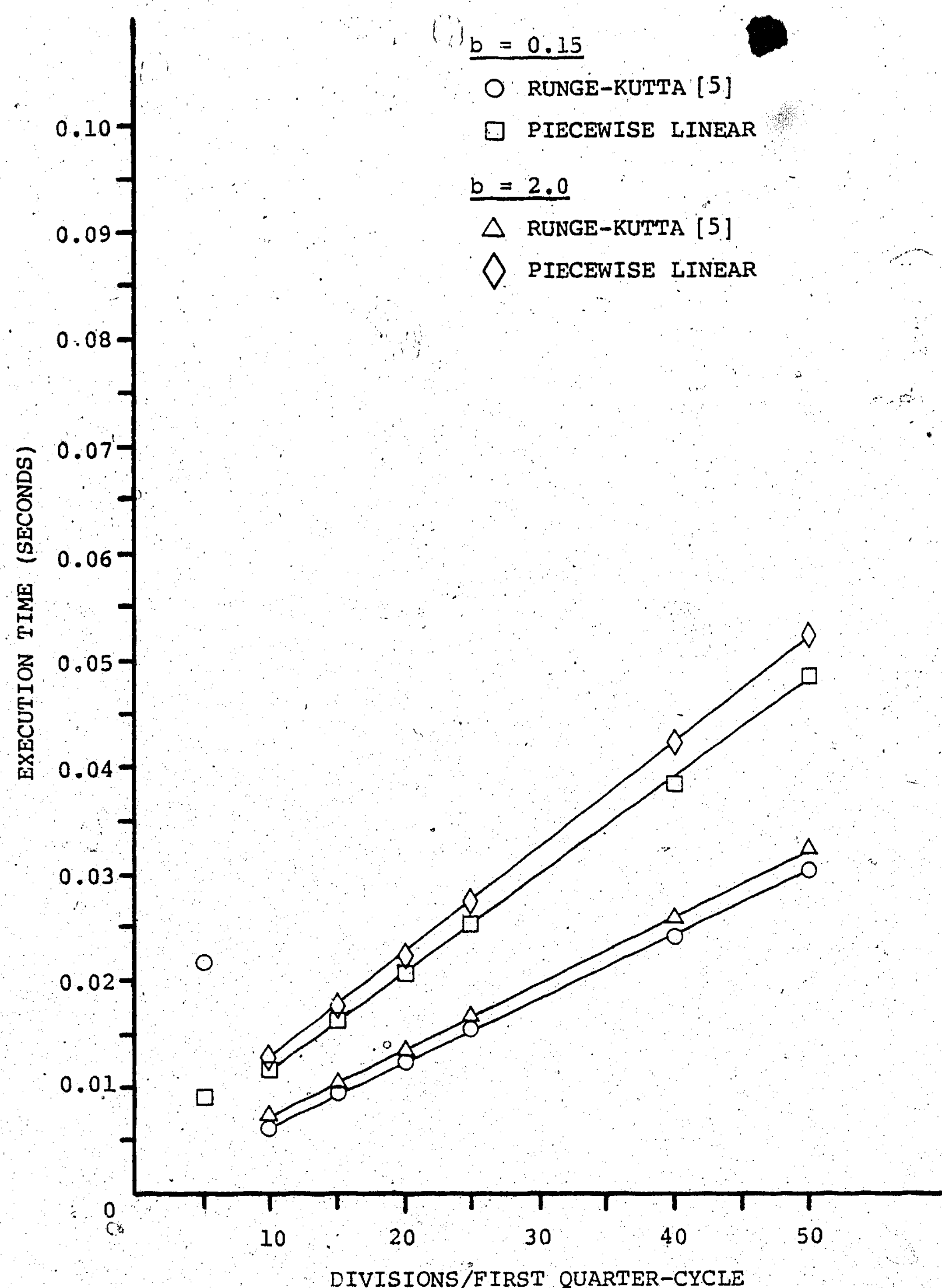


FIGURE 3.10 - EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (LINEAR DAMPING) FOR $a = 1.0, \frac{C}{M} = 0.25, x(0) = 1.0, \dot{x}(0) = 0.0$ [27]

calculations and the execution times for each method of solutions obtained [27]. Again, it should be noted that these values were acquired with an older version of the plot program that was slower and that this was done on the V/7 version of the Amdahl 470 [25], [26]. The results can be seen in Figures 3.8 - 3.10.

Before continuing, an explanation is necessary concerning the timing results. As mentioned earlier, the piecewise method used an interval of displacement while the Runge-Kutta method [5] made use of time. For the purposes of determining correspondence between the two solutions, this would be sufficient, but this would not be valid for comparing the execution times, since Δx and Δt would scarcely provide any means for judgement between the two methods. However, the number of divisions (or overall iterations) for the first quarter-cycle would possibly be a benchmark. To an extent, this was done for the piecewise method, since the system initially had a known displacement, and Δx was chosen from that.

For the Runge-Kutta method [5], a similar approach can be used based upon the first quarter-period as calculated by the piecewise method. This timespan can be divided into the same number of intervals as was used by the first solution, and the execution time is obtained with the new value for Δt.

The timing program kept track of the number of points calculated by the piecewise solution (including the first trough, which generally falls inside an interval) based on Δx while the Runge-Kutta routine [5] calculates this same number of points, but using the new Δt as was just described. This should give a reasonable correspondence to the piecewise solution except that the Runge-Kutta method [5] will likely go past the first trough as before.

One might expect a linear relationship between the execution time and the number of divisions of the first-quarter cycle. Since the Runge-Kutta solution [5] is from a packaged program, presumably it should be as efficient as possible, which would be a reasonable explanation as to why the execution times for it are lower as compared to the piecewise linear method. However, what is of note is what happens at 5 divisions.

For some of the runs, no values were obtained, as the execution time limit that had been set prior to the run had been exceeded. Attempts to restart the program by adding additional seconds to bring the total execution time limit to at least 15 seconds failed to give results [31]. Some clue may be available from those runs that used what may appear to be excessive run time values.

For runs such as for a = 1.0, b = 0.15, and $\frac{C}{M} = 0.25$, the points are far off the line. This may indicate that there could be a lower limit for which either solution would work (or be economical), as from what was obtained, one can see that the Runge-Kutta [5] took longer to finish at this point than the piecewise linear method, which was not expected.

IV. HARD SPRING EQUATION (NONLINEAR DAMPING)

A. Preliminary Comments

This situation is essentially the same as that examined in the previous chapter, with the exception that the damping is no longer linear, but is related to the square of the velocity. The solution for the linear segments is the same, but the means by which it is obtained are slightly different. The equation considered is:

$$\ddot{x} + \frac{q}{M}|\dot{x}|^2 \operatorname{sgn}(\dot{x}) + ax + bx^3 = 0 , \qquad (4.1.1)$$

where:

q = Nonlinear damping constant,

M,a,b = as used in Chapter III.

This equation describes the system shown in Figure 4.1 [28].

As in the case of equation (3.1.1), no exact solution seems to exist, and so the same Runge-Kutta solution [5] was used as a basis for comparison. The same values for a and b were used, while the values for $\frac{q}{M}$ were 0.0, 0.1, and 0.25, restricting the investigation to the under-damped case. The same initial conditions were used, with X_0 equal to 1.0.

B. Derivation of Piecewise Linear Solution

The derivation for the equation of the linear segments is exactly the same as was described in the previous chapters, using essentially the same method to find the necessary factors such as x_0^* and p_0^*. A slight modification in the procedure to determine the damping constant $\frac{c}{M}$

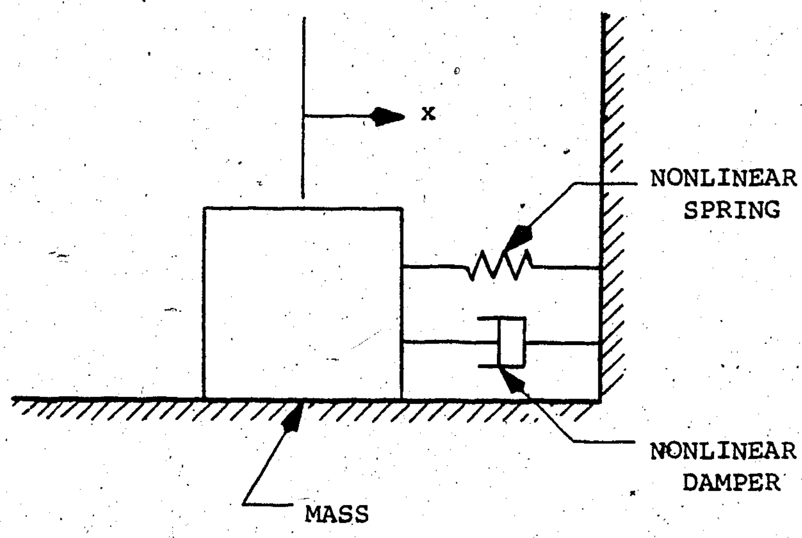


FIGURE 4.1 - MASS-SPRING SYSTEM (NONLINEAR DAMPING)[28]

is required as it changes in value from one segment to another. Since it is desired to obtain a solution whose form is similar to that of (1.2), the expression for the damping as given in (4.1.1) is replaced by an equivalent one resembling what was seen in (3.1.1), or, in other words, a linear approximation over a small increment of displacement

The starting point for this is part of the problem. One possibility involves an iteration within an iteration in order to obtain an average value of the velocity, and thus the damping force. This is done as follows.

At the beginning of motion, no damping is assumed, and the form of the equation is taken to be [19]:

$$\frac{d^2}{dt^2}(x - x_0^*) + \frac{k_0}{M}(x - x_0^*) = 0 .$$

From this, an end velocity can be found. This is averaged, and a damping coefficient is determined using:

$$c = \frac{q|\dot{x}_{AV}|^2 \operatorname{sgn}(\dot{x}_{AV})}{\dot{x}_{AV}} ,$$

$\dot{x}_{AV}$ = Average segment velocity.

The iteration is done again, with the same initial conditions, but the equation now becomes [19], [29]:

$$\frac{d^2}{dt^2}(x - x_0) + \frac{c}{M}\dot{x} + \frac{k_0}{M}(x - x_0^*) = 0 .$$

Now the current velocity is calculated and compared with the previous final value. If their difference is greater than a given tolerance, the iteration starts again, with a new damping coefficient calculated from the average segment velocity, based upon this new final speed. Once this tolerance is achieved, the end conditions become the initial conditions for the next segment, and the entire procedure starts again as before, except now for all subsequent segments, the new initial velocity is used each time.

Since the derivation is the same as before, the solution will essentially be identical to that of the previous chapter [19], [29] or:

$$x = x_0^* + e^{-n(t - t_0)} \left[A \cos p^*(t - t_0) + B \sin p^*(t - t_0) \right] , \tag{4.2.1}$$

$$\dot{x} = e^{-n(t - t_0)} \left[(-nA + p^*B) \cos p^*(t - t_0) - (p^*A + nB) \sin p^*(t - t_0) \right] , \tag{4.2.2}$$

where the coefficients were determined as before, and are:

$$A = x_0 - x_0^* ,$$

$$B = \frac{\dot{x}_0 + n(x_0 - x_0^*)}{p^*} ,$$

$$n = \frac{c}{2M} .$$

The values for $t_1 - t_0$ were calculated using the Newton-Raphson method [30], and are subject to the conditions described in the previous chapter, including the procedure for finding the point at which the system bottoms out in the first trough.

C. Results

The equations for the piecewise linear solution and the required commands to use DVERK [5] (including parameters and the calling statement) were assembled into a computer program, and solution plots generated by running them on the Amdahl 470V/8 [26] mentioned earlier. The size of the displacement increment used by the piecewise linear method was the same as the one used in the previous chapter ($\Delta x = 0.1$), as well as the same size of time increment for the Runge-Kutta method [5] ($\Delta t = 0.05$).

The results of the computer plot program that calculated both solutions can be seen in Figures 4.2 - 4.7. Once again, one can see that a good agreement was achieved, even when using a fairly large value for Δx, and it would seem from the output that the piecewise linear method successfully calculated the location of the first trough. It can be concluded from this that the parameter values and the size of the displacement interval were sufficient to give satisfactory results for the given conditions.

The program was stripped to the essential steps that contained only the calculations and the execution times were determined for it [27], which should give one a reasonable estimate as to the relative speeds (and hence, the costs) of the two methods. As in the previous chapter, the basis of judgement was the number of divisions of the first quarter-cycle. This was because the piecewise linear solution used an increment of displacement, while the Runge-Kutta method [5] required increments of time.

The values obtained are to be found on Figures 4.8 - 4.10, and were taken from tests conducted on the Amdahl 470V/7 [25], using an older and

FIGURE 4.2 - SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR $a = 1.0$, $b = 0.15$, $\frac{g}{M} = 0.0$, $\Delta x(0) = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta method uses [5].)

a = 1.0000 b = 0.1500
q/M = 0.0000

Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500

x(0) = 1.0000 ẋ(0) = 0.0000

—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION

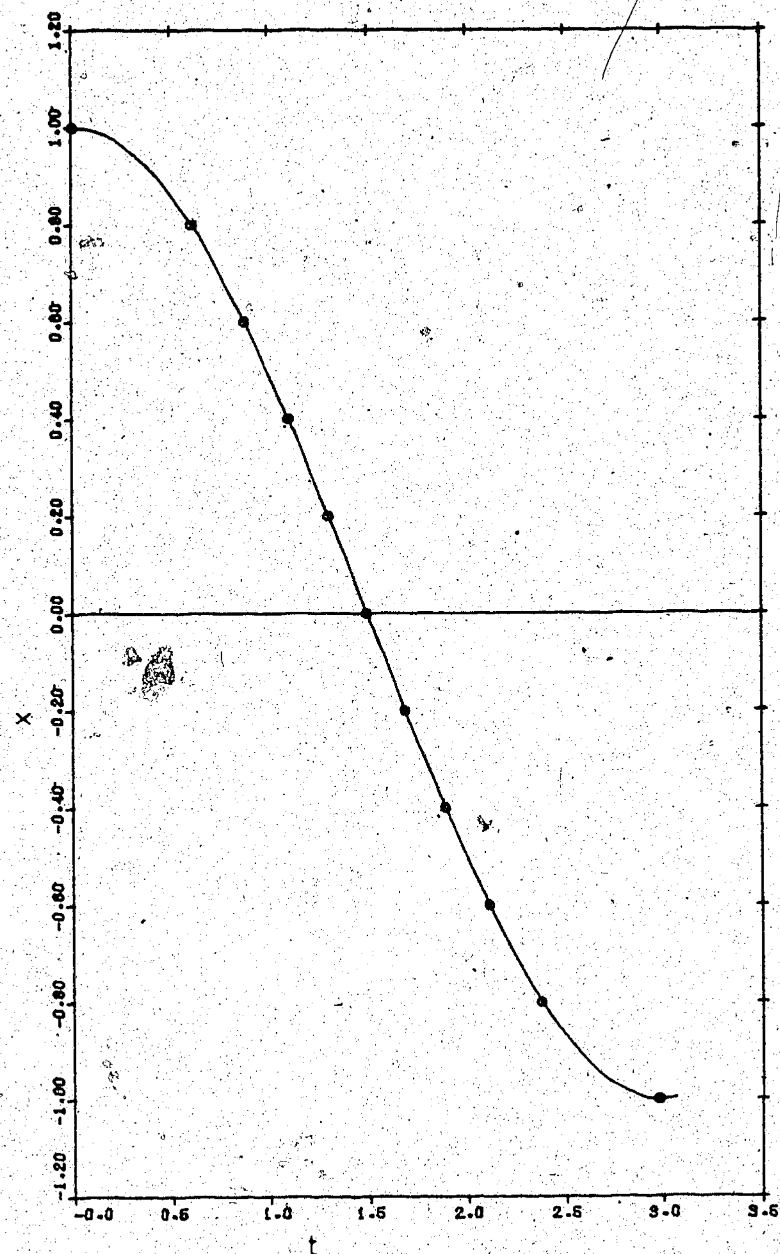


FIGURE 4.3 - SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR a = 1.0, b = 2.0, $\frac{q}{M}$ = 0.0, $\Delta x(0)$ = 0.10, Δt = 0.05, $x(0)$ = 1.0, $\dot{x}(0)$ = 0.0

(N.B.: Runge-Kutta method uses [5].)

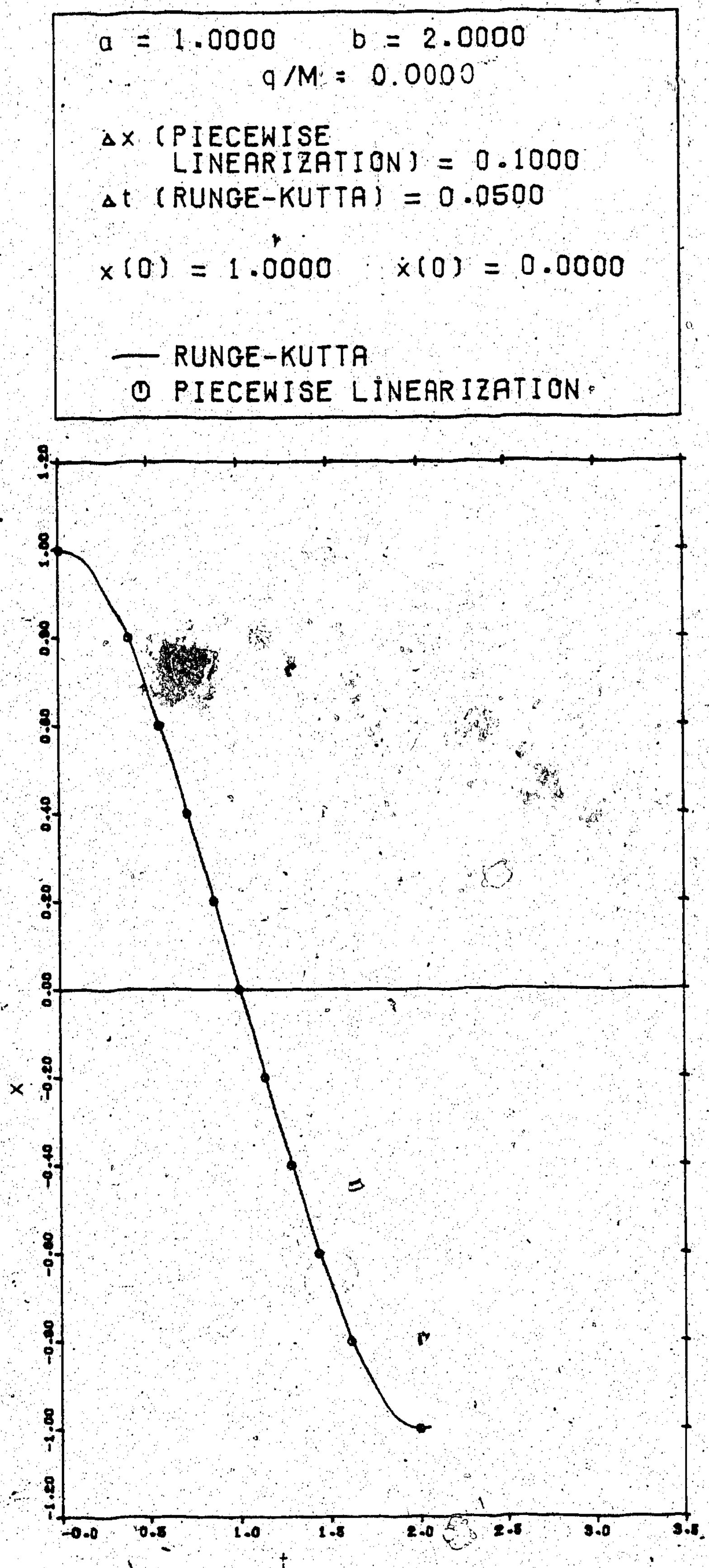

a = 1.0000 b = 2.0000
q/M = 0.0000
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
x
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
t

FIGURE 4.4 - SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR $a = 1.0$, $b = 0.15$, $\frac{g}{M} = 0.10$, $\Delta x(0) = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta method uses [5].)

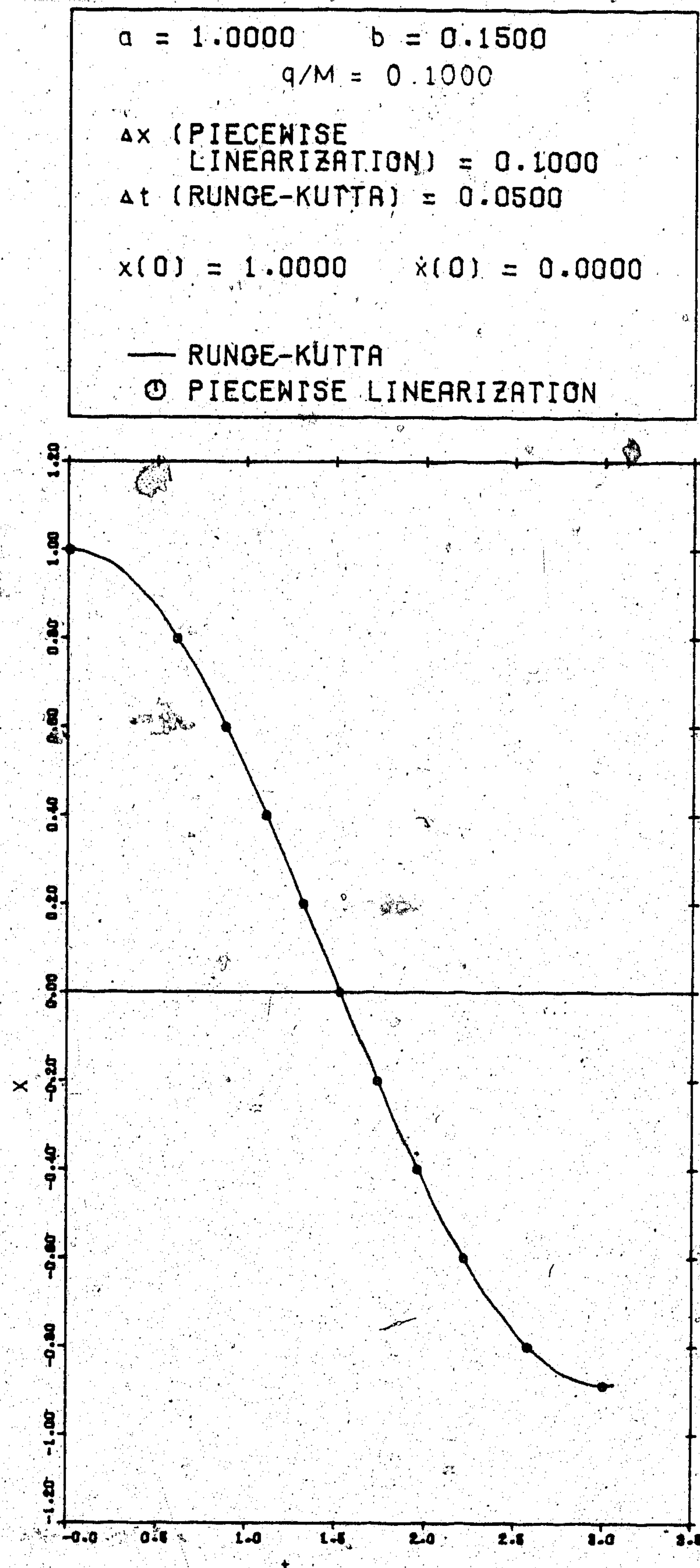

a = 1.0000 b = 0.1500
q/M = 0.1000
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
X
t

FIGURE 4.5 - SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR $a = 1.0$, $b = 2.0$, $\frac{q}{M} = 0.10$, $\Delta x(0) = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta method uses [5].)

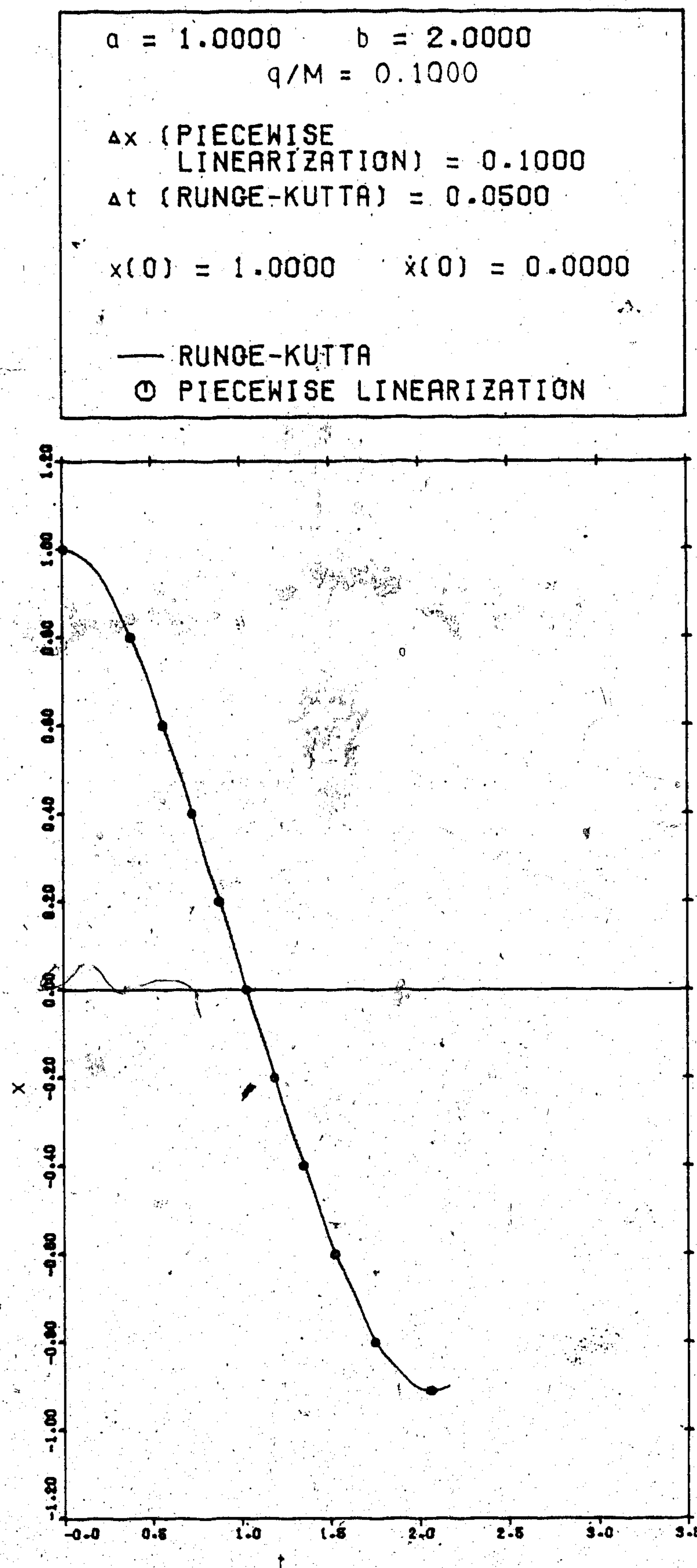

a = 1.0000 b = 2.0000
q/M = 0.1000
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
x
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
t

FIGURE 4.6 - SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR a = 1.0, b = 0.15, $\frac{g}{M}$ = 0.25, $\Delta x(0)$ = 0.10, Δt = 0.05, x(0) = 1.0, $\dot{x}(0)$ = 0.0

(N.B.: Runge-Kutta method uses [5].)

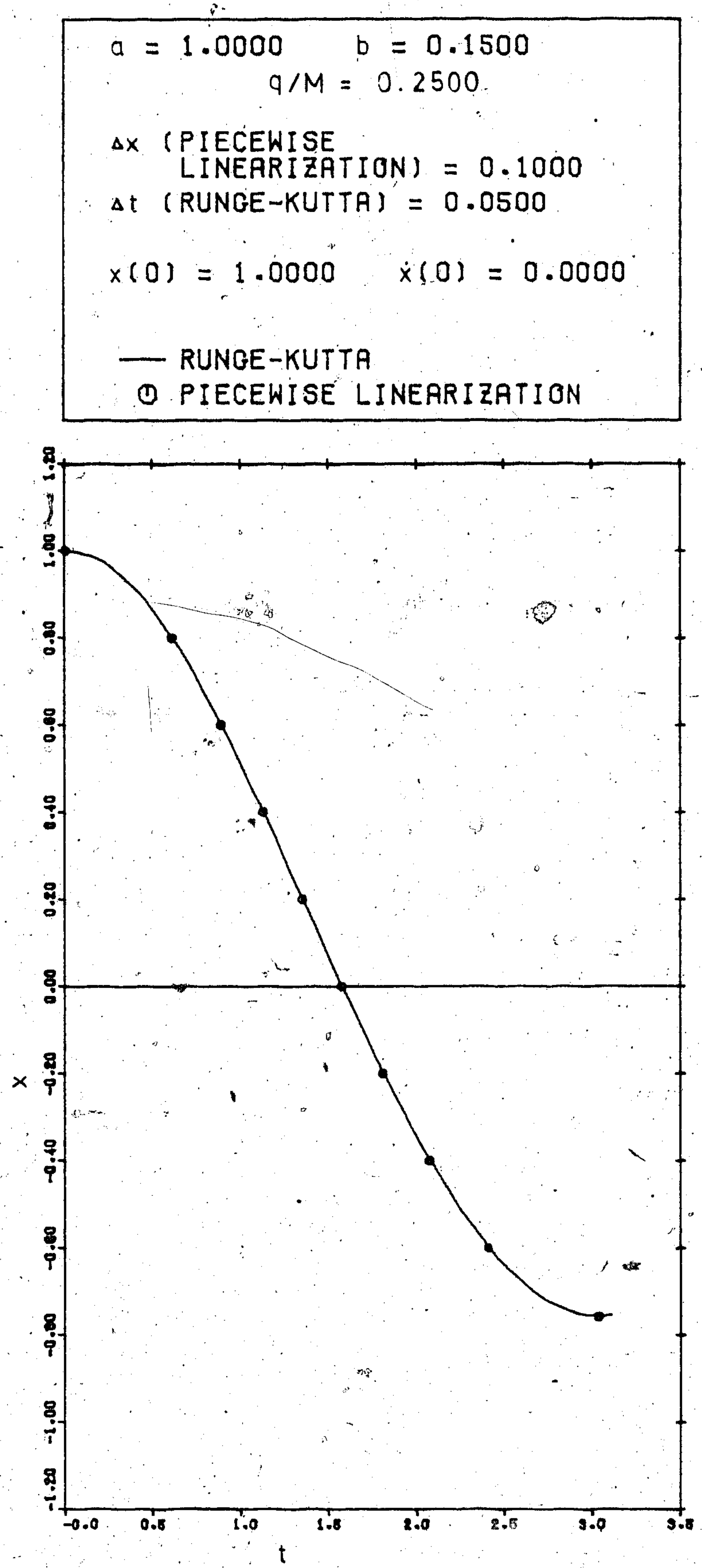
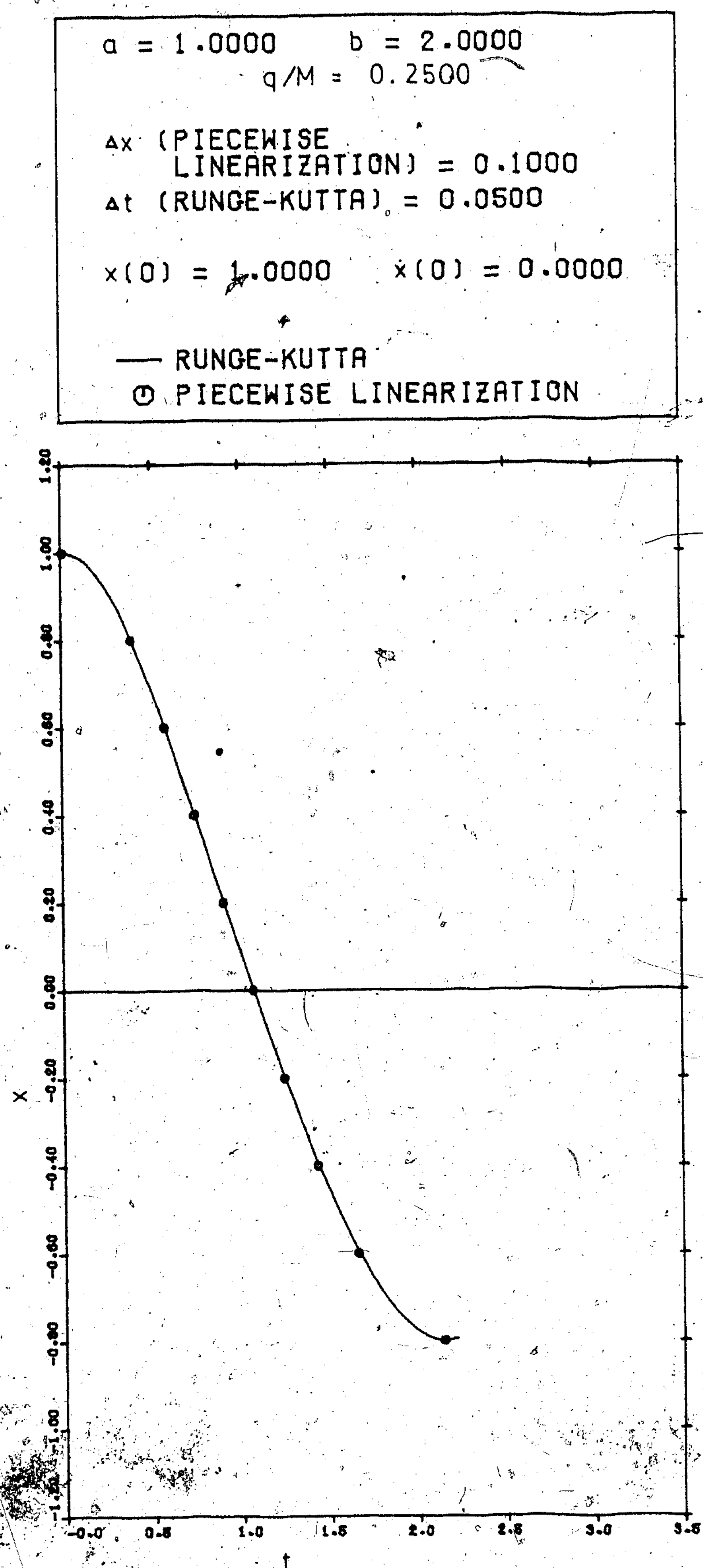

a = 1.0000 b = 0.1500
q/M = 0.2500
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
x
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
t

FIGURE 4.7 - SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR $a = 1.0$, $b = 2.0$, $\frac{q}{M} = 0.25$, $\Delta x(0) = 0.10$, $\Delta t = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta method uses [5].)


a = 1.0000 b = 2.0000
q/M = 0.2500
Δx (PIECEWISE LINEARIZATION) = 0.1000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000 ẋ(0) = 0.0000
— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
x
-0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
t

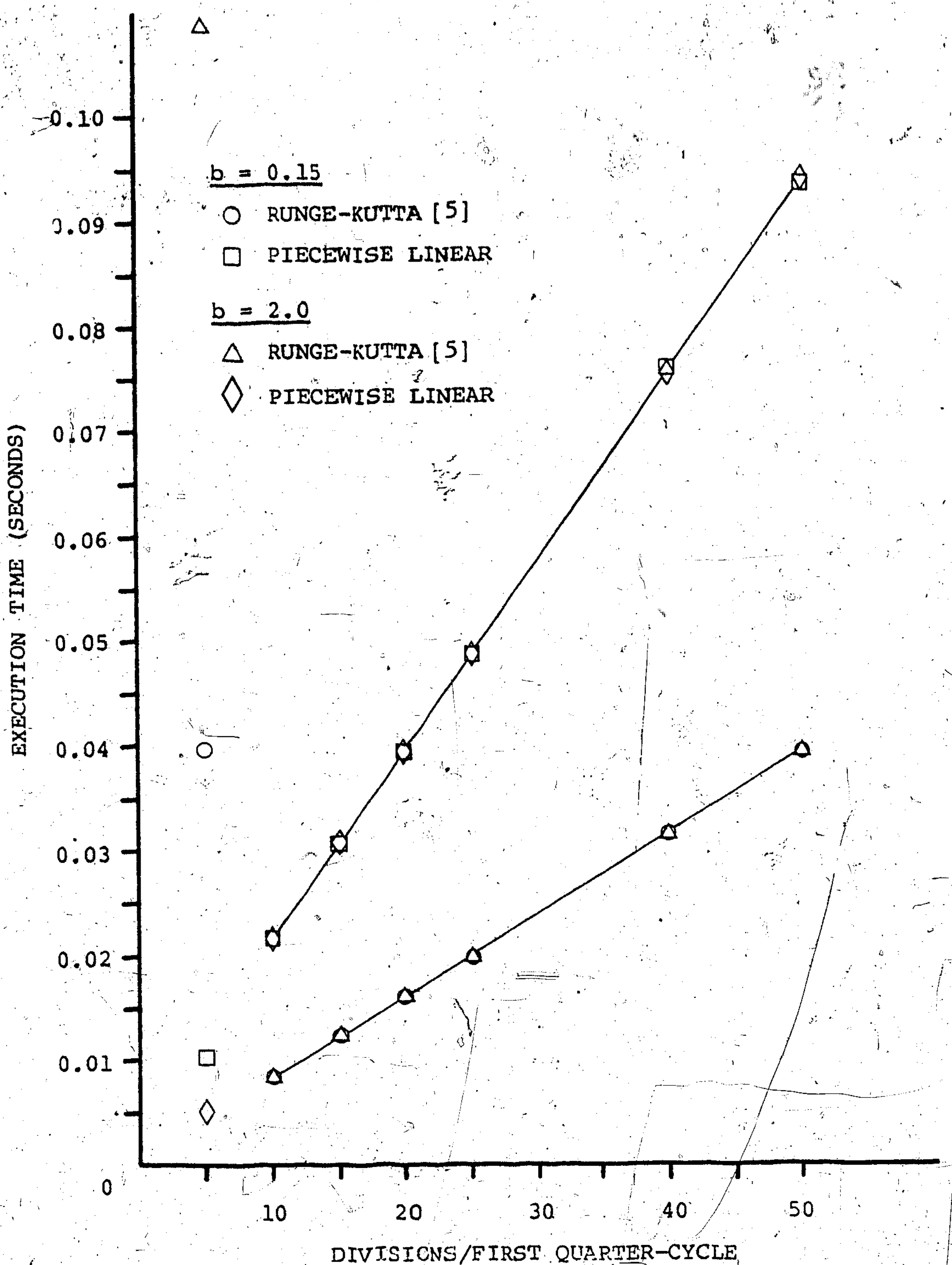


FIGURE 4.8 – EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR $a = 1.0$, $\frac{q}{M} = 0.0$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$ [27]

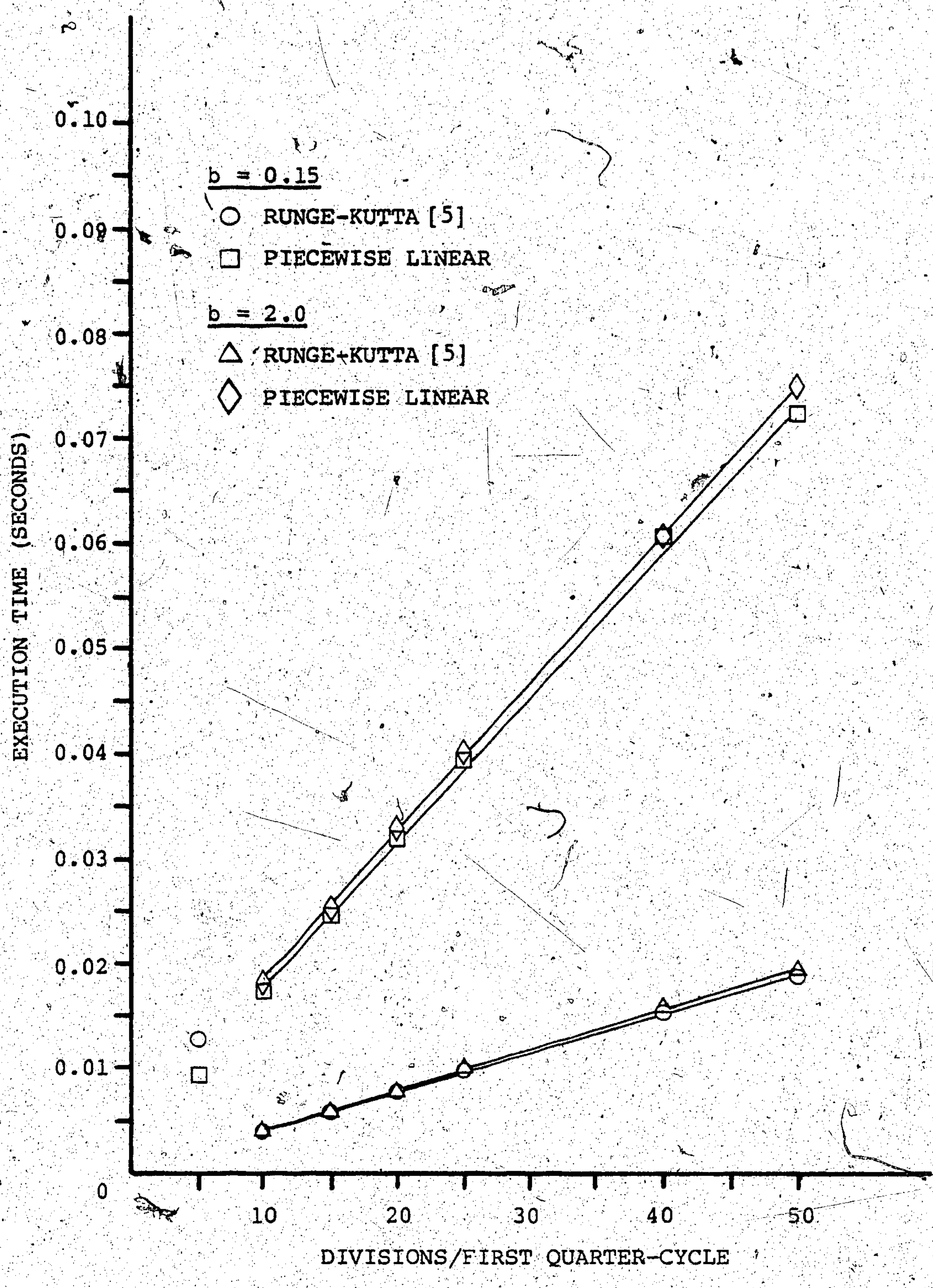


FIGURE 4.9 - EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR a = 1.0, $\frac{q}{M}$ = 0.10, x(0) = 1.0, $\dot{x}(0)$ = 0.0 [27]

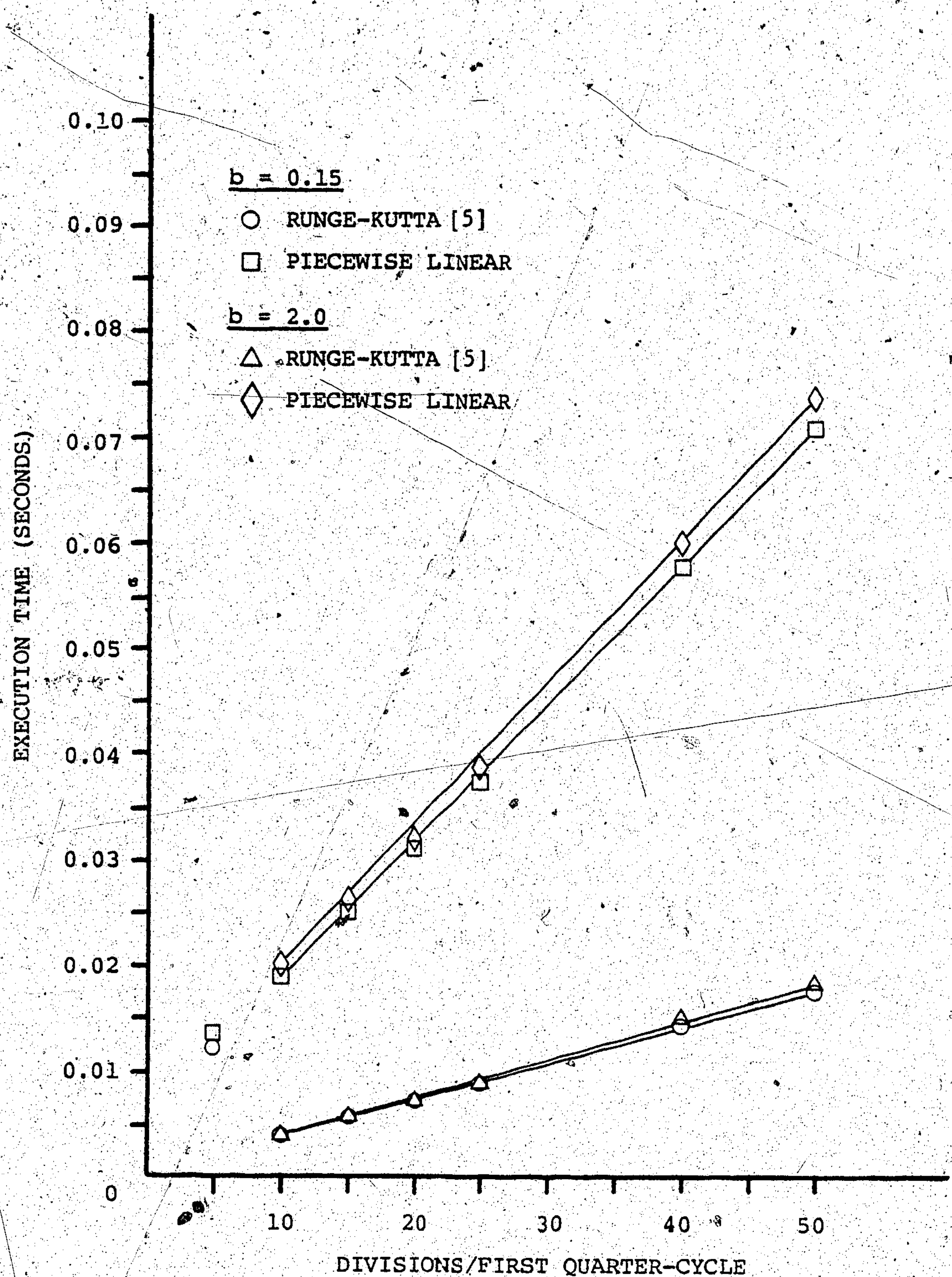


FIGURE 4.10 - EXECUTION TIMES FOR SOLUTIONS TO HARD SPRING EQUATION (NONLINEAR DAMPING) FOR a = 1.0, $\frac{q}{M} = 0.25$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$ [27]

slower version of the program.

Once again, it was found that the time required was linear with respect to the number of divisions per first quarter-cycle. As before, for 5 divisions of the first quarter-cycle, problems were experienced in obtaining a reasonable execution time for either result, depending upon the value of b and $\frac{q}{M}$ (since an increase in either or both caused the program to exceed its pre-set execution time limit [31]). This leads to the same conclusion as before that there may be a lower limit to the number of points that can be used for either method to be effective or economical. One of the things that was surprising was the lowering of the execution time for the piecewise method for a = 1.0, b = 2.0, and $\frac{q}{M}$ = 0.0. This could be regarded as coincidence, unless there was some factor in the computer system itself that may have led to this which is not immediately apparent.

V. VAN DER POL EQUATION

A. Preliminary Comments

The methods of solution described in the previous two chapters gave stable results since, when the damping factors were varied over a large range for the underdamped case, no erratic behaviour or improbable wave forms were encountered. In this chapter, however, an equation will be examined that is different in that the term representing its resisting force varies with respect to both the displacement and the velocity, its solution curve varies radically as a parameter is varied over a wide range, and its phase trajectory exhibits a limit cycle. This is Van der Pol's equation [32]:

$$\ddot{x} + \mu(x^2 - 1)\dot{x} + x = 0 \quad . \qquad (5.1.1)$$

The solutions to the linear equations are essentially the same as were found in the previous chapters, except that instead of being restricted to only the underdamped situation, the critically damped and overdamped cases are solved for as well, owing to the nature of the solution to (5.1.1).

Owing to the wide variation in behaviour that this equation can exhibit, the initial conditions will be the same as had been previously used with the exception that X_0 will vary in value to allow investigation inside, on, and outside the limit cycle. The parameter μ will have the values 0.25, 1.0, 2.0, and 5.0 so that the variations in the solution curves can be examined.

While the previous systems had nonlinear restoring forces, (5.1.1) does not, and so, this will mean that many of the calculations that were needed in order to set up the solution to the linear equation are not needed in this situation. This also means that an interval of time can be used instead of displacement with resulting simplification in the calculations.

Since an exact solution does not seem to exist, the same Runge-Kutta solution [5] that had been used in the previous two chapters will be the benchmark against which the piecewise linearization will be compared.

B. Derivation of Piecewise Linear Solution

Equation (5.1.1) can be approximated over a small increment of time by the equation [29]:

$$\ddot{x} + 2n\dot{x} + p^2 x = 0 \,, \tag{5.2.1}$$

where:

$$n \approx \frac{\mu(x^2 - 1)}{2} \,, \tag{5.2.2}$$

$$p^2 = 1.0 \,.$$

The solutions of (5.2.1) can be found in texts (for example, Timoshenko, et. al. [29]), and are as follows.

For $p^2 - n^2 > 0$ [35], [36]:

$$x = e^{-nt}\left[x_0 \cos p^*t + \left(\frac{nx_0 + \dot{x}_0}{p^*}\right)\sin p^*t\right] \,, \tag{5.2.3}$$

$$\dot{x} = e^{-nt}\{\dot{x}_0 \cos p^*t + \left[\frac{-n(nx_0 + \dot{x}_0)}{p^*} - p^*x_0\right] \sin p^*t\} , \tag{5.2.4}$$

$$p^* = \sqrt{p^2 - n^2} .$$

For $p^2 - n^2 = 0$ [36], [37]:

$$x = e^{-pt}\left[x_0 + (nx_0 + \dot{x}_0)t\right] , \tag{5.2.5}$$

$$\dot{x} = e^{-pt}\left[-px_0 + (nx_0 + \dot{x}_0)(1 - pt)\right] . \tag{5.2.6}$$

For $p^2 - n^2 < 0$ [36], [37]:

$$x = e^{-nt}\left[x_0 \cosh (p^{**}t) + \left(\frac{nx_0 + \dot{x}_0}{p^{**}}\right) \sinh (p^{**}t)\right] , \tag{5.2.7}$$

$$\dot{x} = e^{-nt}\{\dot{x}_0 \cosh (p^{**}t) + \left[\frac{-n(nx_0 + \dot{x}_0)}{p^{**}} + p^{**}x_0\right] \sinh (p^{**}t)\} , \tag{5.2.8}$$

$$p^{**} = \sqrt{n^2 - p^2} .$$

The major problem here is how to estimate n. Assume at the beginning of a segment some initial conditions of x_0 and $\dot{x}_0$, and a time interval of Δt_p. An initial guess would use x_0 for x in (5.2.2) and an initial estimate of the change in displacement is calculated (which will

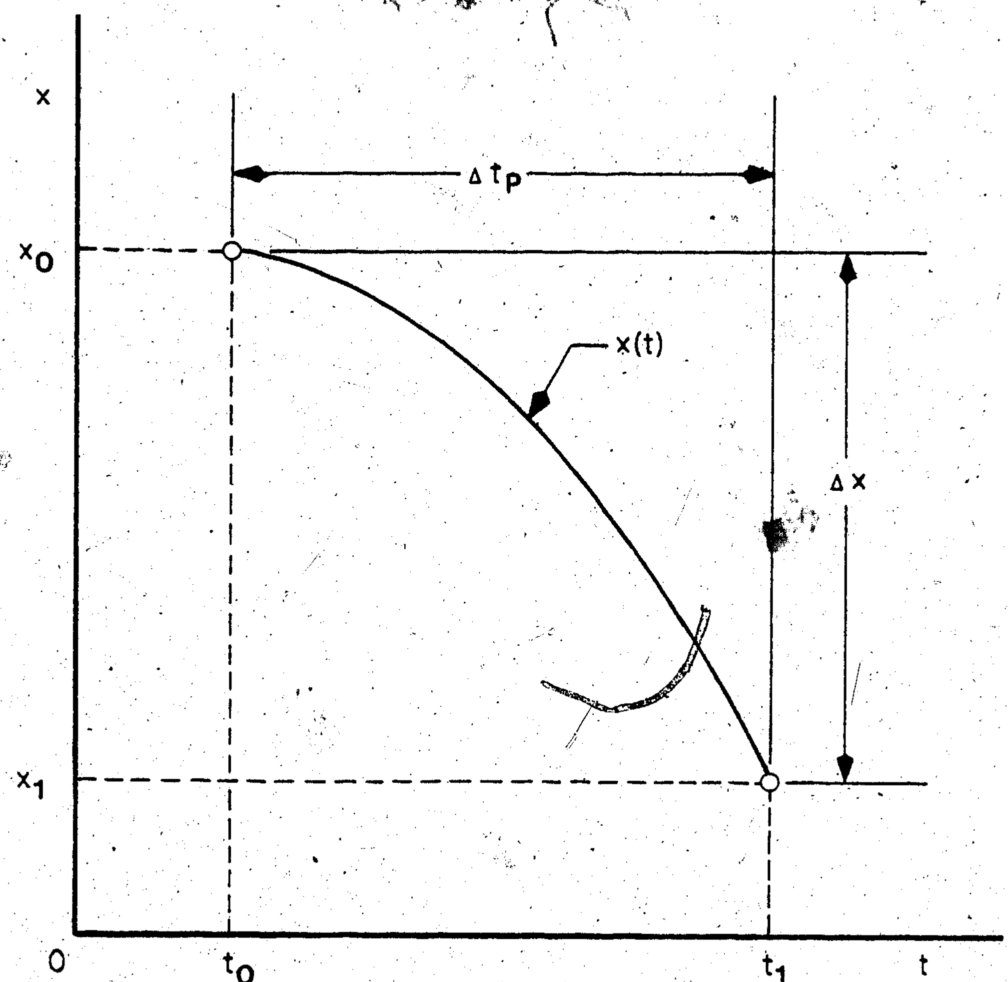


FIGURE 5.1 - TYPICAL INTERVAL LAYOUT (VAN DER POL EQUATION)

be explained further on). The new displacement x_1 is $x_0 + \Delta x$, and this is compared to a previously calculated value of x_1 (x_{PREV}). Should the difference between x_1 and x_{PREV} be very small, x_1 is the final displacement at the end of Δt_p. If not, x_1 becomes x_{PREV}, x now is $x_0 + \Delta x/2$, and a new n is found, as well as new values for Δx and x . Figure 5.1 should clarify matters.

This is how the solution is carried out in general for each interval. At the beginning of the iteration,

t_0 = Initial time,

$t_0 + \Delta t_p$ = Final time

$= t_1$,

x_0 = Initial displacement

$x_0 + \Delta x$ = Final displacement

$= x_1$,

$\dot{x}_0$ = Initial velocity,

$\dot{x}_1$ = Final velocity.

Once the displacement approaches either a peak or trough, a variation in this pattern is introduced to find the time and displacement at these points. (See Figure 5.2 for details.) The procedure is as follows. Starting at x_0 and t_0, x_1 is found as before. The value of $\text{sgn}(\dot{x}_1)$ is calculated, and if found to be different from $\text{sgn}(\dot{x}_0)$, Δt_1 (the time taken to reach x_{11}) is found by determining the time interval needed for $\dot{x}$ to reach zero using a Newton-Raphson method [30], starting at x_0. This assumes that $\dot{x}$ not zero at the end of a complete segment. Should it be, this section is circumvented.

The value for x_{11} is found by iteration much like the bottom of the

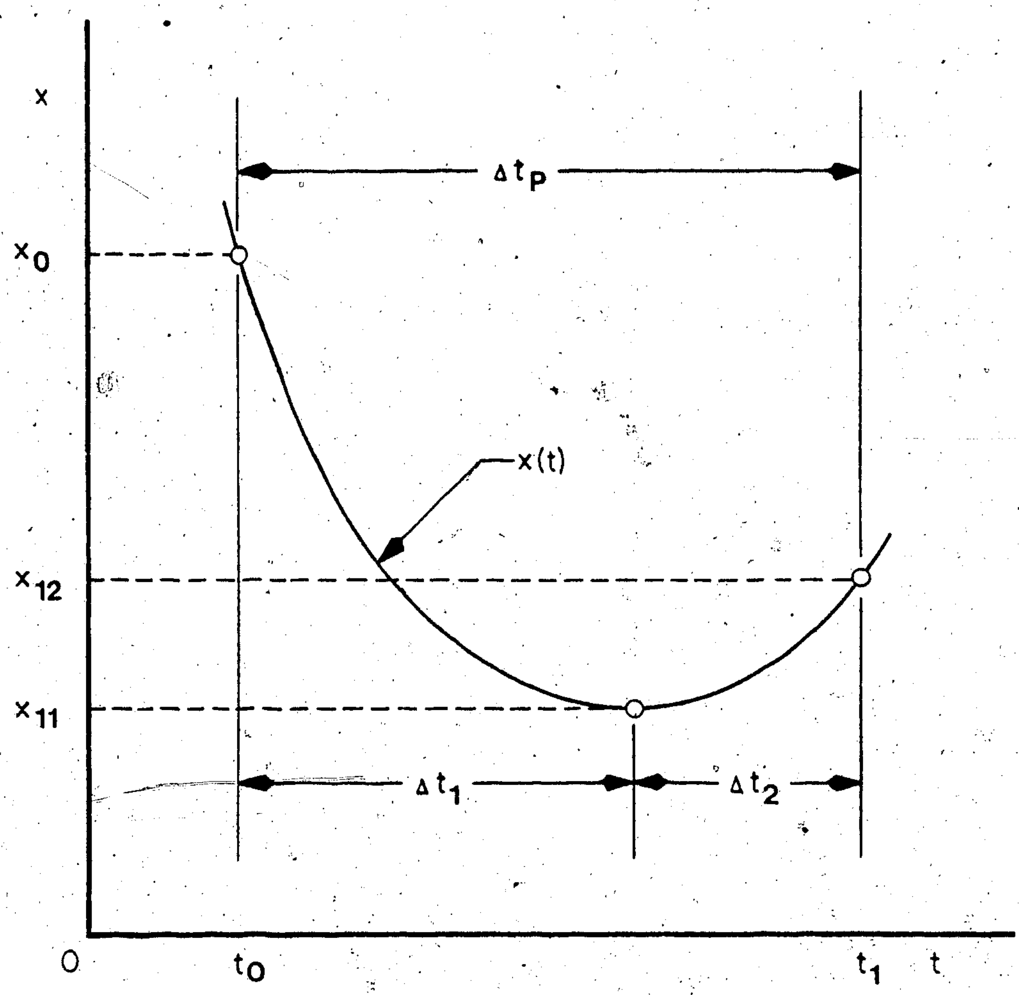


FIGURE 5.2 - LAYOUT OF PEAK OR TROUGH (VAN DER POL EQUATION)

first trough was determined in the previous two chapters. The estimated value is x_F, with the first guess being x_{12}. Then, using Δt_1, a change in displacement is found (a new Δx). The value of $(x_0 + \Delta x) - x_F$ is determined, and if very small, x_{11} is equal to $x_0 + \Delta x$. If not, $x_0 + \Delta x$ becomes x_F and the iteration starts again.

This completes the first half-cycle.

Since the peak or trough is most likely not going to occur at the end of a time interval, and it would be convenient to find the displacement at the end of Δt_P (in order to continue afterwards with even time intervals until the next peak or trough), x_{12} is found based upon Δt_P, with the remainder of the half-cycle as was previously done. The displacement-time curve and phase-plane diagram can now be obtained.

It is necessary to go through this procedure as one never knows in advance how long the system takes to reach the limit cycle, since it depends upon the value of as well as the initial conditions. So, for convenience, the number of half-cycles that had been calculated will be monitored and the calculation stops once the required number had been achieved.

When the displacement approaches a peak or trough, Δt_1 is determined using the Newton-Raphson method [30]:

$$(\Delta t)_{i+1} = (\Delta t)_i - \frac{\dot{x}[(\Delta t)_i]}{\ddot{x}[(\Delta t)_i]}$$

where $\dot{x}[(\Delta t)_i]$ is the expression for the velocity (which particular one would depend upon what value $p^2 - n^2$ has), and $\ddot{x}[(\Delta t)_i]$ is the acceleration for that particular situation.

The following approximations to the velocities and accelerations are used, based upon Timoshenko, et. al. [34].

For $p^2 - n^2 > 0$ [35], [36]:

$$\dot{x}\left[(\Delta t)_i\right] = e^{-n(\Delta t)_i}\{A\cos\left[p^*(\Delta t)_i\right] + B\sin\left[p^*(\Delta t)_i\right]\} , \tag{5.2.9}$$

$$\ddot{x}\left[(\Delta t)_i\right] = e^{-n(\Delta t)_i}\{(-nA + p^*B) \cdot \cos\left[p^*(\Delta t)_i\right] + (-p^*A - nB)\sin\left[p^*(\Delta t)_i\right]\} , \tag{5.2.10}$$

$$A = \dot{x}_0,$$

$$B = -\frac{n(nx_0 + \dot{x}_0)}{p^*} - p^*x_0 .$$

For $p^2 - n^2 = 0$ [35], [36]:

$$\dot{x}\left[(\Delta t)_i\right] = e^{-p(\Delta t)_i}\{-px_0 + (nx_0 + \dot{x}_0) \cdot \left[1 - p(\Delta t)_i\right]\} , \tag{5.2.11}$$

$$\ddot{x}\left[(\Delta t)_i\right] = e^{-p(\Delta t)_i}\{p^2x_0 - p(nx_0 + \dot{x}_0) \cdot \left[2 - p(\Delta t)_i\right]\} . \tag{5.2.12}$$

For $p^2 - n^2 < 0$ [35], [36]:

$$\dot{x}\left[(\Delta t)_i\right] = e^{-n(\Delta t)_i}\{C\cosh\left[p^{**}(\Delta t)_i\right] + D\sinh\left[p^{**}(\Delta t)_i\right]\} , \tag{5.2.13}$$

$$\dot{x}\left[(\Delta t)_i\right] = e^{-n(\Delta t)_i}\{(-nC + p^{**}D) \cdot \cosh\left[p^{**}(\Delta t)_i\right] + (p^{**}C - nD)\sinh\left[p^{**}(\Delta t)_i\right]\} , \quad (5.2.14)$$

$$C = \dot{x}_0,$$

$$D = -\frac{n(nx_0 + \dot{x}_0)}{p^{**}} + p^{**}x_0 .$$

During these calculations, the value of n is found, and this ultimately determines what the solution for the segment will be. The change in displacement, Δx, is found by finding $(x_1 - x_0)$, corresponding to the appropriate value of n. Since equal time intervals are used, Δt_p is used instead of t in equations (5.2.3) - (5.2.8), where $\Delta t_p = t_1 - t_0$ (based on the functions being dependent upon $t - t_0$ instead of t, due to the piecewise linearization).

C. Results

Equations (5.2.3) - (5.2.14) were assembled into a computer program, as well as a Runge-Kutta solution using DVERK [5], and the results obtained by running it on the Amdahl 470 V/8 [26].

Initially, the approximation was tested by using a small value for μ as a check that it worked. This was arbitrarily chosen to be 0.25 and was tested for X_0 being equal to 1.0, 2.0, and 2.5 with Δt_p for the piecewise linearization being equal to 0.2, and the Δt_{RK} for the Runge-Kutta solution [5] being 0.05. The size of the time intervals was chosen arbitrarily. The results are seen on Figures 5.3 - 5.8. (All solutions and phase trajectories in this chapter are based on [38],

FIGURE 5.3 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 0.25$, $\Delta t_p = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

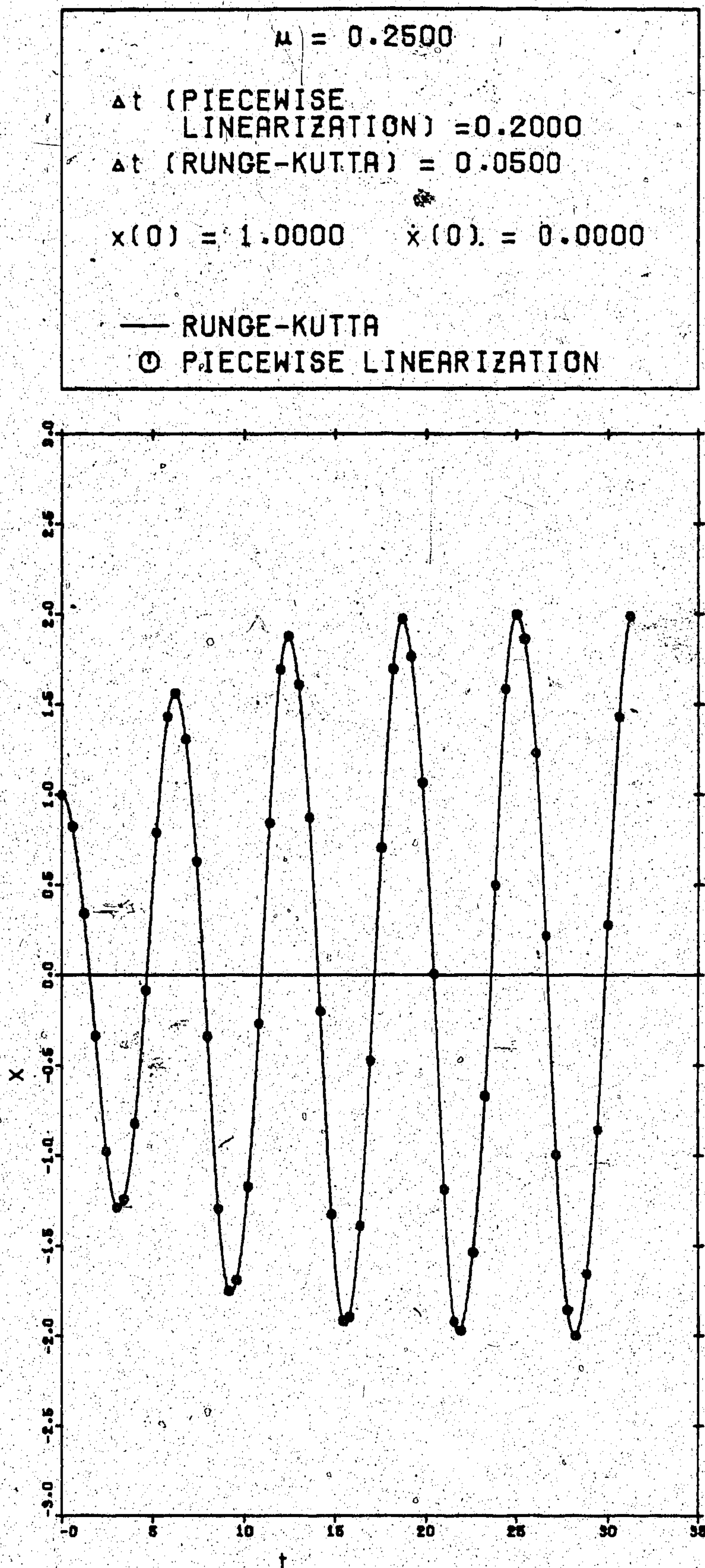
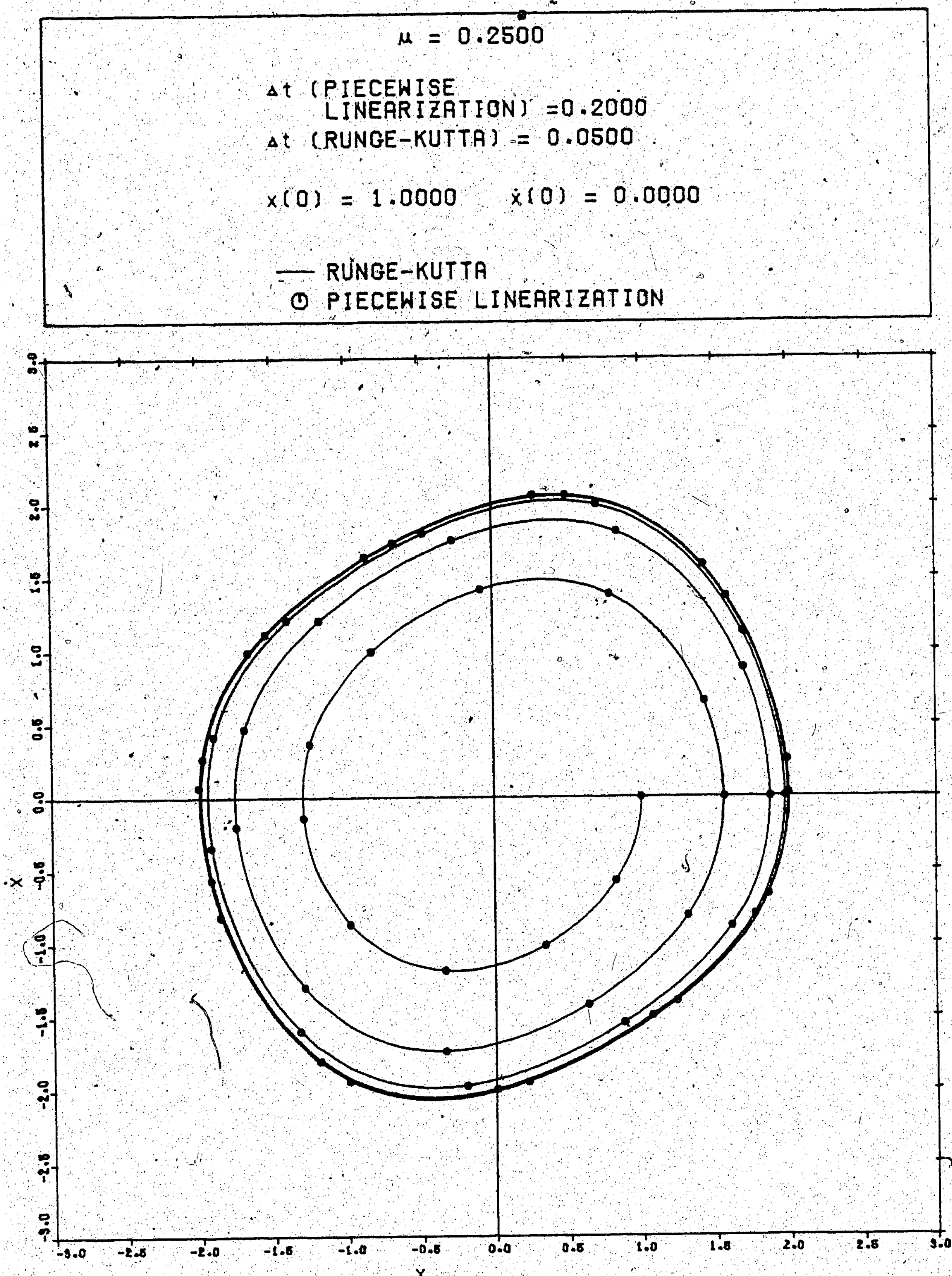

μ = 0.2500
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
X
t
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-0
5
10
15
20
25
30
35

FIGURE 5.4 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 0.25$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)



μ = 0.2500
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000   ẋ(0) = 0.0000
—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
ẋ
X
-3.0
-2.5
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

FIGURE 5.5 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 0.25$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

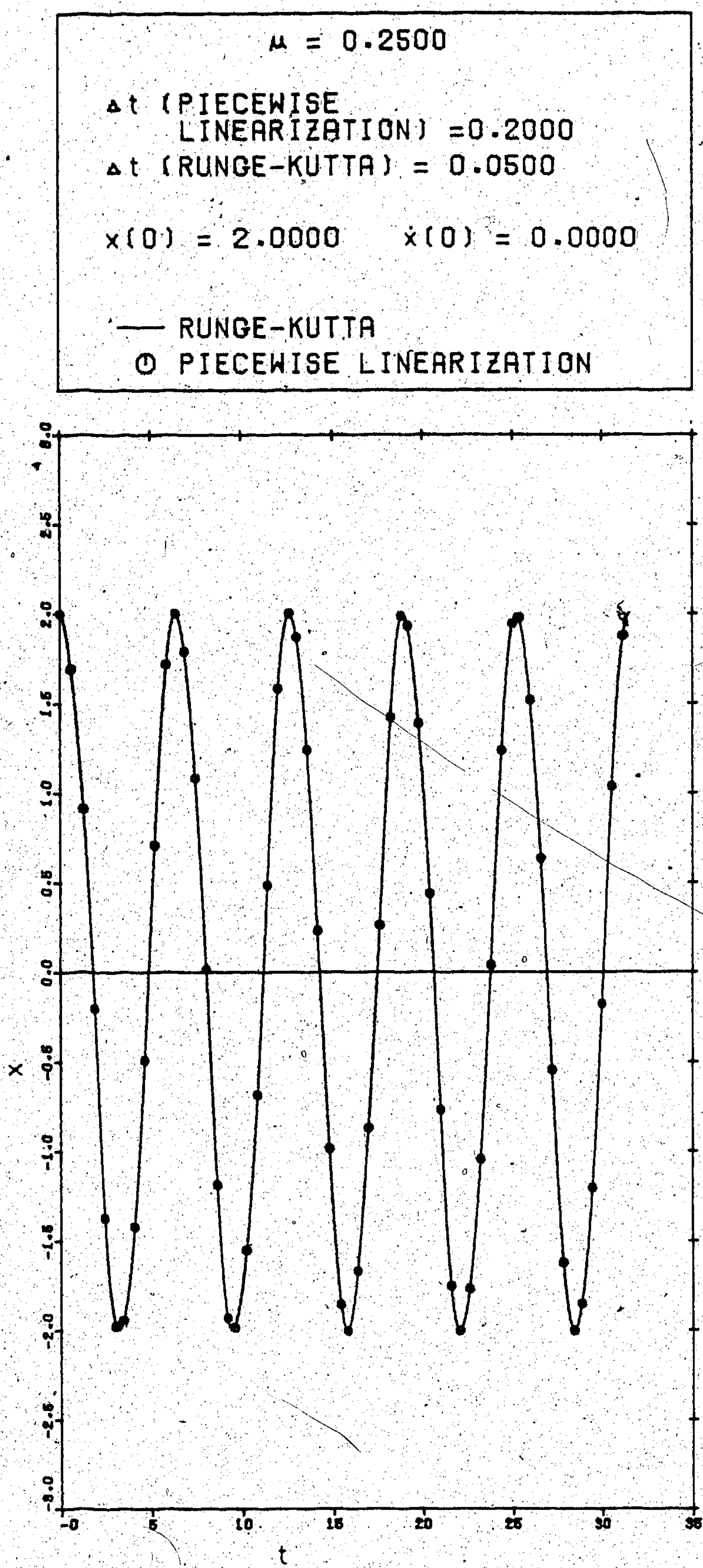

μ = 0.2500
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.0000   ẋ(0) = 0.0000
—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
X
t
0
5
10
15
20
25
30
35
-3.0
-2.5
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

FIGURE 5.6 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 0.25$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)

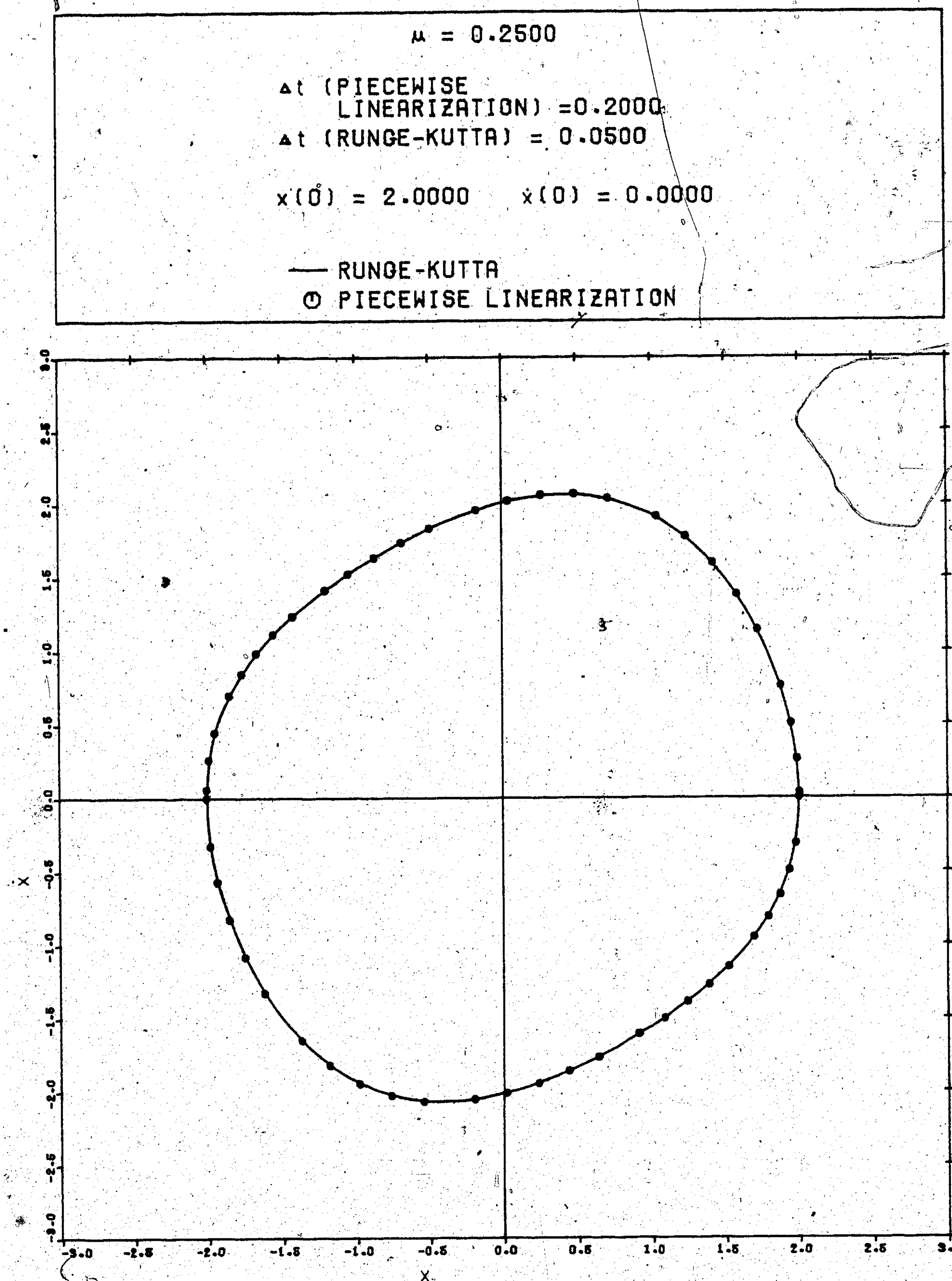

μ = 0.2500
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.0000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
ẋ
X

FIGURE 5.7 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 0.25$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

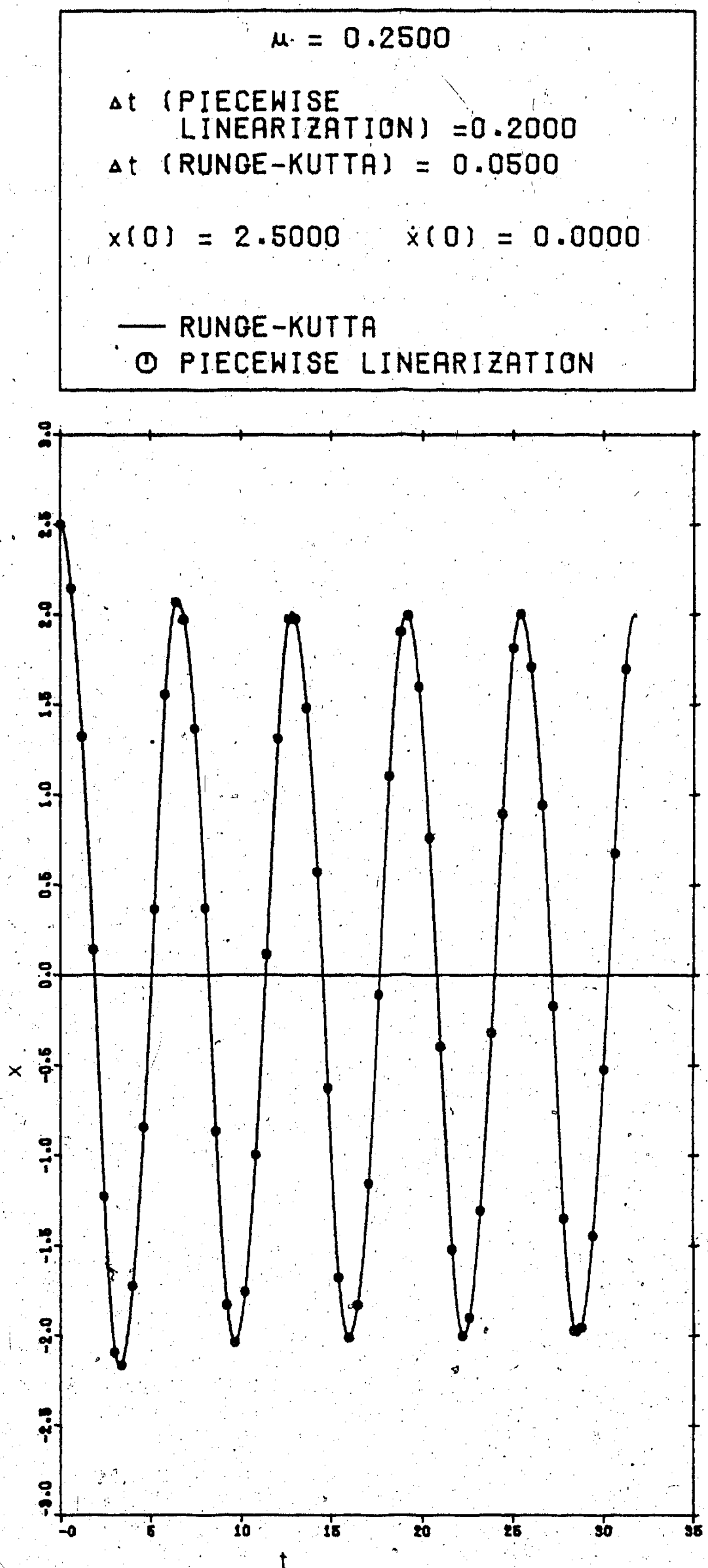
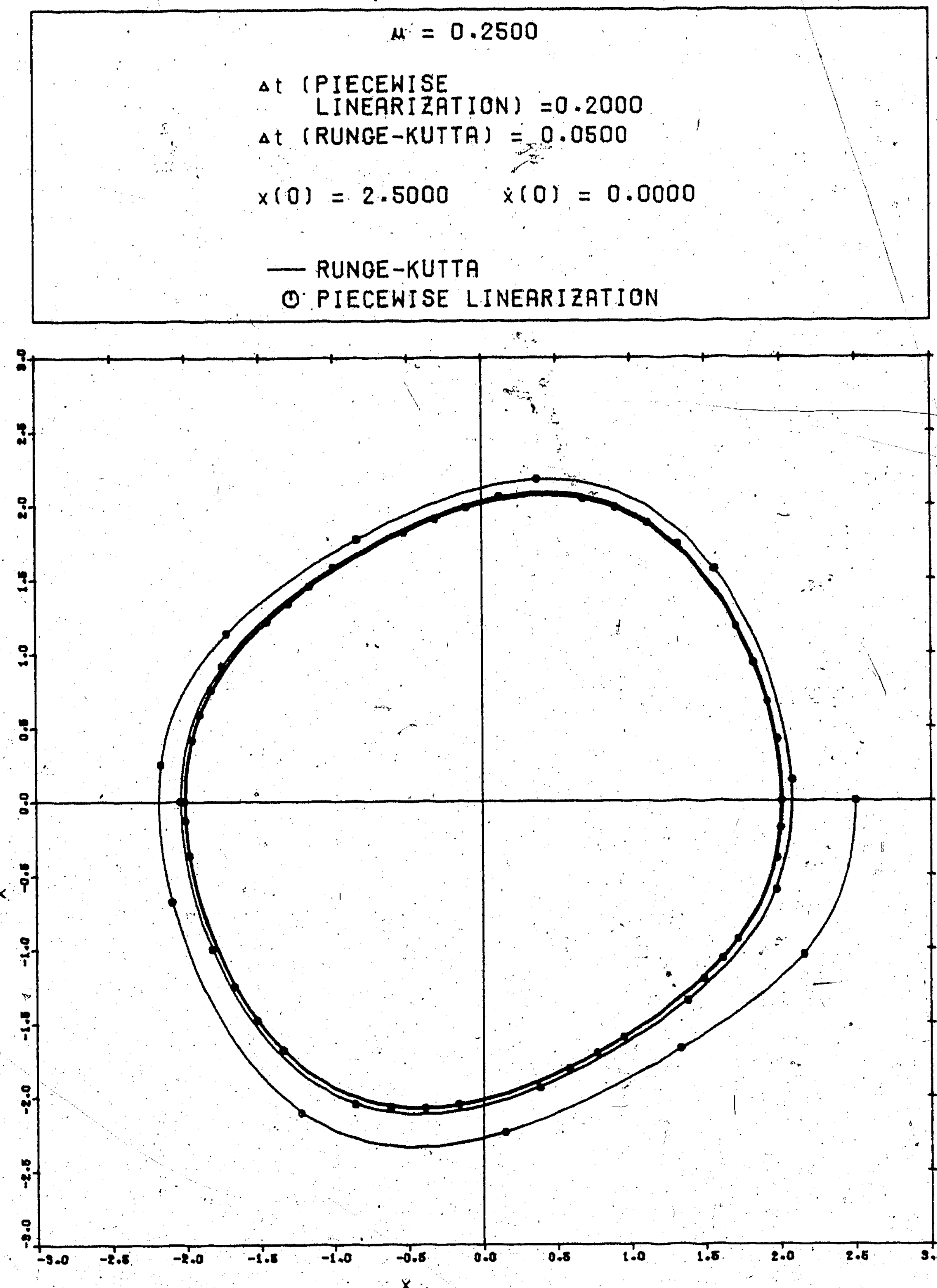

μ = 0.2500
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
x
t
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
0
5
10
15
20
25
30
35

FIGURE 5.8 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 0.25$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)


μ = 0.2500
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000   ẋ(0) = 0.0000
—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
-3.0
-2.5
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X
Ẋ

[39].) The development of the limit cycle can be seen in the phase-plane diagrams and there appear to be no major difficulties in achieving the desired results. As expected for an initial displacement outside the limit cycle, the solution moves in towards it, while moving out to it when X_0 is inside it. Even for the initial displacement being on the limit cycle, there seem to be no problems.

The next thing was to increase the value of μ while keeping the values of the respective time intervals the same. It had also been decided to restrict the remaining investigation to the situation of the initial displacement outside the limit cycle, except for μ of 5.0 where X_0 will be varied as had been done for μ of 0.25.

The results of this can be seen in Figures 5.9 - 5.18 [38], [39], but something unusual appears in Figures 5.13 - 5.18 [38], [39]. Up to this point, the plots had given good fits to the solution being compared (with some minor deviations for μ of 2.0), while for these curves, this does not seem to be the case. Immediately the piecewise linearization was suspected to be at fault and both the derivation and the computer program were checked several times. Nothing that could introduce what would appear to be a phase difference was found. Internal roundoff error in the computer was also suspected, and the tolerances were reset to smaller values and, as far as possible, the program was reset to operate in double precision. But, the results were still the same--a phase difference, or what appeared to be as such, somehow arose in the system.

A smaller time interval of 0.05 was tried for the piecewise linearization, with the results for $x(0) = 2.5$ and $\mu = 5.0$ to be seen in Figures 5.19 and 5.20 [38], [39]. This seems to be the key to the

FIGURE 5.9 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 1.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

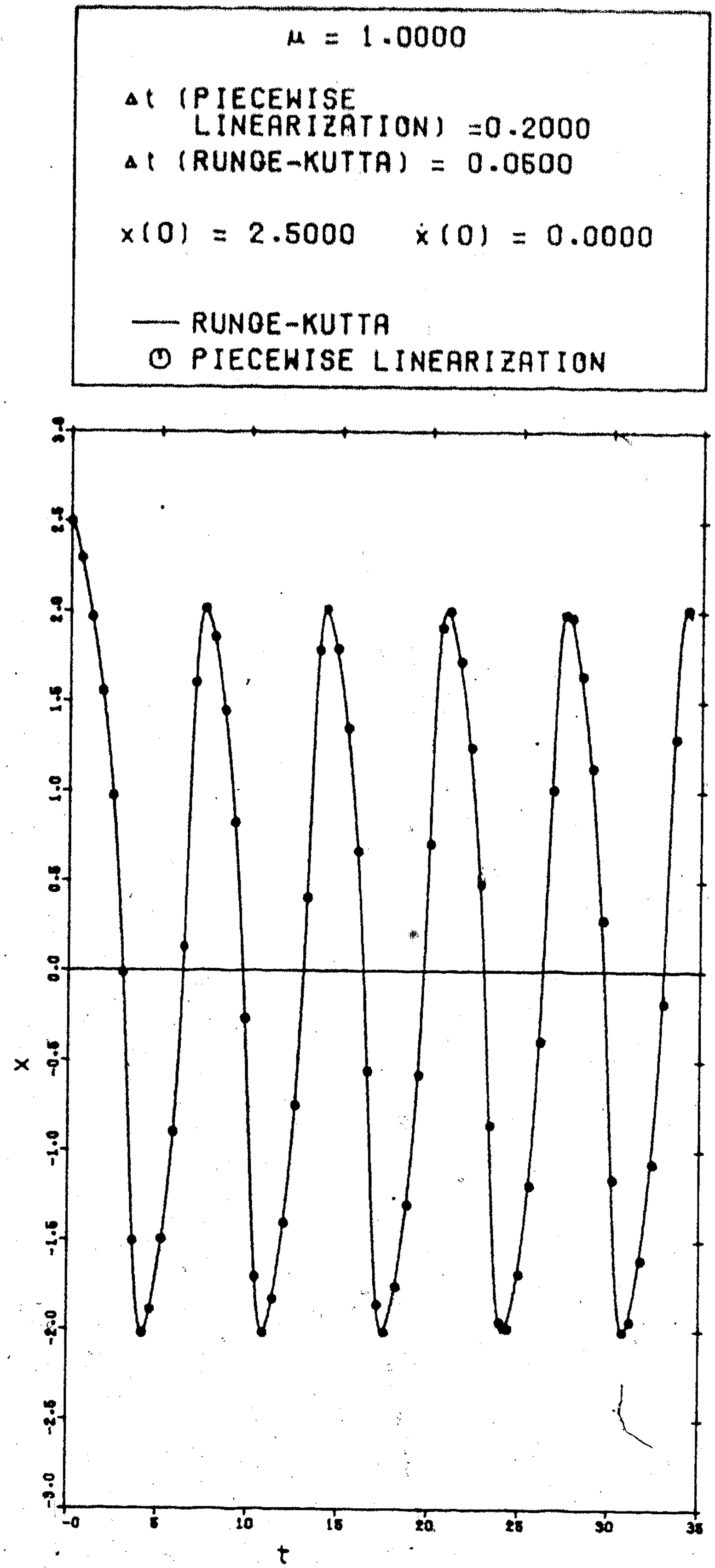
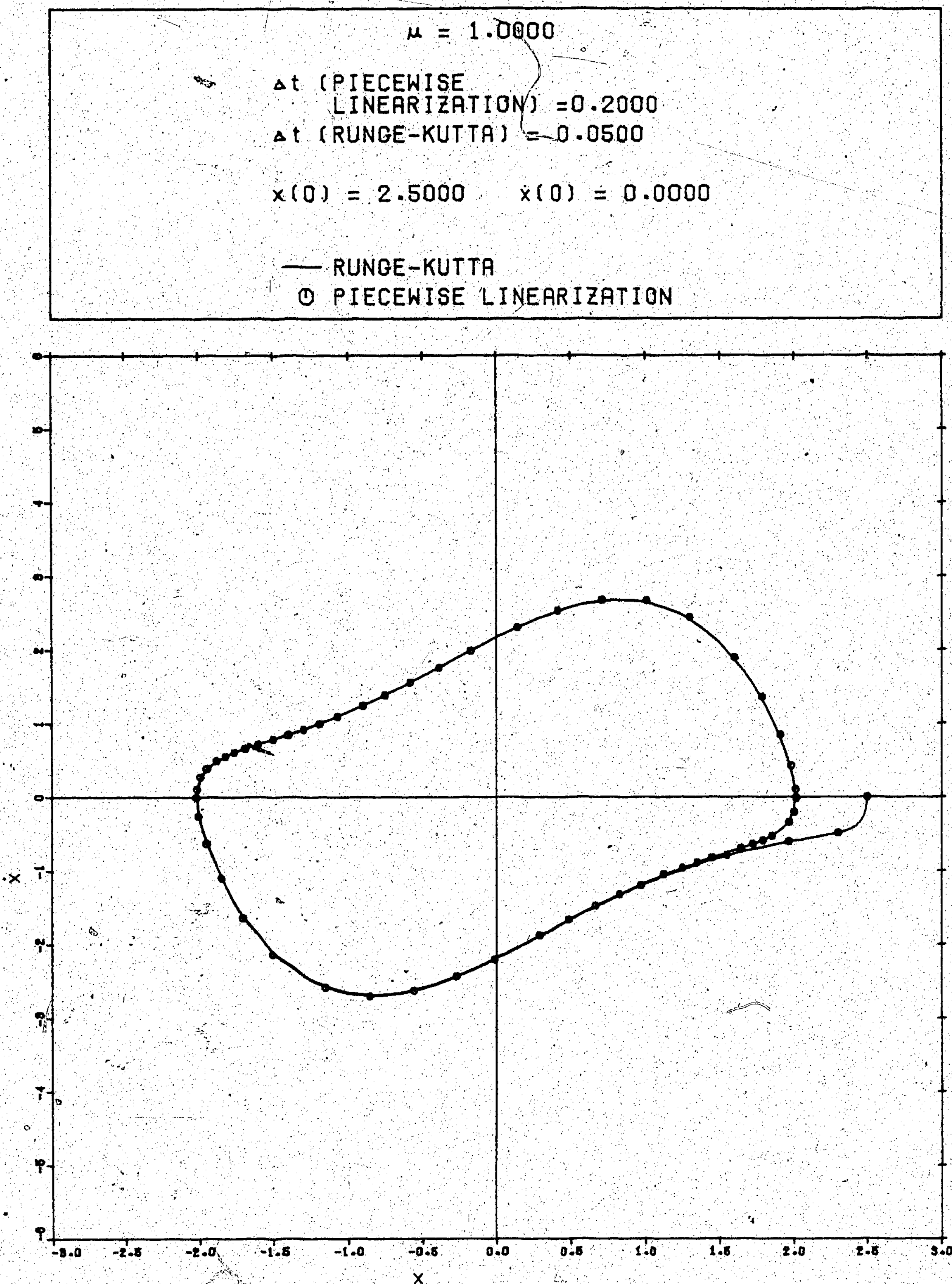

μ = 1.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0600
x(0) = 2.5000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
x
t
0
5
10
15
20
25
30
35

FIGURE 5.10 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 1.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)



μ = 1.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000   ẋ(0) = 0.0000
—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
-3.0
-2.5
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X
Ẋ

FIGURE 5.11 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 2.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

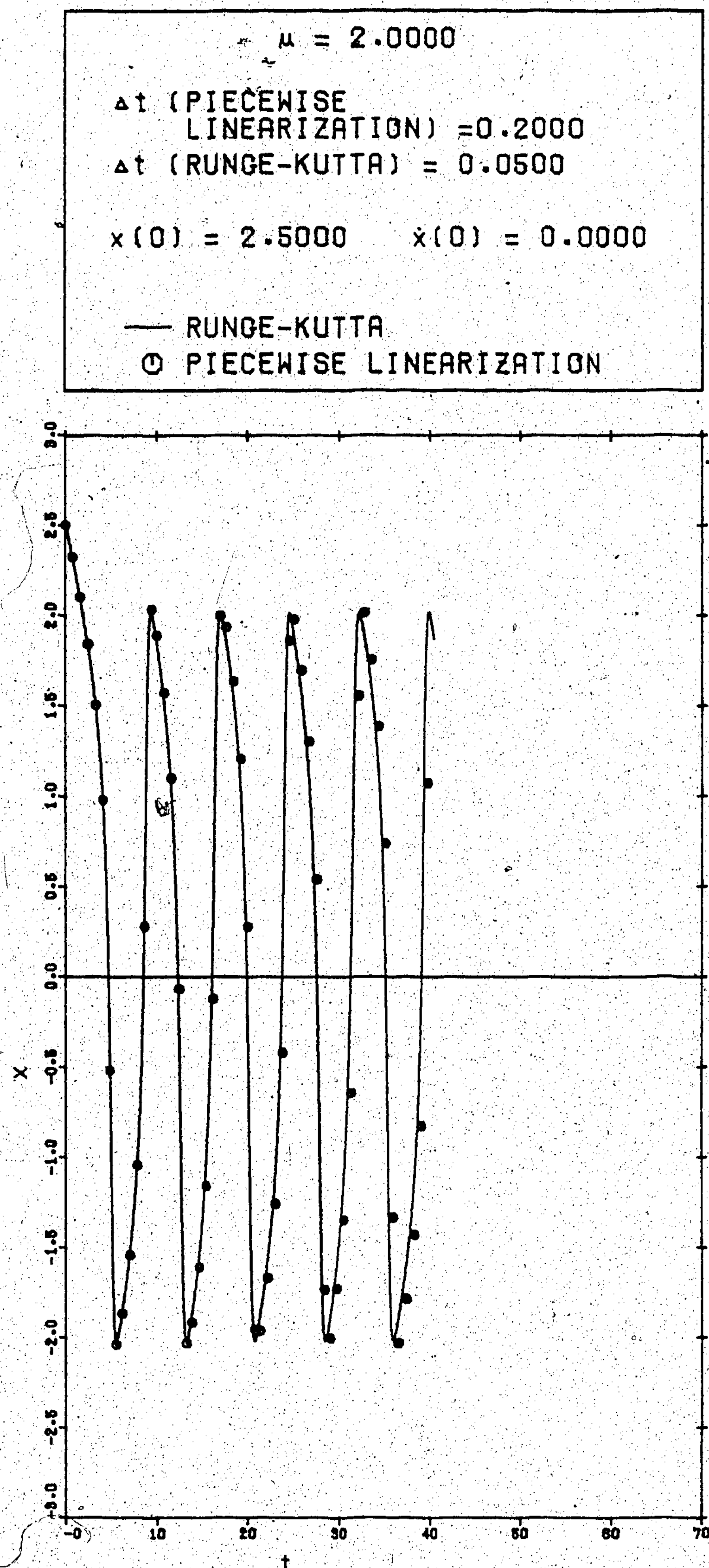

μ = 2.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000   ẋ(0) = 0.0000
—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
X
t
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
0
10
20
30
40
50
60
70

FIGURE 5.12 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 2.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)

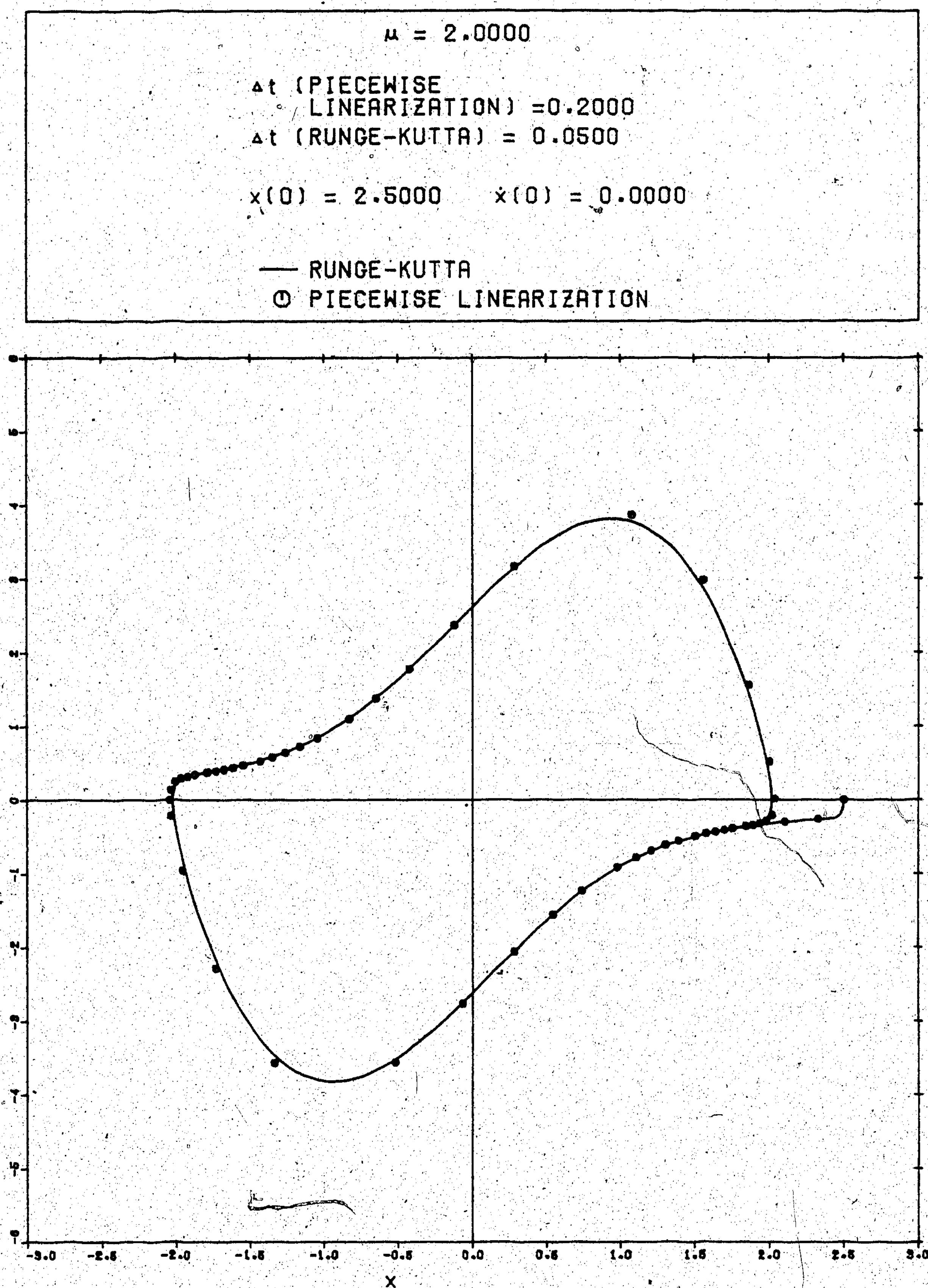

μ = 2.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000   ẋ(0) = 0.0000
—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
-3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0
X
Ẋ

FIGURE 5.13 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

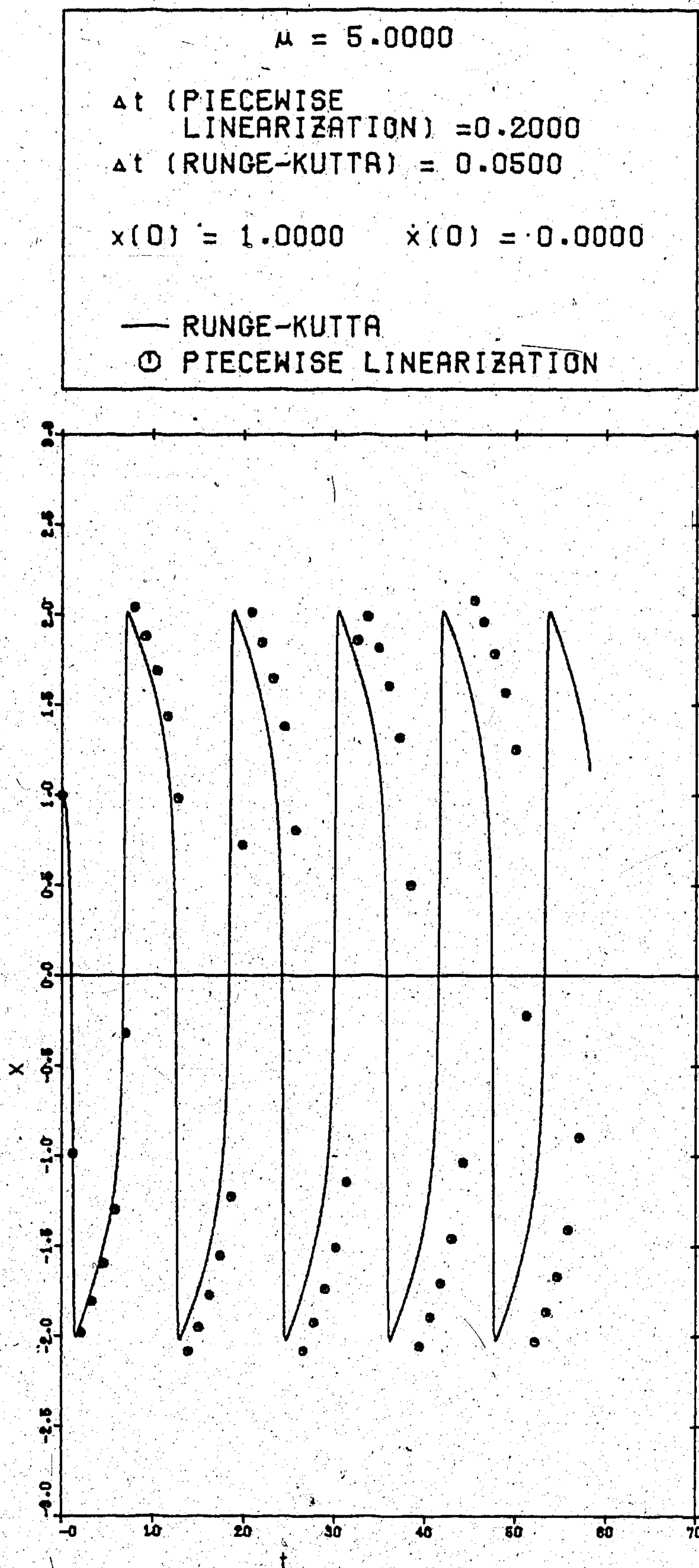

μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
X
t
10
20
30
40
50
60
70

FIGURE 5.14 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 1.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)

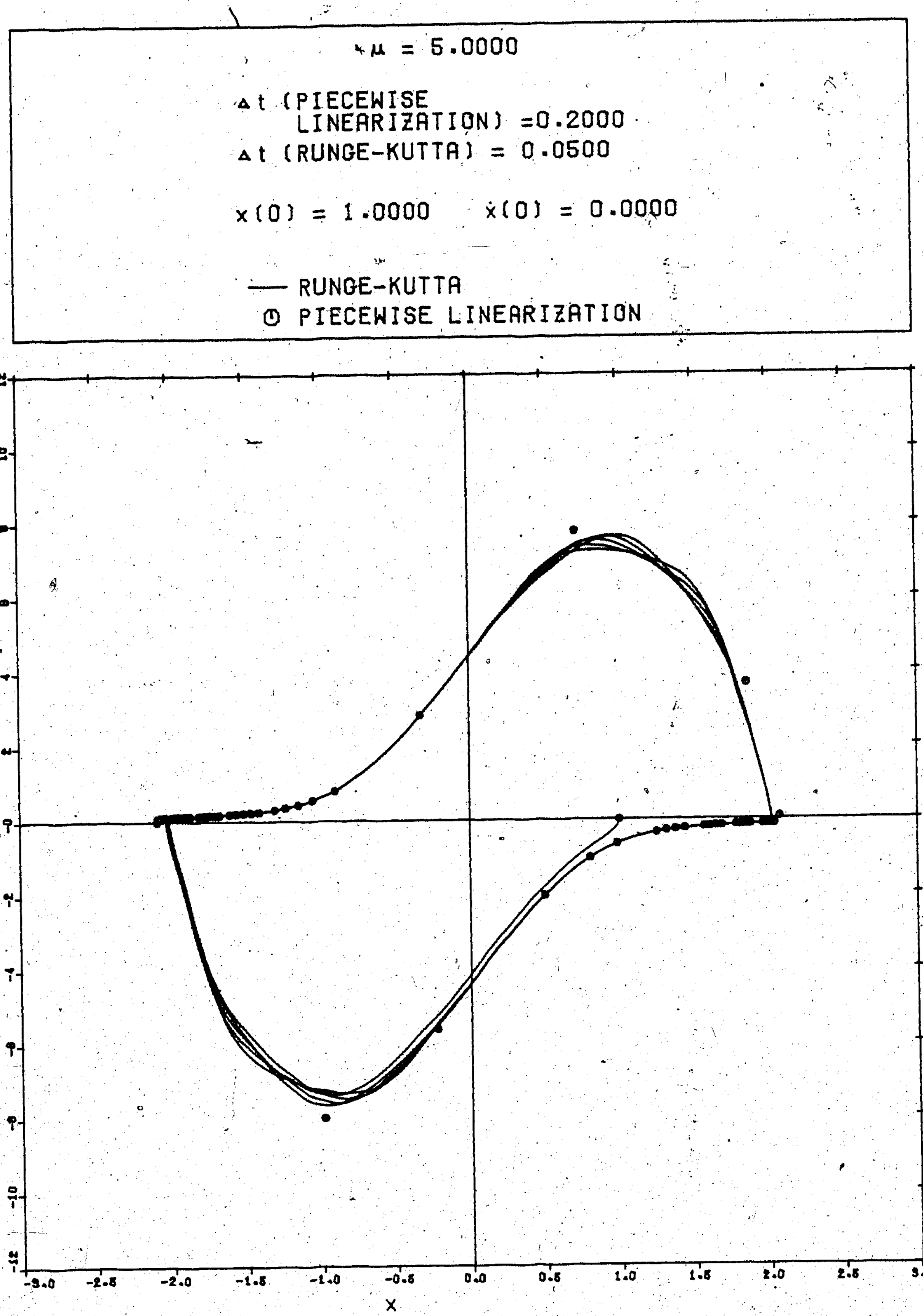

*μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 1.0000   ẋ(0) = 0.0000
— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
-3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0
X
Ẋ

FIGURE 5.15 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

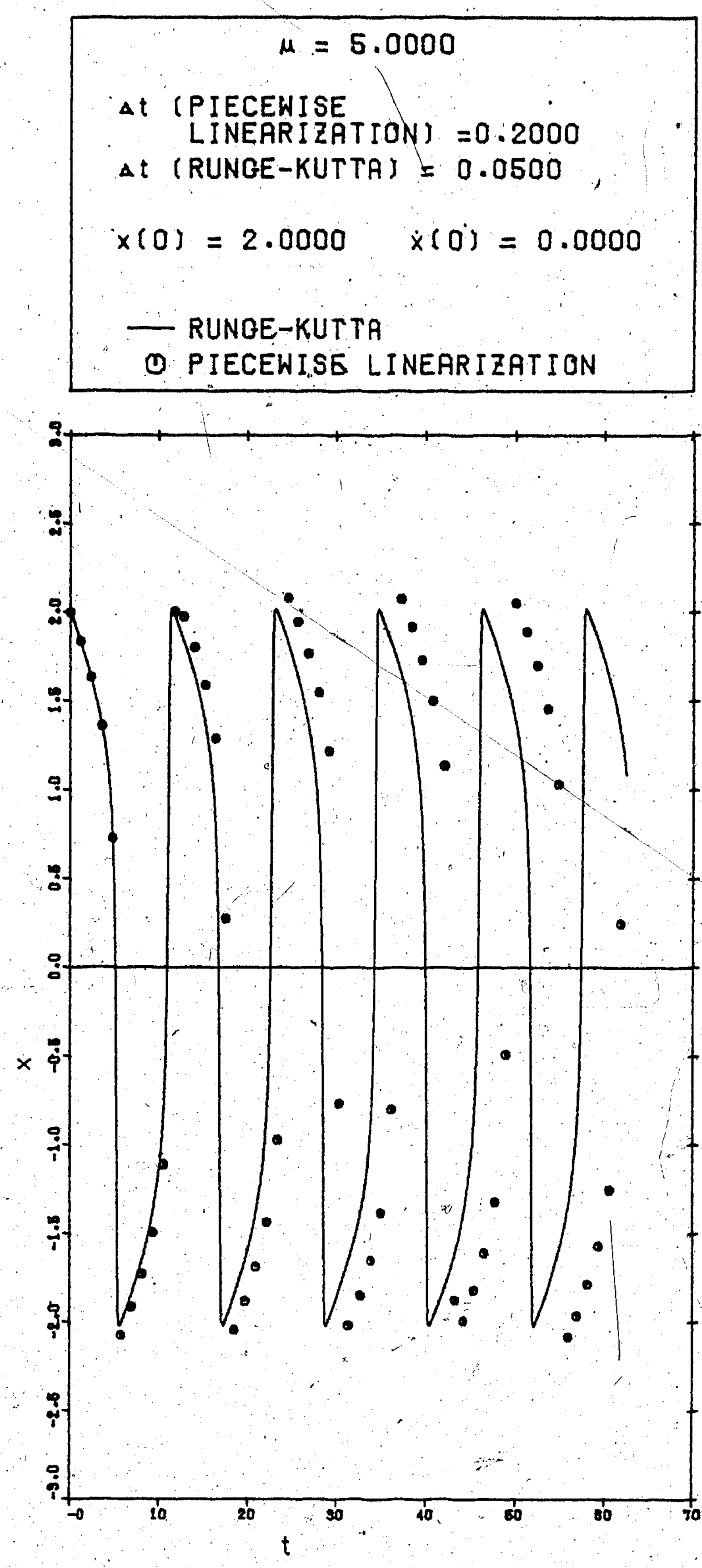
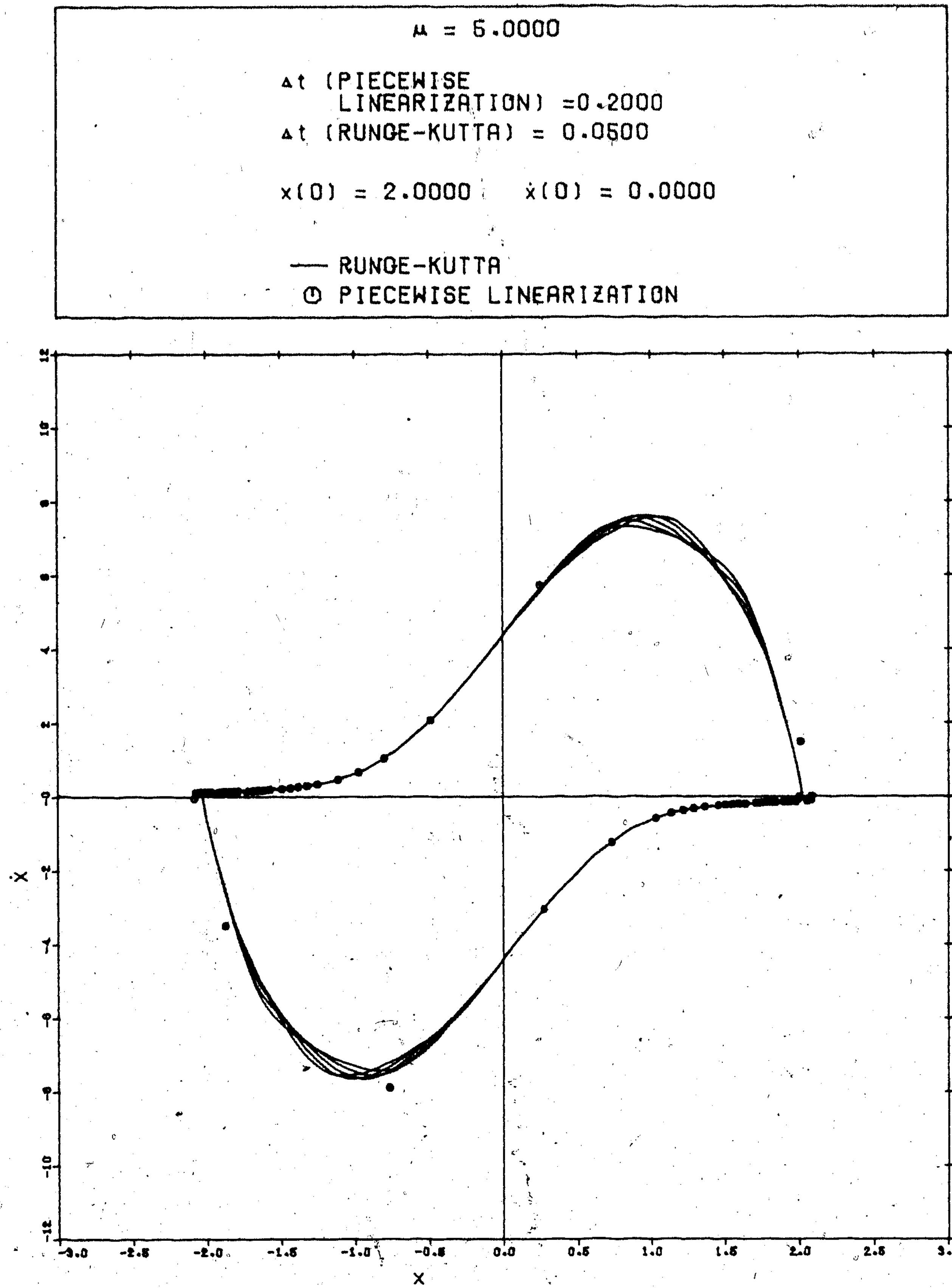

μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.0000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
x
t
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-0
10
20
30
40
50
60
70

FIGURE 5.16 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.0$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)



μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.0000   ẋ(0) = 0.0000
—— RUNGE-KUTTA
⊙ PIECEWISE LINEARIZATION
ẋ
X

FIGURE 5.17 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

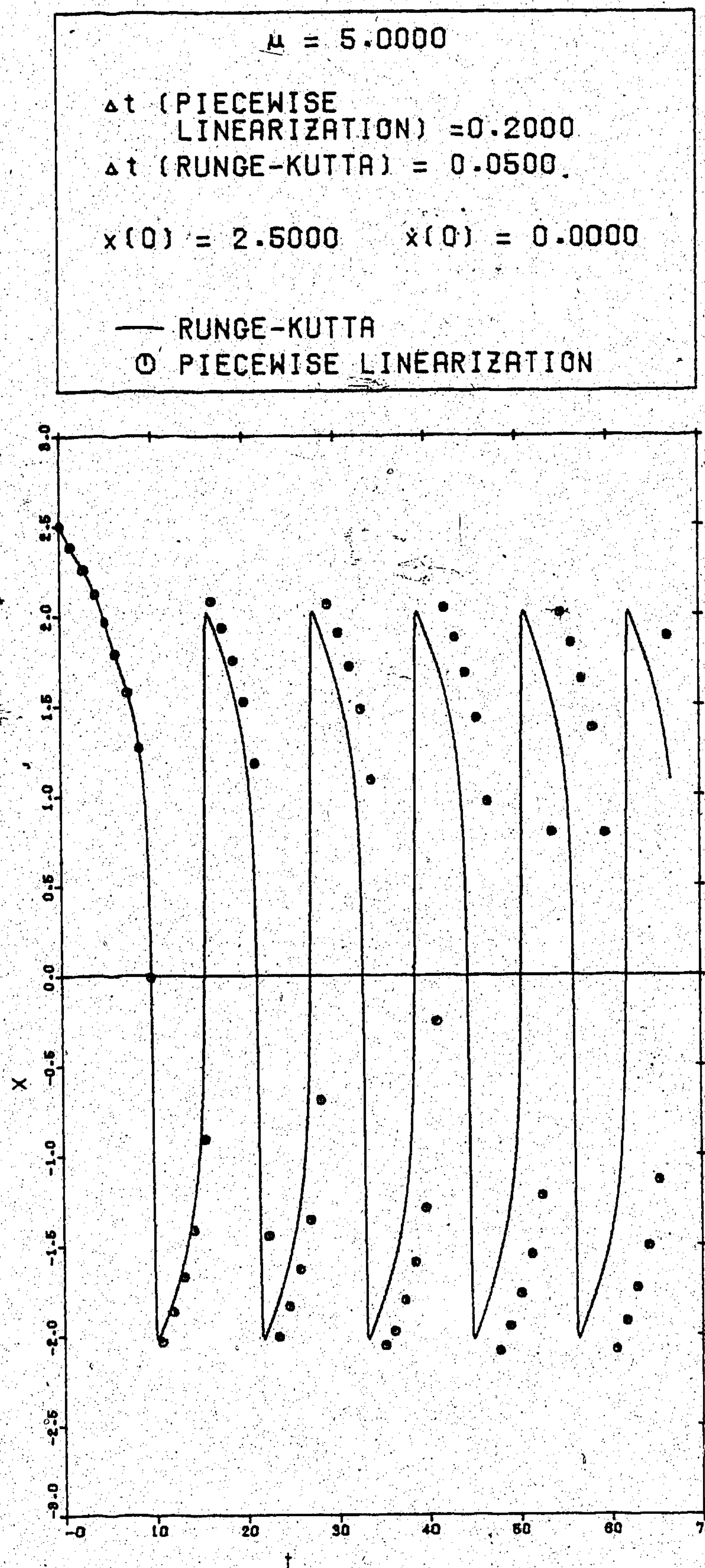

μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
X
t
-3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0
0 10 20 30 40 50 60 70

FIGURE 5.18 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.20$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)

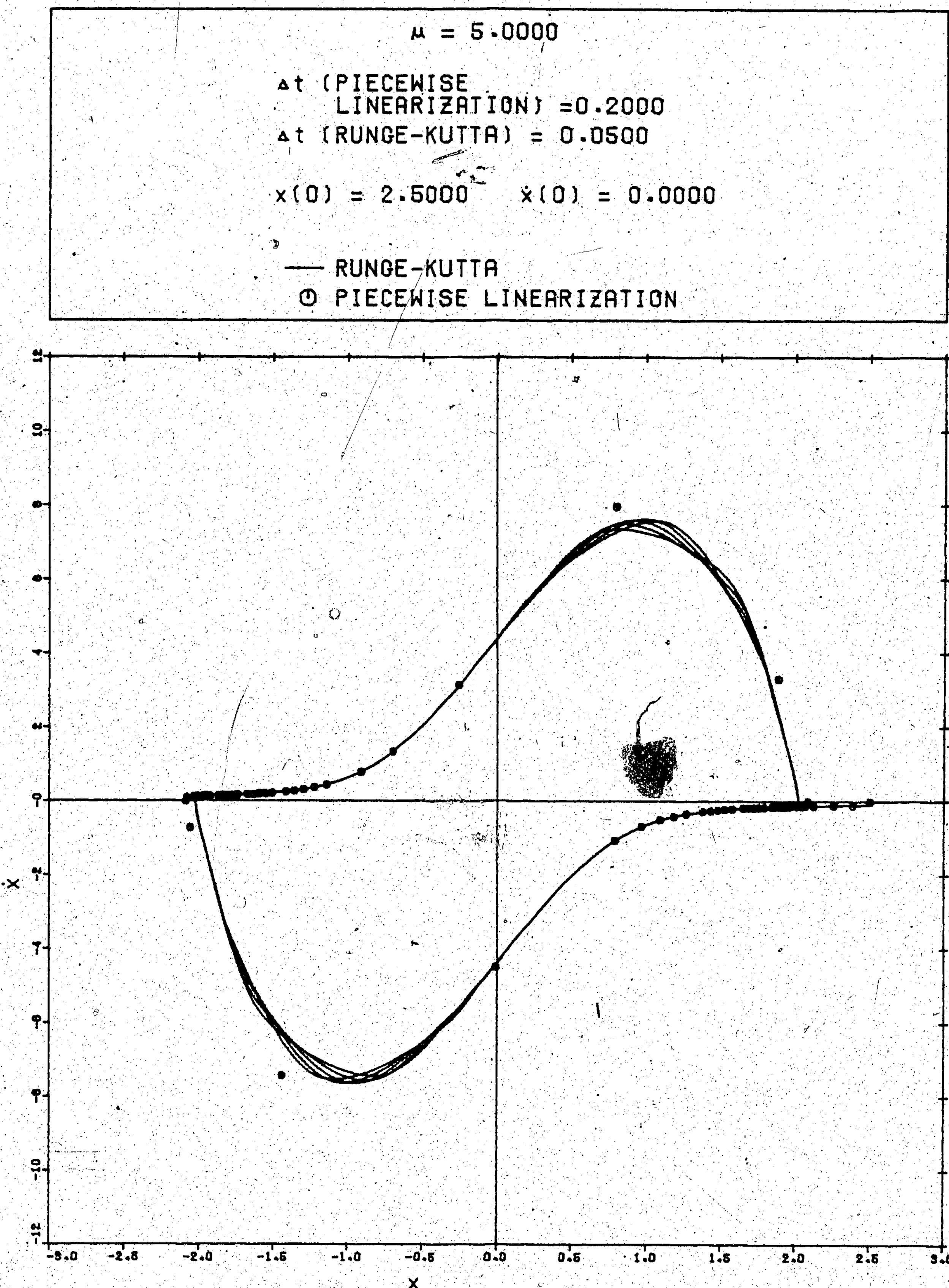

μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.2000
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000 ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
ẋ
X
-3.0
-2.5
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
12
10
8
6
4
2
0
-2
-4
-6
-8
-10
-12

FIGURE 5.19 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.05$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

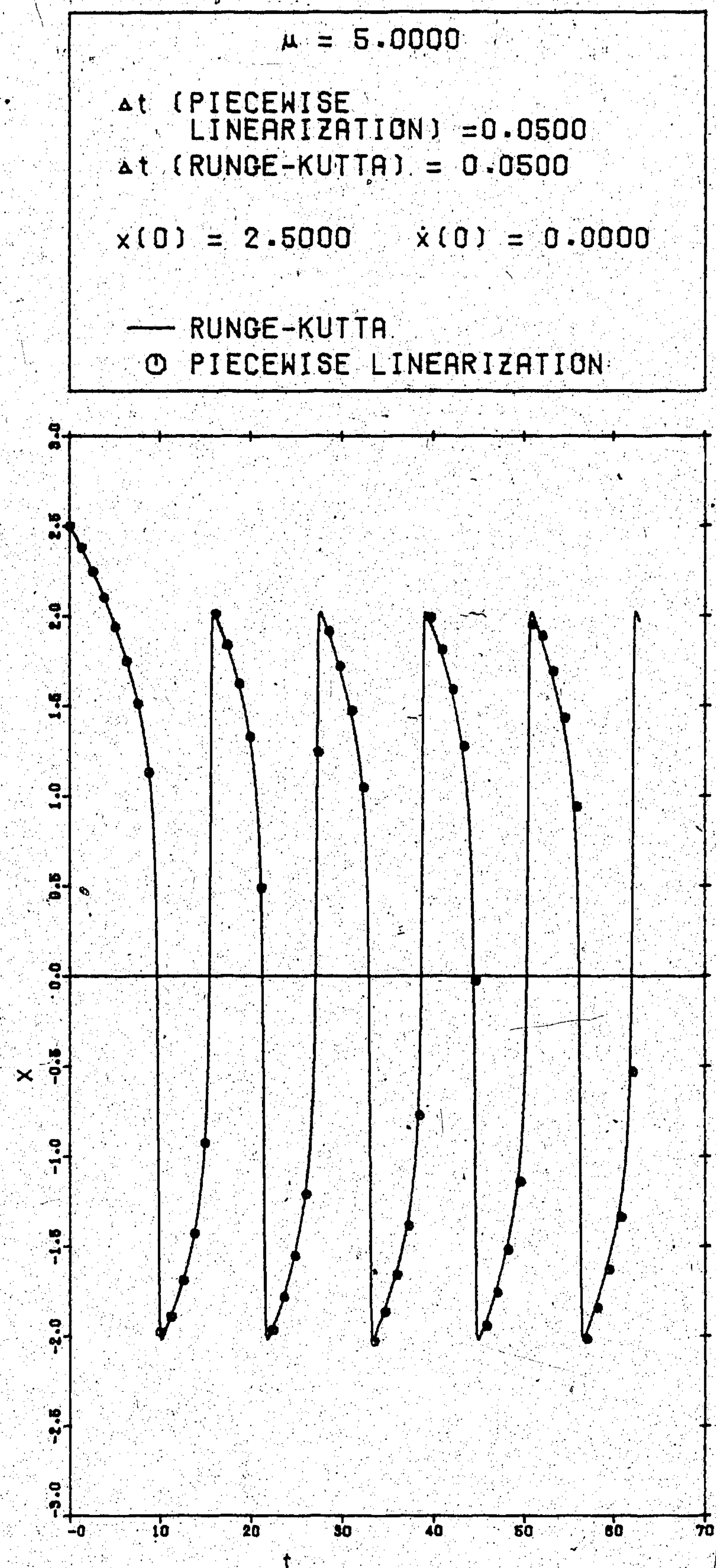

μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.0500
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
X
t

FIGURE 5.20.- PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.05$, $\Delta t_{RK} = 0.05$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)

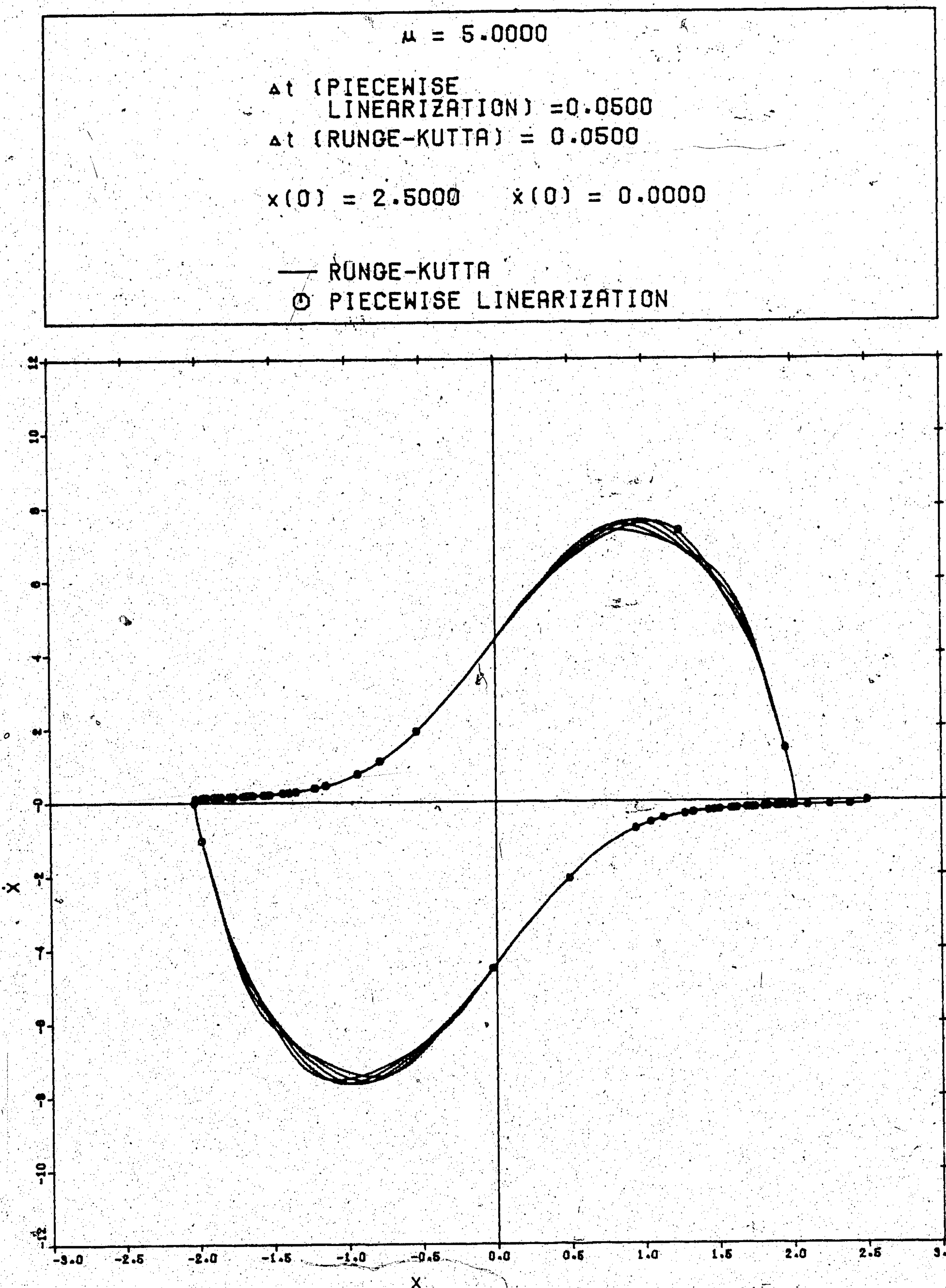

μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.0500
Δt (RUNGE-KUTTA) = 0.0500
x(0) = 2.5000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
12
10
8
6
4
2
0
-2
-4
-6
-8
-10
-12
ẋ
-3.0
-2.5
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X

FIGURE 5.21 - SOLUTIONS TO VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.25$, $\Delta t_{RK} = 0.25$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [38].)

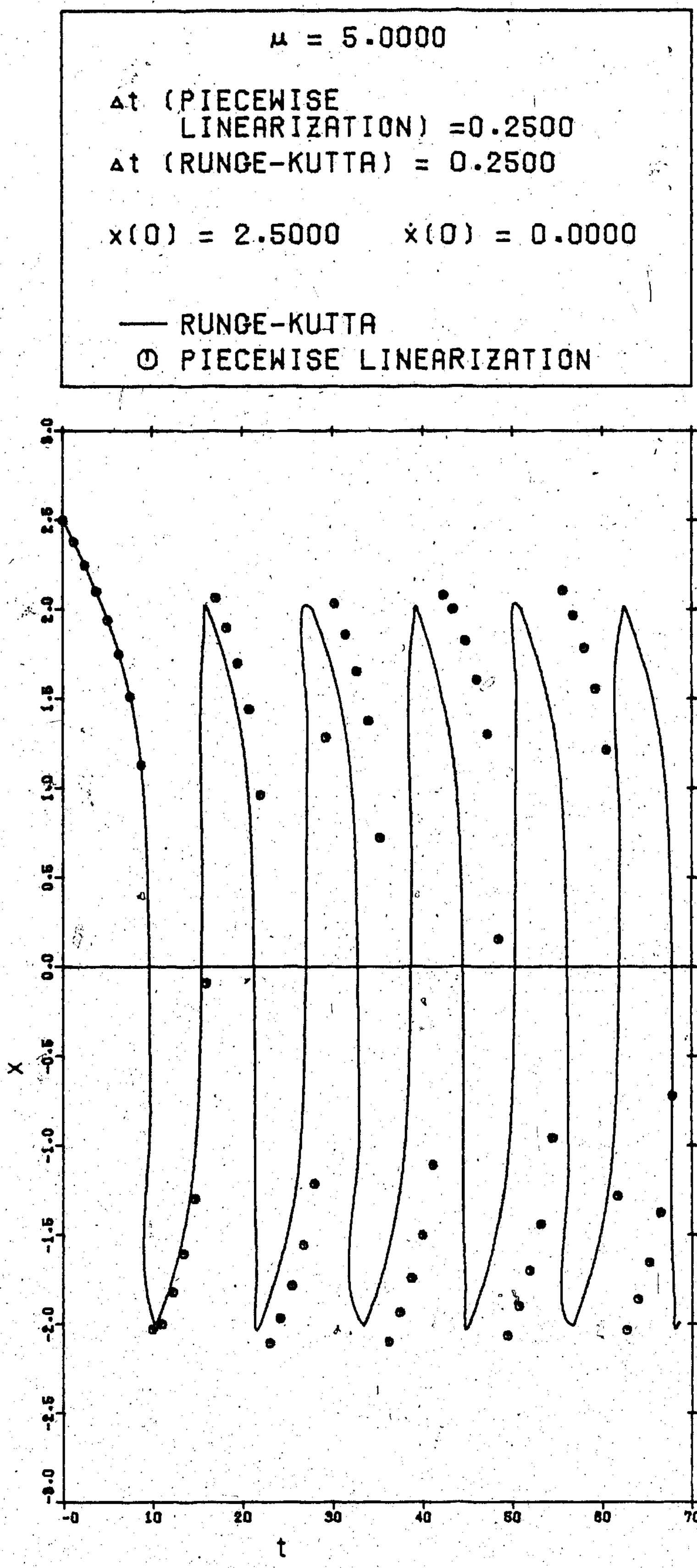

μ = 5.0000
Δt (PIECEWISE LINEARIZATION) =0.2500
Δt (RUNGE-KUTTA) = 0.2500
x(0) = 2.5000   ẋ(0) = 0.0000
RUNGE-KUTTA
PIECEWISE LINEARIZATION
X
t

FIGURE 5.22 - PHASE TRAJECTORIES FOR VAN DER POL EQUATION FOR $\mu = 5.0$, $\Delta t_P = 0.25$, $\Delta t_{RK} = 0.25$, $x(0) = 2.5$, $\dot{x}(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5]. Diagram based on [39].)

μ = 5.0000

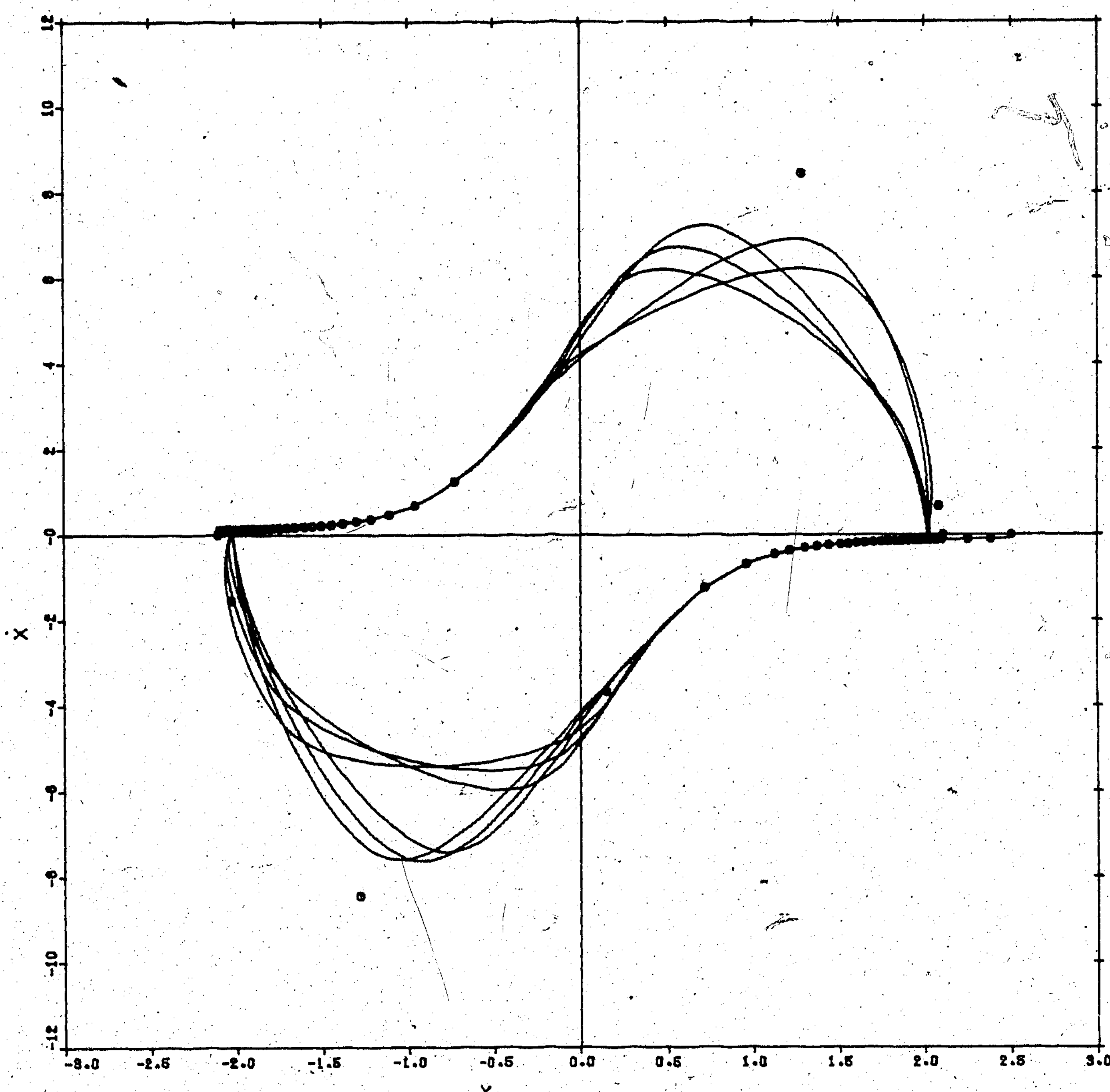
Δt (PIECEWISE LINEARIZATION) =0.2500

Δt (RUNGE-KUTTA) = 0.2500

$x(0)$ = 2.5000 $\dot{x}(0)$ = 0.0000

—— RUNGE-KUTTA

⊙ PIECEWISE LINEARIZATION



problem, since the situation appeared to be rectified to some extent in these diagrams, and completely resolved for $\mu = 2.0$ over the time span examined though the plots in question are not shown here.

This raises some interesting questions concerning the solution of this equation. One is concerned with stability, but this will be examined later as a similar problem occurs in the next chapter. Another is that if the piecewise linear method is affected by this behaviour, then the Runge-Kutta solution [5] as used here may, at times, also be suspect.

An example of the latter can be seen in Figures 5.21 and 5.22 [38], [39]. The time interval for each solution is rather large (keeping μ at 5.0), and some distortion can be seen in the displacement-time curve, while the phase plane diagram is very much affected by this to the extent that one cannot tell which part of the curve is the limit cycle.

Figure 5.21 [38] displays a paradox, as there would appear to be more than one value for the displacement as calculated by DVERK [5] for a given time (such as for t of about 23, where the value of x would be about 2.1, 1.6, and -0.9). This is physically impossible, and can lead to an erroneous conclusion about the results.

A possible source may be that the graphics routine used [40] was unable to produce a smooth curve due to a low number of data points being plotted since Δt was large. (By comparison, the curve for Figure 5.19 [38] was more accurate since a much larger number of values had been used.) The examination of the calculated numerical values may give further insight as to what may have brought this about.

Of interest now are the execution times for the two solutions. As before, the program was stripped of all non-essential steps, such as comment statements and write commands and then run for various values of μ, Δt_p, and Δt_{RK}, with $x(0) = 3.0$ and $\dot{x}(0) = 0.0$ using [27]. The runs were for the time taken to complete 10 half-cycles for the piecewise linearization and the closest time to that value for the Runge-Kutta solution [5]. These were obtained by running an earlier version of the program on the Amdahl 470V/7 [25], with the results seen in Figures 5.23 - 5.26. As expected for this system, the Runge-Kutta [5] results were significantly lower than those for the piecewise linearization, but noteworthy is the fact that the times go up for the piecewise linear method after $\Delta t_p = 0.25$ for $\mu = 2.0$ and $\mu = 5.0$. The calculations for the damping coefficient described in the derivation may explain why this arises, because of the number of iterations required to achieve a solution. Also, for these values of μ, no results were to be obtained for $\Delta t_p = 0.5$, as the program kept exceeding its set time limit, even after a setting of 15.0 seconds [31]. It was decided to abandon any further attempts as the runs would have proved too costly, but this might indicate that an economic upper limit may exist for these values of μ.

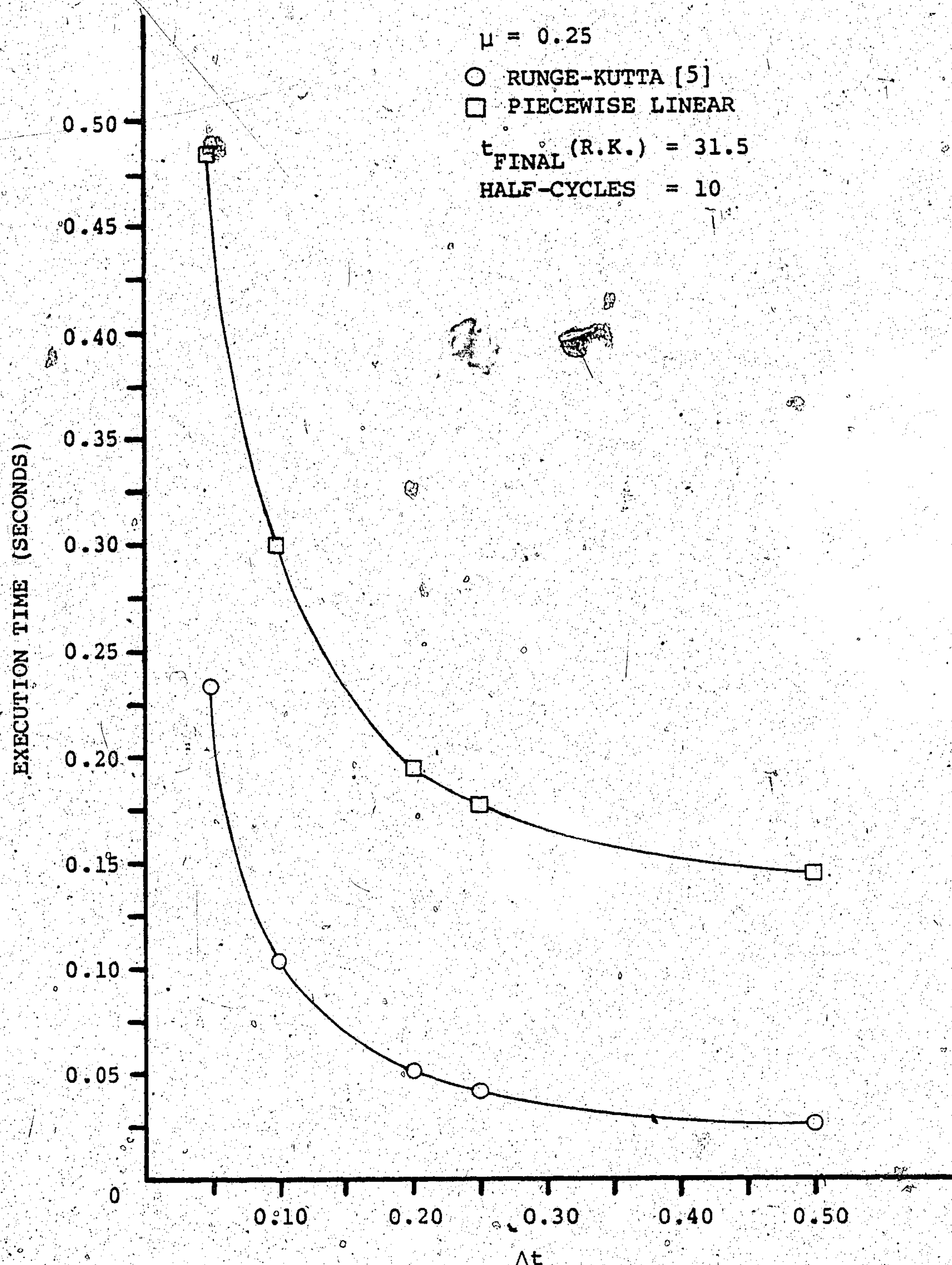


FIGURE 5.23 - EXECUTION TIMES FOR VAN DER POL EQUATION SOLUTIONS FOR $\mu = 0.25$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$ [27]

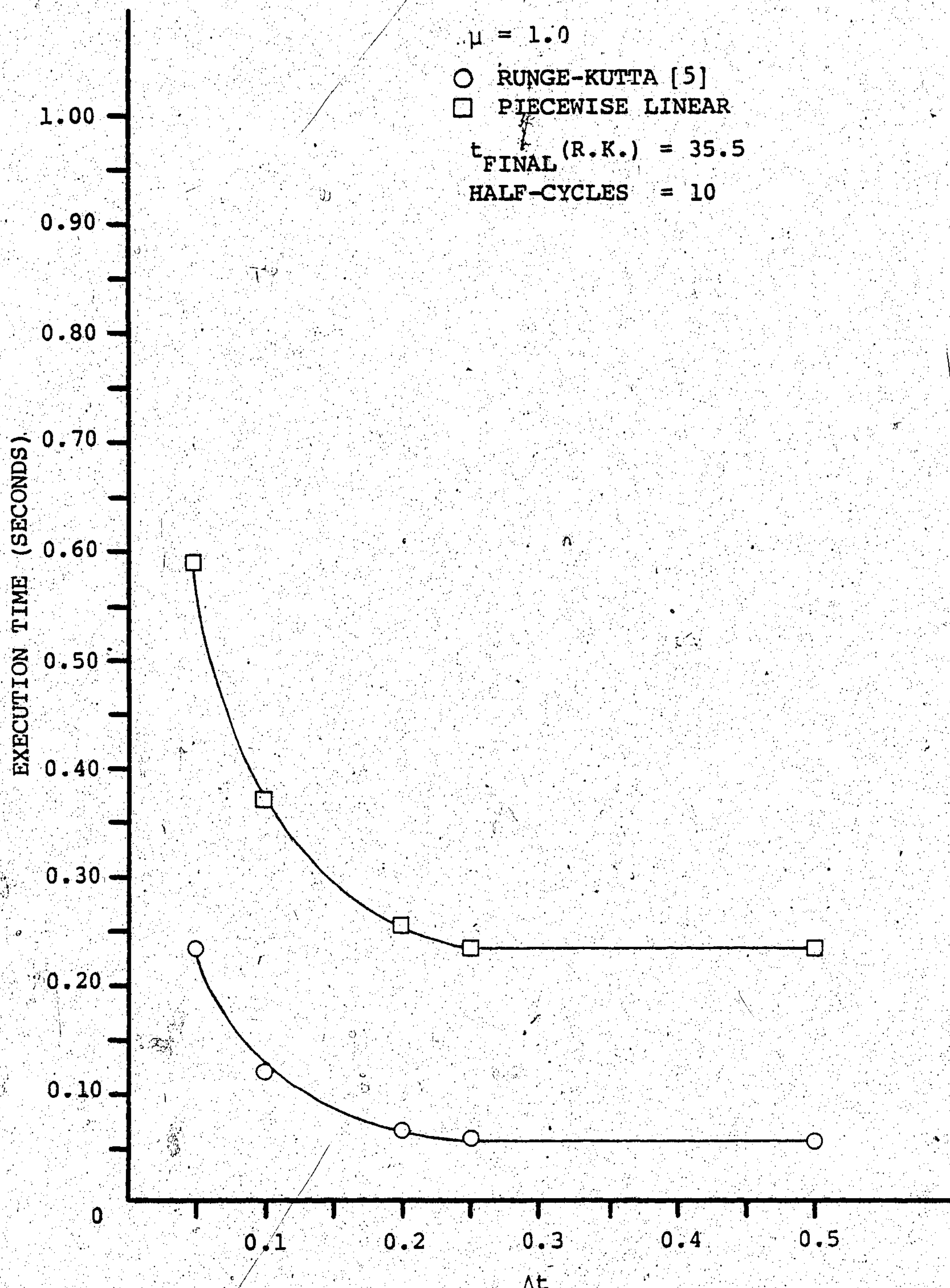


FIGURE 5.24 - EXECUTION TIMES FOR VAN DER POL EQUATION SOLUTIONS FOR $\mu = 1.0$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$ [27]

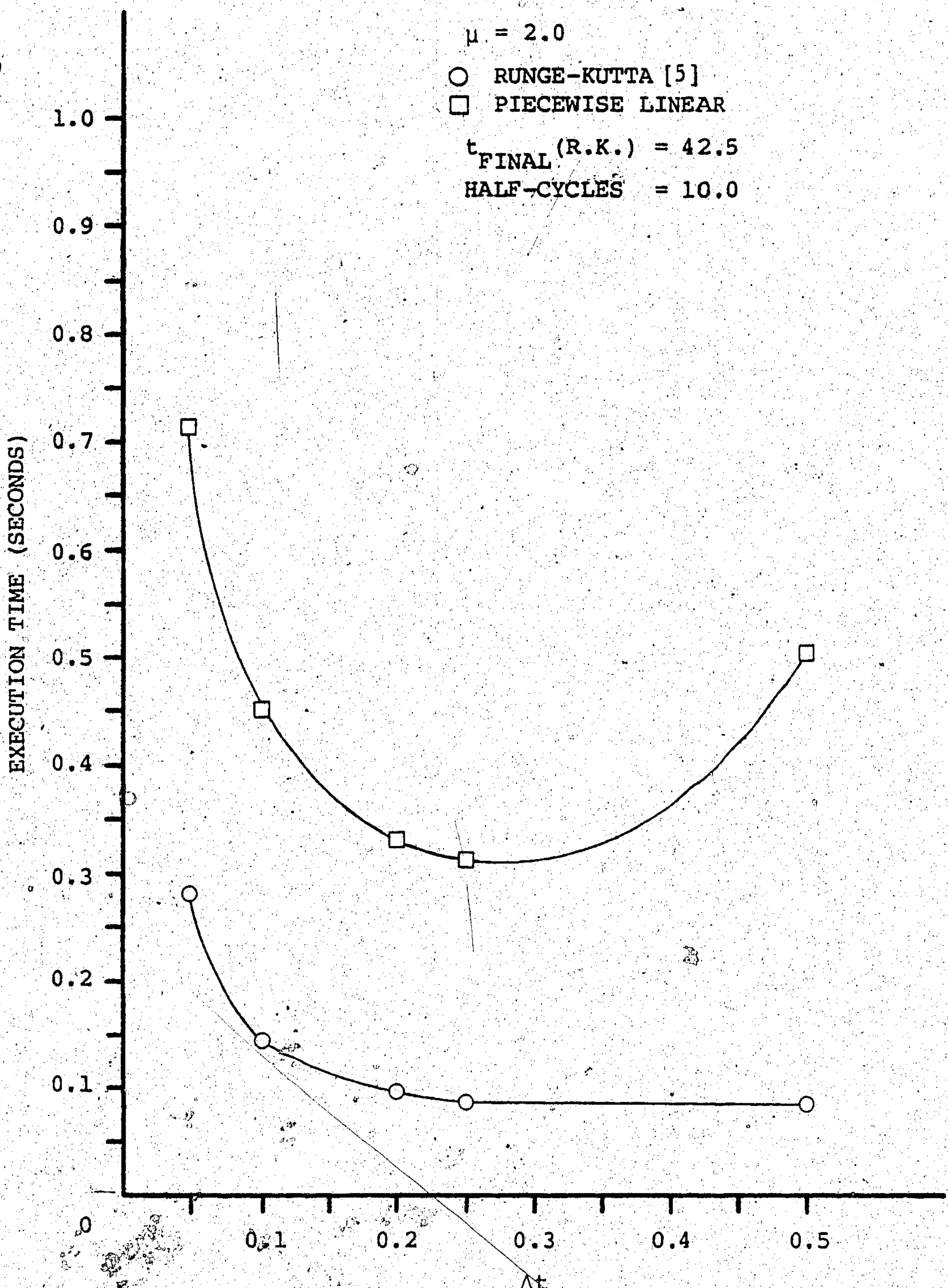


FIGURE 5.25 - EXECUTION TIMES FOR VAN DER POL EQUATION SOLUTIONS FOR $\mu = 2.0$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$ [27].

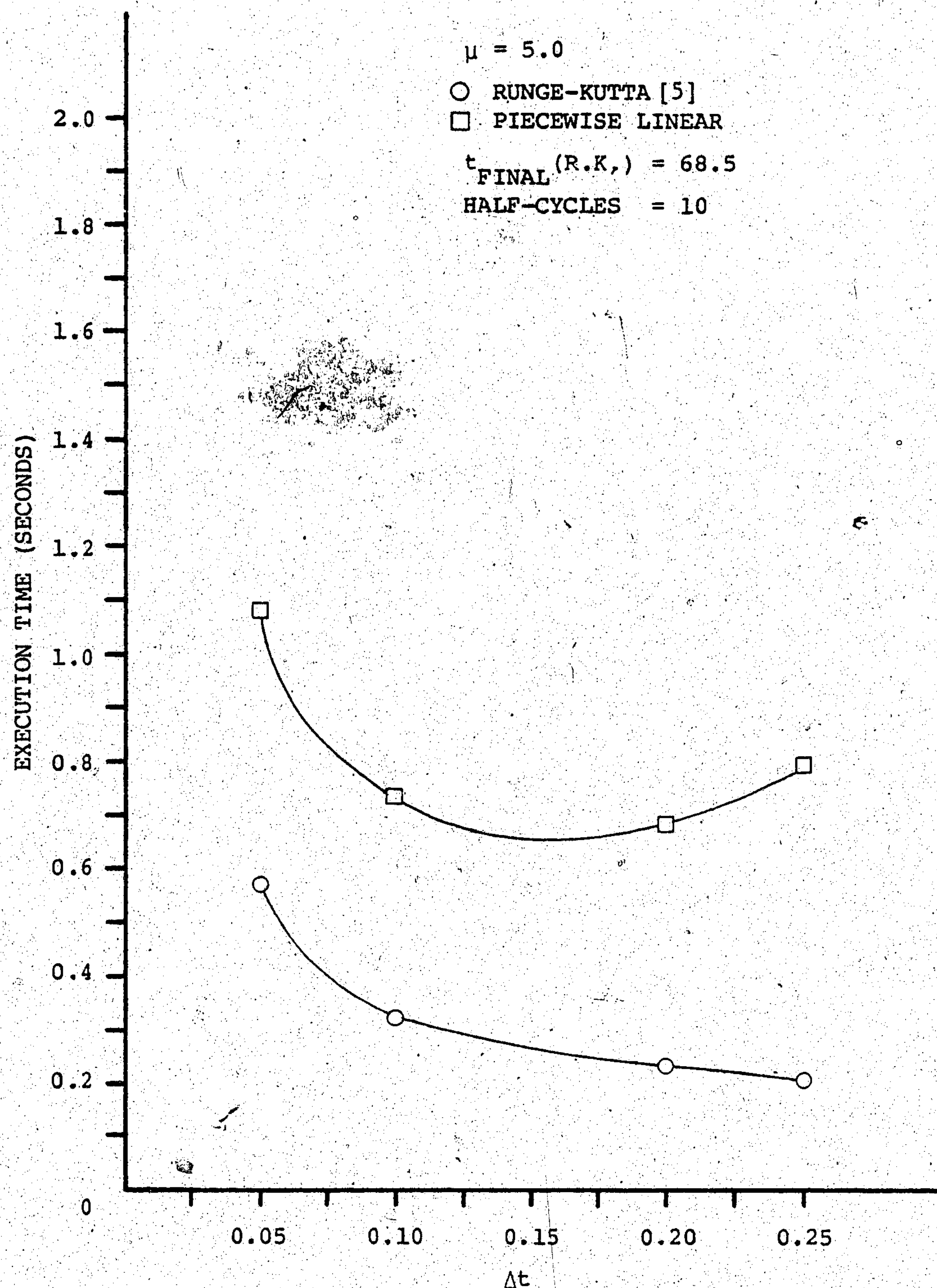


FIGURE 5.26 - EXECUTION TIMES FOR VAN DER POL EQUATION SOLUTIONS FOR $\mu = 5.0$, $x(0) = 3.0$, $\dot{x}(0) = 0.0$ [27]

VI. MATHIEU EQUATION

A. Preliminary Comments

In this chapter, an example of a linear equation with a periodically varying coefficient, namely the Mathieu equation, is examined. It has the canonical form [41]:

$$y'' + (a - 2q\cos 2z)y = 0 \,, \qquad (6.1.1)$$

a = Characteristic number.

The initial conditions considered are:

$$y(0) = Y_0, \qquad y'(0) = Y'_0 \,. \qquad (6.1.2)$$

Fortunately, an exact solution exists in terms of some special functions, and a means of generating a series approximation to them will be derived. This will be used as the basis by which to judge the piecewise linearization. The same Runge-Kutta routine [5] that had been used in the earlier chapters will also be used as confirmation of the results. The investigation will be limited to those cases of integral order.

B. Generation of Characteristic Number

The major references for this chapter are Ince [42] and McLachlan [43].

The appropriate characteristic number depends upon the initial conditions from which the solution is generated, the order of the solution (which determines the period τ) and is summarized as [44]:

		$\tau = \pi$	$\tau = 2\pi$
$y(0) = Y_0$,	$y'(0) = 0$	a_{2n}	a_{2n+1}
$y(0) = 0$,	$y'(0) = Y'_0$	b_{2n+2}	b_{2n+1}

One method for this makes use of the recurrence relations for the coefficients of the series form of the exact solutions [45] which are shown below.

For a_{2n}:

$$ce_{2n}(z,q) = \sum_{r=0}^{\infty} A_{2r}^{(2n)} \cos(2rz) \quad . \tag{6.2.1}$$

For a_{2n+1}:

$$ce_{2n+1}(z,q) = \sum_{r=0}^{\infty} A_{2r+1}^{(2n+1)} \cos(2r+1)z \quad . \tag{6.2.2}$$

For b_{2n+1}:

$$se_{2n+1}(z,q) = \sum_{r=0}^{\infty} B_{2r+1}^{(2n+1)} \sin(2r+1)z \quad . \tag{6.2.3}$$

For b_{2n+2}:

$$se_{2n+2}(z,q) = \sum_{r=0}^{\infty} B_{2r+2}^{(2n+2)} \sin(2r+2)z \quad . \tag{6.2.4}$$

The coefficients for the equations are calculated using the recurrence relations described in McLachlan [46].

For $ce_{2n}(z,q)$:

$$\left.\begin{aligned}
&aA_0 - qA_2 = 0,\\
&(a-4)A_2 - q(A_4 + 2A_0) = 0,\\
&(a-4r^2)A_{2r} - q(A_{2r+2} + A_{2r-2}) = 0,\\
&\qquad\qquad (r \geq 2).
\end{aligned}\right\} \quad (6.2.5)$$

For $ce_{2n+1}(z,q)$:

$$\left.\begin{aligned}
&(a-1-q)A_1 - qA_3 = 0\\
&\left[a-(2r+1)^2\right]A_{2r+1}\\
&- q(A_{2r+3} + A_{2r-1}) = 0,\\
&\qquad\qquad (r \geq 1).
\end{aligned}\right\} \quad (6.2.6)$$

For $se_{2n+1}(z,q)$:

$$\left.\begin{aligned}
&(b-1+q)B_1 - qB_3 = 0,\\
&\left[b-(2r+1)^2\right]B_{2r+1}\\
&- q(B_{2r+3} + B_{2r-1}) = 0,\\
&\qquad\qquad (r \geq 1).
\end{aligned}\right\} \quad (6.2.7)$$

For $se_{2n+2}(z,q)$:

$$\left.\begin{aligned}
&(b-4)B_2 - qB_4 = 0,\\
&(b-4r^2)B_{2r}\\
&- q(B_{2r+2} + B_{2r-2}) = 0,\\
&\qquad\qquad (r \geq 2).
\end{aligned}\right\} \quad (6.2.8)$$

With Ince's notation [39] for the continued fraction:

$$\mathop{\Phi}_{r=1}^{\infty} \frac{\alpha_\nu}{\beta_\nu} = \frac{1}{\beta_r} + \frac{\alpha_{r+1}}{\beta_{r+1}} + \frac{\alpha_{r+2}}{\beta_{r+2}} + \ldots .$$

and the rearrangement of some of the terms, the above relations lead to the following. For $ce_{2n}(z,q)$, the result is [47]:

$$\frac{A_{2r}}{A_{2r-2}} = -\frac{q}{4r^2} \mathop{\Phi}_{r}^{\infty} \frac{-\dfrac{q^2}{16\nu^2(\nu-1)^2}}{1-\dfrac{a}{4\nu^2}},$$

which yields the continued fraction [48]:

$$a_{2n} = -\frac{q^2/2}{1-\frac{a}{4}} - \frac{q^2/64}{1-\frac{a}{16}} - \frac{q^2/576}{1-\frac{a}{36}} - \ldots \qquad (6.2.9)$$

For $se_{2n+1}(z,q)$, the continued fraction is [49]:

$$\frac{B_{2r-1}}{B_{2r+1}} = \frac{-q}{(2r+1)^2} \mathop{\Phi}_{r}^{\infty} \frac{-\dfrac{q^2}{(4\nu^2-1)^2}}{1-\dfrac{b}{(2\nu+1)^2}},$$

which yields the continued fraction for the characteristic number [49]:

$$b_{2n+1} = 1 - q - \frac{q^2/9}{1-\frac{b}{9}} - \frac{q^2/225}{1-\frac{b}{25}} - \frac{q^2/1225}{1-\frac{b}{49}} - \ldots \qquad (6.2.10)$$

The case of $ce_{2n+1}(z,q)$ is similar to the previous function since the continued fraction for the coefficients is the same, but with a change of sign in q, yielding the value for the characteristic number [49]:

$$a_{2n+1}(q) = b_{2n+1}(-q) . \qquad (6.2.11)$$

Finally, for $se_{2n+2}(z,q)$ the continued fraction is [49]:

$$\frac{B_{2r}}{B_{2r-2}} = -\frac{q}{4r^2} \overset{\infty}{\underset{r}{\Phi}} \frac{\dfrac{q^2}{16\nu^2(\nu-1)^2}}{1-\dfrac{b}{4\nu^2}} ,$$

and the characteristic number continued fraction is [49]:

$$b_{2n+2} = 4 - \frac{q^2/16}{1-\frac{b}{16}} - \frac{q^2/576}{1-\frac{b}{36}} - \frac{q^2/2304}{1-\frac{b}{64}} - \ldots . \qquad (6.2.12)$$

The continued fractions have terms of the form [48]:

$$\gamma_n = \frac{f(a,q,n)}{\phi_{n+1}} ,$$

$$\phi_{n+1} = f(a,q,n) - \gamma_{n+1} .$$

Calculation is halted when, for the accuracy required, $\phi_{n+1} = 1$. It was decided that 25 terms would be sufficient.

A characteristic number can now be generated using the continued fractions just given together with a suggested approach given by Ince [48]. He states how someone doing this task using a calculator, paper and pencil could arrive at a final result. The author took this method and first went through the procedure by hand for tabulated values for the characteristic number and flowcharting the decision processes required, and then testing this method on a Hewlett-Packard HP-67 hand calculator, transferring the final version of the logic to a computer program, which was later run on the Amdahl 470V/8 [26]. The results were compared with the tabulated results given by Ince [50] and there was agreement to seven or eight significant figures.

The following method can be considered as consisting of two parts. The first one (Steps 1-3) is a scan through a range of values for the characteristic number, for an estimated value for a (a_{GUESS}) and a given q, until an upper and a lower bound, between which the final value will lie, are found. The second one (Steps 4-6) is an interval halving search for this final value. The particular continued fraction used is dependent upon the initial conditions in effect. If at any time during the scan a_{IN} is approximately equal to a_{OUT}, the process stops.

1. A value for a_{OUT} is found using a_{IN}.

2. If a_{IN} is greater than a_{OUT}, assume a_{IN} to be high, decrement it and repeat Step 1 until either a_{IN} is less than a_{OUT}, or the sign of the present a_{OUT} is different than that for the previous a_{OUT} and the present and previous a_{IN}. The upper bound (a_{UP}) is the next-to-last a_{IN} and the lower bound (a_{LOW}) the final one.

3. If a_{IN} is less than a_{OUT}, assume a_{IN} to be low, increment it and repeat Step 1 until a_{IN} is greater than a_{OUT}. The upper bound is the final a_{IN} and the lower one is the previous value.

4. The midpoint a_{MID} is calculated from the average of the upper and lower bounds, and an output value a_{MIDOUT} is found. If a_{MID} is nearly equal to a_{MIDOUT}, a_{MID} is the final value for a. If not, continue below.

5. Consider the case of a positive a_{MID}. If a_{MIDOUT} is less than or equal to zero, a_{LOW} is the present a_{MID}, and Step 4 is repeated. If a_{MIDOUT} is greater than zero and a_{MID} is less than or equal to a_{MIDOUT}, the present a_{MID} becomes a_{LOW} and Step 4 is done again. However, should a_{MIDOUT} be greater than zero, and a_{MID} be greater than a_{MIDOUT}, a_{MID} becomes a_{UP} and the process returns to Step 4.

6. The following applies to a_{MID} less than or equal to zero. If a_{MIDOUT} is greater than zero, a_{MID} becomes the new a_{LOW}, and the next move is to Step 4. For a_{MIDOUT} less than or equal to zero and a_{MID} greater than a_{MIDOUT}, the new a_{UP} is the present a_{MID}, and then to Step 4. If a_{MIDOUT} is less than or equal to zero, and a_{MID} is less than or equal to a_{MIDOUT}, a_{LOW} takes on the value of the present a_{MID} and Step 4 is repeated.

This covers the method for generating a value for a characteristic number. Specifically, the particular calculation chosen depends upon the initial conditions and the proximity of the first guess to the final value.

To elaborate, for a non-zero initial displacement and zero initial velocity, a_{2n} and a_{2n+1} would be the appropriate values, with b_{2n+1} and b_{2b+2} being in effect for zero initial displacement and non-zero initial velocity. The program that was written for this purpose calculated the odd order value first (this choice having been arbitrary) and then the even order one was found. Whichever was closest to a_{GUESS} became the final result. This was then taken to generate the three solutions used.

C. Series Approximation to Exact Solution

As mentioned earlier, Mathieu functions can be approximated by series solutions [45], which were given as (6.2.1) to (6.2.4). The question now is how to generate the coefficients that were given. According to McLachlan [46], [51], the recurrence relations that were given in (6.2.5) to (6.2.8) can be used. However, a more suitable form for these relations must be obtained in order to facilitate calculation of the coefficients.

Consider $ce_{2n}(z,q)$ [52]. The first relation of (6.2.5) gives:

$$\frac{A_2}{A_0} = \frac{a}{q} .$$

The next one yields:

$$\frac{A_4}{A_2} = \frac{a - 4}{q} - \frac{2}{A_2/A_0} .$$

The final one gives:

$$\frac{A_{2r}}{A_{2r-2}} = \frac{q}{a - 4r^2 - q\left(\dfrac{A_{2r+2}}{A_{2r}}\right)} , \qquad (r \geq 3) .$$

It had been decided to terminate calculation after 25 terms, since ratios beyond A_{52}/A_{50} would become so small that they could be neglected when compared with the other terms [53]. Once the various ratios have been found, the normalization [54]:

$$1 = 2A_0^2 + A_2^2 + A_4^2 + \ldots\ ,$$

$$\frac{1}{A_0^2} = 2 + \left(\frac{A_2}{A_0}\right)^2 + \left(\frac{A_4}{A_0}\right)^2 + \ldots$$

is required, and so the values for A_{2r}/A_0 are needed. This is done by using:

$$\frac{A_{2r}}{A_0} = \frac{A_{2r}}{A_{2r-2}} \cdot \frac{A_{2r-2}}{A_0}\ ,$$

starting with A_2/A_0 and A_4/A_2, and working sequentially through all the ratios. Once this has been accomplished, using the different values for A_{2r}/A_0, A_0 and A_{2r} can be found.

With the various A_{2r}'s, a series approximation for $ce_{2n}(z,q)$ can be generated.

Take now $ce_{2n+1}(z,q)$ [55]. The first ratio from (6.2.6) gives:

$$\frac{A_3}{A_1} = \frac{a - 1 - q}{q}\ .$$

The remaining ratios are of the form:

$$\frac{A_{2r+1}}{A_{2r-1}} = \frac{q}{a - (2r + 1)^2 - q\left[\dfrac{A_{2r+3}}{A_{2r+1}}\right]}\ ,$$

$$(r \geq 2).$$

As was the case for the previous function, calculation was halted after 25 terms because ratios beyond A_{51}/A_{49} can be considered as being negligible when compared with the others. The resulting normalization used is [56]:

$$1 = A_1^2 + A_3^2 + A_5^2 + \ldots ,$$

and the series approximation for $ce_{2n+1}(z,q)$ using the values for A_{2r+1} is found using the same method as before.

Consider now $se_{2n+1}(z,q)$. Owing to the similarities between (6.2.6) and (6.2.7), the derivations can be omitted and the results presented [49]:

$$\frac{B_3}{B_1} = \frac{b - 1 + q}{q} ,$$

$$\frac{B_{2r+1}}{B_{2r-1}} = \cfrac{q}{b - (2r + 1)^2 - q\left(\cfrac{B_{2r+3}}{B_{2r-1}}\right)} , \qquad (r \geq 2).$$

$$1 = B_1^2 + B_3^2 + B_5^2 + \ldots .$$

From this, $se_{2n+1}(z,q)$ can be found since the coefficients can be determined from the above relations.

Finally, the results for $se_{2n+2}(z,q)$ will be shown. Since (6.2.8) is similar to (6.2.5), the ratios are as follows, since the derivations would be alike [55], [56]:

$$\frac{B_4}{B_2} = \frac{b - 4}{q} ,$$

$$\frac{B_{2r}}{B_{2r-2}} = \frac{q}{b - 4r^2 - q\left(\frac{B_{2r+2}}{B_{2r}}\right)} ,$$

$$(r \geq 3) .$$

$$1 = B_2^2 + B_4^2 + B_6^2 + \ldots .$$

From this, an approximation to a Mathieu function of integral order can be generated, thus allowing one to consider the complete solution to the original equation. A computer program was written to make use of the relations just derived, and the results were checked against tabulated values that Ince presented [57], and were found to be in agreement to at least seven significant figures in almost all cases.

Using these coefficients, (6.2.1) to (6.2.4) can now be calculated, and thus, an approximation to the exact solution generated. For integral order solutions, one or the other of (6.1.2) is zero. For:

$$y(0) = Y_0 ,$$

$$y'(0) = 0 ,$$

the solution uses either (6.2.1) or (6.2.2), giving a complete result of:

$$y = \alpha \, ce_{\phi(r)}(z,q)$$

$$= \alpha \sum_{r=0}^{\infty} A_{\phi(r)} \cos\left[\phi(r) z\right] ,$$

where:

$$\phi(r) = 2r , \qquad (\tau = \pi)$$

$$= 2r+1 , \qquad (\tau = 2\pi)$$

Applying (6.1.2) yields the final form:

$$Y_0 = \alpha \sum_{r=0}^{\infty} A_{\phi(r)} \cos (0) ,$$

$$\alpha = \frac{Y_0}{\sum_{r=0}^{\infty} A_{\phi(r)}} ,$$

$$y = \frac{Y_0}{\sum_{r=0}^{\infty} A_{\phi(r)}} \sum_{r=0}^{\infty} A_{\phi(r)} \cos \left[\phi(r) z\right] .$$

For:

$$y(0) = 0 ,$$

$$y'(0) = Y_0' ,$$

the solution is of the form of either (6.2.3) or (6.2.4), thus yielding:

$$y = \beta se_{\phi(r)}(z,q)$$

$$= \beta \sum_{r=0}^{\infty} B_{\phi(r)} \sin \left[\phi(r) z\right] ,$$

with:

$$\phi(r) = 2r+2 , \qquad (\tau = \pi)$$

$$= 2r+1 , \qquad (\tau = 2\pi)$$

giving, after using (6.1.2):

$$y' = \beta \sum_{r=0}^{\infty} \{\phi(r) B_{\phi(r)} \cos[\phi(r) z]\} ,$$

$$Y'_0 = \beta \sum_{r=0}^{\infty} [\phi(r) B_{\phi(r)} \cos(0)] ,$$

$$\beta = \frac{Y'_0}{\sum_{r=0}^{\infty} \phi(r) B_{\phi(r)}} ,$$

$$y = \frac{Y'_0}{\sum_{r=0}^{\infty} \phi(r) B_{\phi(r)}} \sum_{r=0}^{\infty} B_{\phi(r)} \sin[\phi(r) z] .$$

D. Derivation of Piecewise Linear Solution

Since it is desired to have a piecewise linear solution of the form shown in (1.2), (6.2.1) can be approximated by:

$$y'' + p^2 y = 0$$

over a small interval, Δz. The displacement at the beginning of this interval is y_0 and the velocity y'_0.

This equation has the following solutions (based on [34]):

A. $p^2 > 0$ [58]:

$$y = y_0 \cos pz + \frac{y'_0}{p} \sin pz .$$

B. $p^2 = 0$ [37]:

$$y = y'_0 z + y_0 .$$

C. $p^2 < 0$ [37]:

$$y = y_0 \cosh sz + \frac{y_0'}{s} \sinh sz \ ,$$

$$p^2 = -s^2 \ .$$

The final value for the characteristic number, a, and q are given such that the solution would be periodic. Δz can be of any reasonable size.

Consider now the (m - 1)th and mth points. The displacements that would correspond to them are y_{m-1} and y_m, respectively. See Figure 6.1 for details. As the system moves through Δz, p^2 will change. It is better to have it as a constant value as it passes through an interval, and so, it is calculated at the midpoint of the interval, or at $(m - 1/2)\Delta z$. It is obvious that this will be the case as Δz gets smaller. That is to say, $a - 2q\cos[2(m-1)\Delta z]$ and $a - 2q\cos(2m\Delta z)$ will approach $a - 2q\cos[(2m-1)\Delta z]$.

So, for the mth point, the solutions will be as follows. For $a - 2q\cos[(2m-1)\Delta z]$ greater than zero [58]:

$$y_m = y_0 \cos p_m\Delta z + \frac{y_0'}{p_m} \sin p_m\Delta z \ ,$$

$$p_m = \left[a - 2q \cos (2m - 1)\Delta z\right]^{1/2} \ .$$

For $a - 2q\cos[(2m-1)\Delta z]$ equal to zero, the solution becomes [37]:

$$y_m = y_0'\Delta z + y_0 \ .$$

And finally, for $a - 2q\cos[(2m-1)\Delta z]$ less than zero [37]:

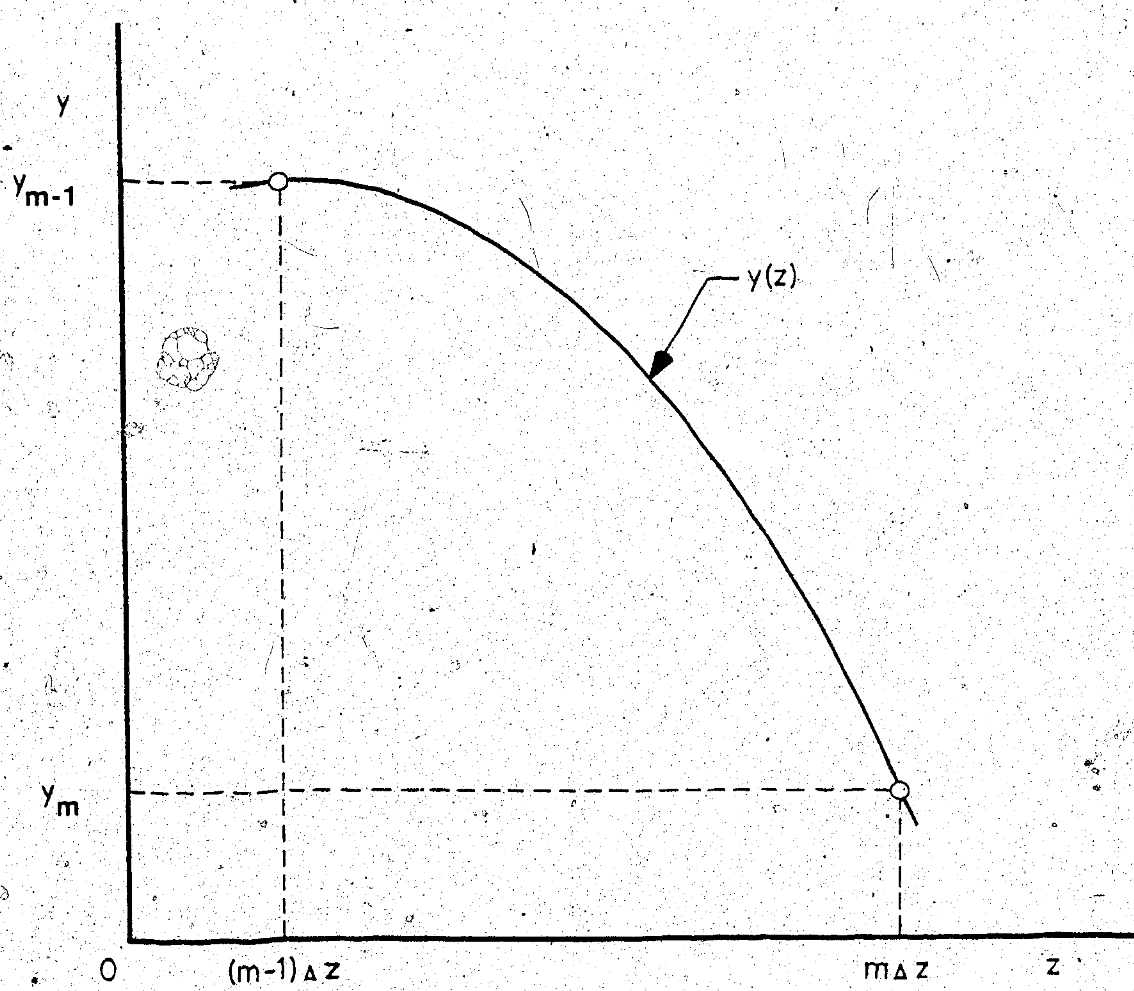


FIGURE 6.1 - TYPICAL INTERVAL LAYOUT (MATHIEU EQUATION)

$$y_m = y_0 \cosh s_m \Delta z + \frac{y_0'}{s_m} \sinh s_m \Delta z \ ,$$

$$s_m = \{-[a - 2q \cos (2m - 1)\Delta z]\}^{1/2} \ .$$

For each successive interval,

$$y_0 = y_{m-1} \ ,$$

$$y_0' = y_{m-1}' \ .$$

At the end of the interval:

$$y_0 = y_m \ ,$$

$$y_0' = y_m' \ .$$

and the next set of calculations for finding y_m can begin.

E. Results

Since there are numberous combinations of the characteristic number, q, and initial conditions that one can examine, it was decided to take a few selected values for the parameters and initial conditions, which would illustrate what can be encountered.

It was decided to take the initial guesses for the characteristic number as -50.0, 10.0, and 30.0, with the values for q being 1.0, 10.0, 20.0, 30.0, and 40.0. This enabled one to check some of the values obtained against tabulated results.

For the case of q = 1.0, both the piecewise linearization and the Runge-Kutta solution [5] had a close correspondence with the series approximation to the exact solution, as can be seen in Figures 6.2 - 6.4. For most of the values, the same applies for q = 10.0 (Figures 6.5 and 6.6 being examples of the results obtained). However, something unusual occurred for the case of q = 10.0, a_{GUESS} = -50.0, y(0) = 0.0, y'(0) = 1.0 (Figure 6.7), where the piecewise linearization exhibits erratic behaviour.

Consider the cases where q = 20.0. Figure 6.8 is a typical example of the good correspondence between all three solutions, but Figure 6.9 shows some anomalous results. In fact, tests for this case have shown that the amplitude of the piecewise linear results can go as high as 5×10^5, thus making the Runge-Kutta solution [5] and series approximation insignificant by comparison. A somewhat minor deviation was found to exist for the same values of a_{GUESS} and q, but for the other set of initial conditions, though this occurs near the end of the first cycle, and was less than a third of the maximum values of the other solutions. Up to that point, however, it had given satisfactory results.

Since this has been observed to happen, the question now is whether this occurs for other values of a_{GUESS} and q. Indeed it does. Compare the difference between a "good" result (that is, correspondence between all three solutions) and one that is not (Figures 6.10 and 6.11, respectively, for q = 30.0, and Figures 6.12 and 6.13, respectively, for q = 40.0).

The magnitude and form of the deviations was totally unexpected, since they did not even follow the basic pattern of the other solutions, but seem to be random occurrences, quite unlike what was seen in the previous chapter where the shape of the resulting piecewise linear solution curve at least was the same as that of the solution being

FIGURE 6.2 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 1.0$, $\Delta z = \pi/200.00$, $y(0) = 0.0$, $y'(0) = 1.0$

(N.B.: Runge-Kutta solution uses [5].)

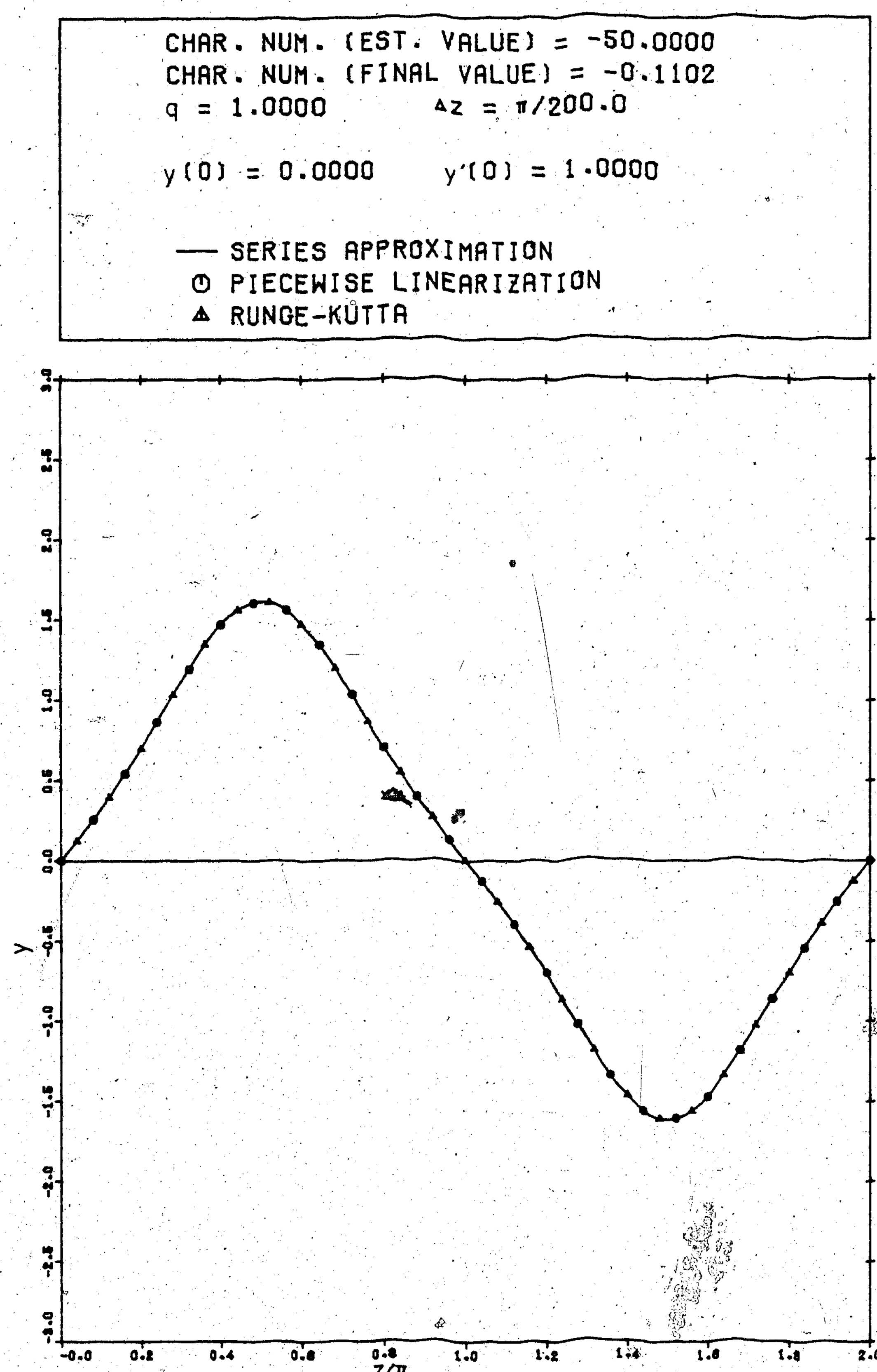

CHAR. NUM. (EST. VALUE) = -50.0000
CHAR. NUM. (FINAL VALUE) = -0.1102
q = 1.0000
Δz = π/200.0
y(0) = 0.0000
y'(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π
-0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0

FIGURE 6.3 - SOLUTIONS TO MATHIEU EQUATION FOR a_{GUESS} = 10.0, q = 1.0, $\Delta z = \pi/200.00$, y(0) = 1.0, y'(0) = 1.0

(N.B.: Runge-Kutta solution uses [5].)

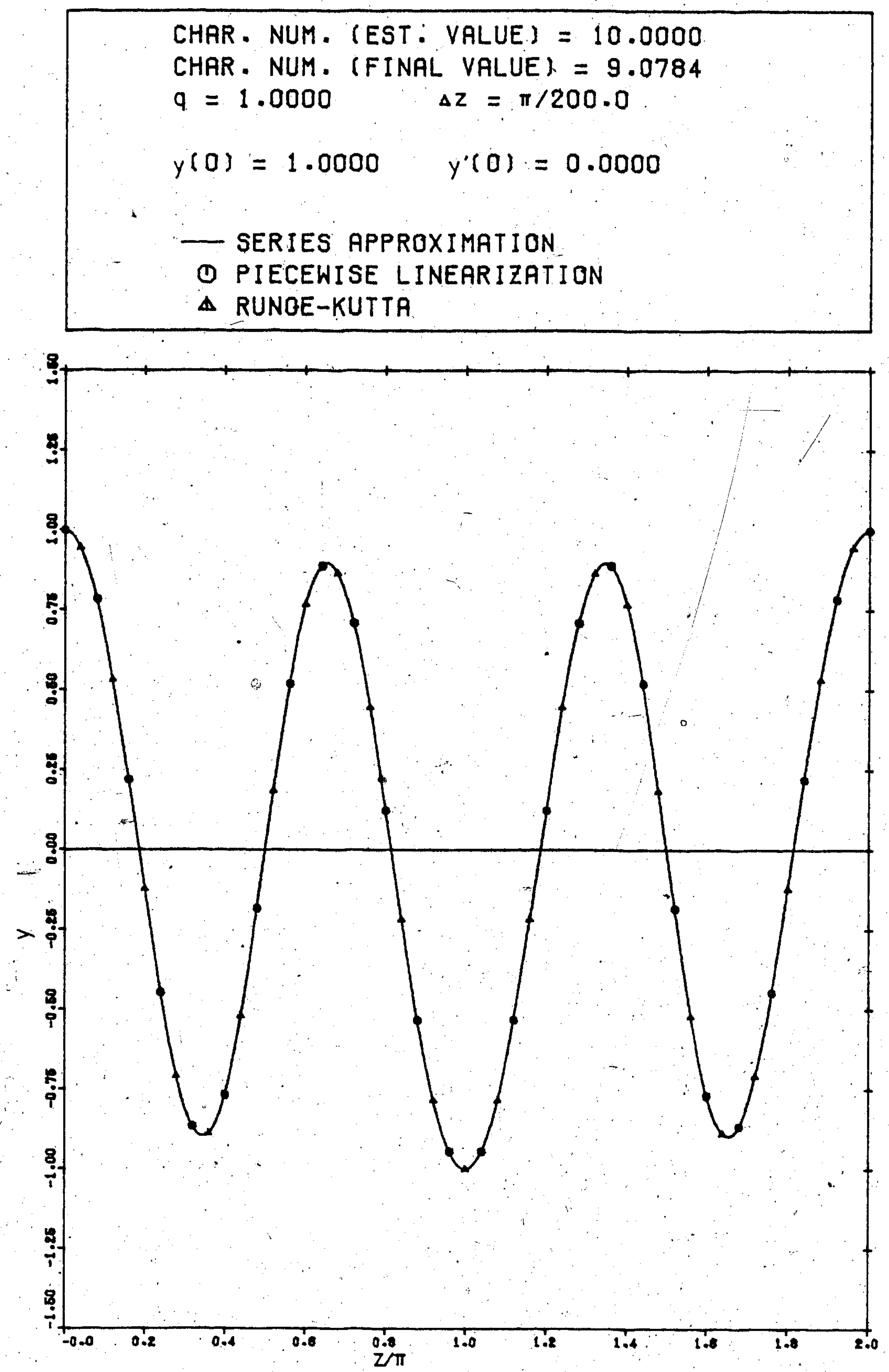

CHAR. NUM. (EST. VALUE) = 10.0000
CHAR. NUM. (FINAL VALUE) = 9.0784
q = 1.0000
ΔZ = π/200.0
y(0) = 1.0000
y'(0) = 0.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π
-0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
1.50
1.25
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
-1.25
-1.50

FIGURE 6.4 - SOLUTIONS TO MATHIEU EQUATION FOR a_{GUESS} = 30.0, q = 1.0, Δz = π/200.00, y(0) = 0.0, y'(0) = 1.0

(N.B.: Runge-Kutta solution uses [5].)

CHAR. NUM. (EST. VALUE) = 30.0000
CHAR. NUM. (FINAL VALUE) = 25.0208
q = 1.0000 $\Delta z = \pi/200.0$

$y(0)$ = 0.0000 $y'(0)$ = 1.0000

— SERIES APPROXIMATION
⊙ PIECEWISE LINEARIZATION
△ RUNGE-KUTTA

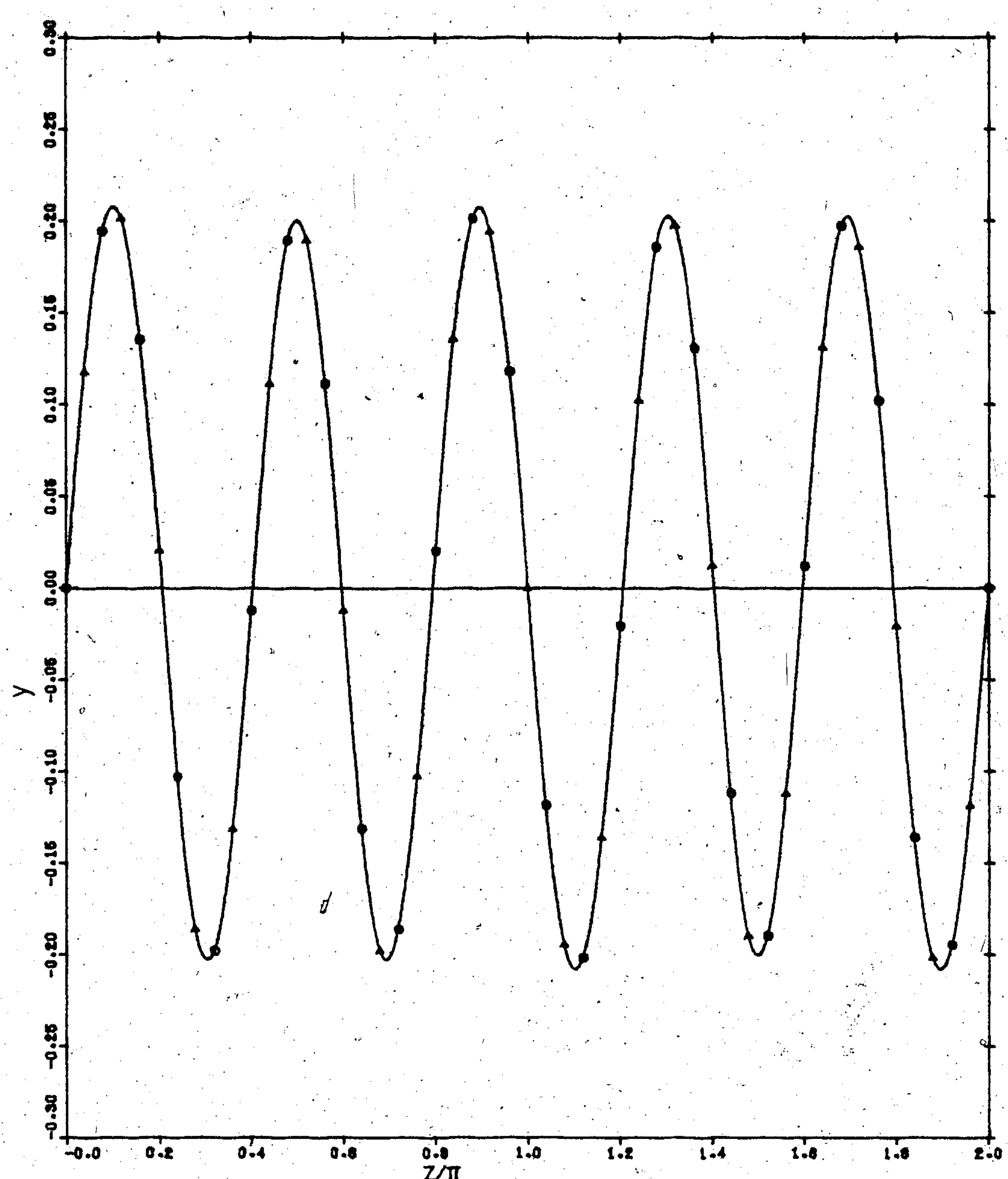


FIGURE 6.5 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = 10.0$, $q = 10.0$, $\Delta z = \pi/200.00$, $y(0) = 0.0$, $y'(0) = 1.0$

(N.B.: Runge-Kutta solution uses [5].)

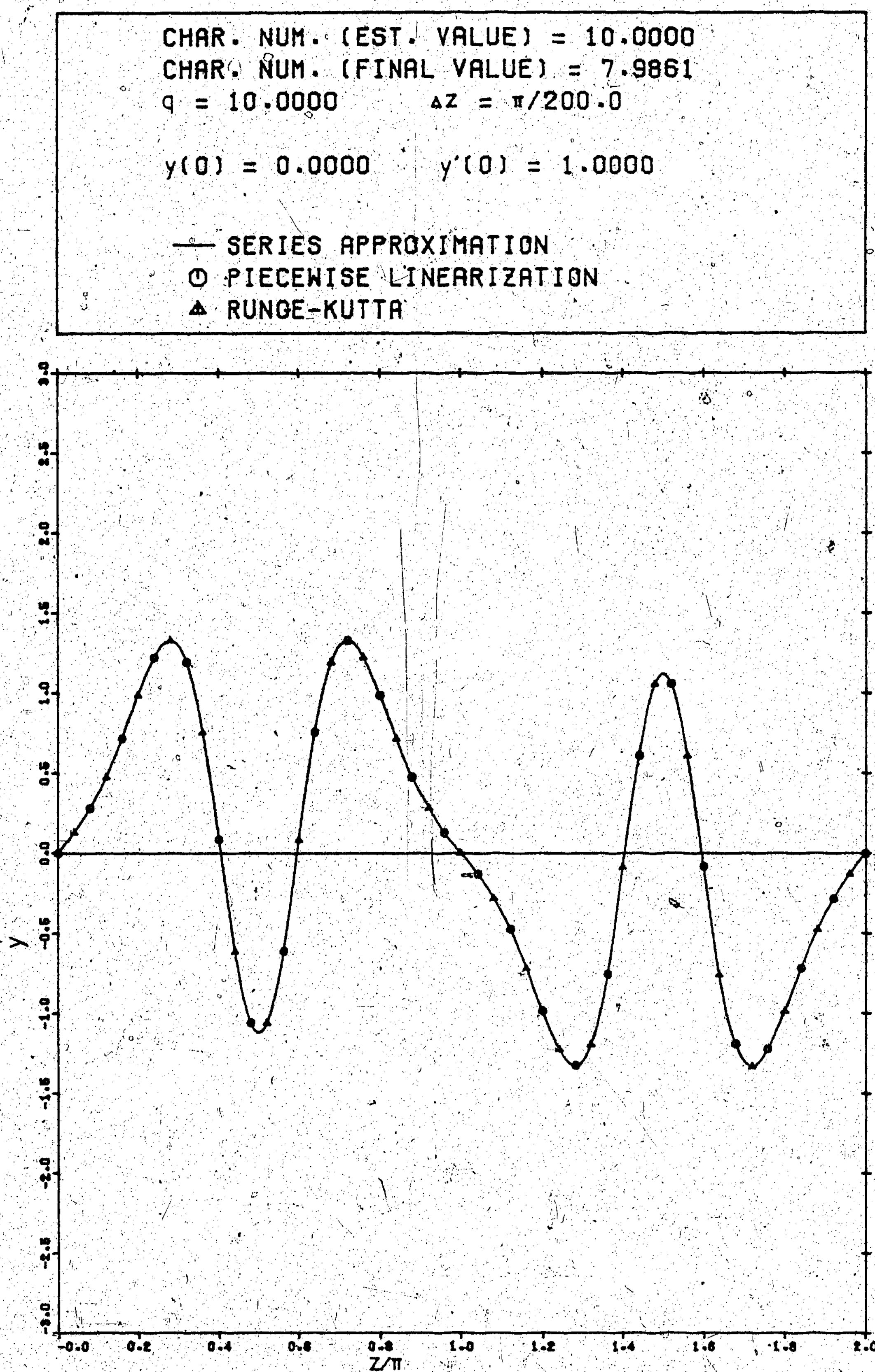
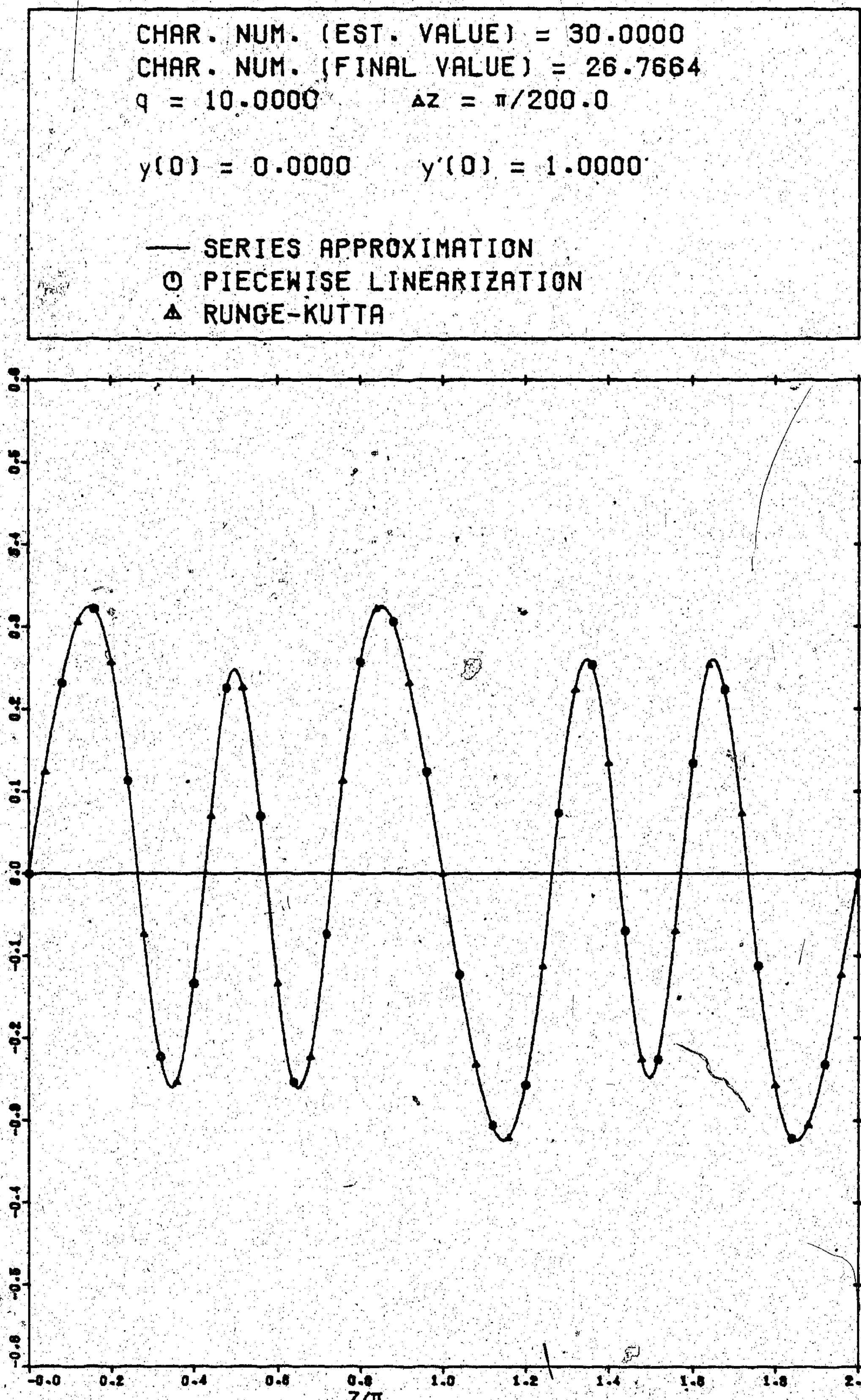

CHAR. NUM. (EST. VALUE) = 10.0000
CHAR. NUM. (FINAL VALUE) = 7.9861
q = 10.0000 ΔZ = π/200.0
y(0) = 0.0000 y′(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
Z/π

FIGURE 6.6 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = 30.0$, $q = 10.0$, $\Delta z = \pi/200.00$, $y(0) = 0.0$, $y'(0) = 1.0$

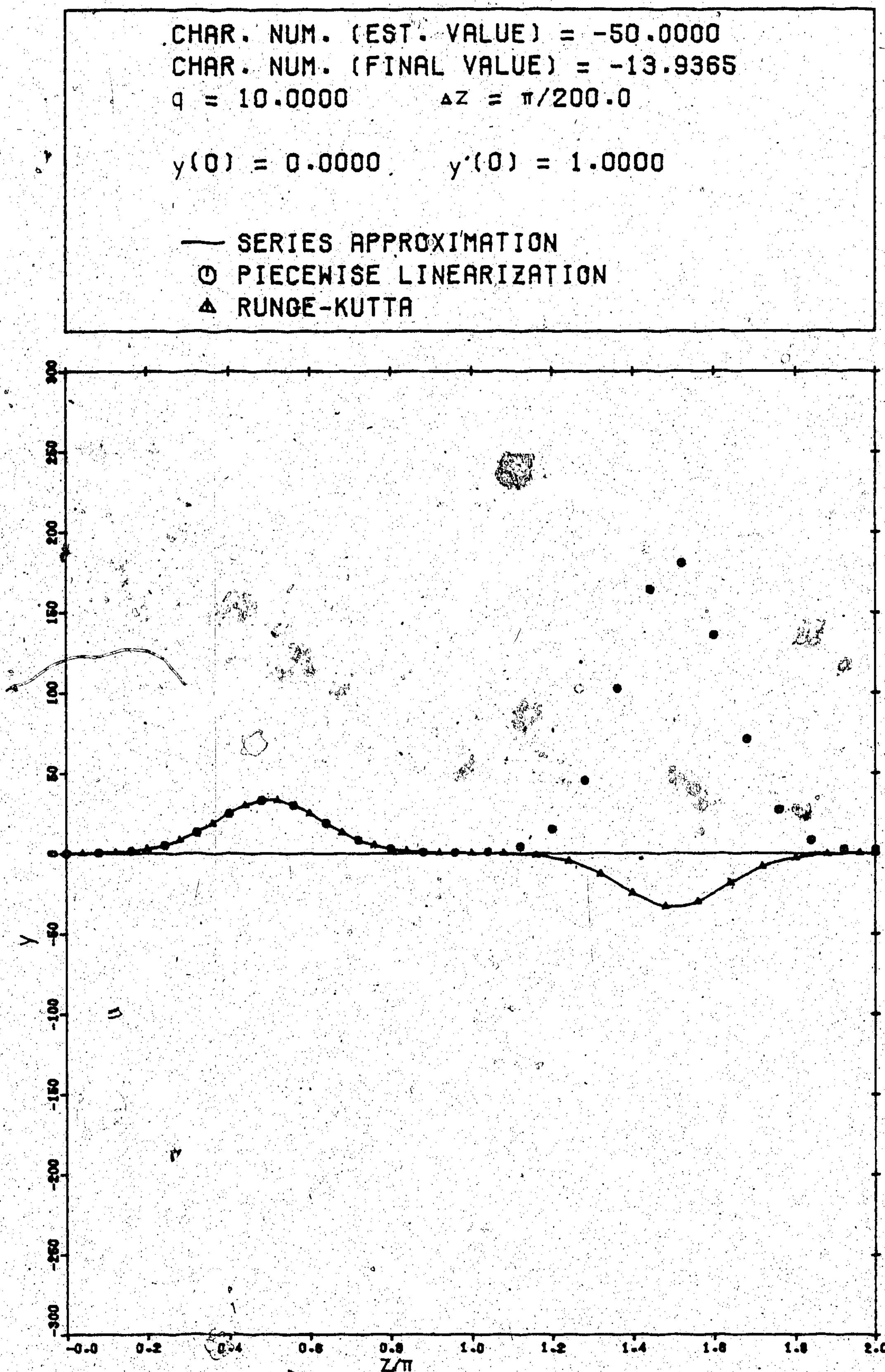
(N.B.: Runge-Kutta solution uses [5].)



CHAR. NUM. (EST. VALUE) = 30.0000
CHAR. NUM. (FINAL VALUE) = 26.7664
q = 10.0000 ΔZ = π/200.0
y(0) = 0.0000 y′(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π

FIGURE 6.7 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 10.0$, $\Delta z = \pi/200.00$, $y(0) = 0.0$, $y'(0) = 1.0$

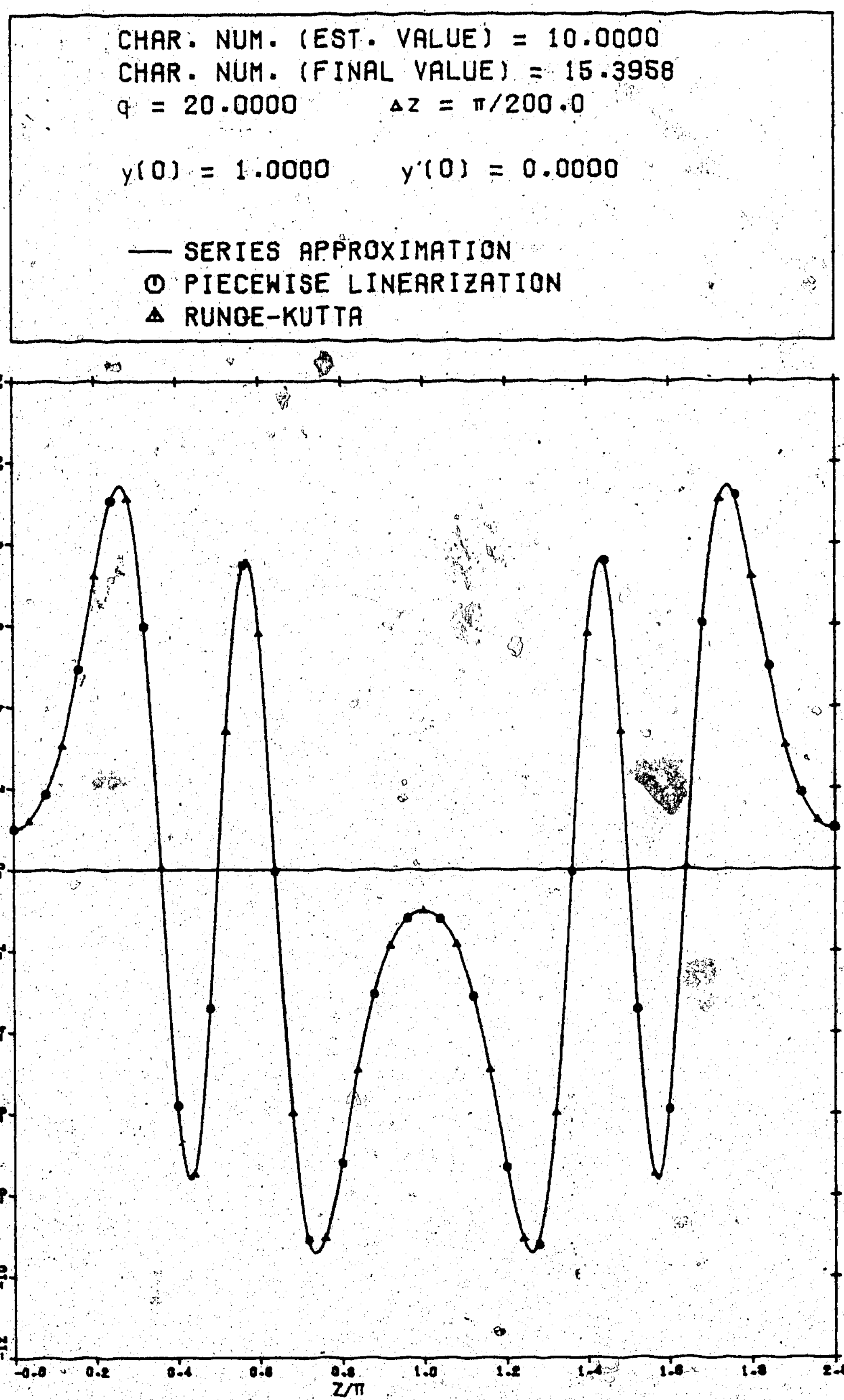
(N.B.: Runge-Kutta solution uses [5].)



CHAR. NUM. (EST. VALUE) = -50.0000
CHAR. NUM. (FINAL VALUE) = -13.9365
q = 10.0000   ΔZ = π/200.0
y(0) = 0.0000   y'(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π

FIGURE 6.8 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = 10.0$, $q = 20.0$, $\Delta z = \pi/200.00$, $y(0) = 1.0$, $y'(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)



CHAR. NUM. (EST. VALUE) = 10.0000
CHAR. NUM. (FINAL VALUE) = 15.3958
q = 20.0000 Δz = π/200.0
y(0) = 1.0000 y'(0) = 0.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π

FIGURE 6.9 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 20.0$, $\Delta z = \pi/200.00$, $y(0) = 0.0$, $y'(0) = 1.0$

(N.B.: Runge-Kutta solution uses [5].)

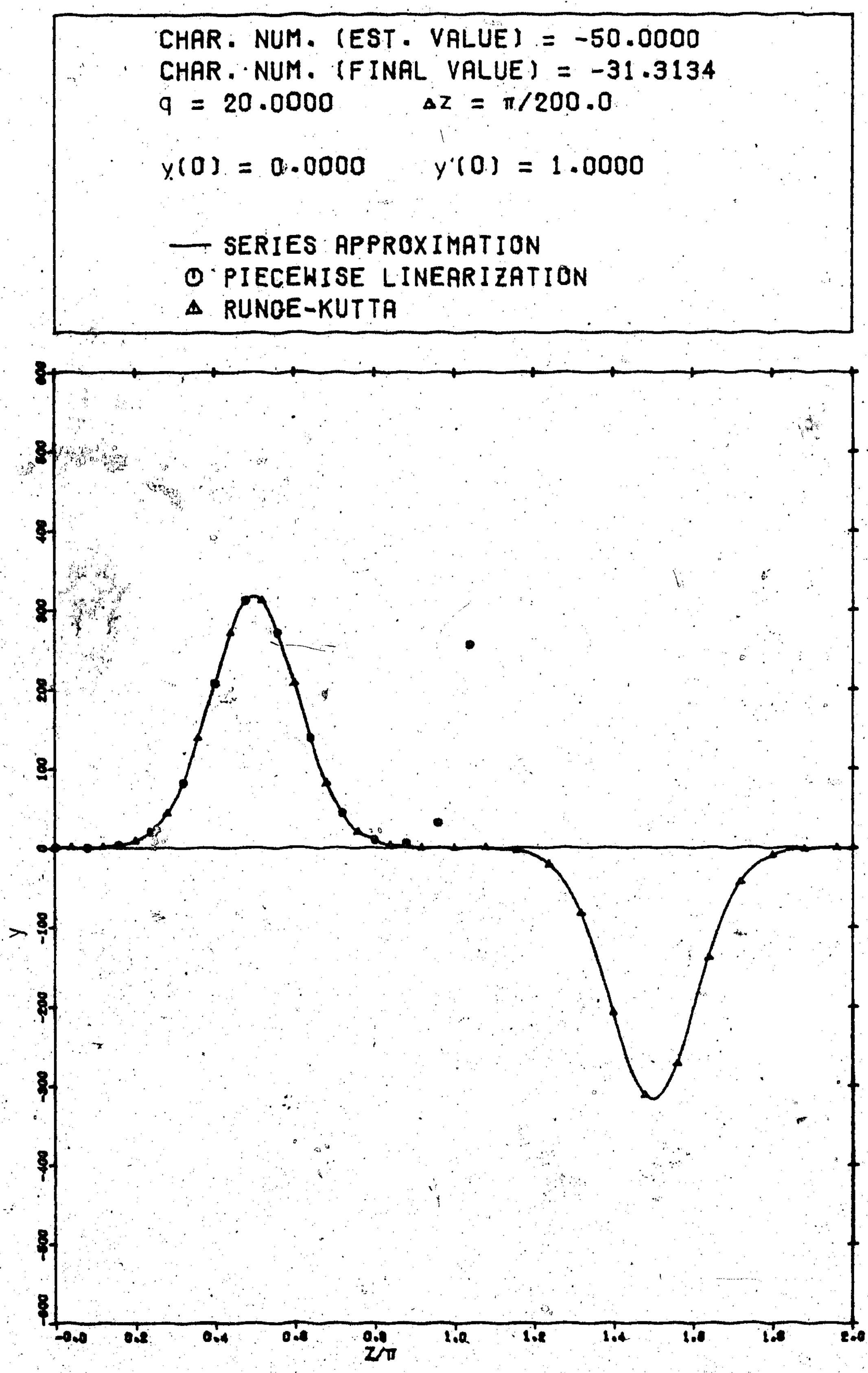

CHAR. NUM. (EST. VALUE) = -50.0000
CHAR. NUM. (FINAL VALUE) = -31.3134
q = 20.0000   Δz = π/200.0
y(0) = 0.0000   y'(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π
-0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
600
500
400
300
200
100
0
-100
-200
-300
-400
-500
-600

FIGURE 6.10 - SOLUTIONS TO MATHIEU EQUATION FOR
$a_{GUESS} = 30.0$, $q = 30.0$, $\Delta z = \pi/200.00$,
$y(0) = 0.0$, $y'(0) = 1.0$

(N.B.: Runge-Kutta solution uses [5].)

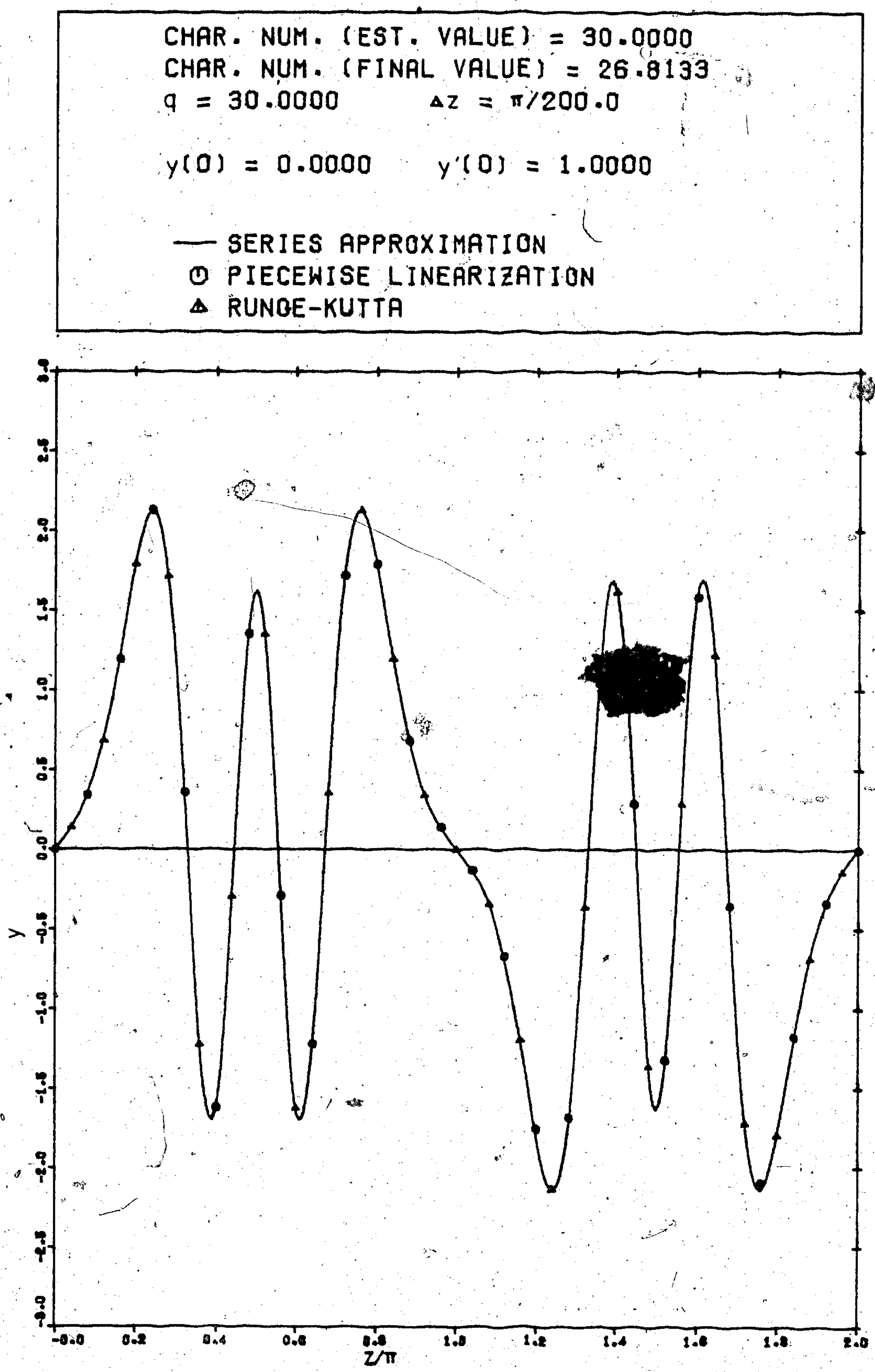

CHAR. NUM. (EST. VALUE) = 30.0000
CHAR. NUM. (FINAL VALUE) = 26.8133
q = 30.0000 ΔZ = π/200.0
y(0) = 0.0000 y'(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π
-0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0

FIGURE 6.11 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 30.0$, $\Delta z = \pi/200.00$, $y(0) = 0.0$, $y'(0) = 1.0$

(N.B.: Runge-Kutta solution uses [5].)

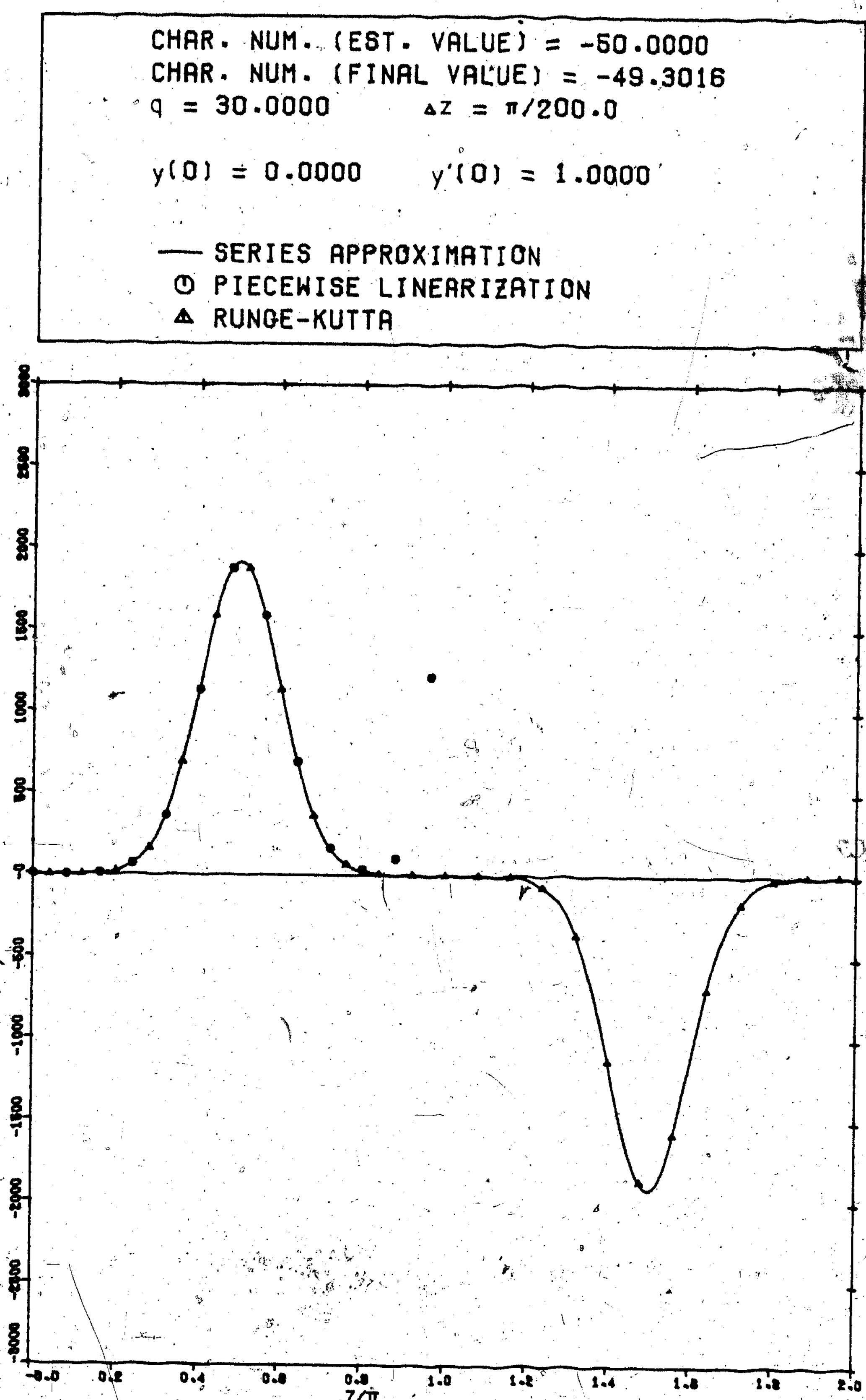

CHAR. NUM. (EST. VALUE) = -50.0000
CHAR. NUM. (FINAL VALUE) = -49.3016
q = 30.0000
ΔZ = π/200.0
y(0) = 0.0000
y'(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0

FIGURE 6.12 - SOLUTIONS TO MATHIEU EQUATION FOR
a_{GUESS} = 30.0, q = 40.0, $\Delta z = \pi/200.00$,
y(0) = 1.0, y'(0) = 0.0

(N.B.: Runge-Kutta solution uses [5].)

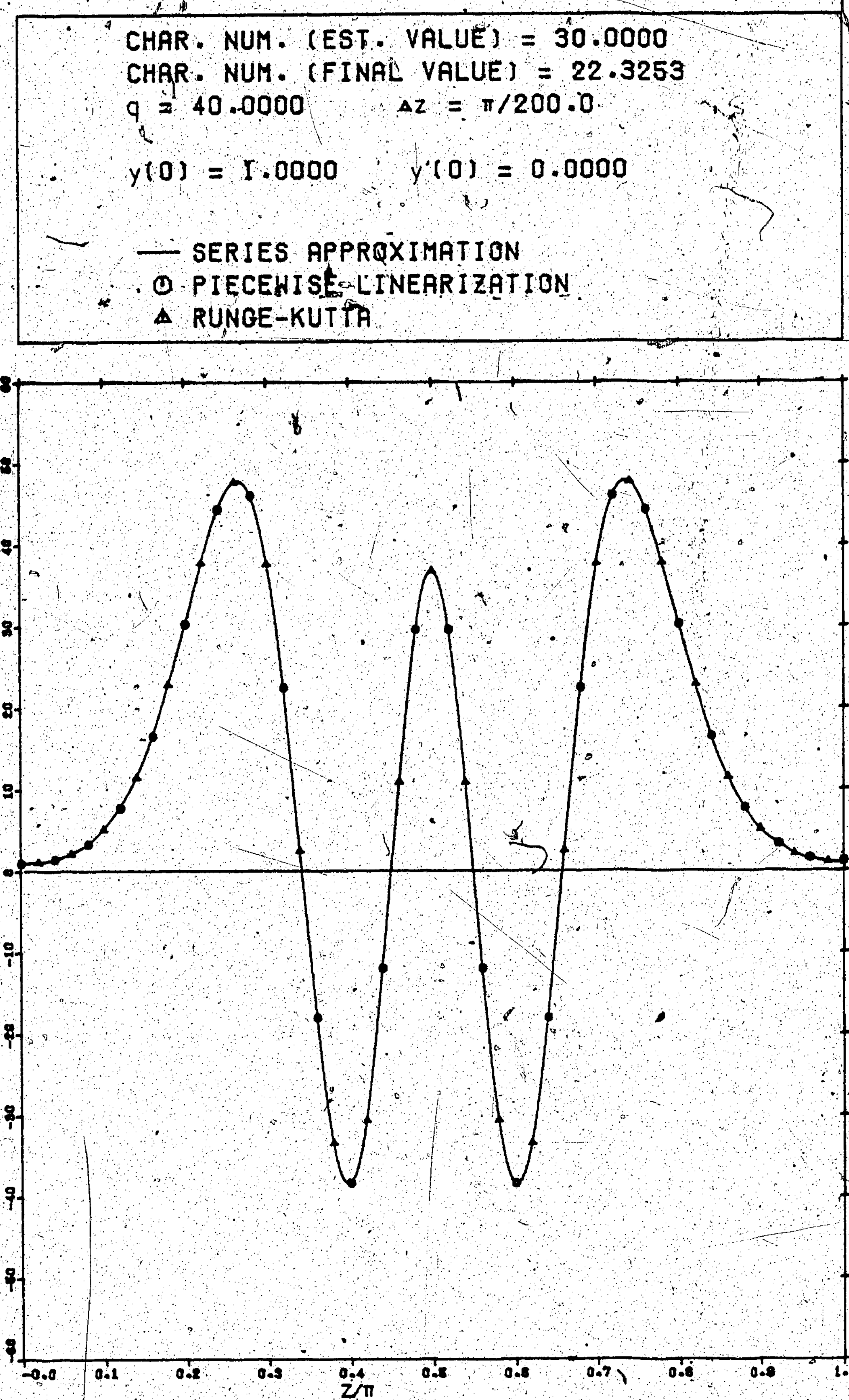
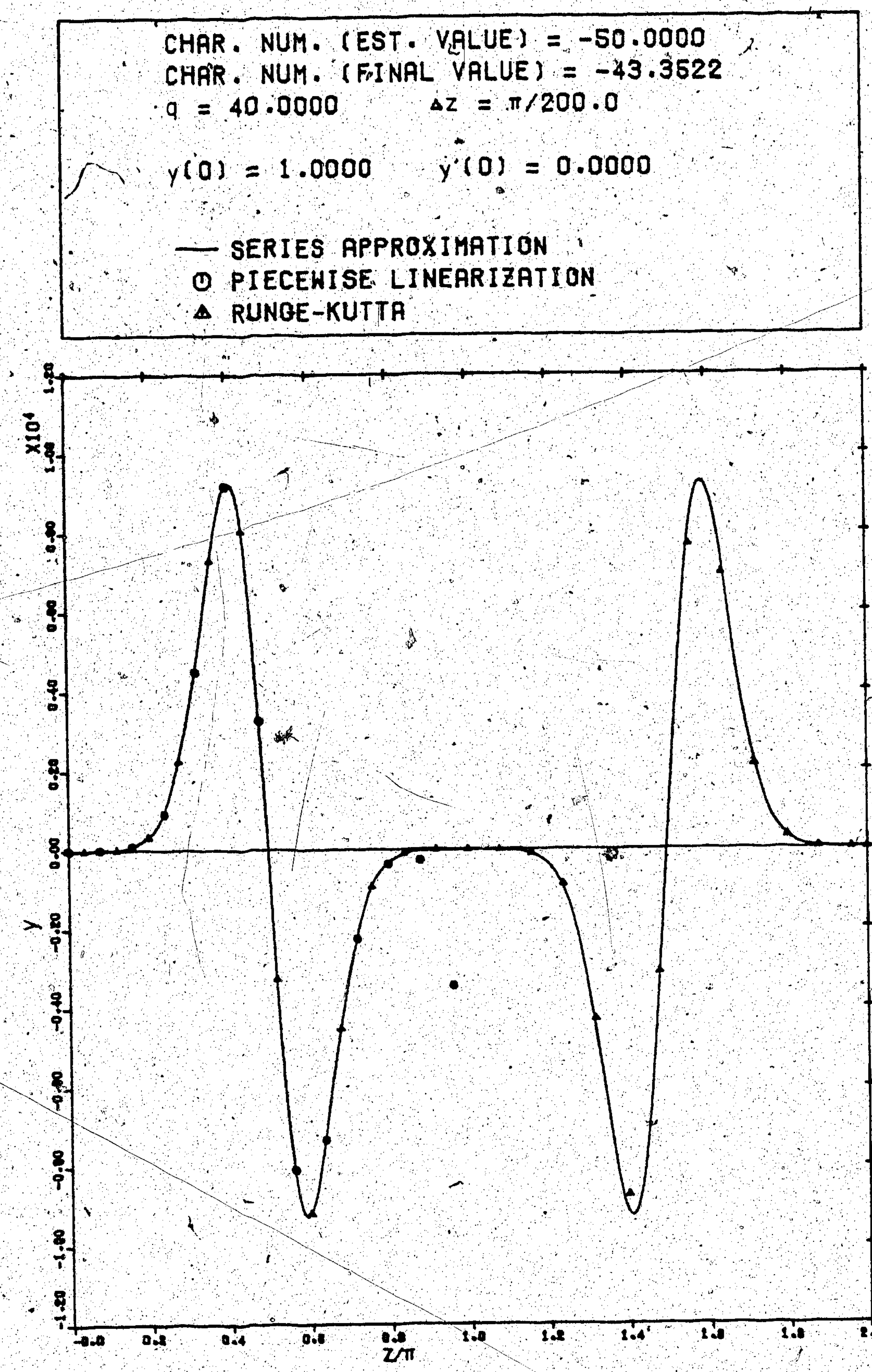

CHAR. NUM. (EST. VALUE) = 30.0000
CHAR. NUM. (FINAL VALUE) = 22.3253
q = 40.0000
Δz = π/200.0
y(0) = 1.0000
y'(0) = 0.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
-0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Z/π

FIGURE 6.13 – SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 40.0$, $\Delta z = \pi/200.00$, $y(0) = 1.0$, $y'(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)


CHAR. NUM. (EST. VALUE) = -50.0000
CHAR. NUM. (FINAL VALUE) = -43.3522
q = 40.0000
Δz = π/200.0
y(0) = 1.0000
y'(0) = 0.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
X10⁴
y
Z/π

compared. Since changing the size of the increment of z seemed to work for the Van der Pol equation, it was felt that this may solve the problem. A typical result of that was as can be seen in Figure 6.14. This particular case was chosen as this phenomenon was first observed during the initial development of the computer program using the same conditions, but a_{GUESS} being -45.0.

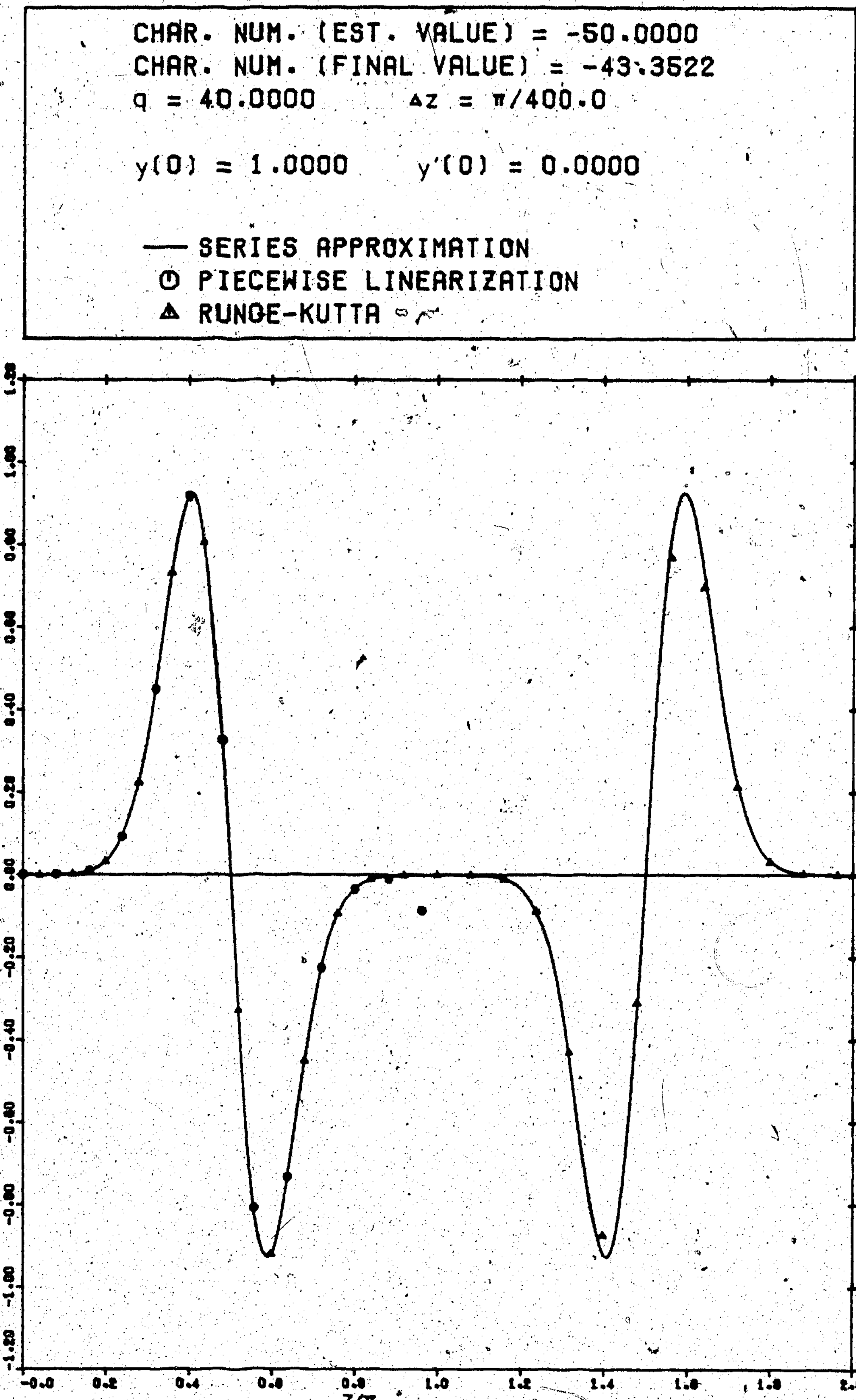
It was suggested that the computer program written to generate the solutions and plots could be run up to a point just prior to the break-down, and then using the final values as a new set of initial conditions. Unfortunately, this helped very little. As a check as to whether the computer program was at fault, a test program was run on a Hewlett-Packard HP-67 hand calculator using the same conditions, and it was found that the same situation occurred with essentially the same outcome. This was double-checked by running a Runge-Kutta routine designed for the HP-67 by Hewlett-Packard [59], but with the same result.

At present, the question as to what occurred, and why, remains unanswered. However, it would appear from the plots obtained for the region of $q \leq 20.0$ that no problems would be experienced for a positive value of the characteristic number. Possibly, a small negative value might do so, but this was not investigated. There are indications that this may even extend up to $q = 30.0$. Problems quite definitely develop for $q = 40.0$, and it appears that regardless of what the value for the characteristic number is, deviations from the series approximation will occur.

There does not appear to be any symmetry in the severity of the deviations as far as initial conditions are concerned. An example of

FIGURE 6.14 - SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 40.0$, $\Delta z = \pi/400.00$, $y(0) = 1.0$, $y'(0) = 0.0$

(N.B.: Runge-Kutta solution uses [5].)


CHAR. NUM. (EST. VALUE) = -50.0000
CHAR. NUM. (FINAL VALUE) = -43.3522
q = 40.0000 Δz = π/400.0
y(0) = 1.0000 y'(0) = 0.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
X10⁴
Y
Z/π
1.0

this is the case of $a_{GUESS} = -50.0$ and $q = 10.0$. For the conditions of $y(0) = 1.0$, and $y'(0) = 0.0$, there is a slight deviation near the end of the first cycle, but only a fraction of the maximum displacement. However, for the same characteristic number estimate and q, but $y(0) = 0.0$ and $y'(0) = 1.0$, a totally different situation arises in which the maximum displacement for the piecewise linearization dwarfs the other solutions. Though this is an isolated example, there may possibly be a problem here.

As had been shown in the previous chapters, the program had been stripped to its essential calculations, removing comment and output statements, and the timings of the three solutions were obtained [27] for selected cases. The results of this can be seen in Figures 6.15 - 6.17. This was run on the Amdahl 470V/8 [26].

For the piecewise linearization and the series approximation, a linear relationship with the number of divisions of π (and hence, the size of the increment of z) and the execution time exists, but the Runge-Kutta method [5] displays some unusual features. First, its execution times are almost always larger than those for the other two solutions, except for Figure 6.15 and the smaller values of Δz in Figure 6.16. In fact, it is considerably larger than what was obtained for the piecewise method, being larger by about a factor of 10 for a large Δz but down to four times as large for the smaller values.

The other unusual feature is the fact that the Runge-Kutta results [5] show some scatter, particularly seen on Figure 6.16. This is different than what was observed for the hard spring equations (in which the results were linear) and for the Van der Pol equation where the results were exponential.

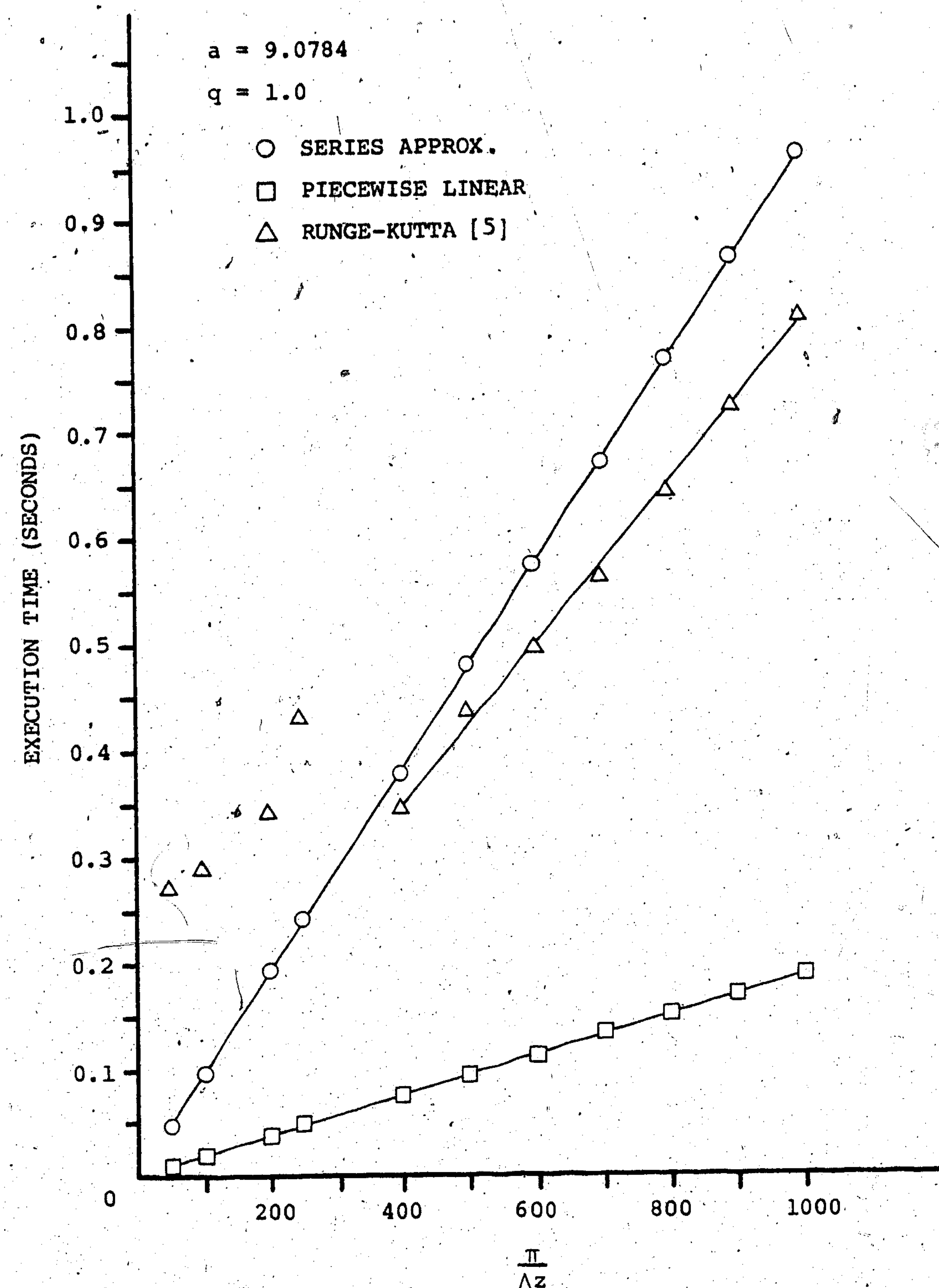


FIGURE 6.15 - EXECUTION TIMES FOR SOLUTIONS TO MATHIEU EQUATION FOR a_{GUESS} = 10.0, q = 1.0, y(0) = 1.0, y'(0) = 0.0 [27]

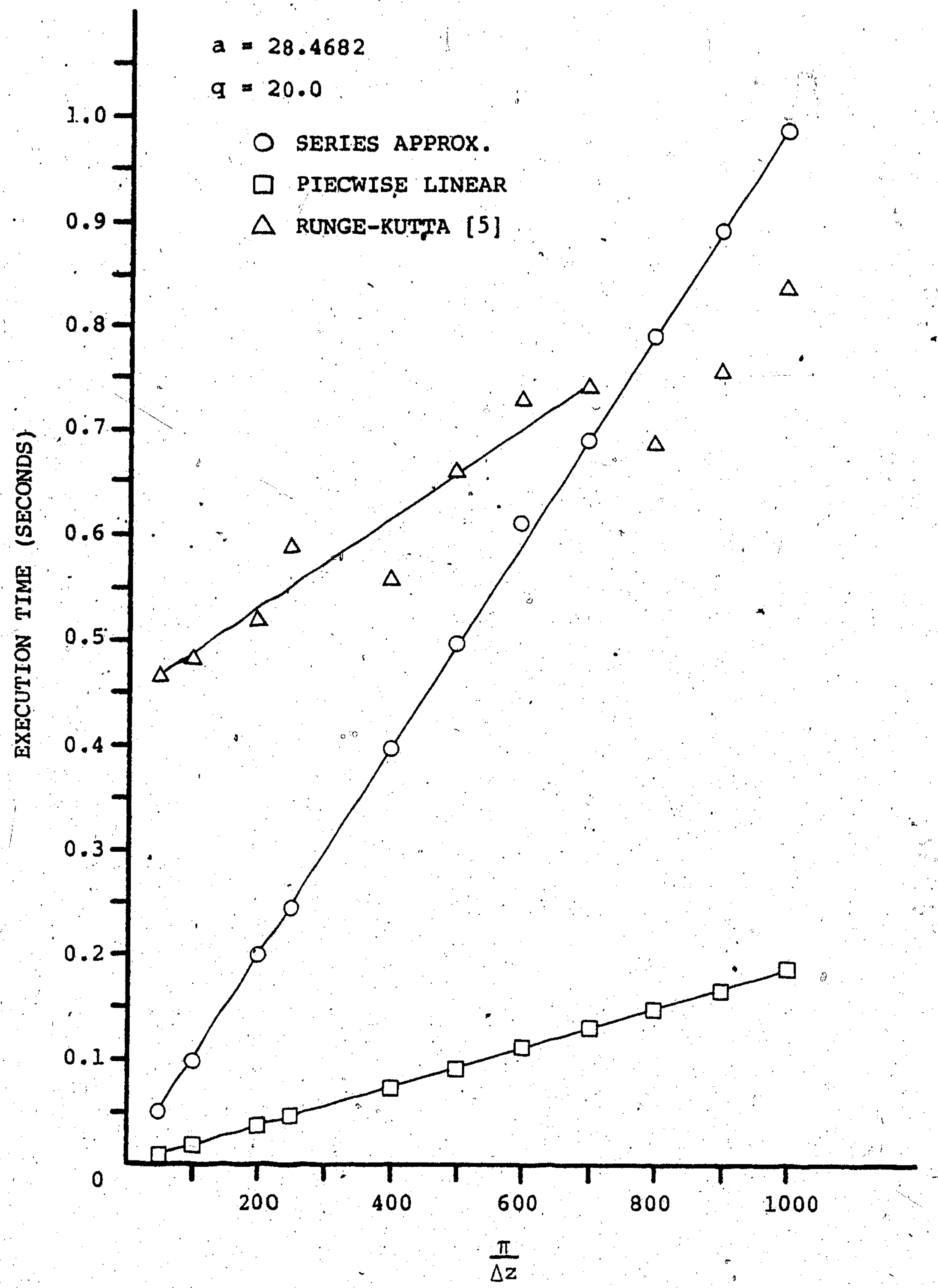


FIGURE 6.16 - EXECUTION TIMES FOR SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = 30.0$, $q = 20.0$, $y(0) = 1.0$, $y'(0) = 0.0$ [27]

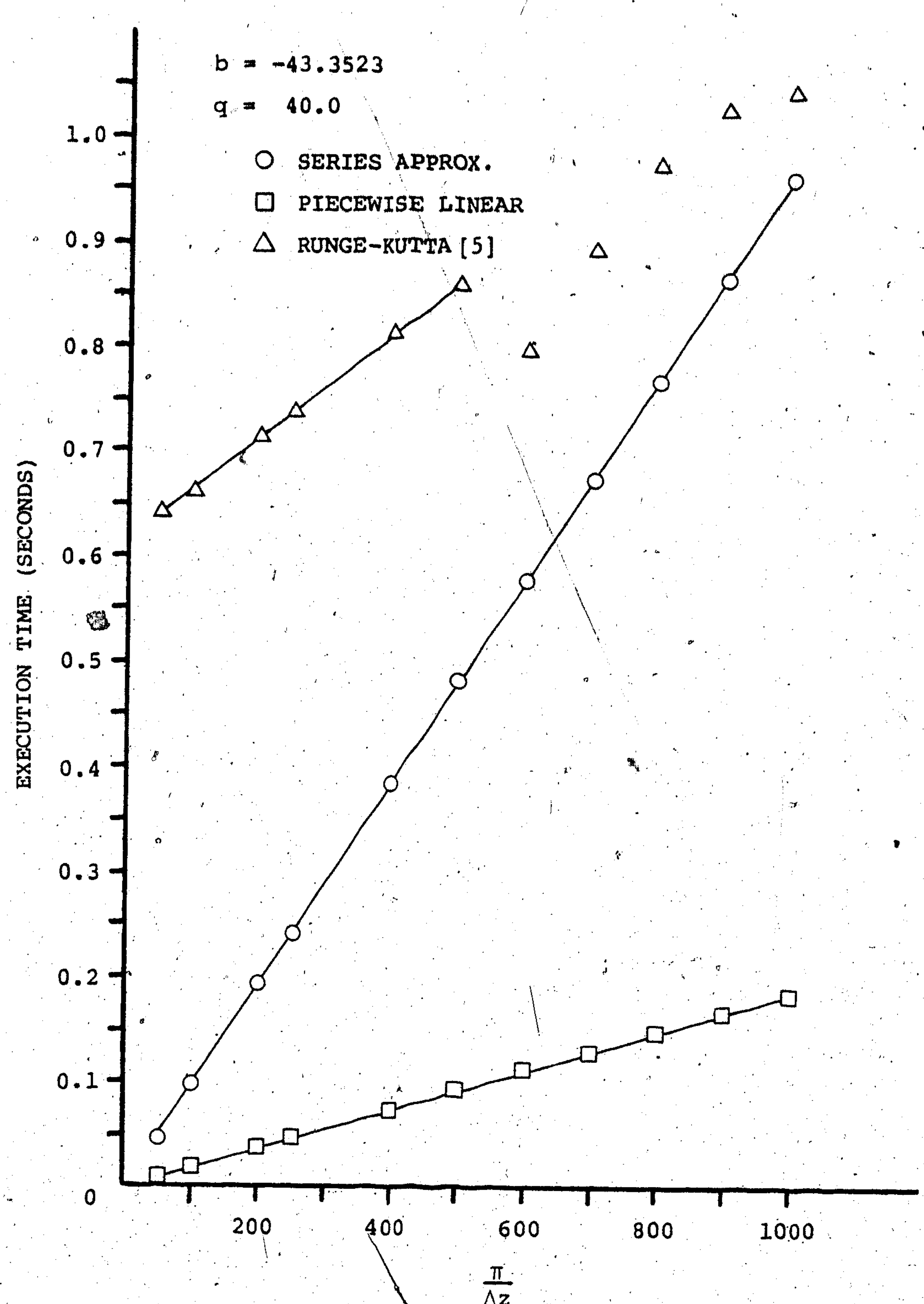


FIGURE 6.17 - EXECUTION TIMES FOR SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 40.0$
$y(0) = 1.0$, $y'(0) = 0.0$ [27]

This behaviour was discussed with a consultant from the University of Alberta Computing Services [60] in an attempt to determine what had brought this about. A suggested explanation is that the routine that generated the Runge-Kutta results [5] may have been affected in whole or in part by the actual size of the Δz used. However, the exact cause of the execution time variation remains unknown.

VII. DISCUSSION

As was mentioned in Chapter 2, the method of tangents for solving the undamped hard spring equation had a greater execution time than the method of chords because of the extra calculations that had to be carried out in order to arrive at a solution. The period of oscillation as calculated by the former method was closer to the value obtained by the approximation to the exact solution [4], [15], [17] than the period calculated by the latter one, but the method of chords had a closer fit to the solution curve of this approximation than did the one using tangents.

For 5 segments, the period as calculated by the approximation to the exact solution [4], [15], [17] for a = 1.0 and b = 2.0 is 4.004308722, while the one obtained by chords is 3.976017631 with the one by tangents, 4.018171439. The period found by the method of chords was 0.707% less than (2.2.5) [17], while the value for the method of tangents was 0.346% larger. If one were to consider the case of 100 segments for a = 1.0 and b = 0.15, where (2.2.5) [17] was found to be 5.958299670, the chords gave 5.958281464 (0.0003% less) while the tangents gave 5.958310457 (0.0002% greater). So, the tangents gave a more accurate calculation, as far as the magnitude of error is concerned, for the period.

However, in judging a method of solution, the period itself is not necessarily the only standard. The accuracy of the fit of the calculated solution with respect to the one that had been chosen to be the benchmark [4], [15], [17] is perhaps more important, and it is here that the method of tangents starts showing a small, but significant, difference. If one were to look at Figure 2.8, one can see that the tangents are off by a slight amount, while the chords are much closer. For Figures 2.6 and 2.7, this does not become immediately apparent, which indicates that the values of a and b would have some bearing on the closeness of the fit.

As mentioned before, it was because the method of chords gave a closer fit that this method was adopted throughout the remainder of the thesis as the method of piecewise linearization, as well as the fact that it was quicker to calculate.

The execution time provided some interesting results. Since the solution chosen to be the standard was based on a packaged program library [5], one would have expected it to have a shorter execution time than the other solutions, but since (2.2.4) [15] was calculated each time, this apparently unusual result appears to be resolved. It should be noted that while setting up the timing program, the author had run it in which (2.2.4) [15] was calculated only once and the solutions generated after that, and the approximation to the exact solution did, in fact, turn out to be faster by about 10% than that of the method of chords for most of the runs.

The two cases of the damped hard spring were an extension of the undamped one. For the situations examined, a good agreement between the results of the piecewise solution and those of the Runge-Kutta method [5] was obtained, which leads one to conclude that for the size of the displacement used, the piecewise linearization was a suitable approximation to the chosen standard, since an exact solution is not available.

Because the same calculations are carried out for each iteration, the timing results (using [27]) were linear with respect to the number of divisions per first quarter-cycle will occur, and also, the quantities being incremented were different in each solution. Also, as expected, the Runge-Kutta method [5] used less time than the piecewise method. This can be seen if one considers that this is from a packaged program library, and so, should have the advantage of development and modifications. In fact, the IMSL package used was in its seventh edition [5], and presumably, should be as efficient as possible, as compared to the piecewise linearization portion of the computer program used to generate the solutions.

What is unusual is that for a large increment (that is, a small number of divisions per first quarter-cycle), the timings show a large deviation, which probably means that the increment chosen was so large that, in fact, the solution does not have time to converge (if a time was obtained) or that both solutions have to work harder to achieve a result. This appears further on for the Van der Pol equation.

The hard spring equation piecewise linear solutions gave a close correspondence with the results from the Runge-Kutta method [5]. However, this was not the case for the piecewise linearization of the Van der Pol equation and the same Runge-Kutta method [5], and for the piecewise linear results for the Mathieu equation and the approximation to the exact solution.

The apparent "phase difference" noticed in the Van der Pol equation results, and which apparently was resolved by reducing the size of the time increment used, was the first sign that the piecewise linearization could have some limitations. Some insight was gained by considering the subject of stability, since this may have some bearing on the results obtained for the Mathieu equation.

Several things should be noted first before continuing. One is that because of the apparently strange results obtained for the Van der Pol equation for the higher values of μ, there may be a limit beyond which the piecewise linearization becomes unreliable. In other words, for the equation parameters and increment sizes, there may be an area in which this method is ineffective. This is indicated by Figures 5.21 and 5.22 [38], [39].

If this is the case, then, can one predict, if, and when, this will occur for a particular situation? Is it actually due to the numerical instability or possibly machine error? The answer to the first question is that it may not always be possible to do so. This is indicated by Greenspan [61], and it would therefore appear that a trial-and-error approach is the only sure method of determining if an equation becomes unstable for a given set of parameters.

The question now is whether this error was caused by instability or machine error. The internal tolerance for calculations like the Newton-Raphson [30] relation that was used for determining the remaining time until a peak or trough had been reached had been set to smaller values to ensure that perhaps this may not have been the source. This was because the solutions appeared to be all right for the smaller values for μ. Also, the tolerance for the Runge-Kutta program had also been reset to a smaller value [5], but the result was that virtually no change was noticed in this solution. Since the smaller value for the time increment appeared to resolve the problem, it was concluded that numerical instability may have caused it.

This led to the consideration of the accuracy of the solution to (5.1.1) obtained by using the Runge-Kutta method [5]. Figures 5.21 and

5.2.2 [38], [39] indicated that it may become inaccurate with a large time interval, but perhaps not to the same extent as the piecewise linearization. It can therefore be concluded that the results from the Runge-Kutta method [5] may be misleading if factors like this are not taken into account.

Next, the timing runs were done [27]. The piecewise linearization had a parabolic relationship between execution time and the size of the time increment, and the Runge-Kutta [5] had an exponential relation. This can be easily explained by referring for a moment back to the results for the damped hard spring equations.

There, for a small number of divisions for the first quarter-cycle, the times for both solutions deviated significantly from the remainder of the results, which indicated that both solutions had to go through more iterations at this stage. The same probably occurs here, since there were some runs for which times were not obtained because the computer kept exceeding its set time limit [31]. This occurred for the higher values of Δt_P and Δt_{RK}. For a small value of Δt a large number of iterations had to be carried out, though, because of the small increment, convergence would be rather quick [60].

The aspect of instability occurs once again with the Mathieu equation, and this will now be examined. For the lower values of q, there was generally no problem with the characteristic number was large and negative, and what makes it particularly puzzling is the fact that the situation described by Figure 6.7 does not seem to follow any particular pattern relating to the other solutions. For the Van der Pol equation, the deviations at least looked like the Runge-Kutta solution [5], but expanded along the time axis. However, for the Mathieu equation, this does not seem to happen.

Figure 6.7 may very well represent part of the unstable region, as had just been discussed. Figures 6.9, 6.11, 6.13, and 6.14 were interesting in the fact that the maximum amplitude of the piecewise linearization completely dwarfed the other solutions as was seen when the initial results were obtained. (It should be noted that the plots were scaled by taking the largest amplitude of the three solutions, and so, this result will occur.) The program was temporarily modified to show the other solutions, and so, the piecewise linearization results end up going off scale.

In particular, Figures 6.13 and 6.14 display that changing the size of Δz does not seem to have any major effect on this phenomenon. The analysis is further complicated through the piecewise linear results following the series approximation to the exact solution quite closely until nearly the end of the first half-cycle, and then becoming unstable.

Though not shown here, it should be noted that the results from the Runge-Kutta method [5] had exhibited similar behaviour. This was observed by the author when developing the program for solving the Mathieu equation for the same initial conditions for Figures 6.13 and 6.14, but using an a_{GUESS} of -45.0 instead of -50.0. It became unstable to a smaller extent than the piecewise linearization and took place in roughly the same area, while the series approximation was "well-behaved".

Perhaps some insight can be gained by examining the subject of numerical instability. First, one must consider what sort of instabilities exist.

Most texts on numerical analysis will give four definitions, depending upon what the situation in question is, such as [62], [63], [64], [65], [66]. (It should be noted that the definitions to be given are as used in the references.)

"Inherent" instability occurs when a small change in the starting value will cause large changes in the solution to occur, but this is independent of the numerical method chosen to solve the problem [62]. It is the equation itself that is insensitive, and not the means used to solve it [63].

"Partial" instability arises when the solution deviates significantly from the true one as more steps are taken. The situation is dependent upon the equation that is being solved, the method used, and especially the step size. This appears to be resolved when a smaller increment is chosen [64].

"Strong" instability arises when one solves the equation, but additional terms arise that do not vanish as the step size gets very small. These terms increase faster than the desired solution, and lack of convergence as well as stability is implied here [65], [66].

"Weak" instability is defined as having the same behaviour as what characterizes strong instability occurs, but the solution converges [65].

In view of these definitions, the problems with the Mathieu equation piecewise linear solution seem to display a combination of all of these, but cannot be completely described by any one of them. As was mentioned before, the initial conditions were changed by a slight amount, but this did not give rise to the radical changes that "inherent" instability would imply. "Partial" instability [64] seems to describe what had occurred with the Van der Pol equation, but still, a change in the step size did not seem to matter much, though there is some indication that if one were to take an extremely small step size (such as on the order of $\pi/5000$), a partial success is achieved in that the amplitude of the piecewise linear method is reduced by at least a factor of 10, but the "takeoff" point for this irregularity does not seem to have been affected much by being delayed for a few more increments, as compared to what happened with the Van der Pol equation. So, it would seem that "partial" instability [64] can be eliminated as a probable cause.

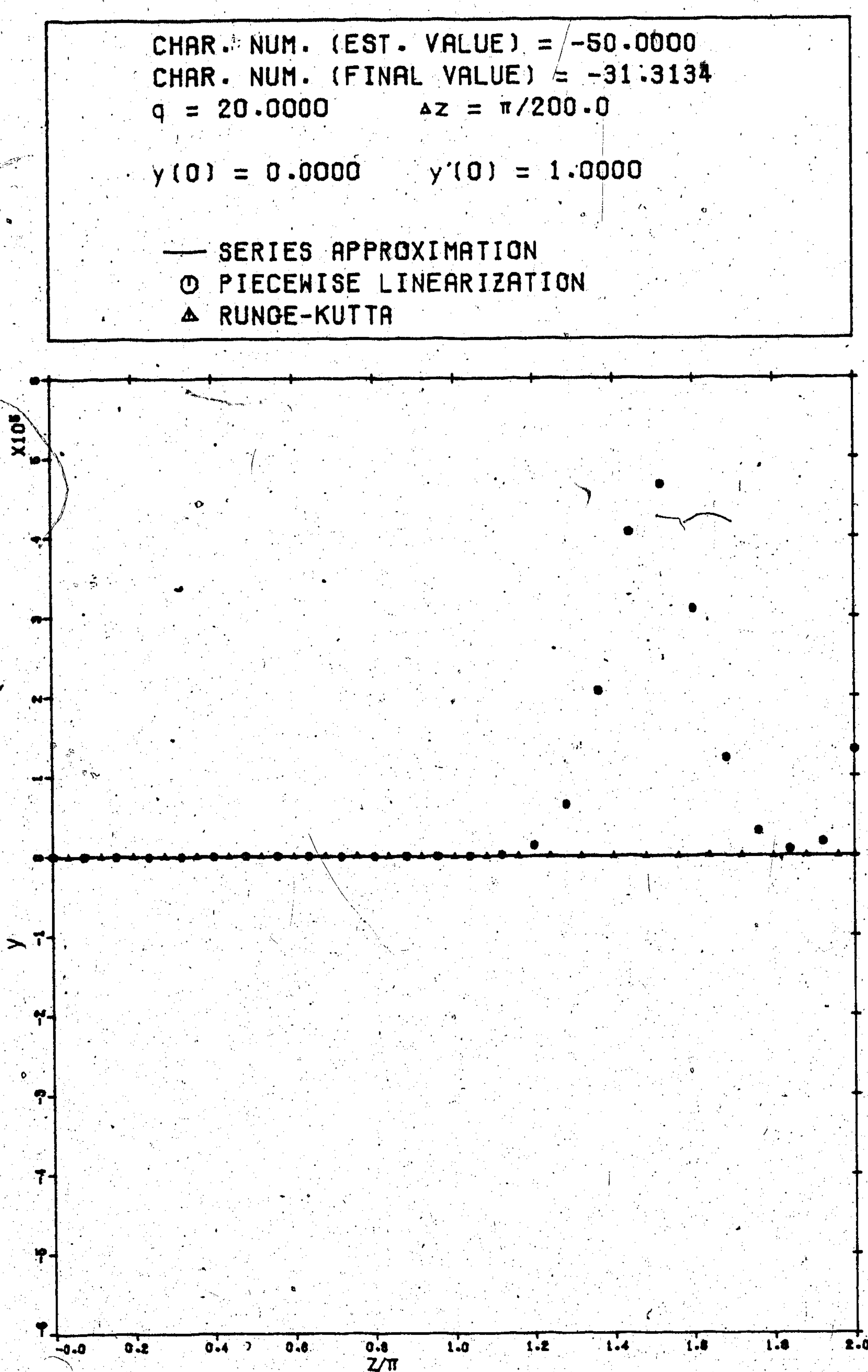
After breakdown, the solution appears to follow a pattern in the resulting curve rather than having the points randomly distributed. This may be an indication of "weak" instability [65], but because there was a good correspondence between the piecewise linearization and the series approximation to the exact solution, it cannot be the sole cause. The possible clue concerning "weak" instability [65] was obtained from the original plot from which Figure 6.9 was derived (Figure 7.1)

Since no single definition of instability fully describes the observed results, it appears that the reasons for the failure of the piecewise linear method are perhaps more complex than first imagined.

Figure 7.2 is a plot of the characteristic numbers obtained versus q, and the distinction is made between those values for which serious deviations occur and those for which the piecewise linear solution is reasonably well-behaved. There appears to be a transition zone between which the solutions go from stable to unstable. Unfortunately, there is not enough data available to give a better resolution of the boundaries, and so, Figure 7.2 has to be taken as a very rough estimate. What it

FIGURE 7.1 : SOLUTIONS TO MATHIEU EQUATION FOR $a_{GUESS} = -50.0$, $q = 20.0$, $\Delta z = \pi/200.00$, $y(0) = 0.0$, $y'(0) = 1.0$

(N.B.: Runge-Kutta solution uses [5].)



CHAR. NUM. (EST. VALUE) = -50.0000
CHAR. NUM. (FINAL VALUE) = -31.3134
q = 20.0000
Δz = π/200.0
y(0) = 0.0000
y'(0) = 1.0000
SERIES APPROXIMATION
PIECEWISE LINEARIZATION
RUNGE-KUTTA
y
Z/π

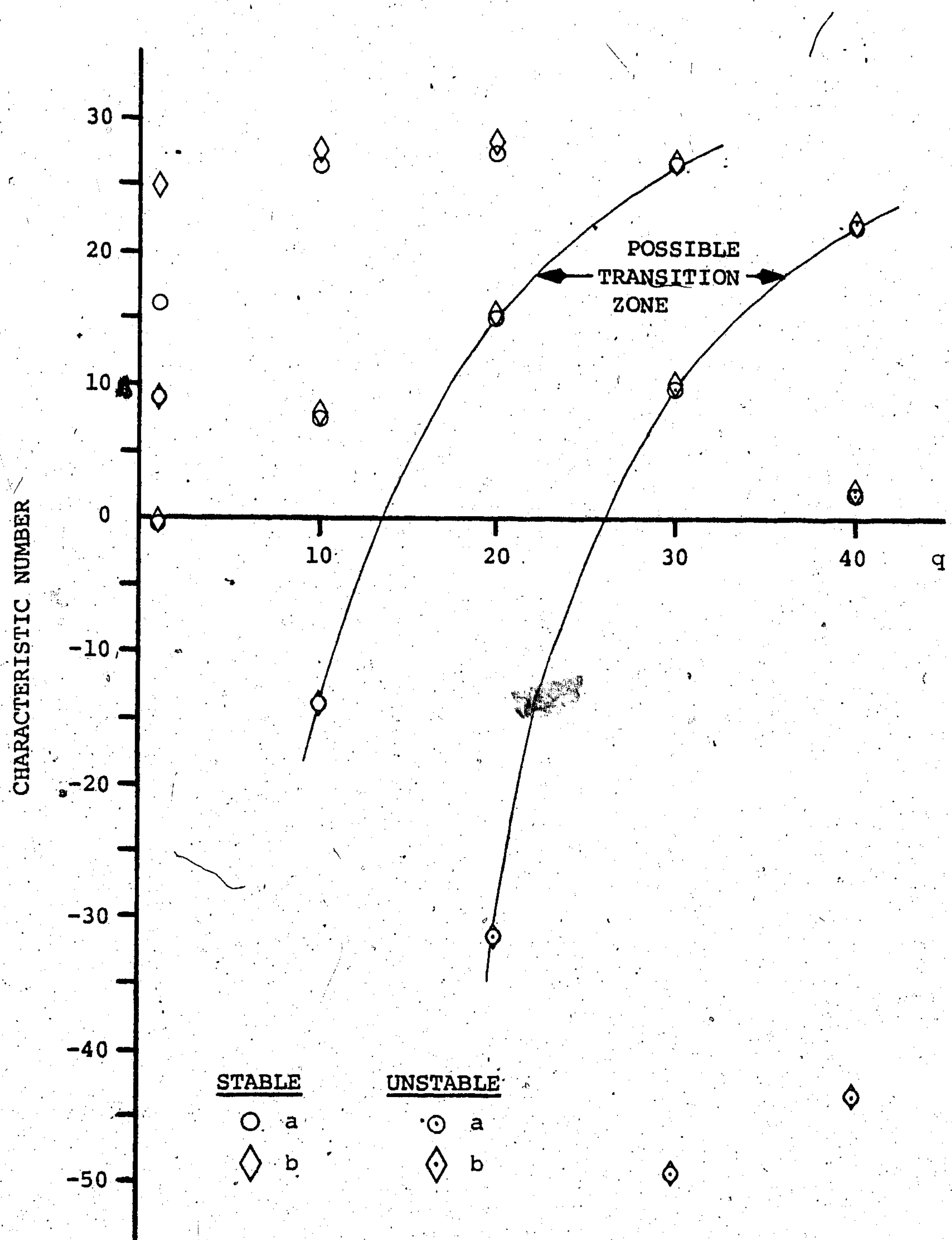


FIGURE 7.2 - STABILITY CHART FOR PIECEWISE LINEAR SOLUTION TO MATHIEU EQUATION

does indicate is that if this problem of the runaway piecewise linear solution cannot be resolved, this chart (or rather, a more accurate version of it) would give one an idea as to where the piecewise linear solution to the Mathieu equation becomes ineffective and another method should be pursued.

The timing results really posed no major problems as far as most of the values were concerned [27]. The piecewise linear method was fairly straightforward, and so, one can expect a linear solution, since as the number of increments is increased, the same calculations have to be done that many times more. The same applies to the series approximation to the exact solution, where the solution is mainly composed of the manipulation of the coefficient ratios.

The Runge-Kutta method [5] did give some unusual results. First, the magnitude of its times were suspect since one would expect the method to be faster than the others based on results for (3.1.1), (4.1.1), and (5.1.1). However, since the Mathieu equation is actually a linear one with a time-varying coefficient, it may be a difficult one for the Runge-Kutta routine [5] to manipulate. The piecewise linear method merely approximates the coefficient with a constant, and the series approximation to the exact solution is based upon the Mathieu functions.

The scatter in the timing results may have been a consequence of the size of the interval of z rather than the equation [60]. It appears that DVERK [5] itself may have been directly affected by the magnitude of Δz since (6.1.1) is a different type of equation than had been encountered in the previous chapters. How it is affected by it, and subsequent steps it executes could also explain why the execution times, as determined by use of [27], were so large compared with those for the others.

VIII. CONCLUSIONS

1. The accuracy of the piecewise linear solution with respect to the one being compared will vary with the equation being examined. This is dependent upon the presence of damping, the degree of nonlinearity (if any) and whether the coefficients in these equations are constant or variable.

2. For those cases in which correspondence between the piecewise linear solution and the solution being compared is poor, there appear to be areas for which the piecewise linear method is unstable, or at least limited. The degree to which this occurs will depend upon the equation parameters and constants, as well as the size of the increments chosen, whether displacement or time.

3. The type of instabilities displayed by the piecewise linear solutions may be equation-dependent. That is, the instability seen in the Van der Pol equation appeared to be different than that displayed by the Mathieu equation. This may have to be confirmed by further work with these two equations plus the investigation of others.

4. For large increments of time or displacement (depending upon which equation was being solved), the piecewise linear method can become uneconomical since it will have a larger execution time compared with a smaller increment, and quite often can exceed the time limit set for the program used. However, under certain circumstances, the execution time of the piecewise linear method can be close to that of the chosen standard solution (such as the Runge-Kutta method [5] used as a comparison for the Van der Pol equation), or even cheaper (as was seen for the undamped hard spring and Mathieu equation).

5. Generally, the piecewise linearization is slower than the solution being compared for the system being examined except in the case of the undamped hard spring and the Mathieu equations. For the former, it was because the approximation to the complete elliptic integral of the first kind [15] had to be calculated each time the number of segments was increased, while for the Mathieu equation it was because the piecewise linearization was simpler than the approximation to the exact solution and the Runge-Kutta method [5]. It can be seen here, then, that the method of approximating the equation will determine the speed (and hence, the cost) of a run.

6. The tangential method of linearization is slower than that using chords, because of the various calculations that have to be carried out. The correspondence with the approximation to the exact solution [4], [15], [17] appears to be better for chords than for tangents, though the latter is more accurate when it comes to calculating period for the undamped hard spring equation.

IX. RECOMMENDATIONS FOR FURTHER WORK

Several questions arising from the results obtained still need to be answered. At the same time, there are some areas that can be investigated in more detail, extending past what has already been examined. The following, then, are some suggested subjects for further study.

A. Mathieu's Equation

The first thing that comes to mind is the situation with the instability of the piecewise linear method. The exact nature as to what was occurring has to be resolved, since it is unusual that the piecewise linearization solution had behaved rather well until it apparently broke down. The actual mechanism behind this has to be examined in depth to determine not only what is taking place, but also why, and if there is any means of rectifying this.

On the other hand, suppose that the reason for the instability observed in Figures 6.7, 6.9, 6.11, 6.13, 6.14, and 7.1 cannot be resolved. One alternative is to try and pursue other methods of solving this equation, using a different means of approximating the linear segment. If, however, all attempts to solve this problem fail, the only other thing that can be suggested is to obtain more data and fill in the gaps in Figure 7.2, to see if a boundary between stability and instability can be clearly defined.

In McLachlan [67], reference was made to an earlier paper of his [68] that dealt with fractional orders for the solutions. This reference, in turn, referred to a paper by Ince on the same subject [69].

In [68], a method for determining the order of the solution for a particular a_{GUESS} and q is given, based on [69]. This may make the solution of (6.1.1) more general in nature compared with what had been done in Chgapter VI, as fractional orders can now be included.

B. Two-Degree-of-Freedom Nonlinear Equations

Since many physical phenomena can be described by two-degree-of-freedom nonlinear equations, one possible approximate method of solution is piecewise linearization. However, the major difficutly is in the separation of the coupled equations that describe the situation into separate modes.

Some work on this separation has already been done in this area by Rosenberg [70], [71], which deals with specific classes of two-degree-of-freedom nonlinear differential equations. He also produced a major work dealing with this subject, and it appears that this may be of a more general nature [72].

Since this subject was merely touched upon, some work would be required in seeing what is needed for solving specific cases. Once the equations have been uncoupled (if such is possible for the particular situation being examined), the piecewise linear method can then be applied. These references may provide a starting point.

REFERENCES

1. Cunningham, W. J. Introduction to Nonlinear Analysis. New York, McGraw-Hill Book Company, 1958, p. 135-140.
2. Ibid., p. 133-135.
3. Ibid., p. 123-133.
4. Madderom, P. "JCBYF" in UBC Function--Mathematical (or Special) Functions. Vancouver, B. C., University of British Columbia Computing Centre, June, 1979, p. 40-41.
5. "DVERK", in IMSL Library Reference Manual, Seventh Edition, Vol. 1. Houston, Tex., IMSL, Inc., Jan., 1979, DVERK1-DVERK9.
6. Cunningham, op. cit., p. 66.
7. This is based on a diagram found on p.25, ch. 2 of the course notes for ELEC 557 given from Sept., 1979 to Apr., 1980 by Dr. A. C. Soudack, Professor, Dept. of Electrical Engineering, University of British Columbia, Vancouver, B. C.
8. Cunningham, op. cit., p. 75.
9. McLachlan, N. W. Ordinary Non-Linear Differential Equations In Engineering and Physical Sciences. London, Oxford University Press, 1956, p. 24-27.
10. Cunningham, op. cit., p. 74.
11. Soudack, op. cit., p. 5-63 - 5-86.
12. Cunningham, op. cit., p. 77-78.
13. Soudack, op. cit., p. 5-84 - 5-85.
14. Milne-Thomson, L. M. "Elliptic Functions", in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, ed. M. Abramowitz and I. G. Stegun. New York, Dover

Publications, Inc., 1972, p. 590.

15. Ibid., p. 591.

16. Milne-Thomson, L. M. "Jacobian Elliptic Functions and Theta Functions", in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, ed. M. Abramowitz and I. G. Stegun. New York, Dover Publications, Inc., 1972, p. 571.

17. McLachlan, op. cit., p. 27.

18. Timoshenko, S., et. al. Vibration Problems in Engineering, Fourth Edition. New York, John Wiley & Sons. Inc., 1974, p. 190-193.

19. Ibid., p. 191.

20. Riddle, Douglas F. Calculus and Analytic Geometry. Belmont, Calif., Wadsworth Publishing Co., Inc., 1970, p. 6.

21. Timoshenko, et. al., op. cit., p. 192.

22. Franklin, Phillip. "Algebra", in Mark's Standard Handbook for Mechanical Engineering, Eighth Edition, ed. Theodore Baumeister, et. al. New York, McGraw-Hill Book Company, 1978, p. 2-19.

23. C R C Standard Mathematical Tables, 25th Edition, ed. William H. Beyer. Boca Raton, Florida, C R C Press, Inc., 1979, p. 170.

24. Riddle, op. cit., p. 21, 22.

25. University of Alberta Computing Services Overview 0120-0480, Computing Services. Edmonton, Alta., April 1980, p. 8.

26. University of Alberta Computing Services Bulletin, Edmonton, Alta., N1511.0681, June 5, 1981, p. 1.

27. University of Alberta Computing Services subroutine TIME. Its use was suggested by fellow graduate student, Roger Toogood, Dept. of Mechanical Engineering.

28. Based on a diagram in Soudack, op. cit., p. 48, ch. 3.

29. Timoshenko, et. al., op. cit., p. 63-66.

30. Greenspan, Donald. Discrete Numerical Methods in Physics and Engineering. New York, Academic Press, Inc., 1974, p. 15.

31. This time limit was set by the author through his on-campus terminal which was connected to the University of Alberta Computing Services facilities.

32. Soudack, op. cit., p. 69, 70, ch. 3.

33. Timoshenko, et. al., op. cit., p. 64.

34. Timoshenko, et. al., op. cit., p. 63-71.

35. Timoshenko, et. al., op. cit., p. 66.

36. Timoshenko, et. al., op. cit., p. 71.

37. Timoshenko, et. al., op. cit., p. 69.

38. Based on diagrams in Soudack, op. cit., p. 72, ch. 3.

39. Based on diagrams in Soudack, op. cit., p. 71, ch. 3.

40. University of Alberta Computing Services Reference R131.0280, CGPL/CGPL2. Edmonton, Alta., February 1980, p. 1-12.

41. McLachlan, N. W. Theory and Application of Mathieu Functions. Oxford, Clarendon Press, 1947, p. 10.

42. Ince, E. L. Tables of the Elliptic Cylinder Functions, Proc. Roy. Soc. Edinburgh, Vol. 52, 1932, p. 355-423.

43. McLachlan, Theory and Application of Mathieu Functions, p. 10-41.

44. Ibid., p. 18.

45. Ibid., p. 21.

46. Ibid., p. 28.

47. Ince, op. cit., p. 358.

48. Ince, op. cit., p. 359.

49. Ince, op. cit., p. 360.

50. Ince, op. cit., p. 363.

51. McLachlan, Theory and Application of Mathieu Functions, p. 37.

52. Ibid., p. 29-31.

53. Ibid., p. 31.

54. Ibid., p. 30.

55. Ibid., p. 37.

56. Ibid., p. 25.

57. Ince, op. cit. p. 364-375.

58. Timoshenko, et. al., op. cit., p. 3, 4.

59. Program MA1-11A, "Differential Equations", Hewlett-Packard HP-67/HP-97 Math Pac 1, Corvallis, Oregon, 1976, p. 11-01 - 11-04, L11-01 - L11-02.

60. This information was obtained through consultation with Dr. R. Torgerson, Senior Analyst, University of Alberta Computing Centre, Edmonton, Alta.

61. Greenspan, op. cit., p. 52.

62. Fröberg, Carl-Erik. Introduction to Numerical Analysis, Second Edition. Reading, Mass., Addison-Wesley Publishing Co., Inc., 1969, p. 277.

63. Dorn, William S. and McCracken, Daniel D. Numerical Methods with Fortran IV Case Studies. New York, John Wiley and Sons, Inc., 1972, p. 389.

64. Ibid., p. 378-379.

65. Mayers, D. F. "Stability of Step-by-Step Methods", in Numerical Solutions of Ordinary and Partial Differential Equations, ed. L. Fox. Reading, Mass., Addison-Wesley Publishing Co., Inc., 1962, p. 48.

66. Ibid., p. 51.

67. McLachlan, Theory and Application of Mathieu Functions, p. 79, 378.

68. McLachlan, N. W. "Computation of the Solutions of Mathieu's Equation", Philosophical Magazine, Vol. 36, Series 7, 1945, p. 403-414.

69. Ince, E. L. "Mathieu Functions of Stable Type", Philosophical Magazine, Vol. 6, Series 7, 1928, p. 547-558.

70. Rosenberg, R. M. "On Normal Vibrations of a General Class of Nonlinear Dual-Mode Systems", Journal of Applied Mechanics, Transactions of the ASME, Vol. 83, Series E, June, 1961, p. 275-283.

71. Rosenberg, R. M. "The Normal Modes of Nonlinear n-Degree-of-Freedom Systems", Journal of Applied Mechanics, Transactions of the ASME, Vol. 84, Series E, March, 1962, p. 7-14.

72. Rosenberg, R. M. "On Nonlinear Vibrations of Systems with Many Degrees of Freedom", Advances in Applied Mechanics, Vol. 9. New York, Academic Press, p. 155-242.

73. Buttuls, Peter and Dembo, Stevyn. University of Alberta Computing Services Tutorial T11.1277, Digital Plotting System. Edmonton, Alberta, University of Alberta Computing Services, December, 1977.

74. "Mathematical Notation - Special Functions", in McGraw-Hill Dictionary of Scientific and Technical Terms, Second Edition, ed. Daniel N. Lapedes. New York, McGraw-Hill Book Company, 1978, p. A9.

75. Cress, Paul, et. al. FORTRAN IV with WATFOR and WATFIV. Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1970.

APPENDIX

The following programs were used in obtaining the results for this thesis:

A. HSPRING - Hard spring equation (undamped)

B. HSPRINGC - Hard spring equation (linear damping)

C. HSPRINGQ - Hard spring equation (nonlinear damping)

D. VANDERPOL - Van der Pol equation

E. MATHIEU - Mathieu eauation.

These programs were developed and run on the facilities of the University of Alberta Computing Services [25], [26], with the plot routines used being [40], [73]. Material upon which these programs are based is found in:

HSPRING - [4], [8], [9], [10], [11], [12], [14], [15], [16], [18], [20], [22], [23], [24];

HSPRINGC - [5], [18], [20], [29], [30];

HSPRINGQ - [5], [18], [20], [29], [30], [74];

VANDERPOL - [5], [30], [32], [34], [38], [39], [74];

MATHIEU - [5], [34], [42], [43], [58].

The signum function as defined in [74] was used in HSPRINGQ and VANDERPOL. The major reference for the program code was [75].

```
C***********************************************************************
C*                                                                     *
C*     PROGRAM HSPRING                                                 *
C*                                                                     *
C*                                                                     *
C*                                                                     *
C*     THE FOLLOWING PROGRAM CALCULATES AND COMPARES THE SOLUTIONS     *
C*     FOR THE EQUATION:                                               *
C*                                                                     *
C*     X(DOUBLE-DOT)+A*X+B*(X**3)=0                                    *
C*                                                                     *
C*     OBTAINED BY PIECEWISE LINERIZATION (USING CHORDS AND            *
C*     TANGENTS) WITH THAT OF AN EXACT RESULT, AND PLOTTING ALL        *
C*     THREE SETS OF VALUES.                                           *
C*                                                                     *
C*                                                                     *
C*     THE RESULTS ARE FOUND FROM THE FOLLOWING APPROXIMATIONS:        *
C*                                                                     *
C*     EXACT:  X=XOE*CN(LAMBDA,OMEGAE*T)                               *
C*                                                                     *
C*     CHORDS:   X=XOSC+(XOC-XOSC)*COS(PC*(T-TOC))                     *
C*                 +(XDOTOC/PC)*SIN(PC*(T-TOC))                        *
C*                                                                     *
C*               X(DOT)=-PC*(XOC-XOSC)*SIN(PC*(T-TOC))                 *
C*                      +XDOTO*COS(PC*(T-TOC))                         *
C*                                                                     *
C*     TANGENTS:   X=XOST+(XOC1-XOST)*COS(PT*(T-TOT))                  *
C*                  +(XDOTOT/PT)*SIN(PT*(T-TOT))                       *
C*                                                                     *
C*                X(DOT)=-PT*(XOC1-XOST)*SIN(PT*(T-TOT))               *
C*                       +XDOTOT*COS(PT*(T-TOT))                       *
C*                                                                     *
C*     THE VARIOUS PARAMETERS ARE EXPLAINED AND CALCULATED IN THE      *
C*     PROGRAM.                                                        *
C*                                                                     *
C***********************************************************************
C
C
C
C
C
C
C
C
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 DELTAX(200),TAUC(200),TAUT(200)
      REAL*8 J,K,LAMBDA,M
      REAL*4 AMPLE(200),AMPLC(200),AMPLT(200),TE(200),TC(200),TT(200)
      REAL*4 YC(200),TCP(200),YT(200),TTP(200)
      REAL*4 XA(2),YA(2)
      REAL*4 HA,HB,HC,VA,VB,VC
      INTEGER*4 ALPH(20)
      COMMON/COEFF/A,B
      COMMON/ELLIP/PI,LAMBDA,K
      COMMON/ENDOC/XOC
      COMMON/END1C/X1C
      COMMON/CROSSC/XOSC
      COMMON/VIBC/PC
      COMMON/ENDOT/XOT
      COMMON/END1T/X1T
      COMMON/CROSST/XOST
      COMMON/SHARE/XOC2
      COMMON/VIBT/PT
C#######################################################################
C
C     THE FOLLOWING VALUES ARE READ FROM DATA FILE HSPRINGDATA:
C
C
C     A,B=AS IN THE EQUATION
C
C     DISPLO=INITIAL AMPLITUDE
C     VO=INITIAL VELOCITY
C     TO=INITIAL TIME
C
C     HA,HC=PLOT PARAMETERS FOR HORIZONTAL AXIS (HC AN INTEGRAL MULTIPLE
C           OF 8.0)
C
C     VC=PLOT PARAMETER FOR VERTICAL AXIS (AN INTEGRAL MULTIPLE OF 7.0,
C        7.0 GIVING BEST RESULTS)
C
C     XA,YA=PARAMETERS FOR PLOTTING ZERO LINE
C
C     ALPH=ARRAY FOR PLOT AXIS LABELS
C
C#######################################################################
      READ(5,10) A,B
   10 FORMAT(2D10.6)
      READ(5,20) DISPLO,VO,TO
   20 FORMAT(3D10.6)
      READ(5,30) HA,HC
   30 FORMAT(2F10.4)
      READ(5,40) VC
   40 FORMAT(F10.4)
      READ(5,50) XA(1)
   50 FORMAT(F10.4)
      READ(5,60) YA(1),YA(2)
   60 FORMAT(2F10.4)
      READ(5,70) (ALPH(I),I=1,12)
   70 FORMAT(12A4)
      READ(5,80) (ALPH(I),I=13,16)
   80 FORMAT(4A4)
      READ(5,90) (ALPH(I),I=17,20)
   90 FORMAT(4A4)
C
C
      PI=3.141592653589793
      BETA=B*(DISPLO**2)/A
C#######################################################################
```

POOR COPY
COPIE DE QUALITEE INFERIEURE

```
C
C     XOE=AMPLITUDE OF EXACT SOLUTION
C
C     LAMBDA=MODULUS (FOR EXACT SOLUTION)
C
C**********************************************************************
      XOE=DISPLO
      LAMBDA=DSQRT(0.5/((1.0/BETA)+1.0))
C**********************************************************************
C
C     ELLINT=SUBROUTINE FOR CALCULATING THE COMPLETE ELLIPTIC INTEGRAL
C           OF THE FIRST KIND
C
C**********************************************************************
      CALL ELLINT
C**********************************************************************
C
C     OMEGAE=EXACT SOLUTION ANGULAR FREQUENCY
C     TIME=CUMULATIVE TIME FOR EXACT SOLUTION
C     DELTAT=TIME INCREMENT
C
C**********************************************************************
      OMEGAE=DSQRT(A*(1.0+BETA))
      TIME=TO
      DELTAT=K/(100.0*DSQRT(A*(1.0+BETA)))
      DO 100 N=1,101
      U=OMEGAE*TIME
C**********************************************************************
C
C     DJCBYF=SUBROUTINE FOR EVALUATING JACOBIAN ELLIPTIC FUNCTIONS
C            (SEE UBC MANUAL "UBC FUNCTION" FOR DETAILS)
C
C**********************************************************************
      CALL DJCBYF(U,LAMBDA,DSN,DCN,DDN,IE,6)
C**********************************************************************
C
C     AMPLE=ARRAY FOR INSTANTANEOUS AMPLITUDE (FOR PLOTTING)
C     TE=ARRAY FOR TIME (FOR PLOTTING)
C
C**********************************************************************
      AMPLE(N)=XOE*DCN
      TE(N)=TIME
      TIME=TIME+DELTAT
  100 CONTINUE
C**********************************************************************
C
C     TAUE=EXACT PERIOD
C     BIG=LARGEST VALUE OF QUARTER-PERIOD (FOR PLOT SCALING)
C
C**********************************************************************
      TAUE=4.0*K/DSQRT(A*(1.0+BETA))
      BIG=TAUE/4.0
C**********************************************************************
C
C     THE FOLLOWING DO-LOOP (ENDING AT STEP 250) CALCULATES THE
C     PIECEWISE LINEAR PERIODS (FOR BOTH METHODS) BASED ON THE
C     PARTICULAR NUMBER OF SEGMENTS CHOSEN.
C
C
C
C     NC=NUMBER OF CHORDS (INITIALLY SET AT 5)
C     NT=NUMBER OF TANGENTS (INITIALLY SET AT 6)
C
C     XOC,XOT=INITIAL DISPLACEMENT FOR EACH SUCCESSIVE SEGMENT FOR
C             CHORDS AND TANGENTS, RESPECTIVELY
C
C     XDOTOC,XDOTOT=INITIAL VELOCITY FOR EACH SUCCESSIVE SEGMENT, AS
C                   ABOVE
C     TOC,TOT=INITIAL TIME, FOR CHORDS AND TANGENTS, RESPECTIVELY
C
C     XOC1=INITIAL CROSSING POINT FOR TANGENT LINES OF TWO ADJACENT
C          SEGMENTS (SEE SUBROUTINE XSHARE BELOW)
C
C     DELTA=LENGTH OF SEGMENT
C     DELTAX=ARRAY FOR DELTA
C
C**********************************************************************
      NC=5
      DO 250 I=1,20
      NT=NC+1
      H=NC
      XOC=DISPLO
      XOT=DISPLO
      XDOTOC=VO
      XDOTOT=VO
      TOC=TO
      TOT=TO
      XOC1=DISPLO
      DELTA=DISPLO/H
      DELTAX(I)=DELTA
C**********************************************************************
C
C     THE FOLLOWING SECTION CALCULATES THE PERIOD FOR THE METHOD OF
C     CHORDS (ENDING AT STEP 110).
C
C
C
C     X1C=DISPLACEMENT AT END OF SEGMENT
C
C**********************************************************************
      X1C=XOC-DELTA
      DO 110 N=1,NC
C**********************************************************************
C
C     XPC=SUBROUTINE FOR CALCULATING POINT WHERE SEGMENT CHORD LINE
C         CROSSES X-AXIS
C
C     FREQC=SUBROUTINE FOR CALCULATING ANGULAR FREQUENCY
C
C**********************************************************************
```

```
      CALL XPC
      CALL FREQC
C**********************************************************************
C
C     THE FOLLOWING SOLVES THE TRIGONOMETRIC EQUATION:
C
C     C3C=C1C*COS(THETAC)+C2C*SIN(THETAC)
C
C     WHERE:
C
C     THETAC=PC(T1C-TOC)
C
C     T1C BEING THE CUMULATIVE TIME TAKEN TO REACH THE END OF THE
C     SEGMENT BEING EXAMINED.
C
C**********************************************************************
      C1C=XOC-XOSC
      C2C=XDOTOC/PC
      C3C=X1C-XOSC
      THETAC=DATAN(C2C/C1C)+DARCOS(C3C/DSQRT(C1C**2+C2C**2))
      T1C=TOC+THETAC/PC
C**********************************************************************
C
C     AMPLC=ARRAY FOR XOC (FOR PLOTTING)
C     TC=ARRAY FOR TOC (FOR PLOTTING)
C
C**********************************************************************
      AMPLC(N)=XOC
      TC(N)=TOC
C**********************************************************************
C
C     RESETTING THE INITIAL CONDITIONS FOR THE NEXT SEGMENT.
C
C**********************************************************************
      XOC=X1C
      X1C=XOC-DELTA
      XDOTOC=-PC*(C1C*DSIN(THETAC)-C2C*DCOS(THETAC))
      TOC=T1C
  110 CONTINUE
C**********************************************************************
C
C     TAUC=ARRAY FOR PERIOD FOUND USING CHORDS
C
C**********************************************************************
      TAUC(I)=4.0*T1C
      IF(T1C.GE.BIG) BIG=TIC
      NDC=NC+1
      AMPLC(NDC)=XOC
      TC(NDC)=TOC
C**********************************************************************
C
C     THE FOLLOWING SECTION CALCULATES THE PERIOD FOR THE METHOD OF
C     TANGENTS (ENDING AT STEP 120).
C
C
C
C     X1T=DISPLACEMENT AT END OF SEGMENT
C
C
C**********************************************************************
      X1T=XOT-DELTA
      DO 120 N=1,NT
C**********************************************************************
C
C     XSHARE=SUBROUTINE FOR CALCULATING THE CROSSING POINT FOR TANGENT
C            LINES OF TWO ADJACENT SEGMENTS
C
C     XPT=SUBROUTINE FOR CALCULATING POINT WHERE SEGMENT TANGENT LINE
C         CROSSES X-AXIS
C
C     FREQT=SUBROUTINE FOR CALCULATING ANGULAR FREQUENCY
C
C**********************************************************************
      CALL XSHARE
      CALL XPT
      CALL FREQT
C**********************************************************************
C
C     THE FOLLOWING SOLVES THE TRIGONOMETRIC EQUATION:
C
C     C3T=C1T*COS(THETAC)+C2T*SIN(THETAC)
C
C     WHERE:
C
C     THETAT=PT(T1T-TOT)
C
C     T1T BEING THE CUMULATIVE TIME TAKEN TO REACH THE END OF THE
C     SEGMENT BEING EXAMINED.
C
C**********************************************************************
      C1T=XOC1-XOST
      C2T=XDOTOT/PT
      C3T=XOC2-XOST
      IF(DABS(XOT).LT.1.0D-10) C3T=-XOST
      THETAT=DATAN(C2T/C1T)+DARCOS(C3T/DSQRT(C1T**2+C2T**2))
      T1T=TOT+THETAT/PT
C**********************************************************************
C
C     AMPLT=ARRAY FOR XOC1 (FOR PLOTTING)
C     TT=ARRAY FOR TOT (FOR PLOTTING)
C
C**********************************************************************
      AMPLT(N)=XOC1
      TT(N)=TOT
C**********************************************************************
C
C     RESETTING THE INITIAL CONDITIONS FOR THE NEXT SEGMENT.
C
C**********************************************************************
      XOT=X1T
```

```
      X1T=XOT-DELTA
      XDOTOT=-PT*(C1T*DSIN(THETAT)-C2T*DCOS(THETAT))
      TOT=T1T
      XOC1=XOC2
  120 CONTINUE
C###################################################################
C
C     TAUT=PERIOD FOUND USING TANGENTS
C
C###################################################################
      TAUT(I)=4.0*T1T
      IF(T1T.GE.BIG) BIG=T1T
      NDT=NT+1
      AMPLT(NDT)=XOC1
      TT(NDT)=TOT
C###################################################################
C
C     THE NEXT FEW LINES ARE USED IN FINALIZING ALL THE PLOT PARAMETERS
C     WHICH INCLUDES AXIS SCALING.
C
C###################################################################
      M=1
      IF(DISPLO.LE.1.0) GO TO 140
  130 CONTINUE
      DECD=DISPLO/(10.0**M)
      IF(DECD.LE.1.0) GO TO 160
      M=M+1
      GO TO 130
  140 CONTINUE
      DIGD=DISPLO*(10.0**M)
      IF(DIGD.GE.1.0) GO TO 150
      M=M+1
      GO TO 140
  150 CONTINUE
      DECD=DIGD/10.0
      M=-(M-1)
  160 CONTINUE
      IF(DECD.LE.0.1) VFACT=0.1
      IF((DECD.GT.0.1).AND.(DECD.LE.0.25)) VFACT=0.25
      IF((DECD.GT.0.25).AND.(DECD.LE.0.5)) VFACT=0.5
      IF((DECD.GT.0.5).AND.(DECD.LE.1.0)) VFACT=1.0
      VB=(VFACT/(VC-2.0))*10.0**M
      VA=-VB
C
      M=1
      IF(BIG.LE.1.0) GO TO 180
  170 CONTINUE
      DECT=BIG/(10.0**M)
      IF(DECT.LE.1.0) GO TO 200
      M=M+1
      GO TO 170
  180 CONTINUE
      DIGT=BIG*(10.0**M)
      IF(DIGT.GE.1.0) GO TO 190
      M=M+1
      GO TO 180
  190 CONTINUE
      DECT=DIGT/10.0
      M=-(M-1)
  200 CONTINUE
      IF(DECT.LE.0.08) HFACT=0.1
      IF((DECT.GT.0.08).AND.(DECT.LE.0.16)) HFACT=0.2
      IF((DECT.GT.0.16).AND.(DECT.LE.0.4)) HFACT=0.5
      IF((DECT.GT.0.4).AND.(DECT.LE.0.8)) HFACT=1.0
      IF((DECT.GT.0.8).AND.(DECT.LE.1.0)) HFACT=2.0
      HB=HFACT*10.0**(M-1)
      LONG=HC
      XA(2)=DFLOAT(LONG)*HB-0.005
C###################################################################
C
C     THE FOLLOWING SECTION SETS UP THE CHORD AND TANGENT RESULTS SO
C     THAT 6 POINTS OF THE CHORD SOLUTION AND 7 OF THE TANGENT VALUES
C     ARE PLOTTED FOR A CLEARER PRESENTATION, WITH THE PLOTS BEING
C     ALTERNATED FOR COMPARISON PURPOSES.
C
C     YC=ARRAY FOR PLOTTED CHORD DISPLACEMENTS
C     TCP=ARRAY FOR PLOTTED CHORD TIME VALUES
C     YT=ARRAY FOR PLOTTED TANGENT DISPLACEMENTS
C     TTP=ARRAY FOR PLOTTED TANGENT TIME VALUES
C
C###################################################################
      INCC=NDC/5
      INCT=NDT/5
      INCRT=INCT/2
      IF(NDT.LT.10) INCRT=1
      NCC=1
      NCT=1
      NPC=1
      NPT=1
  210 CONTINUE
      YC(NPC)=AMPLC(NCC)
      TCP(NPC)=TC(NCC)
      IF(NCC.EQ.NDC) GO TO 220
      NPC=NPC+1
      NCC=NCC+INCC
      GO TO 210
  220 CONTINUE
      YT(NPT)=AMPLT(NCT)
      TTP(NPT)=TT(NCT)
      IF(NCT.EQ.1) GO TO 230
      NCT=NCT+INCT
      GO TO 240
  230 CONTINUE
      NCT=NCT+INCRT
  240 CONTINUE
      IF(NCT.GE.NDT) GO TO 250
      NPT=NPT+1
      GO TO 220
  250 CONTINUE
      NPT=NPT+1
```

```
      YT(NPT)=AMPLT(NDT)
      TTP(NPT)=TT(NDT)
C*************************************************************************
C
C     PLOTTING THE SOLUTIONS.  CONSULT THE WRITEUP ON CGPL/CGPL2 AND
C     THE MANUAL ON DIGITAL PLOTTING FOR DETAILS.
C
C*************************************************************************
      ND=101
      NF=1
      CALL CGPL2(TE,AMPLE,ND,NF,5,HA,HB,HC,VA,VB,VC,ALPH)
      ND=NPC
      CALL CGPL2(TCP,YC,ND,2,1,HA,HB,HC,VA,VB,VC,ALPH)
      ND=NPT
      CALL CGPL2(TTP,YT,ND,3,1,HA,HB,HC,VA,VB,VC,ALPH)
      NF=4
      CALL CGPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
      HORIZ=HA
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=HC
      CALL PLOT(HORIZ,VERT,2)
      VERT=VERT+6.0
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=HA
      CALL PLOT(HORIZ,VERT,2)
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,2)
      STARTX=(HC-7.0)/2.0
      STARTY=VC+0.5
      HORIZ=STARTX+0.2
      VERT=STARTY+4.6
      CALL SYMBOL(HORIZ,VERT,0.2,'A =',0.0,3)
      HORIZ=STARTX+1.0
      CALL NUMBER(HORIZ,VERT,0.2,A,0.0,4)
      HORIZ=STARTX+3.5
      CALL SYMBOL(HORIZ,VERT,0.2,'B =',0.0,3)
      HORIZ=STARTX+4.3
      CALL NUMBER(HORIZ,VERT,0.2,B,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+3.8
      CALL SYMBOL(HORIZ,VERT,0.2,'NO. OF SEGMENTS (APPROX. TO EXACT',0.0
     1,33)
      HORIZ=STARTX+0.6
      VERT=STARTY+3.6
      CALL SYMBOL(HORIZ,VERT,0.2,'SOLN.) = 100',0.0,12)
      HORIZ=STARTX+0.2
      VERT=STARTY+3.2
      CALL SYMBOL(HORIZ,VERT,0.2,'NO. OF CHORDS =',0.0,15)
      HORIZ=STARTX+3.4
      CPOINT=DFLOAT(NC)
      CALL NUMBER(HORIZ,VERT,0.2,CPOINT,0.0,-1)
      HORIZ=STARTX+0.2
      VERT=STARTY+2.6
      CALL SYMBOL(HORIZ,VERT,0.2,'NO. OF TANGENTS =',0.0,17)
      HORIZ=STARTX+3.8
      TPOINT=DFLOAT(NT)
      CALL NUMBER(HORIZ,VERT,0.2,TPOINT,0.0,-1)
      HORIZ=STARTX+0.2
      VERT=STARTY+2.0
      CALL SYMBOL(HORIZ,VERT,0.2,'X(0) =',0.0,6)
      HORIZ=STARTX+1.6
      CALL NUMBER(HORIZ,VERT,0.2,DISPLO,0.0,4)
      HORIZ=STARTX+3.5
      CALL SYMBOL(HORIZ,VERT,0.2,'X(0) =',0.0,6)
      HORIZ=STARTX+3.525
      VERT=STARTY+2.3
      CALL SYMBOL(HORIZ,VERT,0.2,75,0.0,-1)
      HORIZ=STARTX+4.9
      VERT=STARTY+2.0
      CALL NUMBER(HORIZ,VERT,0.2,VO,0.0,4)
      HORIZ=STARTX+0.3
      VERT=STARTY+1.1
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=STARTX+0.8
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+1.0
      VERT=STARTY+1.0
      CALL SYMBOL(HORIZ,VERT,0.2,'APPROX. TO EXACT SOLN.',0.0,22)
      HORIZ=STARTX+0.6
      VERT=STARTY+0.7
      CALL SYMBOL(HORIZ,VERT,0.2,1,0.0,-1)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.6
      CALL SYMBOL(HORIZ,VERT,0.2,'CHORDS',0.0,6)
      HORIZ=STARTX+0.8
      VERT=STARTY+0.3
      CALL SYMBOL(HORIZ,VERT,0.2,2,0.0,-1)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.2
      CALL SYMBOL(HORIZ,VERT,0.2,'TANGENTS',0.0,8)
      NF=0
      CALL CGPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
C*************************************************************************
C
C     INCREMENTING THE NUMBER OF LINEAR SEGMENTS.
C
C*************************************************************************
      NC=NC+5
  250 CONTINUE
C*************************************************************************
C
C     THE CALCULATED VALUES ARE WRITTEN INTO OUTPUT FILE HSPRINGNUM.
C
C*************************************************************************
      WRITE(10,270)
  270 FORMAT('1','PERIODS FOR THE EQUATION:')
      WRITE(10,280)
  280 FORMAT('0','X(DOUBLE-DOT)+A*X+B*(X**3)=0')
      WRITE(10,290)
```

```
  280 FORMAT('0','OBTAINED BY PIECEWISE LINEARIZATION')
      WRITE(10,300) A,B
  300 FORMAT('0','A=',G20.10,2X,'B=',G20.10)
      WRITE(10,310) DISPLO,VO
  310 FORMAT('0','X(0)=',G20.10,2X,'X(DOT)(0)=',G20.10)
      WRITE(10,320) TAUE
  320 FORMAT('0','EXACT SOLUTION PERIOD=',G20.10)
      WRITE(10,330)
  330 FORMAT('-',8X,'DELTAX',16X,'CHORDS',15X,'TANGENTS')
      DO 350 I=1,20
      WRITE(10,340) DELTAX(I),TAUC(I),TAUT(I)
  340 FORMAT(1X,3(G20.10,2X))
  350 CONTINUE
      STOP
      END
C
C
C
C
C
C
C
C
C
C
C*******************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE FIRST ONE HUNDRED TERMS OF
C     AN INFINITE SERIES APPROXIMATION OF THE COMPLETE ELLIPTIC INTEGRAL
C     OF THE FIRST KIND.  SEE SECT. 17.3, P. 591, "HANDBOOK OF
C     MATHEMATICAL FUNCTIONS WITH FORMULAS, GRAPHS, AND MATHEMATICAL
C     TABLES", M. ABRAMOWITZ AND I. G. STEGUN, FOR DETAILS.
C
C     K=CALCULATED VALUE
C
C*******************************************************************
      SUBROUTINE ELLINT
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 J,K,LAMBDA,M
      COMMON/ELLIP/PI,LAMBDA,K
      M=LAMBDA**2
      COEFF=1.0/2.0
      SUM=0.25*M
      DO 300 N=1,98
      J=N+1.0
      COEFF=COEFF*(2.0*J-1.0)/(2.0*J)
      TERM=(COEFF**2)*(M**(N+1))
      SUM=SUM+TERM
  300 CONTINUE
      K=(PI/2.0)*(1.0+SUM)
      RETURN
      END
C
C
C
C*******************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE POINT WHERE THE INTERVAL
C     CHORD LINE WOULD CROSS THE X-AXIS.
C
C     PC=CALCULATED VALUE
C     XOSC=CALCULATED VALUE
C
C*******************************************************************
      SUBROUTINE XPC
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/COEFF/A,B
      COMMON/ENDOC/XOC
      COMMON/END1C/X1C
      COMMON/CROSSC/XOSC
      FO=A*XOC+B*(XOC**3)
      F1=A*X1C+B*(X1C**3)
      XOSC=(X1C-(F1/FO)*XOC)/(1-(F1/FO))
      RETURN
      END
C
C
C
C*******************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE ANGULAR FREQUENCY FOR
C     AN INTERVAL USING CHORDS.
C
C*******************************************************************
      SUBROUTINE FREQC
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/COEFF/A,B
      COMMON/ENDOC/XOC
      COMMON/END1C/X1C
      COMMON/VIBC/PC
      FO=A*XOC+B*(XOC**3)
      F1=A*X1C+B*(X1C**3)
      SLOPE=(FO-F1)/(XOC-X1C)
      PC=DSQRT(SLOPE)
      RETURN
      END
C
C
C
C*******************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE POINT WHERE THE TANGENT
C     LINES OF TWO ADJACENT INTERVALS WOULD CROSS.
C
C     XOC2=CALCULATED VALUE
C
C*******************************************************************
      SUBROUTINE XSHARE
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/COEFF/A,B
```

```
      COMMON/ENDOT/XOT
      COMMON/END1T/X1T
      COMMON/SHARE/XOC2
      IF(DABS(XOT).LT.1.0D-10) GO TO 400
      SLOPE0=A+3.0*B*(XOT**2)
      SLOPE1=A+3.0*B*(X1T**2)
      F0=A*XOT+B*(XOT**3)
      F1=A*X1T+B*(X1T**3)
      XOC2=(SLOPE0*XOT-SLOPE1*X1T-(F0-F1))/(SLOPE0-SLOPE1)
      GO TO 410
  400 CONTINUE
      XOC2=0.D0
  410 CONTINUE
      RETURN
      END
C
C
C
C**********************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE POINT WHERE THE INTERVAL
C     TANGENT LINE WOULD CROSS THE X-AXIS.
C
C     XOST=CALCULATED VALUE
C
C**********************************************************
      SUBROUTINE XPT
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/COEFF/A,B
      COMMON/ENDOT/XOT
      COMMON/END1T/X1T
      COMMON/CROSST/XOST
      F=A*XOT+B*(XOT**3)
      SLOPE=A+3.0*B*(XOT**2)
      XOST=XOT-F/SLOPE
      RETURN
      END
C
C
C
C**********************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE ANGULAR FREQUENCY FOR
C     AN INTERVAL USING TANGENTS.
C
C     PT=CALCULATED VALUE
C
C**********************************************************
      SUBROUTINE FREQT
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/COEFF/A,B
      COMMON/ENDOT/XOT
      COMMON/VIBT/PT
      SLOPE=A+3.0*B*(XOT**2)
      PT=DSQRT(SLOPE)
      RETURN
      END
END OF FILE
```

```
C*********************************************************************
C*                                                                   *
C*   PROGRAM HSPRINGC                                                *
C*                                                                   *
C*                                                                   *
C*                                                                   *
C*   THE FOLLOWING PROGRAM CALCULATES THE SOLUTION FOR THE           *
C*   EQUATION:                                                       *
C*                                                                   *
C*   X(DOUBLE-DOT)+C*X(DOT)+A*X+B*(X**3)=0                           *
C*                                                                   *
C*   OBTAINED BY PIECEWISE LINEARIZATION USING CHORDS, AS WELL AS    *
C*   FINDING THE FIRST TROUGH ON THE DISPLACEMENT-TIME CURVE.        *
C*                                                                   *
C*                                                                   *
C*   THE RESULTS ARE FOUND FROM THE FOLLOWING:                       *
C*                                                                   *
C*   X=XSTAR+EXP(-N*(T-TO))*(C1*COS(PSTAR*(T-TO))                    *
C*                           +C2*SIN(PSTAR*(T-TO)))                  *
C*                                                                   *
C*   X(DOT)=EXP(-N*(T-TO))*(C3*COS(PSTAR*(T-TO)                      *
C*                         +C4*SIN(PSTAR*(T-TO)))                    *
C*                                                                   *
C*   C1=XO-XSTAR                                                     *
C*   C2=(N*(XO-XSTAR)+XDOTO)/P                                       *
C*   C3=-N*C1+P*C2                                                   *
C*   C4=-P*C1-N*C2                                                   *
C*                                                                   *
C*   THE VARIOUS PARAMETERS ARE EXPLAINED AND CALCULATED IN THE      *
C*   PROGRAM.                                                        *
C*                                                                   *
C*                                                                   *
C*   THE PROGRAM ALSO SOLVES THE EQUATION USING FIFTH- AND SIXTH-    *
C*   ORDER RUNGE-KUTTA METHODS.  BOTH SOLUTIONS ARE PLOTTED.         *
C*                                                                   *
C*********************************************************************
C
C
C
C
C
C
C
C
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL FCN
      REAL*8 Y(2),CN(24),W(2,20)
      REAL*8 N
      REAL*4 AMPLP(200),AMPLRK(200),TP(200),TRK(200)
      REAL*4 YP(2000),TPP(200)
      REAL*4 XA(2),YA(2)
      REAL*4 HA,HB,HC,VA,VB,VC
      INTEGER*4 ALPH(20)
      COMMON/COEFF/A,B
      COMMON/DAMP/C
C####################################################################
C
C     THE FOLLOWING VALUES ARE READ FROM DATA FILE HSPRINGCDATA:
C
C
C     A,B,C=SAME AS ABOVE
C
C     DISPLO=INITIAL AMPLITUDE
C     VO=INITIAL VELOCITY
C     TIMEO=INITIAL TIME
C
C     H=NUMBER OF DIVISIONS OF INITIAL DISPLACEMENT
C     DTIME=TIME INCREMENT (RUNGE-KUTTA SOLUTION)
C
C     HA,HC=PLOT PARAMETERS FOR HORIZONTAL AXIS (HC AN INTEGRAL MULTIPLE
C           OF 7.0)
C
C     VC=PLOT PARAMETER FOR VERTICAL AXIS (AN INTEGRAL MULTIPLE OF 12.0,
C        12.0 GIVING BEST RESULTS)
C
C     XA,YA=PARAMETERS FOR PLOTTING ZERO LINE
C
C     ALPH=ARRAY FOR PLOT AXIS LABELS
C
C####################################################################
      READ(5,10) A,B,C
   10 FORMAT(3D10.5)
      READ(5,20) DISPLO,VO,TIMEO
   20 FORMAT(3D10.5)
      READ(5,30) H,DTIME
   30 FORMAT(2D10.5)
      READ(5,40) HA,HC
   40 FORMAT(2F10.4)
      READ(5,50) VC
   50 FORMAT(F10.4)
      READ(5,60) XA(1)
   60 FORMAT(F10.4)
      READ(5,70) YA(1),YA(2)
   70 FORMAT(2F10.4)
      READ(5,80) (ALPH(I),I=1,12)
   80 FORMAT(12A4)
      READ(5,90) (ALPH(I),I=13,16)
   90 FORMAT(4A4)
      READ(5,100) (ALPH(I),I=17,20)
  100 FORMAT(4A4)
C####################################################################
C
C     N=DAMPING FACTOR
C
C####################################################################
      N=C/2.0
C####################################################################
C
```

```
C     SETTING THE INITIAL CONDITIONS.
C
C     X0=INITIAL DISPLACEMENT FOR EACH SUCCESSIVE SEGMENT
C     XDOT0=INITIAL VELOCITY FOR EACH SEGMENT
C     T0=CUMULATIVE TIME TAKEN TO REACH BEGINNING OF SEGMENT
C     DELT=INITIAL GUESS AT TIME INTERVAL FOR THE SEGMENT BEING EXAMINED
C          (SEE SUBROUTINE TIME BELOW)
C
C     DELTAX=LENGTH OF SEGMENT
C
C     I=COUNTER FOR DATA POINTS TO BE PLOTTED
C
C**********************************************************************
      X0=DISPL0
      XDOT0=V0
      T0=TIME0
      DELT=T0
      DELTAX=DISPL0/N
      DELX=DELTAX
      I=0
C**********************************************************************
C
C     THE EQUATION PARAMETERS ARE WRITTEN INTO OUTPUT FILE HSPRINGCNUM.
C
C**********************************************************************
      WRITE(10,110)
  110 FORMAT('1','PIECEWISE LINEAR PERIOD FOR THE EQUATION:')
      WRITE(10,120)
  120 FORMAT('0','X(DOUBLE-DOT)+C*X(DOT)+A*X+B*(X**3)=0')
      WRITE(10,130) A,B,C
  130 FORMAT('0','A=',G20.10,2X,'B=',G20.10,2X,'C=',G20.10)
      WRITE(10,140) DISPL0,V0
  140 FORMAT('0','X(0)=',G20.10,2X,'X(DOT)(0)=',G20.10)
      WRITE(10,150)
  150 FORMAT('-',8X,'AMPLITUDE',10X,'CUMULATIVE TIME')
  160 CONTINUE
      I=I+1
C**********************************************************************
C
C     X1=DISPLACEMENT AT END OF SEGMENT
C
C     XPRIME=SUBROUTINE FOR CALCULATING POINT WHERE SEGMENT CHORD LINE
C          CROSSES X-AXIS
C     XSTAR=CALCULATED VALUE
C
C     FREQ=SUBROUTINE FOR CALCULATING ANGULAR FREQUENCY
C     PSTAR=CALCULATED VALUE
C
C     TIME=SUBROUTINE FOR CALCULATING TIME INTERVAL FOR A SEGMENT
C     DELTAT=CALCULATED VALUE
C
C     SPEED=SUBROUTINE FOR CALCULATING SPEED AT END OF SEGMENT
C     XDOT1=CALCULATED VALUE
C
C**NOTE**THE PROGRAM STOPS IF THE DISPLACEMENT AT THE END OF A SEGMENT
C          IS EQUAL TO -DISPL (I. E., UNDAMPED CASE).
C
C**********************************************************************
      X1=X0-DELTAX
      CALL XPRIME(X0,X1,XSTAR)
      CALL FREQ(X0,X1,N,PSTAR)
      CALL TIME(X0,XDOT0,X1,DELT,XSTAR,N,PSTAR,DELTAT)
      CALL SPEED(X0,XDOT0,XSTAR,N,DELTAT,PSTAR,XDOT1)
C**********************************************************************
C
C     STARTING AT STEP 160 AND CONTINUING AT 200, THE PROGRAM TESTS FOR
C     ZERO VELOCITY.  IF IT OCCURS AT THE END OF THE SEGMENT, THE
C     PROGRAM STOPS (SEE STEP 300).  IF NOT, THE LOCATION OF THE FIRST
C     TROUGH IS SOUGHT (STEPS 250 TO 270).
C
C**********************************************************************
      DELT=DELTAT
      T1=T0+DELTAT
      IF(X0.GE.0.0) GO TO 170
      GO TO 190
  170 CONTINUE
C**********************************************************************
C
C     AMPLP=ARRAY FOR X0
C     TP=ARRAY FOR T0
C
C**********************************************************************
      AMPLP(I)=X0
      TP(I)=T0
C**********************************************************************
C
C     THE INTERIM VALUES ARE WRITTEN INTO OUTPUT FILE HSPRINGCNUM.
C
C**********************************************************************
      WRITE(10,180) X0,T0
  180 FORMAT(1X,2(G20.10,2X))
C**********************************************************************
C
C     RESETTING THE INITIAL CONDITIONS.
C
C**********************************************************************
      X0=X1
      XDOT0=XDOT1
      T0=T1
      GO TO 160
  190 CONTINUE
      IF(DABS(XDOT1).LT.1.0D-06) GO TO 200
      GO TO 170
  200 CONTINUE
C**********************************************************************
C
C     THE SEGMENT IN WHICH THE FIRST TROUGH OCCURS IS WRITTEN INTO
C     OUTPUT FILE HSPRINGCNUM.
C
C**********************************************************************
```

```
      AMPLP(I)=XO
      TP(I)=TO
      WRITE(10,210) XO,TO
  210 FORMAT(1X,2(G20.10,2X))
      DIFF=-DISPLO-XO
      IF(DABS(DIFF).LT.1.0D-08) GO TO 300
      WRITE(10,220) X1,T1
  220 FORMAT(1X,2(G20.10,2X))
      WRITE(10,230)
  230 FORMAT('-','TROUGH ITERATIONS')
      WRITE(10,240)
  240 FORMAT('0',10X,'XO',20X,'X1',14X,'X1(CALCULATED)')
C*********************************************************************
C
C     THE FOLLOWING (ENDING AT STEP 270) IS THE CALCULATION OF THE FIRST
C     TROUGH.  USING THE TIME INTERVAL REQUIRED TO REACH ZERO VELOCITY,
C     A NEW VALUE OF X1 (XF) IS CALCULATED AND COMPARED TO THE PREVIOUS
C     VALUE.  THIS NEW VALUE BECOMES THE NEW X1 IF THE DIFFERENCE
C     BETWEEN XF AND THE OLD X1 IS NOT WITHIN A SET TOLERANCE, AND THE
C     PROCESS STARTS AGAIN.
C
C     THESE ITERATIONS ARE WRITTEN INTO OUTPUT FILE HSPRINGCNUM.
C
C*********************************************************************
  250 CONTINUE
      C1=XO-XSTAR
      C2=(N*(XO-XSTAR)+XDOTO)/PSTAR
      P=PSTAR
      XF=XSTAR+DEXP(-N*DELTAT)*(C1*DCOS(P*DELTAT)+C2*DSIN(P*DELTAT))
      WRITE(10,260) XO,X1,XF
  260 FORMAT(1X,3(G20.10,2X))
      DIFF=X1-XF
      IF(DABS(DIFF).LT.1.0D-08) GO TO 270
      X1=XF
      DIFF=XO-XF
      IF(DABS(DIFF).LT.1.0D-08) GO TO 270
      CALL XPRIME(XO,X1,XSTAR)
      CALL FREQ(XO,X1,N,PSTAR)
      CALL TIME(XO,XDOTO,X1,DELT,XSTAR,N,PSTAR,DELTAT)
      GO TO 250
  270 CONTINUE
C*********************************************************************
C
C     THE FINAL VALUES ARE WRITTEN INTO OUTPUT FILE HSPRINGCNUM.
C
C*********************************************************************
      TF=T1
      XF=X1
      I=I+1
      NPOINT=I
      AMPLP(I)=XF
      TP(I)=TF
      TAU=2.0*T1
      WRITE(10,280) DELTAX,TAU
  280 FORMAT('-','DELTAX=',G20.10,2X,'PERIOD=',G20.10)
      WRITE(10,290) XF
  290 FORMAT('0','FIRST TROUGH OCCURS AT X=',G20.10)
      GO TO 320
  300 CONTINUE
C*********************************************************************
C
C     THE FINAL VALUES ARE WRITTEN INTO OUTPUT FILE HSPRINGCNUM (UNDAMPED
C     CASE).
C
C*********************************************************************
      NPOINT=1
      TAU=2.0*T1
      WRITE(10,310) DELTAX,TAU
  310 FORMAT('-','DELTAX=',G20.10,2X,'PERIOD=',G20.10)
  320 CONTINUE
C*********************************************************************
C
C     THE FOLLOWING SECTION (ENDING AT LINE 330) IS THE RUNGE-KUTTA
C     SOLUTION OF THE SAME ORIGINAL EQUATION.
C
C
C
C     TFINAL=FINAL TIME
C     INTVL=NUMBER OF INCREMENTS
C
C*********************************************************************
      TFINAL=T1+0.1
      INTVL=TFINAL/DTIME+1
C*********************************************************************
C
C     THE FOLLOWING ARE PARAMETERS FOR THE NUMERICAL SUBROUTINE.
C     CONSULT THE IMSL MANUAL ON "DVERK" FOR FURTHER DETAILS.
C
C*********************************************************************
      XRK=TIMEO
      Y(1)=DISPLO
      Y(2)=VO
      NEQN=2
      NW=2
      TOL=1.0D-12
      IND=1
      XEND=DTIME
C*********************************************************************
C
C     ZEIT=CUMULATIVE VALUE OF TIME AT END OF EACH INCREMENT
C
C*********************************************************************
      ZEIT=0.0
C*********************************************************************
C
C     AMPLRK=ARRAY FOR AMPLITUDE VALUES
C     TRK=ARRAY FOR TIME VALUES
C
C*********************************************************************
      DO 330 J=1,INTVL
```

```
      AMPLRK(J)=Y(1)
      TRK(J)=ZEIT
      CALL DVERK(NEQN,FCN,XRK,Y,XEND,TOL,IND,CN,NW,W,IER)
C*********************************************************************
C
C     INCREMENTING THE TIME VALUES.
C
C*********************************************************************
      ZEIT=XEND
      XEND=XEND+DTIME
  330 CONTINUE
C*********************************************************************
C
C     THE CALCULATED VALUES ARE WRITTEN INTO OUTPUT FILE NSPRINGCRK.
C
C*********************************************************************
      WRITE(11,340)
  340 FORMAT('1','RUNGE-KUTTA RESULTS FOR THE EQUATION:')
      WRITE(11,350)
  350 FORMAT('0','X(DOUBLE-DOT)+C*X(DOT)+A*X+B*(X**3)=0')
      WRITE(11,360) A,B,C
  360 FORMAT('0','A=',G20.10,2X,'B=',G20.10,2X,'C=',G20.10)
      WRITE(11,370) DISPLO,VO
  370 FORMAT('0','X(0)=',G20.10,2X,'X(DOT)(0)=',G20.10)
      WRITE(11,380)
  380 FORMAT('-',8X,'TIME',18X,'AMPLITUDE')
      DO 400 NO=1,INTVL
      WRITE(11,390) TRK(NO),AMPLRK(NO)
  390 FORMAT(1X,2(G20.10,2X))
  400 CONTINUE
C*********************************************************************
C
C     THE NEXT FEW LINES ARE USED IN FINALIZING ALL THE PLOT PARAMETERS
C     WHICH INCLUDES AXIS SCALING.
C
C*********************************************************************
      M=1
      IF(DISPLO.LE.1.0) GO TO 420
  410 CONTINUE
      DECD=DISPLO/(10.0**M)
      IF(DECD.LE.1.0) GO TO 440
      M=M+1
      GO TO 410
  420 CONTINUE
      DIGD=DISPLO*(10.0**M)
      IF(DIGD.GE.1.0) GO TO 430
      M=M+1
      GO TO 420
  430 CONTINUE
      DECD=DIGD/10.0
      M=-(M-1)
  440 CONTINUE
      IF(DECD.LE.0.1) VFACT=0.1
      IF((DECD.GT.0.1).AND.(DECD.LE.0.25)) VFACT=0.25
      IF((DECD.GT.0.25).AND.(DECD.LE.0.5)) VFACT=0.5
      IF((DECD.GT.0.5).AND.(DECD.LE.1.0)) VFACT=1.0
      SFACT=VC/10.0
      VA=-VFACT*SFACT*10.0**M
      DEN=VC/(2.0*SFACT)
      VB=(VFACT/DEN)*10.0**M
C
      M=1
      IF(TFINAL.LE.1.0) GO TO 460
  450 CONTINUE
      DECT=TFINAL/(10.0**M)
      IF(DECT.LE.1.0) GO TO 480
      M=M+1
      GO TO 450
  460 CONTINUE
      DIGT=TFINAL*(10.0**M)
      IF(DIGT.GE.1.0) GO TO 470
      M=M+1
      GO TO 460
  470 CONTINUE
      DECT=DIGT/10.0
      M=-(M-1)
  480 CONTINUE
      IF(DECT.LE.0.07) HFACT=0.1
      IF((DECT.GT.0.07).AND.(DECT.LE.0.14)) HFACT=0.2
      IF((DECT.GT.0.14).AND.(DECT.LE.0.35)) HFACT=0.5
      IF((DECT.GT.0.35).AND.(DECT.LE.0.7)) HFACT=1.0
      IF((DECT.GT.0.7).AND.(DECT.LE.1.0)) HFACT=2.0
      HB=HFACT*10.0**(M-1)
      LONG=HC
      XA(2)=DFLOAT(LONG)*HB-0.005
C*********************************************************************
C
C     THE FOLLOWING SECTION SETS UP THE PIECEWISE LINEAR RESULTS SO THAT
C     NO MORE THAN 13 POINTS ARE PLOTTED FOR A CLEARER PRESENTATION.
C
C     YP=ARRAY FOR PLOTTED PIECEWISE LINEAR DISPLACEMENTS
C     TPP=ARRAY FOR PLOTTED PIECEWISE LINEAR TIME VALUES
C
C*********************************************************************
      COUNT=DFLOAT(NPOINT)
      INC=COUNT/10.0
      DELINC=COUNT/10.0-DFLOAT(INC)
      IF(DELINC.GE.0.5) INC=INC+1
      NP=1
      ND=1
  490 CONTINUE
      YP(NP)=AMPLP(ND)
      TPP(NP)=TP(ND)
      NP=NP+1
      ND=ND+INC
      IF(ND.GE.NPOINT) GO TO 500
      GO TO 490
  500 CONTINUE
      YP(NP)=AMPLP(NPOINT)
      TPP(NP)=TP(NPOINT)
```

```
C****************************************************************
C
C      PLOTTING THE SOLUTIONS.  CONSULT THE WRITEUP ON CGPL/CGPL2 AND
C      THE MANUAL ON DIGITAL PLOTTING FOR DETAILS.
C
C****************************************************************
      ND=INTVL
      NF=1
      CALL CGPL2(TRK,AMPLRK,ND,NF,5,HA,HB,HC,VA,VB,VC,ALPH)
      ND=NP
      CALL CGPL2(TPP,YP,ND,2,1,HA,HB,HC,VA,VB,VC,ALPH)
      NF=4
      CALL CGPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
      STARTX=(HC-5.3)/2.0
      STARTY=VC+0.5
      HORIZ=HA
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=HC
      CALL PLOT(HORIZ,VERT,2)
      VERT=VERT+4.5
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=HA
      CALL PLOT(HORIZ,VERT,2)
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+0.2
      VERT=STARTY+4.2
      CALL SYMBOL(HORIZ,VERT,0.2,'A =',0.0,3)
      HORIZ=STARTX+1.0
      CALL NUMBER(HORIZ,VERT,0.2,A,0.0,4)
      HORIZ=STARTX+3.0
      CALL SYMBOL(HORIZ,VERT,0.2,'B =',0.0,3)
      HORIZ=STARTX+3.8
      CALL NUMBER(HORIZ,VERT,0.2,B,0.0,4)
      HORIZ=STARTX+2.0
      VERT=STARTY+3.6
      CALL SYMBOL(HORIZ,VERT,0.2,'C =',0.0,3)
      HORIZ=STARTX+2.8
      CALL NUMBER(HORIZ,VERT,0.2,C,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+3.1
      CALL SYMBOL(HORIZ,VERT,0.2,52,0.0,-1)
      HORIZ=STARTX+0.4
      CALL SYMBOL(HORIZ,VERT,0.2,'X (PIECEWISE',0.0,11)
      HORIZ=STARTX+1.0
      VERT=STARTY+2.8
      CALL SYMBOL(HORIZ,VERT,0.2,'LINEARIZATION) =',0.0,16)
      HORIZ=STARTX+4.4
      CALL NUMBER(HORIZ,VERT,0.2,DELX,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+2.4
      CALL SYMBOL(HORIZ,VERT,0.2,52,0.0,-1)
      HORIZ=STARTX+0.4
      CALL SYMBOL(HORIZ,VERT,0.2,'T (RUNGE-KUTTA) =',0.0,17)
      HORIZ=STARTX+4.0
      CALL NUMBER(HORIZ,VERT,0.2,DTIME,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+1.8
      CALL SYMBOL(HORIZ,VERT,0.2,'X(0) =',0.0,6)
      HORIZ=STARTX+1.5
      CALL NUMBER(HORIZ,VERT,0.2,DISPLO,0.0,4)
      HORIZ=STARTX+3.5
      CALL SYMBOL(HORIZ,VERT,0.2,'X(0) =',0.0,6)
      HORIZ=STARTX+3.525
      VERT=STARTY+1.9
      CALL SYMBOL(HORIZ,VERT,0.2,75,0.0,-1)
      HORIZ=STARTX+4.8
      VERT=STARTY+1.6
      CALL NUMBER(HORIZ,VERT,0.2,V0,0.0,4)
      HORIZ=STARTX+0.3
      VERT=STARTY+0.7
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=STARTX+0.8
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.5
      CALL SYMBOL(HORIZ,VERT,0.2,'RUNGE-KUTTA',0.0,11)
      HORIZ=STARTX+0.6
      VERT=STARTY+0.3
      CALL SYMBOL(HORIZ,VERT,0.2,1,0.0,-1)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.2
      CALL SYMBOL(HORIZ,VERT,0.2,'PIECEWISE LINEARIZATION',0.0,23)
      NF=0
      CALL CGPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
      STOP
      END
C
C
C
C
C
C
C
C
C
C
C****************************************************************
C
C      THE FOLLOWING SUBROUTINE CALCULATES THE POINT WHERE THE INTERVAL
C      CHORD LINE WOULD CROSS THE X-AXIS.
C
C****************************************************************
      SUBROUTINE XPRIME(X0,X1,XSTAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/COEFF/A,B
      IF(DABS(X0).LT.1.0D-08) GO TO 500
      F0=A*X0+B*(X0**3)
      F1=A*X1+B*(X1**3)
```

```
      XSTAR=(X1-(F1/F0)*X0)/(1-(F1/F0))
      GO TO 610
  600 CONTINUE
      XSTAR=0.D0
  610 CONTINUE
      RETURN
      END
C
C
C
C#################################################################
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE ANGULAR FREQUENCY FOR
C     AN INTERVAL.
C
C#################################################################
      SUBROUTINE FREQ(X0,X1,N,PSTAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      COMMON/COEFF/A,B
      F0=A*X0+B*(X0**3)
      F1=A*X1+B*(X1**3)
      SLOPE=(F0-F1)/(X0-X1)
      PSTAR=DSQRT(SLOPE-N**2)
      RETURN
      END
C
C
C
C#################################################################
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE TIME INTERVAL FOR A
C     PARTICULAR SEGMENT BY USING A NEWTON-RAPHSON METHOD.  IT ALSO HAS
C     A PROVISION FOR FINDING THE LAST TIME INTERVAL PRIOR TO THE FIRST
C     TROUGH ON THE DISPLACEMENT-TIME CURVE.
C
C#################################################################
      SUBROUTINE TIME(X0,XDOT0,X1,DELT,XSTAR,N,P,DELTAT)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      DELTA=DELT+0.05
      C1=X0-XSTAR
      C2=(N*(X0-XSTAR)+XDOT0)/P
      C3=-N*C1+P*C2
      C4=-P*C1-N*C2
      C5=-(P**2-N**2)*C1-2.0*P*N*C2
      C6=2.0*P*N*C1-(P**2-N**2)*C2
C#################################################################
C
C     F=MAIN FUNCTION (DISPLACEMENT)
C     FP=F-PRIME (VELOCITY)
C
C#################################################################
  700 CONTINUE
      F=X1-XSTAR-DEXP(-N*DELTA)*(C1*DCOS(P*DELTA)+C2*DSIN(P*DELTA))
      FP=-DEXP(-N*DELTA)*(C3*DCOS(P*DELTA)+C4*DSIN(P*DELTA))
      IF(FP.LT.-1.0D-15) GO TO 710
      DELTA1=DELTA-F/FP
      DIFF=DELTA1-DELTA
      IF(DABS(DIFF).LT.1.0D-08) GO TO 730
      DELTA=DELTA1
      GO TO 700
C#################################################################
C
C     AS THE DISPLACEMENT APPROACHES A PEAK OR TROUGH, THE VELOCITY
C     APPROACHES ZERO.  THE RESULT IS THAT INSTEAD OF F AND FP, THE
C     FOLLOWING ARE USED IN THE RESPECTIVE POSITIONS:
C
C     FP=MAIN FUNCTION (VELOCITY)
C     FPP=FP-PRIME (ACCELERATION)
C
C     AND THE CALCULATION CONTINUES.
C
C#################################################################
  710 CONTINUE
      DELTA=DELT+0.05
  720 CONTINUE
      FP=-DEXP(-N*DELTA)*(C3*DCOS(P*DELTA)+C4*DSIN(P*DELTA))
      FPP=-DEXP(-N*DELTA)*(C5*DCOS(P*DELTA)+C6*DSIN(P*DELTA))
      DELTA1=DELTA-FP/FPP
      DIFF=DELTA1-DELTA
      IF(DABS(DIFF).LT.1.0D-08) GO TO 730
      DELTA=DELTA1
      GO TO 720
  730 CONTINUE
      DELTAT=DELTA
      RETURN
      END
C
C
C
C#################################################################
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE VELOCITY AT THE END OF THE
C     SEGMENT BEING EXAMINED.
C
C#################################################################
      SUBROUTINE SPEED(X0,XDOT0,XSTAR,N,DELTAT,P,XDOT1)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      C1=XDOT0
      C2=-N*(N*(X0-XSTAR)+XDOT0)/P-P*(X0-XSTAR)
      XDOT1=DEXP(-N*DELTAT)*(C1*DCOS(P*DELTAT)+C2*DSIN(P*DELTAT))
      RETURN
      END
C
C
C
C#################################################################
C
```

```
C     THE FOLLOWING SUBROUTINE CONTAINS THE DIFFERENTIAL EQUATIONS TO BE
C     SOLVED BY DVERK:  THE INSTANTANEOUS VELOCITY AND THE ORIGINAL
C     EQUATION.  "DFORCE" IS THE DAMPING FORCE.
C
C**************************************************************************
      SUBROUTINE FCN(NEON,X,Y,YPRIME)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 Y(NEON),YPRIME(NEON),X
      COMMON/COEFF/A,B
      COMMON/DAMP/C
      YPRIME(1)=Y(2)
      DFORCE=C*Y(2)
      YPRIME(2)=-(DFORCE+A*Y(1)+B*(Y(1)**3))
      RETURN
      END
END OF FILE
```

```fortran
C*************************************************************************
C*
C*    PROGRAM HSPRINGQ
C*
C*
C*
C*    THE FOLLOWING PROGRAM CALCULATES THE SOLUTION FOR THE
C*    EQUATION:
C*
C*    X(DOUBLE-DOT)+Q*(ABS(X(DOT))**2)*SGN(X(DOT))+A*X+B*(X**3)=0
C*
C*    OBTAINED BY PIECEWISE LINEARIZATION USING CHORDS, AS WELL AS
C*    FINDING THE FIRST TROUGH ON THE DISPLACEMENT-TIME CURVE.
C*
C*
C*    THE RESULTS ARE FOUND FROM THE FOLLOWING:
C*
C*    X=XSTAR+EXP(-N*(T-TO))*(C1*COS(PSTAR*(T-TO))
C*                          +C2*SIN(PSTAR*(T-TO)))
C*
C*    X(DOT)=EXP(-N*(T-TO))*(C3*COS(PSTAR*(T-TO)
C*                          +C4*SIN(PSTAR*(T-TO)))
C*
C*    C1=XO-XSTAR
C*    C2=(N*(XO-XSTAR)+XDOTO)/P
C*    C3=-N*C1+P*C2
C*    C4=-P*C1-[illegible]C2
C*
C*    THE VARIOUS PARAMETERS ARE EXPLAINED AND CALCULATED IN THE
C*    PROGRAM.
C*
C*
C*    THE PROGRAM ALSO SOLVES THE EQUATION USING FIFTH- AND SIXTH-
C*    ORDER RUNGE-KUTTA METHODS.  BOTH SOLUTIONS ARE PLOTTED.
C*
C*************************************************************************
C
C
C
C
C
C
C
C
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL FCN
      REAL*8 Y(2),CN(24),W(2,20)
      REAL*8 N
      REAL*4 AMPLP(200),AMPLRK(200),TP(200),TRK(200)
      REAL*4 YP(2000),TPP(200)
      REAL*4 XA(2),YA(2)
      REAL*4 HA,HB,HC,VA,VB,VC
      INTEGER*4 ALPH(20)
      COMMON/COEFF/A,B
      COMMON/DAMP/Q
C#########################################################################
C
C     THE FOLLOWING VALUES ARE READ FROM DATA FILE PFITODATA:
C
C
C     A,B,Q=AS IN THE EQUATION
C
C     DISPLO=INITIAL AMPLITUDE
C     VO=INITIAL VELOCITY
C     TIMEO=INITIAL TIME
C
C     H=NUMBER OF DIVISIONS OF INITIAL DISPLACEMENT
C     DTIME=TIME INCREMENT (RUNGE-KUTTA SOLUTION)
C
C     HA,HC=PLOT PARAMETERS FOR HORIZONTAL AXIS (HC AN INTEGRAL MULTIPLE
C           OF 7.0)
C
C     VC=PLOT PARAMETER FOR VERTICAL AXIS (AN INTEGRAL MULTIPLE OF 12.0,
C         12.0 GIVING BEST RESULTS)
C
C     XA,YA=PARAMETERS FOR PLOTTING ZERO LINE
C
C     ALPH=ARRAY FOR PLOT AXIS LABELS
C
C#########################################################################
      READ(5,10) A,B,Q
   10 FORMAT(3D10.5)
      READ(5,20) DISPLO,VO,TIMEO
   20 FORMAT(3D10.5)
      READ(5,30) H,DTIME
   30 FORMAT(2D10.5)
      READ(5,40) HA,HC
   40 FORMAT(2F10.4)
      READ(5,50) VC
   50 FORMAT(F10.4)
      READ(5,60) XA(1)
   60 FORMAT(F10.4)
      READ(5,70) YA(1),YA(2)
   70 FORMAT(2F10.4)
      READ(5,80) (ALPH(I),I=1,12)
   80 FORMAT(12A4)
      READ(5,90) (ALPH(I),I=13,16)
   90 FORMAT(4A4)
      READ(5,100) (ALPH(I),I=17,20)
  100 FORMAT(4A4)
C#########################################################################
C
C     C=DAMPING COEFFICIENT (INITIALLY SET AT ZERO)
C     N=DAMPING FACTOR
C
C#########################################################################
      C=0.0
      N=C/2.0
```

```
C######################################################################
C
C     SETTING THE INITIAL CONDITIONS.
C
C     XO=INITIAL DISPLACEMENT FOR EACH SUCCESSIVE SEGMENT
C     XDOTO=INITIAL VELOCITY FOR EACH SEGMENT
C     XDOTP=PREVIOUS FINAL SEGMENT VELOCITY
C     TO=CUMULATIVE TIME TAKEN TO REACH BEGINNING OF SEGMENT
C     DELT=INITIAL GUESS AT TIME INTERVAL FOR THE SEGMENT BEING EXAMINED
C          (SEE SUBROUTINE TIME BELOW)
C
C     DELTAX=LENGTH OF SEGMENT
C
C     I=COUNTER FOR DATA POINTS TO BE PLOTTED
C
C######################################################################
      XO=DISPLO
      XDOTO=VO
      XDOTP=VO
      TO=TIMEO
      DELT=TIMEO
      DELTAX=DISPLO/N
      DELX=DELTAX
      I=0
C######################################################################
C
C     THE EQUATION PARAMETERS ARE WRITTEN INTO OUTPUT FILE PFITONUM.
C
C######################################################################
      WRITE(10,110)
  110 FORMAT('1','PIECEWISE LINEAR PERIOD FOR THE EQUATION:')
      WRITE(10,120)
  120 FORMAT('0','X(DOUBLE-DOT)+Q*(ABS(X(DOT))**2)*SGN(X(DOT))+A*X+B(X**
     13)=0')
      WRITE(10,130) A,B,Q
  130 FORMAT('0','A=',G20.10,2X,'B=',G20.10,2X,'Q=',G20.10)
      WRITE(10,140) DISPLO,VO
  140 FORMAT('0','X(0)=',G20.10,2X,'X(DOT)(0)=',G20.10)
      WRITE(10,150)
  150 FORMAT('-',5X,'AMPLITUDE',10X,'CUMULATIVE TIME')
C######################################################################
C
C     X1=DISPLACEMENT AT END OF SEGMENT
C
C######################################################################
      X1=XO-DELTAX
  160 CONTINUE
      I=I+1
  170 CONTINUE
C######################################################################
C
C     XPRIME=SUBROUTINE FOR CALCULATING POINT WHERE SEGMENT CHORD LINE
C          CROSSES X-AXIS
C     XSTAR=CALCULATED VALUE
C
C     FREQ=SUBROUTINE FOR CALCULATING ANGULAR FREQUENCY
C     PSTAR=CALCULATED VALUE
C
C     TIME=SUBROUTINE FOR CALCULATING TIME INTERVAL FOR A SEGMENT
C     DELTAT=CALCULATED VALUE
C
C     SPEED=SUBROUTINE FOR CALCULATING SPEED AT END OF SEGMENT
C     XDOT1=CALCULATED VALUE
C
C**NOTE**THE PROGRAM STOPS IF THE DISPLACEMENT AT THE END OF A SEGMENT
C         IS EQUAL TO -DISPLO (I. E., UNDAMPED CASE).
C
C######################################################################
      CALL XPRIME(XO,X1,XSTAR)
      CALL FREQ(XO,X1,N,PSTAR)
      CALL TIME(XO,XDOTO,X1,DELT,XSTAR,N,PSTAR,DELTAT)
      CALL SPEED(XO,XDOTO,XSTAR,N,DELTAT,PSTAR,XDOT1)
C######################################################################
C
C     THE FOLLOWING IS THE ITERATION TO FIND THE END VELOCITY FOR THE
C     PRESENT SEGMENT.
C
C     DELTAV=DIFFERENCE BETWEEN XDOT1 AND XDOTP
C     XDOTAV=AVERAGE VELOCITY FOR SEGMENT
C     SGNVAV=SGN(XDOTAV)
C
C     C=NEW DAMPING COEFFICIENT FOR NEXT ITERATION
C
C######################################################################
      DELTAV=XDOT1-XDOTP
      IF(DABS(DELTAV).LT.1.0D-05) GO TO 180
      XDOTP=XDOT1
      XDOTAV=(XDOT1+XDOTO)/2.0
      SGNVAV=XDOTAV/DABS(XDOTAV)
      C=Q*((DABS(XDOTAV))**2)*SGNVAV/XDOTAV
      N=C/2.0
      GO TO 170
  180 CONTINUE
C######################################################################
C
C     STARTING AT STEP 210 AND CONTINUING AT 220, THE PROGRAM TESTS FOR
C     ZERO VELOCITY.  IF IT OCCURS AT THE END OF THE SEGMENT, THE
C     PROGRAM STOPS (SEE STEP 320).  IF NOT, THE LOCATION OF THE FIRST
C     TROUGH IS SOUGHT (STEPS 270 TO 290).
C
C######################################################################
      DELT=DELTAT
      T1=TO+DELTAT
      IF(XO.GE.0.0) GO TO 190
      GO TO 210
  190 CONTINUE
C######################################################################
C
C     AMPLP=ARRAY FOR XO
C     TP=ARRAY FOR TO
```

```
      C
      C*********************************************************************
            AMPLP(I)=XO
            TP(I)=TO
      C*********************************************************************
      C
      C     THE INTERIM VALUES ARE WRITTEN INTO OUTPUT FILE PFITONUM.
      C
      C*********************************************************************
            WRITE(10,200) XO,TO
        200 FORMAT(1X,2(G20.10))
      C*********************************************************************
      C
      C     RESETTING THE INITIAL CONDITIONS.
      C
      C*********************************************************************
            XO=X1
            X1=XO-DELTAX
            XDOTO=XDOT1
            TO=T1
            GO TO 160
        210 CONTINUE
            IF(DABS(XDOT1).LT.1.0D-08) GO TO 220
            GO TO 180
        220 CONTINUE
      C*********************************************************************
      C
      C     THE SEGMENT IN WHICH THE FIRST TROUGH OCCURS IS WRITTEN INTO
      C     OUTPUT FILE PFITONUM.
      C
      C*********************************************************************
            AMPLP(I)=XO
            TP(I)=TO
            WRITE(10,230) XO,TO
        230 FORMAT(1X,2(G20.10))
            DIFF=-DISPLO-XO
            IF(DABS(DIFF).LT.1.0D-08) GO TO 320
            WRITE(10,240) X1,T1
        240 FORMAT(1X,2(G20.10))
            WRITE(10,250)
        250 FORMAT('-','TROUGH ITERATIONS')
            WRITE(10,260)
        260 FORMAT('0',10X,'XO',20X,'X1',14X,'X1(CALCULATED)')
      C*********************************************************************
      C
      C     THE FOLLOWING (ENDING AT STEP 290) IS THE CALCULATION OF THE FIRST
      C     TROUGH.  USING THE TIME INTERVAL REQUIRED TO REACH ZERO VELOCITY,
      C     A NEW VALUE OF X1 (XF) IS CALCULATED AND COMPARED TO THE PREVIOUS
      C     VALUE.  THIS NEW VALUE BECOMES THE NEW X1 IF THE DIFFERENCE
      C     BETWEEN XF AND THE OLD X1 IS NOT WITHIN A SET TOLERANCE, AND THE
      C     PROCESS STARTS AGAIN.
      C
      C     THESE ITERATIONS ARE WRITTEN INTO OUTPUT FILE PFITONUM.
      C
      C*********************************************************************
        270 CONTINUE
            C1=XO-XSTAR
            C2=(N*(XO-XSTAR)+XDOTO)/PSTAR
            P=PSTAR
            XF=XSTAR+DEXP(-N*DELTAT)*(C1*DCOS(P*DELTAT)+C2*DSIN(P*DELTAT))
            WRITE(10,280) XO,X1,XF
        280 FORMAT(2(F12.8,2X),2X,F12.8)
            DIFF=X1-XF
            IF(DABS(DIFF).LT.1.0D-08) GO TO 290
            X1=XF
            DIFF=XO-XF
            IF(DABS(DIFF).LT.1.0D-08) GO TO 290
            CALL XPRIME(XO,X1,XSTAR)
            CALL FREQ(XO,X1,N,PSTAR)
            CALL TIME(XO,XDOTO,X1,DELT,XSTAR,N,PSTAR,DELTAT)
            GO TO 270
        290 CONTINUE
      C*********************************************************************
      C
      C     THE FINAL VALUES ARE WRITTEN INTO OUTPUT FILE HSPRINGONUM.
      C
      C*********************************************************************
            TF=T1
            XF=X1
            I=I+1
            NPOINT=I
            AMPLP(I)=XF
            TP(I)=TF
            TAU=2.0*T1
            WRITE(10,300) DELTAX,TAU
        300 FORMAT('-','DELTAX=',G20.10,2X,'PERIOD=',G20.10)
            WRITE(10,310) XF
        310 FORMAT('0','FIRST TROUGH OCCURS AT X=',G20.10)
            GO TO 340
        320 CONTINUE
      C*********************************************************************
      C
      C     THE FINAL VALUES ARE WRITTEN INTO OUTPUT FILE HSPRINGONUM (UNDAMPED
      C     CASE).
      C
      C*********************************************************************
            NPOINT=I
            TAU=2.0*T1
            WRITE(10,330) DELTAX,TAU
        330 FORMAT('-','DELTAX=',G20.10,2X,'PERIOD=',G20.10)
        340 CONTINUE
      C*********************************************************************
      C
      C     THE FOLLOWING SECTION (ENDING AT LINE 350) IS THE RUNGE-KUTTA
      C     SOLUTION OF THE SAME ORIGINAL EQUATION.
      C
      C
      C
      C     TFINAL=FINAL TIME
      C     INTVL=NUMBER OF INCREMENTS
```

```
C
C**********************************************************************
      TFINAL=TI+0.1
      INTVL=TFINAL/DTIME+1
C**********************************************************************
C
C     THE FOLLOWING ARE PARAMETERS FOR THE NUMERICAL SUBROUTINE.
C     CONSULT THE IMSL MANUAL ON "DVERK" FOR FURTHER DETAILS.
C
C**********************************************************************
      XRK=TIME0
      Y(1)=DISPL0
      Y(2)=V0
      NEQN=2
      NW=2
      TOL=1.0D-12
      IND=1
      XEND=DTIME
C**********************************************************************
C
C     ZEIT=CUMULATIVE VALUE OF TIME AT END OF EACH INCREMENT
C
C**********************************************************************
      ZEIT=0.0
C**********************************************************************
C
C     AMPLRK=ARRAY FOR AMPLITUDE VALUES
C     TRK=ARRAY FOR TIME VALUES
C
C**********************************************************************
      DO 350 J=1,INTVL
      AMPLRK(J)=Y(1)
      TRK(J)=ZEIT
      CALL DVERK(NEQN,FCN,XRK,Y,XEND,TOL,IND,CN,NW,W,IER)
C**********************************************************************
C
C     INCREMENTING THE TIME VALUES.
C
C**********************************************************************
      ZEIT=XEND
      XEND=XEND+DTIME
  350 CONTINUE
C**********************************************************************
C
C     THE CALCULATED VALUES ARE WRITTEN INTO OUTPUT FILE HSPRINGORK.
C
C**********************************************************************
      WRITE(11,360)
  360 FORMAT('1','RUNGE-KUTTA RESULTS FOR THE EQUATION:')
      WRITE(11,370)
  370 FORMAT('0','X(DOUBLE-DOT)+C*X(DOT)+A*X+B*(X**3)=0')
      WRITE(11,380) A,B,C
  380 FORMAT('0','A=',G20.10,2X,'B=',G20.10,2X,'C=',G20.10)
      WRITE(11,390) DISPL0,V0
  390 FORMAT('0','X(0)=',G20.10,2X,'X(DOT)(0)=',G20.10)
      WRITE(11,400)
  400 FORMAT('-',9X,'TIME',15X,'AMPLITUDE')
      DO 420 NO=1,INTVL
      WRITE(11,410) TRK(NO),AMPLRK(NO)
  410 FORMAT(1X,2(G20.10,2X))
  420 CONTINUE
C**********************************************************************
C
C     THE NEXT FEW LINES ARE USED IN FINALIZING ALL THE PLOT PARAMETERS
C     WHICH INCLUDES AXIS SCALING.
C
C**********************************************************************
      M=1
      IF(DISPL0.LE.1.0) GO TO 440
  430 CONTINUE
      DECD=DISPL0/(10.0**M)
      IF(DECD.LE.1.0) GO TO 460
      M=M+1
      GO TO 430
  440 CONTINUE
      DIGD=DISPL0*(10.0**M)
      IF(DIGD.GE.1.0) GO TO 450
      M=M+1
      GO TO 440
  450 CONTINUE
      DECD=DIGD/10.0
      M=-(M-1)
  460 CONTINUE
      IF(DECD.LE.0.1) VFACT=0.1
      IF((DECD.GT.0.1).AND.(DECD.LE.0.25)) VFACT=0.25
      IF((DECD.GT.0.25).AND.(DECD.LE.0.5)) VFACT=0.5
      IF((DECD.GT.0.5).AND.(DECD.LE.1.0)) VFACT=1.0
      SFACT=VC/10.0
      VA=-VFACT*SFACT*10.0**M
      DEN=VC/(2.0*SFACT)
      VB=(VFACT/DEN)*10.0**M
C
      M=1
      IF(TFINAL.LE.1.0) GO TO 480
  470 CONTINUE
      DECT=TFINAL/(10.0**M)
      IF(DECT.LE.1.0) GO TO 500
      M=M+1
      GO TO 470
  480 CONTINUE
      DIGT=TFINAL*(10.0**M)
      IF(DIGT.GE.1.0) GO TO 490
      M=M+1
      GO TO 480
  490 CONTINUE
      DECT=DIGT/10.0
      M=-(M-1)
  500 CONTINUE
      IF(DECT.LE.0.07) HFACT=0.1
      IF((DECT.GT.0.07).AND.(DECT.LE.0.14)) HFACT=0.2
```

```
      IF((DECT.GT.0.14).AND.(DECT.LE.0.35)) HFACT=0.5
      IF((DECT.GT.0.35).AND.(DECT.LE.0.7)) HFACT=1.0
      IF((DECT.GT.0.7).AND.(DECT.LE.1.0)) HFACT=2.0
      HB=HFACT*10.0**(M-1)
      LONG=HC
      XA(2)=DFLOAT(LONG)*HB-0.005
C######################################################################
C
C     THE FOLLOWING SECTION SETS UP THE PIECEWISE LINEAR RESULTS SO THAT
C     NO MORE THAN 12 POINTS ARE PLOTTED FOR A CLEARER PRESENTATION.
C
C     YP=ARRAY FOR PLOTTED PIECEWISE LINEAR DISPLACEMENTS
C     TPP=ARRAY FOR PLOTTED PIECEWISE LINEAR TIME VALUES
C
C######################################################################
      COUNT=DFLOAT(NPOINT)
      INC=COUNT/10.0
      DELINC=COUNT/10.0-DFLOAT(INC)
      IF(DELINC.GE.0.5) INC=INC+1
      NP=1
      ND=1
  510 CONTINUE
      YP(NP)=AMPLP(ND)
      TPP(NP)=TP(ND)
      NP=NP+1
      ND=ND+INC
      IF(ND.GE.NPOINT) GO TO 520
      GO TO 510
  520 CONTINUE
      YP(NP)=AMPLP(NPOINT)
      TPP(NP)=TP(NPOINT)
C######################################################################
C
C     PLOTTING THE SOLUTIONS.  CONSULT THE WRITEUP ON CGPL/CGPL2 AND
C     THE MANUAL ON DIGITAL PLOTTING FOR DETAILS.
C
C######################################################################
      ND=INTVL
      NF=1
      CALL CGPL2(TRK,AMPLRK,ND,NF,5,HA,HB,HC,VA,VB,VC,ALPH)
      ND=NP
      CALL CGPL2(TPP,YP,ND,2,1,HA,HB,HC,VA,VB,VC,ALPH)
      NF=4
      CALL CGPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
      STARTX=(HC-6.3)/2.0
      STARTY=VC+0.5
      HORIZ=HA
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=HC
      CALL PLOT(HORIZ,VERT,2)
      VERT=VERT+4.5
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=HA
      CALL PLOT(HORIZ,VERT,2)
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+0.2
      VERT=STARTY+4.2
      CALL SYMBOL(HORIZ,VERT,0.2,'A =',0.0,3)
      HORIZ=STARTX+1.0
      CALL NUMBER(HORIZ,VERT,0.2,A,0.0,4)
      HORIZ=STARTX+3.0
      CALL SYMBOL(HORIZ,VERT,0.2,'B =',0.0,3)
      HORIZ=STARTX+3.8
      CALL NUMBER(HORIZ,VERT,0.2,B,0.0,4)
      HORIZ=STARTX+2.0
      VERT=STARTY+3.8
      CALL SYMBOL(HORIZ,VERT,0.2,'Q =',0.0,3)
      HORIZ=STARTX+2.8
      CALL NUMBER(HORIZ,VERT,0.2,Q,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+3.1
      CALL SYMBOL(HORIZ,VERT,0.2,52,0.0,-1)
      HORIZ=STARTX+0.4
      CALL SYMBOL(HORIZ,VERT,0.2,'X (PIECEWISE',0.0,11)
      HORIZ=STARTX+1.0
      VERT=STARTY+2.8
      CALL SYMBOL(HORIZ,VERT,0.2,'LINEARIZATION) =',0.0,16)
      HORIZ=STARTX+4.4
      CALL NUMBER(HORIZ,VERT,0.2,DELX,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+2.4
      CALL SYMBOL(HORIZ,VERT,0.2,52,0.0,-1)
      HORIZ=STARTX+0.4
      CALL SYMBOL(HORIZ,VERT,0.2,'T (RUNGE-KUTTA) =',0.0,17)
      HORIZ=STARTX+4.0
      CALL NUMBER(HORIZ,VERT,0.2,DTIME,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+1.8
      CALL SYMBOL(HORIZ,VERT,0.2,'X(0) =',0.0,6)
      HORIZ=STARTX+1.5
      CALL NUMBER(HORIZ,VERT,0.2,DISPLO,0.0,4)
      HORIZ=STARTX+3.5
      CALL SYMBOL(HORIZ,VERT,0.2,'X(0) =',0.0,6)
      HORIZ=STARTX+3.525
      VERT=STARTY+1.9
      CALL SYMBOL(HORIZ,VERT,0.2,75,0.0,-1)
      HORIZ=STARTX+4.9
      VERT=STARTY+1.8
      CALL NUMBER(HORIZ,VERT,0.2,VO,0.0,4)
      HORIZ=STARTX+0.3
      VERT=STARTY+0.7
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=STARTX+0.8
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.6
      CALL SYMBOL(HORIZ,VERT,0.2,'RUNGE-KUTTA',0.0,11)
      HORIZ=STARTX+0.6
```

```
      VERT=STARTY+0.3
      CALL SYMBOL(HORIZ,VERT,0.2,1,0.0,-1)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.2
      CALL SYMBOL(HORIZ,VERT,0.2,'PIECEWISE LINEARIZATION',0.0,23)
      NF=0
      CALL COPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
      STOP
      END
C
C
C
C
C
C
C
C
C
C
C##################################################################
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE POINT WHERE THE INTERVAL
C     CHORD LINE WOULD CROSS THE X-AXIS.
C
C##################################################################
      SUBROUTINE XPRIME(XO,X1,XSTAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/COEFF/A,B
      IF(DABS(XO).LT.1.0D-08) GO TO 600
      FO=A*XO+B*(XO**3)
      F1=A*X1+B*(X1**3)
      XSTAR=(X1-(F1/FO)*XO)/(1-(F1/FO))
      GO TO 610
  600 CONTINUE
      XSTAR=0.D0
  610 CONTINUE
      RETURN
      END
C
C
C
C##################################################################
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE ANGULAR FREQUENCY FOR
C     AN INTERVAL.
C
C##################################################################
      SUBROUTINE FREQ(XO,X1,N,PSTAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      COMMON/COEFF/A,B
      FO=A*XO+B*(XO**3)
      F1=A*X1+B*(X1**3)
      SLOPE=(FO-F1)/(XO-X1)
      PSTAR=DSQRT(SLOPE-N**2)
      RETURN
      END
C
C
C
C##################################################################
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE TIME INTERVAL FOR A
C     PARTICULAR SEGMENT BY USING A NEWTON-RAPHSON METHOD.  IT ALSO HAS
C     A PROVISION FOR FINDING THE LAST TIME INTERVAL PRIOR TO THE FIRST
C     TROUGH ON THE DISPLACEMENT-TIME CURVE.
C
C##################################################################
      SUBROUTINE TIME(XO,XDOTO,X1,DELT,XSTAR,N,P,DELTAT)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      DELTA=DELT+0.05
      C1=XO-XSTAR
      C2=(N*(XO-XSTAR)+XDOTO)/P
      C3=-N*C1+P*C2
      C4=-P*C1-N*C2
      C5=-(P**2-N**2)*C1-2.0*P*N*C2
      C6=2.0*P*N*C1-(P**2-N**2)*C2
C##################################################################
C
C     F=MAIN FUNCTION (DISPLACEMENT)
C     FP=F-PRIME (VELOCITY)
C
C##################################################################
  700 CONTINUE
      F=X1-XSTAR-DEXP(-N*DELTA)*(C1*DCOS(P*DELTA)+C2*DSIN(P*DELTA))
      FP=-DEXP(-N*DELTA)*(C3*DCOS(P*DELTA)+C4*DSIN(P*DELTA))
      IF(FP.LT.-1.0D-15) GO TO 710
      DELTA1=DELTA-F/FP
      DIFF=DELTA1-DELTA
      IF(DABS(DIFF).LT.1.0D-08) GO TO 730
      DELTA=DELTA1
      GO TO 700
C##################################################################
C
C     AS THE DISPLACEMENT APPROACHES A PEAK OR TROUGH, THE VELOCITY
C     APPROACHES ZERO.  THE RESULT IS THAT INSTEAD OF F AND FP, THE
C     FOLLOWING ARE USED IN THE RESPECTIVE POSITIONS:
C
C     FP=MAIN FUNCTION (VELOCITY)
C     FPP=FP-PRIME (ACCELERATION)
C
C     AND THE CALCULATION CONTINUES.
C
C##################################################################
  710 CONTINUE
      DELTA=DELT+0.05
  720 CONTINUE
      FP=-DEXP(-N*DELTA)*(C3*DCOS(P*DELTA)+C4*DSIN(P*DELTA))
      FPP=-DEXP(-N*DELTA)*(C5*DCOS(P*DELTA)+C6*DSIN(P*DELTA))
```

```
      DELTA1=DELTA-FP/FPP
      DIFF=DELTA1-DELTA
      IF(DABS(DIFF).LT.1.0D-08) GO TO 730
      DELTA=DELTA1
      GO TO 720
  730 DELTAT=DELTA
      RETURN
      END
C
C
C
C*********************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE VELOCITY AT THE END OF THE
C     SEGMENT BEING EXAMINED.
C
C*********************************************************************
      SUBROUTINE SPEED(XO,XDOTO,XSTAR,N,DELTAT,P,XDOT1)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      C1=XDOTO
      C2=-N*(N*(XO-XSTAR)+XDOTO)/P-P*(XO-XSTAR)
      XDOT1=DEXP(-N*DELTAT)*(C1*DCOS(P*DELTAT)+C2*DSIN(P*DELTAT))
      RETURN
      END
C
C
C
C*********************************************************************
C
C     THE FOLLOWING SUBROUTINE CONTAINS THE DIFFERENTIAL EQUATIONS TO BE
C     SOLVED BY DVERK:  THE INSTANTANEOUS VELOCITY AND THE ORIGINAL
C     EQUATION.  "DFORCE" IS THE DAMPING FORCE.
C
C*********************************************************************
      SUBROUTINE FCN(N,X,Y,YPRIME)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 Y(N),YPRIME(N),X
      COMMON/COEFF/A,B
      COMMON/DAMP/Q
      YPRIME(1)=Y(2)
      IF(DABS(YPRIME(1)).LT.1.0D-08) GO TO 800
      SGNV=YPRIME(1)/DABS(YPRIME(1))
      GO TO 810
  800 CONTINUE
      SGNV=0.D0
  810 CONTINUE
      DFORCE=Q*(DABS(YPRIME(1))**2)*SGNV
      YPRIME(2)=-(DFORCE+A*Y(1)+B*(Y(1)**3))
      RETURN
      END
END OF FILE
```

```
C*********************************************************************
C*                                                                   *
C*    PROGRAM VANDERPOL                                              *
C*                                                                   *
C*                                                                   *
C*                                                                   *
C*    THE FOLLOWING PROGRAM CALCULATES THE SOLUTION FOR THE VAN DER  *
C*    POL EQUATION:                                                  *
C*                                                                   *
C*    X(DOUBLE-DOT)+MU*(X**2-1)*X(DOT)+X=0                           *
C*                                                                   *
C*    OBTAINED BY PIECEWISE LINEARIZATION USING CHORDS OVER SEVERAL  *
C*    CYCLES OF THE DISPLACEMENT-TIME CURVE.                         *
C*                                                                   *
C*                                                                   *
C*    THE RESULTS ARE BASED ON THE STANDARD FORM:                    *
C*                                                                   *
C*    X=C*EXP(R*DELTAT)                                              *
C*                                                                   *
C*    R1=-N+SQRT(N**2-P**2)                                          *
C*    R2=-N-SQRT(N**2-P**2)                                          *
C*                                                                   *
C*    WHERE THE UNDERDAMPED, CRITICALLY DAMPED, AND OVERDAMPED       *
C*    APPROXIMATIONS WOULD DEPEND UPON MU AND AN ESTIMATED           *
C*    DISPLACEMENT.                                                  *
C*                                                                   *
C*    THE VARIOUS PARAMETERS ARE EXPLAINED AND CALCULATED IN THE     *
C*    PROGRAM.                                                       *
C*                                                                   *
C*                                                                   *
C*    THE PROGRAM ALSO SOLVES THE EQUATION USING FIFTH AND SIXTH-    *
C*    ORDER RUNGE-KUTTA METHODS.  BOTH SOLUTIONS ARE PLOTTED, AS     *
C*    WELL AS THE PHASE-PLANE DIAGRAMS.                              *
C*                                                                   *
C*********************************************************************
C
C
C
C
C
C
C
C
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL FCN
      REAL*8 Y(2),CN(24),W(2,20)
      REAL*8 K,MU,N
      REAL*4 AMPLP(10000),VP(10000),TP(10000)
      REAL*4 AMPLRK(10000),VRK(10000),TRK(10000)
      REAL*4 YP(10000),VPP(10000),TPP(10000)
      REAL*4 YRK(10000),VPRK(10000),TPRK(10000)
      REAL*4 HAD,HBD,HCD,VAD,VBD,VCD
      REAL*4 HAP,HBP,HCP,VAP,VBP,VCP
      REAL*4 XAD(2),YAD(2),XAPH(2),YAPH(2),XAPV(2),YAPV(2)
      INTEGER*4 ALPHD(20),ALPHP(20)
      COMMON MU
C#####################################################################
C
C     THE FOLLOWING VALUES ARE READ FROM DATA FILE VDPPDATA:
C
C
C     MU=AS IN THE EQUATION
C
C     DISPLO=INITIAL AMPLITUDE
C     VO=INITIAL VELOCITY
C     TIMEO=INITIAL TIME
C
C     DELTAT=TIME INCREMENT (PIECEWISE SOLUTION)
C     HCYCLE=NUMBER OF HALF-CYCLES (PIECEWISE SOLUTION)
C     DT=TIME INCREMENT (RUNGE-KUTTA SOLUTION)
C
C     HAD,HCD=PLOT PARAMETERS FOR HORIZONTAL AXIS OF DISPLACEMENT-TIME
C             CURVE (HCD AN INTEGRAL MULTIPLE OF 7.0)
C
C     VCD=PLOT PARAMETER FOR VERTICAL AXIS OF DISPLACEMENT-TIME CURVE
C         (AN INTEGRAL MULTIPLE OF 12.0, 12.0 GIVING BEST RESULTS)
C
C     XAD,YAD=PLOT PARAMETERS FOR PLOTTING ZERO LINE OF DISPLACEMENT-
C             TIME CURVE.
C
C     ALPHD=ARRAY FOR PLOT AXIS LABELS OF DISPLACEMENT-TIME CURVE
C
C     VEP=PLOT PARAETER FOR VERTICAL AXIS OF PHASE-PLAE DIAGRAM (AN
C         INTEGRAL MULTIPLE OF 12.0, 12.0 GIVING BEST RESULTS)
C
C     ALPHP=ARRAY FOR PLOT AXIS LABELS OF PHASE-PLANE DIAGRAM
C
C#####################################################################
      READ(5,10) MU
   10 FORMAT(D10.6)
      READ(5,20) DISPLO,VO,TIMEO
   20 FORMAT(3D10.6)
      READ(5,30) DELTAT,HCYCLE,DT
   30 FORMAT(3D10.6)
      READ(5,40) HAD,HCD
   40 FORMAT(3F10.4)
      READ(5,50) VCD
   50 FORMAT(F10.4)
      READ(5,60) XAD(1)
   60 FORMAT(F10.4)
      READ(5,70) YAD(1),YAD(2)
   70 FORMAT(2F10.4)
      READ(5,80) (ALPHD(I),I=1,12)
   80 FORMAT(12A4)
      READ(5,90) (ALPHD(I),I=13,16)
   90 FORMAT(4A4)
      READ(5,100) (ALPHD(I),I=17,20)
  100 FORMAT(4A4)
```

```
      READ(5,110) VCP
  110 FORMAT(F10.4)
      READ(5,120) (ALPHP(I),I=1,12)
  120 FORMAT(12A4)
      READ(5,130) (ALPHP(I),I=13,16)
  130 FORMAT(4A4)
      READ(5,140) (ALPHP(I),I=17,20)
  140 FORMAT(4A4)
C**********************************************************************
C
C     SETTING THE INITIAL CONDITIONS.
C
C     XO=INITIAL DISPLACEMENT FOR EACH SUCCESSIVE SEGMENT
C     XDOTO=INITIAL VELOCITY FOR EACH SEGMENT
C     TO=INITIAL TIME FOR EACH SEGMENT
C     XPREV=PREVIOUS FINAL SEGMENT DISPLACEMENT
C
C**********************************************************************
      XO=DISPLO
      XDOTO=VO
      TO=TIMEO
      XPREV=DISPLO
C**********************************************************************
C
C     SIG=SUBROUTINE FOR CALCULATING SGN(X)
C     SGNXO=CALCULATED VALUE
C
C**********************************************************************
      CALL SIG(XO,SGNXO)
C**********************************************************************
C
C     I=COUNTER USED FOR PLOTTING PURPOSES
C
C     K=COUNTER FOR THE NUMBER OF HALF-CYCLES (AN INTEGRAL MULTIPLE OF
C       10.0, 10.0 GIVING BEST RESULTS)
C
C**********************************************************************
      I=0
      K=1.0
C**********************************************************************
C
C     THE EQUATION PARAMETERS ARE WRITTEN INTO OUTPUT FILE VOPPNUM.
C
C**********************************************************************
      WRITE(10,150)
  150 FORMAT('1','PIECEWISE LINEAR RESULTS FOR THE EQUATION:')
      WRITE(10,160)
  160 FORMAT('0','X(DOUBLE-DOT)+MU*(X**2-1)*X(DOT)+X=0')
      WRITE(10,170) MU
  170 FORMAT('0','MU=',G20.10)
      WRITE(10,180) DISPLO,VO
  180 FORMAT('0','X(0)=',G20.10,2X,'X(DOT)(0)=',G20.10)
      WRITE(10,190)
  190 FORMAT('-',5X,'TIME',18X,'X',18X,'VELOCITY')
C**********************************************************************
C
C     THE FOLLOWING IS THE CALCULATION OF THE SOLUTION FOR A PARTICULAR
C     HALF-CYCLE, STARTING AT STEP 200 AND ENDING AT STEP 250.
C
C     X=DISPLACEMENT USED FOR CALCULATING A DAMPING COEFFICIENT (SEE
C       BELOW)
C     T1=CUMULATIVE TIME
C
C**********************************************************************
  200 CONTINUE
      I=I+1
      X=XO
      T1=TO+DELTAT
C**********************************************************************
C
C     THE FOLLOWING IS THE ITERATION TO FIND THE END DISPLACEMENT FOR
C     THE PRESENT SEGMENT.
C
C     C=DAMPING COEFFICIENT
C     N=DAMPING FACTOR
C
C     DAMP=SUBROUTINE FOR CALCULATING DEGREE OF DAMPING AND
C          CORRESPONDING DAMPED ANGULAR FREQUENCY
C     PSTAR=CALCULATED VALUE.
C
C     DISPL=SUBROUTINE FOR CALCULATING SEGMENT LENGTH
C     DELTAX=CALCULATED VALUE
C
C     SPEED=SUBROUTINE FOR CALCULATING END VELOCITY
C     XDOT1=CALCULATED VALUE
C
C     X1=SEGMENT END
C
C**********************************************************************
  210 CONTINUE
      C=MU*(X**2-1.0)
      N=C/2.0
      CALL DAMP(N,PSTAR)
      CALL DISPL(XO,XDOTO,N,DELTAT,PSTAR,DELTAX)
      CALL SPEED(XO,XDOTO,N,DELTAT,PSTAR,XDOT1)
      X1=XO+DELTAX
      DIFF=X1-XPREV
      IF(DABS(DIFF).LT.1.0D-08) GO TO 220
      XPREV=X1
      X=XO+DELTAX/2.0
      GO TO 210
  220 CONTINUE
C**********************************************************************
C
C     TESTING FOR THE CHANGE IN SIGN OF THE DISPLACEMENT.  IF IT IS
C     DIFFERENT, THE PROGRAM BEGINS TESTING FOR EITHER A PEAK OR TROUGH,
C     STARTING AT STEP 250.  IF NOT, THE ITERATION CONTINUES.
C
C**********************************************************************
      CALL SIG(X1,SGNX1)
```

```
      IF(SGNX1.NE.SGNXO) GO TO 250
  230 CONTINUE
C###################################################################
C
C     TP=ARRAY FOR TO
C     AMPLP=ARRAY FOR XO
C     VP=ARRAY FOR XDOTO
C
C###################################################################
      TP(I)=TO
      AMPLP(I)=XO
      VP(I)=XDOTO
C###################################################################
C
C     THE INTERIM VALUES ARE WRITTEN INTO OUTPUT FILE VDPPNUM.
C
C###################################################################
      WRITE(10,240) TO,XO,XDOTO
  240 FORMAT(1X,3(G20.10,2X))
C###################################################################
C
C     RESETTING THE INITIAL CONDITIONS FOR THE NEXT SEGMENT.
C
C###################################################################
      XO=X1
      XDOTO=XDOT1
      TO=T1
      GO TO 200
C###################################################################
C
C     STARTING AT STEP 250, THE PROGRAM TESTS FOR EITHER A PEAK OR
C     TROUGH (I. E., ZERO VELOCITY).  IF THE VELOCITY IS ZERO AT THE END
C     OF THE CURRENT TIME INTERVAL, THE PROGRAM MOVES TO STEP 370 AND
C     STARTS THE CALCULATIONS FOR THE NEXT HALF-CYCLE.  IF NOT, A
C     POSSIBLE CHANGE OF SIGN FOR THE VELOCITY IS TESTED FOR.  SHOULD IT
C     BE NEGATIVE, THE PROGRAM GOES TO STEP 230 AND CONTINUES.  A
C     POSITIVE RESULT ALLOWS THE PROGRAM TO ESTIMATE THE VALUE OF THE
C     DISPLACEMENT AT THIS PEAK OR TROUGH, XF (SEE STEPS 310 TO 320).
C     THIS VALUE IS COMPARED TO A PREVIOUS VALUE (XPREV), AND IF THE
C     DIFFERENCE IS VERY SMALL, THE PROGRAM GOES TO STEP 320 AND
C     CONTINUES.  IF NOT, XF BECOMES XPREV, THE PARAMETERS ARE RESET,
C     AND THE ITERATION CONTINUES AT STEP 310.
C
C###################################################################
  250 CONTINUE
      IF(DABS(XDOT1).GT.1.0D-08) GO TO 290
      I=I+1
      TP(I)=T1
      AMPLP(I)=X1
      VP(I)=XDOT1
      WRITE(10,260)
  260 FORMAT('0')
      WRITE(10,270) T1,X1,XDOT1
  270 FORMAT(1X,3(G20.10,2X))
      WRITE(10,280)
  280 FORMAT('0')
      SGNXO=-SGNXO
      GO TO 370
  290 CONTINUE
      CALL DISPL(XO,XDOTO,N,DELTAT,PSTAR,DELX)
      CALL SPEED(XO,XDOTO,N,DELTAT,PSTAR,XDOT1)
      CALL SIG(XDOT1,SGNV1)
      IF(SGNV1.EQ.SGNX1) GO TO 230
      TP(I)=TO
      AMPLP(I)=XO
      VP(I)=XDOTO
      WRITE(10,300) TO,XO,XDOTO
  300 FORMAT(1X,3(G20.10,2X))
      XF=XO+DELX
      T1=T1-DELTAT
C###################################################################
C
C     THE ITERATION FOR EITHER THE PEAK OR TROUGH IS DONE FROM STEPS 310
C     TO 320.
C
C     DELT=ESTIMATED VALUE FOR THE TIME INTERVAL TO REACH ZERO VELOCITY
C     TIME=SUBROUTINE FOR CALCULATING THE PRECISE VALUE FOR THIS TIME
C          INTERVAL
C     DTIME=CALCULATED VALUE
C
C###################################################################
  310 CONTINUE
      DELT=DELTAT
      CALL TIME(XO,XDOTO,DELT,N,PSTAR,DTIME)
      CALL DISPL(XO,XDOTO,N,DTIME,PSTAR,DELTAX)
      CALL SPEED(XO,XDOTO,N,DTIME,PSTAR,XDOT1)
      X1=XO+DELTAX
      DELX=X1-XF
      IF(DABS(DELX).LT.1.0D-08) GO TO 320
      XF=X1
      N=MU*(XF**2-1.0)/2.0
      CALL DAMP(N,PSTAR)
      GO TO 310
  320 CONTINUE
C###################################################################
C
C     THE VALUES FOR THE PEAK OR TROUGH ARE WRITTEN INTO OUTPUT FILE
C     VDPPNUM.
C
C###################################################################
      T1=T1+DTIME
      CALL SPEED(XO,XDOTO,N,DTIME,PSTAR,XDOT1)
      I=I+1
      TP(I)=T1
      AMPLP(I)=X1
      VP(I)=XDOT1
      WRITE(10,330)
  330 FORMAT('0')
      WRITE(10,340) T1,X1,XDOT1
  340 FORMAT(1X,3(G20.10,2X))
```

```
      WRITE(10,350)
  350 FORMAT(1X,3(G20.10,2X))
C**********************************************************************
C
C     TESTING FOR THE NUMBER OF HALF-CYCLES.
C
C**********************************************************************
      IF(K.EQ.NCYCLE) GO TO 380
C**********************************************************************
C
C     RESETTING THE INITIAL CONDITIONS FOR THE NEXT HALF-CYCLE.
C
C**********************************************************************
      X0=X1
      XDOT0=XDOT1
      XPREV=X0
      X=X0
      SGNX0=-SGNX0
      K=K+1.0
C**********************************************************************
C
C     ONCE THE PEAK OR TROUGH CO-ORDINATES HAVE BEEN FOUND, THE VALUES
C     AT THE END OF THE NEXT TIME INCREMENT ARE CALCULATED, SINCE THE
C     FORMER DO NOT OCCUR AT THE END OF THE PREVIOUS ONE.  THE
C     DISPLACEMENT IS FOUND EXACTLY AS WAS DONE FROM STEPS 210 TO 220,
C     EXCEPT WITH THE TIME INTERVAL BEING DIFFT.
C
C**********************************************************************
      DIFFT=DELTAT-DTIME
      T0=T1+DIFFT
  360 CONTINUE
      C=MU*(X**2-1.0)
      N=C/2.0
      CALL DAMP(N,PSTAR)
      CALL DISPL(X0,XDOT0,N,DIFFT,PSTAR,DELTAX)
      CALL SPEED(X0,XDOT0,N,DIFFT,PSTAR,XDOT1)
      X1=X0+DELTAX
      DIFFX=X1-XPREV
      IF(DABS(DIFFX).LT.1.0D-08) GO TO 370
      XPREV=X1
      X=X0+DELTAX/2.0
      GO TO 360
  370 CONTINUE
      X0=X1
      XDOT0=XDOT1
      GO TO 200
  380 CONTINUE
      NPOINT=I
C**********************************************************************
C
C     THE FOLLOWING SECTION (ENDING AT LINE 390) IS THE RUNGE-KUTTA
C     SOLUTION OF THE SAME ORIGINAL EQUATION.
C
C
C
C     TFINAL=FINAL TIME
C     INTVL=NUMBER OF INCREMENTS
C
C**********************************************************************
      TFINAL=T1+0.1
      INTVL=TFINAL/DT+1
C**********************************************************************
C
C     THE FOLLOWING ARE PARAMETERS FOR THE RUNGE-KUTTA SUBROUTINE.
C     CONSULT THE IMSL MANUAL FOR "DVERK" FOR FURTHER DETAILS.
C
C**********************************************************************
      XRK=TIME0
      Y(1)=DISPL0
      Y(2)=V0
C**********************************************************************
C
C     BIGRK=LARGEST VALUE OF RUNGE-KUTTA VELOCITY (FOR PLOT SCALING)
C
C**********************************************************************
      NEQN=2
      NW=2
      TOL=1.0D-04
      IND=1
      XEND=DT
C**********************************************************************
C
C     ZEIT=CUMULATIVE VALUE OF TIME AT END OF EACH INCREMENT
C
C**********************************************************************
      ZEIT=0.0
C**********************************************************************
C
C     AMPLRK=ARRAY FOR AMPLITUDE VALUES
C     TRK=ARRAY FOR TIME VALUES
C     VRK=ARRAY FOR VELOCITY VALUES
C
C**********************************************************************
      DO 390 J=1,INTVL
      AMPLRK(J)=Y(1)
      TRK(J)=ZEIT
      VRK(J)=Y(2)
      CALL DVERK(NEQN,FCN,XRK,Y,XEND,TOL,IND,CN,NW,W,IER)
C**********************************************************************
C
C     INCREMENTING THE TIME VALUES.
C
C**********************************************************************
      ZEIT=XEND
      XEND=XEND+DT
  390 CONTINUE
C**********************************************************************
C
C     THE CALCULATED VALUES ARE WRITTEN INTO OUTPUT FILE VDPNNUM.
C
```

```
C***********************************************************************
      WRITE(11,400)
  400 FORMAT('1','RUNGE-KUTTA RESULTS FOR THE EQUATION:')
      WRITE(11,410)
  410 FORMAT('0','X(DOUBLE-DOT)+MU*(X**2-1)*X(DOT)+X=0')
      WRITE(11,420) MU
  420 FORMAT('0','MU=',G20.10)
      WRITE(11,430) DISPLO,VO
  430 FORMAT('0','X(0)=',G20.10,2X,'X(DOT)(0)=',G20.10)
      WRITE(11,440)
  440 FORMAT('-',4X,'TIME',10X,'X(T)',8X,'VELOCITY')
      DO 460 M=1,INTVL
      WRITE(11,450) TRK(M),AMPLRK(M),VRK(M)
  450 FORMAT(1X,3(G20.10,2X))
  460 CONTINUE
C***********************************************************************
C
C     THE FOLLOWING FINDS THE LARGEST MAGNITUDES OF THE DISPLACEMENTS
C     AND VELOCITIES FOR THE SOLUTIONS FOR PLOT SCALING PURPOSES.
C
C     BIGDP=LARGEST VALUE OF PIECEWISE LINEAR DISPLACEMENT
C     BIGVP=LARGEST VALUE OF PIECEWISE LINEAR VELOCITY
C
C     BIGDRK=LARGEST VALUE OF RUNGE-KUTTA DISPLACEMENT
C     BIGVRK=LARGEST VALUE OF RUNGE-KUTTA VELOCITY
C
C***********************************************************************
      BIGDP=DISPLO
      BIGVP=VO
      DO 470 IB=1,NPOINT
      DISP=AMPLP(IB)
      VEL=VP(IB)
      IF(DABS(DISP).GE.BIGDP) BIGDP=DABS(DISP)
      IF(DABS(VEL).GE.BIGVP) BIGVP=DABS(VEL)
  470 CONTINUE
      BIGDRK=DISPLO
      BIGVRK=VO
      DO 480 IB=1,INTVL
      DISP=AMPLRK(IB)
      VEL=VRK(IB)
      IF(DABS(DISP).GE.BIGDRK) BIGDRK=DABS(DISP)
      IF(DABS(VEL).GE.BIGVRK) BIGVRK=DABS(VEL)
  480 CONTINUE
C***********************************************************************
C
C     THE NEXT FEW LINES ARE USED IN FINALIZING ALL THE PLOT PARAMETERS
C     WHICH INCLUDES AXIS SCALING.
C
C***********************************************************************
      BIGD=BIGDRK
      IF(BIGDP.GE.BIGDRK) BIGD=BIGDP
      M=1
      IF(BIGD.LE.1.0) GO TO 500
  490 CONTINUE
      DECD=BIGD/(10.0**M)
      IF(DECD.LE.1.0) GO TO 520
      M=M+1
      GO TO 490
  500 CONTINUE
      DIGD=BIGD*(10.0**M)
      IF(DIGD.GE.1.0) GO TO 510
      M=M+1
      GO TO 500
  510 CONTINUE
      DECD=DIGD/10.0
      M=-(M-1)
  520 CONTINUE
      IF(DECD.LE.0.1) VFACT=0.1
      IF((DECD.GT.0.1).AND.(DECD.LE.0.25)) VFACT=0.25
      IF((DECD.GT.0.25).AND.(DECD.LE.0.5)) VFACT=0.5
      IF((DECD.GT.0.5).AND.(DECD.LE.1.0)) VFACT=1.0
      SFACT=VCD/10.0
      VAD=-VFACT*SFACT*10.0**M
      DEN=VCD/(2.0*SFACT)
      VBD=(VFACT/DEN)*10.0**M
C
      M=1
      IF(TFINAL.LE.1.0) GO TO 540
  530 CONTINUE
      DECT=TFINAL/(10.0**M)
      IF(DECT.LE.1.0) GO TO 560
      M=M+1
      GO TO 530
  540 CONTINUE
      DIGT=TFINAL*(10.0**M)
      IF(DIGT.GE.1.0) GO TO 550
      M=M+1
      GO TO 540
  550 CONTINUE
      DECT=DIGT/10.0
      M=-(M-1)
  560 CONTINUE
      IF(DECT.LE.0.07) HFACT=0.1
      IF((DECT.GT.0.07).AND.(DECT.LE.0.14)) HFACT=0.2
      IF((DECT.GT.0.14).AND.(DECT.LE.0.35)) HFACT=0.5
      IF((DECT.GT.0.35).AND.(DECT.LE.0.7)) HFACT=1.0
      IF((DECT.GT.0.7).AND.(DECT.LE.1.0)) HFACT=2.0
      HBD=HFACT*10.0**(M-1)
      LONG=HCD
      XAD(2)=DFLOAT(LONG)*HBD-0.005
C
      BIGV=BIGVRK
      IF(BIGVP.GE.BIGVRK) BIGV=BIGVP
      M=1
      IF(BIGV.LE.1.0) GO TO 580
  570 CONTINUE
      DECV=BIGV/(10.0**M)
      IF(DECV.LE.1.0) GO TO 600
      M=M+1
      GO TO 570
```

```
  560 CONTINUE
      DIGV=DIGV*(10.0**M)
      IF(DIGV.GE.1.0) GO TO 590
      M=M+1
      GO TO 560
  590 CONTINUE
      DECV=DIGV/10.0
      M=-(M-1)
  600 CONTINUE
      IF(DECV.LE.0.1) VFACT=0.1
      IF((DECV.GT.0.1).AND.(DECV.LE.0.25)) VFACT=0.25
      IF((DECV.GT.0.25).AND.(DECV.LE.0.5)) VFACT=0.5
      IF((DECV.GT.0.5).AND.(DECV.LE.1.0)) VFACT=1.0
      HAP=-VBD*(VCD/2.0)
      HBP=VBD
      HCP=VCD
      SFACT=VCP/10.0
      VAP=-VFACT*SFACT*10.0**M
      DEN=VCP/(2.0*SFACT)
      VBP=(VFACT/DEN)*10.0**M
      XAPH(1)=HAP+0.005
      XAPH(2)=-HAP-0.005
      YAPH(1)=0.0
      YAPH(2)=0.0
      XAPV(1)=0.0
      XAPV(2)=0.0
      YAPV(1)=VAP+0.005
      YAPV(2)=-VAP-0.005
C***********************************************************************
C
C     THE FOLLOWING SECTION SETS UP THE PIECEWISE LINEAR RESULTS SO THAT
C     BETWEEN 50 AND 60 POINTS ARE PLOTTED, DEPENDING ON HOW CLOSE THE
C     TOTAL NUMBER OF CALCULATIONS (NPOINT) IS TO AN INTEGRAL MULTIPLE
C     OF 50.
C
C     YP=ARRAY FOR PLOTTED PIECEWISE LINEAR DISPLACEMENTS
C     TPP=ARRAY FOR PLOTTED PIECEWISE LINEAR TIME VALUES
C     VPP=ARRAY FOR PLOTTED PIECEWISE LINEAR VELOCITIES
C
C***********************************************************************
      INCP=NPOINT/50
      IDEC=DFLOAT(NPOINT/50)-INCP
      IF(IDEC.GE.0.5) INCP=NPOINT/50+1
      NPP=1
      NDP=1
  610 CONTINUE
      YP(NPP)=AMPLP(NDP)
      TPP(NPP)=TP(NDP)
      VPP(NPP)=VP(NDP)
      NPP=NPP+1
      NDP=NDP+INCP
      IF(NDP.GE.NPOINT) GO TO 620
      GO TO 610
  620 CONTINUE
      IPOINT=(NDP-NPOINT)-INCP/2
      IHALF=INCP/2
      IF(IPOINT.LE.IHALF) GO TO 630
      YP(NPP)=AMPLP(NDP)
      TPP(NPP)=TP(NDP)
      VPP(NPP)=VP(NDP)
      GO TO 640
  630 CONTINUE
      NPP=NPP-1
      NDP=NDP-INCP
  640 CONTINUE
C***********************************************************************
C
C     DUE TO LIMITATIONS IN THE PLOT ROUTINES USED FURTHER ON, THE
C     FOLLOWING SETS UP THE RUNGE-KUTTA RESULTS SO THAT EVERY SECOND
C     POINT IS PLOTTED IF MORE THAN 1000 POINTS WERE CALCULATED.
C
C***********************************************************************
      INCRK=2
      IF(INTVL.LE.999) INCRK=1
      NPRK=1
      NDRK=1
  650 CONTINUE
      YRK(NPRK)=AMPLRK(NDRK)
      TPRK(NPRK)=TRK(NDRK)
      VPRK(NPRK)=VRK(NDRK)
      NPRK=NPRK+1
      NDRK=NDRK+INCRK
      IF(NDRK.GE.INTVL) GO TO 660
      GO TO 650
  660 CONTINUE
      IF(INCRK.EQ.1) GO TO 670
      COUNT=DFLOAT(INTVL)
      INC=COUNT/2.0
      DELINC=COUNT/2.0-DFLOAT(INC)
      IF(DELINC.EQ.0) GO TO 680
  670 CONTINUE
      YRK(NPRK)=AMPLRK(INTVL)
      TPRK(NPRK)=TRK(INTVL)
      VPRK(NPRK)=VRK(INTVL)
      GO TO 690
  680 CONTINUE
      NPRK=NPRK-1
      NDRK=NDRK-INCRK
  690 CONTINUE
C***********************************************************************
C
C     PLOTTING THE SOLUTIONS.  CONSULT THE WRITEUP ON CGPL/CGPL2 AND
C     THE MANUAL ON DIGITAL PLOTTING FOR DETAILS.
C
C***********************************************************************
      ND=NPRK
      NF=1
      CALL CGPL2(TPRK,YRK,ND,NF,5,HAD,HBD,HCD,VAD,VBD,VCD,ALPHD)
      ND=NPP
      CALL CGPL2(TPP,YP,ND,2,1,HAD,HBD,HCD,VAD,VBD,VCD,ALPHD)
```

```
      NF=4
      CALL CGPL2(XAD,YAD,2,NF,4,HAD,HBD,HCD,VAD,VBD,VCD,ALPHD)
      HA=HAD
      HC=HCD
      VA=VAD
      VC=VCD
      IPLOT=0
  700 CONTINUE
      STARTX=(HC-8.3)/2.0
      STARTY=VC+0.5
      HORIZ=HA
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=HC
      CALL PLOT(HORIZ,VERT,2)
      VERT=VERT+4.2
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=HA
      CALL PLOT(HORIZ,VERT,2)
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+2.0
      VERT=STARTY+3.8
      CALL SYMBOL(HORIZ,VERT,0.2,34,0.0,-1)
      HORIZ=STARTX+2.4
      CALL SYMBOL(HORIZ,VERT,0.2,'=',0.0,1)
      HORIZ=STARTX+2.8
      CALL NUMBER(HORIZ,VERT,0.2,MU,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+3.1
      CALL SYMBOL(HORIZ,VERT,0.2,52,0.0,-1)
      HORIZ=STARTX+0.4
      CALL SYMBOL(HORIZ,VERT,0.2,'T (PIECEWISE ',0.0,12)
      HORIZ=STARTX+1.0
      VERT=STARTY+2.8
      CALL SYMBOL(HORIZ,VERT,0.2,'LINEARIZATION) =',0.0,16)
      HORIZ=STARTX+4.2
      CALL NUMBER(HORIZ,VERT,0.2,DELTAT,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+2.4
      CALL SYMBOL(HORIZ,VERT,0.2,52,0.0,-1)
      HORIZ=STARTX+0.4
      CALL SYMBOL(HORIZ,VERT,0.2,'T (RUNGE-KUTTA) =',0.0,17)
      HORIZ=STARTX+4.0
      CALL NUMBER(HORIZ,VERT,0.2,DT,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+1.6
      CALL SYMBOL(HORIZ,VERT,0.2,'X(O) =',0.0,6)
      HORIZ=STARTX+1.6
      CALL NUMBER(HORIZ,VERT,0.2,DISPLO,0.0,4)
      HORIZ=STARTX+3.5
      CALL SYMBOL(HORIZ,VERT,0.2,'X(O) =',0.0,6)
      HORIZ=STARTX+3.525
      VERT=STARTY+1.8
      CALL SYMBOL(HORIZ,VERT,0.2,75,0.0,-1)
      HORIZ=STARTX+4.8
      VERT=STARTY+1.6
      CALL NUMBER(HORIZ,VERT,0.2,VO,0.0,4)
      HORIZ=STARTX+0.3
      VERT=STARTY+0.7
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=STARTX+0.8
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.6
      CALL SYMBOL(HORIZ,VERT,0.2,'RUNGE-KUTTA',0.0,11)
      HORIZ=STARTX+0.6
      VERT=STARTY+0.3
      CALL SYMBOL(HORIZ,VERT,0.2,1,0.0,-1)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.2
      CALL SYMBOL(HORIZ,VERT,0.2,'PIECEWISE LINEARIZATION',0.0,23)
      IF(IPLOT.NE.0) GO TO 710
      NF=0
      CALL CGPL2(XAD,YAD,2,NF,4,HAD,HBD,HCD,VAD,VBD,VCD,ALPHD)
      ND=NPRK
      NF=1
      CALL CGPL2(YRK,VPRK,ND,NF,5,HAP,HBP,HCP,VAP,VBP,VCP,ALPHP)
      ND=NPP
      CALL CGPL2(YP,VPP,ND,2,1,HAP,HBP,HCP,VAP,VBP,VCP,ALPHP)
      NF=4
      CALL CGPL2(XAPH,YAPH,2,NF,4,HAP,HBP,HCP,VAP,VBP,VCP,ALPHP)
      NF=4
      CALL CGPL2(XAPV,YAPV,2,NF,4,HAP,HBP,HCP,VAP,VBP,VCP,ALPHP)
      HA=0.0
      HC=HCP
      VA=VAP
      VC=VCP
      IPLOT=1
      GO TO 700
  710 CONTINUE
      NF=0
      CALL CGPL2(XAPV,YAPV,2,NF,4,HAP,HBP,HCP,VAP,VBP,VCP,ALPVP)
      STOP
      END
C
C
C
C
C
C
C
C
C
C
C**********************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE SIGN OF A QUANTITY.
C
C**********************************************************************
```

```
      SUBROUTINE SIG(X,SGN)
      IMPLICIT REAL*8 (A-H,O-Z)
      IF(DABS(X).LT.1.0D-08) GO TO 800
      SGN=X/DABS(X)
      GO TO 810
  800 CONTINUE
      SGN=0.D0
  810 CONTINUE
      RETURN
      END
C
C
C
C*****************************************************************
C
C     THE FOLLOWING PROGRAM TESTS FOR THE DEGREE OF DAMPING
C     (UNDERDAMPING, CRITICAL DAMPING, OR OVERDAMPING) AND THEN
C     CALCULATES THE CORRESPONDING DAMPED ANGULAR FREQUENCY.
C
C*****************************************************************
      SUBROUTINE DAMP(N,PSTAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      SQ=1.0-N**2
      CALL SIG(SQ,SGNSQ)
      IF(SGNSQ.GE.1.0D-08) PSTAR=DSQRT(SQ)
      IF(DABS(SGNSQ).LT.1.0D-08) PSTAR=1.0
      IF(SGNSQ.LE.-1.0D-08) PSTAR=DSQRT(-SQ)
      RETURN
      END
C
C
C
C*****************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE DISPLACEMENT AT THE END
C     OF THE SEGMENT BEING EXAMINED.
C
C*****************************************************************
      SUBROUTINE DISPL(XO,XDOTO,N,DELTAT,P,DELTAX)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      C1=XO
      C2=(N*XO+XDOTO)/P
C*****************************************************************
C
C     TESTING FOR THE DEGREE OF DAMPING OF THE LINEAR APPROXIMATION.  IF
C     UNDERDAMPED, THE PROGRAM MOVES TO STEP 900.  IF THERE IS CRITICAL
C     DAMPING, IT GOES TO STEP 910.  FOR OVERDAMPING, THE PROGRAM
C     CONTINUES AT STEP 920.
C
C*****************************************************************
      SQ=1.0-N**2
      IF(SQ.GE.1.0D-08) GO TO 900
      IF(DABS(SQ).LT.1.0D-08) GO TO 910
      IF(SQ.LE.-1.0D-08) GO TO 920
  900 CONTINUE
      DELTAX=DEXP(-N*DELTAT)*(C1*DCOS(P*DELTAT)+C2*DSIN(P*DELTAT))-XO
      GO TO 930
  910 CONTINUE
      DELTAX=DEXP(-P*DELTAT)*(XO+(N*XO+XDOTO)*DELTAT)-XO
      GO TO 930
  920 CONTINUE
      DELTAX=DEXP(-N*DELTAT)*(C1*DCOSH(P*DELTAT)+C2*DSINH(P*DELTAT))-XO
  930 CONTINUE
      RETURN
      END
C
C
C
C*****************************************************************
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE VELOCITY AT THE END OF THE
C     SEGMENT BEING EXAMINED.
C
C*****************************************************************
      SUBROUTINE SPEED(XO,XDOTO,N,DELTAT,P,XDOT1)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
C*****************************************************************
C
C     TESTING FOR THE DEGREE OF DAMPING OF THE LINEAR APPROXIMATION.  IF
C     UNDERDAMPED, THE PROGRAM MOVES TO STEP 1000.  IF THERE IS CRITICAL
C     DAMPING, IT GOES TO STEP 1010.  FOR OVERDAMPING, THE PROGRAM
C     CONTINUES AT STEP 1030.
C
C*****************************************************************
      SQ=1.0-N**2
      IF(SQ.GE.1.0D-08) GO TO 1000
      IF(DABS(SQ).LT.1.0D-08) GO TO 1010
      IF(SQ.LE.-1.0D-08) GO TO 1020
 1000 CONTINUE
      C1=XDOTO
      C2=-N*(N*XO+XDOTO)/P-P*XO
      XDOT1=DEXP(-N*DELTAT)*(C1*DCOS(P*DELTAT)+C2*DSIN(P*DELTAT))
      GO TO 1030
 1010 CONTINUE
      XDOT1=DEXP(-P*DELTAT)*(-P*XO+(N*XO+XDOTO)*(1.0-P*DELTAT))
      GO TO 1030
 1020 CONTINUE
      C1=XDOTO
      C2=-N*(N*XO+XDOTO)/P+P*XO
      XDOT1=DEXP(-N*DELTAT)*(C1*DCOSH(P*DELTAT)+C2*DSINH(P*DELTAT))
 1030 CONTINUE
      RETURN
      END
C
C
C
C*****************************************************************
```

```
C
C      THE FOLLOWING SUBROUTINE CALCULATES THE TIME INTERVAL REQUIRED TO
C      REACH ZERO VELOCITY (I. E., A PEAK OR TROUGH ON THE DISPLACEMENT-
C      TIME CURVE) BY USING A NEWTON-RAPHSON METHOD.
C
C**********************************************************************
      SUBROUTINE TIME(XO,XDOTO,DELT,N,P,DTIME)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 N
      DELTA=DELT+0.05
C**********************************************************************
C
C      TESTING FOR THE DEGREE OF DAMPING OF THE LINEAR APPROXIMATION.  IF
C      UNDERDAMPED, THE PROGRAM MOVES TO STEP 1100.  IF THERE IS CRITICAL
C      DAMPING, IT GOES TO STEP 1120.  FOR OVERDAMPING, THE PROGRAM
C      CONTINUES AT STEP 1140.
C
C**********************************************************************
      SQ=1.0-N**2
      IF(SQ.GE.1.0D-08) GO TO 1100
      IF(DABS(SQ).LT.1.0D-08) GO TO 1120
      IF(SQ.LE.-1.0D-08) GO TO 1140
C**********************************************************************
C
C      F=MAIN FUNCTION (VELOCITY)
C      FP=F-PRIME (ACCELERATION)
C
C**********************************************************************
 1100 CONTINUE
      C1=XDOTO
      C2=-N*(N*XO+XDOTO)/P-P*XO
      C3=-N*C1+P*C2
      C4=-P*C1-N*C2
 1110 CONTINUE
      F=DEXP(-N*DELTA)*(C1*DCOS(P*DELTA)+C2*DSIN(P*DELTA))
      FP=DEXP(-N*DELTA)*(C3*DCOS(P*DELTA)+C4*DSIN(P*DELTA))
      DELTA1=DELTA-F/FP
      DIFF=DELTA1-DELTA
      IF(DABS(DIFF).LT.1.0D-08) GO TO 1160
      DELTA=DELTA1
      GO TO 1110
 1120 CONTINUE
      F=-P*XO+(N*XO+XDOTO)*(1.0-P*DELTA)
      FP=P*(P*XO-(N*XO+XDOTO)*(2.0-P*DELTA))
 1130 CONTINUE
      DELTA1=DELTA-F/FP
      DIFF=DELTA1-DELTA
      IF(DABS(DIFF).LT.1.0D-08) GO TO 1160
      DELTA=DELTA1
      GO TO 1130
 1140 CONTINUE
      C1=XDOTO
      C2=-N*(N*XO+XDOTO)/P+P*XO
      C3=-N*C1+P*C2
      C4=P*C1-N*C2
 1150 CONTINUE
      F=DEXP(-N*DELTA)*(C1*DCOSH(P*DELTA)+C2*DSINH(P*DELTA))
      FP=DEXP(-N*DELTA)*(C3*DCOSH(P*DELTA)+C4*DSINH(P*DELTA))
      DELTA1=DELTA-F/FP
      DIFF=DELTA1-DELTA
      IF(DABS(DIFF).LT.1.0D-08) GO TO 1160
      DELTA=DELTA1
      GO TO 1150
 1160 CONTINUE
      DTIME=DELTA
      RETURN
      END
C
C
C
C**********************************************************************
C
C      THE FOLLOWING SUBROUTINE CONTAINS THE DIFFERENTIAL EQUATIONS TO BE
C      SOLVED BY DVERK:  THE INSTANTANEOUS VELOCITY AND THE ORIGINAL
C      EQUATION.
C
C**********************************************************************
      SUBROUTINE FCN(N,X,Y,YPRIME)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 MU
      REAL*8 Y(N),YPRIME(N),X
      COMMON MU
      YPRIME(1)=Y(2)
      YPRIME(2)=-(MU*(Y(1)**2-1.0)*YPRIME(1)+Y(1))
      RETURN
      END
END OF FILE
```

```
C***********************************************************************
C*                                                                     *
C*    PROGRAM MATHIEU                                                  *
C*                                                                     *
C*                                                                     *
C*                                                                     *
C*    THE FOLLOWING PROGRAM CALCULATES THE INTEGRAL ORDER SOLUTION     *
C*    FOR THE MATHIEU EQUATION:                                        *
C*                                                                     *
C*    Y(DOUBLE-PRIME)+(A-2*Q*COS(2*Z))*Y=0                             *
C*                                                                     *
C*    OBTAINED BY PIECEWISE LINEARIZATION USING CHORDS.                *
C*                                                                     *
C*                                                                     *
C*    THE RESULTS ARE BASED ON THE STANDARD FORM:                      *
C*                                                                     *
C*    Y=ALPHA*COS(P*Z)+BETA*SIN(P*Z)                                   *
C*                                                                     *
C*    P=SQRT(A-2*Q*COS(2*Z))                                           *
C*                                                                     *
C*                                                                     *
C*    THE CHARACTERISTIC NUMBER, A, IS FOUND BY MAKING AN INITIAL      *
C*    GUESS AND FINDING THE CLOSEST VALUE (EITHER ODD OR EVEN          *
C*    ORDER) USING A METHOD OF CONTINUED FRACTIONS, DESCRIBED          *
C*    FURTHER ON.                                                      *
C*                                                                     *
C*                                                                     *
C*    THE PROGRAM ALSO CALCULATES THE APPROPRIATE MATHIEU FUNCTIONS    *
C*    USING A SERIES APPROXIMATION, AS WELL AS A RUNGE-KUTTA           *
C*    SOLUTION USING FIFTH- AND SIXTH-ORDER METHODS.  ALL THREE        *
C*    RESULTS ARE PLOTTED.                                             *
C*                                                                     *
C***********************************************************************
C
C
C
C
C
C
C
C
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL FCN
      REAL*8 Y(2),C(24),W(2,20)
      REAL*8 LOW,LOWOUT,MID,MIDOUT
      REAL*8 RATIO1(100),RATIO2(100),COEFF(100),T(1000)
      REAL*4 AMPLF(1000),AMPLS(1000),AMPLRK(1000),Z(1000)
      REAL*4 YP(1000),ZP(1000),YRK(1000),ZPRK(1000)
      REAL*4 HA,HB,HC,VA,VB,VC
      REAL*4 XA(2),YA(2)
      INTEGER*4 ALPH(20)
      COMMON/CALC/AIN,DISPLO,DIFF,AOUT,IPAR
      COMMON/CHAR/A
      COMMON/PARAM/Q
C#######################################################################
C
C     THE FOLLOWING VALUES ARE READ FROM DATA FILE MATHIEUDATA:
C
C
C     AGUESS=INITIAL GUESS FOR CHARACTERISTIC NUMBER
C     Q=SAME AS IN EQUATION
C
C     DISPLO=INITIAL AMPLITUDE
C     VO=INITIAL VELOCITY
C     TO=INITIAL TIME
C
C     DELTA=INCREMENT/DECREMENT FOR A (USED IN SCAN PHASE)
C     H=NUMBER OF DIVISIONS OF PI (AN INTEGRAL MULTIPLE OF 50.0, USED
C       FOR FINDING INCREMENT OF PI)
C
C     HA,HC=PLOT PARAMETERS FOR HORIZONTAL AXIS (HC AN INTEGRAL MULTIPLE
C         OF 10.0)
C
C     VC=PLOT PARAMETER FOR VERTICAL AXIS (AN INTEGRAL MULTIPLE OF 12.0,
C         12.0 GIVING BEST RESULTS)
C
C     XA,YA=PARAMETERS FOR PLOTTING ZERO LINE
C
C     ALPH=ARRAY FOR PLOT AXIS LABELS
C
C#######################################################################
      READ(5,10) AGUESS,Q
   10 FORMAT(2D10.6)
      READ(5,20) DISPLO,VO,ZO
   20 FORMAT(3D10.6)
      READ(5,30) DELTA,H
   30 FORMAT(2D10.4)
      READ(5,40) HA,HC
   40 FORMAT(2F10.4)
      READ(5,50) VC
   50 FORMAT(F10.4)
      READ(5,60) XA(1)
   60 FORMAT(F10.4)
      READ(5,70) YA(1),YA(2)
   70 FORMAT(2F10.4)
      READ(5,80) (ALPH(I),I=1,12)
   80 FORMAT(12A4)
      READ(5,90) (ALPH(I),I=13,16)
   90 FORMAT(4A4)
      READ(5,100) (ALPH(I),I=17,20)
  100 FORMAT(4A4)
C#######################################################################
C
C     DELTAZ=INCREMENT OF Z (SINCE SOLUTIONS ARE PERIODIC IN EITHER PI
C            OR 2*PI)
C
C#######################################################################
      PI=3.141592653589793
```

```
      DELTAZ=PI/H
C**************************************************************
C
C     THE FOLLOWING ARE USED AS PROGRAM FLAGS FOR CALCULATING EITHER THE
C     ODD OR EVEN INTEGRAL ORDER CHARACTERISTIC NUMBERS.  THE ODD ONE IS
C     FOUND FIRST AND THEN THE EVEN.
C
C     IPAR=FLAG FOR DETERMINING ORDER ("0" INDICATES ODD, "1" EVEN)
C     I=COUNTER FOR DETERMINING HOW MANY TIMES THE SET OF CALCULATIONS
C       HAVE BEEN DONE ("1" MEANS THE ODD ORDER VALUE IS BEING FOUND,
C       "2" THE EVEN)
C
C**************************************************************
      IPAR=0
      I=1
C**************************************************************
C
C     THE EQUATION PARAMETERS ARE WRITTEN INTO OUTPUT FILE MATHIEUNUM.
C
C**************************************************************
      WRITE(10,110)
  110 FORMAT('1','RESULTS FOR THE EQUATION:')
      WRITE(10,120)
  120 FORMAT('0','Y(DOUBLE-PRIME)+(A-2*Q*COS(2*Z))*Y=0')
      WRITE(10,130) DISPLO,VO
  130 FORMAT('0','Y(0)=',1X,G20.10,2X,'Y(PRIME)(0)=',1X,G20.10)
      WRITE(10,140) AGUESS,Q
  140 FORMAT('0','A(GUESS)=',1X,G20.10,2X,'Q=',1X,G20.10)
C**************************************************************
C
C     THE FOLLOWING SECTION (ENDING AT STEP 450) CALCULATES THE
C     CHARACTERISTIC NUMBER FOR A GIVEN AGUESS AND Q, BASED UPON THE
C     INITIAL DISPLACEMENT AND THE PROXIMITY OF AGUESS TO THE
C     APPROPRIATE ODD AND EVEN ORDER RESULTS.
C
C     INITIALLY, A RANGE OF VALUES BETWEEN WHICH THE FINAL RESULT FOR
C     THE PARTICULAR ORDER IN QUESTION LIES IS FOUND (STEPS 160 TO 300)
C     THIS IS DONE BY EITHER INCREMENTING OR DECREMENTING AGUESS BY A
C     FIXED AMOUNT UNTIL THE LIMITS ARE FOUND, WITH THE DIRECTION OF THE
C     SEARCH (EITHER ABOVE OR BELOW AGUESS) DEPENDING UPON THE OUTPUT
C     VALUE FROM A SET OF RECURRENCE RELATIONS USING AGUESS AS THE
C     INPUT.
C
C     ONCE THE BOUNDS ARE KNOWN, THE PROGRAM FOCUSES TO THE FINAL RESULT
C     BY INTERVAL HALVING (STEPS 300 TO 350).  HERE THE DIRECTION IS
C     DETERMININED BY THE OUTPUT VALUE FROM THE SAME RECURRENCE RELATIONS
C     USING THE FIRST MIDPOINT AS THE INITIAL INPUT
C
C     ONCE THE CHARACTERISTIC NUMBERS FOR THE ODD AND EVEN ORDERS HAVE
C     BEEN FOUND, THE ONE CLOSEST TO AGUESS IS THE VALUE THAT IS USED
C     IN CALCULATING THE SOLUTION.
C
C
C
C     AIN=INPUT VALUE FOR RECURRENCE RELATIONS
C     RECUR=SUBROUTINE FOR RECURRENCE RELATIONS
C     DIFF=DIFFERENCE BETWEEN AIN AND OUTPUT VALUE
C
C
C
C**NOTE**IF AT ANY POINT THE INPUT AND OUTPUT VALUES ARE NEARLY EQUAL,
C         THE SEARCH FOR THE UPPER AND LOWER LIMITS OR THE FOCUSING PHASE
C         STOPS, THE VALUE FOR IPAR IS FOUND (WHICH DETERMINES THE
C         PERIOD), AND THE OUTPUT VALUE TAKEN AS THE FINAL RESULT.
C
C**************************************************************
  150 CONTINUE
      AIN=AGUESS
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 410
C**************************************************************
C
C     THE FOLLOWING CONSIDERS A POSITIVE INPUT VALUE FOR THE
C     CHARACTERISTIC NUMBER (STEPS 160 TO 230).
C
C     AOUT=OUTPUT VALUE FROM RECUR
C     UP=UPPER LIMIT
C     LOW=LOWER LIMIT
C
C
C     1  AOUT POSITIVE
C
C     IF AIN IS GREATER THAN AOUT (SEE STEP 180), AIN IS DECREMENTED AND
C     A NEW AOUT CALCULATED (SHOULD AIN BECOME LESS THAN ZERO, THE
C     PROGRAM CONTINUES AT STEP 230)   IF AIN IS GREATER THAN AOUT, THE
C     PROGRAM GOES TO STEP 190   IF AOUT IS POSITIVE, THE PPROGRAM
C     RETURNS TO STEP 180.  IF AIN IS LESS THAN AOUT OR IF AIN IS
C     GREATER THAN AOUT (AOUT BEING NEGATIVE), STEP 200 IS NEXT.   THE
C     LOWER BOUND IS THE CURRENT AIN, THE UPPER LIMIT THE PREVIOUS ONE,
C     AND THE PROGRAM MOVES TO STEP 300.
C
C     IF AIN IS LESS THAN AOUT, AIN IS INCREMENTED AND A NEW AOUT FOUND
C     UNTIL AIN IS GREATER THAN AOUT (STARTING AT STEP 210, RETURNING
C     TO STEP 210 IF NOT).  THE UPPER BOUND IS THE CURRENT AIN AND THE
C     LOWER THE PREVIOUS ONE.  THE PROGRAM THEN MOVES TO STEP 300 TO
C     BEGIN THE FINAL SET OF ITERATIONS.
C
C
C     2  AOUT NEGATIVE
C
C     AIN IS DECREMENTED (STARTING AT STEP 220) AND A NEW AOUT IS FOUND
C     UNTIL AOUT BECOMES POSITIVE, THE PROGRAM CONTINUING AT STEP 170.
C     IF AOUT IS STILL NEGATIVE, IT GOES BACK TO STEP 220.  IF AIN GETS
C     LESS THAN ZERO, THE PROGRAM GOES TO STEP 230.
C
C**************************************************************
      IF(AIN.LE.0.D0) GO TO 230
  160 CONTINUE
      IF(AOUT.LE.0.D0) GO TO 220
  170 CONTINUE
```

```
      IF(AIN.LE.AOUT) GO TO 210
C
  180 CONTINUE
      AIN=AIN-DELTA
      IF(AIN.LE.0.D0) GO TO 230
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 410
      IF(AIN.GT.AOUT) GO TO 190
      GO TO 200
  190 CONTINUE
      IF(AOUT.GT.0.D0) GO TO 180
  200 CONTINUE
      UP=AIN+DELTA
      LOW=AIN
      GO TO 300
C
  210 CONTINUE
      AIN=AIN+DELTA
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 410
      IF(AIN.LE.AOUT) GO TO 210
      UP=AIN
      LOW=AIN-DELTA
      GO TO 300
C
  220 CONTINUE
      AIN=AIN-DELTA
      IF(AIN.LE.0.D0) GO TO 230
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 410
      IF(AOUT.GT.0.D0) GO TO 170
      GO TO 220
C#########################################################################
C
C     THE FOLLOWING APPLIES FOR A NEGATIVE AIN (STEPS 230 TO 300).
C
C
C     1. AOUT NEGATIVE
C
C     IF AIN IS LESS THAN AOUT (STARTING AT SETP 250), AIN IS
C     INCREMENTED AND A NEW AOUT FOUND, THE PROGRAM GOING TO STEP 160
C     FOR A POSITIVE VALUE OF AIN.  IF AIN IS LESS THAN AOUT, THE
C     PROGRAM MOVES TO STEP 260.  IF AOUT IS STILL NEGATIVE, THE NEXT
C     IS TO STEP 270.  THE UPPER LIMIT IS THE CURRENT AIN AND THE
C     LOWER THE PREVIOUS ONE.  THE FINAL STAGE THEN STARTS AT STEP
C     300.
C
C     IF AIN IS GREATER THAN AOUT, AIN IS DECREMENTED (STARTING AT STEP
C     280) AND A NEW AOUT FOUND UNTIL AIN IS LESS THAN AOUT, THE PROGRAM
C     RETURNING TO STEP 280 IF NOT.  THE UPPER LIMIT IS THE PREVIOUS AIN
C     AND THE LOWER THE CURRENT ONE.  STEP 300 IS NEXT.
C
C
C     2. AOUT POSITIVE
C
C     STARTING AT STEP 290, AIN IS INCREMENTED AND A NOW AOUT IS FOUND
C     UNTIL AOUT IS NEGATIVE, THEN MOVING TO STEP 240 AND CONTINUING
C     FROM THERE.  IF AOUT REMAINS POSITIVE, THE PROGRAM RETURNS TO STEP
C     290.  SHOULD AIN BECOME POSITIVE, THE PROGRAM MOVES TO STEP 160.
C
C
C#########################################################################
  230 CONTINUE
      IF(AOUT.GT.0.D0) GO TO 290
  240 CONTINUE
      IF(AIN.GT.AOUT) GO TO 280
C
  250 CONTINUE
      AIN=AIN+DELTA
      IF(AIN.GT.0.D0) GO TO 160
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 410
      IF(AIN.LE.AOUT) GO TO 260
      GO TO 270
  260 CONTINUE
      IF(AOUT.LE.0.D0) GO TO 250
  270 CONTINUE
      UP=AIN
      LOW=AIN-DELTA
      GO TO 300
C
  280 CONTINUE
      AIN=AIN-DELTA
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 410
      IF(AIN.GT.AOUT) GO TO 280
      UP=AIN+DELTA
      LOW=AIN
      GO TO 300
C
  290 CONTINUE
      AIN=AIN+DELTA
      IF(AIN.GT.0.D0) GO TO 160
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 410
      IF(AOUT.LE.0.D0) GO TO 240
      GO TO 290
C#########################################################################
C
C     THE FOLLOWING SECTION IS THE FINAL SET OF ITERATIONS TO FIND THE
C     CHARACTERISTIC NUMBER BY USING INTERVAL HALVING.
C
C     MID=INTERVAL MIDPOINT (AVERAGE OF UP AND LOW)
C     MIDOUT=OUTPUT FROM RECUR USING MID AS INPUT
C
C     BASICALLY, IF MID IS GREATER THAN MIDOUT, MID BECOMES THE NEW
C     UPPER LIMIT, AND IF LESS THAN MIDOUT, THE NEW LOWER BOUND.
C
C#########################################################################
  300 CONTINUE
```

```
      MID=(UP+LOW)/2.0
      AIN=MID
      CALL RECUR
      IF(DABS(DIFF).LE.1.0D-08) GO TO 350
      MIDOUT=AOUT
      IF(MID.LE.0.D0) GO TO 330
      IF(MIDOUT.LE.0.D0) GO TO 320
      IF(MID.LE.MIDOUT) GO TO 320
  310 CONTINUE
      UP=MID
      GO TO 300
  320 CONTINUE
      LOW=MID
      GO TO 300
  330 CONTINUE
      IF(MIDOUT.LE.0.D0) GO TO 340
      GO TO 320
  340 CONTINUE
      IF(MID.LE.MIDOUT) GO TO 320
      GO TO 310
C#################################################################
C
C     FROM STEPS 350 TO 390, THE PROGRAM DETERMINES IF BOTH THE ODD AND
C     EVEN ORDER SOLUTIONS HAVE BEEN CALCULATED, AND THEN WRITING OUT
C     THE APPROPRIATE VALUES INTO OUTPUT FILE MATHIEUNUM ONCE THIS HAD
C     BEEN DONE.
C
C     AODD=ODD ORDER CHARACTERISTIC NUMBER
C     AEVEN=EVEN ORDER NUMBER
C
C#################################################################
  350 CONTINUE
      IF(I.NE.1) GO TO 360
      AODD=AOUT
      IPAR=1
      I=2
      GO TO 150
  360 CONTINUE
      AEVEN=AOUT
      WRITE(10,370) AODD,AEVEN
  370 FORMAT('0','A(ODD)=',1X,G20.10,2X,'A(EVEN)=',1X,G20.10)
C#################################################################
C
C     HERE THE PROGRAM DETERMINES WHICH CHARACTERISTIC NUMBER THAT
C     AGUESS IS CLOSER TO, AODD OR AEVEN, AND THEN WRITING INTO
C     MATHIEUNUM THE RESULT, USING THIS VALUE FOR CALCULATING THE
C     SOLUTIONS.
C
C     FACT=FACTOR FOR DETERMINING SOLUTION PERIOD (1.0 IS FOR EVEN ORDER
C          AND 2.0 FOR ODD)
C
C#################################################################
      IF(AODD.LE.AEVEN) GO TO 390
      IF(AGUESS.LE.AODD) GO TO 380
      GO TO 440
  380 CONTINUE
      IF(AGUESS.LE.AEVEN) GO TO 420
      DIFF1=AODD-AGUESS
      DIFF2=AGUESS-AEVEN
      IF(DIFF1.LE.DIFF2) GO TO 440
      GO TO 420
C
  390 CONTINUE
      IF(AGUESS.LE.AEVEN) GO TO 400
      GO TO 420
  400 CONTINUE
      IF(AGUESS.LE.AODD) GO TO 440
      DIFF1=AEVEN-AGUESS
      DIFF2=AGUESS-AODD
      IF(DIFF1.GT.DIFF2) GO TO 440
      GO TO 420
  410 CONTINUE
      IF(IPAR.EQ.1) GO TO 440
C
  420 CONTINUE
      A=AEVEN
      IPAR=1
      FACT=1.0
      WRITE(10,430) A
  430 FORMAT('0','CHARACTERISTIC NUMBER (EVEN ORDER)=',1X,G20.10)
      GO TO 460
  440 CONTINUE
      A=AODD
      FACT=2.0
      IPAR=0
      WRITE(10,450) A
  450 FORMAT('0','CHARACTERISTIC NUMBER (ODD ORDER)=',1X,G20.10)
C#################################################################
C
C     STEPS 460 TO 500 ARE THE PIECEWISE LINEAR SOLUTION.
C
C     AMPLP=ARRAY FOR PIECEWISE LINEAR DISPLACEMENT
C     BIGP=LARGEST VALUE FOR AMPLP (TO BE USED FOR PLOT SCALING)
C     Z=ARRAY FOR INSTANTANEOUS NORMALIZED VALUES FOR Z
C     XO=INITIAL DISPLACEMENT OF LINEAR SEGMENT
C     XPO=INITIAL VELOCITY OF LINE
C     NPOINT=NUMBER OF DATA POINTS (USED FOR PLOTTING)
C
C#################################################################
  460 CONTINUE
      AMPLP(1)=DISPLO
      BIGP=DABS(DISPLO)
      Z(1)=Z0/PI
      T(1)=Z0
      XO=DISPLO
      XPO=VO
      NPOINT=FACT*H
C#################################################################
C
C     IK=COUNTER FOR FILLING REMAINING ELEMENTS OF AMPLP AND Z
```

```
C       PSQ=SQUARE OF ANGULAR FREQUENCY OF APPROXIMATED
C       X=FINAL DISPLACEMENT OF LINEAR SEGMENT
C       XP=FINAL LINE VELOCITY
C
C
C       IF PSQ IS POSITIVE, THE SOLUTION IS OF THE FAMILIAR TRIGONOMETRIC
C       FORM.  IF ZERO, THE SOLUTION IS LINEAR (STEP 470) AND IF NEGATIVE,
C       HYPERBOLIC (STEP 460).
C
C**************************************************************
        DO 500 I=1,NPOINT
        IP=I+1
        PSQ=A-2.0*Q*DCOS((2.0*DFLOAT(I)-1.0)*DELTAZ)
        IF(PSQ.LT.-1.0D-10) GO TO 460
        IF(DABS(PSQ).LE.1.0D-10) GO TO 470
        P=DSQRT(PSQ)
        X=XO*DCOS(P*DELTAZ)+(XPO/P)*DSIN(P*DELTAZ)
        XP=-P*XO*DSIN(P*DELTAZ)+XPO*DCOS(P*DELTAZ)
        GO TO 490
  470   CONTINUE
        X=XPO*DELTAZ+XO
        XP=XPO
        GO TO 490
  460   CONTINUE
        P=DSQRT(-PSQ)
        X=XO*DCOSH(P*DELTAZ)+(XPO/P)*DSINH(P*DELTAZ)
        XP=P*XO*DSINH(P*DELTAZ)+XPO*DCOSH(P*DELTAZ)
  490   CONTINUE
        AMPLP(IP)=X
        Z(IP)=DFLOAT(I)*DELTAZ/PI
C**************************************************************
C
C       T=DOUBLE-PRECISION ARRAY FOR Z (TO BE USED FOR FINDING SERIES
C         APPROXIMATION)
C
C**************************************************************
        T(IP)=DFLOAT(I)*DELTAZ
        XO=X
        XPO=XP
        IF(DABS(X).GT.BIGP) BIGP=DABS(X)
  500   CONTINUE
C**************************************************************
C
C       THE FOLLOWING CALCULATES THE COEFFICIENTS OF THE SERIES
C       APPROXIMATION.  USING THE METHOD DESCRIBED BY E.L. INCE IN
C       "TABLES OF THE ELLIPTIC CYLINDER FUNCTIONS" (PROC. ROY. SOC.
C       EDINBURGH, VOL. 52, P. 355-423, 1932) AND N.W. MCLACHLAN IN
C       "THEORY AND APPLICATION OF MATHIEU FUNCTIONS", CH. 3, P. 28 FF.,
C       OXFORD UNIVERSITY PRESS, 1947
C
C       IR1=COUNTER FOR CALCULATING PRELIMINARY COEFFICIENT RATIOS
C       GAMMA=FACTOR USED FOR CALCULATING CONTINUED FRACTIONS USED FOR
C             ABOVE
C
C       FOR A NON-ZERO INITIAL DISPLACEMENT AND AN EVEN ORDER
C       CHARACTERISTIC NUMBER, THE PROGRAM MOVES TO STEP 510.  FOR A ZERO
C       INITIAL DISPLACEMENT AND EVEN ORDER, THE CALCULATIONS START AT
C       STEP 570.  FOR NON-ZERO INITIAL DISPLACEMENT AND ODD ORDER, THE
C       ITERATIONS BEGIN AT STEP 580.  FOR ODD ORDER AND ZERO
C       DISPLACEMENT, STEP 600 IS WHERE THE PROGRAM CONTINUES.
C
C**************************************************************
        IR1=26
        GAMMA=0.0
        IF((DISPLO.NE.0.0).AND.(IPAR.EQ.1)) GO TO 510
        IF((DISPLO.EQ.0.0).AND.(IPAR.EQ.1)) GO TO 570
        IF((DISPLO.NE.0.0).AND.(IPAR.EQ.0)) GO TO 580
        GO TO 600
C**************************************************************
C
C       THE CASE OF NON-ZERO INITIAL DISPLACEMENT AND EVEN ORDER.
C
C       RATIO=RATIO BETWEEN COEFFICIENT FOR SPECIFIC TERM AND ONE FOR
C             PREVIOUS TERM
C       SUM=TERM USED IN SUMMING SQUARES OF VARIOUS RATIO VALUES
C       IEND=COUNTER USED IN CALCULATING VARIOUS VALUES FOR RATIO
C
C**************************************************************
  510   CONTINUE
        RATIO1(1)=A/Q
        RATIO1(2)=(A-4.0)/Q-2.0/RATIO1(1)
        SUM=2.0
        IEND=2
C**************************************************************
C
C       THE CONTINUED FRACTION FOR CALCULATING RATIO.
C
C       I1STAR=COUNTER USED FOR RATIO1
C
C**************************************************************
  520   CONTINUE
        I1STAR=IR1
        IF((DISPLO.EQ.0.0).AND.(IPAR.EQ.1)) I1STAR=IR1-1
        RATIO1(I1STAR)=Q/(A-(2.0*DFLOAT(IR1))**2-Q*GAMMA)
        GAMMA=RATIO1(I1STAR)
        IR1=IR1-1
        IF(IR1.EQ.IEND) GO TO 530
        GO TO 520
C**************************************************************
C
C       RATIO2=RATIO BETWEEN COEFFICIENT FOR SPECIFIC TERM AND COEFFICIENT
C             FOR PREVIOUS ONE
C       IR2,I2STAR=COUNTERS FOR RATIO2
C
C**************************************************************
  530   CONTINUE
        RATIO2(1)=RATIO1(1)
        DO 540 IR2=1,23
        I2STAR=IR2+1
        RATIO2(I2STAR)=RATIO2(IR2)*RATIO1(I2STAR)
```

```
  540 CONTINUE
C*********************************************************************
C
C     BY SUMMING ALL THE VARIOUS VALUES FOR RATIO2, AND INVERTING THE
C     SQUARE ROOT, THE VALUE FOR THE FIRST COEFFICIENT IS DETERMINED AND
C     SUBSEQUENTLY, ALL THE REMAINING ONES.
C
C     COEFF=ARRAY FOR COEFFICIENT VALUES
C     IC1,IC2=COUNTERS FOR COEFF
C
C*********************************************************************
      IR1=24
      DO 550 IS=1,IR1
      SUM=SUM+RATIO2(IS)**2
  550 CONTINUE
      COEFF(1)=1.0/DSQRT(SUM)
      DO 560 IC1=1,IR1
      IC2=IC1+1
      COEFF(IC2)=COEFF(1)*RATIO2(IC1)
  560 CONTINUE
      GO TO 610
C*********************************************************************
C
C     INITIALIZING THE CALCULATIONS FOR THE SITUATION OF ZERO INITIAL
C     AND EVEN ORDER.
C
C*********************************************************************
  570 CONTINUE
      RATIO1(1)=(A-4.0)/Q
      SUM=1.0
      IEND=1
      IR1=25
      GO TO 520
C*********************************************************************
C
C     SETTING UP THE CASE OF NON-ZERO DISPLACEMENT AND ODD ORDER.
C
C*********************************************************************
  580 CONTINUE
      RATIO1(1)=(A-1.0-Q)/Q
      SUM=1.0
      IEND=1
  590 CONTINUE
     *RATIO1(IR1)=Q/(A-(2.0*DFLOAT(IR1)+1.0)**2-Q*GAMMA)
      GAMMA=RATIO1(IR1)
      IR1=IR1-1
      IF(IR1.EQ.IEND) GO TO 530
      GO TO 590
C*********************************************************************
C
C     THE SITUATION OF ZERO INITIAL DISPLACEMENT AND ODD ORDER.
C
C*********************************************************************
  600 CONTINUE
      RATIO1(1)=(A-1.0+Q)/Q
      SUM=1.0
      IEND=1
      GO TO 590
C*********************************************************************
C
C     THE SECTION FROM STEP 610 TO STEP 650 IS THE SERIES APPROXIMATION
C     THAT IS DESCRIBED IN INCE AND MCLACHLAN.
C
C     BIGS=LARGEST VALUE FOR SERIES AMPLITUDE (FOR PLOT SCALING)
C
C*********************************************************************
  610 CONTINUE
      BIGS=DABS(DISPLO)
      NPOINT=FACT*H+1.0
C*********************************************************************
C
C     INITIALIZING THE CALCULATIONS OF THE SERIES TERMS.
C
C     DENOM=SUMMATION OF SERIES COEFFICIENTS (FOR NON-ZERO INITIAL
C           DISPLACEMENTS) OR COEFFICIENTS AND TERM NUMBER (FOR ZERO
C           INITIAL DISPLACEMENTS) USED IN DETERMINING MATHIEU FUNCTION
C           COEFFICIENT FOR COMPLETE SOLUTION
C     TERM=SUM OF INDIVIDUAL TERMS FOR MATHIEU FUNCTION APPROXIMATION
C
C*********************************************************************
      DO 650 J=1,NPOINT
      DENOM=0.0
      TERM=0.0
      IF(DISPLO.EQ.0.0) GO TO 630
C*********************************************************************
C
C     RESULTS FOR NON-ZERO INITIAL DISPLACEMENT.
C
C     THETA=TERM NUMBER
C
C*********************************************************************
      DO 620 K=1,25
      IF(IPAR.EQ.0) THETA=2.0*DFLOAT(K)-1.0
      IF(IPAR.EQ.1) THETA=2.0*DFLOAT(K)-2.0
      DENOM=DENOM+COEFF(K)
      TERM=TERM+COEFF(K)*DCOS(THETA*T(J))
  620 CONTINUE
C*********************************************************************
C
C     AMPLS=ARRAY FOR INSTANTANEOUS AMPLITUDE FOR SERIES SOLUTION
C     DISPLS=DOUBLE-PRECISION VALUE OF AMPLS
C
C*********************************************************************
      AMPLS(J)=DISPLO*TERM/DENOM
      DISPLS=DBLE(AMPLS(J))
      IF(DABS(DISPLS).GT.BIGS) BIGS=DABS(DISPLS)
      GO TO 650
  630 CONTINUE
C*********************************************************************
C
```

```
C     CALCULATIONS FOR ZERO INITIAL DISPLACEMENT.
C
C*****************************************************************
      DO 840 K=1,25
      IF(IPAR.EQ.0) THETA=2.0*DFLOAT(K)-1.0
      IF(IPAR.EQ.1) THETA=2.0*FLOAT(K)
      DENOM=DENOM+THETA*COEFF(K)
      TERM=TERM+COEFF(K)*DSIN(THETA*T(J))
  840 CONTINUE
      AMPLS(J)=VO*TERM/DENOM
      DISPLS=DBLE(AMPLS(J))
      IF(DABS(DISPLS).GT.BIGS) BIGS=DABS(DISPLS)
  850 CONTINUE
C*****************************************************************
C
C     THE SECTION ENDING AT STEP 860 IS THE RUNGE-KUTTA SOLUTION.
C
C
C
C     THE FOLLOWING ARE PARAMETERS FOR THE RUNGE-KUTTA ROUTINE.
C     CONSULT THE IMSL MANUAL ON "DVERK" FOR FURTHER DETAILS.
C
C*****************************************************************
      NPOINT=FACT*H+1.0
      BIGRK=DABS(DISPLO)
      Y(1)=DISPLO
      Y(2)=VO
      ZRK=Z0
      NEQN=2
      NW=2
      TOL=1.0D-12
      IND=1
      ZEND=DELTAZ
      DO 860 K=1,NPOINT
      AMPLRK(K)=Y(1)
      CALL DVERK(NEQN,FCN,ZRK,Y,ZEND,TOL,IND,C,NW,W,IER)
C*****************************************************************
C
C     INCREMENTING ZEND.
C
C*****************************************************************
      ZEND=ZEND+DELTAZ
      IF(DABS(Y(1)).GT.BIGRK) BIGRK=Y(1)
  860 CONTINUE
C*****************************************************************
C
C     WRITING THE SOLUTIONS INTO OUTPUT FILE MATHIEUNUM.
C
C*****************************************************************
      WRITE(10,870)
  870 FORMAT('0',11X,'Z',13X,'PIECEWISE LINEARIZATION',2X,'SERIES APPROX
     1IMATION',8X,'RUNGE-KUTTA')
      DO 890 L=1,NPOINT
      WRITE(10,880) Z(L),AMPLP(L),AMPLS(L),AMPLRK(L)
  880 FORMAT(1X,4(G20.10,4X))
  890 CONTINUE
C*****************************************************************
C
C     THE NEXT FEW LINES ARE USED IN FINALIZING ALL THE PLOT PARAMETERS
C     WHICH INCLUDES AXIS SCALING.
C
C*****************************************************************
      INC=HC
      XA(2)=Z(NPOINT)-0.005
      BIG=BIGRK
      IF((BIGP.GT.BIGS).AND.(BIGP.GT.BIGRK)) BIG=BIGP
      IF((BIGS.GT.BIGP).AND.(BIGS.GT.BIGRK)) BIG=BIGS
      M=1
      IF(BIG.LE.1.0) GO TO 710
  700 CONTINUE
      DEC=BIG/(10.0**M)
      IF(DEC.LE.1.0) GO TO 730
      M=M+1
      GO TO 700
  710 CONTINUE
      DIG=BIG*(10.0**M)
      IF(DIG.GE.1.0) GO TO 720
      M=M+1
      GO TO 710
  720 CONTINUE
      DEC=DIG/10.0
      M=-(M-1)
  730 CONTINUE
      IF(DEC.LE.0.125) VFACT=0.125
      IF((DEC.GT.0.125).AND.(DEC.LE.0.25)) VFACT=0.25
      IF((DEC.GT.0.25).AND.(DEC.LE.0.5)) VFACT=0.5
      IF((DEC.GT.0.5).AND.(DEC.LE.1.0)) VFACT=1.0
      SFACT=VC/10.0
      VA=-VFACT*SFACT*10.0**M
      DEN=VC/(2.0*SFACT)
      VB=(VFACT/DEN)*10.0**M
      HB=Z(NPOINT)/DFLOAT(INC)
C*****************************************************************
C
C     THE FOLLOWING SECTION SETS UP THE PIECEWISE LINEAR AND RUNGE-KUTTA
C     RESULTS SO THAT 26 POINTS OF THE PIECEWISE SOLUTION AND 27 OF THE
C     RUNGE-KUTTA VALUES ARE PLOTTED FOR A CLEARER REPRESENTATION, WITH
C     THE PLOTS BEING ALTERNATED FOR COMPARISON PURPOSES.
C
C     YP=ARRAY FOR PLOTTED PIECEWISE LINEAR DISPLACEMENTS
C     ZP=ARRAY FOR PLOTTED PIECEWISE LINEAR Z VALUES
C     YRK=ARRAY FOR PLOTTED RUNGE-KUTTA DISPLACEMENTS
C     ZPRK=ARRAY FOR PLOTTED RUNGE-KUTTA Z VALUES
C
C
C**NOTE**THE NUMBER OF POINTS TO BE PLOTTED MUST NOT EXCEED 1000
C
C*****************************************************************
       INCR=NPOINT/25
       INCRRK=INCR/2
```

```
      NPP=1
      NPRK=1
      NDP=1
      NDRK=1
  740 CONTINUE
      YP(NPP)=AMPLP(NDP)
      ZP(NPP)=Z(NDP)
      IF(NDP.EQ.NPOINT) GO TO 750
      NPP=NPP+1
      NDP=NDP+INCR
      GO TO 740
  750 CONTINUE
      YRK(NPRK)=AMPLRK(NDRK)
      ZPRK(NPRK)=Z(NDRK)
      IF(NDRK.EQ.1) GO TO 760
      NDRK=NDRK+INCR
      GO TO 770
  760 CONTINUE
      NDRK=NDRK+INCRRK
  770 CONTINUE
      IF(NDRK.GT.NPOINT) GO TO 780
      NPRK=NPRK+1
      GO TO 750
  780 CONTINUE
      NPRK=NPRK+1
      YRK(NPRK)=AMPLRK(NPOINT)
      ZPRK(NPRK)=Z(NPOINT)
C*********************************************************************
C
C     PLOTTING THE SOLUTIONS.  CONSULT THE WRITEUP ON CGPL/CGPL2 AND
C     THE MANUAL ON DIGITAL PLOTTING FOR DETAILS.
C
C*********************************************************************
      NF=1
      ND=NPOINT
      CALL CGPL2(Z,AMPLS,ND,NF,5,HA,HB,HC,VA,VB,VC,ALPH)
      ND=NPP
      CALL CGPL2(ZP,YP,ND,2,1,HA,HB,HC,VA,VB,VC,ALPH)
      ND=NPRK
      CALL CGPL2(ZPRK,YRK,ND,3,1,HA,HB,HC,VA,VB,VC,ALPH)
      NF=4
      CALL CGPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
      HORIZ=4.15
      VERT=-0.475
      CALL SYMBOL(HORIZ,VERT,0.2,163,0.0,-1)
      HORIZ=HA
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=HC
      CALL PLOT(HORIZ,VERT,2)
      VERT=VERT+4.0
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=HA
      CALL PLOT(HORIZ,VERT,2)
      VERT=VC+0.5
      CALL PLOT(HORIZ,VERT,2)
      STARTX=(HC-7.7)/2.0
      STARTY=VC+0.5
      HORIZ=STARTX+0.2
      VERT=STARTY+3.6
      CALL SYMBOL(HORIZ,VERT,0.2,'CHAR. NUM. (EST. VALUE) =',0.0,25)
      HORIZ=STARTX+5.4
      CALL NUMBER(HORIZ,VERT,0.2,AGUESS,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+3.2
      CALL SYMBOL(HORIZ,VERT,0.2,'CHAR. NUM. (FINAL VALUE) =',0.0,26)
      HORIZ=STARTX+5.6
      CALL NUMBER(HORIZ,VERT,0.2,A,0.0,4)
      HORIZ=STARTX+0.2
      VERT=STARTY+2.8
      CALL SYMBOL(HORIZ,VERT,0.2,'Q =',0.0,3)
      HORIZ=STARTX+1.0
      CALL NUMBER(HORIZ,VERT,0.2,Q,0.0,4)
      HORIZ=STARTX+3.5
      CALL SYMBOL(HORIZ,VERT,0.2,52,0.0,-1)
      HORIZ=STARTX+3.7
      CALL SYMBOL(HORIZ,VERT,0.2,'Z =',0.0,3)
      HORIZ=STARTX+4.5
      CALL SYMBOL(HORIZ,VERT,0.2,163,0.0,-1)
      HORIZ=STARTX+4.7
      CALL SYMBOL(HORIZ,VERT,0.2,97,0.0,-1)
      HORIZ=STARTX+4.9
      CALL NUMBER(HORIZ,VERT,0.2,H,0.0,1)
      HORIZ=STARTX+0.2
      VERT=STARTY+2.0
      CALL SYMBOL(HORIZ,VERT,0.2,'Y(0) =',0.0,6)
      HORIZ=STARTX+1.6
      CALL NUMBER(HORIZ,VERT,0.2,DISPLO,0.0,4)
      HORIZ=STARTX+3.5
      CALL SYMBOL(HORIZ,VERT,0.2,'Y',0.0,1)
      HORIZ=STARTX+3.7
      CALL SYMBOL(HORIZ,VERT,0.2,125,0.0,-1)
      HORIZ=STARTX+3.9
      CALL SYMBOL(HORIZ,VERT,0.2,'(0) =',0.0,5)
      HORIZ=STARTX+5.1
      CALL NUMBER(HORIZ,VERT,0.2,VO,0.0,4)
      HORIZ=STARTX+0.3
      VERT=STARTY+1.1
      CALL PLOT(HORIZ,VERT,3)
      HORIZ=STARTX+0.8
      CALL PLOT(HORIZ,VERT,2)
      HORIZ=STARTX+1.0
      VERT=STARTY+1.0
      CALL SYMBOL(HORIZ,VERT,0.2,'SERIES APPROXIMATION',0.0,20)
      HORIZ=STARTX+0.6
      VERT=STARTY+0.7
      CALL SYMBOL(HORIZ,VERT,0.2,1,0.0,-1)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.6
      CALL SYMBOL(HORIZ,VERT,0.2,'PIECEWISE LINEARIZATION',0.0,23)
```

```
      HORIZ=STARTX+0.5
      VERT=STARTY+0.3
      CALL SYMBOL(HORIZ,VERT,0.2,2,0.0,-1)
      HORIZ=STARTX+1.0
      VERT=STARTY+0.2
      CALL SYMBOL(HORIZ,VERT,0.2,'RUNGE-KUTTA',0.0,11)
      NF=0
      CALL CGPL2(XA,YA,2,NF,4,HA,HB,HC,VA,VB,VC,ALPH)
      STOP
      END
C
C
C
C
C
C
C
C
C
C
C#################################################################
C
C     THE FOLLOWING SUBROUTINE CALCULATES THE OUTPUT VALUE FOR A GUESS
C     FOR THE CHARACTERISTIC NUMBER, USING THE RECURRENCE RELATIONS THAT
C     ARE DESCRIBED IN MCLACHLAN AND INCE.
C
C#################################################################
      SUBROUTINE RECUR
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON/CALC/AIN,DISPLO,DIFF,AOUT,IPAR
      COMMON/PARAM/Q
C#################################################################
C
C     R=COUNTER FOR THE NUMBER OF ITERATIONS FOR THE PARTICULAR RELATION
C     USED
C     GAMMAR=FACTOR USED IN THE CONTINUED FRACTION (SEE INCE)
C
C     FOR NON-ZERO INITIAL DISPLACEMENT AND EVEN ORDER, THE PROGRAM
C     MOVES TO STEP 800.  FOR ZERO INITIAL DISPLACEMENT AND EVEN ORDER,
C     IT GOES TO STEP 820.  IF THE ORDER IS ODD AND INITIAL DISPLACEMENT
C     NON-ZERO, IT CONTINUES AT STEP 840, AND IF THE DISPLACEMENT
C     HAPPENS TO BE ZERO, IT GOES TO STEP 860.
C
C#################################################################
      R=200.0
      GAMMAR=0.0
      IF((DISPLO.NE.0.0).AND.(IPAR.EQ.1)) GO TO 800
      IF((DISPLO.EQ.0.0).AND.(IPAR.EQ.1)) GO TO 820
      IF((DISPLO.NE.0.0).AND.(IPAR.EQ.0)) GO TO 840
      GO TO 860
C#################################################################
C
C     THE CASE OF NON-ZERO INITIAL DISPLACEMENT AND EVEN ORDER.
C
C     PHIR,DENOMR=FACTORS USED IN CALCULATING CONTINUED FRACTION
C     A=VALUE OBTAINED
C
C#################################################################
  800 CONTINUE
      PHIR=1.0-AIN/(4.0*R**2)-GAMMAR
      DENOMR=(Q**2)/((16.0*R**2)*(R-1.0)**2)
      GAMMAR=DENOMR/PHIR
      R=R-1.0
      IF(R.EQ.1.0) GO TO 810
      GO TO 800
  810 CONTINUE
      A=-(Q**2)/(2.0*(1.0-AIN/4.0-GAMMAR))
      GO TO 870
C#################################################################
C
C     ZERO INITIAL DISPLACEMENT AND EVEN ORDER.
C
C#################################################################
  820 CONTINUE
      PHIR=1.0-AIN/(4.0*R**2)-GAMMAR
      DENOMR=(Q**2)/((16.0*R**2)*(R-1.0)**2)
      GAMMAR=DENOMR/PHIR
      R=R-1.0
      IF(R.EQ.2.0) GO TO 830
      GO TO 820
  830 CONTINUE
      A=4.0-(Q**2)/(16.0*(1.0-AIN/16.0-GAMMAR))
      GO TO 870
C#################################################################
C
C     THE SITUATION OF ODD ORDER AND NON-ZERO INITIAL DISPLACEMENT.
C
C#################################################################
  840 CONTINUE
      PHIR=1.0-AIN/(2.0*R+1.0)**2-GAMMAR
      DENOMR=(Q**2)/((4.0*R**2-1.0)**2)
      GAMMAR=DENOMR/PHIR
      IF(R.EQ.1.0) GO TO 850
      R=R-1.0
      GO TO 840
  850 CONTINUE
      A=1.0+Q-GAMMAR
      IF(Q.LT.0.0) Q=DABS(Q)
      GO TO 870
C#################################################################
C
C     THE CASE OF ODD ORDER AND ZERO INITIAL DISPLACEMENT.
C
C#################################################################
  860 CONTINUE
      Q=-Q
      GO TO 840
  870 CONTINUE
      AOUT=A
C#################################################################
```

```
C
C     DIFF=DIFFERENCE BETWEEN INPUT AND OUTPUT VALUES (USED IN
C           TERMINATING ITERATIONS FOR CHARACTERISTIC NUMBER)
C
C*********************************************************************
      DIFF=AIN-AOUT
      RETURN
      END
C
C
C
C*********************************************************************
C
C     THE FOLLOWING SUBROUTINE CONTAINS THE DIFFERENTIAL EQUATIONS TO BE
C     SOLVED BY DVERK:  THE INSTANTANEOUS VELOCITY AND THE ORIGINAL
C     EQUATION.
C
C*********************************************************************
      SUBROUTINE FCN(NEON,ZRK,Y,YPRIME)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 Y(NEON),YPRIME(NEON),ZRK
      COMMON/CHAR/A
      COMMON/PARAM/Q
      YPRIME(1)=Y(2)
      YPRIME(2)=-(A-2.0*Q*DCOS(2.0*ZRK))*Y(1)
      RETURN
      END
END OF FILE
```

VITA

NAME: Bernhard Michael Jatzeck

PLACE OF BIRTH: Berlin Spandau, Federal Republic of Germany

DATE OF BIRTH: October 12, 1955

POST-SECONDARY EDUCATION AND DEGREES

Faculty of Engineering, Camrose Lutheran College
Camrose, Alberta
1973-1974

University of Alberta
Edmonton, Alberta
1974-1977 B. Sc. (Mec. E.)

Dept. of Mechanical Engineering, University of British Columbia
Vancouver, B. C.
1979-1980 (Started thesis under Dr. V. J. Modi. Transferred course credits to University of Alberta).

Dept. of Mechanical Engineering, University of Alberta
Edmonton, Alberta
1980-1981 (Full-time), 1981-1982 (Part-time)

PROFESSIONAL EXPERIENCE

Junior Plant Engineer
Amoco Canada Petroleum Co. Ltd.
West Whitecourt Plant
June, 1977 - December, 1977

Process Engineer
Amoco Canada Petroleum Co. Ltd.
Crossfield Area Office
Calgary, Alberta
January, 1978 - April, 1978

Junior Project Engineer
Ingersoll-Rand Canada Inc.
Packaged Equipment Division
Calgary, Alberta
May, 1978 - August, 1979

PROFESSIONAL EXPERIENCE (continued)

Mechanical Engineer
SED Systems Inc
Advanced Systems Engineering Group (Aerospace Products)
Saskatoon, Saskatchewan
November, 1981 - Present

PROFESSIONAL SOCIETIES

Registered Professional Engineer
Association of Professional Engineers, Geologists, and Geophysicists of Alberta

Registered Professional Engineer
Association of Professional Engineers of Saskatchewan

Member
Canadian Society for Mechanical Engineering

Member
Engineering Institute of Canada

New Member
American Institute of Aeronautics and Astronautics

Student Member
American Association for the Advancement of Science (to be re-instated as full member upon renewal)